研华MAS控制器

MOTION BASIC使用手册

Version 1.9.7

第 1 章 研华 MOTION BASIC 介绍

1.1 MOTION BASIC 概述

Motion BASIC 是一种用于研华 MAS 控制器的多任务编程语言，它的语法与标准 BASIC 相似。在装有 windows 操作系统的计算机上运行 Motion Studio 就可以进行开发和测试所有 MAS 控制器功能。Motion Studio 提供了 BASIC 程序编辑和丰富的调试工具来进行应用程序的调试 ,使设备应用程序开发变得简单高效。

1.2 特点

- 基本程序指令和Visual BASIC兼容，运动控制指令简单，初学者容易上手。
- 支持10个任务独立执行、同时执行，强大的多任务处理功能提高软件结构和易维护性。
- 支持研华现有强大softmotion功能，并将指令精简化。
- 可整合进外部算法和函数到Motion BASIC指令，非常适合设备整合控制和高效开发。
- 编译执行方式，使得在简单易用的前提下，同时具有高级语言高效执行的优点。
- 稳定性高：当用户程序出现错误时，不会出现系统崩溃。

常见的三种指令特点比较表

	MOTION BASIC	G 代码	梯形图
可理解方式	较容易	容易	困难
执行方式	编译执行	解释执行	编译执行
执行速度	快	较慢	慢
功能	全	较少	全
操作硬件能力	所有	只支持电机、IO	所有
子程序	支持	支持	不支持
任务数	10	1 主任务/3 从任务	不支持
变数	支持	不支持	支持
数组	支持	不支持	不支持
数学函数	支持	不支持	支持

第 2 章 BASIC 指令详解

MOTION BASIC 规则说明

研华 Motion BASIC 指令主要包含运动控制指令和其他指令两大类。根据研华运动控制的特点，我们把运动控制指令分为命令和属性两部分指令，在第 2 章 BASIC 指令详解中有说明每条运动控制指令属于命令还是属性。其他指令的规则基本上类似于标准 BASIC 指令规则。本节将研华 MOTION BASIC 指令说明的一些重要规则描述如下：

- **研华 MOTION BASIC 指令集不区分字母大小写**
- **指令语法说明方法：**

  1. **语法说明中包含( )的部分，必须要使用( )，不可省略。**
  2. **说明中有[ ]的部分，表示[ ]中的内容根据实际需求可以填或不填，不会造成语法错误。**

  如下 LINE 指令说明：

  LINE distance1,distance2 [,distance3]；

  做两轴直线插补时，LINE 指令后面的参数填 distance1 和 distance2 就可以了。

  做三轴直线插补时，LINE 指令后面的参数就填 distance1,distance2,distance3。[,distance3]这个部分是根据实际需求选填的，选填的部分我们用[ ]说明。
- **运动控制属性类指令的赋值说明**

  运动控制属性类指令赋值是用“=”来做赋值的，如要设置加速度：

  正确 V→ACC=value

  错误 X →ACC value 语句则不符合语法规则。
- **运动控制指令中对指定轴的操作说明**

  运动控制指令对指定轴操作时，指定轴号时要使用“AX”这个关键词，仅用数字代表轴号不符合语法规则。如将锁存到的轴 0 理论位置值赋给一个变量 A:

  正确 V→ A=LDPOS(AX(0))

  错误 X →A=LDPOS(0) 语句则不符合语法规则。
- **系统已定义运动控制关键词**

  注意:用户自定义的变量名不能与以下表格中的关键词或枚举名一样，也不能与 Motion BASIC 的所有指令名一样。

关键词或枚举	说明	例子
AX	指定轴号的关键词，AX(0)代表轴 0，AX(5)代表轴 5	MVOE AX(0),1000 表示命令轴 0 运动 1000 个 UNIT
CW	顺时针方向	CIRC 0,5000,0,10000,0 可以用 CIRC CW,5000,0,10000,0 来代替
CCW	逆时针方向	CIRC 1,5000,0,10000,0 可以用 CIRC CCW,5000,0,10000,0 来代替
S	S 型曲线	JK=0 可以用 JK=T 来代替

T	T 型曲线	JK=1 可以用 JK=S 来代替
DONE	轴运动完成事件	WAIT DONE；参考 WAIT 指令说明
COMPARED	比较触发完成事件	WAIT COMPARED；参考 WAIT 指令说明
LATCHED	锁存完成事件	WAIT LATCHED ；参考 WAIT 指令说明
AXIS	系统内部定义的轴关键词	
RUN	系统内部定义的运行关键词	
STOP	系统内部定义的停止关键词	
TASK	系统内部定义的任务关键词	
DIR	系统内部定义的方向关键词	

2.1 流程控制语句

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.1.1	DIM...As...	定义数据类型	×	×
2.1.2	CONST	定义常量	×	×
2.1.3	IF...THEN...ELSEIF ...ELSE...END IF	IF 条件语句	×	×
2.1.4	FOR...TO... STEP...NEXT	FOR 循环语句	×	×
2.1.5	Select Case...End Select	CASE 语句	×	×
2.1.6	WHILE...WEND	While 循环语句	×	×
2.1.7	WAIT	等待事件结束	×	×
2.1.8	CancelWAIT	退出事件的等待	×	×
2.1.9	SLEEP	延时	×	×
2.1.10	TYPE	定义类	×	×
2.1.11	EXIT	退出一个控制流语句块或函数体	×	×
2.1.12	CONTINUE	中断循环体当前这次循环，继续循环体的执行	×	×
2.1.13	TMR 类	定时器类	×	×

2.1.1 DIM...As...

语法 1：DIM varname1 As DataType [,varname2 As DataType,...]

语法 2：DIM As DataType varname1 [,varname2, ...]

描述：定义变量

参数：varname 变量名

注意： 在一个 Task 里用 DIM 定义的局部变量只在该 Task 当前的程序段起到声明的作用，如果要声明到该 Task 的子程序段（如 SUB 段程序），那需要在 DIM 后面加上关键词 shared，如这样定义一个变量 x：DIM shared x As Integer。

例程

```
'定义 1 个变量
DIM a AS DOUBLE                    '定义一个变量名为 a 的 64 位浮点型变量
DIM text AS STRING ="Hi,MAS"       '定义一个变量名为 text 的字符串，并赋值为 HI,MAS

'定义多个变量
DIM AS ULONG b,c                   '定义 b、c 两个变量，数据类型为无符号长整型

'定义数组
DIM Array(2) AS BYTE={1,2,5}       '定义一个长度为 3，名为 Array 的 BYTE 类型数组,并赋值

'SUB，Function 中形参传递定义
DECLARE SUB label(a as ushort)           '变量 a 为 SUB 中的传递参数
DECLARE Function lable(b as ushort) as integer    '变量 b 为 Function 中的传递参数

'SUB，Function 中数组传递定义
DECLARE SUB test1(a() as ushort)         'a()为 SUB test 中的传递数组
DECLARE Function test2(b() as ushort) as integer 'b()为 Function test 中的传递数组

'跨 SUB、Function 的变量定义
DIM SHARED a AS INTEGER=0
SUB abc
    a=a+1    '变量 a 在 SUB abc 外声明，但是要在 SUB abc 中使用，需在 DIM 后加 SHARED 关键字
END SUB
```

2.1.2 CONST

语法：CONST varname = value

描述：定义常量。采用常量名可避免在程序中多处修改同一个数值

参数：varname 变量名

value 常数值

例程

```
CONST max_speed=50000      '定义速度常量
CONST Distance=10000       '定义位移常量
BASE 0
SVON
VH = max_speed             '将速度常量赋给运行速度属性
MOVE Distance              '将位移常量赋给位移指令参数
```

2.1.3 IF...THEN...ELSEIF...ELSE...END IF

语法：IF <condition1> THEN
commands1
[ELSEIF <condition2> THEN]
commands2
[ELSE commands3]
END IF

描述：该指令为条件语句。首先判断条件表达式 1；如果条件表达式 1 成立，则执行指令块 1，然后跳转至 END IF，退出条件语句。如果条件表达式 1 不成立，则判断条件表达式 2；如果条件表达式 2 成立，则执行指令块 2，然后跳转至 endif，退出条件语句。如果条件表达式 2 不成立，则执行指令块 3

参数： condition 条件表达式
commands 指令块（可单条指令也可多条指令）

例程

```
DIM a ASLONG
IF(a=0)   THEN                  '判断条件 1：a 是否为 0
DOUT (2)=1                      '条件 1 成立，则 DO2, DO6 分别赋值 1,1
DOUT (6)=1
ELSEIF( a>0 AND a<=5) THEN      '判断条件 2：a 是否大于 0 且小于等于 5
DOUT(2)=0                       '条件 2 成立，则 DO2, DO6 分别赋值 0,1
DOUT(6)=1
ELSE                            '条件 1、2 都不成立的情况，则 DO2, DO6 分别赋值 1,0
DOUT(2) =1
DOUT(6)=0
END IF
```

2.1.4 FOR...TO...STEP...NEXT

语法：For varname = startvalue TO endvalue [STEP stepvalue]

[commands]

NEXT varname

描述：FOR 循环语句。如果循环变量小于循环结束值，则执行指令块到 NEXT 时，循环变量自动加一个增量，再一次执行指令块；当循环变量大于等于循环结束值时，则停止循环。

参数： varname 循环体变量名

startvalue 循环起始值

endvalue 循环结束值

stepvalue 循环变量的增量，可选；缺省时，增量为 1。增量也可以是小数或负数。增量为负数时，循环起始值要大于循环结束值。

Commands 指令块

注意： 该指令最多嵌套八层

例程

```
DIM i AS ULONG
FOR i=0 To 7
      DOUT (i) =1          '将 DO0~DO7 全部置 1
NEXT i
FOR i=0 To 7 STEP 2
      DOUT (i) =0          '将 DO0、DO2、DO4、DO6 置 0
NEXT i
```

2.1.5 Select Case...End Select

语法：Select Case expression

[Case expressionlist]

[commands]

.....

[Case expressionlist]

[commands]

[Case Else]

[commands]

End Select

描述：类似 C 语言的 Switch...Case 条件语句。根据表达式的内容选择执行哪个指令块，各分支 CASE 表达式条件内容需互斥，表达式内容会逐一匹配各 CASE 的条件内容，直到匹配成功，一旦匹配成功，相应分支 CASE 下的指令块只执行一次，然后程序跳转到 End select，结束 CASE 条件语句。如果写了 Case Else，则等 Case Else 以上的各分支 CASE 都没有匹配成功时，就会执行 Case Else 下的指令块。

参数： expression 表达式

expressionlist case 分支表达式条件内容

commands 指令块

注意： 各 case 分支表达式内容需互斥

例程

```
Dim choice As LONG

Select Case choice
Case 1
Print "number is 1"                    'choice 为 1 时，打印 number is 1
Case 2
Print "number is 2"                    'choice 为 2 时，打印 number is 2
Case 3, 4
Print "number is 3 or 4"               'choice 为 3 或 4 时，打印 number is 3 or 4
Case 5 To 10
Print "number is 5 to 10"              'choice 为 5~10 时，打印 number is 5 to 10
Case Else
Print "number is outsidethe range"     '非以上情况，打印 number is outside the range
End Select
```

2.1.6 WHILE...WEND

语法：WHILE condition
commands
WEND

描述：循环语句。当 condition 条件成立时，执行循环体内的指令块；否则，结束循环体。

参数：condition 条件表达式
commands指令块

例程

```
Dim i AS LONG
WHILE(i=5)          '如果 i=5，则在 WHILE 和 WEND 之间的指令段循环执行
   BASE 0
   SVON
   MOVE 1000
   WAIT DONE
   MOVE -1000
   WAIT DONE
WEND
```

2.1.7 WAIT

语法 1：WAIT [AX(no),]DONE

语法 2：WAIT [AX(no),] LATCHED

语法 3：WAIT [AX(no),] COMPARED

语法 4：WAIT [CYL(no),]CYLDONE

语法 5：WAIT [AX(no),] LTCBUFDONE

描述： WAIT 指令表示等待某个事件结束，WAIT 后面未指定轴或气/油缸时，等待的是当前 BASE 列表中的所有轴或气/油缸的相应事件。WAIT 指令后面跟的事件暂只支持 DONE、LATCHED、COMPARED、CYLDONE、LTCBUFDONE 五种事件。

参数： AX(no) 指定轴号，no 填轴号数字

DONE 指运动结束事件

LATCHED 指锁存发生事件

COMPARED 指比较触发事件

CYLDONE 指气缸动作完成事件

LTCBUFDONE 指锁存缓存中数据个数达到设定数量时触发，完整用法可参考 LBUF_DATA 指令

注意： WAIT 指令使用时要小心，未等到事件发生，程序会一直停在 WAIT 这行。特别是等待多个事件时，要确保事件都会发生。

例程

```
BASE 0,1,2
  MOVE 10000,20000,30000
  WAIT AX(2),DONE          '等待轴 2 运动结束后，程序执行下一行，否则一直等在该行

BASE 0,1
  MOVE 20000,5000
  WAIT DONE                '等待轴 0,1 运动都结束后，程序执行下一行，否则一直等在该行

BASE 0
  MOVE 10000
  WAIT DONE                '等待轴 0 运动结束后，程序执行下一行，否则一直等在该行
```

2.1.8 CancelWAIT

语法 1：CancelWAIT [AX(no),]DONE

语法 2：CancelWAIT[AX(no),] LATCHED

语法 3：CancelWAIT[AX(no),] COMPARED

语法 4：CancelWAIT [CYL(no),]CYLDONE

语法 5：CancelWAIT [AX(no),] LTCBUFDONE

描述： CancelWAIT 指令表示退出等待某个事件结束，一般是对应 WAIT 指令来使用的。当执行了 CancelWAIT 指令后，当前系统所有 Task 正在执行的 Wait 事件语句会退出等待，各 Task 程序会立刻接着执行 Wait 语句后面的程序行。

参数： AX(no) 指定轴号，no 填轴号数字

DONE 指运动结束事件

LATCHED 指锁存发生事件

COMPARED 指比较触发事件

CYLDONE 指气缸动作完成事件

LTCBUFDONE 指锁存缓存中数据个数达到设定数量时触发，完整用法可参考 LBUF_DATA 指令

例程

'例程有两个 Task，先执行 Task1 后，再执行 Task2

'***Task 1***' BASE 0 SVON FORWARD '执行正向连续运动 WAIT DONE '等待运动结束 STOPDEC '减速停止 ＊Task 1 开始执行后会，轴 0 一直处于正向连续运动状态，程序会等待在 WAIT DONE 指令行。	'***Task 2***' BASE 0 CancelWAIT DONE ＊当 Task 2 执行时，会执行 CancelWAIT DONE，此时 Task 1 中 WAIT DONE 指令就会退出等待，执行 STOPDEC 这行指令

2.1.9 SLEEP

语法：SLEEP delay_time

描述：延时指令

参数：delay_time　延时时间，单位：ms

例程

```
BASE 0
  MOVE 10000
  WAIT DONE
  SLEEP 500           '延时 500 毫秒
  MOVE -10000
```

2.1.10 TYPE

语法：TYPE type_name

描述：定义一个类（类似面向对象语言中的类）

参数：type_name 类的名称

例程

```
Type MyValueType              '定义一个 MyValueType 类
  count As Single
  sum As Single
  Declare Sub AddValue( ByVal x As Single )
  Declare Sub ShowResults( )
End Type

Sub MyValueType.AddValue( ByVal x As Single )
  count += 1
  sum += x
End Sub

Sub MyValueType.ShowResults( )
  Print "Number of Values = "; count
  Print "Average          = ";
  If( count > 0 ) Then
Print sum / count
  End If
End Sub

'主程序
Dim MyValue As MyValueType          '定义一个变量 MyValue, 变量类型为 MyValueType

MyValue.AddValue 17.5
MyValue.AddValue 20.1
MyValue.AddValue 22.3
MyValue.AddValue 16.9
MyValue.ShowResults                 '打印结果

'最终打印出来的结果如下
'Number of Values =  4
'Average          =  19.2
```

2.1.11 EXIT

语法： EXIT Do | For | While | Select

EXIT Sub | Function

EXIT DO[, DO[,...]]

EXIT For[, For[,...]]

EXIT While[, While[,...]]

EXIT Select[, Select[,...]]

描述： 退出一个控制流语句块或函数体。

例程

```
DIM i AS USHORT

FOR i=0 To 7
DOUT (i) =1
IF(i=5) THEN
       EXIT FOR          '当 i 为 5 时，结束 FOR 循环体
END IF
NEXT i

DIM As Integer i, j
For i = 1 To 10
    For j = 1 To 10
        Exit For, For    '嵌套的控制流语句块退出指令，结束 FOR 嵌套循环体
    Next j
    Print "I will never be shown"
Next i
```

2.1.12 CONTINUE

语法： CONTINUE Do | For | While

描述： 结束当前这次循环，如循环条件满足，将继续执行当前循环体。

例程

```
Dim As Integer n

Print "Here are odd numbers between 0 and 10!"
For n = 0 To 10
  If ( n Mod 2 ) = 0 Then
    Continue For        '当 n 是偶数时，结束当前这次循环，For 循环体继续执行。
  End If

  Print n           '因 n 是偶数时，循环被中断，所以打印出的数字为 1,3,5,7,9
Next n
```

2.1.13 TMR

定时器类。详情请参考“模块类”章节的 TMR 类说明。

、

2.2 子程序、多任务控制语句

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.2.1	SUB	定义一个函数体	×	×
2.2.2	END	结束当前任务或程序	×	×
2.2.3	RUN_TASK	运行指定的 Task	√	×
2.2.4	STOP_TASK	停止运行指定的 Task	√	×
2.2.5	STOP_ALL	停止运行所有 Task，并停止所有轴的运动	√	×
2.2.6	Task_Status	读取 Task 的状态	√	×
2.2.7	Task_Pause	暂停 Task 程序运行	×	×
2.2.8	Task_Resume	恢复运行已暂停的 Task	×	×
2.2.9	Function	带返回值的函数体	×	×
2.2.10	Return	从被调函数返回到主流程继续执行	×	×

2.2.1 SUB

语法：SUB　name

描述：定义一个函数体

参数：name 函数体名

注意：调用函数时，SUB 函数体必须要写在调用该函数之前，否则编译时会出错。如果想将定义的 SUB 函数体写在任意位置，可以在 TASK 开头用 **Declare Sub name** 语句先定义。

Declare 指令特别说明：

Declare 为声明的指令，用于声明 SUB（无返回值的函数体）和 Function（有返回值的函数体）,语法如下：

Declare Sub name [param_list]
Declare Function name [param_list] As return_type

例程

```
'例程 1：不传递参数的 SUB 使用
DECLARE SUB DO1_2sPulse      '声明 SUB: DO1_2sPulse
BASE 0
SVON
MOVEABS 100          '轴 0 移动到位置 100
WAIT DONE            '等待轴 0 运动结束
DO1_2sPulse          '执行 SUB DO1_2sPulse：DOUT(1)置 1 两秒后置 0
MOVEABS 0            '轴 0 移动到位置 0
WAIT DONE
'SUB DO1_2sPulse：DOUT(1)输出 1 个两秒的脉冲
SUB DO1_2sPulse
    DOUT(1)=1
    SLEEP 2000
    DOUT(1)=0
END SUB

'例程 2：需传递参数的 SUB 使用
DECLARE SUB AXIS_MOVE(ax AS USHORT)    '声明 SUB:AXIS_MOVE(ax AS USHORT)
AXIS_MOVE(0)   '执行 SUB AXIS_MOVE:轴 0 正向移动 100 个 UNIT
AXIS_MOVE(2)   '执行 SUB AXIS_MOVE:轴 2 正向移动 100 个 UNIT

'SUB AXIS_MOVE(ax AS USHORT)： 填入轴号，让该轴正向移动 100 个 UNIT
SUB AXIS_MOVE(ax AS USHORT)
    BASE ax
    MOVE 100
    WAIT DONE
END SUB
```

2.2.2 END

语法：END

描述：结束当前任务或程序，END 指令后面可以跟 SUB，IF 等

例程

```
'可以参考 IF、SUB 等指令例程
```

2.2.3 RUN_TASK

所属：命令

语法：RUN_TASK “task_name”

描述：运行指定的 Task

参数：task_name TASK 文本名称，此处名称不能加 .bas 后缀，只需填.bas 前面的名称

例程

'如用户创建了 3 个 TASK ,分别为 test1.bas ,test2.bas,test3.bas ,需在 test3.bas 里运行 test1.bas ,test2.bas 这两个 TASK，可在 test3.bas 里写如下**例程**

```
******Task3******
RUN_TASK "test1"
RUN_TASK "test2"
```

2.2.4 STOP_TASK

所属：命令

语法：STOP_TASK “task_name”

描述：停止运行指定的 Task

参数：task_name TASK 文本名称，此处名称不能加 .bas 后缀，只需填.bas 前面的名称

例程

```
STOP_TASK "test"      ' 停止运行名为 test 的 TASK
```

2.2.5 STOP_ALL

所属：命令

语法：STOP_ALL

描述：停止运行所有 Task，并停止所有轴的运动

例程

```
RUN_TASK "test1"  ' 运行名为 test1 的 TASK
RUN_TASK "test2"  ' 运行名为 test2 的 TASK
Sleep 1000        '延时 1 秒
STOP_ALL          ' 停止运行所有 TASK，并停止所有轴运动
```

2.2.6 Task_Status

语法：value=Task_Status (taskname)

描述：读取 Task 的状态

参数：taskname 要读取状态的 Task 名称；**类型：**String

返回值：0：停止；1：运行中　　**类型：**ULONG

例程

```
DIM A AS ULONG
A=Task_Status("test1")        'test1 为其中一个 Task 的名称
```

2.2.7 Task_Pause

所属：命令

语法：Task_Pause no

描述：暂停 Task 程序运行。Task_Pause 指令被执行时，该指令所在的 Task 就会被暂停运行。该指令不能暂停其它 Task 运行。Task_Pause 后面的编号可以是 ULONG 类型范围内的数值，一个 Task 中可以有多个编号的 Task_Pause ,各自独立 ,该指令需与 Task_Resume 指令配套使用 ,配套的 Task_Pause 和 Task_Resume 编号需一致。

参数：no Task_Pause 编号；**类型：**ULONG

注意：该指令使用不好容易造成程序错乱，不推荐使用。

例程

```
'有 2 个 Task。Task0 和 Task1 ,Task0 负责恢复 Task 运行,Task1 负责在需要暂停的地方添加暂停指令。
' VR(0)的值为 1 时，代表上位界面“恢复运行按钮”按下
' VR(1)的值为 1 时，代表上位界面“暂停按钮”按下
'-------------------------------------------------------
'Task0 的程序：循环检测“暂停恢复按钮”是否有被按下，如按下，则恢复运行 Task1 程序
WHILE 1
    IF(VR(0)=1) THEN
        VR(0)=0
        TASK_RESUME 1
    END IF
    SLEEP 10
WEND
'-------------------------------------------------------
'Task1 的程序
SUB PAUSE_Point()          '定义 1 个名为 PAUSE_Point()的 SUB，用做暂停节点
    IF(VR(1)=1) THEN
        VR(1)=0
        TASK_PAUSE 1
    END IF
END SUB

WHILE 1
    BASE 0,1
    MOVEABS 10000,5000
    WAIT DONE
    PAUSE_Point()          '插入 1 个暂停节点，如执行该语句前发生暂停，TASK1 将执行该句后暂停

    MOVEABS -10000,-5000
    WAIT DONE
    PAUSE_Point()          '插入 1 个暂停节点，如执行该语句前发生暂停，TASK1 将执行该句后暂停

    MOVEABS 20000,20000
    WAIT DONE

    SLEEP 10
```

```
WEND
```

2.2.8 Task_Resume

所属：命令

语法：Task_Resume　no

描述：恢复运行对应编号的 Task 暂停程序。Task_Resume 指令需与 Task_Pause 指令在不同的 Task。

参数：no　Task_Resume 编号；**类型：**ULONG

注意：该指令使用不好容易造成程序错乱，不推荐使用。

例程

'请参考 Task_Pause 指令章节例程。

2.2.9 Function

语法：Function lable( ) As DataType

描述：定义一个有返回值的函数体

注意：调用 Function 时，Function 函数体必须要写在调用该函数之前，否则编译时会出错。如果要将 Function 函数体写在任意位置，可以在 TASK 开头用 **Declare** 先声明。

例程

```
DECLARE FUNCTION Test() AS INTEGER    '声明 Test() FUNCTION
DIM A AS INTEGER=0    '声明一个整型变量 A
VR(0)=5          '将 VR(0)赋值为 5
A=Test()         '将 Test()执行后的返回值传给 A
PRINT A          '因 VR(0)的值为非 0，所以此时 A 的值为 1

VR(0)=0          '将 VR(0)赋值为 0
A=Test()         '将 Test()执行后的返回值传给 A
PRINT A          '因 VR(0)的值为 0，所以此时 A 的值为 2

'Function 里对 VR(0)做了判断，VR(0)的值为非 0 时返回 1，为 0 时返回 2
FUNCTION Test() AS INTEGER
    IF(VR(0)<>0) Then
        Return 1
    else
        Return 2
    End IF
End FUNCTION

'带参数传递的 Function 使用方法请参考 SUB 指令章节的例程用法
```

2.2.10 Return

语法：Return [*expression*]

描述：从被调函数返回到主程序继续执行，从 Function 中返回时，可附带一个 Function 的返回值。

例程

```
'例程 1：从 SUB 中 Return
DECLARE SUB DO1_2sPulse      '声明 SUB: DO1_2sPulse
BASE 0
SVON
DO1_2sPulse     '执行 SUB DO1_2sPulse：DOUT(1)置 1 两秒后置 0
MOVE 1000       '轴 0 移动到位置 1000
WAIT DONE

'SUB DO1_2sPulse：DOUT(1)输出 1 个两秒的脉冲
SUB DO1_2sPulse
    '如果 DIN(0)为 1，跳出这个 SUB，主程序 SUB DO1_2sPulse 后的代码继续执行
    IF(DIN(0)=1) THEN Return
    DOUT(1)=1
    SLEEP 2000
    DOUT(1)=0
END SUB


'例程 2：从 Function 中 Return 返回值
```

请参考 Function 指令章节的例程

2.3 运算符及数学函数

2.3.1 运算符

当表达式包含多种运算符时，首先计算算术运算符，然后计算比较运算符，最后计算逻辑运算符。所有比较运算符的优先级相同，即按照从左到右的顺序计算比较运算符。

当乘号与除号同时出现在一个表达式中时，按从左到右的顺序计算乘、除运算符。同样当加与减同时出现在一个表达式中时，按从左到右的顺序计算加、减运算符。

算术运算符说明如表 3.1 所示。

表 3.1 运算符说明

算术运算符		比较运算符		逻辑运算符	
描述	符号	描述	符号	描述	符号
加	+	等于	=	逻辑非	NOT
减	-	不等于	<>	逻辑与	AND
乘	*	小于	<	逻辑或	OR
除	/	大于	>	逻辑异或	XOR
整除	\	小于等于	<=	逻辑等价	EQV
求余数	Mod	大于等于	>=		
求相反数	-				
幂	^				

2.3.1.1 NOT

语法：NOT expression

描述：对表达式进行非操作或对数值按二进制的位进行“非”运算

参数：expression 表达式

注意：只针对表达式的值的整数部分运算

2.3.1.2 AND

语法：expression1 AND expression2

描述：对两个表达式进行与操作或对两个数值按二进制的位进行“与”运算

参数：expression1 表达式 1

expression2 表达式 2

注意：只针对表达式的值的整数部分运算

2.3.1.3 OR

语法：expression1 OR expression2

描述：对两个表达式进行或操作或对两个数值按二进制的位进行“或”运算

参数：expression1 表达式 1

expression2 表达式 2

注意：只针对表达式的值的整数部分运算

2.3.1.4 XOR

语法：expression1 XOR expression2

描述：对两个表达式进行异或操作或对两个数值按二进制的位进行“异或”运算

参数：expression1 表达式 1

expression2 表达式 2

注意：只针对表达式的值的整数部分运算

2.3.1.5 EQV

语法：expression1 EQV expression2

描述：对两个表达式进行同或操作或对两个数值按二进制的位进行“同或”运算

参数：expression1　　表达式 1

expression2　　表达式 2

注意：只针对表达式的值的整数部分运算

2.3.2 数学函数

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.3.2.1	ABS	求绝对值	×	×
2.3.2.2	SIN	正弦函数	×	×
2.3.2.3	ASIN	反正弦函数	×	×
2.3.2.4	COS	余弦函数	×	×
2.3.2.5	ACOS	反余弦函数	×	×
2.3.2.6	TAN	正切函数	×	×
2.3.2.7	ATN	反正切函数	×	×
2.3.2.8	ATAN2	用比值求反正切函数	×	×
2.3.2.9	SQR	求平方根	×	×
2.3.2.10	LOG	自然对数	×	×
2.3.2.11	BITRESET	位操作置 0	×	×
2.3.2.12	BIT	位操作读值	×	×
2.3.2.13	BITSET	位操作置 1	×	×
2.3.2.14	FRAC	求小数部分的值	×	×
2.3.2.15	INT	求整数部分的值	×	×
2.3.2.16	SGN	判断值的符号	×	×

2.3.2.1 ABS

语法：ABS(expression)

描述：求表达式的绝对值

参数：expression　　表达式

2.3.2.2 SIN

语法：SIN(expression)

描述：计算表达式的正弦函数

参数：expression　　表达式，单位：弧度

例程

```
CONST PI AS DOUBLE = 3.1415926535897932
DIM a AS DOUBLE           '定义一个变量用于存角度值
DIM r AS DOUBLE           '定义一个变量用于存弧度值
DIM sin_value as DOUBLE   '定义一个变量用于取正弦值
a=90
r = a*PI/180              '把 a 转换成弧度值存于 r
sin_value=SIN (r)
PRINT r
PRINT sin_value
```

运行结果：

```
r 为 1.570796326794897
sin_value 为 1
```

2.3.2.3 ASIN

语法：ASIN(expression)

描述：计算表达式的反正弦函数，返回值单位：弧度

参数：expression　　表达式

2.3.2.4 COS

语法：COS(expression)

描述：计算表达式的余弦函数

参数：expression　　表达式，单位：弧度

2.3.2.5 ACOS

语法：ACOS (expression)

描述：计算表达式的反余弦函数，返回值单位：弧度

参数：expression 表达式

2.3.2.6 TAN

语法：TAN(expression)

描述：计算表达式的正切函数

参数：expression 表达式，单位：弧度

2.3.2.7 ATN

语法：ATN(expression)

描述：计算表达式的反正切函数，返回值单位：弧度

参数：expression

2.3.2.8 ATAN2

语法：ATAN2 (number1，number2)

描述：计算 number1/number2 比值的反正切函数，返回值单位：弧度

参数：number1 比值分子

number2 比值分母

2.3.2.9 SQR

语法：SQR(expression)

描述：计算表达式的平方根

参数：expression 表达式

2.3.2.10 LOG

语法：LOG (expression)

描述：计算表达式的自然对数（以 e 为底的对数）

参数：expression 表达式

2.3.2.11 BITRESET

语法：BITRESET(value，bit_num)

描述：将操作数的二进制的第 bit 位清 0

参数：bit_num 位编号：0~31（二进制数的位，从低（右）至高（左）排列）

Value 操作数

注意：只针对操作数的整数部分操作

例程

```
DIM AS ULONG a,b
a=5
b=BITRESET(a,0)        'a 为 5，二进制即 101，清掉 0 位，即 b 得到 100
PRINT b                'b 为 100，打印出来即为 4
b=BITRESET(a,2)        'a 为 5，二进制即 101，清掉 2 位，即 b 得到 001
PRINT b                'b 为 001，打印出来即为 1
```

2.3.2.12 BIT

语法：BIT(value，bit_num)

描述：读取操作数的二进制的第 bit 位的值，读到 bit 位为 0 值，返回 0；读到 bit 位为 1 值，返回-1

参数：bit_num 位编号：0~31（二进制数的位，从低（右）至高（左）排列）

Value 操作数

注意：只针对操作数的整数部分操作

2.3.2.13 BITSET

语法：BITSET(value，bit_num)

描述：将操作数的二进制的第 bit 位的置 1

参数：bit_num 位编号：0~31（二进制数的位，从低（右）至高（左）排列）

Value 操作数

注意：只针对操作数的整数部分操作

2.3.2.14 FRAC

语法：FRAC(expression)

描述：返回表达式的小数部分

参数：expression 表达式

注意：此函数仅支持大于 0 的表达式

2.3.2.15 INT

语法：INT(expression)

描述：返回表达式的整数部分

参数：expression 表达式

注意：当表达式的值小于 0 时，返回值比其整数小 1

2.3.2.16 SGN

语法：SGN(expression)

描述：判断表达式是大于 0、等于 0，还是小于 0。当表达式大于 0，函数的返回值为 1；当表达式等于 0 时，函数的返回值为 0；当表达式小于 0，函数的返回值为 - 1

参数：expression 表达式

2.4 控制器系统指令

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.4.1	BASE	该指令后面所有的轴指令和轴参数设置和读取都基于该指令选定的轴号	√	×
2.4.2	UNIT_NUM	UNIT 分子	√	√
2.4.3	UNIT_DENOM	UNIT 分母	√	√
2.4.4	POUT_MODE	指令脉冲输出类型	√	√
2.4.5	POUT_REVERSE	指令脉冲输出逻辑反相	√	√
2.4.6	PIN_MODE	编码器脉冲输入类型	√	√
2.4.7	PIN_MAXFREQ	编码器脉冲输入的最高频率限值	√	√
2.4.8	PIN_LOGIC	编码器脉冲输入逻辑反相	√	√
2.4.9	DPOS	理论位置值（指令脉冲）	√	√
2.4.10	MPOS	实际位置值（编码器回馈脉冲）	√	√
2.4.11	SVON	使能伺服	√	×
2.4.12	SVOFF	禁用使能伺服	√	×
2.4.13	ERROR_AXIS	当前哪些轴发生了轴状态错误	√	×
2.4.14	RUN_ERROR	轴错误信息	√	×
2.4.15	SYSTEM_ERROR	系统级错误信息	√	×
2.4.16	CLEAR_ERROR	清除系统错误状态	√	×
2.4.17	PRINT	在 Motion Studio 中的信息输出窗体打印信息	×	×
2.4.18	DATE	获取当前控制器日期：月-日-年	×	×
2.4.19	TIME	获取当前控制器日期：小时 ： 分钟 ： 秒	×	×
2.4.20	SETDATE	给控制器系统设置新的日期	×	×
2.4.21	SETTIME	给控制器系统设置新的时间	×	×
2.4.22	TIMER	返回程序段程序执行的时间	×	×
2.4.23	CLEAR_3DPATH	清除 3D 轨迹工具中的轨迹	√	×
2.4.24	Pt 类	位置点（P 点）类	×	×

2.4.25	AxIsReady	判断哪些轴是否处于 Ready 状态	×	×

2.4.1 BASE

所属：命令

语法：BASE　axis no [,second axis][,third axis] ...

描述：为了简化编程，可以用该指令选择要参与运动的轴号，其后的指令就没必要填写所有轴的参数，只填写参与运动的轴参数即可。

参数：axis no 轴号；**范围：**根据控制器实际硬件决定。

注意：1 个 BASE 后面参与的轴数最多是 8 个轴。

例程

```
BASE 0,1,2,3
VH=8000                          '轴 0,1,2,3 的最大运行速度都设置为 8000
MOVE 10000,4000,2000,6000        '轴 0,1,2,3 都执行单轴相对点位运动
WAIT DONE                        '等待轴 0,1,2,3 运动结束
BASE 1
VH=1000                          '轴 1 的最大运行速度设置为 1000
MOVE 10000                       '轴 1 执行单轴相对点位运动
WAIT DONE
```

2.4.2 UNIT_NUM

所属：属性

语法：UNIT_NUM = value

类型：ULONG

描述：设置/读取 UNIT 分子

范围：大于 0，默认值 1

例程

```
BASE 0
UNIT_NUM  =10   '设置轴 0 的 UNIT 分子
```

2.4.3 UNIT_DENOM

所属：属性

语法：UNIT_DENOM = value

类型：ULONG

描述：设置/读取 UNIT 分母

范围：(0, MAX_PULSE)，默认值 1

例程

```
BASE 0
UNIT_DENOM=40  '设置轴 0 的 UNIT 分母为 40
```

2.4.4 POUT_MODE

所属：属性

语法：POUT_MODE= value

类型：ULONG

描述：设置/读取指令脉冲输出类型

范围：如下设定值，默认值 5

0：OUT/DIR

1：OUT/DIR，OUT 负逻辑

2：OUT/DIR，DIR 负逻辑

3：OUT/DIR，OUT&DIR 负逻辑

4：CW/CCW

5：CW/CCW，CW&CCW 负逻辑

例程

```
BASE 0
POUT_MODE =2 '设置指令脉冲输出类型为 OUT/DIR，DIR 负逻辑
```

2.4.5 POUT_REVERSE

所属：属性

语法：POUT_REVERSE = value

类型：ULONG

描述：启用/禁用指令脉冲输出端口信号对调

范围：如下设定值，默认值 0

0：禁用

1：启用

例程

```
BASE 0
POUT_REVERSE =1 '启用指令脉冲输出端口信号对调
```

2.4.6 PIN_MODE

所属：属性

语法：PIN_MODE= value

类型：ULONG

描述：设置/读取编码器输入脉冲类型

范围：如下设定值，默认值 2

0：1XAB

1：2XAB

2：4XAB

3：CCW/CW

例程

```
BASE 0
PIN_MODE =3 '设置编码器输入脉冲类型为 CCW/CW
```

2.4.7 PIN_MAXFREQ

所属：属性

语法：PIN_MAXFREQ = value

类型：ULONG

描述：设置/读取编码器输入脉冲的最高频率

范围：如下设定值，默认值 0

0：500KHz

1：1MHz

2：2MHz

3：4MHz

例程

```
BASE 0
PIN_MAXFREQ =1     '设置编码器输入脉冲的最高频率为 1MHz
```

2.4.8 PIN_LOGIC

所属：属性

语法：PIN_LOGIC = value

类型：ULONG

描述：设置/读取编码器输入脉冲的逻辑

范围：如下设定值，默认值 0

0：不反转方向

1：反转方向

例程

```
BASE 0
PIN_LOGIC =1  '设置编码器输入脉冲逻辑为反转方向
```

2.4.9 DPOS

所属：属性

语法：DPOS = value

类型：DOUBLE

描述：设置/读取轴当前的理论位置

范围：64 位浮点数据类型范围

注意：EtherCAT 控制器中，如果将 DPOS 清零，不同厂牌的伺服驱动器，产生的效果会不一样。有些厂牌的伺服不允许清位置，会引起报错。有些厂牌的伺服会执行 CiA402 定义的模式 35 回原点动作。

例程

```
BASE 0
DIM A AS DOUBLE
A=DPOS   '将轴 0 的当前理论位置赋值给变量 A
BASE 1
VR(10)=DPOS  '将轴 1 的当前理论位置赋值给全局变量 VR（10）
DPOS=0  '将轴 1 的当前理论位置计数器赋 0

'不使用 BASE，将轴的 DPOS 值赋给变量。
VR(11)=DPOS(2)  '将轴 2 的 DPOS 赋值给 VR(11)
```

2.4.10 MPOS

所属：属性

语法：MPOS = value

类型：DOUBLE

描述：设置/读取轴当前的编码器回馈位置

范围：64 位浮点数据类型范围

例程

```
BASE 0
DIM A AS DOUBLE
A=MPOS    '将轴 0 的当前实际位置赋值给变量 A
BASE 1
VR(10)=MPOS  '将轴 1 的当前实际位置赋值给全局变量 VR（10）
MPOS=0  '将轴 1 的当前实际位置计数器赋 0

'不使用 BASE，将轴的 MPOS 值赋给变量。
VR(11)=MPOS(2)  '将轴 2 的 MPOS 赋值给 VR(11)
```

2.4.11 SVON

所属：命令

语法 1：SVON

语法 2：SVON AX(axis no)

描述：BASE 轴列表的轴或指定轴，使能轴

参数：axis no 轴号；**范围：**根据控制器实际硬件决定

例程

```
'对 BASE 列表中的轴使能，即使能伺服
BASE 2
SVON            '使能轴 2
BASE 0,1,3,5
SVON            '使能轴 0、1、3、5
'指定轴使能
SVON AX(2)          '使能轴 2
```

2.4.12 SVOFF

所属：命令

语法 1：SVOFF

语法 2：SVOFF AX(axis no)

描述：BASE 轴列表的轴或指定轴，禁用轴使能

参数：axis no 轴号；**范围：**根据控制器实际硬件决定

例程

```
'对 BASE 列表中的禁用轴使能
BASE 2
SVOFF        '禁用轴 2 使能
BASE 0,1,3,5
SVOFF       '禁用轴 0、1、3、5 使能
SVOFF AX(2)     '禁用轴 2 使能
```

2.4.13 ERROR_AXIS

所属：属性 (只读)

语法：value=ERROR_AXIS

类型：ULONG

描述：读取当前哪些轴发生了轴状态错误。该属性为 32 位寄存器，每一位代表一个轴。位值为 0 表示该轴无错误发生，位值为 1 表示该轴发生了轴状态错误。当发生轴状态错误时，可以用 RESETERR 指令清除轴状态错误。

返回值：0：无错误发生；1：发生了轴状态错误

例程

```
DIM ErrorReturn As ULONG
ErrorReturn=ERROR_AXIS
IF (ErrorReturn=4) THEN  'ErrorReturn 为 4 时，表示轴 2 发生了轴状态错误
RESETERR AX(2)          '清除轴 2 的轴状态错误
End if
```

2.4.14 RUN_ERROR

所属：属性 (只读)

语法：value=RUN_ERROR

类型：ULONG

描述：根据 BASE 轴列表中的轴，读取轴错误信息。错误代码信息请参照章节 2.15 RUN_ERROR 错误代码信息表。

返回值：错误代码

例程

```
DIM ErrorCode As ULONG
BASE 0
ErrorCode = RUN_ERROR
```

2.4.15 SYSTEM_ERROR

所属：属性 (只读)

语法：value=SYSTEM_ERROR

类型：ULONG

描述：读取系统级错误信息。错误代码信息请参考章节 2.15 SYSTEM_ERROR 错误代码信息表

返回值：错误代码

例程

```
DIM ErrorCode As ULONG
ErrorCode = SYSTEM_ERROR
```

2.4.16 CLEAR_ERROR

所属：命令

语法：CLEAR_ERROR

描述：清除系统错误状态。该命令仅用于清除 SYSTEM_ERROR 对应的系统错误状态

2.4.17 PRINT

所属：命令

描述：在 Motion Studio 中的信息输出窗体打印信息。

例程

```
Dim A As ULONG
A=42
Print "Hello" '打印字符串
Print VL   '打印初速度属性值
Print A    '变数值
Print 3*4  '打印一个表达式结果，打印结果为 12
```

2.4.18 DATE

语法：value=DATE

类型：String

描述：获取当前控制器日期：月-日-年

例程

```
DIM str1 AS string
str1=DATE
print str1     '如现在是 2016 年 3 月 15 日，打印结果为：03-15-2016
Print "the current date is: "; DATE    '打印结果：the current date is: 03-15-2016
```

2.4.19 TIME

语法：value=TIME

类型：String

描述：获取当前控制器日期：小时 ：分钟 ：秒

例程

```
DIM str1 AS string
str1=TIME
print str1     '如现在时间是 14 点 22 分 51 秒，打印结果为：14:22:51
Print "the current time is: "; TIME    '打印结果：the current date is: 14:22:51
```

2.4.20 SETDATE

语法：SETDATE(newdate)

描述：给控制器系统设置新的日期

参数：newdate 新的日期，类型为 string

例程

```
SETDATE "03/15/2016" '将当前控制器系统日期设置为 2016 年 3 月 15 号
```

2.4.21 SETTIME

语法：SETTIME(newtime)

描述：给控制器系统设置新的时间

参数：newtime 新的时间，类型为 string

例程

```
SETTIME"14:20:31" '将当前控制器系统时间设置为 14 点 20 分 31 秒
```

2.4.22 TIMER

语法：value=TIMER

描述：以秒为单位，返回开始参考的时间到现在消逝的时间总和。该功能主要用于计算 TASK 里一个程序段跑了多少时间，从而去找出从程序一行执行到另一行所花的时间。

例程

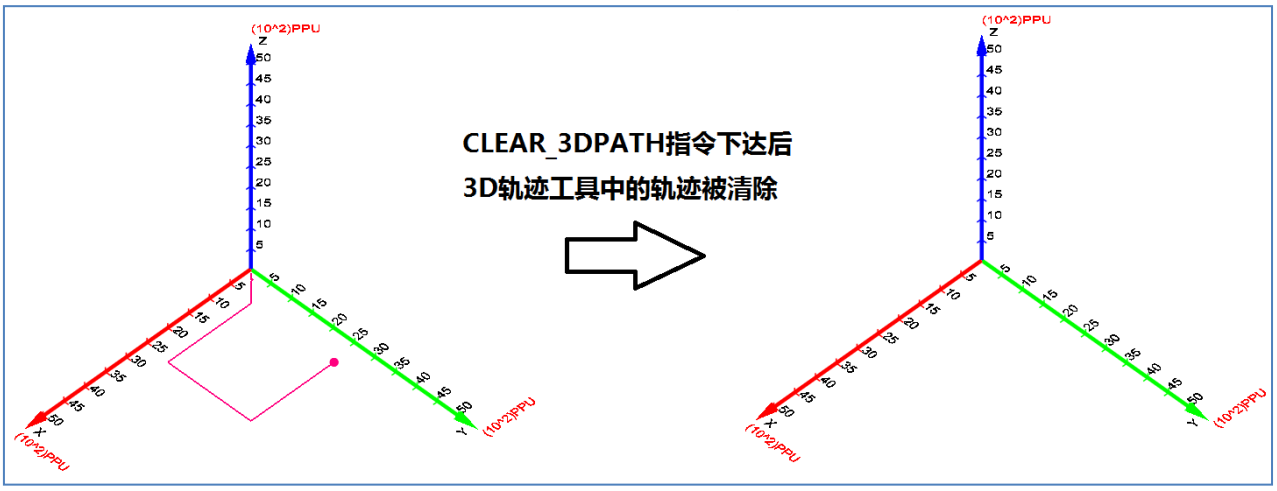
```
Dim Start As Double
Print "Wait 2.5 seconds."
Start = Timer
Do
Sleep 1, 1
Loop Until (Timer - Start) > 2.5
Print "Done."
```

2.4.23 CLEAR_3DPATH

所属：命令

语法：CLEAR_3DPATH

描述：清除 Motion Studio 中“3D 轨迹工具”中的轨迹。



2.4.24 Pt

位置点（P 点）类。详情请参考“模块类”章节的 Pt 类说明。

2.4.25 AxIsReady

语法：value=AxIsReady (axis no [,second axis][,third axis] ...)

描述：用于判断哪些轴是否处于 Ready 状态，该指令经常用于等待哪些轴运动完成，相比用 Wait Done 指令，该指令可以写成让程序不阻塞的方式去处理。该指令适用于 16 轴以内的任意组合。

参数：axis no 轴号

返回值：0：指定轴中至少有一个未处于 Ready 状态；1：指定的所有轴都处于 Ready 状态

例程

'简易示例如：AxIsReady(1,5,6,7)；AxIsReady(1,2)；AxIsReady(0)等。

'下面是一段顺序动作流程程序中 AxIsReady 的写法

```
WHILE 1
SELECT CASE  CINT(VR(0)) '通过控制 VR(0)的值控制步骤
    CASE 0                          'CASE0: 轴 1 单轴运动到 100
        BASE 1
        MOVEABS 100                 '轴 1 单轴运动到 100
        VR(0)=1                     '切到下一步
    CASE 1                          'CASE1: 轴 2,3 直线插补到 10,20
        IF(AxIsReady(1)=1) THEN     '判断上一步轴动作是否运动完成
            BASE 2,3
            LINEABS 10,20           '轴 2,3 直线插补到 10,20
            VR(0)=2                 '切到下一步
        END IF
    CASE 2                          'CASE2: 轴 0,1,3 单轴运动到 30,30,30
        IF(AxIsReady(2,3)=1) THEN   '判断上一步轴动作是否运动完成
            BASE 0,1,3
            MOVEABS 30,30,30        '轴 0,1,3 单轴运动到 30,30,30
            VR(0)=3                 '切到下一步
        END IF
    CASE 3                          'CASE3: 结束步，打印一个 Done 信息
        IF(AxIsReady(0,1,3)=1) THEN '判断上一步轴动作是否运动完成
            PRINT "Done"            '打印 Done
            VR(0)=4                 '切到一下步,可以切到一个空步
        END IF
    CASE 4

END SELECT
SLEEP 10
WEND
```

2.5 单轴点位运动

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.5.1	VL	单轴初速度	√	√
2.5.2	VH	单轴运行速度	√	√
2.5.3	ACC	单轴加速度	√	√
2.5.4	DEC	单轴减速度	√	√
2.5.5	JK	单轴速度曲线类型	√	√
2.5.6	DSPEED	当前轴指令速度值	√	√
2.5.7	STATE	轴当前运动状态	√	√
2.5.8	MOVE	单轴相对点位运动	√	×
2.5.9	MOVEABS	单轴绝对点位运动	√	×
2.5.10	PCHANGE	单轴运动过程中改变终点位置	√	×
2.5.11	STOPDEC	减速停止	√	×
2.5.12	STOPEMG	立即停止	√	×
2.5.13	BACKLASH_EN	背隙补偿功能使能	√	×
2.5.14	BACKLASH_PULSE	背隙补偿距离	√	√
2.5.15	BACKLASH_VEL	背隙补偿过程运动速度	√	√
2.5.16	MAXVEL	单轴最大速度限值	√	√
2.5.17	MAXACC	单轴最大加速度限值	√	√
2.5.18	MAXDEC	单轴最大减速度限值	√	√
2.5.19	RESETERR	清除当前轴错误状态	√	×

2.5.1 VL

所属：属性

语法：VL = value

类型：DOUBLE

描述：设置/读取轴的初速度，VL 的单位为 UNIT/s

范围：【0,MAXVEL】，默认值 2000

例程

```
BASE 0
VL=2000 '设置轴 0 的初速度为 2000 个 UNIT/s
```

2.5.2 VH

所属：属性

语法：VH = value

类型：DOUBLE

描述：设置/读取轴的最大运行速度，VH 的单位为 UNIT/s

范围：（VL,MAXVEL），默认值 8000

例程

```
BASE 0
VH=2000 '设置轴 0 的运行速度为 2000 个 UNIT/s
```

2.5.3 ACC

所属：属性

语法：ACC = value

类型：DOUBLE

描述：设置/读取轴的加速度，ACC 的单位为 UNIT/s^2

范围：（0,MAXACC），默认值 10000

例程

```
BASE 0
ACC=20000 '设置轴 0 的加速度为 20000 个 UNIT/s^2
```

2.5.4 DEC

所属：属性

语法：DEC = value

类型：DOUBLE

描述：设置/读取轴的减速度，DEC 的单位为 UNIT/s^2

范围：（0,MAXDEC），默认值 10000

例程

```
BASE 0
DEC=20000 '设置轴 0 的减速度为 20000 个 UNIT/s^2
```

2.5.5 JK

所属：属性

语法：JK = value

类型：ULONG

描述：设置/读取单轴点位运动的速度曲线类型

范围：【0,1】，0：T 型曲线；1：S 型曲线，默认值 0

例程

```
BASE 0
JK=1    '设置轴 0 单轴点位元运动的速度曲线类型为 S 型曲线
```

2.5.6 DSPEED

所属：属性（只读）

语法：value = DSPEED

类型：DOUBLE

描述：根据当前 BASE 列表中的轴，读取轴当前指令理论速度

例程

```
BASE 0
DIM A AS DOUBLE
A=DSPEED   '将轴 0 的当前指令速度赋值给变量 A
```

2.5.7 STATE

所属：属性(只读)

语法：value = STATE

类型：ULONG

描述：读取当前轴运动状态。

返回值：如下

0：轴不可用状态

1：Ready 状态

2：轴停止状态，但未 Ready

3：轴处于错误状态，轴被停止运动

4：轴正在执行回原点运动中

5：轴正在执行单轴点位运动中

6：轴正在执行单轴连续运动中

7：轴正在参与插补运动中或同步运动中

8：轴处于外部 JOG 模式中

9：轴处于外部 MPG 模式中

例程

```
BASE 0
DIM A AS ULONG
A=STATE    '将轴 0 的当前轴运动状态对应的值赋值给变量 A
A=STATE(AX(2))     '将轴 2 的当前轴运动状态对应的值赋值给变量 A。A=STATE AX(2)这种语法不对。
```

2.5.8 MOVE

所属：命令

语法 1：MOVE distance1[,distance2][,distance3].....

语法 2：MOVE AX(axis no)，distance

描述： BASE 轴列表的轴，以当前位置为原点，在相对坐标下运动到指定距离的位置。

参数： distance 相对移动距离；**类型：**DOUBLE

axis no 轴号；**范围：**根据控制器实际硬件决定。

例程

```
'选择要操作的轴，并设定速度等相关参数
BASE 0,1,2
VL=2000
VH=8000
ACC=10000
DEC=10000
'开始轴 0 的相对运动
BASE 0
MOVE 1000
WAIT DONE
'开始轴 2,3 的相对运动
BASE 1,2
MOVE 2000, 3000
WAIT DONE
'指定轴 1，开始相对运动
MOVE AX(1),5000
'使用 P 点进行运动
Dim P0 As Pt              '实例化出一个位置点对象 P0
P0=Pt(1000,2000)          '位置点赋值为(1000,2000)
BASE 0,1
MOVE P0                   '轴 0、轴 1 分别进行相对运动的距离为 1000，2000
WAIT DONE
```

2.5.9 MOVEABS

所属：命令

语法 1：MOVEABS position1[，position2] [，position3]……

语法 2：MOVEABS AX(axis no)，positon

描述：BASE 轴列表的轴，在绝对坐标下运动到的指定位置。

参数：position 绝对位置；**类型：**DOUBLE

axis no 轴号；**范围：**根据控制器实际硬件决定。

例程

```
'选择要操作的轴，并设定速度等相关参数
BASE 0,1,2
VL=2000
VH=8000
ACC=10000
DEC=10000
'开始轴 0 的绝对运动
BASE 0
MOVEABS 1000
WAIT DONE
'开始轴 2,3 的绝对运动
BASE 1,2
MOVEABS 2000, 3000
WAIT DONE
'指定轴 1，开始绝对运动
MOVEABS AX(1),5000
'使用 P 点进行运动
Dim P0 As Pt                    '实例化出一个位置点对象 P0
P0=Pt(1000,2000,3000)           '位置点赋值为(1000,2000,3000)
BASE 0,1,2
MOVEABS P0                      '轴 0、1、2 绝对运动到位置点(1000,2000,3000)
WAIT DONE
```

2.5.10 PCHANGE

所属：命令

语法 1：PCHANGE distance

语法 2：PCHANGE AX(axis no) ,distance

描述：修改当前运动的终点位置，如果当前轴不在运动中，下该指令会报错。

参数：distance 改变相对运动距离；**类型：**DOUBLE

axis no 轴号；**范围：**根据控制器实际硬件决定。

例程

```
BASE 0
MOVE 20000
SLEEP 200
PCHANGE 25000       '改变位移为 25000
WAIT DONE
BASE 0,1
SVON
MOVE 10000,10000
SLEEP 200
PCHANGE AX(1),5000  '改变位移为 5000
WAIT DONE
```

2.5.11 STOPDEC

所属：命令

语法 1：STOPDEC

语法 2：STOPDEC　AX(axis no)，dec

语法 3：STOPDEC　dec1，dec2，dec3，……，dec32

描述：BASE 轴清单中的轴或指定轴、指定减速度对轴下减速停止运动命令

参数：dec 减速度；**类型：**DOUBLE

axis no 轴号；**范围：**根据控制器实际硬件决定。

例程

```
BASE 0,1,2
SVON
VL=1000
VH=10000
ACC=50000
DEC=50000
'多个轴下减速停止运动命令
MOVE 40000,20000,35000
SLEEP 1000
STOPDEC
WAIT DONE
'指定轴下减速停止运动命令
FORWARD AX(0)
SLEEP 2000
STOPDEC AX(0)
WAIT AX(0),DONE
'指定多个轴用新的减速度下减速停止运动命令
MOVE 40000,30000,35000
SLEEP 2000
STOPDEC 200000,200000,200000
```

2.5.12 STOPEMG

所属：命令

语法 1：STOPEMG

语法 2：STOPEMG　AX(axis no)

描述：BASE 轴列表中的轴或指定轴对轴下立即停止运动命令

参数：axis no 轴号；**范围：**根据控制器实际硬件决定。

例程

```
BASE 0,1,2
SVON
VL=1000
VH=10000
ACC=50000
DEC=50000
'多个轴下立即停止运动命令
MOVE 40000,20000,35000
SLEEP 1000
STOPEMG
WAIT DONE
'指定轴下立即停止运动命令
FORWARD AX(0)
SLEEP 2000
STOPEMG AX(0)
WAIT AX(0),DONE
```

2.5.13 BACKLASH_EN

所属：属性

语法：BACKLASH_EN = value

类型：ULONG

描述：启用/禁用背隙补偿功能

范围：如下设定值，默认值 0

0：禁用

1：启用

例程

```
BASE 0
BACKLASH_EN=1 '启用轴 0 的背隙补偿功能
```

2.5.14 BACKLASH_PULSE

所属：属性

语法：BACKLASH_PULSE = value

类型：ULONG

描述：设置/读取背隙补偿的脉冲个数

范围：【0,4095】，默认值 10

例程

```
BASE 0
BACKLASH_PULSE=10 '设置轴 0 的背隙补偿的脉冲个数为 10 个
```

2.5.15 BACKLASH_VEL

所属：属性

语法：BACKLASH_VEL = value

类型：ULONG

描述：设置/读取进行补偿距离移动时的速度，单位元为脉冲/s

范围：（0,MAXVEL），默认值 1000

例程

```
BASE 0
BACKLASH_VEL=1000 '设置轴 0 的背隙补偿速度为 1000 个脉冲/s
```

2.5.16 MAXVEL

所属：属性

语法：MAXVEL = value

类型：DOUBLE

描述：设定/读取运动轴的运行速度限值，单位元为 UNIT/s。VL、VH、HOME_VH 等跟轴速度相关的设定值都不能超过该限值，否则设定会不成功。

范围：【1,5000000】，默认值 1000000

例程

```
MAXVEL=200000 '设置轴的最大运行速度限值为 200000UNIT/s
```

2.5.17 MAXACC

所属：属性

语法：MAXACC = value

类型：DOUBLE

描述：设定/读取运动轴的加速度限值，单位为 UNIT/s^2。ACC、HOME_ACC 等跟轴加速度相关的设定值都不能超过该限值，否则设定会不成功。

范围：【1,500000000】，默认值 500000000

例程

```
MAXACC=200000 '设置轴的加速度限值为 200000UNIT/s^2
```

2.5.18 MAXDEC

所属：属性

语法：MAXDEC = value

类型：DOUBLE

描述：设定/读取运动轴的减速度限值，单位元为 UNIT/s^2 。DEC、HOME_DEC 等跟轴减速度相关的设定值都不能超过该限值，否则设定会不成功。

范围：【1,500000000】，默认值 500000000

例程

```
MAXDEC=200000 '设置轴的减速度限值为 200000UNIT/s^2
```

2.5.19 RESETERR

所属：命令

语法 1：RESETERR

语法 2：RESETERR AX(axis no)

描述：BASE 轴清单的轴或指定轴，清除轴错误

参数：axis no 轴号；**范围：**根据控制器实际硬件决定。

例程

```
BASE 0,1,2
RESETERR   '清除轴 0、1、2 的错误
RESETERR AX(1)  '清除轴 1 的错误
```

2.6 单轴定速运动

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.6.1	FORWARD	正向恒速连续运动	√	×
2.6.2	REVERSE	反向恒速连续运动	√	×
2.6.3	VCHANGE	运动中改变速度	√	×
2.6.4	VCHANGE_RATE	运动中改变速度（百分比）	√	×

2.6.1 FORWARD

所属：命令

语法 1：FORWARD

语法 2：FORWARD AX(axis no)

语法 3：FORWARD dir1[，dir2][，dir3].....dir 为 0 时，方向与 FORWARD 同向 dir 为 1 时，方向与 FORWARD 反向

描述：BASE 轴列表的轴或指定轴，开始正向连续运动

参数：dir 方向；**类型：**ULONG

axis no 轴号；**范围：**根据控制器实际硬件决定。

例程

```
BASE 0,1
SVON
VL=1000          '设置初速度
VH=10000         '设置运行速度
ACC=100000       '设置加速度
DEC=ACC          '设置减速度
'单轴或多轴同方向执行连续运动
FORWARD    '轴 0,1 都执行正向连续运动
SLEEP 2000   '延时 2000ms
STOPDEC    '减速停止轴 0,1 运动
WAIT DONE   '等待运动停止,如运动未停止的状态，下面语句再对该轴操作会执行不成功
REVERSE    '轴 0,1 都执行负向连续运动
SLEEP 2000
STOPDEC    '减速停止轴 0,1 运动
WAIT DONE   '等待运动停止
'指定多个轴按不同方向执行连续运动
FORWARD 0,1   '轴 0 正向连续运动，轴 1 负向连续运动
SLEEP 2000
STOPEMG    '立即停止轴 0,1 运动
WAIT DONE
'指定一个轴执行连续运动
FORWARD AX(0)    '轴 0 执行正向连续运动
SLEEP 1000
STOPDEC AX(0)    '指定轴 0 下减速停止命令
WAIT AX(0),DONE   '指定轴等待运动停止
```

2.6.2 REVERSE

所属：命令

语法 1：REVERSE

语法 2：REVERSEAX(axis no)

语法 3：REVERSEdir1[，dir2][，dir3]……dir 为 0 时，方向与 REVERSE 同向 dir 为 1 时，方向与 REVERSE 反向

描述：BASE 轴列表的轴或指定轴，开始反向连续运动

参数：dir　方向；**类型：**ULONG

axis no 轴号；**范围：**根据控制器实际硬件决定。

例程

```
BASE 0,1
SVON
VL=1000            '设置初速度
VH=10000           '设置运行速度
ACC=100000         '设置加速度
DEC=ACC            '设置减速度
'单轴或多轴同方向执行连续运动
FORWARD    '轴 0,1 都执行正向连续运动
SLEEP 2000    '延时 2000ms
STOPDEC    '减速停止轴 0,1 运动
WAIT DONE    '等待运动停止,如运动未停止的状态，下面语句再对该轴操作会执行不成功
REVERSE    '轴 0,1 都执行负向连续运动
SLEEP 2000
STOPDEC    '减速停止轴 0,1 运动
WAIT DONE    '等待运动停止
'指定多个轴按不同方向执行连续运动
REVERSE 0,1    '轴 0 负向连续运动，轴 1 正向连续运动
SLEEP 2000
STOPEMG    '立即停止轴 0,1 运动
WAIT DONE
'指定一个轴执行连续运动
REVERSE AX(0)    '轴 0 执行负向连续运动
SLEEP 1000
STOPDEC AX(0)    '指定轴 0 下减速停止命令
WAIT AX(0),DONE    '指定轴等待运动停止
```

2.6.3 VCHANGE

所属：命令

语法 1：VCHANGE vel

语法 2：VCHANGE AX（axis no），vel

语法 3：VCHANGE vel, acc, dec

语法 4：VCHANGE AX（axis no）, vel，acc, dec

描述：BASE 轴列表的第一个轴，开始更改速度运动；或指定轴和新的速度开始更改速度运动。该指令可以对 MOVE、MOVEABS、FORWARD、REVERSE 起作用，如果当前轴不在运动中，下该指令会报错

参数：vel　　运行速度；**类型：**DOUBLE

acc　　改变速度时的加速度；**类型：**DOUBLE

dec　　改变速度时的减速度；**类型：**DOUBLE

axis no　　轴号；**范围：**根据控制器实际硬件决定

例程

```
BASE 0,1
SVON
VL=1000              '设置初速度
VH=10000             '设置运行速度
ACC=100000           '设置加速度
DEC=ACC              '设置减速度
FORWARD    '轴 0,1 都执行正向连续运动
SLEEP 2000   '延时 2000ms
VCHANGE 30000   '对 BASE 列表中第一个轴起作用，将轴 0 的速度改为 30000
SLEEP 2000
VCHANGE AX(1),5000  '将轴 1 的速度改为 50000
SLEEP 2000
VCHANGE 20000,10000,10000  '将轴 0 的速度改为 20000，加、减速度都改为 10000
SLEEP 3000
VCHANGE AX(1),20000,50000,50000 '将轴 1 的速度改为 20000，加、减速度都改为 50000
SLEEP 2000
STOPDEC
```

2.6.4 VCHANGE_RATE

所属：命令

语法 1：VCHANGE_RATE　rate

语法 2：VCHANGE_RATE　AX（axis no），rate

语法 3：VCHANGE_RATE　rate, acc, dec

语法 4：VCHANGE_RATE　AX（axis no）, rate，acc, dec

描述 :BASE 轴列表的第一个轴 ,开始按百分比更改速度运动 ;或指定轴和新的百分比速度开始更改速度运动。该指令可以对 MOVE、MOVEABS、FORWARD、REVERSE 起作用，如果当前轴不在运动中，下该指令会报错

参数：rate　　原设置速度的百分比速度；**类型：**DOUBLE

acc　　改变速度时的加速度；**类型：**DOUBLE

dec　　改变速度时的减速度；**类型：**DOUBLE

axis no　轴号；**范围：**根据控制器实际硬件决定

例程

```
BASE 0,1
SVON
VL=1000            '设置初速度
VH=10000           '设置运行速度
ACC=100000         '设置加速度
DEC=ACC            '设置减速度
FORWARD           '轴 0,1 都执行正向连续运动
SLEEP 2000           '延时 2000ms
'运动中改变速度的功能应用中：新设定的速度要高于 VL
VCHANGE_RATE 200      '对 BASE 列表中第一个轴起作用，将轴 0 的速度改为设定的 VH 的 200%
SLEEP 2000
VCHANGE_RATE AX(1),30  '将轴 1 的速度改为设定的 VH 的 30%
SLEEP 2000
VCHANGE_RATE 50,10000,10000  '将轴 0 的速度改为 VH 的 50%，加、减速度都改为 10000
SLEEP 3000
VCHANGE_RATE AX(1),400,50000,50000 '将轴 1 的速度改为 VH 的 400%，加、减速度都改为 50000
SLEEP 2000
STOPDEC
```

2.7 多轴插补运动

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.7.1	GVL	插补初速度	×	×
2.7.2	GVH	插补运行速度	×	×
2.7.3	GACC	插补加速度	×	×
2.7.4	GDEC	插补减速度	×	×
2.7.5	GJK	插补速度曲线类型	×	×
2.7.6	GDSPEED	当前插补运动指令速度值	×	×
2.7.7	GSTATE	当前插补运动状态	×	×
2.7.8	LINE	2-3 轴直线相对插补运动	×	×
2.7.9	LINEABS	2-3 轴直线绝对插补运动	×	×
2.7.10	DIRECT	2-8 轴线性相对插补运动	×	×
2.7.11	DIRECTABS	2-8 轴线性绝对插补运动	×	×
2.7.12	CIRC	2 轴相对圆弧插补（指定圆心、终点）	×	×
2.7.13	CIRCABS	2 轴绝对圆弧插补（指定圆心、终点）	×	×
2.7.14	CIRC_3P	2 轴相对圆弧插补（指定圆上 3 点）	×	×
2.7.15	CIRCABS_3P	2 轴绝对圆弧插补（指定圆上 3 点）	×	×
2.7.16	CIRC_A	2 轴相对圆弧插补（指定圆弧角度、终点）	×	×
2.7.17	CIRCABS_A	2 轴绝对圆弧插补（指定圆弧角度、终点）	×	×
2.7.18	HELIX	3 轴相对螺旋插补（指定圆弧中心、终点、高度）	×	×
2.7.19	HELIXABS	3 轴绝对螺旋插补（指定圆弧中心、终点、高度）	×	×
2.7.20	HELIX_3P	3 轴相对螺旋插补（指定螺旋在线 3 点）	×	×

2.7.21	HELIXABS_3P	3 轴绝对螺旋插补（指定螺旋在线 3 点）	×	×
2.7.22	HELIX_A	3 轴相对螺旋插补（指定圆弧角度、终点、高度）	×	×
2.7.23	HELIXABS_A	3 轴绝对螺旋插补（指定圆弧角度、终点、高度）	×	×
2.7.24	GPAUSE	插补运动暂停指令	×	×
2.7.25	GRESUME	插补运动暂定后恢复运动指令	×	×

2.7.1 GVL

所属：属性

语法：GVL = value

类型：DOUBLE

描述：设置/读取插补运动的初速度，GVL 的单位为 UNIT/s

范围：（0,MAXVEL），默认值 2000

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1
GVL=2000 '设置轴 0,1 的插补运动的初速度为 2000 个 UNIT/s
```

2.7.2 GVH

所属：属性

语法：GVH = value

类型：DOUBLE

描述：设置/读取插补运动的最大运行速度，GVH 的单位为 UNIT/s

范围：（GVL,MAXVEL），默认值 8000

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1
GVH=10000 '设置轴 0,1 的插补运动的最大运行速度为 10000 个 UNIT/s
```

2.7.3 GACC

所属：属性

语法：GACC = value

类型：DOUBLE

描述：设置/读取插补运动的加速度，GACC 的单位为 UNIT/s^2

范围：（0,MAXACC），默认值 10000

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1
GACC=20000 '设置轴 0,1 的插补运动的加速度为 20000 个 UNIT/s^2
```

2.7.4 GDEC

所属：属性

语法：GDEC = value

类型：DOUBLE

描述：设置/读取插补运动的减速度，GDEC 的单位为 UNIT/s^2

范围：（0,MAXDEC），默认值 10000

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1
GDEC=20000 '设置轴 0,1 的插补运动的减速度为 20000 个 UNIT/s^2
```

2.7.5 GJK

所属：属性

语法：GJK = value

类型：ULONG

描述：设置/读取插补运动的速度曲线类型

范围：【0,1】，0：T 型曲线；1：S 型曲线，默认值 0

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1
GJK=1  '设置轴 0、1 的插补运动的速度曲线类型为 S 型曲线
```

2.7.6 GDSPEED

所属：属性（只读）

语法：value = GDSPEED

类型：DOUBLE

描述：读取当前插补指令理论速度

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1
DIM A AS DOUBLE
A=GDSPEED   '将轴 0,1 插补运动的当前指令速度赋值给变量 A
```

2.7.7 GSTATE

所属：属性（只读）

语法：value = GSTATE

类型：ULONG

描述：读取当前插补运动状态。

返回值：如下

0：插补不可用状态

1：插补运动处于 Ready 状态

2：插补运动处于停止状态，但未 Ready

3：插补运动处于错误状态，插补运动被停止运动

4：BASE 轴正在执行插补运动中

5：保留

6：BASE 轴正在执行连续插补运动中

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1
DIM A AS USHORT
A=GSTATE   '将轴 0、1 当前的插补运动状态对应的值赋值给变量 A
```

2.7.8 LINE

所属：命令

语法：LINE　distance1,distance2 [,distance3]

描述：指定插补轴的移动距离，开始 2 轴或 3 轴的相对直线插补运动。LINE 指令仅支持 2 轴或 3 轴的直线插补运动，3 轴以上不支持。

参数：distance　各轴的相对移动距离；**类型：**DOUBLE

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1
SVON
GVL=1000            '设置插补初速度
GVH=10000           '设置插补运行速度
GACC=100000         '设置插补加速度
GDEC=GACC           '设置插补减速度
GJK=0                '设置插补速度曲线为 T 型
 '绝对直线插补运动
LINEABS 0,5000  '运动到目标位置（0,5000）
WAIT DONE   '等待 LINEABS 运动走完
 '相对直线插补运动
LINE 8000,-15000   '轴 0、1 方向的运动距离分别为 8000、-15000
WAIT DONE
 '用数组填写位置进行直线插补
DIM EndPos(1) As Double={1000,2000}
LINEABS EndPOS()      '运动到目标位置（1000,2000）
WAIT DONE
'使用 P 点进行运动
Dim P0 As Pt          '实例化出一个位置点对象 P0
P0=Pt(1000,2000)      '位置点赋值为(1000,2000)
BASE 0,1
LINE P0               '轴 0、轴 1 进行相对直线插补的距离分别为 1000，2000
WAIT DONE
```

2.7.9 LINEABS

所属：命令

语法：LINEABS　position1,position2[,position3]

描述：指定插补轴的终点，开始 2 轴或 3 轴的绝对直线插补运动。LINEABS 指令仅支持 2 轴或 3 轴的直线插补运动，3 轴以上不支持。

参数：position 各轴的终点位置；**类型：**DOUBLE

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1
SVON
GVL=1000            '设置插补初速度
GVH=10000           '设置插补运行速度
GACC=100000         '设置插补加速度
GDEC=GACC           '设置插补减速度
GJK=0                '设置插补速度曲线为 T 型
 '绝对直线插补运动
LINEABS 0,5000  '运动到目标位置（0,5000）
WAIT DONE   '等待 LINEABS 运动走完
 '相对直线插补运动
LINE 8000,-15000   '轴 0、1 方向的运动距离分别为 8000、-15000
WAIT DONE
 '用数组填写位置进行直线插补
DIM EndPos(1) As Double={1000,2000}
LINEABS EndPOS()      '运动到目标位置（1000,2000）
WAIT DONE
'使用 P 点进行运动
Dim P0 As Pt              '实例化出一个位置点对象 P0
P0=Pt(1000,2000,3000)     '位置点赋值为(1000,2000,3000)
BASE 0,1,2
LINEABS P0                '轴 0、1、2 绝对直线插补到位置点(1000,2000,3000)
WAIT DONE
```

2.7.10 DIRECT

所属：命令

语法：DIRECT distance1,distance2[,distance3]... [,distance8]

描述：指定插补轴的移动距离，开始 2 轴-8 轴的相对线性插补运动。最多支援到 8 个轴的 DIRECT 线性插补运动

参数：distance 各轴的相对移动距离；**类型：**DOUBLE

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1,2,3
SVON
'DIRECT 插补运动中，以下速度、加减速参数设置的是参与插补运动中移动距离最长轴的参数
GVL=1000            '设置插补初速度
GVH=10000           '设置插补运行速度
GACC=100000         '设置插补加速度
GDEC=GACC           '设置插补减速度
GJK=0               '设置插补速度曲线为 T 型
'绝对线性插补
DIRECTABS 0,5000,-500,1000   '运动到目标位置（0,5000,-500,1000）
WAIT DONE              '等待 DIRECTABS 运动走完
'相对线性插补
DIRECT 8000,-15000,0,2000    '轴 0、1、2、3 方向的运动距离分别为 8000、-15000、0、2000
WAIT DONE
'用数组填写位置进行线性插补
DIM EndPos(3) As Double={1000,2000,3000,4000}
DIRECTABS EndPOS()        '运动到目标位置（1000,2000,3000,4000）
WAIT DONE
'使用 P 点进行运动
Dim P0 As Pt            '实例化出一个位置点对象 P0
P0=Pt(1000,2000)        '位置点赋值为(1000,2000)
BASE 0,1
DIRECT P0               '轴 0、轴 1 进行相对线性插补的距离分别为 1000，2000
WAIT DONE
```

2.7.11 DIRECTABS

所属：命令

语法：DIRECTABS　position1,position2[,position3]... [,position8]

描述：指定插补轴的终点，开始 2 轴-8 轴的绝对线性插补运动。最多支援到 8 个轴的 DIRECTABS 线性插补运动

参数：position　各轴的终点位置；**类型：**DOUBLE

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1,2,3
SVON
'DIRECT 插补运动中，以下速度、加减速参数设置的是参与插补运动中移动距离最长轴的参数
GVL=1000            '设置插补初速度
GVH=10000           '设置插补运行速度
GACC=100000         '设置插补加速度
GDEC=GACC           '设置插补减速度
GJK=0               '设置插补速度曲线为 T 型
'绝对线性插补
DIRECTABS 0,5000,-500,1000   '运动到目标位置（0,5000,-500,1000）
WAIT DONE                '等待 DIRECTABS 运动走完
'相对线性插补
DIRECT 8000,-15000,0,2000    '轴 0、1、2、3 方向的运动距离分别为 8000、-15000、0、2000
WAIT DONE
'用数组填写位置进行线性插补
DIM EndPos(3) As Double={1000,2000,3000,4000}
DIRECTABS EndPOS()          '运动到目标位置（1000,2000,3000,4000）
WAIT DONE
'使用 P 点进行运动
Dim P0 As Pt                '实例化出一个位置点对象 P0
P0=Pt(1000,2000,3000)       '位置点赋值为(1000,2000,3000)
BASE 0,1,2
DIRECTABS P0                '轴 0、1、2 绝对线性插补到位置点(1000,2000,3000)
WAIT DONE
```

2.7.12 CIRC

所属：命令

语法：CIRC dir,center1,center2,end1,end2

描述：指定圆方向、圆心、终点，开始两轴的相对圆弧插补运动

参数：dir 圆弧运动方向：0-顺时针；1-逆时针；**类型：**ULONG

center1 第一个轴圆心的相对坐标；**类型：**DOUBLE

center2 第二个轴圆心的相对坐标；**类型：**DOUBLE

end1 第一个轴圆弧终点的相对坐标；**类型：**DOUBLE

end2 第二个轴圆弧终点的相对坐标；**类型：**DOUBLE

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1
SVON
'执行相对圆弧插补运动
CIRC 0,5000,0,10000,0    '顺时针，圆心相对距离为（5000,0），圆弧终点相对距离为（10000,0）
WAIT DONE

'执行绝对圆弧插补运动
DPOS=0                   '当前理论位置清零
MPOS=0                   '当前实际位置清零
CIRCABS 1,5000,0,0,0     '逆时针，圆心位置为（5000,0），圆弧终点位置为（0,0）
WAIT DONE

'用数组填写位置进行圆弧插补
DIM CenPos(1) As Double={5000,0}
DIM EndPos(1) As Double={10000,0}
CIRC 0,CenPos(),EndPos()  '顺时针，圆心相对距离为（5000,0），圆弧终点相对距离为（10000,0）
WAIT DONE

'使用 P 点进行运动
Dim As Pt P_Cen,P_End     '实例化出 2 个位置点对象，名分别为 P_Cen,P_End
P_Cen=Pt(5000,0)          '位置点 P_Cen 赋值为(5000,0)
P_End=Pt(10000,0)         '位置点 P_End 赋值为(10000,0)
BASE 0,1
CIRC 0,P_Cen,P_End    '顺时针，圆心相对距离为（5000,0），圆弧终点相对距离为（10000,0）
WAIT DONE
```

2.7.13 CIRCABS

所属：命令

语法：CIRCABSdir,center1,center2,end1,end2

描述：指定圆方向、圆心、终点，开始两轴的绝对圆弧插补运动

参数：dir 圆弧运动方向：0-顺时针；1-逆时针；**类型：**ULONG

center1 第一个轴圆心的绝对坐标；**类型：**DOUBLE

center2 第二个轴圆心的绝对坐标；**类型：**DOUBLE

end1 第一个轴圆弧终点的绝对坐标；**类型：**DOUBLE

end2 第二个轴圆弧终点的绝对坐标；**类型：**DOUBLE

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1
SVON
'执行相对圆弧插补运动
CIRC 0,5000,0,10000,0    '顺时针，圆心相对距离为（5000,0），圆弧终点相对距离为（10000,0）
WAIT DONE

'执行绝对圆弧插补运动
DPOS=0                   '当前理论位置清零
MPOS=0                   '当前实际位置清零
CIRCABS 1,5000,0,0,0     '逆时针，圆心位置为（5000,0），圆弧终点位置为（0,0）
WAIT DONE

'用数组填写位置进行圆弧插补
DIM CenPos(1) As Double={5000,0}
DIM EndPos(1) As Double={10000,0}
CIRC 0,CenPos(),EndPos()  '顺时针，圆心相对距离为（5000,0），圆弧终点相对距离为（10000,0）
WAIT DONE

'使用 P 点进行运动
Dim As Pt P_Cen,P_End    '实例化出 2 个位置点对象，名分别为 P_Cen,P_End
P_Cen=Pt(5000,0)         '位置点 P_Cen 赋值为(5000,0)
P_End=Pt(0,0)            '位置点 P_End 赋值为(0,0)
BASE 0,1
DPOS=0                   '当前理论位置清零
MPOS=0                   '当前实际位置清零
CIRCABS 1,P_Cen,P_End    '逆时针，圆心位置为（5000,0），圆弧终点位置为（0,0）
WAIT DONE
```

2.7.14 CIRC_3P

所属：命令

语法：CIRC_3Pdir,ref1,ref2,end1,end2

描述：指定圆方向和圆上 3 点，开始两轴的相对圆弧插补运动

参数：dir 圆弧运动方向：0-顺时针；1-逆时针；**类型：**ULONG

ref1 第一个轴中间点的相对坐标；**类型：**DOUBLE

ref2 第二个轴中间点的相对坐标；**类型：**DOUBLE

end1 第一个轴圆弧终点的相对坐标；**类型：**DOUBLE

end2 第二个轴圆弧终点的相对坐标；**类型：**DOUBLE

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1
SVON
'执行 3 点相对圆弧插补运动
CIRC_3P 0,10000,10000,20000,0
WAIT DONE

'执行 3 点绝对圆弧插补运动
DPOS=0                          '当前理论位置清零
MPOS=0                          '当前实际位置清零
CIRCABS_3P 1,10000,10000,20000,0
WAIT DONE

'用数组填写位置进行 3 点圆弧插补
DIM RefPos(1) As Double={10000,10000}
DIM EndPos(1) As Double={20000,0}
CIRC_3P 0,RefPos(),EndPos()
WAIT DONE

'使用 P 点进行运动
Dim As Pt P_Ref,P_End     '实例化出 2 个位置点对象，名分别为 P_Ref,P_End
P_Ref=Pt(10000,10000)     '位置点 P_Ref 赋值为(10000,10000)
P_End=Pt(20000,0)         '位置点 P_End 赋值为(20000,0)
BASE 0,1
CIRC_3P 0,P_Ref,P_End     '顺时针执行 3 点相对圆弧插补运动
WAIT DONE
```

2.7.15 CIRCABS_3P

所属：命令

语法：CIRCABS_3P dir,ref1,ref2,end1,end2

描述：指定圆方向和圆上 3 点，开始两轴的绝对圆弧插补运动

参数：dir 圆弧运动方向：0-顺时针；1-逆时针；**类型：**ULONG

ref1 第一个轴中间点的绝对坐标；**类型：**DOUBLE

ref2 第二个轴中间点的绝对坐标；**类型：**DOUBLE

end1 第一个轴圆弧终点的绝对坐标；**类型：**DOUBLE

end2 第二个轴圆弧终点的绝对坐标；**类型：**DOUBLE

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1
SVON
'执行 3 点相对圆弧插补运动
CIRC_3P 0,10000,10000,20000,0
WAIT DONE

'执行 3 点绝对圆弧插补运动
DPOS=0                          '当前理论位置清零
MPOS=0                          '当前实际位置清零
CIRCABS_3P 1,10000,10000,20000,0
WAIT DONE

'用数组填写位置进行 3 点圆弧插补
DIM RefPos(1) As Double={10000,10000}
DIM EndPos(1) As Double={20000,0}
CIRC_3P 0,RefPos(),EndPos()
WAIT DONE

'使用 P 点进行运动
Dim As Pt P_Ref,P_End     '实例化出 2 个位置点对象，名分别为 P_Ref,P_End
P_Ref=Pt(10000,10000)     '位置点 P_Ref 赋值为(10000,10000)
P_End=Pt(20000,0)         '位置点 P_End 赋值为(20000,0)
BASE 0,1
CIRCABS_3P 0,P_Ref,P_End '顺时针执行 3 点绝对圆弧插补运动
WAIT DONE
```

2.7.16 CIRC_A

所属：命令

语法：CIRC_A dir,center1,center2,degree

描述：指定圆方向、圆心、角度，开始两轴的相对圆弧插补运动

参数：dir 圆弧运动方向：0-顺时针；1-逆时针；**类型：**ULONG

center1 第一个轴圆心的相对坐标；**类型：**DOUBLE

center2 第二个轴圆心的相对坐标；**类型：**DOUBLE

degree 圆弧角度，单位为角度；**类型：**DOUBLE

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1
SVON
'执行相对圆弧插补运动
CIRC_A 0,10000,0,180    '顺时针，圆心相对距离为（10000,0）,圆弧角度为 180 度
WAIT DONE

'执行绝对圆弧插补运动
CIRCABS_A 1,10000,0,90  '逆时针，圆心位置为（10000,0）,圆弧角度为 90 度
WAIT DONE

'用数组填写位置进行圆弧插补
DIM CenPos(1) As Double={10000,0}
DIM angle As Double =190.0
CIRC_A 0,CenPos(),angle
WAIT DONE
```

2.7.17 CIRCABS_A

所属：命令

语法：CIRCABS_A dir,center1,center2,degree

描述：指定圆方向、圆心、角度，开始两轴的绝对圆弧插补运动

参数：dir 圆弧运动方向：0-顺时针；1-逆时针；**类型：**ULONG

center1 第一个轴圆心的绝对坐标；**类型：**DOUBLE

center2 第二个轴圆心的绝对坐标；**类型：**DOUBLE

degree 圆弧角度，单位为角度；**类型：**DOUBLE

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1
SVON
'执行相对圆弧插补运动
CIRC_A 0,10000,0,180    '顺时针，圆心相对距离为（10000,0）,圆弧角度为 180 度
WAIT DONE

'执行绝对圆弧插补运动
CIRCABS_A 1,10000,0,90  '逆时针，圆心位置为（10000,0）,圆弧角度为 90 度
WAIT DONE

'用数组填写位置进行圆弧插补
DIM CenPos(1) As Double={10000,0}
DIM angle As Double =190.0
CIRC_A 0,CenPos(),angle
WAIT DONE
```

2.7.18 HELIX

所属：命令

语法：HELIXdir,center1,center2,end1,end2,distance

描述：指定螺旋旋转方向、中心、螺旋高度，开始 3 轴的相对螺旋插补运动

参数：dir 螺旋运动方向：0：顺时针；1：逆时针

center1 第一个轴圆心的相对坐标；

center2 第二个轴圆心的相对坐标；

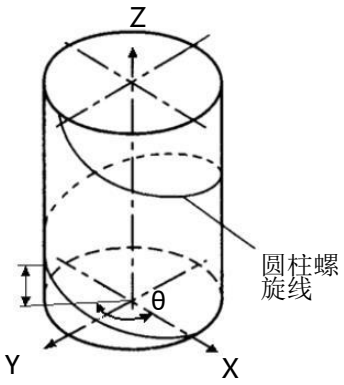
end1 第一个轴螺旋终点的相对坐标；

end2 第二个轴螺旋终点的相对坐标；

distance 螺旋高度，单位为 UNIT

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程




```
BASE 0,1,2
GVL=1000          '设置插补初速度
GVH=10000         '设置插补运行速度
GACC=100000       '设置插补加速度
GDEC=GACC         '设置插补减速度
GJK=0             '设置插补速度曲线为 T 型
DPOS=0           '当前理论位置清零
MPOS=0            '当前回馈位置清零
SVON
'相对螺旋插补，顺时针，圆心相对坐标为（5000,0），圆弧终点相对坐标为（10000,0），螺旋高度为 5000
HELIX 0,5000,0,10000,0,5000
WAIT DONE
'绝对螺旋插补，逆时针，圆心绝对坐标为（5000,0），圆弧终点绝对坐标为（0,0），螺旋 Z 轴终点位置为 0
HELIXABS 1,5000,0,0,0,0
WAIT DONE
```

2.7.19 HELIXABS

所属：命令

语法：HELIXABSdir,center1,center2,end1,end2,position

描述：指定螺旋方向、中心、终点，开始 3 轴的绝对螺旋插补运动

参数：dir 螺旋运动方向：0：顺时针；1：逆时针

center1 第一个轴圆心的绝对坐标

center2 第二个轴圆心的绝对坐标

end1 第一个轴螺旋终点的绝对坐标

end2 第二个轴螺旋终点的绝对坐标

position 第三个轴终点位置坐标

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1,2
GVL=1000          '设置插补初速度
GVH=10000         '设置插补运行速度
GACC=100000       '设置插补加速度
GDEC=GACC         '设置插补减速度
GJK=0             '设置插补速度曲线为 T 型
DPOS=0            '当前理论位置清零
MPOS=0            '当前回馈位置清零
SVON
'相对螺旋插补，顺时针，圆心相对坐标为（5000,0），圆弧终点相对坐标为（10000,0），螺旋高度为 5000
HELIX 0,5000,0,10000,0,5000
WAIT DONE
'绝对螺旋插补，逆时针，圆心绝对坐标为（5000,0），圆弧终点位置为（0,0），螺旋 Z 轴终点位置为 0
HELIXABS 1,5000,0,0,0,0
WAIT DONE
```

2.7.20 HELIX_3P

所属：命令

语法：HELIX_3P dir,ref1,ref2,ref3,end1,end2,end3

描述：指定螺旋方向和螺旋在线的 3 点，开始 3 轴的相对螺旋插补运动

参数： dir 螺旋运动方向：0：顺时针；1：逆时针

ref1 第一个轴中间点的相对坐标；

ref2 第二个轴中间点的相对坐标；

ref3 第三个轴中间点的相对坐标；

end1 第一个轴螺旋终点的相对坐标；

end2 第二个轴螺旋终点的相对坐标；

end3 第三个轴螺旋终点的相对坐标；

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1,2
GVL=1000          '设置插补初速度
GVH=10000         '设置插补运行速度
GACC=100000       '设置插补加速度
GDEC=GACC         '设置插补减速度
GJK=0             '设置插补速度曲线为 T 型
DPOS=0            '当前理论位置清零
MPOS=0            '当前回馈位置清零
SVON
'相对螺旋插补。顺时针，中间参考点相对坐标为（0,5000,0），终点相对坐标为（10000,0,5000）
HELIX_3P0,0,5000,0,10000,0,5000
WAIT DONE
'绝对螺旋插补。逆时针，中间参考点绝对坐标为（0,5000,2500），终点绝对坐标为（0,0,0）
HELIXABS_3P 1,0,5000,2500,0,0,0
WAIT DONE
```

2.7.21 HELIXABS_3P

所属：命令

语法：HELIXABS_3P dir,ref1,ref2,ref3,end1,end2,end3

描述：指定螺旋方向和螺旋在线的 3 点，开始 3 轴的绝对螺旋插补运动

参数： dir 螺旋运动方向：0：顺时针；1：逆时针

ref1 第一个轴中间点的绝对坐标；

ref2 第二个轴中间点的绝对坐标；

ref3 第三个轴中间点的绝对坐标；

end1 第一个轴螺旋终点的绝对坐标；

end2 第二个轴螺旋终点的绝对坐标；

end3 第三个轴螺旋终点的绝对坐标；

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1,2
GVL=1000          '设置插补初速度
GVH=10000         '设置插补运行速度
GACC=100000       '设置插补加速度
GDEC=GACC         '设置插补减速度
GJK=0             '设置插补速度曲线为 T 型
DPOS=0            '当前理论位置清零
MPOS=0            '当前回馈位置清零
SVON
'相对螺旋插补。顺时针，中间参考点相对坐标为（0,5000,0），终点相对坐标为（10000,0,5000）
HELIX_3P0,0,5000,0,10000,0,5000
WAIT DONE
'绝对螺旋插补。逆时针，中间参考点绝对坐标为（0,5000,2500），终点绝对坐标为（0,0,0）
HELIXABS_3P 1,0,5000,2500,0,0,0
WAIT DONE
```

2.7.22 HELIX_A

所属：命令

语法：HELIX_A dir,center1,center2,degree,distance

描述：指定螺旋方向、螺旋圆面上的旋转角度、螺旋高度，开始 3 轴的相对螺旋插补运动

参数：dir 螺旋运动方向：0：顺时针；1：逆时针

center1 第一个轴圆心的相对坐标

center2 第二个轴圆心的相对坐标

degree 螺旋线形成的圆柱截面上的投影圆弧运动的圆心角度

distance 螺旋高度，单位为 UNIT

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1,2
GVL=1000          '设置插补初速度
GVH=10000         '设置插补运行速度
GACC=100000       '设置插补加速度
GDEC=GACC         '设置插补减速度
GJK=0             '设置插补速度曲线为 T 型
DPOS=0            '当前理论位置清零
MPOS=0            '当前回馈位置清零
SVON
'相对螺旋插补，顺时针，圆心相对坐标为（5000,0），运动的圆心角度为 180 度，，螺旋高度为 5000
HELIX_A 0,5000,0,180,5000
WAIT DONE
'绝对螺旋插补，逆时针，圆心绝对坐标为（5000,0），运动的圆心角度为 180 度，，螺旋 Z 轴终点位置为 0
HELIXABS_A1,5000,0,180,0
WAIT DONE
```

2.7.23 HELIXABS_A

所属：命令

语法：HELIXABS_A dir,center1,center2,degree,position

描述：指定螺旋方向、螺旋圆面上的旋转角度、终点位置，开始 3 轴的绝对螺旋插补运动

参数：dir 螺旋运动方向：0：顺时针；1：逆时针

center1 第一个轴圆心的绝对坐标

center2 第二个轴圆心的绝对坐标

degree 螺旋线形成的圆柱截面上的投影圆弧运动的圆心角度

position 第三个轴终点位置坐标

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1,2
GVL=1000          '设置插补初速度
GVH=10000         '设置插补运行速度
GACC=100000       '设置插补加速度
GDEC=GACC         '设置插补减速度
GJK=0             '设置插补速度曲线为 T 型
DPOS=0            '当前理论位置清零
MPOS=0            '当前回馈位置清零
SVON
'相对螺旋插补，顺时针，圆心相对坐标为（5000,0），运动的圆心角度为 180 度，，螺旋高度为 5000
HELIX_A 0,5000,0,180,5000
WAIT DONE
'绝对螺旋插补，逆时针，圆心绝对坐标为（5000,0），运动的圆心角度为 180 度，，螺旋 Z 轴终点位置为 0
HELIXABS_A1,5000,0,180,0
WAIT DONE
```

2.7.24 GPAUSE

所属：命令

语法：GPAUSE

描述：暂定当前 TASK 的插补运动。该指令对插补运动和连续插补运动都起作用。

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1
LINE 11000,20000
SLEEP 500
GPAUSE          '暂停插补运动
IF(DIN(1)=1) THEN
GRESUME         '恢复插补运动，继续执行未执行的插补运动
END IF
```

2.7.25 GRESUME

所属：命令

语法：GRESUME

描述：恢复当前 TASK 的插补运动暂定状态。该指令对插补运动和连续插补运动都起作用。

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1
LINE 11000,20000
SLEEP 500
GPAUSE          '暂停插补运动
IF(DIN(1)=1) THEN '如果 DIN(1)为 1 时，恢复插补运动
GRESUME         '恢复插补运动，继续执行未执行的插补运动
END IF
```

2.8 多轴连续插补运动

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.8.1	PATHBEGIN	开始进入连续插补状态	×	×
2.8.2	PATHEND	结束连续插补状态路径缓存添加	×	×
2.8.3	MERGEON	用 fly mode 交接模式执行连续插补运动	×	×
2.8.4	MERGEOFF	用 buffer mode 交接模式执行连续插补运动	×	×
2.8.5	DELAY	连续插补路径中的延时段指令	×	×
2.8.6	DOUT	连续插补路径中的数字量输出指令	×	×
2.8.7	PATHRESET	清除连续插补路径缓存中的路径数据	×	×
2.8.8	PATH_Status	读取连续插补运动中相关参数	×	×
2.8.9	GSPDFWD	开启或关闭速度前瞻模式	×	×

2.8.1 PATHBEGIN

所属：命令

语法：PATHBEGIN [num]

描述：指定路径缓存里添加多少段插补指令后，开始进入连续插补模式。Num 值不填或 0 时，会将 PATHBEGIN 与 PATHEND 间所有的段都添加到缓存区里，再开始执行连续插补运动。

参数：num 预先添加到连续插补缓存区段数；**类型：**ULONG；范围【0,10000】

注意：连续插补段不允许存在点位元运动指令的段

该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

```
BASE 0,1
SVON
GVL=1000            '设置插补初速度
GVH=10000           '设置插补运行速度
GACC=100000         '设置插补加速度
GDEC=ACC            '设置插补减速度
LINEABS 0,0     '运动到目标位置（0,0）
WAIT DONE
PATHRESET       '清除路径缓存
'PATHBEGIN 开始到 PATHEND 之前的路径段为连续插补运动
'PATHBEGIN 后面跟的编号不写或写 0 时，表明路径添加完再开始执行连续插补，其他数值代表添加该数值段后开始执行运动
PATHBEGIN     '进入连续插补状态
MERGEON     '连续插补速度交接模式设置为 fly mode
CIRCABS 1,10000,0,10000,-10000   '圆弧插补段
LINEABS 25000,-10000          '直线插补段
DELAY=500                   '延时段，延时 500ms
CIRCABS 1,25000,0,35000,0    '圆弧插补段
CIRCABS 1,25000,0,25000,10000    '圆弧插补段
LINEABS 10000,10000           '直线插补段
CIRCABS 1,10000,0,0,0             '圆弧插补段
PATHEND     '连续插补路径段结束指令，退出连续状态
```

2.8.2 PATHEND

所属：命令

语法：PATHEND

描述：PATHBEGIN 对应的结束指令，PATHBEGIN 和 PATHEND 之间用于路径缓存添加设定。

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

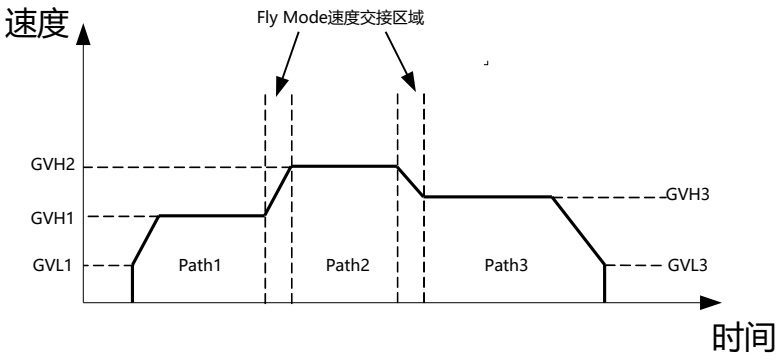
请参考 PATHBEGIN 指令中**例程。**

2.8.3 MERGEON

所属：命令

语法：MERGEON

描述：用 fly mode 速度交接模式执行连续插补运动。flymode 速度交接模式，是前一段路径的 VH 加速或减速到后一段路径的 VH 的速度交接方式。



注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

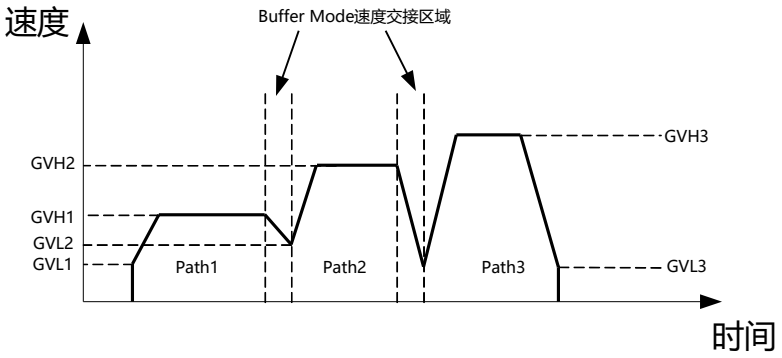
请参考 PATHBEGIN 指令中**例程。**

2.8.4 MERGEOFF

所属：命令

语法：MERGEOFF

描述：用 buffer mode 速度交接模式执行连续插补运动。buffermode 速度点交接模式，是前一段路径的 VH 加速或减速到后一段路径的 VL 的速度交接方式。



注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

请参考 PATHBEGIN 指令中**例程**。

2.8.5 DELAY

所属：命令

语法：DELAY= time

描述：连续插补中，延时段指令。表示 DELAY 指令上一段完成与 DELAY 指令下一段开始间延时的时间，该延时时间非常精准，单位为 ms。

注意：该指令不能在 Motion Studio 中的“终端”和“观察变量”工具中使用

例程

请参考 PATHBEGIN 指令中**例程**。

2.8.6 DOUT

所属：命令

语法：DOUT (no)=value

描述：连续插补运动中，在 PathBegin 和 PathEnd 程序段中添加 DOUT 指令时，表示该数字量输出操作作为一个轨迹段命令，连续插补执行到该段时会做相应的数字量输出操作。

参数：no 数字量输出的编号，在控制器配置时，系统会根据控制器硬件分配对应的编号；**类型：**ULONG

value 数字量输出端口的状态，范围：0 或 1

返回值：输出口的电平，0 或 1

注意：连续插补中使用 DOUT 指令，该 DO 需是 BASE 轴“轴上的 DO”或“设备的 DO”，否则该 DO 操作会不起作用。“轴上的 DO”和“设备的 DO”说明请参照附录章节 3.4“MAS 控制器中的控制单元”。

例程

```
BASE 0,1
SVON
PATHRESET       '清除路径缓存
PATHBEGIN       '进入连续插补状态
LINEABS 25000,-10000      '直线插补段到位置(25000,-10000)
DOUT(0)=1                 '上一段直线插补运动完，将 DOUT(0)置 1
LINEABS 10000,10000       '直线插补段到位置(10000,10000)
DOUT(0)=0                 '上一段直线插补运动完，将 DOUT(0)置 0
PATHEND
```

2.8.7 PATHRESET

所属：命令

语法：PATHRESET

描述：清除连续插补路径缓存中的路径数据。

注意：PATHBEGIN 与 PATHEND 之间的路径段为路径缓存中的路径。如果连续插补执行过程中被停止，下次执行连续插补将从路径缓存中上次剩余的未执行路径开始执行。如果用户不想从剩余未执行的路径段开始执行连续插补，则需先用 PATHRESET 指令清除路径缓存。然后重新添加需执行的路径到缓存中，再执行连续插补。

2.8.8 PATH_Status

所属：命令

语法：PATH_Status RemainPath,FreeBuffer [,curIndex] [,curCmd]

描述：读取连续插补运动中相关参数。

参数：RemainPath 剩余未执行的路径段**类型：**ULONG

FreeBuffer 总路径缓存中的剩余缓存数**类型：**ULONG

curIndex 当前执行的路径索引**类型：**ULONG

curCmd 当前执行的路径指令**类型：**ULONG

curCmd 的枚举如下：

0：EndPath ；

1：Abs2DLine ；

2：Rel2DLine ；

3：Abs2DArcCW ；

4：Abs2DArcCCW ；

5：Rel2DArcCW ；

6：Rel2DArcCCW ；

7：Abs3DLine ；

8：Rel3DLine ；

9：Abs4DLine ；（不支持）

10：Rel4DLine ；（不支持）

11：Abs2DDirect ；

12：Rel2DDirect ；

13：Abs3DDirect ；

14：Rel3DDirect ；

15：Abs4DDirect ；

16：Rel4DDirect ；

17：Abs5DDirect ；

18：Rel5DDirect ；

19：Abs6DDirect ；

20：Rel6DDirect ；

21：Abs3DArcCW ；（不支持）

22：Rel3DArcCW ；（不支持）

23：Abs3DArcCCW ；（不支持）

24：Rel3DArcCCW ；（不支持）

25：Abs3DhelixCW；

26：Rel3DhelixCW；

27：Abs3DHelixCCW；

28：Rel3DHelixCCW；

29：GPDELAY （单位：ms）

90：DOUT

例程

```
DIM AS ULONG RemainPath, FreeBuffer, curIndex,curCmd
BASE 0,1
SVON
GVL=1000              '设置插补初速度
GVH=10000             '设置插补运行速度
GACC=100000          '设置插补加速度
GDEC=ACC              '设置插补减速度
LINEABS 0,0             '运动到目标位置（0,0）
WAIT DONE
PATHRESET             '清除路径缓存
'PATHBEGIN 开始到 PATHEND 之前的路径段为连续插补运动
'PATHBEGIN 后面跟的编号不写或写 0 时，表明路径添加完再开始执行连续插补，其他数值代表添加该数值段后开始执行运动

PATHBEGIN             '进入连续插补状态
MERGEON               '连续插补速度交接模式设置为 fly mode
CIRCABS 1,10000,0,10000,-10000     '圆弧插补段
LINEABS 25000,-10000                  '直线插补段
DELAY=500                                  '延时段，延时 500ms
CIRCABS 1,25000,0,35000,0            '圆弧插补段
CIRCABS 1,25000,0,25000,10000      '圆弧插补段
LINEABS 10000,10000                   '直线插补段
CIRCABS 1,10000,0,0,0                   '圆弧插补段
PATHEND
SLEEP 500
PATH_Status   RemainPath, FreeBuffer, curIndex, curCmd
Print  RemainPath, FreeBuffer, curIndex, curCmd   '打印出 6，9994,1,4
```

2.8.9 GSPDFWD

所属：属性

语法：GSPDFWD = value

类型：ULONG

描述：开启或关闭速度前瞻模式。速度前瞻模式是 Flymode 速度交接模式下的一种(参考 MERGEON 指令，需在 MERGEON 状态下开启速度前瞻模式才起作用)。速度前瞻模式开启后，会将连续插补的路径段距离整体提前规划，整个路径段会使用一个运行速度运行，使路径段间的速度交接更平滑，运行速度使用 GVH 设置的速度。

范围:设定值和返回值如下，默认值 0

0：不开启速度前瞻(默认值)

1：开启速度前瞻

注意：GSPDFWD=1 为开启速度前瞻模式，但是只有在 MERGEON 的模式下开启，速度前瞻模式才启作用。

例程

```
BASE 0,1
SVON
GVL=1000            '设置连续插补初速度
GVH=10000           '设置连续插补运行速度
GACC=100000         '设置连续插补加速度
GDEC=ACC            '设置连续插补减速度
PATHRESET           '清除路径缓存
PATHBEGIN
    MERGEON          '连续插补速度交接模式设置为 fly mode
    GSPDFWD=1       'flymode 模式下开启速度前瞻
    LINE 10000,10000
    LINE 10000,10000
    LINE 10000,10000
    LINE 10000,10000
    LINE 10000,10000
    LINE 10000,10000
PATHEND
WAIT DONE
```

2.9 同步运动

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.9.1	STA_SRC	同步启/停触发源	√	√
2.9.2	SETSTA	以单轴相对运动模式状态等待同步启动信号	√	×
2.9.3	SETSTA_ABS	以单轴绝对运动模式状态等待同步启动信号	√	×
2.9.4	SETSTA_VEL	以恒速连续运动模式状态等待同步启动信号	√	×
2.9.5	STARTSTA	STA 或同张板卡同步轴开始启动运动	√	×
2.9.6	STOPSTA	STA 或同张板卡同步轴开始停止运动	√	×
2.9.7	GANTRY	龙门运动	√	×
2.9.8	GEAR	电子齿轮运动	√	×
2.9.9	PHASE	电子齿轮运动过程中超前或落后相位运动	√	×
2.9.10	TANGENT	切向跟随运动	√	×
2.9.11	MODULO	切向跟随运动中旋转刀轴旋转一周的指令脉冲数	√	√
2.9.12	CAMTABLE	设定凸轮表数据	×	×
2.9.13	CAMSET	设定凸轮表相关配置	×	×
2.9.14	CAMIN	启动电子凸轮，建立主从关系	×	×
2.9.15	CAMSTOP	解除主从凸轮关系	×	×

2.9.1 STA_SRC

所属：属性

语法：STA_SRC=value

类型：ULONG

描述：设置/读取轴的同步启停运动的触发源，触发源被触发后，处于等待同步启/停的轴会根据等待的模式，开始执行相应的运动。STA 信号可以由运动控制卡上硬件 STA 针脚触发产生，也可以由 STARTSTA 指令触发产生。

范围：设定值和返回值如下，默认值 1

0：禁用

1：板卡 STA 信号

2：轴 0 的比较信号

3：轴 1 的比较信号

4：轴 2 的比较信号

5：轴 3 的比较信号

6：轴 4 的比较信号

7：轴 5 的比较信号

8：轴 6 的比较信号

9：轴 7 的比较信号

10：轴 0 的停止信号

11：轴 1 的停止信号

12：轴 2 的停止信号

13：轴 3 的停止信号

14：轴 4 的停止信号

15：轴 5 的停止信号

16：轴 6 的停止信号

17：轴 7 的停止信号

例程

```
BASE 1
STA_SRC =1 '设置轴 1 的同步启停运动触发源为板卡 STA 信号
```

2.9.2 SETSTA

所属：命令

语法：SETSTAdistance 1[,distance 2] [,distance 3]……

描述：设置同步轴为点位元运动模式并处于等待触发状态，同时设置同步轴的相对移动距离。

参数：distance　相对移动距离；**类型：**DOUBLE

例程

```
BASE 0,1,2,3
STA_SRC=1  '设置同步源为板卡 STA 信号
SETSTA 10000,5000,30000,-8000 '设置轴 0,1,2,3 处于等待同步相对运动状态，并设置运动距离
STARTSTA     '同步启，同时启动 4 轴运动
WAIT DONE
SETSTA_ABS 0,2000,5000,20000  '设置轴 0,1,2,3 处于等待同步绝对运动状态，并设置目标位置
SLEEP 500
STARTSTA       '同步启，同时启动 4 轴运动
WAIT DONE
SETSTA_VEL 0,1,0,1 '设置轴 0,1,2,3 处于等待同步连续运动状态，并设置各轴方向
STARTSTA       '同步启，同时启动 4 轴运动
Sleep 3000
STOPSTA        '同步停，同时停止 4 轴运动
```

2.9.3 SETSTA_ABS

所属：命令

语法：SETSTA_ABSposition1[,position2] [,position3]……

描述：设置同步轴为点位元运动模式并处于等待触发状态，同时设置同步轴绝对移动位置

参数：position　绝对位置；**类型：**DOUBLE

例程

```
BASE 0,1,2,3
STA_SRC=1  '设置同步源为板卡 STA 信号
SETSTA 10000,5000,30000,-8000 '设置轴 0,1,2,3 处于等待同步相对运动状态，并设置运动距离
STARTSTA     '同步启，同时启动 4 轴运动
WAIT DONE
SETSTA_ABS 0,2000,5000,20000  '设置轴 0,1,2,3 处于等待同步绝对运动状态，并设置目标位置
SLEEP 500
STARTSTA       '同步启，同时启动 4 轴运动
WAIT DONE
SETSTA_VEL 0,1,0,1 '设置轴 0,1,2,3 处于等待同步连续运动状态，并设置各轴方向
STARTSTA       '同步启，同时启动 4 轴运动
Sleep 3000
STOPSTA        '同步停，同时停止 4 轴运动
```

2.9.4 SETSTA_VEL

所属：命令

语法：SETSTA_VEL dir1[,dir2][,dir3]……

描述：设置同步轴为连续运动模式并处于等待触发状态，同时设置同步轴的连续运动方向。

参数：dir　方向，0：正向，1：反向；**类型：**ULONG

例程

```
BASE 0,1,2,3
STA_SRC=1  ''设置同步源为板卡 STA 信号
SETSTA 10000,5000,30000,-8000 '设置轴 0,1,2,3 处于等待同步相对运动状态，并设置运动距离
STARTSTA     '同步启，同时启动 4 轴运动
WAIT DONE
SETSTA_ABS 0,2000,5000,20000  '设置轴 0,1,2,3 处于等待同步绝对运动状态，并设置目标位置
SLEEP 500
STARTSTA       '同步启，同时启动 4 轴运动
WAIT DONE
SETSTA_VEL 0,1,0,1 '设置轴 0,1,2,3 处于等待同步连续运动状态，并设置各轴方向
STARTSTA       '同步启，同时启动 4 轴运动
Sleep 3000
STOPSTA        '同步停，同时停止 4 轴运动
```

2.9.5 STARTSTA

所属：命令

语法：STARTSTA

描述：当轴的同步启/停触发源为 STA 信号时，使用该指令开始同步启动运动

例程

请参考 SETSTA、SETSTA_ABS、SETSTA_VEL 等指令**例程**

2.9.6 STOPSTA

所属：命令

语法：STOPSTA

描述：当轴的同步启/停触发源为 STA 信号时，使用该指令开始同步停止运动

例程

请参考 SETSTA、SETSTA_ABS、SETSTA_VEL 等指令**例程**

2.9.7 GANTRY

所属：命令

语法：GANTRY AX(slaveaxis no),refsrc,dir

描述：指定龙门运动从轴、参考源、龙门运动方向，建立龙门同步关系。GANTRY 的主轴是以该指令执行时，当前 BASE 列表中的第一个轴为主轴的。主轴需在 GANTRY 指令执行前指定。

参数：slave axis no　从轴的轴号；**范围：**根据控制器实际硬件决定

refsrc　从轴跟随主轴的位置源；**范围：**0：理论位置，1：实际位置（暂不支持）

dir　从轴与主轴的运动方向一致性；**范围：**0：相同，1：相反

注意：龙门一旦建立龙门关系，主从轴状态会变成同步状态，要解除龙门关系，需对从轴下 STOPDEC 或 STOPEMG 命令，龙门关系即解除。

例程

```
BASE 0      '指定 GANTRY 的主轴为轴 0
STOPDEC AX(1) '龙门运动中，对从轴下 STOPDEC 或 STOPEMG 可以解除龙门关系
'如果已有龙门关系存在，再下 GANTRY 命令，会报无效轴状态错误
GANTRY AX(1),0,0   '设定龙门关系：从轴为轴 1，参考源为主轴的理论位置，从轴方向与主轴同向
BASE 0
MOVE 20000   '主轴执行距离为 20000 的正向相对运动，从轴这时会与跟随主轴一起执行龙门运动
```

2.9.8 GEAR

所属：命令

语法：GEAR AX(slaveaxis no),numerator,denominator,refsrc,mode

描述：指定电子齿轮运动从轴、齿轮分子、齿轮分母、参考源、运动模式，建立电子齿轮同步关系。GEAR 的主轴是以该指令执行时，当前 BASE 列表中的第一个轴为主轴的。主轴需在 GEAR 指令执行前指定。

参数：slave axis no　从轴的轴号；**范围：**根据控制器实际硬件决定

numerator　电子齿轮比分子；类型 ULONG

denominator　电子齿轮比分子；类型 ULONG

refsrc　从轴跟随主轴的位置源；**范围：**0：理论位置，1：实际位置

mode　主从轴齿轮关系模式；**范围：**0：相对位置主从模式，1：绝对位置主从模式

注意：一旦建立电子齿轮关系，主从轴状态会变成同步状态，要解除电子齿轮关系，需对从轴下 STOPDEC 或 STOPEMG 命令，齿轮关系即解除。

例程

```
BASE 0          '指定 GEAR 的主轴为轴 0
STOPDEC AX(1) '电子齿轮运动中，对从轴下 STOPDEC 或 STOPEMG 可以解除齿轮关系
'如果已有齿轮关系存在，再下 GEAR 命令，会报无效轴状态错误
GEAR AX(1),1,1,0,0    '从轴为轴 1，齿轮分子分母分别为 1,1，从轴参考主轴理论位置，相对位置模式
BASE 0
MOVE 20000    '主轴执行距离为 20000 的正向相对运动，从轴这时会与跟随主轴一起执行电子齿轮运动
```

2.9.9 PHASE

所属：命令

语法：PHASE AX(slaveaxis no),acc,dec,phase_speed,phase_dist

描述：电子齿轮或电子凸轮过程中使从轴进行相位超前或落后运动

参数：slave axis no 从轴的轴号；**范围：**根据控制器实际硬件决定

acc 相位运动的加速度；类型 DOUBLE

dec 相位运动的减速度；类型 DOUBLE

phase_speed 相位运动的运行速度；类型 DOUBLE

phase_dist 相位运动的超前或落后距离；类型 DOUBLE。Phase_dist >0,从轴做相位超前运动；phase_dist <0,从轴做相位落后运动。

注意：相位运动指令需在电子齿轮或电子凸轮运动过程中执行才起作用。

例程

```
BASE 0,1        '指定 GEAR 的主轴为轴 0
SVON
'齿轮关系的所有轴速度等参数由主轴参数决定
VL=1000
VH=40000
ACC=200000
DEC=200000
STOPDEC AX(1) '电子齿轮运动中，对从轴下 STOPDEC 或 STOPEMG 可以解除齿轮关系
'如果已有齿轮关系存在，再下 GEAR 命令，会报无效轴状态错误
GEAR AX(1),1,1,0,0    '从轴为轴 1，齿轮分子分母分别为 1,1，从轴参考主轴理论位置，相对位置模式
BASE 0
MOVE 80000    '主轴执行距离为 20000 的正向相对运动，从轴这时会与跟随主轴一起执行齿轮运动
Sleep 1000
Phase AX(1),50000,50000,30000,10000 '轴 1 进行距离为 10000 个 UNIT 的相位超前运动。
```

2.9.10 TANGENT

所属：命令

语法：TANGENTAX(axis no), start_vector*, dir[,module_range]

描述：指定切向跟随轴、起始切向向量、运动方向，建立切向跟随关系

参数：AX(axis no) 切向跟随轴的轴号

start_vector* 起始切向向量数组地址

dir 跟随轴旋转方向；**范围：**0：与切向方向相同，1：与切向方向相反

module_range 刀向旋转轴旋转一周的指令脉冲数

注意：一旦建立切向跟随，主从轴状态会变成同步状态，要解除切向跟随关系，需对从轴下 STOPDEC 或 STOPEMG 命令，切向跟随关系即解除。

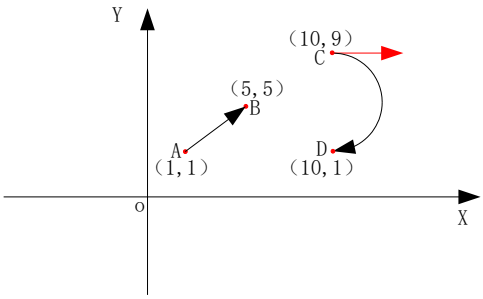
起始切向向量说明：

- 旋转轴跟随的运动是直线，起始切向向量的方向与直线运动的方向一致；
- 旋转轴跟随的运动是圆弧或曲线，起始切向向量的方向与圆弧或曲线的当前点切向运动方向一致。

如下图：

AB 段直线：旋转轴要在直线 AB 段进行切向跟随运动，从 A（1,1）运动到 B(5,5)。起始切向向量即为下图 AB 段箭头的向量（B 点坐标减去 A 点坐标），因跟随的是 XY 平面的运动，Z 方向为零，所以起始切向向量为（5-1,5-1,0），即（4,4,0）。

CD 段圆弧：旋转轴要在半圆 CD 段进行切向跟随运动，从 C（10,9）运动到 D(10,1)。起始切向向量为下图红色箭头方向，因刚好是半径为 4 的半圆，起始切向向量刚好与 X 轴正向一致，所以起始切向向量为(1,0,0)。



例程

```
BASE 0,1
DIM StartArray(2) as SHORT
StartArray(0)=0
StartArray(1)=1
StartArray(2)=0
'跟随的旋转刀控制轴为轴 2，起始刀向向量为（0,1,0），轴 1 组成的平面，旋转方向与刀向方向相同，旋转刀轴运动一圈需要 3600 个脉冲
TANGENT AX(2),StartArray(),0,3600
CIRC 0,8000, 0,16000, 0   ' 轴 0，轴 1 进行圆弧插补运动，这时轴 2 会同时执行刀向跟随运动
WAIT DONE
STOPDEC AX(2)
```

2.9.11 MODULO

所属：属性

语法：MODULO=value

类型：ULONG

描述：设置/读取 ModuleRange 值。切向跟随功能中，该值为刀向旋转轴旋转一周的指令脉冲数

范围：【0，8000000】，该值必须为 4 的倍数；默认值为 0

例程

```
BASE 0
MODULO =10000 '设定轴 0 的 MODULO 值为 10000
```

2.9.12 CAMTABLE

语法：CAMTABLE CamIndex, AX(MasAxisNo), AX(SlvAxisNo),VR_Start,DataCount,CamID,IsSpdSet

描述：设定凸轮表数据。凸轮表可通过 IDE 上的凸轮工具产生，详细请参考 MotionStudio 使用手册。

参数：

- CamIndex：0~7，凸轮表在 Motion Studio 的编号（使用不同的凸轮表需指定不同的 index）
- AX(MasAxisNo)：指定主轴轴号
- AX(SlvAxisNo)：指定从轴轴号
- VR_Start: 存放凸轮关系数据的起始 Index，在 IDE 上的凸轮工具上设定的 VR Index 一致。
- DataCount：凸轮表中的点数
- CamID：指定 CamIndex 与 Device 上的实际 ID 对应关系（每张板卡支持两个 CamID：0/1）
- IsSpdSet：1：使用自定义速度；0：使用板卡自动计算速度。在 MotionStudio 上可在凸轮工具中修改每个点主从速度比。

  1---使用用户规划速度 ,使用 VR(VR_Start + 2* DataCount)~ VR(VR_Start +2*DataCount+1)的速度。

  0---使用板卡内部自动计算速度。

凸轮表数据对应 VR 说明：

VR(VR_Start) ---第 1 个点的 X 值，单位：pulse

VR(VR_Start +1) ---第 1 个点的 Y 值，单位：pulse

VR(VR_Start +2) ---第 2 个点的 X 值，单位：pulse

VR(VR_Start +3) ---第 2 个点的 Y 值，单位：pulse

……

VR(VR_Start + 2* DataCount -2)---最后 1 个点的 X 值，单位：pulse

VR(VR_Start + 2*DataCount -1)---最后 1 个点的 Y 值，单位：pulse

VR(VR_Start + 2* DataCount) ---第 1 个点的主从速度比

VR(VR_Start + 2* DataCount +1) ---第 2 个点的主从速度比

……

VR(VR_Start + 3* DataCount - 1) ---最后 1 个点的主从速度比

例程：

'如下例程已先使用 MotionStudio 上的凸轮工具产生了凸轮表，并将数据写入了 VR(1000)开始的位置。

```
base 0,1
'CamIndex 为 1,AX(0)为主轴,AX(1)为从轴,
'VR(1000)~VR(1011)为 6 个凸轮點座標值 ,VR(1012)~VR(1017)6 个凸轮點速度值，使用版卡 CamID 0,
使用自定义速度规划
CAMTABLE(1, AX(0), AX(1), 1000, 6, 0,1)
CAMSET 1,0,0,1  '设定主轴和从轴都是相对模式,周期性跟随
CAMIN 1         '建立主从跟随关系

BASE 0
```

```
MOVE 10000 '主轴开始移动，从轴按照凸轮表跟随移动
WAIT DONE

CAMSTOP AX(1)  '解除从轴的跟随关系
```

2.9.13 CAMSET

语法：CAMSET CamIndex[,MasAbsOrRel][, SlvAbsOrRel][,Periodic]

描述：设定凸轮表，需先使用 CAMTABLE 将数据写入硬件中。

参数：CamIndex：0~7. 与 CAMTABLE 一致

MasAbsOrRel：主轴绝对/相对匹配凸轮表，0—相对，1—绝对。

SlvAbsOrRel：从轴绝对/相对匹配凸轮表，0—相对，1—绝对。

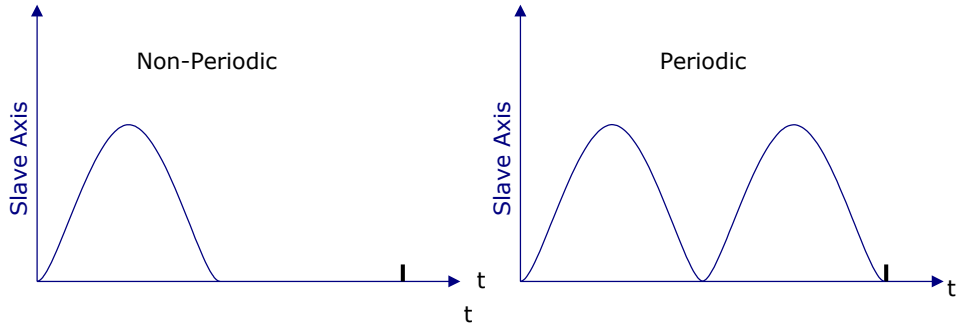
绝对：当绝对凸轮系统设置后，基于 CamTable 的控制值或从值将被认为是绝对的。系统会补偿同步过程中主从轴之间的任何偏差。当达到同步时，控制值和从值之间将建立一种确定的相位关系。

相对：当相对凸轮系统设置后，则表示系统不会补偿在同步过程中控制值和从值之间的任何偏移。

Periodic：

0—非周期跟随，随着主轴的移动，凸轮表完成一个周期后将从轴停止动作；

1—周期跟随，随着主轴的移动，从轴周期性地跟随主轴依照凸轮表动作。



例程：参考 CAMTABLE 例程。

2.9.14 CAMIN

语法：CAMIN CamIndex, [MasOffset=0,][SlvOffset=0,][MasRatioNum =1,][MasRatioDemo =1,] [SlvRatioNum =1,][SlvRatioDemo = 1,][RefSrc = 0]

描述： 建立主从轴的凸轮关系，并设定相关数据，调用此指令后，主从轴会建立跟随关系，从轴的状态会从 ready 状态变成同步状态。

参数：CamIndex：0~7. 与 CAMTABLE 一致

MasOffset：主轴方向的坐标偏移值。

SlvOffset：从轴方向的坐标偏移值。

MasRatioNum：主轴坐标中 CAMTABLE 的比例因子的分子。

MasRatioDemo：主轴坐标中 CAMTABLE 的比例因子的分母。

SlvRatioNum：从轴坐标中 CAMTABLE 的比例因子的分子。

SlvRatioDemo：从轴坐标中 CAMTABLE 的比例因子的分母。

RefSrc：凸轮跟随参照源。0—理论位置，1—回馈位置。

例程：参考 CAMTABLE **例程。**

2.9.15 CAMSTOP

语法 1：CAMSTOP

语法 2：CAMSTOP AX(SlvAxNo)

描述：解除从轴的跟随关系。

参数：SlvAxNo 从轴轴号

例程：参考 CAMTABLE **例程。**

2.10 输入输出端口控制

2.10.1 通用数字量输入/输出控制

MAS 控制器会自动将控制器中所有运动控制单元的“轴上 DIO”和“设备 DIO”进行顺序编号，用户再通过编号去操作对应的 DIO。比如 MAS 控制器中有 2 张 PCI-1245L-MAS 的控制单元，每张 PCI-1245L-MAS 有 16 个 DI,16 个 DO，那 MAS 控制器中的 DIO 映射编号会有：DIN(0)~DIN(31)；DOUT(0)~DOUT(31)。哪个控制单元对应哪个区间的编号可以在 Motion Studio 的“硬件配置”中查询到。“轴上 DIO”和“设备 DIO”的说明请参照附录章节 3.4“MAS 控制器中的控制单元”

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.10.1.1	DIN	数字量输入	√	√
2.10.1.2	DOUT	数字量输出	√	√
2.10.1.3	DORESET	将指定区域 DO 值设为初始值	×	√

2.10.1.1 DIN

所属：命令（只读）

语法：DIN (no)

描述：读取一个通用数字量输入端口的状态

参数：no，数字量输入的编号，在控制器配置时，系统会根据控制器硬件分配对应的编号；**类型：**ULONG

返回值：0：低电平，1：高电平

例程

```
WHILE 1
IF DIN (0)=1 THEN     '判断 DI0 信号是否有信号
   DOUT (0)=1       'DO0,DO1 置 1，DO2 置 0
   DOUT (1)=1
   DOUT (2)=0
ELSE
   DOUT (2)=1
END IF
WEND
```

2.10.1.2 DOUT

所属：命令

语法：DOUT (no)=value

描述：设置/读取一个通用数字量输出端口的状态

参数：no 数字量输出的编号，在控制器配置时，系统会根据控制器硬件分配对应的编号；**类型：**ULONG

value 数字量输出端口的状态，范围：0 或 1

返回值：输出口的电平，0 或 1

例程

```
WHILE 1
IF DIN (0)=1 THEN     '判断 DI0 信号是否有信号
   DOUT (0)=1       'DO0,DO1 置 1，DO2 置 0
   DOUT (1)=1
   DOUT (2)=0
ELSE
   DOUT (2)=1
END IF
WEND
```

2.10.1.3 DORESET

所属：命令

语法：DORESET (StartIndex, DoCnt)

类型：ULONG

描述：将指定区域的 DO 值设置为初始值

参数：StartIndex：起始 DO index。（第一个 Index 为 0）

DoCnt：要初始化的 DO 个数。

例程

```
DORESET (0, 10)  '将 0~9 这 10 个 DO 的值设置为初始值
```

2.10.2 轴运动相关的输入/输出控制

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.10.2.1	EMG_LOGIC	EMG 逻辑电平	√	√
2.10.2.2	EMG_FILTER	EMG 端口滤波	√	×
2.10.2.3	ALM_EN	伺服报警使能	√	×
2.10.2.4	ALM_FILTER	伺服报警端口滤波	√	√
2.10.2.5	ALM_LOGIC	伺服报警端口逻辑电平	√	√
2.10.2.6	ALM_MODE	伺服报警触发后运动停止模式	√	√
2.10.2.7	IN1STOP_EN	IN1STOP 功能使能	√	×
2.10.2.8	IN1_FILTER	IN1 端口滤波	√	√
2.10.2.9	IN1STOP_EDGE	IN1STOP 功能的触发条件	√	√
2.10.2.10	IN1STOP_MODE	IN1STOP 触发后运动停止模式	√	√
2.10.2.11	IN1STOP_OFFSET	IN1STOP 信号触发后再移动的偏移距离	√	√
2.10.2.12	IN2STOP_EN	IN2STOP 功能使能	√	×
2.10.2.13	IN2_FILTER	IN2 端口滤波	√	√
2.10.2.14	IN2STOP_EDGE	IN2STOP 功能的触发条件	√	√
2.10.2.15	IN2STOP_MODE	IN2STOP 触发后运动停止模式	√	√
2.10.2.16	IN2STOP_OFFSET	IN2STOP 信号触发后再移动的偏移距离	√	√
2.10.2.17	IN4STOP_EN	IN4STOP 功能使能	√	×
2.10.2.18	IN4_FILTER	IN4 端口滤波	√	√
2.10.2.19	IN4STOP_EDGE	IN4STOP 功能的触发条件	√	√
2.10.2.20	IN4STOP_MODE	IN4STOP 触发后运动停止模式	√	√
2.10.2.21	IN4STOP_OFFSET	IN4STOP 信号触发后再移动的偏移距离	√	√
2.10.2.22	IN5STOP_EN	IN5STOP 功能使能	√	×

2.10.2.23	IN5_FILTER	IN5 端口滤波	√	√
2.10.2.24	IN5STOP_EDGE	IN5STOP 功能的触发条件	√	√
2.10.2.25	IN5STOP_MODE	IN5STOP 触发后运动停止模式	√	√
2.10.2.26	IN5STOP_OFFSET	IN5STOP 信号触发后再移动的偏移距离	√	√
2.10.2.27	INSTOP_DEC	INSTOP 功能触发后减速停止	√	×
2.10.2.28	INP_EN	伺服到位功能使能	√	×
2.10.2.29	INP_LOGIC	伺服到位端口逻辑电平	√	√
2.10.2.30	CAMDO_EN	CAMDO 功能使能	√	×
2.10.2.31	CAMDO_LOGIC	CAMDO 端口逻辑电平	√	√
2.10.2.32	CAMDO_LPOS	CAMDO 低限位位置值	√	√
2.10.2.33	CAMDO_HPOS	CAMDO 高限位位置值	√	√
2.10.2.34	CAMDO_SRC	CAMDO 比较触发源	√	√
2.10.2.35	MIO	运动控制相关的 I/O 状态	×	×

2.10.2.1 EMG_LOGIC

所属：属性

语法：EMG_LOGIC = value

类型：ULONG

描述：设置/读取板卡的 EMG 逻辑电平

范围：设定值和返回值如下，默认值 0

0：低电平

1：高电平

例程

```
BASE 0
EMG_LOGIC=1 '设置板卡的 EMG 逻辑电平为高电平
```

2.10.2.2 EMG_FILTER

所属：属性

语法：EMG_FILTER = value

类型：ULONG

描述：设置/读取 EMG 端口的滤波时间

范围：设定值和返回值如下，默认值 0

0：5us

1：100us

2：200us

3：500us

例程

```
BASE 0
EMG_FILTER =1  '设置 EMG 信号的滤波时间为 100us
```

2.10.2.3 ALM_EN

所属：属性

语法：ALM_EN = value

类型：ULONG

描述：启用/禁用运动报警功能，报警是当电机驱动处于报警状态时，电机驱动器生成的信号

范围：设定值和返回值如下，默认值 0

0：禁用(默认值)

1：启用

例程

```
BASE 0
ALM_EN=1    '启用轴 0 检测报警功能
```

2.10.2.4 ALM_FILTER

所属：属性

语法：ALM_FILTER = value

类型：ULONG

描述：设置/读取轴的报警(ALM) 输入端口的滤波参数

范围：设定值和返回值如下，默认值 0

0：5us

1：100us

2：200us

3：500us

例程

```
BASE 0
ALM_FILTER=1 '设定轴 0 的 ALM 输入端口滤波时间为 100us
```

2.10.2.5 ALM_LOGIC

所属：属性

语法：ALM_LOGIC = value

类型：ULONG

描述：设置/读取报警输入信号的有效逻辑电平

范围 :设定值和返回值如下，默认值 0

0：低电平

1：高电平

例程

```
BASE 0
ALM_LOGIC =1  '设置轴 0 的报警输入信号高电平有效
```

2.10.2.6 ALM_MODE

所属：属性

语法：ALM_MODE = value

类型：ULONG

描述：设置/读取接收报警信号时电机的停止模式

范围：设定值和返回值如下，默认值 0

0：立即停止

1：减速停止

例程

```
BASE 0
ALM_MODE=1  '设置接收到报警信号后电机减速停止
```

2.10.2.7 IN1STOP_EN

所属：属性

语法：IN1STOP_EN = value

类型：ULONG

描述：启用/禁用 IN1 的触发停止功能，该功能启动用后，IN1 信号一旦有效，在运动中的指定电机会被控制停止运动。研华运动控制的每个轴都关联着 4 个 DI 端口，分别称为 IN1，IN2，IN4，IN5。4 个端口都可以指定启用 INSTOP 功能。

范围：设定值和返回值如下，默认值 0

0：禁用

1：启用

例程

```
BASE 0
IN1STOP_EN =1  '启用轴 0 的 IN1 触发停止功能
```

2.10.2.8 IN1_FILTER

所属：属性

语法：IN1_FILTER = value

类型：ULONG

描述：设置/读取 IN1 端口的滤波时间

范围：设定值和返回值如下，默认值 0

0：5us

1：100us

2：200us

3：500us

例程

```
BASE 0
IN1_FILTER =1  '设置轴 0 的 IN1 信号滤波时间 100us
```

2.10.2.9 IN1STOP_EDGE

所属：属性

语法：IN1STOP_EDGE = value

类型：ULONG

描述：设置/读取 IN1STOP 功能的触发条件。设置上升沿触发时，IN1 端口有上升沿信号时， IN1STOP 功能被触发。设置下降沿触发时，IN1 端口有下降沿信号时，IN1STOP 功能被触发。

范围：设定值和返回值如下，默认值 0

0：上升沿

1：下降沿

例程

```
BASE 0
IN1STOP_EDGE =1 '设置 IN1STOP 功能的触发条件为下降沿触发有效。
```

2.10.2.10 IN1STOP_MODE

所属：属性

语法：IN1STOP_MODE = value

类型：ULONG

描述：设置/读取 IN1 触发时电机的停止模式

范围：设定值和返回值如下，默认值 1

0：立即停止

1：减速停止

例程

```
BASE 0
IN1STOP_MODE =1 '设置 IN1 触发时电机作减速停止运动
```

2.10.2.11 IN1STOP_OFFSET

所属：属性

语法：IN1STOP_OFFSET = value

类型：DOUBLE

描述：设置/读取 IN1STOP 信号触发后再移动的偏移距离。

例程

```
BASE 0
IN1STOP_OFFSET =5 '设置轴 0 的 IN1STOP 信号触发后再移动的偏移距离为 5 个 UNIT
```

2.10.2.12 IN2STOP_EN

所属：属性

语法：IN2STOP_EN = value

类型：ULONG

描述：启用/禁用 IN2 的触发停止功能，该功能启动用后，IN2 信号一旦有效，在运

动中的指定电机会被控制停止运动。研华运动控制的每个轴都关联着 4 个 DI 端口 ,分别称为 IN1 ,IN2 ,IN4 , IN5。4 个端口都可以指定启用 INSTOP 功能。

范围：设定值和返回值如下，默认值 0

0：禁用

1：启用

例程

```
BASE 0
IN2STOP_EN =1  '启用轴 0 的 IN2 触发停止功能
```

2.10.2.13 IN2_FILTER

所属：属性

语法：IN2_FILTER = value

类型：ULONG

描述：设置/读取 IN2 端口的滤波时间

范围：设定值和返回值如下，默认值 0

0：5us

1：100us

2：200us

3：500us

例程

```
BASE 0
IN2_FILTER =1  '设置轴 0 的 IN2 信号滤波时间 100us
```

2.10.2.14 IN2STOP_EDGE

所属：属性

语法：IN2STOP_EDGE = value

类型：ULONG

描述：设置/读取 IN2STOP 功能的触发条件。设置上升沿触发时，IN2 端口有上升沿信号时，IN2STOP 功能被触发。设置下降沿触发时，IN2 端口有下降沿信号时，IN2STOP 功能被触发。

范围 :设定值和返回值如下，默认值 0

0：上升沿

1：下降沿

例程

```
BASE 0
IN2STOP_EDGE =1 '设置 IN2STOP 功能的触发条件为下降沿触发有效。
```

2.10.2.15 IN2STOP_MODE

所属：属性

语法：IN2STOP_MODE = value

类型：ULONG

描述：设置/读取 IN2 触发时电机的停止模式

范围：设定值和返回值如下，默认值 1

0：立即停止

1：减速停止

例程

```
BASE 0
IN2STOP_MODE =1 '设置 IN2 触发时电机作减速停止运动
```

2.10.2.16 IN2STOP_OFFSET

所属：属性

语法：IN2STOP_OFFSET = value

类型：DOUBLE

描述：设置/读取 IN2STOP 信号触发后再移动的偏移距离。

例程

```
BASE 0
IN2STOP_OFFSET =5 '设置轴 0 的 IN2STOP 信号触发后再移动的偏移距离为 5 个 UNIT
```

2.10.2.17 IN4STOP_EN

所属：属性

语法：IN4STOP_EN = value

类型：ULONG

描述：启用/禁用 IN4 的触发停止功能，该功能启动用后，IN4 信号一旦有效，在动

中的指定电机会被控制停止运动。研华运动控制的每个轴都关联着 4 个 DI 端、

口，分别称为 IN1，IN2，IN4，IN5。4 个端口都可以指定启用 INSTOP 功能。

范围：设定值和返回值如下，默认值 0

0：禁用

1：启用

例程

```
BASE 0
IN4STOP_EN =1  '启用轴 0 的 IN4 触发停止功能
```

2.10.2.18 IN4_FILTER

所属：属性

语法：IN4_FILTER = value

类型：ULONG

描述：设置/读取 IN4 端口的滤波时间

范围：设定值和返回值如下，默认值 0

0：5us

1：100us

2：200us

3：500us

例程

```
BASE 0
IN4_FILTER =1  '设置轴 0 的 IN4 信号滤波时间 100us
```

2.10.2.19 IN4STOP_EDGE

所属：属性

语法：IN4STOP_EDGE = value

类型：ULONG

描述：设置/读取 IN4STOP 功能的触发条件。设置上升沿触发时，IN4 端口有上升沿信号时， IN4STOP 功能被触发。设置下降沿触发时，IN4 端口有下降沿信号时，IN4STOP 功能被触发。

范围：设定值和返回值如下，默认值 0

0：上升沿

1：下降沿

例程

```
BASE 0
IN4STOP_EDGE =1 '设置 IN4STOP 功能的触发条件为下降沿触发有效。
```

2.10.2.20 IN4STOP_MODE

所属：属性

语法：IN4STOP_MODE = value

类型：ULONG

描述：设置/读取 IN4 触发时电机的停止模式

范围：设定值和返回值如下，默认值 1

0：立即停止

1：减速停止

例程

```
BASE 0
IN4STOP_MODE =1 '设置 IN4 触发时电机作减速停止运动
```

2.10.2.21 IN4STOP_OFFSET

所属：属性

语法：IN4STOP_OFFSET = value

类型：DOUBLE

描述：设置/读取 IN4STOP 信号触发后再移动的偏移距离。

例程

```
BASE 0
IN4STOP_OFFSET =5 '设置轴 0 的 IN4STOP 信号触发后再移动的偏移距离为 5 个 UNIT
```

2.10.2.22 IN5STOP_EN

所属：属性

语法：IN5STOP_EN = value

类型：ULONG

描述：启用/禁用 IN5 的触发停止功能，该功能启动用后，IN5 信号一旦有效，在运动中的指定电机会被控制停止运动。研华运动控制的每个轴都关联着 4 个 DI

端口，分别称为 IN1，IN2，IN4，IN5。4 个端口都可以指定启用 INSTOP 功

能。

范围：设定值和返回值如下，默认值 0

0：禁用

1：启用

例程

```
BASE 0
IN5STOP_EN =1  '启用轴 0 的 IN5 触发停止功能
```

2.10.2.23 IN5_FILTER

所属：属性

语法：IN5_FILTER = value

类型：ULONG

描述：设置/读取 IN5 端口的滤波时间

范围：设定值和返回值如下，默认值 0

0：5us

1：100us

2：200us

3：500us

例程

```
BASE 0
IN5_FILTER =1  '设置轴 0 的 IN5 信号滤波时间 100us
```

2.10.2.24 IN5STOP_EDGE

所属：属性

语法：IN5STOP_EDGE = value

类型：ULONG

描述：设置/读取 IN5STOP 功能的触发条件。设置上升沿触发时，IN5 端口有上升沿信号时， IN5STOP 功能被触发。设置下降沿触发时，IN5 端口有下降沿信号时，IN5STOP 功能被触发。

范围：设定值和返回值如下，默认值 0

0：上升沿

1：下降沿

例程

```
BASE 0
IN5STOP_EDGE =1 '设置 IN5STOP 功能的触发条件为下降沿触发有效。
```

2.10.2.25 IN5STOP_MODE

所属：属性

语法：IN5STOP_MODE = value

类型：ULONG

描述：设置/读取 IN5 触发时电机的停止模式

范围：设定值和返回值如下，默认值 1

0：立即停止

1：减速停止

例程

```
BASE 0
IN5STOP_MODE =1   '设置 IN5 触发时电机作减速停止运动
```

2.10.2.26 IN5STOP_OFFSET

所属：属性

语法：IN5STOP_OFFSET = value

类型：DOUBLE

描述：设置/读取 IN5STOP 信号触发后再移动的偏移距离。

例程

```
BASE 0
IN5STOP_OFFSET =5 '设置轴 0 的 IN5STOP 信号触发后再移动的偏移距离为 5 个 UNIT
```

2.10.2.27 INSTOP_DEC

所属：属性

语法：INSTOP_DEC = value

类型：DOUBLE

描述：设置/读取 INSTOP 用减速停止模式时的减速度，单位元为 UNIT/s^2

范围：(0,MAXDEC) ，默认值 100000

例程

```
BASE 0
INSTOP_DEC=20000 '设置轴 0 的 INSTOP 减速度为 20000 个 UNIT/s^2
```

2.10.2.28 INP_EN

所属：属性

语法：INP_EN = value

类型：ULONG

描述：启用/禁用检测电机运动到位功能，到位信号是当电机运动到位时，电机驱动器生成到位信号。

范围:设定值和返回值如下，默认值 0

0：禁用

1：启用

例程

```
BASE 0
INP_EN =1 '启用轴 0 的到位检测功能
```

注意：
启用到位检测功能后，控制轴进行运动时，脉冲命令输出完后，轴状态不会立即变为 ready，要等到轴的 INP 端口接收到伺服送出的到位信号，轴状态才会变为 ready。轴状态未变为 ready 时，对该轴下运动指令将不成功。

2.10.2.29 INP_LOGIC

所属：属性

语法：INP_LOGIC = value

类型：ULONG

描述：设置/读取到位输入信号的有效逻辑电平

范围：设定值和返回值如下，默认值 1

0：低电平

1：高电平

例程

```
BASE 0
INP_LOGIC =1     '设置轴 0 的到位输入信号高电平有效
```

2.10.2.30 CAMDO_EN

所属：属性

语法：CAMDO_EN = value

类型：ULONG

描述：启动/禁用 CAMDO 功能，研华运动控制的每个轴都关联着 4 个 DO 端口，

分别称为 OUT4，OUT5，OUT6，OUT7，每个轴的 CAMDO 输出由 OUT4 输出端口产生，启用 CAMDO 功能后，一旦触发产生，OUT4 即输出信号。

范围：设定值和返回值如下，默认值 0

0：禁用

1：启用

例程

```
BASE 0
CAMDO_EN=1    '启用轴 0 上的 CAMDO 功能
```

2.10.2.31 CAMDO_LOGIC

所属：属性

语法：CAMDO_LOGIC = value

类型：ULONG

描述：设置/读取 CAMDO 有效输出时 DO 的电平逻辑

范围：设定值和返回值如下，默认值 1

0：低电平

1：高电平

例程

```
BASE 0
CAMDO_LOGIC  =1 '设置 CAMDO 有效输出时的电平为高电平
```

2.10.2.32 CAMDO_LPOS

所属：属性

语法：CAMDO_LPOS = value

类型：ULONG

描述：设置/读取 CAMDO 低限位位置值，单位为 UNIT

范围：设定值和返回值为【-2147483648，2147483647】，默认值 10000

例程

```
BASE 0
CAMDO_LPOS=1000  '设置 CAMDO 低限位位置值为 1000
```

2.10.2.33 CAMDO_HPOS

所属：属性

语法：CAMDO_HPOS = value

类型：ULONG

描述：设置/读取 CAMDO 高限位位置值，单位为 UNIT

范围：设定值和返回值为【-2147483648，2147483647】，默认值 20000

例程

```
BASE 0
CAMDO_HPOS=1000  '设置 CAMDO 高限位位置值为 1000
```

2.10.2.34 CAMDO_SRC

所属：属性

语法：CAMDO_SRC = value

类型：ULONG

描述：设置/读取 CAMDO 的位置比较源

范围：设定值和返回值如下，默认值 0

0：理论位置

1：实际位置

例程

```
BASE 0
CAMDO_SRC=1 '设置轴 0 的 CAMDO 位置比较源为实际位置
```

2.10.2.35 MIO

所属：属性（只读）

语法：如下枚举

- value=MIO.RDY ，读取伺服驱动器 Ready 信号
- value=MIO.ALM，读取伺服驱动器报警信号
- value=MIO.PEL，读取正方向硬极限信号
- value=MIO.NEL，读取负方向硬极限信号
- value=MIO.ORG，读取硬件原点信号
- value=MIO.DIR，读取轴运动方向信号
- value=MIO.EMG，读取轴卡上 EMG 信号
- value=MIO.PCS，暂不支持
- value= MIO.ERC，暂不支持
- value= MIO.EZ，读取编码器 Z 相输入信号
- value= MIO.CLR，暂不支持
- value= MIO.LTC，读取锁存信号
- value= MIO.SD，暂不支持
- value= MIO.INP，读取伺服驱动器到位信号
- value= MIO.SVON，读取伺服使能输出信号
- value=MIO.ALRM，读取报警复位信号输出状态
- value= MIO.SPEL，读取正方向软极限信号
- value= MIO.SNEL，读取负方向软极限信号
- value= MIO.CMP，读取比较触发输出信号
- value= MIO.CAMDO，读取 CAM DO 输出信号

类型：ULONG

描述：读取跟运动控制相关的 I/O 状态

返回值：0 或 1

2.10.3 高速位置锁存

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.10.3.1	LTC_EN	锁存功能使能	√	×
2.10.3.2	LTC_LOGIC	锁存端口逻辑电平	√	√
2.10.3.3	LDPOS	锁存到的理论位置	√	×
2.10.3.4	LMPOS	锁存到的实际位置	√	×
2.10.3.5	TRIGLTC	软件触发锁存功能	√	×
2.10.3.6	LTC_FLAG	轴锁存信号标志	√	×
2.10.3.7	RESETLTC	清除锁存位置和标识	√	×
2.10.3.8	LBUF_RESET	清除 Latch Buffer 中所有数据	×	×
2.10.3.9	LBUF_EN	Latch Buffer 使能	×	×
2.10.3.10	LBUF_DIST	Latch Buffer 锁存距离间隔	×	×
2.10.3.11	LBUF_SRC	Latch Buffer 比较触发源	×	×
2.10.3.12	LBUF_ID	设定/读取 LatchBuffer 中的数据源	×	×
2.10.3.13	LBUF_EVTNUM	产生 LTCBUFDONE 事件锁存到的数据个数	×	×
2.10.3.14	LBUF_STATUS	获取 latchbuffer 中数据个数以及空间大小	×	×
2.10.3.15	LBUF_DATA	获取 Latchbuffer 中的资料	×	×

2.10.3.1 LTC_EN

所属：属性

语法：LTC_EN = value

类型：ULONG

描述：启用/禁用轴的锁存功能，研华运动控制的每个轴都关联着 4 个 DI 端口，分

别称为 IN1，IN2，IN4，IN5，每个轴的锁存信号由 IN1 输入端口产生，启用锁存功能后，一旦 IN1 端口接收到有效电平，控制器会立即锁存指令理论位置值和编码器实际位置值。

范围：设定值和返回值如下，默认值 0

0：禁用

1：启用

例程

```
BASE 0
LTC_EN=1 '启用轴 0 锁存功能
```

2.10.3.2 LTC_LOGIC

所属：属性

语法：LTC_LOGIC = value

类型：ULONG

描述：设置/读取锁存输入信号的有效逻辑电平

范围：设定值和返回值如下，默认值 0

0：低电平

1：高电平

例程

```
BASE 0
LTC_LOGIC =0  '设置轴 0 锁存输入信号低电平有效
```

2.10.3.3 LDPOS

所属：命令

语法：value=LDPOS(AX(no))

类型：DOUBLE

描述：读取触发锁存得到的理论位置值

返回值：轴指令理论位置值

例程

```
'完整例程可参考 TrigLTC 指令
DIM B AS DOUBLE
B=LDPOS(AX(1)) '获取轴 1 的锁存理论位置值
```

2.10.3.4 LMPOS

所属：命令

语法：value=LMPOS(AX(no))

类型：DOUBLE

描述：读取触发锁存得到的编码器实际位置值

返回值：轴编码器实际位置值

例程

```
'完整例程可参考 TrigLTC 指令
DIM B AS DOUBLE
B=LMPOS(AX(1))'获取轴 1 的锁存编码器实际位置值
```

2.10.3.5 TrigLTC

所属：命令

语法：TrigLTC AX(axis no)

描述：用软件命令触发产生锁存信号。实际应用中，锁存信号基本上由硬件信号触发，该命令多用于测试。

参数：axis no 轴号；**范围：**根据控制器实际硬件决定。

例程

```
DIM LTC_CmdDATA AS DOUBLE
DIM LTC_FBDATA AS DOUBLE
BASE 1
SVON
LTC_EN=1
MOVE 50000
SLEEP 2000
TrigLTC AX(1)      '触发轴 1 的锁存信号，进行位置锁存
LTC_CmdDATA=LDPOS(AX(1)) '将锁存到的指令理论位置读取出来
PRINT LTC_CmdDATA
LTC_FBDATA=LMPOS(AX(1))      '将锁存到的编码器实际位置读取出来
PRINT LTC_FBDATA
RESETLTC AX(1)               '清除锁存标记，锁存位置值
```

2.10.3.6 LTC_Flag

所属：命令

语法：value=LTC_Flag(AX(no))

类型：ushort

描述：读取轴锁存标志，捕捉到锁存信号时，LTC_Flag 会置 1。要清除锁存标志，

需要用 ResetLTC 指令清除。

返回值：锁存标志信号

例程

```
DIM B AS USHORT
B= LTC_Flag(AX(1)) '获取轴 1 的锁存标志信号
```

2.10.3.7 RESETLTC

所属：命令

语法：RESETLTCAX(axis no)

描述：清除锁存数据、锁存标记。未清除锁存标记，下次接收到锁存信号时，将不进行位置值的锁存。

参数：axis no 轴号；**范围：**根据控制器实际硬件决定。

例程

```
'参考 TrigLTC 指令例程
```

2.10.3.8 LBUF_RESET

所属：命令

语法：LBUF_RESET　AX(axis no)

类型：ULONG

描述：清除锁存缓存中的资料。

参数：axis no　　轴号

范围：根据控制器实际硬件决定

例程

```
'完整例程可参考 LBUF_EN 指令
LBUF_RESET AX(0)    '清除轴 0 锁存数据
```

2.10.3.9 LBUF_EN

所属：属性

语法：LBUF_EN = value

类型：ULONG

描述：启用/禁用 latch buffer 功能，使用此功能，需现将 LTC_EN 打开。

范围：设定值和返回值如下，默认值 0

0：禁用(默认值)

1：启用

例程

```
BASE 3
LBUF_RESET AX(3)
LTC_EN=1
LTC_LOGIC =0
LBUF_EN=1       '启用轴 3latch buffer 功能
LBUF_DIST=1000
LBUF_SRC=0
LBUF_ID=0
LBUF_EVTNUM=5
MOVE 100000
WAIT DONE
```

2.10.3.10 LBUF_DIST

所属：属性

语法：LBUF_DIST = value

类型：ULONG

描述：设置/读取相邻两笔锁存数据的间隔距离，单位为 Pulse.

如果两次相邻锁存数据的间隔长度小于设定值，则第二个 latch 会被忽略。

范围：【0，2147483647】，默认值 1000

例程

```
'完整例程可参考 LBUF_EN 指令
BASE 0
LBUF_DIST  =2000 '设置轴 0 相邻两笔锁存数据的间隔距离为 2000
```

2.10.3.11 LBUF_SRC

所属：属性

语法：LBUF_SRC = value

类型：ULONG

描述：设置/读取 LatchBuffer 比较触发的比较源。

范围：设定值和返回值如下，默认值 0

0：理论位置

1：实际位置

例程

```
BASE 0
LBUF_SRC  =1  '设置轴 0 的位置比较源为实际位置
```

2.10.3.12 LBUF_ID

所属：属性

语法：LBUF_ID = value

类型：ULONG

描述：设置/读取 LatchBuffer 中的数据来自哪一轴的位置，目前只支持板卡 PCI1285-MAS

注意：该 ID 号是基于程序中所 BASE 的轴号来设定的，用法见**例程**

范围：轴号，根据控制器实际硬件决定

例程

```
'完整例程可参考 LBUF_EN 指令
BASE 4,5,6
LBUF_ID=1   '轴 4、5、6 锁存的都是轴 1 的位置数据
```

2.10.3.13 LBUF_EVTNUM

所属：属性

语法：LBUF_EVTNUM= value

类型：ULONG

描述：当锁存的笔数达到 LBUF_EVTNUM 时，产生 LTCBUFDONE 事件

范围：【0，128】，默认值 128

例程

```
'完整例程可参考 LBUF_EN 指令
BASE 0
LBUF_EVTNUM =10 '设置轴 0LatchBuffer 中锁存到 10 个数据时，产生 LTCBUFDONE 事件
```

2.10.3.14 LBUF_STATUS

所属：命令

语法：LBUF_STATUS AX(AxNo), RemCnt, SpaceCnt

描述：获取 latchbuffer 中尚未读取的数据个数以及剩余空间，总空间大小为 128

参数：AxNo：轴号

RemCnt：latchbuffer 中尚未读取的数据个数

SpaceCnt：latchbuffer 中剩余空间大小

例程

```
'请参考 LBUF_DATA 指令
```

2.10.3.15 LBUF_DATA

所属：命令

语法：LBUF_DATA AX(AxNo), DataArray(), DataCnt

描述：从 Latch buffer 中读取指定数量的数据

参数：AxNo：轴号

DataArray()：读取到的 LatchBuffer 中的数据清单，单位：UNIT

DataCnt：指定读取的数据个数

例程

```
DIM DataArray(0 to 30) as Double
DIM AS ULONG RemCnt,SpaceCnt,DataCnt
BASE 4
WAIT LTCBUFDONE '等待 latch buffer 中数据个数大于或等于 LBUF_EVTNUM 的设定时触发事件。
LBUF_STATUS(AX(4),RemCnt,SpaceCnt)
LBUF_DATA(AX(4),DataArray(),30)
PRINT RemCnt
PRINT SpaceCnt
PRINT DataArray(0)
PRINT DataArray(1)
PRINT DataArray(2)
PRINT DataArray(3)
PRINT DataArray(4)
```

2.10.4 单轴比较触发

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.10.4.1	CMP_EN	比较触发功能使能	√	×
2.10.4.2	CMP_LOGIC	比较触发端口逻辑电平	√	√
2.10.4.3	CMP_METHOD	比较触发功能比较方法	√	√
2.10.4.4	CMP_MODE	比较触发的 DO 输出模式	√	√
2.10.4.5	CMP_SRC	比较触发的比较源	√	√
2.10.4.6	CMP_WIDTH	比较触发的 DO 输出模式为脉冲模式时的电平宽度	√	√
2.10.4.7	CPOS	当前比较位置数据	√	×
2.10.4.8	CMP	比较触发模式选择	√	×
2.10.4.9	CMP_FLAG	比较触发信号标志	√	×
2.10.4.10	RESETCMP	清除比较触发信号标志	√	×
2.10.4.11	CMPSETDO	手动控制 CMP_DO	×	√

2.10.4.1 CMP_EN

所属：属性

语法：CMP_EN = value

类型：ULONG

描述：启用/禁用轴比较触发功能，研华运动控制的每个轴都关联着 4 个 DO 端口，

分别称为 OUT4，OUT5，OUT6，OUT7，每个轴的比较触发输出由 OUT5 输出端口产生，启用比较触发功能后，一旦比较触发产生，OUT5 即输出信号。

范围：设定值和返回值如下，默认值 0

0：禁用

1：启用

例程

```
BASE 0
CMP_EN =1 '启用轴 0 比较触发功能
```

2.10.4.2 CMP_LOGIC

所属：属性

语法：CMP_LOGIC = value

类型：ULONG

描述：设置/读取轴比较触发有效输出时的 DO 逻辑电平

范围：设定值和返回值如下，默认值 0

0：低电平

1：高电平

例程

```
BASE 0
CMP_LOGIC  =0 '设置比较触发输出的 DO 逻辑电平为低电平
```

2.10.4.3 CMP_METHOD

所属：属性

语法：CMP_METHOD = value

类型：ULONG

描述：设置/读取轴比较触发的比较方法

范围：设定值和返回值如下，默认值 0

0：>= 位置计数器

1：<= 位置计数器

2：= 计数器（无方向）

例程

```
BASE 0
CMP_METHOD  =1'设置轴 0 的比较方法为小于等于位置计数器
```

2.10.4.4 CMP_MODE

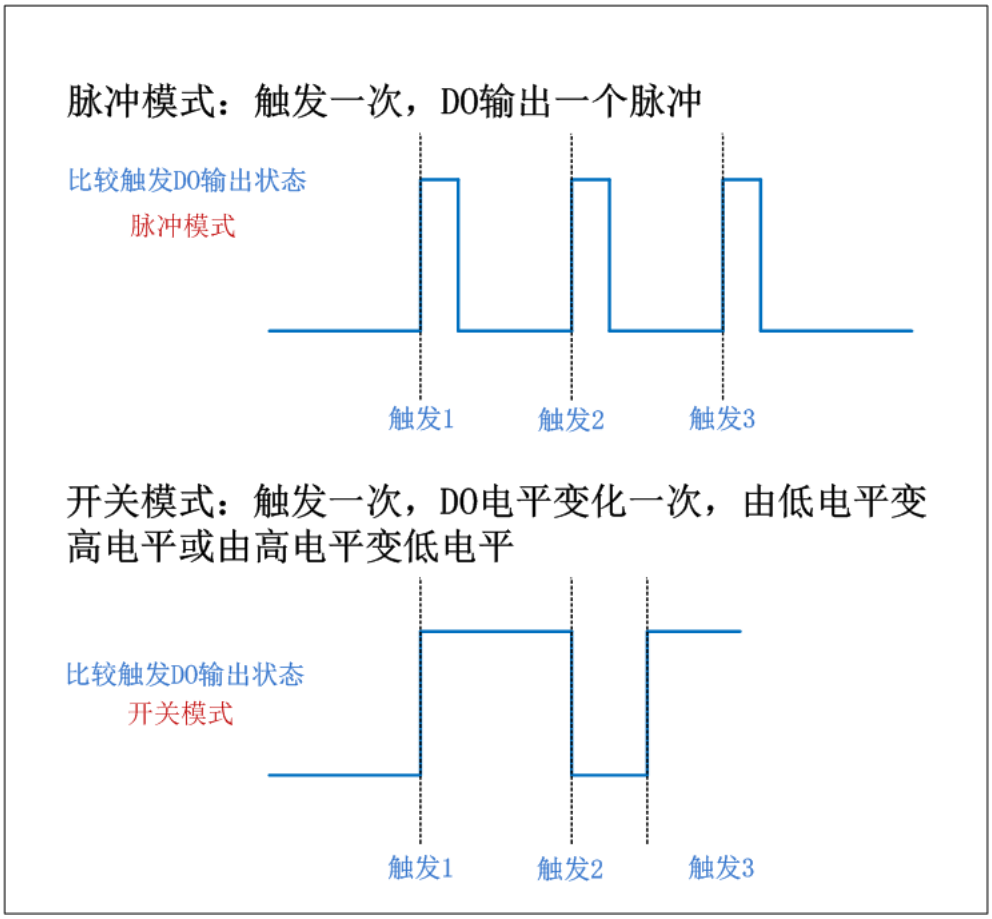
所属：属性

语法：CMP_MODE = value

类型：ULONG

描述：设置/读取轴比较触发的 DO 输出模式，DO 输出模式说明如下图。

比较触发 DO 输出模式说明



范围：设定值和返回值如下，默认值 0

0：脉冲模式

1：开关模式

例程

```
BASE 0
CMP_MODE  =1 '设置轴 0 的比较触发 DO 输出模式为开关模式
```

2.10.4.5 CMP_SRC

所属：属性

语法：CMP_SRC = value

类型：ULONG

描述：设置/读取轴比较触发的比较源

范围：设定值和返回值如下，默认值 0

0：理论位置

1：实际位置

例程

```
BASE 0
CMP_SRC  =1  '设置轴 0 的位置比较源为实际位置
```

2.10.4.6 CMP_WIDTH

所属：属性

语法：CMP_WIDTH = value

类型：ULONG

描述：设置/读取轴比较触发的 DO 输出模式为脉冲模式时的电平宽度，单位为微秒

范围：0~85899000us，默认值 5

例程

```
BASE 0
CMP_WIDTH  =50 '设置轴 0 比较触发的 DO 电平宽度为 50us
```

2.10.4.7 CPOS

所属：命令

语法：value=CPOS(AX(no))

类型：DOUBLE

描述：读取比较触发中的当前比较位置数据

返回值：轴当前设定的比较位置数据值

例程

```
DIM B AS DOUBLE
B=CPOS(AX(1)) '获取轴 1 的当前笔比较位置值
```

2.10.4.8 CMP

所属：命令

语法 1：CMP AX(axis no)， position

语法 2：CMP AX(axis no)，start_position，end_position, interval

语法 3：CMP AX(axis no)，tablearray( ),array_num

描述：比较触发的模式分 3 种。

- ✧ 语法 1 用于单点比较触发，设置一个比较点的比较位置值。
- ✧ 语法 2 用于等距间隔多点比较触发，设置起始比较点位置，终点比较点位置，间隔距离。
- ✧ 语法 3 用于随机点多点比较触发，设置多个要比较的位置值。

参数：axis no 轴号；**范围：**根据控制器实际硬件决定。

Position 比较触发的位置值

start_position 起始比较点位置值，用于等距间隔多点比较模式

end_position 终点比较点位置值，用于等距间隔多点比较模式

interval 间隔距离，用于等距间隔多点比较模式

tablearray（）随机比较点数组，将要比较的点依次赋值到一个数组里，用于随机点多点比较模式

array_num 比较点个数，用于随机点多点比较模式

例程

```
BASE 0
SVON
DIM cmp_table(3) AS double
cmp_table(0)=70000
cmp_table(1)=73400
cmp_table(2)=79424
CMP_EN=1
CMP_METHOD=0 '大于等于位置源时触发
CMP_LOGIC=0 '触发电平为低电平
CMP_SRC=0  '参考源为理论位置
CMP_WIDTH=100 '脉冲模式输出脉宽为 100us
CMP_MODE=0 'DO 输出模式脉冲模式
'单点比较触发
CMP AX(0),10000 '设置比较位置为 10000
MOVEABS 0 '先运动到位置 0
WAIT DONE
MOVEABS 20000 '运到到 10000 的瞬间，比较触发的 DO 输出
WAIT Done
'均匀间隔比较触发
CMP AX(0),30000,50000,4000 '比较位置从 30000 到 50000，每隔 4000，触发一次 DO
MOVEABS 60000
WAIT DONE
'随机表格位置比较触发
CMP AX(0),cmp_table(),3
MOVEABS 80000
```

2.10.4.9 CMP_Flag

所属：命令

语法：value=CMP_Flag(AX(no))

类型：ushort

描述：读取轴比较触发标志，发生轴比较触发时，CMP_Flag 会置 1。要清除锁存标志，需要用 ResetCMP 指令清除。

返回值：比较触发标志信号

例程

```
DIM B AS USHORT
B=CMP_Flag(AX(1)) '获取轴 1 的比较触发标志信号
```

2.10.4.10 RESETCMP

所属：命令

语法：RESETCMP AX(axis no)

描述：清除比较触发标志。

参数：axis no 轴号；**范围：**根据控制器实际硬件决定。

2.10.4.11 CMPSETDO

所属：命令

语法：CMPSETDO AX(id), OnOrOff

类型：ULONG

描述：手动设置轴的 CMP_DO 输出状态

参数：ax(id)：轴号

OnOrOff：打开或关闭 DO，0：关闭，1：打开

例程

```
CMPSETDO AX(0),1  '触发轴 0 上的 CAMDO 输出
```

2.10.5 多轴比较触发

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.10.5.1	MCMP_EN	多轴比较使能	×	×
2.10.5.2	MCMP_CH	多轴比较对应轴的 OUT5 是否输出	×	×
2.10.5.3	MCMP_MODE	多轴比较 DO 输出模式	×	×
2.10.5.4	MCMP_DEVIA	多轴比较误差范围	×	×
2.10.5.5	MCMP_EMPTY	自动清除多轴比较数据	×	×
2.10.5.6	MCMP_LOGIC	多轴比较逻辑电平	×	√
2.10.5.7	MCMP_WIDTH	多轴比较脉冲模式时的脉宽	×	×
2.10.5.8	MCMP_PWMFREQ	多轴比较 PWM 模式时的频率	×	×
2.10.5.9	MCMP_PWMDUTY	多轴比较 PWM 的占空比	×	×
2.10.5.10	MCMPPWMTIMETABLE	多轴比较的 PWM 输出时间	×	×
2.10.5.11	MCMPSETDO	手动控制 MCMP_DO	×	√
2.10.5.12	MCMPFORCEOUT	手动控制 MCMP 输出	×	√
2.10.5.13	MCMP_PWMMAXFREQ	多轴比较 PWM 的最大频率	×	×
2.10.5.14	MCMP_PWMMINFREQ	多轴比较 PWM 的最小频率	×	×
2.10.5.15	MCMP_PWMMAXDUTY	多轴比较 PWM 的最大占空比	×	×
2.10.5.16	MCMP_PWMMINDUTY	多轴比较 PWM 的最小占空比	×	×
2.10.5.17	GPWM_LINKEN	根据群组速度改变 PWM 输出	×	×
2.10.5.18	GPWM_MODE	群组 PWM 更改模式	×	×
2.10.5.19	GPWM_REFVEL	多轴比较 PWM 模式时的基本速度	×	×

2.10.5.1 MCMP_EN

所属：属性

语法：MCMP_EN = value

类型：ULONG

描述：启用/禁用多轴比较功能。需搭配 BASE 使用，以指定同时比较哪些轴的位置（BASE 中的轴需在同一张板卡中）。

范围：设定值和返回值如下，默认值 0

0：禁用(默认值)

1：启用

例程

```
BASE 0,1,2
VH = 400
DPOS=0
'PWM 模式时,需要先设置比较位置再设置其他参数,PWM 模式才有效.
CMP AX(0),2000,14000,3000
CMP AX(1),2000,14000,3000
CMP AX(2),2000,14000,3000
MCMP_EN=1       '使能多轴比较功能
MCMP_MODE=2     '0:Pulse 1:Toggle 2:PWM 3:PWM Toggle
MCMP_LOGIC=1    'PWM 模式下不起作用
MCMP_CH=7
MCMP_WIDTH= 1000000 '单位:um,PWM 和 toggle 模式下不起作用
MCMP_PWMMINFREQ=1
MCMP_PWMMAXFREQ=200 '1-250000
MCMP_PWMMINDUTY=10
MCMP_PWMMAXDUTY=80
MCMP_PWMFREQ=100    'PWM 频率
MCMP_PWMDUTY=50     '占空比
GPWM_LINKEN=1       '启用根据群组速度改变 PWM 输出功能
GPWM_MODE=0         '0：频率   1：占空比
GPWM_REFVEL=1000
MCMP_EMPTY = 1
MCMP_DEVIA=10
CMP_METHOD=2
CMP_SRC=1
DIM onTimeArray(5) as ULONG
onTimeArray(0)=1000
onTimeArray(1)=500
onTimeArray(2)=1000
onTimeArray(3)=5000
onTimeArray(4)=2000
MCMPPWMTIMETABLE onTimeArray(),3
MOVEABS 15000,15000,15000,15000
WAIT DONE
```

2.10.5.2 MCMP_CH

所属：属性

语法：按位输出, Bit0~3 分别对应轴 0~3, 如下枚举

value=0，禁用 OUT5 输出比较信号

value=1，轴 0 的 OUT5 输出比较信号

value=2，轴 1 的 OUT5 输出比较信号

value=3，轴 0, 1 的 OUT5 输出比较信号

value=4，轴 2 的 OUT5 输出比较信号

value=5，轴 0, 2 的 OUT5 输出比较信号

value=6，轴 1, 2 的 OUT5 输出比较信号

value=7，轴 0, 1, 2 的 OUT5 输出比较信号

value=8，轴 3 的 OUT5 输出比较信号

value=9，轴 0,3 的 OUT5 输出比较信号

value=10，轴 1,3 的 OUT5 输出比较信号

value=11，轴 0,1,3 的 OUT5 输出比较信号

value=12，轴 2,3 的 OUT5 输出比较信号

value=13，轴 0,2,3 的 OUT5 输出比较信号

value=14，轴 1,2,3 的 OUT5 输出比较信号

value=15，轴 0,1,2,3 的 OUT5 输出比较信号

类型：ULONG

描述：启用/禁用多轴比较对应轴的 OUT5 输出

2.10.5.3 MCMP_MODE

所属：属性

语法：MCMP_MODE = value

类型：ULONG

描述：设置/读取多轴比较触发的 DO 输出模式

范围：设定值和返回值如下，默认值 0

0：脉冲模式

1：开关模式

2：PWM 模式

3：PWM-TOGGLE 模式

例程

```
BASE 0,1
MCMP_MODE  =1 '设置轴 0,1 的多轴比较 DO 输出模式为开关模式
```

2.10.5.4 MCMP_DEVIA

所属：属性

语法：MCMP_DEVIA = value

类型：ULONG

描述：设置/读取多轴比较误差范围，单位为脉冲

范围：设定值和返回值为【0，65535】，默认值 0

例程

```
BASE 0,1
MCMP_DEVIA=100  '设置多轴比较误差范围
```

2.10.5.5 MCMP_EMPTY

所属：属性

语法：MCMP_EMPTY = value

类型：ULONG

描述：启用/禁用自动清除多轴比较数据

范围：设定值和返回值如下，默认值 0

0：禁用(默认值)

1：启用

例程

```
BASE 0,1
MCMP_EN=1    '比较结束后自动清除多轴比较数据
```

2.10.5.6 MCMP_LOGIC

所属：属性

语法：MCMP_LOGIC = value

类型：ULONG

描述：设置/读取多轴比较逻辑电平

范围：设定值和返回值如下，默认值 0

0：低电平

1：高电平

例程

```
BASE 0,1
MCMP_LOGIC=1 '设置多轴比较的输出电平为高电平
```

2.10.5.7 MCMP_WIDTH

所属：属性

语法：MCMP_WIDTH = value

类型：ULONG

描述：设置/读取多轴比较为脉冲模式时的脉冲宽度，单位为微秒

范围：【0，85899000】，默认值 1

例程

```
BASE 0,1
MCMP_WIDTH  =1000000 '设置多轴比较脉冲宽度为 1000000us
```

2.10.5.8 MCMP_PWMFREQ

所属：属性

语法：MCMP_PWMDUTY = value

类型：ULONG

描述：设置/读取多轴比较为 PWM 模式时的频率

范围：【1Hz – 250,000Hz】，默认值 100000Hz。

例程

```
BASE 0,1
MCMP_PWMFREQ  =10000 '设置多轴比较 PWM 频率为 10000
```

2.10.5.9 MCMP_PWMDUTY

所属：属性

语法：MCMP_PWMDUTY = value

描述：设置/读取多轴比较为 PWM 模式时的占空比

范围：【0，100】，默认值 0

例程

```
BASE 0,1
MCMP_PWMDUTY  =50 '设置多轴比较 PWM 占空比为 50
```

2.10.5.10 MCMPPWMTIMETABLE

所属：命令

语法：MCMPPWMTIMETABLEOnTimeArray(), Cnt

描述：设置多轴比较 PWM 与 PWM-Toggle 模式时的 PWM 输出时间

参数：OnTimeArray()每个比较位置启动后的持续时间(单位：us)一次最多设定 30000 笔。

Cnt：有效数据的个数。

注意：

1. 当所设置的比较位置的个数大于所设置的比较时间个数时,剩余比较位置的 PWM 输出时间会沿用上个比较位置设置的输出时间.

2.需要先设置比较位置再设置时间表才有效。

范围：设定值和返回值为【-2147483647，2147483647】，默认值 20000

例程

```
BASE 0,1
DIM onTimeArray(5) as ULONG
onTimeArray(0)=1000
onTimeArray(1)=5000
onTimeArray(2)=1000
onTimeArray(3)=5000
onTimeArray(4)=2000
MCMPPWMTIMETABLE onTimeArray(),3
'设置 5 笔 PWM 输出时间，其中前 3 个有效
```

2.10.5.11 MCMPSETDO

所属：命令

语法：MCMPSETDO OnOrOff

类型：ULONG

描述：手动设置多轴比较各轴的 CMP_DO 输出状态

参数：OnOrOff：打开或关闭 DO，0：关闭，1：打开

例程

```
BASE 0,1,2
MCMPSETDO (1)  '触发轴 0,1,2 上的 CAMDO 输出
```

2.10.5.12 MCMPFORCEOUT

所属：命令

语法：MCMPFORCEOUT

类型：ULONG

描述：手动触发多轴比较输出，输出方式以 MCMP_MODE 设定决定。

例程

```
BASE 0,1,2
MCMP_EN=1
MCMP_MODE=0
MCMP_LOGIC=1
MCMP_WIDTH= 1000000
MCMPFORCEOUT '手动控制多轴比较输出
```

2.10.5.13 MCMP_PWMMAXFREQ

所属：属性

语法：MCMP_PWMMAXFREQ = value

类型：ULONG

描述：设置/读取多轴比较为 PWM 模式时的最大频率，单位为赫兹

范围：【1，250,000】

例程

```
BASE 0,1
MCMP_PWMMAXFREQ =200 '设置多轴比较 PWM 最大频率为 200Hz
```

2.10.5.14 MCMP_PWMMINFREQ

所属：属性

语法：MCMP_PWMMINFREQ = value

类型：ULONG

描述：设置/读取多轴比较为 PWM 模式时的最小频率，单位为赫兹

范围：【1，250,000】

例程

```
BASE 0,1
MCMP_PWMMINFREQ =10 '设置多轴比较 PWM 频率为 10Hz
```

2.10.5.15 MCMP_PWMMAXDUTY

所属：属性

语法：MCMP_PWMMAXDUTY = value

类型：ULONG

描述：设置/读取多轴比较为 PWM 模式时的最大占空比

范围：【0，100】，默认值 0

例程

```
BASE 0,1
MCMP_PWMMAXDUTY =80 '设置多轴比较 PWM 最大占空比为 80
```

2.10.5.16 MCMP_PWMMINDUTY

所属：属性

语法：MCMP_PWMMINDUTY = value

类型：ULONG

描述：设置/读取多轴比较为 PWM 模式时的最小占空比

范围：【0，100】，默认值 0

例程

```
BASE 0,1
MCMP_PWMMINDUTY =20 '设置多轴比较 PWM 最小占空比为 20
```

2.10.5.17 GPWM_LINKEN

所属：属性

语法：GPWM_LINKEN= value

类型：ULONG

描述：启用/禁用根据群组速度改变 PWM 输出功能

范围：设定值和返回值如下，默认值 0

0：禁用(默认值)

1：启用

例程

```
BASE 0,1
GPWM_LINKEN=1  '启用根据群组速度改变 PWM 输出功能
```

2.10.5.18 GPWM_MODE

所属：属性

语法：GPWM_MODE = value

类型：ULONG

描述：设置/读取根据群组速度改变依照何种模式更改 PWM 输出

范围：设定值和返回值如下

0：频率

1：占空比

例程

```
BASE 0,1
GPWM_MODE  =0 '最终频率会根据设置的基准速度与原先设置的频率计算得到
```

2.10.5.19 GPWM_REFVEL

所属：属性

语法：GPWM_REFVEL = value

类型：ULONG

描述：设置/读取多轴比较为 PWM 模式时的基本速度，以计算 PWM 的改变因子

例如如果设定模式是修改 PWM 频率，则当前 PwmFreq=(当前群组速度/客户设置基准速度)* MCMP_PWMFREQ，如果 PwmFreq 超过最大 MCMP_PWMMAXFREQ，则使用 MCMP_PWMMAXFREQ 作为当前频率，如果 PwmFreq 低于最小 MCMP_PWMMINFREQ，则使用 MCMP_PWMMINFREQ 作为当前频率。

例程

```
BASE 0,1
GPWM_REFVEL  =1000 '设置多轴比较 PWM 模式时的基本速度为 1000
```

2.10.6 研华 DAQ 系列卡的 I/O 控制

2.10.6.1 DAQ

当 MAS 控制器中需要插入研华 DAQ 系列的 I/O 卡时，如果要使用 Motion Studio 来编程控制 DAQ 系列卡时。请参考“模块类”章节的 DAQ 类说明。

2.11 回原点与限位

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.11.1	HOME_VL	原点运动初速度	√	√
2.11.2	HOME_VH	原点运动运行速度	√	√
2.11.3	HOME_ACC	原点运动加速度	√	√
2.11.4	HOME_DEC	原点运动减速度	√	√
2.11.5	HOME_JK	原点运动速度曲线类型	√	√
2.11.6	HOME_MODE	原点运动模式	√	√
2.11.7	HOMEP	正向原点运动	√	×
2.11.8	HOMEN	反向原点运动	√	×
2.11.9	HOME_CROSS	原点运动跨越距离	√	√
2.11.10	HOME_OFFSETDIST	原点运动完成后再移动的偏移距离	√	√
2.11.11	HOME_OFFSETVEL	原点运动完成后移动偏移距离的速度	√	√
2.11.12	HOME_RESET	原点运动后清零位置值功能	√	×
2.11.13	ORG_LOGIC	原点端口逻辑电平	√	√
2.11.14	ORG_MODE	原点运动结束时的停止模式	√	√
2.11.15	ORG_FILTER	原点端口的滤波	√	√
2.11.16	EZ_LOGIC	Z 相端口逻辑电平	√	√
2.11.17	EL_EN	硬限位功能使能	√	×
2.11.18	EL_LOGIC	硬限位端口逻辑电平	√	√
2.11.19	EL_MODE	硬限位元触发时的停止模式	√	√
2.11.20	PEL_FILTER	正方向硬限位端口滤波	√	√
2.11.21	NEL_FILTER	负方向硬限位端口滤波	√	√
2.11.22	SPEL	正方向软限位值	√	√

2.11.23	SPEL_EN	正方向软限位功能使能	√	×
2.11.24	SPEL_MODE	正方向软限位触发时的停止模式	√	√
2.11.25	SNEL	负方向软限位值	√	√
2.11.26	SNEL_EN	负方向软限位功能使能	√	×
2.11.27	SNEL_MODE	负方向软限位触发时的停止模式	√	√
2.11.28	PEL_TOL_EN	正方向硬极限容差功能使能	√	×
2.11.29	PEL_TOL	正方向硬极限容差值	√	√
2.11.30	NEL_TOL_EN	负方向硬极限容差功能使能	√	×
2.11.31	NEL_TOL	负方向硬极限容差值	√	√
2.11.32	SPEL_TOL_EN	正方向软极限容差功能使能	√	×
2.11.33	SPEL_TOL	正方向软极限容差值	√	√
2.11.34	SNEL_TOL_EN	负方向软极限容差功能使能	√	×
2.11.35	SNEL_TOL	负方向软极限容差值	√	√

2.11.1 HOME_VL

所属：属性

语法：HOME_VL = value

类型：DOUBLE

描述：设置/读取回原点运动的初速度，单位元为 UNIT/s

范围：（0,MAXVEL），默认值 2000

例程

```
BASE 0
HOME_VL=1000 '设定轴 0 回原点运动的初速度为 1000 个 UNIT/s
```

2.11.2 HOME_VH

所属：属性

语法：HOME_VH = value

类型：DOUBLE

描述：设置/读取回原点运动的最大运行速度，单位元为 UNIT/s

范围：（HOME_VL,MAXVEL），默认值 8000

例程

```
BASE 0
HOME_VH=5000 '设定轴 0 回原点运动的运行速度为 5000 个 UNIT/s
```

2.11.3 HOME_ACC

所属：属性

语法：HOME_ACC = value

类型：DOUBLE

描述：设置/读取回原点运动的加速度，单位为 UNIT/s^2

范围：（0,MAXACC），默认值 10000

例程

```
BASE 0
HOME_ACC=20000 '设置轴 0 回原点的加速度为 20000 个 UNIT/s^2
```

2.11.4 HOME_DEC

所属：属性

语法：HOME_DEC = value

类型：DOUBLE

描述：设置/读取回原点运动的减速度，单位元为 UNIT/s^2

范围：（0,MAXDEC），默认值 10000

例程

```
BASE 0
HOME_DEC=20000 '设置轴 0 回原点的减速度为 20000 个 UNIT/s^2
```

2.11.5 HOME_JK

所属：属性

语法：HOME_JK = value

类型：ULONG

描述：设置/读回原点运动的速度曲线类型

范围：【0,1】，0：T 型曲线；1：S 型曲线，默认值 0

例程

```
BASE 0
HOME_JK=1  '设置轴 0 回原点的速度曲线为 S 型曲线
```

2.11.6 HOME_MODE

所属：属性

语法：HOME_MODE = value

类型：ULONG

描述：设置/读取回原点运动的模式

范围：设定值和返回值如下，默认值 0

0：MODE1_Abs

1：MODE2_Lmt

2：MODE3_Ref

3：MODE4_Abs_Ref

4：MODE5_Abs_NegRef

5：MODE6_Lmt_Ref

6：MODE7_AbsSearch

7：MODE8_LmtSearch

8：MODE9_AbsSearch_Ref

9：MODE10_AbsSearch_NegRef

10：MODE11_LmtSearch_Ref

11：MODE12_AbsSearchReFind

12：MODE13_LmtSearchReFind

13：MODE14_AbsSearchReFind_Ref

14：MODE15_AbsSearchReFind_NegR

15：MODE16_LmtSearchReFind_Ref

101~137：CiA402_MODE1 ~ CiA402_MODE37（EtherCAT 伺服用）

例程

```
BASE 0
HOME_MODE=7 '设定轴 0 的回原点模式未 MODE8_LmtSearch
```

2.11.7 HOMEP

所属：命令

语法 1：HOMEP

语法 2：HOMEP AX(axis no)

语法 3：HOMEP dir1[，dir2][，dir3]……

描述：BASE 轴列表的轴或指定轴、方向，开始正向回原点运动。HOMEP 分 3 种方法使用如下。

- ✧ 语法 1 用于对 BASE 轴列表的轴执行正向回原点运动
- ✧ 语法 2 用于指定某个轴执行正向回原点运动
- ✧ 语法 3 用于对 BASE 轴列表的轴，指定不同方向，执行回原点运动。Dir 为 0 时，方向与 HOMEP 同向；dir 为 1 时，方向与 HOMEP 反向

参数：dir0：正向；1：反向

axis no 轴号；**范围：**根据控制器实际硬件决定。

相关指令参考：HOME_MODE；HOME_CROSS；HOME_RESET

例程

```
BASE 0,1,2
HOME_VL=500
HOME_VH=10000
HOME_ACC=50000
HOME_DEC=50000
HOME_CROSS=2000  '设置原点运动中跨越距离
HOME_RESET=1      '原点运动结束后清零理论位置、实际位置
HOME_MODE=6      '6：MODE7_AbsSearch
HOMEP AX(1)    '指定轴 1 执行正向回原点运动
WAIT AX(1),DONE   '等待轴 1 运动停止
BASE 0,1        '基于轴 0,轴 1
HOMEP           '轴 0，轴 1 都执行正向回原点运动
WAIT DONE       '等待两个轴的回原点运动停止
HOMEN    '轴 0，轴 1 都执行负向回原点运动
WAIT DONE
HOMEP 0,1           '轴 0 执行正向回原点运动，轴 1 执行反向回原点运动
WAIT DONE
MOVE 5000,5000   '原点运动停止后，执行两轴相对点位运动
```

2.11.8 HOMEN

所属：命令

语法 1：HOMEN

语法 2：HOMEN AX(axis no)

语法 3：HOMEN dir1[，dir2][，dir3]……

描述：BASE 轴列表的轴或指定轴、方向，开始负向回原点运动。HOMEN 分 3 种方法使用如下。

- ✧ 语法 1 用于对 BASE 轴列表的轴执行负向回原点运动
- ✧ 语法 2 用于指定某个轴执行负向回原点运动
- ✧ 语法 3 用于对 BASE 轴列表的轴，指定不同方向，执行回原点运动。Dir 为 0 时，方向与 HOMEN 同向；dir 为 1 时，方向与 HOMEN 反向

参数：dir　0：反向；1：正向

axis no 轴号；**范围：**根据控制器实际硬件决定。

相关指令参考：HOME_MODE；HOME_CROSS；HOME_RESET

例程

```
BASE 0,1,2
HOME_VL=500
HOME_VH=10000
HOME_ACC=50000
HOME_DEC=50000
HOME_CROSS=2000  '设置原点运动中跨越距离
HOME_RESET=1      '原点运动结束后清零理论位置、实际位置
HOME_MODE=6      '6：MODE7_AbsSearch
HOMEP AX(1)    '指定轴 1 执行正向回原点运动
WAIT AX(1),DONE   '等待轴 1 运动停止
BASE 0,1        '基于轴 0,轴 1
HOMEP           '轴 0，轴 1 都执行正向回原点运动
WAIT DONE       '等待两个轴的回原点运动停止
HOMEN    '轴 0，轴 1 都执行负向回原点运动
WAIT DONE
HOMEP 0,1           '轴 0 执行正向回原点运动，轴 1 执行反向回原点运动
WAIT DONE
MOVE 5000,5000   '原点运动停止后，执行两轴相对点位运动
```

2.11.9 HOME_CROSS

所属：属性

语法：HOME_CROSS= value

类型：DOUBLE

描述：设置/读取回原点运动时的跨越距离。HOME_MODE 里有几种模式会用到 HOME_CROSS，请参考 HOME_MODE 指令说明。

例程

```
BASE 0
HOME_CROSS=100 '设定轴 0 回原点运动时的跨越距离为 100 个 UNIT
```

2.11.10 HOME_OFFSETDIST

所属：属性

语法：HOME_OFFSETDIST = value

类型：DOUBLE

描述：设置/读取回原点运动完成后再移动的偏移距离。

例程

```
BASE 0
HOME_OFFSETDIST =1000 '设置轴 0 的回原点偏移距离为 1000 个 UNIT
```

2.11.11 HOME_OFFSETVEL

所属：属性

语法：HOME_OFFSETVEL = value

类型：DOUBLE

描述：设置/读取回原点运动完成后移动偏移距离的速度

范围：（0,MAXVEL），默认值 8000

例程

```
BASE 0
HOME_OFFSETVEL =1000 '设置轴 0 的回原点偏移速度为 1000 个 UNIT/s
```

2.11.12 HOME_RESET

所属：属性

语法：HOME_RESET= value

类型：ULONG

描述：启用或禁用回原点后清零位置值功能

范围：设定值和返回值如下，默认值 1

0：禁用

1：启用

例程

```
BASE 0
HOME_RESET=1  '启用轴 0 回完原点后清零位置值功能
```

2.11.13 ORG_LOGIC

所属：属性

语法：ORG_LOGIC = value

类型：ULONG

描述：设置/读取 ORG 信号的有效逻辑电平。ORG 专用数字量输入端口用于回原点运动中的几种用到 ORG 信号的模式，请参考 HOME_MODE 指令说明。

范围：设定值和返回值如下，默认值 0

0：低电平

1：高电平

例程

```
BASE 0
ORG_LOGIC =1  '设置轴 0 的 ORG 输入信号高电平有效
```

2.11.14 ORG_MODE

所属：属性

语法：ORG_MODE = value

类型：ULONG

描述：设置/读取回原点运动结束时的停止模式。

范围：设定值和返回值如下，默认值 1

0：立即停止

1：减速停止

例程

```
BASE 0
ORG_MODE =1  '设置轴 0 的回原点运动停止模式未减速停止模式
```

2.11.15 ORG_FILTER

所属：属性

语法：ORG_FILTER = value

类型：ULONG

描述：设置/读取轴的 ORG 输入信号的滤波时间

范围：设定值和返回值如下，默认值 0

0：5us

1：100us

2：200us

3：500us

例程

```
BASE 0
ORG_FILTER =1  '设置 ORG 信号的滤波时间为 100us
```

2.11.16 EZ_LOGIC

所属：属性

语法：EZ_LOGIC = value

类型：ULONG

描述：设置/读取电机编码器 Z 相输入信号的有效逻辑电平。运动控制中，Z 相信号常常用于回原点运动中，请参考 HOME_MODE 的指令说明。

范围：设定值和返回值如下，默认值 0

0：低电平

1：高电平

例程

```
BASE 0
EZ_LOGIC =0  '设置轴 0 的 Z 相输入信号低电平有效
```

2.11.17 EL_EN

所属：属性

语法：EL_EN = value

类型：ULONG

描述：启用/禁用硬件限位功能，启用后限位开关被触发，相应方向上运动的电机会被控制停下来

范围：设定值和返回值如下，默认值 1

0：禁用

1：启用

例程

```
BASE 0
EL_EN =1 '启用硬件限位功能
```

2.11.18 EL_LOGIC

所属：属性

语法：EL_LOGIC = value

类型：ULONG

描述：设置/读取硬件限位输入信号的有效逻辑电平

范围：设定值和返回值如下，默认值 0

0：低电平

1：高电平

例程

```
BASE 0
EL_LOGIC =1 '设置轴 0 的硬极限输入信号高电平有效
```

2.11.19 EL_MODE

所属：属性

语法：EL_MODE = value

类型：ULONG

描述：设置/读取接收硬件限位信号时电机的停止模式

范围：设定值和返回值如下，默认值 0

0：立即停止

1：减速停止

例程

```
BASE 0
EL_MODE =0 '设置轴 0 碰到硬极限时电机立即停止
```

2.11.20 PEL_FILTER

所属：属性

语法：PEL_FILTER = value

类型：ULONG

描述：设置/读取轴的正方向硬限位输入信号的滤波时间

范围：设定值和返回值如下，默认值 0

0：5us

1：100us

2：200us

3：500us

例程

```
BASE 0
PEL_FILTER =1;设置轴 0 的正方向硬限位信号滤波时间为 100us
```

2.11.21 NEL_FILTER

所属：属性

语法：NEL_FILTER = value

类型：ULONG

描述：设置/读取轴的负方向硬限位输入信号的滤波时间

范围：设定值和返回值如下，默认值 0

0：5us

1：100us

2：200us

3：500us

例程

```
BASE 0
NEL_FILTER =1;设置轴 0 的负方向硬限位信号滤波时间为 100us
```

2.11.22 SPEL

所属：属性

语法：SPEL = value

类型：LONG

描述：设置/读取正方向软限位的值，单位为脉冲

例程

```
BASE 0
SPEL =100 '设置正方向软件限位的值为 100
```

2.11.23 SPEL_EN

所属：属性

语法：SPEL_EN = value

类型：ULONG

描述：启用/禁用正方向软限位功能，启用正方向软限位功能后，正向移动的电机指令位置到达 SPEL 设定的值后，马达会被控制停止运动。

范围：设定值和返回值如下，默认值 0

0：禁用

1：启用

例程

```
BASE 0
SPEL_EN =1  '启用轴 0 的正方向软限位功能
```

2.11.24 SPEL_MODE

所属：属性

语法：SPEL_MODE = value

类型：ULONG

描述：设置/读取正方向软件限位功能被触发时电机被控制停止的模式

范围：设定值和返回值如下，默认值 1

0：立即停止

1：减速停止

例程

```
BASE 0
SPEL_MODE =1  '设置轴 0 的正方向软件限位被触发时，电机被控制的停止模式为减速停止
```

2.11.25 SNEL

所属：属性

语法：SNEL = value

类型：LONG

描述：设置/读取负方向软限位的值，单位为脉冲

例程

```
BASE 0
SNEL =100 '设置负方向软件限位的值为 100
```

2.11.26 SNEL_EN

所属：属性

语法：SNEL_EN = value

类型：ULONG

描述：启用/禁用负方向软限位功能，启用负方向软限位功能后，负向移动的电机指令位置到达 SNEL 设定的值后，马达会被控制停止运动。

范围：设定值和返回值如下，默认值 0

0：禁用

1：启用

例程

```
BASE 0
SNEL_EN =1  '启用轴 0 的负方向软限位功能
```

2.11.27 SNEL_MODE

所属：属性

语法：SNEL_MODE = value

类型：ULONG

描述：设置/读取负方向软件限位功能被触发时电机被控制停止的模式

范围：设定值和返回值如下，默认值 1

0：立即停止

1：减速停止

例程

```
BASE 0
SNEL_MODE =1  '设置轴 0 的负方向软件限位被触发时，电机被控制的停止模式为减速停止
```

2.11.28 PEL_TOL_EN

所属：属性

语法：PEL_TOL_EN = value

类型：ULONG

描述：启用/禁用正方向硬极限容差功能。该功能仅在外部手轮操作时使用。当手轮控制电机运动时，碰到极限信号后，由于极限会限制某方向的运动，而且触碰极限会发生轴错误报警，导致手轮不能正常控制电机移出极限。该指令功能开启后，会允许在极限附近的某段范围，触碰极限不产生轴错误报警，使得手轮可以正常控制电机。

范围：设定值和返回值如下，默认值 0

0：禁用

1：启用

例程

```
BASE 0
PEL_TOL_EN =1 '启用正方向硬极限容差功能
```

2.11.29 PEL_TOL

所属：属性

语法：PEL_TOL = value

类型：ULONG

描述：设置/读取轴的正方向硬极限容差值。

范围：设定值和返回值为【0，2147483647】，默认值 5000

例程

```
BASE 0
PEL_TOL =100 '设置轴 0 的正方向硬极限容差值为 100 个脉冲
```

2.11.30 NEL_TOL_EN

所属：属性

语法：NEL_TOL_EN = value

类型：ULONG

描述：启用/禁用负方向硬极限容差功能。该功能仅在外部手轮操作时使用。当手轮控制电机运动时，碰到极限信号后，由于极限会限制某方向的运动，而且触碰极限会发生轴错误报警，导致手轮不能正常控制电机移出极限。该指令功能开启后，会允许在极限附近的某段范围，触碰极限不产生轴错误报警，使得手轮可以正常控制电机。

范围：设定值和返回值如下，默认值 0

0：禁用

1：启用

例程

```
BASE 0
NEL_TOL_EN =1 '启用负方向硬极限容差功能
```

2.11.31 NEL_TOL

所属：属性

语法：NEL_TOL = value

类型：ULONG

描述：设置/读取轴的负方向硬极限容差值。

范围：设定值和返回值为【0，2147483647】，默认值 5000

例程

```
BASE 0
NEL_TOL =100 '设置轴 0 的负方向硬极限容差值为 100 个脉冲
```

2.11.32 SPEL_TOL_EN

所属：属性

语法：SPEL_TOL_EN = value

类型：ULONG

描述：启用/禁用正方向软极限容差功能。该功能仅在外部手轮操作时使用。当手轮控制电机运动时，碰到极限信号后，由于极限会限制某方向的运动，而且触碰极限会发生轴错误报警，导致手轮不能正常控制电机移出极限。该指令功能开启后，会允许在极限附近的某段范围，触碰极限不产生轴错误报警，使得手轮可以正常控制电机。

范围：设定值和返回值如下，默认值 0

0：禁用

1：启用

例程

```
BASE 0
SPEL_TOL_EN =1 '启用正方向软极限容差功能
```

2.11.33 SPEL_TOL

所属：属性

语法：SPEL_TOL = value

类型：ULONG

描述：设置/读取轴的正方向软极限容差值。

范围：设定值和返回值为【0，2147483647】，默认值 5000

例程

```
BASE 0
SPEL_TOL =100 '设置轴 0 的正方向软极限容差值为 100 个脉冲
```

2.11.34 SNEL_TOL_EN

所属：属性

语法：SNEL_TOL_EN = value

类型：ULONG

描述：启用/禁用负方向软极限容差功能。该功能仅在外部手轮操作时使用。当手轮控制电机运动时，碰到极限信号后，由于极限会限制某方向的运动，而且触碰极限会发生轴错误报警，导致手轮不能正常控制电机移出极限。该指令功能开启后，会允许在极限附近的某段范围，触碰极限不产生轴错误报警，使得手轮可以正常控制电机。

范围：设定值和返回值如下，默认值 0

0：禁用

1：启用

例程

```
BASE 0
SNEL_TOL_EN =1 '启用负方向软极限容差功能
```

2.11.35 SNEL_TOL

所属：属性

语法：SNEL_TOL = value

类型：ULONG

描述：设置/读取轴的负方向软极限容差值。

范围 :设定值和返回值为【0，2147483647】，默认值 5000

例程

BASE 0

SNEL_TOL =100 '设置轴 0 的负方向软极限容差值为 100 个脉冲

2.12 JOG 与手轮

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.12.1	JOG_VL	JOG 运动低速段速度	√	√
2.12.2	JOG_VH	JOG 运动高速段速度	√	√
2.12.3	JOG_ACC	JOG 运动加速度	√	√
2.12.4	JOG_DEC	JOG 运动减速度	√	√
2.12.5	JOG_VLTIME	JOG 运动低段速度运行的时间	√	√
2.12.6	JOGP	正向软件 JOG 运动	√	×
2.12.7	JOGN	负向软件 JOG 运动	√	×
2.12.8	JOGON	使能外部驱动的 JOG 功能	√	×
2.12.9	JOGOFF	禁用外部驱动的 JOG 功能	√	×
2.12.10	MPGON	使能外部驱动的手轮功能	√	×
2.12.11	MPGOFF	禁用外部驱动的手轮功能	√	×
2.12.12	EXT_MODE	手轮模式外部驱动的脉冲输入模式	√	√
2.12.13	EXT_PULSE	手轮模式外部驱动时，每个手轮脉冲输入对应多少个指令脉冲输出值	√	√
2.12.14	EXT_SRC	外部驱动的信号接入哪个轴的外部驱动输入端口	√	√

2.12.1 JOG_VL

所属：属性

语法：JOG_VL = value

类型：DOUBLE

描述：设置/读取 JOG 运动的低速段速度，单位元为 UNIT/s。当 JOG_VLTIME 值不为 0 时，JOG_VL 将起作用。

范围：（0,MAXVEL），默认值 2000

例程

```
BASE 0
JOG_VL=1000  '设置轴 0 的 JOG 运动低速段速度为 1000 个 UNIT/s
```

2.12.2 JOG_VH

所属：属性

语法：JOG_VH = value

类型：DOUBLE

描述：设置/读取 JOG 运动的高速段速度，单位元为 UNIT/s

范围：（JOG_VL,MAXVEL），默认值 8000

例程

```
BASE 0
JOG_VH=1000  '设置轴 0 的 JOG 运动高速段速度为 1000 个 UNIT/s
```

2.12.3 JOG_ACC

所属：属性

语法：JOG_ACC = value

类型：DOUBLE

描述：设置/读取 JOG 运动的加速度，单位为 UNIT/s^2

范围：（0,MAXACC），默认值 10000

例程

```
BASE 0
JOG_ACC=20000 '设置轴 0 的 JOG 运动加速度为 20000 个 UNIT/s^2
```

2.12.4 JOG_DEC

所属：属性

语法：JOG_DEC = value

类型：DOUBLE

描述：设置/读取 JOG 运动的减速度，单位元为 UNIT/s^2

范围：（0,MAXDEC），默认值 10000

例程

```
BASE 0
JOG_DEC=20000 '设置轴 0 的 JOG 运动减速度为 20000 个 UNIT/s^2
```

2.12.5 JOG_VLTIME

所属：属性

语法：JOG_VLTIME =value

类型：ULONG

描述：设置/读取 JOG 运动低段速度运行的时间，单位为 ms。研华规划的 JOG 运动分两段速度。JOG 指令下达后，先控制电机的速度为 JOG_VL，JOG_VL 运动 JOG_VLTIME 值的时间后，控制电机加速到 JOG_VH。如果 JOG_VLTIME 值设置为 0，JOG 指令下达后，直接控制电机加速到 JOG_VH，JOG_VL 将不起作用。

范围：大于等于 0，默认值 5000

例程

```
BASE 0
JOG_VLTIME=1000 '设置轴 0 的 JOG 运动高低速切换时间为 1000ms
```

2.12.6 JOGP

所属：命令

语法 1：JOGP

语法 2：JOGP AX(axis no)

语法 3：JOGP dir1[，dir2][，dir3]……

描述：BASE 轴列表的轴或指定轴、方向，开始正向 JOG 运动。JOGP 分 3 种方法使用如下。

- ✧ 语法 1 用于对 BASE 轴列表的轴执行正向 JOG 运动
- ✧ 语法 2 用于指定某个轴执行正向 JOG 运动
- ✧ 语法 3 用于对 BASE 轴列表的轴，指定不同方向，执行 JOG 运动。Dir 为 0 时，方向与 JOGP 同向；dir 为 1 时，方向与 JOGP 反向

参数：dir 0：正向；1：反向

axis no 轴号；**范围：**根据控制器实际硬件决定。

例程

```
BASE 0,1,2
JOG_VL=500
JOG_VH=10000
JOG_ACC=50000
JOG_DEC=50000
JOG_VLTIME=2000    '设定低段速度运行的时间为 2 秒
SLEEP 5000
STOPDEC
JOGP     '轴 0、1、2 都执行正向 JOG 运动
SLEEP 3000
STOPDECJOGP AX(1)     '指定轴 1 执行正向 JOG 运动，轴速度会先加速到 JOG_VL 运行 2 秒，再加速到
JOG_VH

WAIT DONE
JOGP 0,1,0'轴 0,2 执行正向 JOG 运动，轴 1 执行负向 JOG 运动
SLEEP 4000
STOPDEC
```

2.12.7 JOGN

所属：命令

语法 1：JOGN

语法 2：JOGN AX(axis no)

语法 3：JOGN dir1[，dir2][，dir3]……

描述：BASE 轴列表的轴或指定轴、方向，开始负向 JOG 运动。JOGN 分 3 种方法使用如下。

- ✧ 语法 1 用于对 BASE 轴列表的轴执行负向 JOG 运动
- ✧ 语法 2 用于指定某个轴执行负向 JOG 运动
- ✧ 语法 3 用于对 BASE 轴列表的轴，指定不同方向，执行 JOG 运动。Dir 为 0 时，方向与 JOGN 同向；dir 为 1 时，方向与 JOGN 反向

参数：dir 0：反向；1：正向

axis no 轴号；**范围：**根据控制器实际硬件决定。

例程

```
BASE 0,1,2
JOG_VL=500
JOG_VH=10000
JOG_ACC=50000
JOG_DEC=50000
JOG_VLTIME=2000     '设定低段速度运行的时间为 2 秒
JOGN AX(1)    '指定轴 1 执行负向 JOG 运动，轴速度会先加速到 JOG_VL 运行 2 秒，再加速到 JOG_VH
SLEEP 5000
STOPDEC
JOGN        '轴 0、1、2 都执行负向 JOG 运动
SLEEP 3000
STOPDEC
WAIT DONE
JOGN 0,1,0'轴 0,2 执行负向 JOG 运动，轴 1 执行正向 JOG 运动
SLEEP 4000
STOPDEC
```

2.12.8 JOGON

所属：命令

语法：JOGON

描述：BASE 轴列表的第一个轴，使能外部驱动的 JOG 功能。该指令对外部硬件接线控制的 JOG 运动起作用。研华运动控制的每个轴都关联着 4 个 DI 端口，分别称为 IN1，IN2，IN4，IN5。当使用外部驱动的 JOG 功能时，IN4 和 IN5 分别控制 JOG+和 JOG-。

2.12.9 JOGOFF

所属：命令

语法：JOGOFF

描述：BASE 轴列表的第一个轴，禁用外部驱动的 JOG 功能。该指令对外部硬件接线控制的 JOG 运动起作用。研华运动控制的每个轴都关联着 4 个 DI 端口，分别称为 IN1，IN2，IN4，IN5。当使用外部驱动的 JOG 功能时，IN4 和 IN5 分别控制 JOG+和 JOG-。

2.12.10 MPGON

所属：命令

语法：MPGON

描述：BASE 轴列表的第一个轴，使能外部驱动的 MPG 功能。该指令对外部硬件接线控制的 MPG 运动起作用。研华运动控制的每个轴都关联着 4 个 DI 端口，分别称为 IN1，IN2，IN4，IN5。当使用外部驱动的 MPG 功能时，IN4 和 IN5 分别控制对应手轮脉冲输入的 A 相和 B 相。

2.12.11 MPGOFF

所属：命令

语法：MPGOFF

描述：BASE 轴列表的第一个轴，禁用外部驱动的 MPG 功能。该指令对外部硬件接线控制的 MPG 运动起作用。研华运动控制的每个轴都关联着 4 个 DI 端口，分别称为 IN1，IN2，IN4，IN5。当使用外部驱动的 MPG 功能时，IN4 和 IN5 分别控制对应手轮脉冲输入的 A 相和 B 相。

2.12.12 EXT_MODE

所属：属性

语法：EXT_MODE = value

类型：ULONG

描述：设置/读取手轮模式外部驱动的脉冲输入模式

范围:设定值和返回值如下，默认值 2

0：1XAB

1：2XAB

2：4XAB

3：CCW/CW

例程

```
EXT_MODE =1 ‘设置手轮外部驱动的脉冲输入模式为 2XAB
```

2.12.13 EXT_PULSE

所属：属性

语法：EXT_PULSE= value

类型：ULONG

描述：设置/读取手轮模式外部驱动时，每个手轮脉冲输入对应多少个指令脉冲输出值

范围：【1,1000】；默认值为 1

例程

```
EXT_PULSE =2 '设置手轮脉冲输入对应 2 个指令脉冲输出
```

2.12.14 EXT_SRC

所属：属性

语法：EXT_SRC= value

类型：ULONG

描述：设置/读取外部驱动的信号接入哪个轴的外部驱动输入端口

范围：设定值和返回值如下，默认值 0

0： 0 轴

1： 1 轴(暂不支持)

2： 2 轴(暂不支持)

3： 3 轴(暂不支持)

例程

```
BASE 0
EXT_SRC =0 '设置外部驱动信号接到轴 0 的外部驱动端口
```

2.13 通信指令

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.13.1	COM_OPEN	打开串口	×	×
2.13.2	COM_CLOSE	关闭串口	×	×
2.13.3	COM_SET	设置串口通讯参数	×	×
2.13.4	COM_ReadStream	串口自由协议读操作，通过串口读数据	×	×
2.13.5	COM_WriteStream	串口自由协议写操作，通过串口写数据	×	×
2.13.6	COM_ResetBuf	清除串口缓存区资料	×	×
2.13.7	COM_Check	获取串口接收缓冲区的字符数	×	×
2.13.8	TCP_OPEN	打开一个 TCP 通讯连接	×	×
2.13.9	TCP_CLOSE	关闭一个 TCP 通讯连接	×	×
2.13.10	TCP_STATUS	检查 TCP 连接状态	×	×
2.13.11	TCP_WAIT	等待 TCP 连接完成	×	×
2.13.12	TCP_Check	获取 TCP 通信接收缓冲区中的字符个数	×	×
2.13.13	TCP_ ReadSTR	TCP/IP 读字符串操作	×	×
2.13.14	TCP_WriteSTR	TCP/IP 写字符串操作	×	×
2.13.15	TCP_Read	TCP/IP 读整型数操作	×	×
2.13.16	TCP_Write	TCP/IP 写整型数操作	×	×
2.13.17	TCP_ReadVR	TCP/IP 通过 VR 进行读整型数操作	×	×
2.13.18	TCP_WriteVR	TCP/IP 通过 VR 进行写整型数操作	×	×
2.13.19	TCP_ResetBuf	清除 TCP 缓存区数据	×	×
2.13.20	MB_OPEN	打开 Modbus 连接	×	×

2.13.21	MB_CLOSE	关闭 Modbus 连接	×	×
2.13.22	MB_STATUS	获取 Modbus 连接状态	×	×
2.13.23	MB_SETCOIL	设置单个或多个线圈数值	×	×
2.13.24	MB_GETCOIL	获取单个或多个线圈数值	×	×
2.13.25	MB_GETINPUT	获取单个或多个离散输入值	×	×
2.13.26	MB_SETHDREG	设置单个或多个 Holding register 值	×	×
2.13.27	MB_GETHDREG	获取单个或多个 Holding register 值	×	×
2.13.28	MB_GETINREG	获取单个或多个 Input register 值	×	×

2.13.1 COM_OPEN

所属：命令

语法：COM_OPEN port

描述：指定串口编号，打开串口。相应串口端口被打开后，才可以对该串口操作。该指令需要根据本地串口资源进行操作。

参数：port 串口端口号

注意：打开串口操作仅适用于未打开的串口，如果串口资源已经被打开，下该指令操会执行不成功，并返回错误。COM1 默认是被 Motion Runtime 占用，使用串口通信时请使用其它 COM 口。如非要用 COM1，请至控制器 Motion Runtime 文件夹下打开 Guard.ini 配置文件，将 COM_PORT 值改成其它 COM 口。

例程

```
COM_OPEN 2          '打开串口 2
COM_SET 2, 9600, 0, 1, 8  '设置串口波特率 9600，校验位无，停止位 1 位，数据位 8 位
COM_CLOSE 2         '关闭串口 2
```

2.13.2 COM_CLOSE

所属：命令

语法：COM_ClOSE port

描述：指定串口编号，关闭串口。

参数：port 串口端口号

例程

```
COM_OPEN 2          '打开串口 2
COM_SET 2, 9600, 0, 1, 8  '设置串口波特率 9600，校验位无，停止位 1 位，数据位 8 位
COM_CLOSE 2         '关闭串口 2
```

2.13.3 COM_SET

所属：命令

语法：COM_SET port , baudrate, parity, stopbits, databits

描述：设置串口通讯参数。

参数：port　　串口端口号；

Baudrate 波特率；**范围：**4800、9600、19200、38400、57600、115200

Parity 校验方式；**范围：**无（NONE）、奇（ODD）、偶（EVEN）

Stopbits 停止位；**范围：**1、2

Databits　数据位；**范围：**7、8

例程

```
COM_OPEN 2          '打开串口 2
COM_SET 2, 9600, 0, 1, 8  '设置串口波特率 9600，校验位无，停止位 1 位，数据位 8 位
COM_CLOSE 2         '关闭串口 2
```

2.13.4 COM_ReadStream

所属：命令

语法：COM_READSTREAM port, *strarray, num [,timeout]

描述：串口自由协议读操作，通过串口读数据。执行到该指令时，timeout 时间未到，程序会等在该行，直到读到的字节个数和 num 参数指定的个数一致时，程序才会执行到下一行。timeout 时间到后，如还未接收到指定个数的字节，该指令执行就会结束，程序继续执行下一条指令。

参数：port　　串口端口号；

*strarray　存放读到的数据变量地址，一般为数组的地址或字符串地址

num　　读取的字节个数或字符个数

timeout　接收的超时时间，单位为 ms。timeout 时间到后，如还未接收到指定个数的字节，该指令执行就会结束，程序继续执行下一条指令。*strarray 中接收到的数据会被清空。

例程

```
'例程 1：以 BYTE 数值来接收数据
DIM ReadArray(1) AS BYTE
COM_Open 2              '打开串口 2
COM_SET 2,9600,0,1,8 '设置串口波特率 9600，校验位无，停止位 1 位，数据位 8 位
COM_ReadStream 2,ReadArray(),2,2000    '从串口接收缓存区读 2 个字节，timeout 为 2 秒
PRINT ReadArray(0),ReadArray(1)
COM_Close 2             '关闭串口 2

'例程 2：以字符串来接收数据
DIM ReadStr AS STRING
COM_Open 2              '打开串口 2
COM_SET 2,9600,0,1,8 '设置串口波特率 9600，校验位无，停止位 1 位，数据位 8 位
COM_ReadStream 2,ReadStr,4,3000     '从串口接收缓存区读 4 个字节，timeout 为 3 秒
PRINT ReadStr
COM_Close 2             '关闭串口 2
```

2.13.5 COM_WriteStream

所属：命令

语法：COM_WriteStream port, *strarray, num

描述：串口自由协议写操作，通过串口写数据。

参数：port 串口端口号；

*strarray 存放写出的数据变量地址，一般为数组的地址或字符串地址

num 写出的字节个数或字符个数

例程

```
'例程 1：以 BYTE 数值来发送数据
DIM WriteArray(1) AS BYTE={1,2}
DIM WriteStr AS STRING="OK"
COM_Open 2                '打开串口 2
COM_SET 2,9600,0,1,8 '设置串口波特率 9600，校验位无，停止位 1 位，数据位 8 位
COM_WriteStream 2,WriteArray(),2 '控制器发出数组 WriteArray()里的 2 个字节数据
COM_WriteStream 2,WriteStr,2      '控制器写出 WriteStr 中字符串
COM_Close 2                       '关闭串口 2
```

2.13.6 COM_ResetBuf

所属：命令

语法：COM_ResetBufport

描述：清除串口缓存区资料。

参数：port 串口端口号

2.13.7 COM_Check

所属：命令

语法：value=COM_Check(no)

类型：LONG

描述：获取串口通讯接收缓冲区的字符个数

参数：no COM 口端口号；

返回值：如下说明。

0~正值：字符个数

-1：端口使用错误

-2：端口打开失败

-3：接收字符个数失败

例程

```
DIM WriteStr AS STRING="HI,MY COM"
DIM ReadStr AS STRING
DIM ReadNUM AS LONG
COM_Open 2                  '打开串口 2
COM_SET 2,9600,0,1,8        '设置串口波特率 9600，校验位无，停止位 1 位，数据位 8 位
ReadNum=COM_CHECK(2)        '获取串口 2 的接收缓存区里的字符个数
IF ReadNum>0 THEN
  PRINT ReadNum
  COM_ReadStream 2,ReadStr,ReadNum,3000    '串口 2 接收缓存区里的字符取出
  PRINT ReadStr
END IF
COM_Close 2
```

2.13.8 TCP_OPEN

所属：命令

语法：TCP_OPEN no，mode，port[，ipaddress]

描述：指定 TCP/IP 通讯编号、模式、网络端口号[、IP 地址]，打开一个 TCP 通信连接。相应 TCP 通信端口被打开后，才可以对该网口操作。该指令需要根据本地网口资源进行操作。

参数：no TCP 通讯编号。用于控制器内部识别不同 TCP/IP 连接。类型为 ULONG，没用过的编号可以随意指定，比如 0,1,2,3,4,5.....

mode 连接模式；**范围：**0：控制器作为服务器，1：控制器作为客户端。

Port 网络端口号

ipaddress：IP 地址，控制器作为服务器时，不需要填该参数。控制器作为客户端时，该参数填服务器端网口 IP 地址

注意：TCP_Open 创建连接时，需用 TCP_WAIT 指令等待通信连接成功，才可以正常进行通信操作。若 MAS 控制器作为服务器，程序会停在 TCP_WAIT 指令行，直到有客户端连上 MAS 控制器这个服务器，程序才会执行执行下一行。若 MAS 控制器作为客户端，程序不会停在 TCP_WAIT 指令行。需用 TCP_STATUS 判断控制器是否有连上服务器。

例程

```
'MAS 控制器作为服务器
TCP_Open(1,0,5025)              '创建一个 TCP 服务器连接，服务器处于监听状态
TCP_WAIT 1                      '等待连接完成
TCP_Close 1                     '关闭编号为 1 的网络服务器

'MAS 控制器作为客户端
TCP_Open 2,1,5024,"192.168.0.11"    '对接 IP 为 192.168.0.11 的服务器
TCP_WAIT 2                          '等待连接完成
TCP_Close 2                         '关闭编号为 2 的网络客户端
```

2.13.9 TCP_CLOSE

所属：命令

语法：TCP_ClOSE no

描述：指定 TCP 通讯编号，关闭对应 TCP 通信端口

参数：noTCP 通讯编号；**类型：**ULONG

例程

```
'MAS 控制器作为服务器
TCP_Open(1,0,5025)              '创建一个 TCP 服务器连接，服务器处于监听状态
TCP_WAIT 1                      '等待连接完成
TCP_Close 1                     '关闭编号为 1 的网络服务器

'MAS 控制器作为客户端
TCP_Open 2,1,5024,"192.168.0.11"    '对接 IP 为 192.168.0.11 的服务器
```

```
TCP_WAIT 2                          '等待连接完成
TCP_Close 2                         '关闭编号为 2 的网络客户端
```

2.13.10 TCP_STATUS

所属：命令

语法：value=TCP_STATUS (no)

类型：ULONG

描述：检查 TCP 通讯连接状态

参数：no　　TCP 通讯编号

返回值：0：连接不成功；1：连接成功

例程

```
'请参考 TCP_ReadSTR 或 TCP_WriteSTR 指令
```

2.13.11 TCP_WAIT

所属：命令

语法：TCP_WAIT no [,timeout]

描述：等待 TCP 连接完成。执行该指令时，程序会等待在该行直到 TCP 通讯连接成功或 timeout 超时，程序才会继续下一行的执行。

参数：no　　TCP 通讯编号；

timeout　等待超时时间，单位为 ms。Timeout 时间到后，TCP 通讯连接还未成功，程序会继续下一行的执行。

注意：TCP_Open 创建连接时，需用 TCP_WAIT 指令等待通信连接成功，才可以正常进行通信操作。若 MAS 控制器作为服务器，程序会停在 TCP_WAIT 指令行，直到有客户端连上 MAS 控制器这个服务器，程序才会执行执行下一行。若 MAS 控制器作为客户端，程序不会停在 TCP_WAIT 指令行。需用 TCP_STATUS 判断控制器是否有连上服务器。

例程

```
'请参考 TCP_ReadSTR 或 TCP_WriteSTR 指令
```

2.13.12 TCP_Check

所属：命令

语法：value=TCP_Check(no)

类型：LONG

描述：获取 TCP 通讯接收缓存区的字符个数。

参数：no　　TCP 通讯编号；

返回值：如下说明。

0~正值：字符个数

-1：端口有打开，但未建立通信

-2：其它任务在使用该端口

-3：未打开端口

例程

```
Dim CharCount as LONG =0
Dim StrData as string
TCP_Open (0, 1, 8080, "127.0.0.1") '创建一个客户端连接，对接 IP 为"127.0.0.1"的服务器
TCP_Wait 0                         '等待连接完成
TCP_ResetBuf 0                     '清空 TCP 接收缓存区
If TCP_STATUS(0) > 0 Then          '确认通讯编号为 0 的连接是否正常
    WHILE 1
        CharCount = TCP_Check(0)    '读 TCP 接收缓存区中的字符个数
        IF CharCount>0    then
            TCP_ReadSTR(0, StrData, CharCount) '读出接收缓存区中的字符串
            PRINT StrData
        END IF
        SLEEP 10
    WEND
End If
TCP_CLOSE 0                        '断开通讯编号为 0 的 TCP 连接
```

2.13.13 TCP_ReadSTR

所属：命令

语法：TCP_ReadSTR no,strData ,numChars [,strEnd]/[,timeout] [,timeout]

描述：TCP/IP 自由协议读操作，控制器从 TCP 接收缓存区读字符串指令。执行到该指令时，程序会等在该指令行，直到读到字符或 timeout 超时，程序才会继续下一行的执行。

参数：

no　　TCP 通讯编号；**类型：**ULONG

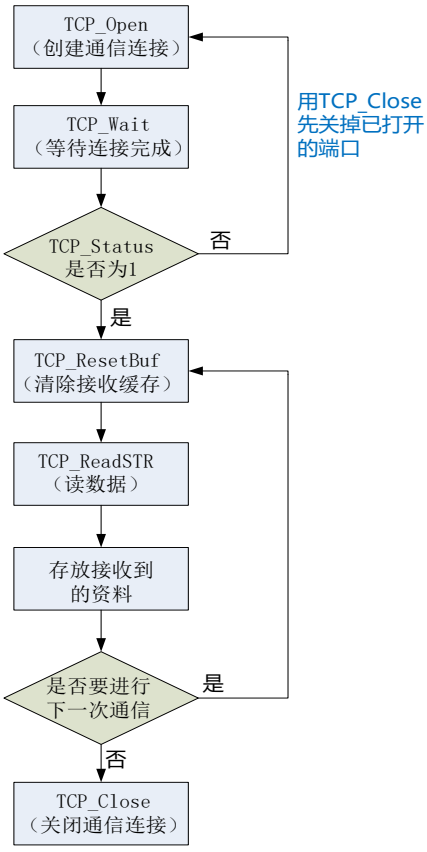
strData　　存放接收的字符串；**类型：**String

numChars　　指定从接收缓存区读的字符个数；**类型：**ULONG。接收缓存区里总共有多少字符个数可由 TCP_check 指令获取得到。

strEnd　　读操作的字符串结束符。如接收缓存区里共有字符串"ab1*cd",而这里结束符填"*"，那么执行 TCP_ReadSTR 指令去读取时，会读到"ab1","*"后面的"cd"将不会读进来。**类型：**String，该参数如果是填整型数值而非 String 类型，那该参数将会作为 timeout 使用。

timeout　　接收的超时时间，单位为 ms。Timeout 时间到后，还未读到任何字符，程序将执行下一行。

注意： TCP 数据读操作的相关指令，如 TCP_ReadSTR、TCP_Read、TCP_ReadVR 要注意通信接收缓存的处理。为避免通信接收缓存中的遗留数据影响新的接收数据，读操作指令前可用 TCP_ResetBuf 清除通信缓存，不然读操作指令会先读到遗留在通信缓存中的数据，导致读到的数据不对。TCP 读操作指令的操作可以参照以下流程图处理。



例程

```
Dim StrData as string
TCP_Open (0, 1, 8080, "127.0.0.1") '创建一个客户端连接，对接 IP 为"127.0.0.1"的服务器
TCP_Wait 0                         '等待连接完成
TCP_ResetBuf 0                     '清空 TCP 接收缓存区
If TCP_STATUS(0) > 0 Then          '确认通讯编号为 0 的连接是否正常
    '指定最多读 10 个字符
    TCP_ReadSTR(0, StrData, 10)
    PRINT StrData

    '指定最多读 10 个字符,且读到字符"*"时就不读了
    TCP_ReadSTR(0, StrData, 10, "*")
    PRINT StrData

    '指定最多读 10 个字符,timeout 为 4s
    TCP_ReadSTR(0, StrData, 10, 4000)  '4s 后还未读到任何字符，程序往下执行
    PRINT StrData
End If
TCP_CLOSE 0                        '断开通讯编号为 0 的 TCP 连接
```

2.13.14 TCP_WriteSTR

所属：命令

语法：TCP_WriteSTRno, strData

描述：TCP/IP 自由协议写操作，控制器发送出字符串指令

参数：no TCP 通讯编号；

strData 发送出去的字符串；**类型：**String

例程

```
Dim StrData as string="Hi,MAS"
TCP_Open (0, 1, 8080, "127.0.0.1") '创建一个客户端连接，对接 IP 为"127.0.0.1"的服务器
TCP_Wait 0                         '等待连接完成
If TCP_STATUS(0) > 0 Then          '确认通讯编号为 0 的连接是否正常
    TCP_WRITESTR 0,"I'm Ready"     '直接写字符串
    TCP_WRITESTR 0,StrData         '通过变量写字符串
End If
TCP_CLOSE 0                        '断开通讯编号为 0 的 TCP 连接
```

2.13.15 TCP_Read

所属：命令

语法：TCP_Read no,array( ),arrayCnt [,timeout]

描述：TCP/IP 自由协议读操作，控制器接收数据。控制器会根据 array( )定义的数据类型（byte 或 short 或 long），将每接收到的数据按 1 个字节或 2 个字节或 4 个字节为一个数据依次放入 array( )中。执行到该指令时，程序会等在该指令行，直到接收到数据或 timeout 超时，程序才会继续下一行的执行。

参数：no　　TCP 通讯编号；**类型：**ULONG

Array( )　存放接收数据的数组；**类型：**byte、short、long

arrayCnt　接收的数据个数；**类型：**ULONG。

Timeout　接收的超时时间，单位为 ms。Timeout 时间到后，还未接收到数据，系统会断开当前通讯编号的 TCP 连接。如还需要进行通讯，需重新创建 TCP 连接。

注意：TCP 接收的相关指令，如 TCP_ReadSTR、TCP_Read、TCP_ReadVR 要注意通信接收缓存的处理。为避免通信接收缓存中的遗留数据影响新的接收数据，接收指令前需用 TCP_ResetBuf 清除通信缓存，不然接收指令会先收到遗留在通信缓存中的数据，导致接收的数据不对。TCP 接收指令的操作可以参照 TCP_ReadSTR 指令说明中的流程图来处理。

例程

```
Dim R_ByteArray(0 to 1) as BYTE
Dim R_ShortArray(0 to 1) as SHORT
Dim R_LongArray(0 to 1) as  LONG
TCP_Open (0, 1, 8080, "127.0.0.1")      '创建一个客户端连接，对接 IP 为"127.0.0.1"的服务器
TCP_Wait 0                              '等待连接完成
If TCP_STATUS(0) > 0 Then    '确认通讯编号为 0 的连接是否连接成功
    '接收到的 2 个 Byte 资料分别存入 R_ByteArray(0)，R_ByteArray(1)
    TCP_READ 0,R_ByteArray(),2
    PRINT R_ByteArray(0),R_ByteArray(1)

    '接收到的前 2 个 Byte 数据组成 Short 类型数据存入 R_ShortArray(0)
    '接收到的后 2 个 Byte 数据组成 short 类型数据存入 R_ShortArray(1)
    TCP_READ 0,R_ShortArray(),2
    PRINT R_ShortArray(0),R_ShortArray(1)

    '接收到的前 4 个 Byte 数据组成 long 类型数据存入 R_LongArray(0)
    '接收到的后 4 个 Byte 数据组成 long 类型数据存入 R_LongArray(1)
    TCP_READ 0,R_LongArray(),2
    PRINT R_LongArray(0),R_LongArray(1)
End If
TCP_CLOSE 0             '断开通讯编号为 0 的 TCP 连接
```

2.13.16 TCP_Write

所属：命令

语法：TCP_Write no, array( ),arrayCnt

描述：TCP/IP 自由协议写操作，控制器发送出数据。发送的数据可以选择 byte、short、long 三种数据类型。控制器会根据数据类型按 byte 的数据送出。如果是 short、long 类型，发出去的 byte 会以低位元组到高字节的顺序发送。

参数：no TCP 通讯编号；

Array( )发送出去数据的数组；**类型：**byte，short，long

ArrayCnt 发送出去的数据个数。

例程

```
Dim W_ByteArray(0 to 1) as BYTE ={-75,121}
Dim W_ShortArray(0 to 1) as SHORT ={135,32753}
Dim W_LongArray(0 to 1) as  LONG={-24,175024}
TCP_Open (0, 1, 8080, "127.0.0.1") '创建一个客户端连接，对接 IP 为"127.0.0.1"的服务器
TCP_Wait 0                          '等待连接完成

'相隔 3 秒，依次发送 W_ByteArray(),W_ShortArray(),W_LongArray()
If TCP_STATUS(0) > 0 Then           '确认通讯编号为 0 的连接是否连接成功
    '服务器端收到的数据为十六进制数：B5 79。对应-75,121
    TCP_Write 0,W_ByteArray(),2
    SLEEP 3000
    '服务器端收到的数据为十六进制数：87 00 F1 7F。对应 135,32753
    TCP_Write 0,W_ShortArray(),2
    SLEEP 3000
    '服务器端收到的数据为十六进制数：E8 FF FF FF B0 AB 02 00。对应-24,175024
    TCP_Write 0,W_LongArray(),2
End If
TCP_CLOSE 0             '断开通讯编号为 0 的 TCP 连接
```

2.13.17 TCP_ReadVR

所属：命令

语法：TCP_ReadVR no,VR_StartIndex ,VRCnt, format [,timeout]

描述：TCP/IP 自由协议读操作，控制器用 VR 变量接收数据。控制器会根据 format 指定的数据类型（byte 或 short 或 long），将每接收到的数据按 1 个字节或 2 个字节或 4 个字节为一个数据依次放入 VR 变数中。执行到该指令时，程序会等在该指令行，直到接收到数据或 timeout 超时，程序才会继续下一行的执行。

参数：no TCP 通讯编号；**类型：**ULONG

VR_StartIndex 存放接收数据的起始 VR；**类型：**byte、short、long

VRCnt 接收的数据个数；**类型：**ULONG

format 指定存放接收数据的类型。

0：byte

1：short

2：long

timeout 接收的超时时间，单位为 ms。Timeout 时间到后，还未接收到数据，系统会断开当前通讯编号的 TCP 连接。如还需要进行通讯，需重新创建 TCP 连接。

注意：TCP 接收的相关指令，如 TCP_ReadSTR、TCP_Read、TCP_ReadVR 要注意通信接收缓存的处理。为避免通信接收缓存中的遗留数据影响新的接收数据，接收指令前需用 TCP_ResetBuf 清除通信缓存，不然接收指令会先收到遗留在通信缓存中的数据，导致接收的数据不对。TCP 接收指令的操作可以参照 TCP_ReadSTR 指令说明中的流程图来处理。

例程

```
TCP_Open (0, 1, 8080, "127.0.0.1") '创建一个客户端连接，对接 IP 为"127.0.0.1"的服务器
TCP_Wait 0                         '等待连接完成
If TCP_STATUS(0) > 0 Then          '确认通讯编号为 0 的连接是否连接成功

    '接收到的 2 个 Byte 资料分别存入 VR(0)，VR(1)
    TCP_ReadVR 0,0,2,0
    PRINT VR(0),VR(1)

    '接收到的前 2 个 Byte 数据组成 Short 类型数据存入 VR(2)
    '接收到的后 2 个 Byte 数据组成 short 类型数据存入 VR(3)
    TCP_ReadVR 0,2,2,1
    PRINT VR(2),VR(3)

    '接收到的前 4 个 Byte 数据组成 long 类型数据存入 VR(4)
    '接收到的后 4 个 Byte 数据组成 long 类型数据存入 VR(5)
    TCP_ReadVR 0,4,2,2
    PRINT VR(4),VR(5)
End If
TCP_CLOSE 0          '断开通讯编号为 0 的 TCP 连接
```

2.13.18 TCP_WriteVR

所属：命令

语法：TCP_WriteVR no, VR_StartIndex ,VRCnt, format

描述：TCP/IP 自由协议写操作，控制器把 VR 变量中的数据发送出去。发送的数据可以选择 byte、short、long 三种数据类型。控制器会根据数据类型按 byte 的数据送出。

参数：no TCP 通讯编号；

VR_StartIndex 发送数据的起始 VR；**类型：**byte、short、long

VRCnt 发送的资料个数；**类型：**ULONG

format 指定发送数据的类型。

0：byte

1：short

2：long

注意：如果发送出去的 VR 变量数据是浮点型数据，接收端接收到的数据会失真。如要发送浮点数据，请用 TCP_WriteSTR 指令，用字符串形式发送出去，接收端接收到字符串后再转数据类型来接收浮点数据。

例程

```
VR(0)=-75
VR(1)=121
VR(2)=135
VR(3)=32753
VR(4)=-24
VR(5)=175024
TCP_Open (0,1,8080,"127.0.0.1")     '创建一个客户端连接，对接 IP 为"127.0.0.1"的服务器
TCP_Wait 0                          '等待连接完成

'相隔 3 秒，依次发送 VR(0)、VR(1) ;VR(2)、VR(3) ;VR(4)、VR(5)
If TCP_STATUS(0) > 0 Then           '确认通讯编号为 0 的连接是否连接成功
'将 VR(0),VR(1)发送出去，服务器端收到的数据为十六进制数：B5 79。对应-75,121
    TCP_WriteVR 0,0,2,0
    SLEEP 3000
'将 VR(2),VR(3)发送出去，服务器端收到的数据为十六进制数：87 00 F1 7F。对应 135,32753
    TCP_WriteVR 0,2,2,1
    SLEEP 3000
'将 VR(4),VR(5)发送出去，服务器端收到的数据为十六进制数：E8 FF FF FF B0 AB 02 00。对应
-24,175024
    TCP_WriteVR 0,4,2,2
End If
TCP_CLOSE 0             '断开通讯编号为 0 的 TCP 连接
```

2.13.19 TCP_ResetBuf

所属：命令

语法：TCP_ResetBufno

描述：清除 TCP 缓存区数据。

参数：no TCP 通讯连接编号

注意：TCP 接收的相关指令，如 TCP_ReadSTR、TCP_Read、TCP_ReadVR 要注意通信接收缓存的处理。为避免通信接收缓存中的遗留数据影响新的接收数据，接收指令前需用 TCP_ResetBuf 清除通信缓存，不然接收指令会先收到遗留在通信缓存中的数据，导致接收的数据不对。

2.13.20 MB_OPEN

所属：命令

语法：VALUE = MB_OPEN(mbindex, connectmode, ip/comid, port/baudrate, deviceID[,parity,stopbits,databits])

类型：BOOLEAN。

描述：指定 modbus 通讯编号、模式、网络端口号、IP port 或波特率、设备 ID，打开一个 modbus tcp 连接或 modbus rtu 串口。相应通信端口或串口被打开后，才可以对该端口或串口进行操作。该指令需要根据本地资源进行操作。

参数：

Mbindex 设定一个 modbus 通信序号，0~4294967294

Connectmode 连接模式 0：Modbus RTU，1：Modbus Tcp client

ip/comid IP 地址或 com 端口号

port/baudrate IP port 或波特率

deviceID Device ID 范围 1~247

parity 奇偶位 0：none；1：even； 2：odd

stopbits 停止位 0：1 ； 1：1.5 ； 2：2

databits 数据位 7/8

返回值： TRUE：打开成功；FALSE：打开失败

例程

```
'完整例程可参考 MB_GETHDREG 指令
MB_OPEN(0, 1, "127.0.0.1", 502, 1)  '打开一个 modbus tcp 客户端连接，
'对接 IP 为 127.0.0.1 的服务器
MB_CLOSE(0)                         '关闭编号为 0 的网络客户端
```

2.13.21 MB_CLOSE

所属：命令

语法：VALUE = MB_CLOSE(mbindex)

类型：BOOLEAN。

描述：关闭指定序号的 modbus 连接。

参数：mbindex 通讯编号

返回值：TRUE—关闭成功，FALSE—关闭失败

例程

```
'完整例程可参考 MB_GETHDREG 指令
MB_CLOSE(0)        '关闭编号为 0 的网络客户端
```

2.13.22 MB_STATUS

所属：命令

语法：VALUE = MB_STATUS(mbindex)

类型：ULONG。

描述：获取 modbus 连接状态

参数：mbindex 通讯编号。

返回值：0：连接不成功；1：连接成功

例程

```
MB_STATUS(0)        '获取编号为 0 的 modbus 连接状态
```

2.13.23 MB_SETCOIL

所属：命令

语法 1：设置单个线圈数值:

VALUE = MB_SETCOIL(mbindex, m_start_address, Value)

语法 2：设置多个线圈数值:

VALUE = MB_SETCOIL(mbindex, m_start_address, ValueArray(), DataCnt)

类型：BOOLEAN。

描述：设置单个或多个线圈数值。

参数：mbindex 通讯编号

m_start_address modbus 相对起始地址(首地址为 0)

Value 设定单个值

ValueArray() 设定多个值

DataCnt 需传输的数值个数

返回值：TRUE—设置成功，FALSE—设置失败

例程

```
'请参考 MB_GETCOIL 指令
```

2.13.24 MB_GETCOIL

所属：命令

语法 1：获取单个线圈数值:

VALUE =MB_GETCOIL(mbindex, m_start_address, OutputValue)

语法 2：获取多个线圈数值:

VALUE=MB_GETCOIL(mbindex, m_start_address, OutputValueArray(),

DataCnt)

类型：BOOLEAN。

描述：获取单个或多个线圈数值。

参数：mbindex　通讯编号

m_start_address　modbus 相对起始地址(首地址为 0)

OutputValue　读取时用于接收值

OutputValueArray　用于接收获取到的多个数值的数组

DataCnt　需传输的数值个数

返回值：TRUE—获取成功，FALSE—获取失败

例程

```
'设置或获取单个线圈数值
DIM coil_data AS BYTE = 0
DIM As INTEGER mb_Index =0, i
DIM As USHORT startAddress = 0, data_count = 3
MB_SETCOIL(mb_Index, startAddress, 1)
MB_GETCOIL(mb_Index, startAddress, coil_data)
IF coil_data<>1 THEN
 PRINT "Sigle coil failed."
END IF
'设置或获取多个线圈数值
DIM temp_in(11) As BYTE = {1,1,1,0,0,1,1,1,0,0,1}
MB_SETCOIL(mb_Index, startAddress, temp_in(), data_count)
DIM temp_out(11) As BYTE
MB_GETCOIL(mb_Index, startAddress, temp_out(), data_count)
FOR i As INTEGER = 0 to data_count-1
 PRINT "Coil address ";startAddress+i;" data = ";temp_out(i)
 IF temp_out(i)<>temp_in(i) THEN
  PRINT "Multiple coil failed."
 END IF
NEXT i
```

2.13.25 MB_GETINPUT

所属：命令

语法 1：获取单个离散输入值:

VALUE=MB_GETINPUT(mbindex, m_start_address, OutputValue)

语法 2：获取多个离散输入值:

VALUE=MB_GETINPUT (mbindex, m_start_address,OutputValueArray(),

DataCnt)

类型：BOOLEAN。

描述：获取单个或多个离散输入值。

返回值：TRUE—获取成功，FALSE—获取失败

参数：mbindex　通讯编号

m_start_address　modbus 相对起始地址(首地址为 0)

OutputValue　读取时用于接收值

OutputValueArray　用于接收获取到的多个数值的数组

DataCnt　需传输的数值个数(非地址个数)，实际地址个数依照传入的数据类型而定

例程

```
'设置或获取多个离散输入值
DIM temp_input(11) As BYTE
MB_GETINPUT(mb_Index, startAddress, temp_input(), data_count)
FOR i As INTEGER = 0 to data_count-1
 PRINT "Input bit address ";startAddress + i;" data = ";temp_input(i)
NEXT i
```

2.13.26 MB_SETHDREG

所属：命令

语法 1：设置单个 Holding register 值:

VALUE=MB_SETHDREG(mbindex,m_start_address,Value[, DataType])

语法 2：设置多个 Holding register 值:

VALUE =MB_SETHDREG(mbindex, m_start_address, ValueArray(), DataCnt)

类型：BOOLEAN。

描述：设置单个或多个 Holding register 值。

参数：mbindex 通讯编号

m_start_address modbus 相对起始地址(首地址为 0)

Value 设定单个值

ValueArray() 设定多个值

DataCnt 需传输的数值个数(非地址个数)，实际地址个数依照传入的数据类型而定。

DataType：数据类型，目前支持以下几种

DATATYPE_U16 0

DATATYPE_I16 1

DATATYPE_U32 2

DATATYPE_I32 3

DATATYPE_F32 4

DATATYPE_F64 5

返回值：TRUE—设置成功，FALSE—设置失败

例程

```
'也可参考 MB_GETHDREG 指令例程
DIM As INTEGER mb_Index =0, i
DIM As USHORT startAddress = 3, data_count = 3
DIM sData As SHORT
DIM fData AS SINGLE
MB_OPEN(mb_Index, 1, "127.0.0.1", 502, 1)
MB_SETHDREG(mb_Index, startAddress, -10, DATATYPE_I16)
MB_GETHDREG(mb_Index, startAddress, sData)
PRINT "Short data:";sData
MB_SETHDREG(mb_Index, startAddress, -10.123, DATATYPE_F32)
MB_GETHDREG(mb_Index, startAddress, fData)
PRINT "Float data:";fData
MB_CLOSE(0)
SLEEP 1000
```

2.13.27 MB_GETHDREG

所属：命令

语法 1：获取单个 Holding register 值:

VALUE =MB_GETHDREG(mbindex, m_start_address, OutputValue)

语法 2：获取多个 Holding register 值:

VALUE=MB_GETHDREG(mbindex,m_start_address,OutputValueArray(), DataCnt)

类型：BOOLEAN。

描述：获取单个或多个 Holding register 值。

参数：mbindex 通讯编号

m_start_address modbus 相对起始地址(首地址为 0)

OutputValue 读取时用于接收值

OutputValueArray 用于接收获取到的多个数值的数组

DataCnt 需传输的数值个数(非地址个数)，实际地址个数依照传入的数据类型而定

返回值：TRUE—获取成功，FALSE—获取失败

例程

```
Dim As INTEGER mb_Index =0, i
Dim As USHORT startAddress = 0, data_count = 3
IF MB_OPEN(mb_Index, 1, "127.0.0.1", 502, 1)=FALSE THEN
 PRINT "Open modbus failed."
END IF
'Write &Read Ushort register value
Dim usData As USHORT
MB_SETHDREG(mb_Index, startAddress, 65534)
MB_GETHDREG(mb_Index, startAddress, usData)
PRINT "Holding register address";startAddress;", Ushort data =";usData
IF usData<>65534 THEN
 PRINT "Ushort register failed."
END IF
usData = 0
MB_GETINREG(mb_Index, startAddress, usData)
PRINT "Input register address";startAddress;", Uhsortdata =";usData
MB_CLOSE(0)
SLEEP 1000
```

2.13.28 MB_GETINREG

所属：命令

语法 1：获取单个 Input register 值:

VALUE=MB_GETINREG(mbindex, m_start_address, OutputValue)

语法 2：获取多个 Input register 值:

VALUE=MB_GETINREG(mbindex,m_start_address, OutputValueArray(),
DataCnt)

类型：ULONG。

描述：获取单个或多个 Input register 值。

参数：mbindex 通讯编号

m_start_address modbus 相对起始地址(首地址为 0)

OutputValue 读取时用于接收值

OutputValueArray 用于接收获取到的多个数值的数组

DataCnt 需传输的数值个数(非地址个数)，实际地址个数依照传入的数据类型而定

返回值：TRUE—获取成功，FALSE—获取失败

例程

'请参考 MB_GETHDREG 指令**例程**

2.14 字符串处理

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.14.1	ASC	返回字符串中字符的 ASCII 码	×	×
2.14.2	CHR	返回用 ASCII 码表达的值对应的字符	×	×
2.14.3	HEX	返回数值的十六进制结果	×	×
2.14.4	INSTR	查找字符串中第一次出现的字符或者字符串	×	×
2.14.5	LCASE	将字符串中的字母全部转变成小写字母返回	×	×
2.14.6	LEFT	返回字符串从左开始指定字符个数的子串	×	×
2.14.7	LEN	返回字符串的长度（字符个数）或者数据类型的长度（字节数）	×	×
2.14.8	MID	返回一个字符串的子字符串	×	×
2.14.9	RIGHT	返回字符串从右开始指定字符个数的子串	×	×
2.14.10	STR	将一个数转换成字符串	×	×
2.14.11	UCASE	将字符串中的字母全部转变成大写字母返回	×	×
2.14.12	VAL	将字符串转换成一个数值	×	×
2.14.13	PARSESTR	按用户指定的分隔符号号解析字符串	×	×

2.14.1 ASC

语法：value=ASC(string[,position])

描述：返回字符串中字符的 ASCII 码

参数：string　　字符串

position　需返回 ASCII 码字符在字符串中的位置，缺省值为 1

例程

```
PRINT ASC("A")          '结果为 65
PRINT ASC("ABC",1)      '打印第一个字母 A 的 ASCII 码，结果为 65
PRINT ASC("ABC",2)      '打印第二个字母 B 的 ASCII 码，结果为 66
PRINT ASC("ABC",3)      '打印第三个字母 C 的 ASCII 码，结果为 65
PRINT ASC("ABC")        '缺省位置值为 1，即打印 A 的 ASCII 码，结果为 65
```

2.14.2 CHR

语法：value=CHR(number)

描述：返回用 ASCII 码表达的值对应的字符

参数：number　ASCII 码值

例程

```
PRINT CHR(97)      '97 对应的字符为 a，打印结果为 a
PRINT CHR(65)      '65 对应的字符为 A，打印结果为 A
```

ASCII 码表

32	空格	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_	127	

2.14.3 HEX

语法：value=HEX(number [,digits])

描述：返回数值的十六进制结果

参数：number 数值

digits 返回由低位元到高位的位元数

例程

```
'十进制 54321 对应的十六进制数为 D431
Print Hex(54321)          '打印结果为 D431
Print Hex(54321, 2)       '打印结果为 31
Print Hex(54321, 5)       '打印结果为 0D431
```

2.14.4 INSTR

语法：value=INSTR([start,] string, [Any] substring)

描述：查找字符串中第一次出现的字符或者字符串

参数：start 从第几个字符开始查找

string 在 string 这个字符串中查找字符或字符串

Any 加上这个关键词后，string 中先找到 substring 中的任意一个字符就会

返回相应值

substring 需查找的字符或字符串

例程

```
Print InStr(2,"abcdefg", "a")     '打印信息为 0，因从字符串的第 2 位开始找，找不到 a，返回
0
Print InStr("abcdefg", "de")      '打印信息为 4，第 4 位找到 de
Print InStr("abcdefg", "h")       '打印信息为 0，字符串中没有 h
Print InStr("abcdefg", Any "fbc")'打印信息为 2，因加了 any 关键词，所以先找到 b，b 为第 2
位
```

2.14.5 LCASE

语法：value=LCASE(string)

描述：将字符串中的字母全部转变成小写字母返回

参数：string 需要转换的字符串

例程

```
Print Lcase("AeeE")     '打印结果为 aeee
```

2.14.6 LEFT

语法：value=LEFT(string,number)

描述：返回字符串从左开始指定字符个数的子串

参数：string 需要转换的字符串

number 字符个数

例程

```
Print LEFT("Hello Advantech",5)     '打印信息为 Hello
```

2.14.7 LEN

语法：value=LEN(expression)

描述：返回字符串的长度（字符个数）或者数据类型的长度（字节数）

参数：expression 如果是字符串，返回字符个数；如果是数据类型，返回字节数

例程

```
Print Len("hello world") '打印结果为 11，共 11 个字符
Print Len(Integer)       '打印结果为 4，integer 这个数据类型为 4 个字节
```

2.14.8 MID

语法：value=MID(string，start [,number])

描述：返回一个字符串的子字符串

参数：string　需要转换的字符串

start　返回的子字符串的起始转换位

number 子字符串的字符个数。如不填，则返回从 start 位后的所有字符

例程

```
Print Mid("abcdefg", 3, 2)  '打印结果为 cd
Print Mid("abcdefg", 3)     '打印结果为 cdefg
Print Mid("abcdefg", 2, 1)  '打印结果为 b
```

2.14.9 RIGHT

语法：value=RIGHT(string,number)

描述：返回字符串从右开始指定字符个数的子串

参数：string　需要转换的字符串

number　字符个数

例程

```
Print RIGHT("Hello Advantech",9)       '打印信息为 Advantech
```

2.14.10 STR

语法：value=STR(Numeric)

描述：将一个数转换成字符串

参数：Numeric 数值表达式

例程

```
VR(100)=100.32
PRINT STR(VR(100))    '打印结果为字符串"100.32"
```

2.14.11 UCASE

语法：value=UCASE(string)

描述：将字符串中的字母全部转变成大写字母返回

参数：string　需要转换的字符串

例程

```
Print Ucase("AeeE")    '打印结果为 AEEE
```

2.14.12 VAL

语法：value=VAL(string)

描述：将字符串转换成一个数值，字符串转换将从左到右按字符转换，如果先遇到非数值的字符，转换出来的数值将是 0。

参数：string 字符串

例程

```
DIM AS STRING str1,str2
str1="e3t"            '因先遇到非数值字符 e，所有打印结果为 0
str2="325.32"
PRINT VAL(str1),VAL(str2)    '打印结果为 0,325.32
```

2.14.13 PARSESTR

语法：NumStr=ParseSTR(StrInput, StrTokens( ), StrDelimits)

描述：按用户指定的分隔符号号解析字符串。

参数：StrInput　　输入的需分隔的字符串

StrTokens( )　存放分隔出的有效字符串数组

StrDelimits　　指定的分隔符号号

返回值：NumStr　　分隔出的有效字符串个数。**类型：**ULONG

例程

```
Dim StrInput as string = "Hi,MAS,Controller,!"
Dim StrDelimits as string = ","
Dim NumStr as ULONG
Dim StrTokens(0 to 3) as string
NumStr = ParseStr(StrInput, StrTokens(), StrDelimits)
print "num = ", NumStr        '打印出 num=4，有效分隔出 4 个字符串
Dim i as Integer = 0
for i= 0 to (NumStr-1)
    print  StrTokens(i)       '字符串数组依次打印出 HiMASController!
next i
```

2.15 工艺模块指令

2.15.1 气/油缸控制

气/油缸在自动化设备中非常常见，很好的对气/油缸进行控制在系统开发中显得很重要。本章节介绍了Motion BASIC 简单易使用的气/油缸控制指令 ,通过简单配置 ,可以很方便的实现设备中常见的气/油缸控制。为简要说明，本章节指令说明中统一用“气缸”来代替“气/油缸”，“用气缸前进”、“气缸后退来”表示气缸动作的两个方向运动。

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.15.1.1	CYL_BASE	该指令后面所有的气缸指令和参数设置、读取都基于该指令选定的气缸	√	×
2.15.1.2	CYL_FwDoneType	气缸前进到位方式	√	×
2.15.1.3	CYL_BwDoneType	气缸后退到位方式	√	×
2.15.1.4	CYL_FwTime	CYL_FwDoneType 中涉及到延时到位方式的延时时间	√	×
2.15.1.5	CYL_BwTime	CYL_BwDoneType 中涉及到延时到位方式的延时时间	√	×
2.15.1.6	CYL_FwAlmTime	气缸前进开始到到位的最大时间	√	×
2.15.1.7	CYL_BwAlmTime	气缸后退开始到到位的最大时间	√	×
2.15.1.8	CYL_FwEncValue	气缸前进到位编码器值	√	×
2.15.1.9	CYL_BwEncValue	气缸后退到位编码器值	√	×
2.15.1.10	CYL_Status	气缸当前状态	√	×
2.15.1.11	CYL_AlmReset	复位气缸的状态到复位状态	√	×
2.15.1.12	CYL_Move	执行气缸前进或后退动作	√	×
2.15.1.13	CYL_Stop	停止气缸动作	√	×

2.15.1.1 CYL_BASE

语法：CYL_BASE (cyl_no)[,secondcyl][,third cyl] …

描述：为了简化编程，可以用该指令选择要参与运动的气缸号，其后的指令就没必

要填写所有气缸控制的参数，只填写参与运动的气缸参数即可。气缸号要按顺序填写，气缸号可以是 1 个，也可以是 2 个、3 个…

参数： cyl_no 气缸号，由硬件配置决定对应的实体气缸控制；**范围：**根据控制器实际硬件决定

例程

```
'控制单个气缸
CYL_BASE 1
CYL_MOVE 1              '气缸 1 执行前进动作
WAIT CYLDONE            '等待气缸 1 动作到位完成

'控制多个气缸
CYL_BASE 2,3
CYL_MOVE 1,0            '气缸 2,3 分别执行前进、后退动作
Wait CYLDONE            '等待气缸 2,3 动作到位完成
```

2.15.1.2 CYL_FwDoneType

语法：CYL_FwDoneType= value

类型：ULONG

描述：设置/读取气缸前进到位方式

范围 :如下设定值，默认值 0

0：延时到位：气缸动作后，延时指定时间到，即认为气缸动作到位

1：限位到位：气缸动作后，遇到指定限位有效，即认为气缸动作到位

2：（限位+延时）到位：气缸动作后，遇到指定限位有效，再延时指定时间后，即认为气缸动作到位

3：（延时+限位）到位：气缸动作后，延时指定时间后，再检测到指定限位有效，即认为气缸动作到位

4：编码器到位：气缸动作后，编码器到限定数值后，即认为气功动作到位

例程

```
CYL_BASE 0
CYL_FwDoneType=0 '设置气缸 0 前进到位方式为延时到位
```

2.15.1.3 CYL_BwDoneType

语法：CYL_BwDoneType= value

类型：ULONG

描述：设置/读取气缸后退到位方式

范围：如下设定值，默认值 0

0：延时到位：气缸动作后，延时指定时间到，即认为气缸动作到位

1：限位到位：气缸动作后，遇到指定限位有效，即认为气缸动作到位

2：（限位+延时）到位：气缸动作后，遇到指定限位有效，再延时指定时间后，即认为气缸动作到位

3：（延时+限位）到位：气缸动作后，延时指定时间后，再检测到指定限位有效，即认为气缸动作到位

4：编码器到位：气缸动作后，编码器到限定数值后，即认为气功动作到位

例程

```
CYL_BASE 0
CYL_BwDoneType=0 '设置气缸 0 后退到位方式为延时到位
```

2.15.1.4 CYL_FwTime

语法：CYL_FwTime= value

类型：ULONG

描述：设置/读取 CYL_FwDoneType 中涉及到延时到位方式的延时时间。

范围：ULONG 类型范围，默认值 5000（ms）

例程

```
CYL_BASE 0
CYL_FwTime =1000 '设置气缸 0 前进到位方式中的延时时间为 1000 毫秒
```

2.15.1.5 CYL_BwTime

语法：CYL_BwTime= value

类型：ULONG

描述：设置/读取 CYL_BwDoneType 中涉及到延时到位方式的延时时间。

范围：ULONG 类型范围，默认值 5000（ms）

例程

```
CYL_BASE 0
CYL_BwTime =1000 '设置气缸 0 后退到位方式中的延时时间为 1000 毫秒
```

2.15.1.6 CYL_FwAlmTime

语法：CYL_FwAlmTime= value

类型：ULONG

描述：设置/读取气缸前进开始到到位的最大时间，如超过该时间前进动作还未到位，会发生内部报警，CYL_Status 属性值变为 9：气缸到位超时。

范围：ULONG 类型范围，默认值 20000（ms）

例程

```
CYL_BASE 0
CYL_FwAlmTime =1000 '设置气缸 0 前进最大到位时间为 1000 毫秒
```

2.15.1.7 CYL_BwAlmTime

语法：CYL_BwAlmTime= value

类型：ULONG

描述：设置/读取气缸后退开始到到位的最大时间，如超过该时间前进动作还未到位，会发生内部报警，CYL_Status 属性值变为 9：气缸到位超时。

范围：ULONG 类型范围，默认值 20000（ms）

例程

```
CYL_BASE 0
CYL_BwAlmTime =1000 '设置气缸 0 后退最大到位时间为 1000 毫秒
```

2.15.1.8 CYL_FwEncValue

语法：CYL_FwEncValue= value

类型：Double

描述：设置/读取气缸前进动作的到位方式为编码器到位时，指定的到位编码器值。

范围：Double 类型范围，默认值 0

例程

```
CYL_BASE 0
CYL_FwEncValue =1000 '设置气缸 0 前进到位编码器值为 1000 个 UNIT
```

2.15.1.9 CYL_BwEncValue

语法：CYL_BwEncValue= value

类型：Double

描述：设置/读取气缸后退动作的到位方式为编码器到位时，指定的到位编码器值。

范围：Double 类型范围，默认值 0

例程

```
CYL_BASE 0
CYL_BwEncValue =1000 '设置气缸 0 后退到位编码器值为 1000 个 UNIT
```

2.15.1.10 CYL_Status

语法：value=CYL_Status（只读）

类型：ULONG

描述：读取气缸当前状态。状态为 9 时，气缸不能再正常执行动作，要用 CYL_AlmReset 指令复位气缸状态后才能正常控制气缸动作。

返回值：如下

0：复位：原始状态。执行 CYL_Stop、CYL_AlmReset 后的气缸状态都为复位状态

1：前进到位：

2：后退到位

3：前进中

4：后退中

5：保留

6：保留

7：保留

8：保留

9：超过到位时间报警

例程

```
Dim A As ULONG
CYL_BASE 0
A =CYL_Status  '将气缸 0 的当前状态赋值给变量 A
```

2.15.1.11 CYL_AlmReset

语法 1：CYL_AlmReset

语法 2：CYL_AlmReset CYL(no)

描述：BASE 气缸列表的气缸或指定气缸，复位气缸的状态到复位状态。只有当气缸状态为报警状态时，下该指令才会到复位状态，该指令对其它状态不起作用。

参数：no 气缸号；**范围：**根据控制器实际硬件决定。

例程

```
CYL_BASE 0,1,2
CYL_AlmReset          '复位气缸 0、1、2 的状态
CYL_AlmReset cyl(1)   '复位气缸 1 的状态
```

2.15.1.12 CYL_Move

语法 1：CYL_Move dir

语法 2：CYL_Move CYL(no) , dir

描述：BASE 气缸列表的气缸或指定气缸，执行气缸动作。

参数：dir 气缸动作方向。0：气缸后退； 1：气缸前进

no 气缸号；**范围：**根据控制器实际硬件决定。

例程

```
'控制单个气缸
CYL_BASE 1
CYL_MOVE 1            '气缸 1 执行前进动作
WAIT CYLDONE          '等待气缸 1 动作到位完成

'控制多个气缸
CYL_BASE 2,3
CYL_MOVE 1,0          '气缸 2,3 分别执行前进、后退动作
Wait CYLDONE          '等待气缸 2,3 动作到位完成
```

2.15.1.13 CYL_Stop

语法 1：CYL_Stop

语法 2：CYL_Stop CYL(no)

描述：BASE 气缸列表的气缸或指定气缸，停止气缸动作，同时会将该气缸的状态切到复位状态。

参数：no 气缸号；**范围：**根据控制器实际硬件决定。

注意：该指令仅适用对双线圈电磁阀控制的气缸控制，无法控制单线圈电磁阀对应的气缸停止。

例程

```
CYL_BASE 0,1,2,3
CYL_Stop                '停止气缸 0,1,2,3 的动作
CYL_Stop cyl(6)         '停止气缸 6 的动作
```

2.15.2 PATHLINK

本文主要说明如何实现 XYTable 追随传送带上的工件进行加工。加工动作包含：点胶，锁螺丝，取放等动作。追随加工的动作，我们简称 Pathlink。

整个操作流程大致如下：

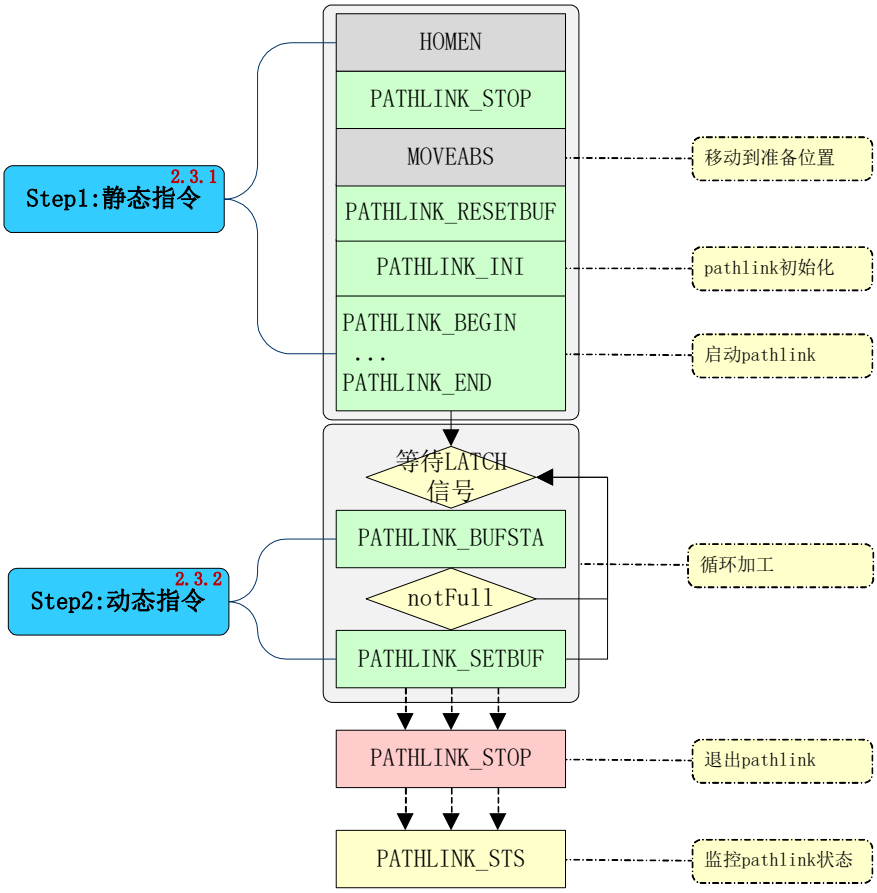


图 2.15.1 PATHLINK 操作流程图

静态指令：把只需操作一次的指令记为静态指令。

动态指令：把需要多次操作的指令记为动态指令。每来一个工件，就需要操作一次。

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.15.2.1	PATHLINK_INI	设置 PATH LINK 初始信息,包含同步位置设定, 图像 MARK 点坐标信息等	×	×
2.15.2.2	PATHLINK_BEGIN	Pathlink 插补轨迹路径起始符	×	×
2.15.2.3	PATHLINK_END	Pathlink 插补轨迹路径结束符	×	×
2.15.2.4	PATHLINK_SETBUF	设定加工时的每次相机拍照时得到 MARK 的位置和角度信息以及锁存位置(理论位置/编码器位置)写入 Buffer 中	×	×
2.15.2.5	PATHLINK_STOP	调用 Acm_PathLinkStop, 解除主轴和从轴的同步关系	√	×
2.15.2.6	PATHLINK_BUFSTATUS	用于获取加工时用于存储拍照时得到 MARK 信息和 latch 数据的 buffer 的状态	×	×
2.15.2.7	PATHLINK_RESETBUF	清空用于存储拍照时得到 MARK 信息和 latch 数据的 buffer	×	×
2.15.2.8	PATHLINK_STATUS	获取当前 PATHLINK 的运动状态	×	×
2.15.2.9	PATHLINK_RDYPOINT	计算 XYTable 跟随之前的等待位置	×	×

2.15.2.1 PATHLINK_INI

语法 :PATHLINK_INI AX(MasAxisNo), SYNINFO_SartVR [, MasOffsetPos] [, MARK1_SartVR] [,MAangle] [, PAngle]

描述：设置 PATH LINK 初始信息。包含同步位置数据，图像 MARK 点位置，各坐标系对应关系等。这些信息需在示教阶段获取，在程序启动配置阶段进行此部分的配置。需搭配 BASE 使用，例如 BASE 0,1 则有轴 0 和轴 1 建立了 XYTable。

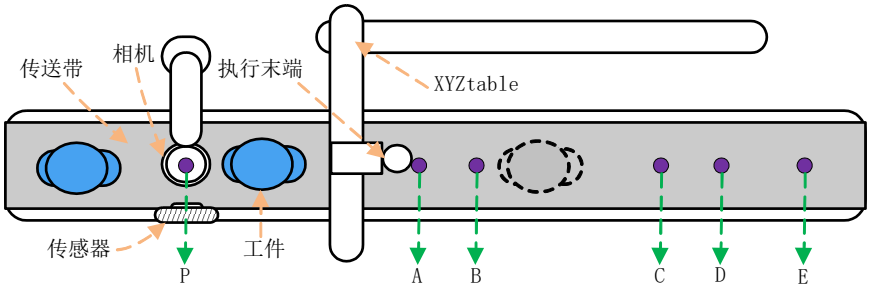


图 2.15.2 机台示意图

说明：

A：图像处理完毕后，启动同步关系

B：XYTable 开始涂胶

C：XYTable 涂胶完成

D：XYTable 停止

E：XYTable 回退等待位置

参数：AX(MasAxisNo)主轴号。

SYNINFO_SartVR: 同步关系曲线信息表对应的 VR 的起始 index，即 VR[SYNINFO_SartVR] ~ VR[SYNINFO_SartVR+3]为同步关系曲线设定信息，单位：PPU。

VR[SYNINFO_SartVR]: XYTable 从等待到开始加工(即进入同步)的这个过程主轴的移动距离，对应图 2.15.1 传送带上 A 到 B 的距离；

VR[SYNINFO_SartVR+1]: XYTable 从等待到加工完成主轴的移动距离，对应图 2.15.1 传送带上 A 到 C 的距离；

VR[SYNINFO_SartVR+2]: XYTable 从等待到减速停止的过程中主轴的移动距离，对应图 2.15.1 传送带上 A 到 D 的距离；

VR[SYNINFO_SartVR+3]: XYTable 从等待经过加工完成并返回到等待位置的过程中主轴的移动距离，对应图 2.15.1 传送带上 A 到 E 的距离；

MasOffsetPos: XY Table 开始进行同步时相对与 MARK 点的距离。缺省为 0.

对应图上 A 点位置相对与标定时 Mark 点的距离（P 点），单位 PPU。

MARK1_SartVR:标定时 MARK 点的示教位置信息所在 VR 起始位置。

即 VR[MARK1_SartVR]~ VR[MARK1_SartVR+2]。缺省则示教位置信息都为 0.

由相机拍照获得。

VR[MARK1_SartVR]：示教时 MARK 点所在世界坐标系（WCS）中的 X 位置。

VR[MARK1_SartVR+1]：示教时 MARK 点所在世界坐标系（WCS）中的 Y 位置。

VR[MARK1_SartVR+2]：示教时工件偏转(相对于标定)弧度。顺时针为正，逆时针为负。

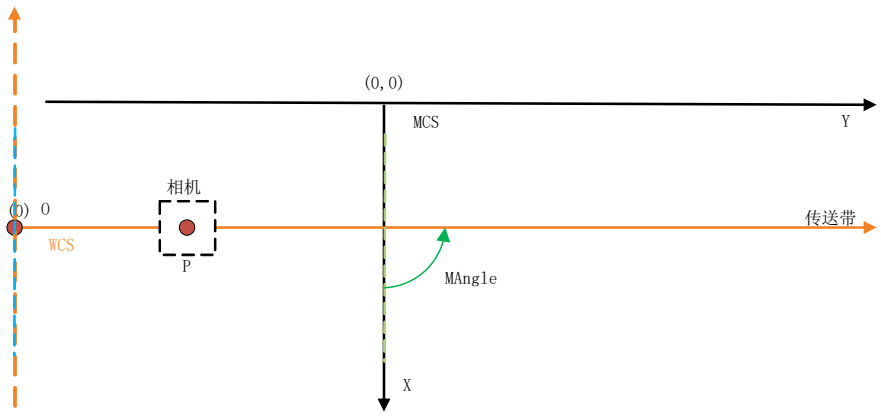


图 2.15.3 坐标系示意图

MAangle: WCS(世界坐标系)与 MCS(机械坐标系)之间的夹角弧度，单位为 rad。

缺省为 0.数值为由 MCS 的 X 轴正方向逆时针旋转到 WCS 的 X 轴正方向。逆时针为正。如图 5.15.2 所示，为 pi/2；

PAngle：PCS(工件坐标系)与 MCS(机械坐标系)之间的夹角弧度，单位为 rad。

缺省为 0.（注：只有加工 CAD 导入轨迹时，才使用此值）

例程

```
Dim As DOUBLE point_x, point_y
BASE 0,1 '轴 0,1 组成 XY Table
GVH= 20000
GACC = 100000
GDEC = 100000
'轴 2 为主轴,VR(0)~VR(3)存放同步数据{2000,20000,30000,40000}
'主轴运行到距离相机 Mark 点 1000 位置，XYTable 开始动作。
'VR(6)~VR(8)存放相机 Mark 点信息
'世界坐标系与机械坐标系夹角弧度值为 0.7853982.
PATHLINK_INI AX(2),0,1000,6,0.7853982,0
PATHLINK_RDYPOINT(point_x, point_y, 0)  '获取 XYTable 的等待位置,让 XYTable 移动到等待位置
MOVE point_x, point_y
WAIT DONE
PATHLINK_RESETBUF AX(2) '清空存放 latch 数据和 mark 信息的 buffer
PATHLINK_BEGIN AX(2) '设定加工轨迹
LINE 0,0
LINE 0,10000
LINE 10000,0
LINE 0,-10000
LINE -10000,0
```

```
PATHLINK_END
VR(20) = PATHLINK_STATUS
IF(VR(20)<1) then
    print "Pathlink failed."
END IF
```

2.15.2.2 PATHLINK_BEGIN

语法：PATHLINK_BEGIN AX(MasAxisNo)

描述：Pathlink 插补轨迹路径起始符。需结合 PATHLINK_END，设定加工轨迹，所有轨迹皆为相对位置，第一段需为加工件的起点相对于 MARK 点的距离，第二段为加工件的第二点相对于第一点的距离，依次类推。轨迹通常由示教所得，有两种示教方法：①机台示教；②图像示教。

参数：AX(MasAxisNo) 主轴号。

例程：参考 PATHLINK_INI **例程**

2.15.2.3 PATHLINK_END

语法：PATHLINK_ENDAX(MasAxisNo)

描述：Pathlink 插补轨迹路径结束符。结合 PATHLINK_BEGIN 设定加工轨迹，执行完此命令，XYTable 和主轴便建立了跟随关系。

参数：AX(MasAxisNo) 主轴号。

例程：参考 PATHLINK_INI **例程**

2.15.2.4 PATHLINK_SETBUF

语法：PATHLINK_SETBUF AX(MasAxisNo), MARK2_StartVR,LatchData

描述：设定加工时的每次相机拍照时得到 MARK 的位置和角度信息以及锁存位置(理论位置/编码器位置)写入 Buffer 中，每次开始加工时，会从 buffer 中读取 mark 信息和锁存位置，buffer 的空闲位置+1，buffer 总大小为 50。

参数：AX(MasAxisNo) 主轴号。

MARK2_StartVR：每次加工前由相机得到的 MARK 点的新位置和偏转角度信息所在 VR 的起始 index。如果不设定，则 VR[MARK2_SartVR]~ VR[MARK2_SartVR+2]都为 0.

VR[MARK2_SartVR]：加工时 MARK 点所在 MCS 中的 X 位置，单位：PPU。

VR[MARK2_SartVR+1]：加工时 MARK 点所在 MCS 中的 Y 位置，单位：PPU

VR[MARK2_SartVR+2]：加工时工件偏转弧度,相对于标定时的角度，单位：rad。

LatchData: 拍照时锁存到的传送带的位置(理论位置/实际位置)，单位：PPU

注 意：

1. 需搭配 BASE X,X 使用。用来指定 XYTable，在启用此指令之前，必须先设定 PATHLINK_INI 以及 PATHLINK_BEGIN，PATHLINK_END。
2. 调用此指令之前，需先通过调用 PATHLINK_BUFSTA 判断 buffer 是否满，如果满则无法写入 buffer。

例程：

```
VR(15) = PATHLINK_BUFSTATUS(AX(2))
IF VR(15)<1 THEN
BASE 0,1
PATHLINK_SETBUF AX(2),10,0
END IF
```

2.15.2.5 PATHLINK_STOP

语法 1：PATHLINK_STOP

语法 2：PATHLINK_STOP AX(MasAxisNo)

描述：解除主轴和从轴的同步关系，且插补运动都停止，主轴不停止。

参数：AX(MasAxisNo) 主轴号。

例程：参考 PATHLINK_INI **例程**

2.15.2.6 PATHLINK_BUFSTATUS

语法 1：PATHLINK_BUFSTATUS

语法 2：PATHLINK_BUFSTATUS AX(MasAxisNo)

描述：用于获取加工时用于存储拍照时得到 MARK 信息和 latch 数据的 buffer 的状态。

当 Buffer 满了之后，需等待有空闲之后再写入，Buffer 总大小为 50。

参数：AX(MasAxisNo) 主轴号。

返回值： 0—未满，尚有空间，1—已满，不可再写入。

例程：参考 PATHLINK_SETBUF **例程。**

2.15.2.7 PATHLINK_RESETBUF

语法 1：PATHLINK_RESETBUF

语法 2：PATHLINK_RESETBUF AX(MasAxisNo)

描述：清空用于存储拍照时得到 MARK 信息和 latch 数据的 buffer。

参数：AX(MasAxisNo) 主轴号。

例程：参考 PATHLINK_INI **例程。**

2.15.2.8 PATHLINK_STATUS

语法 1：PATHLINK_STATUS

语法 2：PATHLINK_STATUS AX(MasAxisNo)

描述：获取当前 PATHLINK 的运动状态，方便与其他工艺配合运动。

参数：AX(MasAxisNo) 主轴号。

返回值：

0 未进入 pathlink

1 等待工件，pathlink 已启动，但是相机未识别到未加工的工件

2 等待加工，相机识别到工件，但是还没有到达加工位置

3 加速区，加速到与传送带同步

4 同步区，加工过程中

5 减速区，加工完成，XYTable 减速过程中

6 回程区，XYTable 回到等待位置

7 异常，运动过程中异常停止。已退出同步区，但 path 还没有走完即执行末端到达 C 的时，path 还没有走完，则报警。

例程：参考 PATHLINK_INI **例程**。

2.15.2.9 PATHLINK_RDYPOINT

语法：PATHLINK_RDYPOINTpoint_x, point_y[, mark_offset]

描述：在 PATHLINK_INI 之后，启动 pathlink 之前，用此指令可获取等待位置，用于将执行末端移动指定位置，即等待位置。

参数：point_x 返回得到的等待位置的 X 坐标值，单位 PPU

Point_y 返回得到的等待位置的 Y 坐标值，单位 PPU

mark_offset 输入相机标定位置的偏移量，单位 PPU

例程：参考 PATHLINK_INI **例程**

2.16 全局变量 VR、Table

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.16.1	VR	实数型全局 VR 变量	√	√
2.16.2	FILE_WRITEVR	写 VR 文件到本地	×	×
2.16.3	FILE_READVR	将本地 VR 档载入专案	×	×
2.16.4	VRCopy	将一段 VR 区域的数据拷贝到另一段 VR 区域	×	×
2.16.5	VRExchange	将一段 VR 区域的数据与另一段长度相等 VR 区域的数据做交换	×	×
2.16.6	VRClear	将一段 VR 区域的数值清零	×	×
2.16.7	StrToVR	将字符串塞入一段 VR 区域	×	×
2.16.8	VRToStr	将一段 VR 区域转成字符串	×	×
2.16.9	VRRESET	将指定区域 VR 值设为初始值	×	√
2.16.10	Table	实数型全局 Table 变量	√	√
2.16.11	Tab 类	Table 变量映射的二维表格类	×	×

2.16.1 VR

语法：VR(no)=value

类型：Double

描述：VR 变数是实数型全局变量。软件平台共提供 10000 个 VR 变量给用户操作：VR(0)~VR(9999)。当 VR 变数用于 Modbus 通讯的自定义变量时，可以选择将 VR 变量对应一个 16 位数据类型寄存器或 32 位数据类型寄存器。

参数：no VR 变量的索引号；**范围：**0~9999，共 10000 个

例程

```
Dim A As ULONG
A=15
VR(A)=200.525 '将 200.525 赋值给 VR(15)
BASE 0
VR(25)=1000
```

```
VL=VR(25)        '将 VR(15)的数值赋给轴 0 的初速度。
```

2.16.2 FILE_WRITEVR

语法：FILE_WRITEVR file_name，vr_start no，vr_end no

描述：将一段 VR 的资料保存到本地文本，目前仅支持 bas 和 csv 两种文件类型。保存的路径为“Motion Studio 安装路径\Advantech\Motion_Runtime\AMI\AMI_User_Files”。用户不能自己指定路径保存文件。

参数：file_name 保存到本地文本的文件名

vr_start no　VR 起始编号

vr_end no　VR 结束编号

例程

```
'将 VR(0)~VR(20)的数据写到本地名为 VR_data 的 bas 文件
FILE_WRITEVR "VR_data.bas",0,20
'将 VR(0)~VR(100)的数据写到本地名为 P_data 的 csv 文件
FILE_WRITEVR "P_data.csv",0,100
```

2.16.3 FILE_READVR

语法：FILE_READVR file_name

描述：将本地 VR 文本的数据写到当前工程的 VR 变量中。读取路径为“Motion Studio 安装路径\Advantech\Motion_Runtime\AMI\AMI_User_Files”。用户不能自己指定路径读取文件。

参数：file_name 保存到本地文本的文件名

例程

```
'将本地名为 VR_data 的 bas 文件 VR 数据读到控制器里，该 bas 文件里的 VR 数据将覆盖控制器里对应的 VR 数据
FILE_READVR "VR_data.bas"
'将本地名为 P_data 的 csv 文件 VR 数据读到控制器里，该 csv 文件里的 VR 数据将覆盖控制器里对应的 VR 数据
FILE_READVR "P_data.csv"
```

2.16.4 VRCopy

语法：VRCopy src_vr_start ,dst_vr_start,count

描述：将一段 VR 区域的数据拷贝到另一段 VR 区域

参数：src_vr_start　　VR 源区域起始编号；**类型：**ULONG

dst_vr_start　　VR 目标区域起始编号；**类型：**ULONG

count　　拷贝 VR 的个数；**类型：**ULONG

例程

```
'将 VR(17),VR(18),VR(19),VR(20)的数据拷贝到 VR(30),VR(31),VR(32),VR(33)
VR(17)=1
VR(18)=2.2
VR(19)=3
VR(20)=4.5
VRCopy(17,30,4)    '拷贝 VR(17)~VR(20)的数据到 VR(30)~VR(33)
Print VR(30)       'VR(30)的数值为 1
Print VR(31)       'VR(31)的数值为 2.2
Print VR(32)       'VR(32)的数值为 3
Print VR(33)       'VR(33)的数值为 4.5
```

2.16.5 VRExchange

语法：VRExchangesrc_vr_start ,dst_vr_start,count

描述：将一段 VR 区域的数据与另一段长度相等 VR 区域的数据做交换

参数：src_vr_start　　VR 源区域起始编号；**类型：**ULONG

dst_vr_start　　VR 目标区域起始编号；**类型：**ULONG

count　　拷贝 VR 的个数；**类型：**ULONG

例程

```
'将 VR(17),VR(18),VR(19),VR(20)的数值与 VR(30),VR(31),VR(32),VR(33)的数据做交换
VR(17)=1
VR(18)=2
VR(19)=3
VR(20)=4
VR(30)=6
VR(31)=7
VR(32)=8
VR(33)=9
VRExchange(17,30,4)   '将 VR(17)~VR(20)的数据与 VR(30)~VR(33)的数据做交换
Print VR(17)       'VR(17)的数值为 6
Print VR(18)       'VR(18)的数值为 7
Print VR(19)       'VR(19)的数值为 8
Print VR(20)       'VR(20)的数值为 9
Print VR(30)       'VR(30)的数值为 1
Print VR(31)       'VR(31)的数值为 2
Print VR(32)       'VR(32)的数值为 3
Print VR(33)       'VR(33)的数值为 4
```

2.16.6 VRClear

语法：VRClear vr_start ,count

描述：将一段 VR 区域的数值清零

参数：vr_start　　VR 区域起始编号；**类型：**ULONG

count　　拷贝 VR 的个数；**类型：**ULONG

例程

```
'将 VR(17),VR(18),VR(19),VR(20)的数值清零
VRClear(17,4)
Print VR(17)       'VR(17)的数值为 0
Print VR(18)       'VR(18)的数值为 0
Print VR(19)       'VR(19)的数值为 0
Print VR(20)       'VR(20)的数值为 0
```

2.16.7 StrToVR

语法：StrToVR StrInput,vr_start

描述：将字符串依次拆分成单个字符，以 ASCII 码值塞入以 vr_start 为起始的 VR 区域，一个 VR 变量对应一个字符。

参数：StrInput　　需转换的字符串；**类型：**String

vr_start　　存放转换后字符的 VR 区域起始编号；**类型：**ULONG

例程

```
'将字符串"mas"拆分成字符塞入以 VR(0)起始的 VR 区域
StrToVR("mas",0)
Print VR(0)        'VR(0)的数值为 109，"m"对应的 ASCII 码值为 109
Print VR(1)        'VR(1)的数值为 97，"a"对应的 ASCII 码值为 97
Print VR(2)        'VR(2)的数值为 115，"s"对应的 ASCII 码值为 115
```

2.16.8 VRToStr

语法：outstr=VRToStr (vr_start,count)

描述：将一段 VR 区域的每个 VR 值对应的字符组成字符串返回。

参数：vr_start　　VR 区域起始编号；**类型：**ULONG

　　count　　存放转换后字符的 VR 区域起始编号

返回值：组成的字符串；**类型：**String

例程

```
Dim A as string
StrToVR("mas",0)  '将“mas”字符串塞入 VR(0)~VR(2)
Print VR(0)       'VR(0)的数值为 109，"m"对应的 ASCII 码值为 109
Print VR(1)       'VR(1)的数值为 97，"a"对应的 ASCII 码值为 97
Print VR(2)       'VR(2)的数值为 115，"s"对应的 ASCII 码值为 115
A=VRToStr(0,3)    '将 VR(0)~VR(2)值对应的字符组成字符串返回给 A 变数
Print A           'A 打印出来为“mas”
```

2.16.9 VRRESET

所属：命令

语法：VRReset　vr_start_index, cnt

类型：ULONG

描述：将指定区域的 VR 进行初始值赋值给当前值

参数：vr_start_index：需操作 VR 区域的起始 VR 索引。

　　cnt：需操作 VR 区域的 VR 个数。

注意：该指令只能对 VR 表工具中存在的 VR 进行处理。

例程

```
VRRESET 100,20 '将 100~119 这 20 个 VR 的初始值赋值给对应 VR 的当前值。
```

2.16.10 Table

语法：Table(no)=value

类型：Double

描述：Table 变量是实数型全局变量。软件平台共提供 40 万个 Table 变量给用户操作：Table(0)~Table(399999)。Table 变量的作为 1 维数据使用时和 VR 变量的使用方法一样，但是 Table 变量在内部可以映射成 2 维数据使用，可以参考手册内 Tab 类指令说明。

参数：no Table 变量的地址索引号；**范围：**0~399999，共 40 万个

例程

```
Dim A As ULONG
A=15
Table(A)=200.525  '将 200.525 赋值给 Table(15)
BASE 0
Table(25)=1000
VL=Table(25)      '将 Table(15)的数值赋给轴 0 的初速度。
```

2.16.11 Tab

Table 变量映射的二维表格类。详情请参考“模块类”章节的 Tab 类说明。

2.17 文件操作

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.17.1	GetFilesCnt	读取目标文件夹下指定文件类型的档个数	×	×
2.17.2	GetFileName	读取目标文件夹下指定索引的文件名	×	×
2.17.3	File_delete	删除默认路径或指定路径下的文件	×	×
2.17.4	File_Find	读取指定文件名在文件夹中的索引	×	×
2.17.5	File_Rename	重命名指定文件夹下的指定文件名	×	×
2.17.6	File_Copy	拷贝指定文件夹下的指定文件，并命名新拷贝出来的文件	×	×
2.17.7	File_Open	打开一个文件，以便对该文件进行读写操作。	×	×
2.17.8	File_Close	关闭一个文件操作	×	×
2.17.9	File_Error	读取文件读写过程中发生的错误信息	×	×
2.17.10	Input	读取纯文本类型文件的内容	×	×
2.17.11	Write	写内容到纯文本类型的文件	×	×
2.17.12	Put	写内容到二进制类型的文件	×	×
2.17.13	Get	读二进制类型文件的内容	×	×

2.17.1 GetFilesCnt

语法：value=GetFilesCnt ([extension][,isFullName][,folder_path])

描述：读取目标文件夹下指定文件类型的档个数。

参数：extension　文件扩展名；**类型：**String

isFullName　决定 GetFileName 指令取得的文件名是否要包含扩展名；
0：不包含扩展名
1：包含扩展名

folder_path　目标文件夹的路径；**类型：**String

返回值：指定文件类型的个数；**类型：**ULONG

注　意：文件夹路径不允许有中文。文件夹路径可以写绝对路径，也可以写相对路径。相对路径的默认路径为：C:\Advantech\Motion_Runtime\AMI\AMI_User_Files

例程

```
DIM A AS ULONG
A=GetFilesCnt()   '读默认路径 C:\Advantech\Motion_Runtime\AMI\AMI_User_Files 下的文件个数
A=GetFilesCnt(".txt")'读默认路径下的 txt 文件类型的文件个数
A= GetFilesCnt(".txt",0,"/abc")'读默认路径下名为 abc 文件夹下的 txt 文件类型的文件个数
A= GetFilesCnt(".txt",0,"C:\Users\Administrator\Desktop\FILE_TEST")'用绝对路径读文件，读桌面 FILE_TEST 文件夹下的 txt 文件类型的文件个数。
```

2.17.2 GetFileName

语法：value =GetFileName (index)

描述：读取目标文件夹下指定索引的文件名。

参数：index　文件索引；**类型：**ULONG

返回值：指定档的文件名；**类型：**String

注意：该指令读出来的文件名字符串是否包含扩展名是由 GetFilesCnt 中的参数 isFullName 决定的；该指令索引的文件夹路径是由 GetFilesCnt 中的参数 folder_path 确定的；该指令参数 index 的编号是 windows 系统自动分配的，所以一般会把指定类型的档全部读出再针对文件名做操作。综上所述，该指令需配合 GetFilesCnt 指令使用。

例程

```
'如默认路径下有 3 个类型为 txt 的档，名称分别为 a1，a2，a3
DIM A AS ULONG
DIM txt_filename(0 to 2) AS STRING
DIM i AS INTEGER
A=GetFilesCnt(".txt") '读默认路径下的 txt 文件类型的文件个数
FOR i=0 to 2
    txt_filename(i)=GetFileName(i)
    Print txt_filename(i)        'For 循环打印出来 3 个文件名为 a1，a2，a3
NEXT i
```

2.17.3 File_delete

语法：File_delete (file_path)

描述：删除默认路径或指定路径下的文件。

参数：file_path　文件路径；**类型：**String

例程

```
FILE_DELETE("a1.txt") '删除默认路径下的“a1.txt”文件
```

```
FILE_DELETE("C:\Users\Administrator\Desktop\FILE_TEST\a1.txt") '删除指定路径下的“a1.txt”文件
```

2.17.4 File_Find

语法：value =File_Find (filename_list( ),filename)

描述：读取指定文件名在文件夹中的索引

参数：filename_list( ) 目标文件夹中文件名列表；**类型：**String

filename 要读取档的文件名；**类型：**String

返回值：读取文件名在文件夹中的索引；**类型：**ULONG

例程

```
'如默认路径下有 3 个类型为 txt 的档，名称分别为 a1, a2, a3
DIM A AS ULONG
DIM txt_filename(0 to 2) AS STRING
DIM i AS INTEGER
A=GetFilesCnt(".txt") '读默认路径下的 txt 文件类型的文件个数
FOR i=0 to 2
    txt_filename(i)=GetFileName(i)
    Print txt_filename(i)        'For 循环打印出来 3 个文件名为 a1，a2，a3
NEXT i
A=File_find(txt_filename( ),"a3")
Print A            '打印出 A 的值为 2，代表 a3 这个档在 txt_filename( )数组中的索引为 2
```

2.17.5 File_Rename

语法：File_Rename src_filename,dst_filename

描述：重命名指定文件夹下的指定文件名

参数：src_filename 需重命名的档的文件名；**类型：**String

dst_filename 新的文件名；**类型：**String

例程

```
'如默认路径下有 3 个类型为 txt 的档，名称分别为 a1, a2, a3
DIM A AS ULONG
A=GetFilesCnt(".txt") '读默认路径下的 txt 文件类型的文件个数
File_Rename "a3.txt","b3.txt"'默认路径下的 a3.txt 档重命名为 b3.txt
```

2.17.6 File_Copy

语法：File_Copy src_filename,dst_filename

描述：拷贝指定文件夹下的指定文件，并命名新拷贝出来的文件

参数：src_filename 需被拷贝文件的文件名；**类型：**String

dst_filename 新拷贝出的文件名；**类型：**String

例程

```
'如默认路径下有 3 个类型为 txt 的档，名称分别为 a1, a2, a3
DIM A AS ULONG
A=GetFilesCnt(".txt")        '读默认路径下的 txt 文件类型的文件个数
File_Copy "a3.txt","b3.txt"'默认路径下的 a3.txt 文件拷贝一份，并将拷贝出来的文件命名为
b3.txt
```

2.17.7 File_Open

语法：File_Open file_no, filename [,file_type, read_write, encoding_type]

描述：指定一个编号，打开一个文件。以便对文件进行读写，读写操作完后需使用 File_Close 进行关闭。打开文件后，才可以对该文件进行读写操作。编号类似于句柄，在 1~255 间随意指定。

参数：file_no 文件操作的编号； **类型：**ULONG ； **范围：**1~255

filename需操作文件的路径名；路径为相对 Motion Runtime 所在目录 AMI\AMI_User_Files\下的路径。**类型：**String

file_type 文件类型。默认是纯文本类型文件。

0（FILE_TYPE_TEXT）： 纯文本类型文件。读用指令 Input，写用指令 Write。

1(FILE_TYPE_BINARY)： 二进制类型文件。读用指令 Put，写用指令 Get。

read_write 指定接下来的读写动作。。当文件不存在时，选择“只读”时会产生错误信息。选择“只写”或者“可读可写”时会创建新文件。默认是可读可写。

纯文本类型文件选择“只读”后，需 File_Close 操作后，重新打开再选择“只写”，才能对文件进行写的操作。同样，选择“只写”后，也需关闭文件操作后重新打开文件选择“只读”，才可以进行读操作。即纯文本类型文件该项不能选择“可读可写”，打开一次文件后，只能进行“读”或“写”的操作。

0(READ_ONLY)：只读；

1(WRITE_ONLY)：只写；

2(READ_WRITE)：可读可写。纯文本类型文件不允许选此项。

encoding_type 指定文本的编码格式。该参数只对纯文本类型文件起作用。缺省值是 ASCII。

0 (ENCODE_ASCII)：ASCII

1 (ENCODE_utf8)：UTF8

2(ENCODE_utf16)：UTF16

3(ENCODE_utf32)：UTF32

例程

```
File_Open  1,"test1.txt",0,1,0  '打开 test1.txt 纯文本类型文件，接下来的操作为只写。
FILE_Close  1      '关掉编号为 1 的文件操作。
```

2.17.8 File_Close

语法：File_Close file_no

描述：指定一个编号，关闭已打开的文件操作。

参数：file_no 文件操作的编号； **类型：**ULONG ； **范围：**1~255

例程

```
File_Open  1,"test1.txt",0,1,0  '打开 test1.txt 纯文本类型文件，接下来的操作为只写。。
FILE_Close  1      '关掉编号为 1 的文件操作。
```

2.17.9 File_Error

语法：value=File_Error

描述：读取文件读写过程中发生的错误信息。

返回值：

0：没有错误

1：非法的函数调用

2：没有找到文件

3：文件读写操作错误

4：没有权限

5：文件已到结尾

例程

```
Dim f As Integer
f = 250
DIM fileName AS STRING =  "filetest.txt"
FILE_OPEN f, fileName,FILE_TYPE_TEXT, Read_ONLY
WRITE #f, 1,"Test,Data"
VR(0) = FILE_ERROR    '将最近一次文件读写过程中发生的错误信息返回值放入 VR(0)
FILE_CLOSE
```

2.17.10 Input

语法：Input #file_no, variable_list

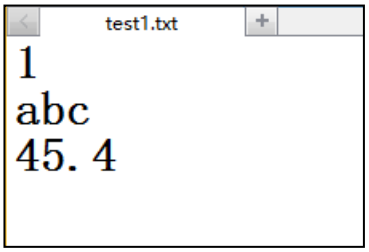
描述：读取纯文本类型文件的内容。

参数：#file_no 文件操作的编号，编号前需加“#”

variable_list 变量列表 ,用逗号分隔。用来获取文件中的值。变量类型根据所需获取的变量类型而定义，如果文件类型为文本类型，通常以逗号或换行符将文本分割，变量列表会跟分割后的文本字串一一对应，并会将字串进行转换为所需数据类型

例程

如下图，Motion Runtime 所在目录 AMI\AMI_User_Files\下有一个“test1.txt”文件。文件的内容如图。




```
Dim f As Integer=1
Dim a as ULONG
dim b as STRING
dim c as DOUBLE
FILE_OPEN f, "test1.txt",0,0,0  '打开"test1.txt"文件，将进行"读"操作
'读上三个变量放入 a、b、c
INPUT #f,a,b,c
PRINT a,b,c             '打印出来得到 1 、abc 、45.4
FILE_CLOSE              '关闭编号为 1 的文件操作
```

2.17.11 Write

语法：Write #file_no [, expressionlist]

描述：写内容到纯文本类型的文件。

参数：#file_no 文件操作的编号，编号前需加“#”

expressionlist 写入文件中的表达式列表，以逗号分隔。表达式列表可以是双引号的字符串，也可以是某类型变量，或常数。各表达式在文本文件中以逗号分隔。

例程

```
Dim f As Integer=1
Dim a as ULONG=5
dim b as STRING="Hi,MAS"
dim c as DOUBLE=11.11
```

```
FILE_OPEN f, "test1.txt",0,1,0  '打开"test1.txt"文件，将进行"写"操作
'将 a、b、c 3 个变量写进"test1.txt"文件
WRITE #f,a,b,c
PRINT a,b,c
FILE_CLOSE          '关闭编号为 1 的文件操作
```

2.17.12 Put

语法：Put #file_no ,[position] , data

描述：写内容到二进制类型的文件。

参数：#file_no 文件操作的编号，编号前需加“#”

position 指定文件开始写入的位置（字符长度），若缺省或 0 则从当前位置开始

data 要写入的内容，类型可为数组类型，字符串类型或数组

例程

```
Dim f As Integer=1
Dim a as byte=5
Dim b as byte
FILE_OPEN f, "test2.txt",1,2     '打开"test2.txt"二进制类型文件，将进行"可读可写"操作
'将 a 的值写进"test2.txt"文件
PUT #f,,a
sleep 500
'将"test2.txt"文件中当前位置的内容读上来放入 b
GET #f,,b
FILE_CLOSE          '关闭编号为 1 的文件操作
```

2.17.13 Get

语法：Get #file_no ,[position] , data

描述：读二进制类型文件的内容。

参数：#file_no 文件操作的编号，编号前需加“#”

position 指定文件开始读取的位置（字符长度），若缺省或 0 则从当前位置开始

data 用于存储读取到的值，可为数值类型，字符串类型或数组类型，如果为数组类型则需加上 (). 读取操

作会尽量填满 data，除非遇到 EOF

例程

```
Dim f As Integer=1
Dim a as byte=5
Dim b as byte
FILE_OPEN f, "test2.txt",1,2     '打开"test2.txt"二进制类型文件，将进行"可读可写"操作
'将 a 的值写进"test2.txt"文件
PUT #f,,a
sleep 500
'将"test2.txt"文件中当前位置的内容读上来放入 b
GET #f,,b
FILE_CLOSE          '关闭编号为 1 的文件操作
```

2.18 Robot 控制

Motion Studio 里提供了关节机械手控制的指令，目前仅支持 SCARA 机械手控制，所以本章提到的 Robot 目前是指 SCARA。Motion Studio 支持同时控制多个 Robot ,由 R_BASE 指令指定要操作的 Robot。MAS 控制器中一个 PCI-1245S-MAS 或 MVP-3245S-MAS 机械手控制单元对应一个 Robot。利用本章节的指令，用户可以很方便的控制 SCARA 完成 JOG、点位、直线插补、圆弧插补、路径连续插补等运动。

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.18.1	R_BASE	指定要操作的 Robot	√	×
2.18.2	R_ARM1	设置/读取 SCARA 机械手的第一个臂的长度	√	×
2.18.3	R_ARM2	设置/读取 SCARA 机械手的第二个臂的长度	√	×
2.18.4	R_ARM3	设置/读取 SCARA 机械手的第三个臂的长度	√	×
2.18.5	R_ARMMODE	设置/读取 SCARA 机械手的手系	√	×
2.18.6	R_RZ_EN	启用/禁用 RZ 耦合功能	√	×
2.18.7	R_RZ_NUM	设置/读取 RZ 耦合系数分子(该指令暂未启用)	×	×
2.18.8	R_RZ_DENOM	设置/读取 RZ 耦合系数分母(该指令暂未启用)	×	×
2.18.9	R_VL	设置/读取机械手的初速度	√	×
2.18.10	R_VH	设置/读取机械手的运行速度	√	×
2.18.11	R_ACC	设置/读取机械手的加速度	√	×
2.18.12	R_DEC	设置/读取机械手的减速度	√	×
2.18.13	R_JK	设置/读取机械手运动的速度曲线类型	√	×
2.18.14	R_RESETERR	复位 Robot 的状态	√	×
2.18.15	R_STATE	读取 Robot 运动状态	√	×
2.18.16	R_DSPEED	读取 Robot 当前的理论运行速度	√	×

2.18.17	R_DPOS	读取 Robot 机构末端在机械坐标系下的当前理论姿态（X、Y、Z、R）	√	×
2.18.18	R_MPOS	读取 Robot 机构末端在机械坐标系下的实际姿态(通过电机编码器位置换算得到的 X、Y、Z、R）	√	×
2.18.19	R_MOVE	在机械坐标系下执行相对点位运动	√	×
2.18.20	R_MOVEABS	在机械坐标系下执行绝对点位运动	√	×
2.18.21	R_VCHANGE	Robot 运动过程中更改 Robot 的运行速度	√	×
2.18.22	R_VCHANGE_RATE	Robot 运动过程中，以设定当前运行速度百分比的方式更改 Robot 的运行速度	√	×
2.18.23	R_STOPDEC	指定机械手，下减速停止命令	√	×
2.18.24	R_STOPEMG	指定机械手，下立即停止命令	√	×
2.18.25	R_MOVE_ANGLE	在关节坐标系下执行各关节相对角度运动	√	×
2.18.26	R_MOVEABS_ANGLE	在关节坐标系下执行各关节绝对角度运动	√	×
2.18.27	R_LINE	在机械坐标系下执行相对直线插补运动	√	×
2.18.28	R_LINEABS	在机械坐标系下执行绝对直线插补运动	√	×
2.18.29	R_CIRC	指定圆心和圆弧终点 ,在机械坐标系下执行相对圆弧插补运动	×	×
2.18.30	R_CIRCABS	指定圆心和圆弧终点 ,在机械坐标系下执行绝对圆弧插补运动	×	×
2.18.31	R_CIRC_3P	指定圆上 3 点，在机械坐标系下执行相对圆弧插补运动	×	×
2.18.32	R_CIRCABS_3P	指定圆上 3 点，在机械坐标系下	×	×

		执行绝对圆弧插补运动		
2.18.33	R_JOGP	在机械坐标系下执行正向 JOG 运动	√	×
2.18.34	R_JOGN	在机械坐标系下执行负向 JOG 运动	√	×
2.18.35	R_HOME	让 Robot 所有关节回到指定的初始位置	√	×
2.18.36	R_PATHBEGIN	Robot 连续插补运动添加路径的起始符	×	×
2.18.37	R_PATHEND	Robot 连续插补运动添加路径的结束符	×	×
2.18.38	R_PATHRESET	清除 Robot 连续插补路径缓存中的路径数据	√	×
2.18.39	R_PATH_STATUS	读取 Robot 连续插补运动中相关参数	√	×
2.18.40	R_DELAY	Robot 连续插补中的延时段指令	×	×
2.18.41	R_PAUSE	暂停 Robot 当前的运动	√	×
2.18.42	R_RESUME	恢复 Robot 已被暂停的运动	√	×

2.18.1 R_BASE

所属：命令

语法：R_BASE Robot no [,second Robot][,third Robot] …

描述：由 R_BASE 指定机械手控制单元，R_BASE 后面的代码表示对其指定机械手进行操作。一个控制系统，可能存在控制多只机械手，Motion Studio 会自动识别系统中机械手控制单元的数量，并对每个机械手控制单元进行编号。用户要对哪个机械手操作，需先由 R_BASE 指定机械手控制单元。

参数：Robot no 机械手控制单元编号；**范围：**[0,4]。

例程

```
R_BASE 0            '指定 Robot 0
  R_VL=4            '将 Robot 0 的点位运动初速度设置为 4
  R_VH=8            '将 Robot 0 的点位运动运行速度设置为 8
  R_MOVE -200,-100,0,0        'Robot 0 执行相对点位运动
R_BASE 1            '指定 Robot 1
  R_VL=2            '将 Robot 1 的点位运动初速度设置为 2
  R_VH=10           '将 Robot 1 的点位运动初速度设置为 10
```

```
R_MOVEABS 200,200,0,0       'Robot 1 执行绝对点位运动
```

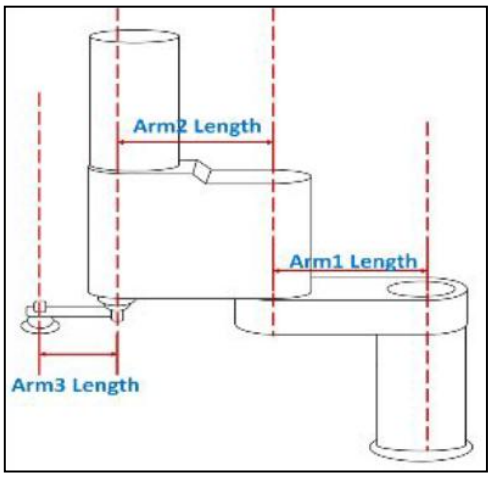
2.18.2 R_ARM1

所属：属性

语法：R_ARM1 = value

类型：DOUBLE

描述：设置/读取 SCARA 机械手的第一个臂的长度，单位为毫米。



范围：[0，4294967.295]，默认值 250

例程

```
R_BASE 0            '指定 Robot 0
  R_ARM1=200        '将编号为 0 的 SCARA 机械手的第一个臂的长度设置为 200mm
```

2.18.3 R_ARM2

所属：属性

语法：R_ARM2 = value

类型：DOUBLE

描述：设置/读取 SCARA 机械手的第二个臂的长度，单位为毫米。

范围：[0，4294967.295]，默认值 250

例程

```
R_BASE 0            '指定 Robot 0
  R_ARM2=200        '将编号为 0 的 SCARA 机械手的第二个臂的长度设置为 200mm
```

2.18.4 R_ARM3

所属：属性

语法：R_ARM3 = value

类型：DOUBLE

描述：设置/读取 SCARA 机械手的第三个臂的长度，单位为毫米。

范围：[0，4294967.295]，默认值 100

例程

```
R_BASE 0            '指定 Robot 0
  R_ARM3=30         '将编号为 0 的 SCARA 机械手的第三个臂的长度设置为 200mm
```

2.18.5 R_ ARMMODE

所属：属性

语法：R_ ARMMODE = value

描述：设置/读取 SCARA 机械手的手系。指定机械手的手系后，其后对机械手的运动命令是基于这个手系的，直到被重新指定手系。

范围：设定值和返回值如下，默认值 0

0：右手系

1：左手系

注意 ：机械手可以在任何位置指定接下来的点位运动手系。 但是只能在复位位置切换接下来的直线插补、圆弧插补、JOG 等运动手系，如机械手在其它位置时需切换接下来的直线插补、圆弧插补、JOG 等运动手系，需先让机械手到复位位置时才能进行切换。

例程

```
R_BASE 0            '指定 Robot 0
  R_ARMMODE=1
R_MOVEABS 200,100,0,0     '用左手系移动到 200,100,0,0 的位置
R_ARMMODE=0
R_MOVEABS 200,90,0,0      '用右手系移动到 200,90,0,0 的位置
```

2.18.6 R_ RZ_EN

所属：属性

语法：R_ RZ_EN = value

描述：启用/禁用 RZ 耦合功能。

范围：设定值和返回值如下，默认值 0

0：禁用

1：启用

例程

```
R_BASE 0            '指定 Robot 0
  R_RZ_EN=1         '启用 Robot 0 的 RZ 耦合功能
```

2.18.7 R_ RZ_NUM

该指令暂未启用，请至 Motion Studio 硬件设定里设置该值。

2.18.8 R_ RZ_DENOM

该指令暂未启用，请至 Motion Studio 硬件设定里设置该值。

2.18.9 R_ VL

所属：属性

语法：R_ VL = value

类型：DOUBLE

描述：设置/读取机械手的初速度，单位为 mm/s。

范围：[0，5,000,000/UNIT（J1 的 UNIT）]，默认值 2

注意：R_VL 的设置值要小于等于 R_VH

例程

```
R_BASE 0            '指定 Robot 0
  R_VL=10           '设置 Robot 0 的初速度为 10mm/s
```

2.18.10 R_ VH

所属：属性

语法：R_ VH = value

类型：DOUBLE

描述：设置/读取机械手的运行速度，单位为 mm/s。

范围：[0，5,000,000/UNIT（J1 的 UNIT）]，默认值 8

例程

```
R_BASE 0            '指定 Robot 0
  R_VH=20           '设置 Robot 0 的运行速度为 20mm/s
```

2.18.11 R_ ACC

所属：属性

语法：R_ ACC = value

类型：DOUBLE

描述：设置/读取机械手的加速度，单位为 mm/s^2。

范围：[0，500,000,000/UNIT（J1 的 UNIT）]，默认值 10

例程

```
R_BASE 0            '指定 Robot 0
  R_ACC=200         '设置 Robot 0 的加速度为 200mm/s^2
```

2.18.12 R_ DEC

所属：属性

语法：R_ DEC = value

类型：DOUBLE

描述：设置/读取机械手的减速度，单位为 mm/ s^2。

范围：[0，500,000,000/UNIT（J1 的 UNIT）]，默认值 10

例程

```
R_BASE 0            '指定 Robot 0
  R_DEC=200         '设置 Robot 0 的减速度为 200mm/s^2
```

2.18.13 R_ JK

所属：属性

语法：R_ JK = value

描述：设置/读取机械手运动的速度曲线类型。

范围：设定值和返回值如下，默认值 0

0：T 型曲线

1：S 型曲线

例程

```
R_BASE 0            '指定 Robot 0
  R_JK= 1           '设置 Robot 0 运动的速度曲线为 S 型曲线
```

2.18.14 R_ RESETERR

所属：命令

语法：R_ RESETERR

描述：复位 Robot 的状态。当 R_STATE 为 STA_RB_ERROR_STOP 的状态时，通过此指令使 Robot 恢复为 ready 状态。

例程

```
R_BASE 0            '指定 Robot 0
  R_RESETERR        '复位 Robot 0 的状态
```

2.18.15 R_STATE

所属：属性(只读)

语法：value = R_STATE

描述：读取 Robot 运动状态。

返回值：如下

0：STA_RB_DISABLE，机械手处于不可用状态。

1：STA_RB_READY，机械手处于准备就绪状态。

2：STA_RB_STOPPING，机械手正在停止的过程中。

3：STA_RB_ERROR_STOP，机械手处于错误停止状态。

4：STA_RB_MOTION，机械手处于运动状态。

5：STA_RB_AX_MOTION，预留状态，暂未启用。

6：STA_RB_MOTION_PATH，机械手处于连续插补运动中。

7：STA_RB_PAUSE，机械手处于暂停的状态中。

8：STA_RB_BUSY，预留状态，暂未启用。

9：STA_RB_EXT_JOG，预留状态，暂未启用。

10：STA_RB_EXT_MPG，机械手处于外部手轮控制状态。

11：STA_RB_EXT_JOG_MOVING，机械手处于 JOG 运动中。

例程

```
R_BASE 0          '指定 Robot 0
DIM A AS ULONG
A=R_STATE   '将 Robot 0 的运动状态赋值给变量 A
```

2.18.16 R_DSPEED

所属：属性(只读)

语法：value = R_DSPEED

类型：DOUBLE

描述：读取 Robot 当前的理论运行速度，单位为 mm/s。

例程

```
R_BASE 0          '指定 Robot 0
DIM A AS ULONG
A=R_DSPEED   '将 Robot 0 的当前理论运行速度值赋给变量 A
```

2.18.17 R_DPOS

所属：属性(只读)

语法：value = R_DPOS.x

value = R_DPOS.y

value = R_DPOS.z

value = R_DPOS.a

类型：DOUBLE

描述：读取 Robot 机构末端在机械坐标系下的当前理论姿态（X、Y、Z、R）。

例程

```
R_BASE 0            '指定 Robot 0
'将 Robot 0 机构末端的当前理论姿态（X、Y、Z、R）依次赋值给 VR(0)~VR(3)
VR(0)=R_DPOS.x
VR(1)=R_DPOS.y
VR(2)=R_DPOS.z
VR(3)=R_DPOS.a
```

2.18.18 R_MPOS

所属：属性(只读)

语法：value = R_MPOS.x

value = R_MPOS.y

value = R_MPOS.z

value = R_MPOS.a

类型：DOUBLE

描述：读取 Robot 机构末端在机械坐标系下的实际姿态（通过电机编码器位置换算得到的 X、Y、Z、R）。

例程

```
R_BASE 0            '指定 Robot 0
'将 Robot 0 机构末端的实际姿态（X、Y、Z、R）依次赋值给 VR(0)~VR(3)
VR(0)=R_MPOS.x
VR(1)=R_MPOS.y
VR(2)=R_MPOS.z
VR(3)=R_MPOS.a
```

2.18.19 R_MOVE

所属：命令

语法：R_MOVE distance_x, distance_y, distance_z, distance_r

描述：指定 Robot 在机械坐标系下的相对移动距离，执行相对点位运动。

参数： distance_x 机械坐标系下 X 方向的相对移动距离；单位：mm；**类型：**DOUBLE

distance_y 机械坐标系下 Y 方向的相对移动距离；单位：mm；**类型：**DOUBLE

distance_z 机械坐标系下 Z 方向的相对移动距离；单位：mm；**类型：**DOUBLE

distance_r 机械坐标系下的相对移动姿态；单位：角度；**类型：**DOUBLE

例程

```
R_BASE 0              '指定 Robot 0
R_ARMMODE=1           '接下来以左手系进行动作
R_MOVE 10,-10,0,-5 '在机械坐标系下的 X、Y、Z、R 方向相对移动 10,-10,0,-5
WAIT DONE             '等待运动结束

'利用数组填写位置执行 R_MOVE 运动
DIM move_distance(3) as DOUBLE={10,-10,0,-5}
R_MOVE move_distance()
WAIT DONE
```

2.18.20 R_MOVEABS

所属：命令

语法：MOVEABS position_x, position_y, position_z, position_r

描述：指定 Robot 在机械坐标系下的绝对移动位置，执行绝对点位运动。

参数： position_x 机械坐标系下 X 方向的绝对移动位置；单位：mm；**类型：**DOUBLE

position_y 机械坐标系下 Y 方向的绝对移动位置；单位：mm；**类型：**DOUBLE

position_z 机械坐标系下 Z 方向的绝对移动位置；单位：mm；**类型：**DOUBLE

position_r 机械坐标系下的绝对移动姿态；单位：角度；**类型：**DOUBLE

例程

```
R_BASE 0            '指定 Robot 0
R_ARMMODE=1         '接下来以左手系进行动作
R_MOVEABS 240,-300,0,-45 '在机械坐标系下绝对移动到位置（240,-300,0,-45）
WAIT DONE           '等待运动结束

'利用数组填写位置执行 R_MOVEABS 运动
DIM move_pos(3) as DOUBLE={400,-120,0,-35}
R_MOVEABS move_pos()
WAIT DONE
```

2.18.21 R_VCHANGE

所属：命令

语法：R_VCHANGE vel

描述：Robot 运动过程中更改 Robot 的运行速度。

参数：vel 运行速度；**类型：**DOUBLE

例程

```
R_BASE 0
R_ARMMODE=0
R_VL=2
R_VH=20
R_ACC=1000
R_DEC=1000
R_MOVE 10,-200,0,10    '以 20 的运行速度开始执行相对运动
SLEEP 2000
R_VCHANGE 100      '延时 2s 后将运行速度改变成 100
WAIT DONE
```

2.18.22 R_VCHANGE_RATE

所属：命令

语法：R_VCHANGE_RATE rate

描述：Robot 运动过程中，以设定当前运行速度百分比的方式更改 Robot 的运行速度。

参数：rate 当前运行速度的百分比；**类型：**DOUBLE

例程

```
R_BASE 0
R_ARMMODE=0
R_VL=1
R_VH=20
R_ACC=1000
R_DEC=1000
R_MOVE 10,-300,0,0     '以 20 的运行速度开始执行相对运动
SLEEP 1000
R_VCHANGE_RATE 20      '延时 1s 后将运行速度改变成 20 的 20%，即将速度改成 4
WAIT DONE
```

2.18.23 R_STOPDEC

所属：命令

语法：R_STOPDEC

描述：搭配 R_BASE 使用，指定机械手，下减速停止命令。

例程

```
R_BASE 0
R_ARMMODE=0
R_VL=1
R_VH=20
R_ACC=1000
R_DEC=1000
R_MOVE -10,200,0,0     'Robot 0 以 20 的运行速度开始执行相对运动
SLEEP 1000
R_STOPDEC              '延时 1s 后，对 Robot 0 下减速停止命令
WAIT DONE
```

2.18.24 R_STOPEMG

所属：命令

语法：R_STOPEMG

描述：搭配 R_BASE 使用，指定机械手，下立即停止命令。

例程

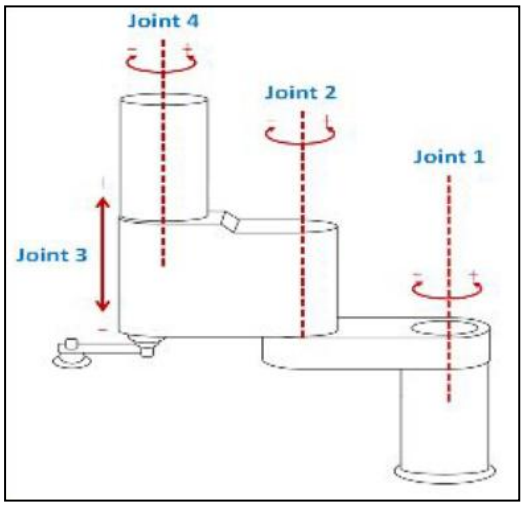
```
R_BASE 0
R_ARMMODE=0
R_VL=1
R_VH=20
R_ACC=1000
R_DEC=1000
R_MOVE -10,200,0,0     'Robot 0 以 20 的运行速度开始执行相对运动
SLEEP 1000
R_STOPEMG              '延时 1s 后，对 Robot 0 下立即停止命令
WAIT DONE
```

2.18.25 R_MOVE_ANGLE

所属：命令

语法：R_MOVE_ANGLE angle_1, angle_2, distance_3, angle_4

描述：指定 Robot 在关节坐标系下的相对移动距离，执行相对运动。



参数： angle_1 J1 关节坐标下的相对移动距离；单位：角度；**类型：**DOUBLE

angle_2 J2 关节坐标下的相对移动距离；单位：角度；**类型：**DOUBLE

distance_3 J3 关节坐标下的相对移动距离；单位：mm；**类型：**DOUBLE

angle_4 J4 关节坐标下的相对移动距离；单位：角度；**类型：**DOUBLE

例程

```
R_BASE 0                    '指定 Robot 0
R_MOVE_ANGLE 5,10,1,15      '使 J1，J2，J3，J4 相对移动 5 度，10 度，1mm,15 度
WAIT DONE                   '等待运动结束

'利用数组填写位置执行 R_MOVE_ANGLE 运动
DIM move_distance(3) as DOUBLE={5,10,1,15}
R_MOVE_ANGLE move_distance()
WAIT DONE
```

2.18.26 R_MOVEABS_ANGLE

所属：命令

语法：R_MOVEABS_ANGLE angle_1, angle_2, position_3, angle_4

描述：指定 Robot 在关节坐标系下的绝对移动位置，执行绝对运动。

参数： angle_1 J1 关节坐标下的绝对移动位置；单位：角度；**类型：**DOUBLE

angle_2 J2 关节坐标下的绝对移动位置；单位：角度；**类型：**DOUBLE

distance_3 J3 关节坐标下的绝对移动位置；单位：mm；**类型：**DOUBLE

angle_4 J4 关节坐标下的绝对移动位置；单位：角度；**类型：**DOUBLE

例程

```
R_BASE 0                        '指定 Robot 0
R_MOVEABS_ANGLE 5,45,10,20      '使 J1，J2，J3，J4 绝对移动到 5 度，45 度，10mm，20 度的位置
WAIT DONE                       '等待运动结束

'利用数组填写位置执行 R_MOVEABS_ANGLE 运动
DIM move_pos(3) as DOUBLE={10,35,0,10}
R_MOVEABS_ANGLE move_pos()
WAIT DONE
```

2.18.27 R_LINE

所属：命令

语法：R_LINE distance_x, distance_y, distance_z, distance_r

描述：指定 Robot 在机械坐标系下的相对移动距离，执行相对直线插补运动。

参数： distance_x 机械坐标系下 X 方向的相对移动距离；单位：mm；**类型：**DOUBLE

distance_y 机械坐标系下 Y 方向的相对移动距离；单位：mm；**类型：**DOUBLE

distance_z 机械坐标系下 Z 方向的相对移动距离；单位：mm；**类型：**DOUBLE

distance_r 机械坐标系下的相对移动姿态；单位：角度；**类型：**DOUBLE

注意：机械手在做直线插补前不要用 R_ARMMODE 去指定手系，因直线插补只支持当前手系下进行直线插补。例如：当前机械手处于右手系，用户用 R_ARMMODE 去指定了左手系再执行直线插补，系统会报错提示，直线插补将不被执行。

例程

```
R_BASE 0              '指定 Robot 0
R_LINE 10,-10,0,-5    '在机械坐标系下以直线插补的方式相对移动 10，-10，0，-5
WAIT DONE             '等待运动结束

'利用数组填写位置执行 R_LINE 运动
DIM move_distance(3) as DOUBLE={10,-10,0,-5}
R_LINE move_distance()
WAIT DONE
```

2.18.28 R_LINEABS

所属：命令

语法：R_LINEABS position_x, position_y, position_z, position_r

描述：指定 Robot 在机械坐标系下的绝对移动位置，执行绝对直线插补运动。

参数： position_x 机械坐标系下 X 方向的绝对移动位置；单位：mm；**类型：**DOUBLE

position_y 机械坐标系下 Y 方向的绝对移动位置；单位：mm；**类型：**DOUBLE

position_z 机械坐标系下 Z 方向的绝对移动位置；单位：mm；**类型：**DOUBLE

position_r 机械坐标系下的绝对移动姿态；单位：角度；**类型：**DOUBLE

例程

```
R_BASE 0            '指定 Robot 0
R_LINEABS 240,-300,0,-45 '在机械坐标系下以绝对直线插补方式到位置（240,-300,0,-45）
WAIT DONE           '等待运动结束

'利用数组填写位置执行 R_LINEABS 运动
DIM move_pos(3) as DOUBLE={250,-320,0,-45}
R_LINEABS move_pos()
WAIT DONE
```

2.18.29 R_CIRC

所属：命令

语法：R_CIRC dir, CenterP( ), EndP( )

描述：指定 Robot 在机械坐标系下的运动方向、相对圆心位置、相对圆弧终点位置，执行相对圆弧插补运动。

参数： dir 圆弧插补运动方向。 0(CW)：顺时针 ； 1(CCW)：逆时针；

CenterP( ) 机械坐标系下圆心的相对位置（相对当前位置）

EndP( ) 机械坐标系下圆弧终点的相对位置（相对当前位置）

例程

```
R_BASE 0
DIM EndP(4) as Double  'EndP:终点坐标
DIM CenP(4) as Double  'CenP:圆心坐标
EndP(0)=100            '圆弧终点相对位置为（100,500,0,0）
EndP(1)=500
EndP(2)=0
EndP(3)=0

CenP(0)=300            '圆弧中心相对位置为（300,200,0,0）
CenP(1)=200
CenP(2)=0
CenP(3)=0
```

```
R_MOVEABS 300,-200,0,0       '执行绝对点位运动到（300,-200,0,0）
WAIT DONE
R_CIRC 0, CenP(), EndP()     '执行相对圆弧插补
WAIT DONE
```

2.18.30 R_CIRCABS

所属：命令

语法：R_CIRCABS dir, CenterP( ), EndP( )

描述：指定 Robot 在机械坐标系下的运动方向、绝对圆心位置、绝对圆弧终点位置，执行绝对圆弧插补运动。

参数： dir 圆弧插补运动方向。 0(CW)：顺时针 ； 1(CCW)：逆时针；

CenterP( ) 机械坐标系下圆心的绝对位置

EndP( ) 机械坐标系下圆弧终点的绝对位置

例程

```
R_BASE 0
DIM EndP(4) as Double  'EndP:终点坐标
DIM CenP(4) as Double  'CenP:圆心坐标
EndP(0)=400            '圆弧终点绝对位置为（400,-100,0,0）
EndP(1)=-100
EndP(2)=0
EndP(3)=0

CenP(0)=300            '圆弧中心绝对位置为（300,-100,0,0）
CenP(1)=-100
CenP(2)=0
CenP(3)=0

R_MOVEABS 300,-200,0,0       '执行绝对点位运动到（300,-200,0,0）
WAIT DONE
R_CIRCABS 1, CenP(), EndP()  '执行绝对圆弧插补
WAIT DONE
```

2.18.31 R_CIRC_3P

所属：命令

语法：R_CIRC_3P dir, RefP( ), EndP( )

描述：3 点圆弧相对插补运动。指定圆弧插补运动方向，圆弧上的 3 个点（当前位置点；圆弧上起、终点之外的一个参考点；圆弧终点），Robot 在机械坐标系下执行相对圆弧插补运动。

参数： dir 圆弧插补运动方向。 0(CW)：顺时针 ； 1(CCW)：逆时针；

RefP( ) 机械坐标系下圆上参考点的相对位置（相对当前位置）

EndP( ) 机械坐标系下圆弧终点的相对位置（相对当前位置）

例程

```
DIM EndP(4) as Double   'EndP:终点坐标
DIM RefP(4) as Double   'RefP:参考点坐标
R_BASE 0
R_ARM1=250
R_ARM2=250
R_ARM3=0
R_ARMMODE=0

RefP(0)=-50    '圆上参考点的相对位置为（-50,-100,0,0）
RefP(1)=-100
RefP(2)=0
RefP(3)=0

EndP(0)=-120   '圆弧中心相对位置为（-120,-120,0,0）
EndP(1)=-120
EndP(2)=0
EndP(3)=0

R_MOVEABS 400,-100,0,0        '执行绝对点位运动到（300,-200,0,0）
WAIT DONE
R_CIRC_3P 0, RefP(), EndP()  '执行相对 3 点圆弧插补
WAIT DONE
```

2.18.32 R_CIRCABS_3P

所属：命令

语法：R_CIRCABS_3P dir, RefP( ), EndP( )

描述：3 点圆弧绝对插补运动。指定圆弧插补运动方向，圆弧上的 3 个点（当前位置点；圆弧上起、终点之外的一个参考点；圆弧终点），Robot 在机械坐标系下执行绝对圆弧插补运动。

参数： dir 圆弧插补运动方向。 0(CW)：顺时针 ； 1(CCW)：逆时针；

RefP( ) 机械坐标系下圆上参考点的绝对位置

EndP( ) 机械坐标系下圆弧终点的绝对位置

例程

```
DIM EndP(4) as Double   'EndP:终点坐标
DIM RefP(4) as Double   'RefP:参考点坐标
R_BASE 0
R_ARM1=250
R_ARM2=250
R_ARM3=0
R_ARMMODE=0

RefP(0)=350    '圆上参考点的绝对位置为（-50,-100,0,0）
RefP(1)=-200
RefP(2)=0
RefP(3)=0

EndP(0)=280    '圆弧中心绝对位置为（-120,-120,0,0）
EndP(1)=-220
EndP(2)=0
EndP(3)=0

R_MOVEABS 400,-100,0,0        '执行绝对点位运动到（300,-200,0,0）
WAIT DONE
R_CIRCABS_3P 0, RefP(), EndP()  '执行绝对 3 点圆弧插补
WAIT DONE
```

2.18.33 R_JOGP

所属：命令

语法：R_JOGP dir

描述：指定机械坐标系下的运动方向，执行正向 JOG 运动。

参数： dir 机械坐标系下的运动方向。 范围如下：

0：机械坐标系下的 X 正方向

1：机械坐标系下的 Y 正方向

2：机械坐标系下的 Z 正方向

3：机械坐标系下的 R 正方向

例程

```
R_BASE 0    '指定 Robot 0
R_JOGP 0    '往 X 正方向进行 JOG 运动
SLEEP 2000 '延时 2s
R_STOPDEC  '减速停止
WAIT DONE  '等待运动结束
```

2.18.34 R_JOGN

所属：命令

语法：R_JOGN dir

描述：指定机械坐标系下的运动方向，执行负向 JOG 运动。

参数： dir 机械坐标系下的运动方向。 范围如下：

0：机械坐标系下的 X 正方向

1：机械坐标系下的 Y 正方向

2：机械坐标系下的 Z 正方向

3：机械坐标系下的 R 正方向

例程

```
R_BASE 0    '指定 Robot 0
R_JOGN 1    '往 Y 负方向进行 JOG 运动
SLEEP 2000 '延时 2s
R_STOPDEC  '减速停止
WAIT DONE  '等待运动结束
```

2.18.35 R_HOME

所属：命令

语法：R_HOME

描述：让 Robot 所有关节回到指定的初始位置。初始位置是在 MotionStudio 中的初始位置校正工具中指定的位置，若未指定，则默认为 0，即各关节回到位置 0.

例程

```
R_BASE 0    '指定 Robot 0
R_HOME      '让 Robot 0 回到初始位置。
WAIT DONE  '等待运动结束
```

2.18.36 R_PATHBEGIN

所属：命令

语法：R_PATHBEGIN [num]

描述：指定路径缓存里添加多少段插补指令后，Robot 开始进入连续插补模式。Num 值不填或 0 时，会将 R_PATHBEGIN 与 R_PATHEND 间所有的段都添加到缓存区里，再开始执行连续插补运动。

参数：num 预先添加到连续插补缓存区段数；**类型：**ULONG；范围【0,10000】

注意：连续插补段不允许存在点位元运动指令的段

例程

```
R_BASE 0
R_PATHRESET                 '清除路径缓存
R_PATHBEGIN                 '进入连续插补状态，等下面路径全部添加完再开始运动
R_MOVEABS 300,200,0,0       '绝对点位运动路径段
R_DELAY = 1000              '延时 1s
R_LINEABS 300,-200,0,0      '绝对直线插补路径段
R_MOVEABS 300,200,0,0       '绝对点位运动路径段
R_LINEABS 400,-100,0,0      '绝对点位运动路径段
R_PATHEND                  '连续插补路径添加结束符，该指令后面的运动指令不属于上述连续插补路径
```

2.18.37 R_PATHEND

所属：命令

语法：R_PATHEND

描述：连续插补路径添加的结束符，该指令对应前一个 R_PATHBEGIN。

例程

'请参考 2.18.36 R_PATHBEGIN **的例程**

2.18.38 R_PATHRESET

所属：命令

语法：R_PATHRESET

描述：清除连续插补路径缓存中的路径数据。

注意：R_PATHBEGIN 与 R_PATHEND 之间的路径段为路径缓存中的路径。如果连续插补执行过程中被停止，下次执行连续插补将从路径缓存中上次剩余的未执行路径开始执行。如果用户不想从剩余未执行的路径段开始执行连续插补，则需先用 R_PATHRESET 指令清除路径缓存。然后重新添加需执行的路径到缓存中，再执行连续插补。

2.18.39 R_PATH_STATUS

所属：命令

语法：R_PATH_STATUS RemainPath,FreeBuffer [,curIndex] [,curCmd]

描述：读取连续插补运动中相关参数。

参数：

RemainPath 剩余未执行的路径段。 **类型：**ULONG

FreeBuffer 总路径缓存中的剩余缓存数。 **类型：**ULONG

curIndex 当前执行的路径段索引。 **类型：**ULONG

curCmd 当前执行的路径段指令。 **类型：**ULONG

curCmd 的枚举如下：

0: EndPath

1: R_MOVEABS_ANGLE（不支持）

2: R_MOVE_ANGLE（不支持）

3: R_MOVEABS

4: R_MOVE

5: R_LINEABS

6: R_LINE

7: 顺时针 R_CIRCABS

8: 顺时针 R_CIRC

9: 逆时针 R_CIRCABS

10: 逆时针 R_CIRC

11: 顺时针 R_CIRCABS_3P

12: 顺时针 R_CIRC_3P

13: 逆时针 R_CIRCABS_3P

14: 逆时针 R_CIRC_3P

20: R_DELAY

21: DOUT

例程

```
DIM AS ULONG RemainPath, FreeBuffer, curIndex,curCmd
R_BASE 0
R_PATHRESET                '清除路径缓存
R_PATHBEGIN                '进入连续插补状态，等下面路径全部添加完再开始运动
R_MOVEABS 300,200,0,0      '绝对点位运动路径段
R_DELAY = 1000             '延时 1s
R_LINEABS 300,-200,0,0     '绝对直线插补路径段
R_MOVEABS 300,200,0,0      '绝对点位运动路径段
R_LINEABS 400,-100,0,0     '绝对点位运动路径段
```

```
R_PATHEND              '连续插补路径添加结束符，该指令后面的运动指令不属于上述连续插补路径
SLEEP 500
R_PATH_Status RemainPath, FreeBuffer, curIndex, curCmd
Print  RemainPath, FreeBuffer, curIndex, curCmd   '打印出 4，9996,1,3
```

2.18.40 R_DELAY

所属：命令

语法：R_DELAY= time

描述：连续插补中的延时段指令。表示 R_DELAY 指令上一段完成与 R_DELAY 指令下一段开始间延时的时间，该延时时间非常精准，单位为 ms。

例程

'请参考 2.18.36 R_PATHBEGIN 指令中**例程。**

2.18.41 R_PAUSE

所属：命令

语法：R_PAUSE

描述：暂停 Robot 的运动，对点到点、直线插补、圆弧插补、连续插补运动起作用。

例程

```
R_BASE 0
R_PAUSE                '暂停 Robot0 正在执行的运动
```

2.18.42 R_RESUME

所属：命令

语法：R_RESUME

描述：恢复已暂停的 Robot 运动，继续执行原本被暂停的运动指令。

例程

```
R_BASE 0
R_RESUME               '恢复 Robot0 已被暂停的运动
```

2.19 EtherCAT

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.19.1	REOPEN	重新扫描 EtherCAT 连接状态	√	×
2.19.2	CTLSTATUS	读取当前 EtherCAT 控制器状态	√	×

2.19.1 REOPEN

所属：命令

语法：REOPEN

描述：重新扫描当前 EtherCAT 控制器的主站与从站之间的连接状态。EtherCAT 线路发生过断线等通信异常后，如已经恢复通信到正常状态，需要使用该指令让出现过断线等通信异常的连接恢复成可使用的状态。

例程

'一般情况下，可以通过 CTLSTATUS 指令获取当前控制器的状态，如果检测到有通信异常的状况，控制器会停止工作。需在排除通信异常的问题后，再通过 REOPEN 指令恢复连接状态。

```
Dim a As ULONG
While 1
    a=CTLSTATUS                    '循环检测控制器状态
If(a<>0) Then                  '如果控制器状态有异常
'假设 VR(0)作为上位 UI 对应的 ReOpen 按钮命令接收器，按钮按下时,VR(0)置 1
       If(vr(0)=1) Then
           vr(0)=0
ReOpen                 '上位 ReOpen 按钮按下，进行 ReOpen 动作
       End If
End If
    Sleep 10
WEnd
```

2.19.2 CTLSTATUS

所属：命令

语法 1：value=CTLSTATUS

语法 2：

- ✧ value=CTLSTATUS. ERRSTOP ：CTLSTATUS. ERRSTOP 即 CTLSTATUS 的 Bit0 位
- ✧ value=CTLSTATUS. MRDIS ：CTLSTATUS. ERRSTOP 即 CTLSTATUS 的 Bit1 位
- ✧ value=CTLSTATUS. IORDIS ：CTLSTATUS. ERRSTOP 即 CTLSTATUS 的 Bit2 位

类型：ULONG

描述：读取当前 EtherCAT 控制器状态。

返回值：如下

1：CTLSTATUS 的 bit0，表示轴发生 Error Stop 或处于 Disable 状态。

2：CTLSTATUS 的 bit1，Motion Ring 断线

4：CTLSTATUS 的 bit2，I/O Ring 断线

例程 1

```
Dim a As ULONG
a=CTLSTATUS
```

例程 2

```
Dim a As Boolean
a=CTLSTATUS. ERRSTOP
a=CTLSTATUS. MRDIS
a=CTLSTATUS. IORDIS
```

2.20 模板框架指令

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.20.1	MS_LOOP(time) ... MS_LEND	带扫描时间的循环指令	×	×
2.20.2	MS_LEXIT	退出 MS_LOOP 循环体指令	×	×
2.20.3	MS_PULSE	全局变量值是否有脉冲发生	×	×
2.20.4	MS_EDGER	全局变量值是否有上升沿变化(0-->非 0)	×	×
2.20.5	MS_EDGEF	全局变量值是否有下降沿发生(非 0-->0)	×	×
2.20.6	MS_STEP0... MS_STEP10	顺序流程的步骤变量	×	×

2.20.1 MS_LOOP(time)...MS_LEND

语法：MS_LOOP(time)

commands

MS_LEND

描述：带扫描时间的回圈循环流程语句。MS_LOOP(10)...MS_LEND 相当于 while(1)...sleep 10 Wend 循环体。

参数：time 回圈中 Sleep 的时间，单位为 ms

commands 指令块

例程

'每隔 10 毫秒执行一次 MS_LOOP 循环体中的指令块。

```
MS_LOOP(10)
   BASE 0
   MOVE 1000
   WAIT DONE
   SLEEP 1000
   MOVE -1000
   WAIT DONE
   SLEEP 1000
MS_LEND
```

2.20.2 MS_LEXIT

语法：MS_LEXIT

描述：退出 MS_LOOP(time)...MS_LEND 回圈循环。

例程

```
MS_LOOP(10)
   BASE 0
   MOVE 1000
   WAIT DONE
   SLEEP 1000
   IF(VR(500)=1) THEN     '当 VR(500)等于 1 时，退出 MS_LOOP(10)...MS_LEND 回圈循环
      MS_LEXIT
   END IF
   MOVE -1000
   WAIT DONE
   SLEEP 1000
MS_LEND
```

2.20.3 MS_PULSE

语法：value=MS_PULSE(全局变量)

描述：当全局变量的值发生变化时，value 的值返回为 1。全局变量的值没有发生变化，value 的值返回为 0。每执行一次该指令都会得到有一个返回值。

注意 1：该指令需在 MS_LOOP(time)...MS_LEND 指令的循环体中才起作用。

注意 2：该指令中的全局变量仅支持如下范围：

- ✧ VR(n): n 的范围为 0~9999
- ✧ MS_STEPn: n 的範圍 0-10
- ✧ DOUT(n): n 的范围 0~可用最大 DO 编号
- ✧ DIN(n): n 的范围 0~可用最大 DI 编号

例程

```
'当 VR(60)的值发生变化时，VR(500)的值加 1
MS_LOOP(10)
   IF MS_PULSE(VR(60)) THEN VR(500)=VR(500)+1
MS_LEND
```

2.20.4 MS_EDGER

语法：value=MS_EDGER(全局变量)

描述：当全局变量的值从 0 变成非 0 值时，value 的值返回为 1。其它情况下，value 的值返回为 0。每执行一次该指令都会得到有一个返回值。

注意 1：该指令需在 MS_LOOP(time)...MS_LEND 指令的循环体中才起作用。

注意 2：该指令中的全局变量仅支持如下范围：

- ✧ VR(n): n 的范围为 0~9999
- ✧ DOUT(n): n 的范围 0~可用最大 DO 编号
- ✧ DIN(n): n 的范围 0~可用最大 DI 编号

例程

```
'当 VR(60)的值从 0 变成非 0 时，VR(500)的值加 1
MS_LOOP(10)
   IF MS_EDGER(VR(60)) THEN VR(500)=VR(500)+1
MS_LEND
```

2.20.5 MS_EDGEF

语法：value=MS_EDGEF(全局变量)

描述：当全局变量的值从非 0 变成 0 值时，value 的值返回为 1。其它情况下，value 的值返回为 0。每执行一次该指令都会得到有一个返回值。

注意 1：该指令需在 MS_LOOP(time)...MS_LEND 指令的循环体中才起作用。

注意 2：该指令中的全局变量仅支持如下范围：

- ✧ VR(n): n 的范围为 0~9999
- ✧ DOUT(n): n 的范围 0~可用最大 DO 编号
- ✧ DIN(n): n 的范围 0~可用最大 DI 编号

例程

```
'当 VR(60)的值从非 0 变成 0 时，VR(500)的值加 1
MS_LOOP(10)
   IF MS_EDGEF(VR(60)) THEN VR(500)=VR(500)+1
MS_LEND
```

2.20.6 MS_STEP0~MS_STEP10

语法：MS_STEPn=value

描述：模板框架中提供了 11 个全局变量 MS_STEP0~MS_STEP10，用于顺序流程控制中的步骤编号。通常 1 个 MS_STEP 用于 1 个顺序流程控制，不同的 Task 中需用不同的 MS_STEP 号。

注意 1：该指令需在 MS_LOOP(time)...MS_LEND 指令的循环体中才起作用。

注意 2：MS_STEPn 不是所有 Task 的全局变量，是 1 个 Task 中的全局变量。

例程

```
'下面顺序流程动作中，用 MS_STEP1 来控制步骤。
' MyInit()是已经定义好的初始化 SUB 函数
' MyHome()是已经定义好的回原点动作 SUB 函数
' MyOrg()是已经定义好的回工作原点 SUB 函数
' MyRun()是已经定义好的加工动作 SUB 函数
' VR(2)的值为 1 时，代表上位界面“回原点按钮”按下
' VR(50)的值为 1 时，代表回原点动作结束
' VR(3)的值为 1 时，代表上位界面“开始加工按钮”按下
' VR(51)的值为 1 时，代表加工程序动作结束
MS_LOOP(10)
   SELECT CASE CINT(MS_STEP1)
   CASE 0                                          '步骤 0：初始化
      IF MS_PULSE(MS_STEP1)  THEN MyInit()            '程序一开始，执行一次初始化动作
      IF MS_EDGER(VR(2))     THEN MS_STEP1 = 1        '当回原点按钮按下，跳到步骤 1
   CASE 1                                          '步骤 1：回原点
      IF MS_PULSE(MS_STEP1)  THEN MyHome()            '当步骤发生变化，执行一次回原点动作
      IF VR(50) = 1          THEN MS_STEP1 = 2        '当回原点动作结束，跳到步骤 2
CASE 2                                          '步骤 2：回工作原点
      IF MS_PULSE(MS_STEP1)  THEN MyOrg()          '当步骤发生变化，执行一次回工作原点动作
      IF MS_EDGER(VR(3))     THEN MS_STEP1 = 3     '当开始加工按钮按下，跳到步骤 3
CASE 3                                          '步骤 3：加工程序
      IF MS_PULSE(MS_STEP1)  THEN MyRun()          '当步骤发生变化，执行一次加工程序动作
      IF VR(51) = 1          THEN MS_STEP1 = 2     '当加工程序结束，回到步骤 2，到工作原点
  END SELECT
MS_LEND
```

2.21 模块类

该章节的指令是进阶的面向对象的类用法。Motion Studio 底层针对一些常用的模块对象，封装了一些类，让用户面对一些复杂的应用时，可以使用更简单的编程来达到目的。

本节指令概览

章节	类	说明	终端工具	观察变量工具
2.21.1	Pt	位置点(P 点)的类	×	×
2.21.2	TMR	定时器的类	×	×
2.21.3	Tab	二维数据表的类	×	×
2.21.4	DAQ	研华 DAQ 系列卡的类	×	×

2.21.1 Pt

所属：类

说明：Pt 是一个位置点的类，可实例化出一个个位置点，我们称为 P 点。每个 P 点是一个 8 维的位置数据点。运动控制中常用到位置点的定位，Pt 类搭配运动指令提高了程序可读性，很方便的实现位置点定位。

- 内建声明：

```
TYPE Pt
    As DOUBLE x,y,z,a,b,c,u,v
END TYPE
```

- 对象实例化：Dim Object_name As Pt

Object_name：实例出的对象名

- 对象赋值：Object_name=Pt(position1, [position2], ..., [position8])

position：位置点。位置点可以任意填写 1~8 维数据，最多到 8 维。

- 类成员说明

属性：x,y,z,a,b,c,u,v

描述：x,y,z,a,b,c,u,v 对应 P 点里的 1~8 维数据。

例程

```
'例程 1：实例 1 个位置点对象进行两轴运动
Dim P0 As Pt              '实例化出一个位置点对象 P0
P0=Pt(10000,20000)        '位置点赋值为(10000,20000)
BASE 0,1
MOVEABS P0                '轴 0，轴 1 执行绝对点位运动到 P0 点(10000,20000)
WAIT DONE
LINE P0                   '轴 0，轴 1 执行相对直线插补运动(10000,20000)
WAIT DONE

'例程 2：实例多个位置点对象，进行多轴运动
Dim As Pt P1,P2,P3              '实例化出 3 个位置点对象,分别为 P1,P2,P3
P1=Pt(5000,100)                 '位置点 P1 赋值为(5000,100)
P2=Pt(100,200,300,400)          '位置点 P2 赋值为(100,200,300,400)
P3=Pt(-100,3000,-5000)          '位置点 P3 赋值为(-100,3000,-5000)
BASE 0,1
MOVEABS P1                '轴 0，轴 1 执行绝对点位运动到 P1 点
WAIT DONE
BASE 0,1,2,3
MOVEABS P2                '轴 0，轴 1，轴 2，轴 3 执行绝对点位运动到 P2 点
WAIT DONE
BASE 0,1,2
```

```
LINEABS P3                '轴 0，轴 1，轴 2 执行绝对直线插补运动到 P3 点
WAIT DONE

'例程 3：P 点间赋值，加减运算
DIM AS PT P0,P1,P2,P3
P0=PT(10,10,10,10)
P1=PT(20,20,20)
P2=P0+P1    '将 P0 点加上 P1 点再赋值给 P2 点
Print P2    'P2 为(30,30,30,10,0,0,0,0)

P3=P0-P1    '将 P0 点减去 P1 点再赋值给 P3 点
Print P3    'P3 为(-10,-10,-10,10,0,0,0,0)

P0=P1       '将 P1 点赋值给 P0 点
Print P0    'P0 为(20,20,20,0,0,0,0,0)

'例程 4：对 P 点里的 1~8 维数据进行单独读写。
DIM AS PT P0
P0=PT(10,20,30,40,50,60,70,80)
P0.x=1
P0.z=3
P0.a=4
P0.v=8
VR(0)=P0.x      '将 P0.x 读出来赋值给 VR(0)
Print P0        '打印出来 P0 为 ( 1,20,3,4,50,60,70,8 )
```

2.21.2 TMR

所属：类

说明：TMR 类提供了定时器功能，用于定时处理一些任务。该定时器是通过扫描执行 On 方法来判断定时是否到位。所以使用定时器时，需要将 On 方法放到循环体中扫描执行。否则定时器将不起作用。

- 内建声明：

```
TYPE TMR
    DECLARE  FUNCTION    On (t_ms AS uLongInt)    AS  BOOLEAN    '定时器计时
    DECLARE  SUB         Reset ( )                '复位定时器
END TYPE
```

- 对象实例化：Dim Object_name As TMR

Object_name：实例出的对象名

- 类成员说明

方法：On (t_ms)

t_ms：定时时长

描述：通过扫描执行 On 方法来计时。当用时达到 t_ms 时，On (t_ms)方法返回 True

例子：定时 1 秒打印出 VR(0)的值

```
Dim T1 As TMR                    '实例 1 个定时器对象 T1
WHILE 1
    IF(T1.On(1000)) THEN             '判断定时是否到达 1 秒
        T1.Reset()               '1 秒到达后，复位定时器，定时器重新计时
        PRINT VR(0)                  '定时器到位后，打印 VR(0)的值
    END IF
    SLEEP 1
WEND
```

方法：Reset ( )

描述：定时器复位的方法，定时器达到定时后，需要使用 Reset ( )复位定时器，否则定时器不会重新启动。程序写法请参考 On 方法里的例子。

2.21.3 Tab

所属：类

说明：Tab 类是将指定索引区间的 Table 变量映射成一个二维表格对象，然后通过这个对象的属性、方法就可以实现对指定索引区间的Table变量进行读写。这个功能常与HMI.NET中的MSDataTable控件组合使用，可以使用户更方便的管理参数、工单等数据。

- 内建声明：

```
TYPE Tab
    DECLARE SUB SetValue (row_no As ULONG,column_no As ULONG,value as DOUBLE )
    DECLARE FUNCTION GetValue (row_no As ULONG,column_no As ULONG )  as DOUBLE
    DECLARE SUB MapPoint(column_no AS ULONG,point_size AS ULONG)
    Point(row_no)  As Pt
END TYPE
```

- 对象实例化：Dim Object_name As Tab

Object_name：实例出的对象名

- 对象赋值：Object_name=Tab(start_address, rows, columns)

start_address：以这个 Table 变量的地址索引为起始地址映射二维表格。

rows：映射的二维表格行数。

columns：映射的二维表格列数。

- 类成员说明

方法：SetValue (row_no,column_no,value)

row_no：行号

column_no：列号

value：要设置表格中的值

描述：对表格（行号、列号）对应格设定一个数值。

例子：如下图，要将表格中的（行 3，列 3）格的值设置成 7


```
Dim Table_A as Tab          '实例一个 Tab 对象，名为 Table_A
Table_A=Tab(1000,5,5)       '如上图，以 TABLE(1000)为起始地址，映射出 1 个 5 行*5 列的二维表格
Table_A.SetValue(3,3,7)             '将（行 3，列 3）格子中的值设为 7
```

方法：GetValue (row_no , column_no)

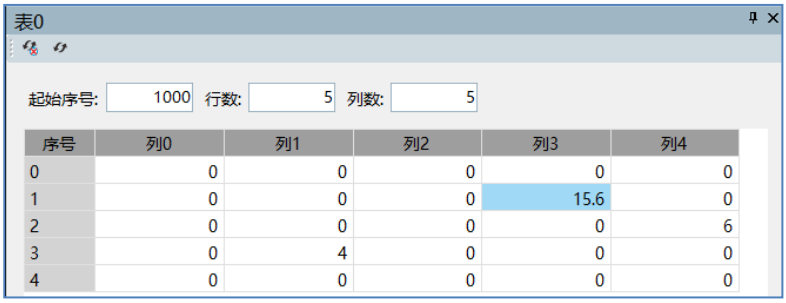
分类：方法

row_no：行号

column_no：列号

描述：根据表格的行号、列号取对应表格内的值。

例子：以下面 Motion Studio 中的 Table 工具显示的表格为例，要取表格（行 1，列 3）的值



序号	列0	列1	列2	列3	列4
0	0	0	0	0	0
1	0	0	0	15.6	0
2	0	0	0	0	6
3	0	4	0	0	0
4	0	0	0	0	0

```
Dim Table_A as Tab          '实例一个 Tab 对象，名为 Table_A
Table_A=Tab(1000,5,5)       '如上图，以 TABLE(1000)为起始地址，映射出 1 个 5 行*5 列的二维表格
Print Table_A.GetValue(1,3)     '取出这个表格中（行 1，列 3）格子中的值，打印出的值为 15.6
```

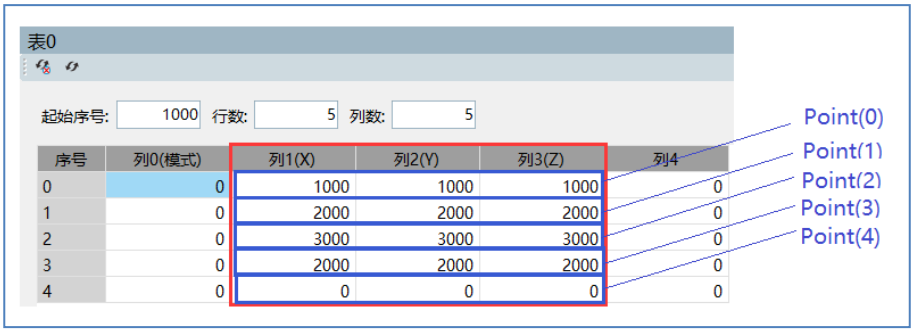
方法：MapPoint(column_no,point_size)

column_no：列号

point_size：P 点大小

描述：将每行 column_no 列开始的 point_size 个列的数据复制到 P 点中，P 点的个数同表格的行数。P 点的维数最大为 8 维。

例子：以下面 Motion Studio 中的 Table 工具显示的表格（5*5）为例，要将每行的列 1 到列 3 的值组成一个 P 点，会得到 5 个 P 点值。让轴 0，轴 1，轴 2 做 3 轴直线插补到 Point(2)。



序号	列0(模式)	列1(X)	列2(Y)	列3(Z)	列4
0	0	1000	1000	1000	0
1	0	2000	2000	2000	0
2	0	3000	3000	3000	0
3	0	2000	2000	2000	0
4	0	0	0	0	0

```
Dim Table_A As Tab          '实例一个 Tab 对象，名为 Table_A
Dim P0 As PT                '实例一个 P 点对象，名为 P0
Table_A=Tab(1000,5,5)       '如上图，以 TABLE(1000)为起始地址，映射出 1 个 5 行*5 列的二维表格
Table_A.MapPoint (1,3)      '将 Table_A 每行的列 1 到列 3 的值复制到 P 点：Point(0), Point(1), Point(2), Point(3), Point(4)
BASE 0,1,2
P0=Table_A.Point(0)         '将第 0 行的列 1 到列 3 表格映射出来 Point(0)赋值给 P 点 P0
```

```
LINEABS P0                    '将轴 0，轴 1，轴 2 做 3 轴直线插补到 P0 点，由图可知 P0 为（1000,1000,1000）
WAIT DONE
LINEABS Table_A.Point(2)   '将轴 0，轴 1，轴 2 做 3 轴直线插补到 Point(2)，由图可知 Point(2)为（3000,3000,3000）
WAIT DONE
```

属性：Point(row_no)

row_no：行号

描述：Point(row_no)为 MapPoint 方法映射出的 P 点。程序写法请参考 MapPoint 方法中例子。

2.21.4 DAQ

当 MAS 控制器中需要插入研华 DAQ 系列的 I/O 卡时，如果要使用 Motion Studio 来编程控制 DAQ 系列卡时，请参照下面说明。

目前 MAS 控制器里直接能识别并映射出 I/O 指令的研华 DAQ 系列卡有以下几种。

- PCI-1750
- PCI-1756
- PCIe-1730
- PCIe-1752
- PCIe-1754
- PCIe-1756
- PCIe-1758DI，PCIe-1758DO, PCIe-1758DIO

用户需要在 Motion Studio 里编程控制 DAQ 系列的除上述几种的其它 I/O 卡，请使用 DAQ 类。

（注意：该做法仅适用于 Motion Studio V1.9（含）以后的版本）。

- 引用 bi：

要在 Motion Studio 里使用研华 DAQ 卡的指令前，需先在控制器里安装对应卡的驱动。然后在 Task 里引用 DAQ 卡的 bi 文件，引用的语句如下：

```
#include once "ExPCIBoard.bi"
```

- 内建声明：

```
TYPE DAQ
    DECLARE FUNCTION Init(port As Ulong, devicename As WString Ptr) As Ulong
    DECLARE FUNCTION ReadDI(n As Ulong) As Ulong
    DECLARE FUNCTION WriteDO(n As Ulong, value As Boolean) As Ulong
    DECLARE FUNCTION ReadDO(n As Ulong) As Ulong
    DECLARE FUNCTION ReadAI(n As Ulong) As DOUBLE
    DECLARE FUNCTION WriteAO(n As Ulong, value As DOUBLE) As Ulong
END TYPE
```

- 对象实例化：Dim Object_name As DAQ

Object_name：实例出的对象名

- 对象赋值及初始化：

```
Dim As Wstring*64 name_str=>"DeviceName"
Object_name.init(0, name_str)
```

- 类成员说明

方法：Init(port，devicename)

port：卡的端口号，该处固定填 0。

devicename：DAQ 卡名，在 Navigator 软件里查询得到。

描述：对 DAQ 卡进行初始化，初始化后才可以使用读写指令。

方法：ReadDI(n)

n：该卡的 DI 索引号。

描述：读取对应索引的 DI 端口状态值。

方法：ReadDO(n)

n：该卡的 DO 索引号。

描述：读取对应索引的 DO 端口状态值。

方法：WriteDO(n,value)

n：该卡的 DO 索引号。

value：0 或 1

描述：将对应索引的 DO 端口置 0 或置 1。

方法：ReadAI(n)

n：该卡的 AI 索引号。

描述：读取对应索引的 AI 端口数值。

方法：WriteAO(n,value)

n：该卡的 AO 索引号。

value：电压值，值的范围为 Navigator 软件中设置的范围。

描述：设置输出对应索引的 AO 电压值，单位是伏特。

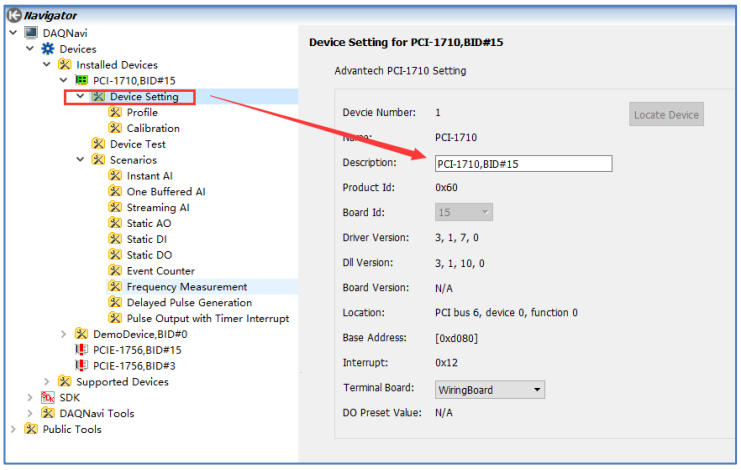
例子：按下面操作步骤即可正常在 Motion Studio 中进行 DAQ 卡的读写控制。

步骤 1：下载 DAQ 卡相关软件和驱动并安装。

至研华官网 www.advantech.com.cn 下载 DAQNavi_SDK 软件和对应的卡 DAQNavi 驱动，并安装到 MAS 控制器上。

步骤 2：使用 Navigator 软件进行调试、配置。

安装完 DAQNavi_SDK 后，打开 Navigator 软件可以进行 DAQ 卡的信息查询，配置，监控等操作。如果要操作模拟量卡，先使用 Navigator 软件进行输入信号种类、量程等配置。如下图，Navigator 软件里展示了 PCI-1710 的相关信息。



步骤 3：编写程序。

```
#include once "ExPCIBoard.bi"          '引用库
Dim DAQ_Card1 As DAQ                   '声明一个 DAQ 类的对象，名为 DAQ_Card1
'声明一个 64 长度的字符串，将要控制的卡名称赋值给这个字符串。控制卡的名称可到 Navigator 软件里查询得到。
Dim As Wstring*64 DeviceName =>"PCI-1710,BID#15"
DAQ_Card1.init(0, DeviceName)   'init()方法的第 1 个参数写 0，第 2 个参数填入 DeviceName

VR(0)=DAQ_Card1.ReadDI(5)          '将该卡的 DI(5)当前值读出来并赋给 VR(0)
VR(1)=DAQ_Card1.ReadDO(5)          '将该卡的 DO(5)当前值读出来并赋给 VR(1)
DAQ_Card1.WriteDO(3,0)             '将该卡的 DO(3)置 0
DAQ_Card1.WriteDO(3,1)             '将该卡的 DO(3)置 1
VR(2)= DAQ_Card1.ReadAI(1)         '将该卡 AI 通道 1 的当前值读出来并赋给 VR(2)
DAQ_Card1.WriteAO(0,2.1)           '将改卡 AO 通道 0 写入 2.1V
```

2.22 数据类型及类型转换指令

2.22.1 常用数据类型说明

数据类型	说明
BOOLEAN	布尔数据类型，True 或者 False
BYTE	长度为 8 位的整数数据类型
UBYTE	长度为 8 位的无符号整数数据类型
DOUBLE	长度为 64 位的双精度浮点数据类型
INTEGER	长度为 32 位或 64 位的整数数据类型
UINTEGER	长度为 32 位或 64 位的无符号整数数据类型
LONG	长度为 32 位的整数数据类型
ULONG	长度为 32 位的无符号整数数据类型
SHORT	长度为 16 位的整数数据类型
USHORT	长度为 16 位的无符号整数数据类型
UNSIGNED	整数类型有无符号的修饰符
STRING	字符串类型
SINGLE	长度为 32 位的单精度浮点数据类型

2.22.2 数据类型转换指令

指令	说明
CBYTE	将数字或字符串的表达式转换成字节类型数据
CDBL	将数字或字符串的表达式转换成双精度浮点数
CHR	返回用 ASCII 码表达的值对应的字符
CINT	将数字或字符串表达式转换成整型数据
CLNG	将数字或字符串表达式转换成长整型数据
CLNGINT	将数字或字符串表达式转换成 64 位长整型数据
CSNG	将数字或字符串的表达式转换成单精度浮点数
CUBYTE	将数字或字符串表达式转换为无符号字节类型数据
CUINT	将数字或字符串表达式转换为无符号整型数据
CULNG	将数字或字符串表达式转换为无符号长整型数据
CULNGINT	将数字或字符串表达式转换为无符号 64 位长整型数据
CUNSG	将一个表达式转换成对应的无符号类型
CUSHORT	将数字或字符串表达式转换为无符号短整型数据
VALINT	将一个字符串转换为一个 INTEGER 类型数据
VAL	将一个字符串转换为一个浮点数
HEX	将一个数用十六进制数返回
OCT	将一个数用八进制数返回
VALLNG	将一个字符串转换为一个 LONG 类型数据
VALUINT	将一个字符串转换为一个 UINTEGER 类型数据
VALULNG	将一个字符串转换为一个 ULONG 类型数据
ASC	返回字符串中字符的 ASCII 码

CINT 例程

```
DIM A AS DOUBLE=15.2
DIM B AS STRING="156"
DIM C AS INTEGER

C=CINT(A)  '将 A 转换成整数类型赋值给 C
PRINT C        '打印出 C 的值为 15
PRINT CINT(B) '将 B 转换成整数类型，打印出值为 156
```

HEX 例程

```
'54321 的十六进制为 D431
Print Hex(54321)      '打印出的结果为：D431
Print Hex(54321, 2)   '打印出的低顺序 2 位为：31
Print Hex(54321, 5)   '打印出的低顺序 5 位为：0D431
```

2.22.3 数据类型简化写法

- 内建声明:

```
Type    I8      As  Byte
Type    I16     As  Short
Type    I32     As  Long
Type    I64     As  LongInt
Type    U8      As  uByte
Type    U16     As  uShort
Type    U32     As  ulong
Type    U64     As  uLongInt
Type    F32     As  Single
Type    F64     As  Double
```

- 说明:

I8

描述：长度为 8 位的整数数据类型。等同于 Byte 类型。

例子：

```
Dim a As I8    '声明一个变量 a，数据类型为 8 位的整数类型。
```

I16

描述：长度为 16 位的整数数据类型。等同于 Short 类型。

例子：

```
Dim a As I16   '声明一个变量 a，数据类型为 16 位的整数类型。
```

I32

描述：长度为 32 位的整数数据类型。等同于 Long 类型。

例子：

```
Dim a As I32   '声明一个变量 a，数据类型为 32 位的整数类型。
```

I64

描述：长度为 64 位的整数数据类型。等同于 LongInt 类型。

例子：

```
Dim a As I64   '声明一个变量 a，数据类型为 64 位的整数类型。
```

U8

描述：长度为 8 位的无符号整数数据类型。等同于 UByte 类型。

例子：

```
Dim a As U8    '声明一个变量 a，数据类型为 8 位的无符号整数类型。
```

U16

描述：长度为 16 位的无符号整数数据类型。等同于 UShort 类型。

例子：

```
Dim a As U16   '声明一个变量 a，数据类型为 16 位的无符号整数类型。
```

U32

描述：长度为 32 位的无符号整数数据类型。等同于 ULong 类型。

例子：

```
Dim a As U32   '声明一个变量 a，数据类型为 32 位的无符号整数类型。
```

U64

描述：长度为 64 位的无符号整数数据类型。等同于 ULongInt 类型。

例子：

```
Dim a As U64   '声明一个变量 a，数据类型为 64 位的无符号整数类型。
```

F32

描述：长度为 32 位的单精度浮点数据类型。等同于 Single 类型。

例子：

```
Dim a As F32   '声明一个变量 a，数据类型为 32 位的浮点类型。
```

F64

描述：长度为 64 位的双精度浮点数据类型。等同于 DOUBLE 类型。

例子：

```
Dim a As F64   '声明一个变量 a，数据类型为 64 位的浮点类型。
```

2.23 日志记录

日志使用说明

- Log 的用途

不管是使用何种编程语言，日志输出几乎无处不再。总结起来，日志大致有 2 种用途：

1. 问题追踪：通过日志不仅仅包括我们程序的一些 bug，也可以在安装配置时，通过日志可以发现问题。
2. 状态监控：通过实时分析日志，可以监控系统的运行状态，做到早发现问题、早处理问题。

- 日志的级别划分

可以分为：Fatal、Error、Warn、Info、Debug、Trace 六个级别：

Fatal: 严重错误，并且软件不能自行恢复到正常的运行状态

Error: 问题已经影响到软件的正常运行，此时需要输出该级别的错误日志。

Warn: 与输入处理相关的失败，此次失败不影响下次业务的执行，通常的结果为外部的输入不能获得期望的结果。

Info：系统运行期间的系统运行状态变化，或关键处理记录等用户或管理员在系统运行期间关注的一些信息。

Debug：软件调试信息，开发人员使用该级别的日志发现程序运行中的一些问题，排除故障。

Trace：基本同上，但显示的信息更详尽。

- 日志对性能影响

不管是多么优秀的日志工具，在日志输出时总会对性能产生或多或少的影响，为了将影响降低到最低，有以下几个准则需要遵守：

注意：频繁产生的日志输出，比如 for、while 循环将对性能造成严重影响。

判断日志级别：

1. 对于可以预见的会频繁产生的日志输出，比如 for、while 循环，定期执行的 job 等，建议先使用 if 对日志级别进行判断后再输出。

2. 对于日志输出内容需要复杂的串行化 ,或输出的某些信息获取时间成本较高时 ,需要对日志级别进行判断。比如日志中需要输出用户名，而用户名需要在日志输出时从数据库获取，此时就需要先判断一下日志级别，看看是否有必要获取这些信息。

3. 优先使用参数 ,减少字符串拼接 :使用参数的方式输出日志信息 ,有助于在性能和代码简洁之间取得平衡。当日志级别限制输出该日志时，参数内容将不会融合到最终输出中，减少了字符串的拼接，从而提升执行效率。

- 什么时候输出日志

日志并不是越多越详细就越好。在分析运行日志，查找问题时，我们经常遇到该出现的日志没有，无用的日志一大堆，或者有效的日志被大量无意义的日志信息淹没，查找起来非常困难。那么什么时候输出日志呢？

以下列出了一些常见的需要输出日志的情况，而且日志的级别基本都是>=INFO，至于 Debug 级别日志的使用场景，本节没有专门列出，需要具体情况具体分析，但也是要追求“恰如其分”，不是越多越好。

本节指令概览

章节	指令	说明	终端工具	观察变量工具
2.23.1	Log_Set	配置日志记录的相关属性	×	×
2.23.2	Log_SetLevel	设置 Log 记录器的过滤级别	×	×
2.23.3	Log_Trace	记录 Trace 级别的日志	×	×
2.23.4	Log_Debug	记录 Debug 级别的日志	×	×
2.23.5	Log_Info	记录 Info 级别的日志	×	×
2.23.6	Log_Warn	记录 Warn 级别的日志	×	×
2.23.7	Log_Error	记录 Error 级别的日志	×	×
2.23.8	Log_Fatal	记录 Fatal 级别的日志	×	×

2.23.1 Log_Set

语法：Log_Set Log_type, Filename [,Maxfilesize_or_Schedule] [,Maxbackupcount] [,Immediateflush]

描述：配置 Log 的方式、Log 文件的名称、1 个 Log 文件的大小或时间，Log 文件最大备份数，是否立即记录到磁盘。使用 Log 相关指令需先使用 Log_Set 进行配置。

参数：

- Log_type 日志的输出方式。类型为 Ushort，值范围说明如下：

0：日志输出到 Motion Studio 中的输出视窗中。

1：按大小回滚日志文件。文件输出的路径为...\Advantech\Motion_Runtime\Motion_Runtime 下。

2：按时间回滚日志文件。文件输出的路径为...\Advantech\Motion_Runtime\Motion_Runtime 下。

- Filename 输出日志文件的名称。Log_type 设置为 1 或 2 时，该项为输出日志文件的名称；Log_type 设置为 0 时，该项不起作用。

- Maxfilesize_or_Schedule 回滚日志文件的大小或回滚日志文件的时间间隔。

Log_type 设置为 1 时，该项为回滚日志文件的大小，单位为 M，默认为 2。即日志记录的输出文件大小最大为 2M。超过 2M，按先进先出的方式覆盖回滚。

Log_type 设置为 2 时，该项为回滚日志文件的时间间隔。时间间隔值如下说明，默认为 2，即按天记录。

0：MONTHLY，按月记录一个文件。超过一个月，后一个月的文件会覆盖前一个月的文件。

1：WEEKLY，按周记录一个文件。超过一周，后一周的文件会覆盖前一周的文件。

2：DAILY，按天记录一个文件。超过一天，后一天的文件会覆盖前一天的文件。

3：TWICE_DAILY，按两天记录一个文件。超过两天，后两天的文件会覆盖前两天的文件。

4：HOURLY，按小时记录一个文件。超过一小时，后一小时的文件会覆盖前一小时的文件。

5：MINUTELY，按分钟记录一个文件。超过一分钟，后一分钟的文件会覆盖前一分钟的文件。

- Maxbackupcount 日志文件最大备份数，默认为 1。比如 Log_type 设置为 1，按大小回滚日志，回滚日志文件的大小设置为 2M，那记录第 1 个 2M 文件后，该文件会备份下来，接着记录第 2 个文件，相当于可以保存最多 4M 的最近日志。该值不建议设置很大，设置很大，时间久了会容易引起磁盘被占的空间很大。

- Immediateflush 被记录的日志是否立即刷新到磁盘日志文件中。默认值为 False。

False：不立即刷新到磁盘日志文件，刷新的频率为系统内部定义。设置为 False 在突然退出 Runtime 的情况下会有机率丢失一些日志的情况。

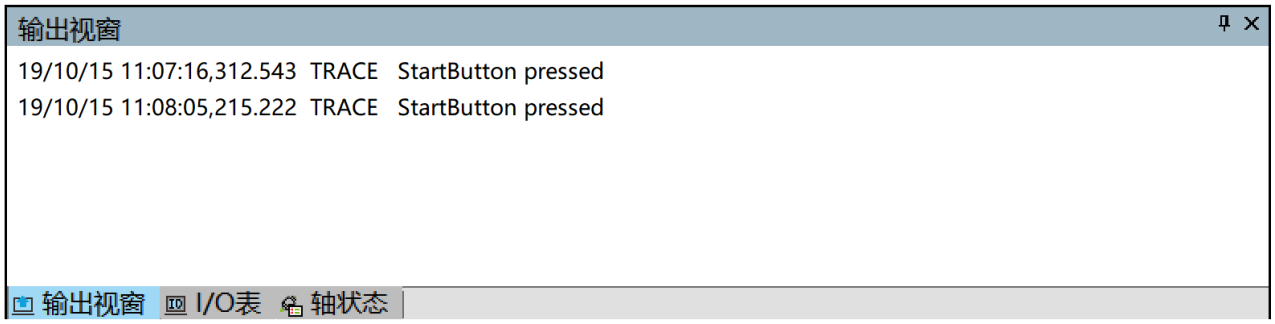
True：被记录后立即刷新到磁盘日志文件中。

例程 1

```
LOG_SET 0,"mylog",1,1,true   '日志输出到 Motion Studio 输出视窗
Log_SetLevel 0               '日志记录器过滤级别为 0，全开。
```

```
WHILE 1
  IF(VR(0)=1) THEN
    VR(0)=0
    LOG_Trace "StartButton pressed"  'VR(0)被触发为 1,记录 1 次"StartButton pressed"
  END IF
  SLEEP 10
WEND
```

'按以上代码将 VR(0)触发为 1，就会在 Motion Studio 的输出视窗中看到日志，如下图。



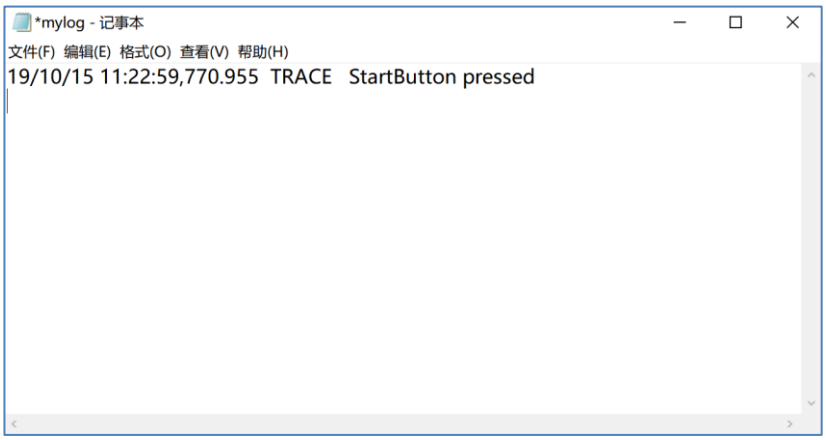
例程 2

'日志输出方式为按大小回滚日志，日志文件名称为 mylog，日志文件大小为 1M，备份数位 1，立即刷新。

```
LOG_SET 1,"mylog",1,1,true
Log_SetLevel 0                    '日志记录器过滤级别为 0，全开。

WHILE 1
  IF(VR(0)=1) THEN
    VR(0)=0
    LOG_Trace "StartButton pressed"  'VR(0)被触发为 1,记录 1 次"StartButton pressed"
  END IF
  SLEEP 10
WEND
```

'按以上代码将 VR(0)触发为 1，在…\Advantech\Motion_Runtime\Motion_Runtime 路径下会看到 1 个名为 mylog 的日志文件，里面被记录下日志，如下图。



2.23.2 Log_SetLevel

语法：Log_SetLevel loglevel

描述：设置 Log 记录器的过滤级别

参数：

- Loglevel Log 记录器的过滤级别。过滤级别共分 6 种，分别为 Trace 级别、Debug 级别、Info 级别、Warn 级别、Error 级别、Fatal 级别。Loglevel 设定值对应的说明如下，默认值为 6。

0：不过滤任何级别的日志。全部级别的日志记录都开放记录。

1：过滤 Trace 级别的日志。只记录 Debug 级别、Info 级别、Warn 级别、Error 级别、Fatal 级别的日志。

2：过滤 Trace 级别、Debug 级别的日志。只记录 Info 级别、Warn 级别、Error 级别、Fatal 级别的日志。

3：过滤 Trace 级别、Debug 级别、Info 级别的日志。只记录 Warn 级别、Error 级别、Fatal 级别的日志。

4：过滤 Trace 级别、Debug 级别、Info 级别、Warn 级别的日志。只记录 Error 级别、Fatal 级别的日志。

5：过滤 Trace 级别、Debug 级别、Info 级别、Warn 级别、Error 级别的日志。只记录 Fatal 级别的日志。

6：过滤所有级别的日志。不记录任何级别的日志。

例程

```
'日志输出方式为按大小回滚日志，日志文件名称为 mylog，日志文件大小为 1M，备份数位 1，立即刷新。
LOG_SET 1,"mylog",1,1,true
'过滤 Trace 级别、Debug 级别、Info 级别的日志。只记录 Warn 级别、Error 级别、Fatal 级别的日志
VR(0)=3
Log_SetLevel VR(0)          '设置过滤级别为 3
LOG_Trace "My Trace"        '因过滤级别为 3，该日志不会被记录
LOG_Debug "My Debug"        '因过滤级别为 3，该日志不会被记录
LOG_info  "My Info"         '因过滤级别为 3，该日志不会被记录
LOG_Warn  "My Warn"         '因过滤级别为 3，该日志会被记录
LOG_Error "My Error"        '因过滤级别为 3，该日志会被记录
LOG_Fatal "My Fatal"        '因过滤级别为 3，该日志会被记录
```

日志输出：

```
My Warn
My Error
My Fatal
```

2.23.3 Log_Trace

语法：Log_Trace log_info

描述：记录 Trace 级别的日志。

参数：

- log_info 日志信息，日志信息的格式为字符串或字符串与变量连接的格式。字符串与变量间要使用&连接符进行连接。如"VR(5) value is :"&VR(5)，如果 VR(5)的值为 3，那记录的日志内容就是 VR(5) value is:3。

例程

```
'日志输出到 Motion Studio 中的输出视窗中。
LOG_SET 0,"mylog"
'不过滤任何级别的日志
Log_SetLevel 0
LOG_Trace "My Trace"      '字符串的格式信息
LOG_Debug "The Position of Axis(0) is "&DPOS(0)      '字符串加变量的格式信息
LOG_info  "VR(5) value is :"&VR(5)&"_Myinfo_"      '字符串加变量加字符串的格式信息
LOG_Warn  "My Warn"
LOG_Error "My Error"
LOG_Fatal "My Fatal"
```

2.23.4 Log_Debug

语法：Log_Debug log_info

描述：记录 Debug 级别的日志。

参数：

- log_info 日志信息，日志信息的格式为字符串或字符串与变量连接的格式。字符串与变量间要使用&连接符进行连接。如"VR(5) value is :"&VR(5)，如果 VR(5)的值为 3，那记录的日志内容就是 VR(5) value is:3。

例程

'请参考 Log_Trace 指令章节的例程。

2.23.5 Log_Info

语法：Log_Info log_info

描述：记录 Info 级别的日志。

参数：

- log_info 日志信息，日志信息的格式为字符串或字符串与变量连接的格式。字符串与变量间要使用&连接符进行连接。如"VR(5) value is :"&VR(5)，如果 VR(5)的值为 3，那记录的日志内容就是 VR(5) value is:3。

例程

'请参考 Log_Trace 指令章节的例程。

2.23.6 Log_Warn

语法：Log_Warn log_info

描述：记录 Warn 级别的日志。

参数：

- log_info 日志信息，日志信息的格式为字符串或字符串与变量连接的格式。字符串与变量间要使用&连接符进行连接。如"VR(5) value is :"&VR(5)，如果 VR(5)的值为 3，那记录的日志内容就是 VR(5) value is:3。

例程

'请参考 Log_Trace 指令章节的例程。

2.23.7 Log_Error

语法：Log_Error log_info

描述：记录 Error 级别的日志。

参数：

- log_info 日志信息，日志信息的格式为字符串或字符串与变量连接的格式。字符串与变量间要使用&连接符进行连接。如"VR(5) value is :"&VR(5)，如果 VR(5)的值为 3，那记录的日志内容就是 VR(5) value is:3。

例程

'请参考 Log_Trace 指令章节的例程。

2.23.8 Log_Fatal

语法：Log_Fatal log_info

描述：记录 Fatal 级别的日志。

参数：

- log_info 日志信息，日志信息的格式为字符串或字符串与变量连接的格式。字符串与变量间要使用&连接符进行连接。如"VR(5) value is :"&VR(5)，如果 VR(5)的值为 3，那记录的日志内容就是 VR(5) value is:3。

例程

'请参考 Log_Trace 指令章节的例程。

第 3 章 附录

3.1 错误代码说明

3.1.1 RUN_ERROR 错误代码表

错误代码	说明
0x00000000	SUCCESS
0x80000000	InvalidDevNumber
0x80000001	DevRegDataLost
0x80000002	LoadDllFailed
0x80000003	GetProcAddrFailed
0x80000004	MemAllocateFailed
0x80000005	InvalidHandle
0x80000006	CreateFileFailed
0x80000007	OpenEventFailed
0x80000008	EventTimeOut
0x80000009	InvalidInputParam
0x8000000a	PropertyIDNotSupport
0x8000000b	PropertyIDReadOnly
0x8000000c	ConnectWinIrqFailed
0x8000000d	InvalidAxCfgVel
0x8000000e	InvalidAxCfgAcc
0x8000000f	InvalidAxCfgDec
0x80000010	InvalidAxCfgJerk
0x80000011	InvalidAxParVelLow
0x80000012	InvalidAxParVelHigh
0x80000013	InvalidAxParAcc
0x80000014	InvalidAxParDec
0x80000015	InvalidAxParJerk

0x80000016	InvalidAxPulseInMode
0x80000017	InvalidAxPulseOutMode
0x80000018	InvalidAxAlarmEn
0x80000019	InvalidAxAlarmLogic
0x8000001a	InvalidAxInPEn
0x8000001b	InvalidAxInPLogic
0x8000001c	InvalidAxHLmtEn
0x8000001d	InvalidAxHLmtLogic
0x8000001e	InvalidAxHLmtReact
0x8000001f	InvalidAxSLmtPEn
0x80000020	InvalidAxSLmtPReact
0x80000021	InvalidAxSLmtPValue
0x80000022	InvalidAxSLmtMEn
0x80000023	InvalidAxSLmtMReact
0x80000024	InvalidAxSLmtMValue
0x80000025	InvalidAxOrgLogic
0x80000026	InvalidAxOrgEnable
0x80000027	InvalidAxEzLogic
0x80000028	InvalidAxEzEnable
0x80000029	InvalidAxEzCount
0x8000002a	InvalidAxState
0x8000002b	InvalidAxInEnable
0x8000002c	InvalidAxSvOnOff
0x8000002d	InvalidAxDistance
0x8000002e	InvalidAxPosition
0x8000002f	InvalidAxHomeModeKw
0x80000030	InvalidAxCntInGp
0x80000031	AxInGpNotFound
0x80000032	AxIsInOtherGp

0x80000033	AxCannotIntoGp
0x80000034	GpInDevNotFound
0x80000035	InvalidGpCfgVel
0x80000036	InvalidGpCfgAcc
0x80000037	InvalidGpCfgDec
0x80000038	InvalidGpCfgJerk
0x80000039	InvalidGpParVelLow
0x8000003a	InvalidGpParVelHigh
0x8000003b	InvalidGpParAcc
0x8000003c	InvalidGpParDec
0x8000003d	InvalidGpParJerk
0x8000003e	JerkNotSupport
0x8000003f	ThreeAxNotSupport
0x80000040	DevIpoNotFinished
0x80000041	InvalidGpState
0x80000042	OpenFileFailed
0x80000043	InvalidPathCnt
0x80000044	InvalidPathHandle
0x80000045	InvalidPath
0x80000046	IoctlError
0x80000047	AmnetRingUsed
0x80000048	DeviceNotOpened
0x80000049	InvalidRing
0x8000004a	InvalidSlaveIP
0x8000004b	InvalidParameter
0x8000004c	InvalidGpCenterPosition
0x8000004d	InvalidGpEndPosition
0x8000004e	InvalidAddress
0x8000004f	DeviceDisconnect

0x80000050	DataOutBufExceeded
0x80000051	SlaveDeviceNotMatch
0x80000052	SlaveDeviceError
0x80000053	SlaveDeviceUnknow
0x80000054	FunctionNotSupport
0x80000055	InvalidPhysicalAxis
0x80000056	InvalidVelocity
0x80000057	InvalidAxPulseInLogic
0x80000058	InvalidAxPulseInSource
0x80000059	InvalidAxErcLogic
0x8000005a	InvalidAxErcOnTime
0x8000005b	InvalidAxErcOffTime
0x8000005c	InvalidAxErcEnableMode
0x8000005d	InvalidAxSdEnable
0x8000005e	InvalidAxSdLogic
0x8000005f	InvalidAxSdReact
0x80000060	InvalidAxSdLatch
0x80000061	InvalidAxHomeResetEnable
0x80000062	InvalidAxBacklashEnable
0x80000063	InvalidAxBacklashPulses
0x80000064	InvalidAxVibrationEnable
0x80000065	InvalidAxVibrationRevTime
0x80000066	InvalidAxVibrationFwdTime
0x80000067	InvalidAxAlarmReact
0x80000068	InvalidAxLatchLogic
0x80000069	InvalidFwMemoryMode
0x8000006a	InvalidConfigFile
0x8000006b	InvalidAxEnEvtArraySize
0x8000006c	InvalidAxEnEvtArray

0x8000006d	InvalidGpEnEvtArraySize
0x8000006e	InvalidGpEnEvtArray
0x8000006f	InvalidIntervalData
0x80000070	InvalidEndPosition
0x80000071	InvalidAxisSelect
0x80000072	InvalidTableSize
0x80000073	InvalidGpHandle
0x80000074	InvalidCmpSource
0x80000075	InvalidCmpMethod
0x80000076	InvalidCmpPulseMode
0x80000077	InvalidCmpPulseLogic
0x80000078	InvalidCmpPulseWidth
0x80000079	InvalidPathFunctionID
0x8000007a	SysBufAllocateFailed
0x8000007b	SpeedFordFunNotSpported
0x8000007c	InvalidNormVector
0x8000007d	InvalidCmpTimeTableCount
0x8000007e	InvalidCmpTime
0x8000007f	FWDownLoading
0x80000080	FWVersionNotMatch
0x80000081	InvalidAxParHomeVelLow
0x80000082	InvalidAxParHomeVelHigh
0x80000083	InvalidAxParHomeAcc
0x80000084	InvalidAxParHomeDec
0x80000085	InvalidAxParHomeJerk
0x80000086	InvalidAxCfgJogVelLow
0x80000087	InvalidAxCfgJogVelHigh
0x80000088	InvalidAxCfgJogAcc
0x80000089	InvalidAxCfgJogDec

0x8000008a	InvalidAxCfgJogJerk
0x8000008b	InvalidAxCfgKillDec
0x8000008c	NotOpenAllAxes
0x8000008d	NotSetServoComPort
0x8000008e	OpenComPortFailed
0x8000008f	ReadComPortTimeOut
0x80000090	SetComPortStateFailed
0x80000091	SevroTypeNotSupport
0x80000092	ReadComBufFailed
0x80000096	SlaveIOUpdateError
0x80000097	NoSlaveDevFound
0x80000098	MasterDevNotOpen
0x80000099	MasterRingNotOpen
0x800000c8	InvalidDIPort
0x800000c9	InvalidDOPort
0x800000ca	InvalidDOValue
0x800000cb	CreateEventFailed
0x800000cc	CreateThreadFailed
0x800000cd	InvalidHomeModeEx
0x800000ce	InvalidDirMode
0x800000cf	AxHomeMotionFailed
0x800000d0	ReadFileFailed
0x800000d1	PathBufIsFull
0x800000d2	PathBufIsEmpty
0x800000d3	GetAuthorityFailed
0x800000d4	GpIDAllocatedFailed
0x800000d5	FirmWareDown
0x800000d6	InvalidGpRadius
0x800000d7	InvalidAxCmd

0x800000d8	InvalidaxExtDrv
0x800000d9	InvalidGpMovCmd
0x800000da	SpeedCurveNotSupported
0x800000db	InvalidCounterNo
0x800000dc	InvalidPathMoveMode
0x800000dd	PathSelStartCantRunInSpeedForwareMode
0x800000de	InvalidCamTableID
0x800000df	InvalidCamPointRange
0x800000e0	CamTableIsEmpty
0x800000e1	InvalidPlaneVector
0x800000e2	MasAxIDSameSlvAxID
0x800000e3	InvalidGpRefPlane
0x800000e4	InvalidAxModuleRange
0x800000e5	DownloadFileFailed
0x800000e6	InvalidFileLength
0x800000e7	InvalidCmpCnt
0x800000e8	JerkExceededMaxValue
0x800000e9	AbsMotionNotSupport
0x800000ea	invalidAiRange
0x800000eb	AIScaleFailed
0x800000ec	AxInRobot
0x800000ed	Invalid3DarcFlat
0x800000ee	InvalidIpoMap
0x800000ef	DataSizeNotCorrect
0x800000f0	AxisNotFound
0x800000f1	InvalidPathVelHigh
0x80002000	HlmtPExceeded
0x80002001	HlmtNExceeded

0x80002002	SlmtPExceeded
0x80002003	SlmtNExceeded
0x80002004	AlarmHappened
0x80002005	EmgHappened
0x80002006	TimeLmtExceeded
0x80002007	DistLmtExceeded
0x80002008	InvalidPositionOverride
0x80002009	OperationErrorHappened
0x8000200a	SimultaneousStopHappened
0x8000200b	OverflowInPAPB
0x8000200c	OverflowInIPO
0x8000200d	STPHappened
0x8000200e	SDHappened
0x8000200f	AxisNoCmpDataLeft
0x10000001	Warning_AxWasInGp
0x10000002	Warning_GpInconsistRate
0x10000003	Warning_GpInconsistPPU
0x10000004	Warning_GpMoveDistanceCanntBeZero
0x80004001	DevEvtTimeOut
0x80004002	DevNoEvt
0x80005001	ERR_SYS_TIME_OUT
0x80005002	Dsp_PropertyIDNotSupport
0x80005003	Dsp_PropertyIDReadOnly
0x80005004	Dsp_InvalidParameter
0x80005005	Dsp_DataOutBufExceeded
0x80005006	Dsp_FunctionNotSupport

0x80005007	Dsp_InvalidConfigFile
0x80005008	Dsp_InvalidIntervalData
0x80005009	Dsp_InvalidTableSize
0x8000500a	Dsp_InvalidTableID
0x8000500b	Dsp_DataIndexExceedBufSize
0x8000500c	Dsp_InvalidCompareInterval
0x8000500d	Dsp_InvalidCompareRange
0x8000500e	Dsp_PropertyIDWriteOnly
0x8000500f	Dsp_NcError
0x80005010	Dsp_CamTableIsInUse
0x80005011	Dsp_EraseBlockFailed
0x80005012	Dsp_ProgramFlashFailed
0x80005013	Dsp_WatchdogError
0x80005014	Dsp_ReadPrivateOverMaxTimes
0x80005015	Dsp_InvalidPrivateID
0x80005016	Dsp_DataNotReady
0x80005017	Dsp_LastOperationNotOver
0x80005018	Dsp_WritePrivateTimeout
0x80005019	Dsp_FwIsDownloading
0x80005020	Dsp_FwDownloadStepError
0x80005101	Dsp_InvalidAxCfgVel
0x80005102	Dsp_InvalidAxCfgAcc
0x80005103	Dsp_InvalidAxCfgDec
0x80005104	Dsp_InvalidAxCfgJerk
0x80005105	Dsp_InvalidAxParVelLow
0x80005106	Dsp_InvalidAxParVelHigh
0x80005107	Dsp_InvalidAxParAcc
0x80005108	Dsp_InvalidAxParDec

0x80005109	Dsp_InvalidAxParJerk
0x8000510a	Dsp_InvalidAxPptValue
0x8000510b	Dsp_InvalidAxState
0x8000510c	Dsp_InvalidAxSvOnOff
0x8000510d	Dsp_InvalidAxDistance
0x8000510e	Dsp_InvalidAxPosition
0x8000510f	Dsp_InvalidAxHomeMode
0x80005110	Dsp_InvalidPhysicalAxis
0x80005111	Dsp_HlmtPExceeded
0x80005112	Dsp_HlmtNExceeded
0x80005113	Dsp_SlmtPExceeded
0x80005114	Dsp_SlmtNExceeded
0x80005115	Dsp_AlarmHappened
0x80005116	Dsp_EmgHappened
0x80005117	Dsp_CmdValidOnlyInConstSec
0x80005118	Dsp_InvalidAxCmd
0x80005119	Dsp_InvalidAxHomeDirMode
0x8000511a	Dsp_AxisMustBeModuloAxis
0x8000511b	Dsp_AxIdCantSameAsMasId
0x8000511c	Dsp_CantResetPosiOfMasAxis
0x8000511d	Dsp_InvalidAxExtDrvOperation
0x8000511e	Dsp_AxAccExceededMaxAcc
0x8000511f	Dsp_AxVelExceededMaxVel
0x80005120	Dsp_NotEnoughPulseForChgV
0x80005121	Dsp_NewVelMustGreaterThanVelLow
0x80005122	Dsp_InvalidAxGearMode
0x80005123	Dsp_InvalidGearRatio
0x80005124	Dsp_InvalidPWMDataCount
0x80005125	Dsp_InvalidAxPWMFreq

0x80005126	Dsp_InvalidAxPWMDuty
0x80005127	Dsp_AxGantryExceedMaxDiffValue
0x80005128	Dsp_ChanelIsDisable
0x80005129	Dsp_ChanelBufferIsFull
0x80005130	Dsp_ChanelBufferIsEmpty
0x80005131	Dsp_InvalidDoChanelID
0x80005132	Dsp_LatchHappened
0x80005201	Dsp_InvalidAxCntInGp
0x80005202	Dsp_AxInGpNotFound
0x80005203	Dsp_AxIsInOtherGp
0x80005204	Dsp_AxCannotIntoGp
0x80005205	Dsp_GpInDevNotFound
0x80005206	Dsp_InvalidGpCfgVel
0x80005207	Dsp_InvalidGpCfgAcc
0x80005208	Dsp_InvalidGpCfgDec
0x80005209	Dsp_InvalidGpCfgJerk
0x8000520a	Dsp_InvalidGpParVelLow
0x8000520b	Dsp_InvalidGpParVelHigh
0x8000520c	Dsp_InvalidGpParAcc
0x8000520d	Dsp_InvalidGpParDec
0x8000520e	Dsp_InvalidGpParJerk
0x8000520f	Dsp_JerkNotSupport
0x80005210	Dsp_ThreeAxNotSupport
0x80005211	Dsp_DevIpoNotFinished
0x80005212	Dsp_InvalidGpState
0x80005213	Dsp_OpenFileFailed
0x80005214	Dsp_InvalidPathCnt
0x80005215	Dsp_InvalidPathHandle

0x80005216	Dsp_InvalidPath
0x80005217	Dsp_GpSlavePositionOverMaster
0x80005218	Dsp_GpPathBufferOverflow
0x80005219	Dsp_InvalidPathFunctionID
0x8000521a	Dsp_SysBufAllocateFailed
0x8000521b	Dsp_InvalidGpCenterPosition
0x8000521c	Dsp_InvalidGpEndPosition
0x8000521d	Dsp_InvalidGpCmd
0x8000521e	Dsp_AxHasBeenInInGp
0x8000521f	Dsp_ThreeAxNotSupport
0x80005220	Dsp_InvalidPathRange
0x80005221	Dsp_InvalidNormVector

3.1.2 SYSTEM_ERROR 错误代码表

错误代码	说明
0	SUCCESS
0x90000000	AMI_NULL_PROJECT_EXIST
0x90000001	AMI_INVALID_INPUT_PARAMS
0x90000002	AMI_INVALID_RETURN
0x90000003	AMI_INVALID_CTRL_MODE
0x90000004	AMI_CONTROLLER_LOCKED
0x90000005	AMI_GET_MAC_FAILED
0x90000006	AMI_INVALID_COMMAND
0x90000007	AMI_SET_MEM_FAILED
0x90000008	AMI_GET_VERSION_FAILED
0x9000000a	AMI_CTRL_ENCODED_ALREADY
0x9000000b	AMI_CTRL_INVALID_PASSWORD
0x9000000c	AMI_GET_VARIABLE_FAILED
0x9000000d	AMI_NUM_CONVERT_FAILED
0x90000032	AMI_ACTION_NOT_ALLOWED

0x90000064	AMI_SOCK_TIME_OUT
0x90000065	AMI_LOAD_FILE_FAILED
0x90000066	AMI_DOWN_FILE_FAILED
0x90000067	AMI_LOAD_PROJECT_FAILED
0x90000068	AMI_DOWN_PROJECT_FAILED
0x90000069	AMI_SOCK_ALREADY_CONNECTED
0x9000006A	AMI_SOCK_COMMU_FAILED
0x90000096	AMI_CONNECTION_FAILED
0x90000097	AMI_DISCONNECTION_FAILED
0x90000098	AMI_SEND_COMMAND_TIMEOUT
0x900000C8	AMI_OPEN_FILE_FAILED
0x900000C9	AMI_CREATE_FILE_FAILED
0x900000CA	AMI_REMOVE_FILE_FAILED
0x900000CB	AMI_PATH_NOT_EXIST
0x900000CC	AMI_SET_NON_BLOCK_FAILED
0x900000CD	AMI_SET_BLOCK_FAILED
0x900000CE	AMI_CFG_FILE_NOT_EXISTS
0x900000CF	AMI_REF_FILE_NOT_EXISTS
0x900000D0	AMI_HEAD_FILE_NOT_EXISTS
0x900000D1	AMI_FILE_NOT_EXISTS
0x900000D2	AMI_FILE_INVALID_FORMAT
0x900000DC	AMI_PRJ_FILE_LOAD_FAILED
0x900000DD	AMI_SOURCE_FILE_NOT_EXISTS
0x900000DE	AMI_DST_FILE_EXISTS_ALREADY
0x900000FA	AMI_XML_LOAD_FAILED
0x900000FB	AMI_XML_CHECK_FAILED
0x900000FC	AMI_XML_SAVE_FAILED

0x900000FD	AMI_XML_ADD_FAILED
0x900000FE	AMI_XML_DELETE_FAILED
0x900000FF	AMI_XML_CREATE_FAILED
0x90000100	AMI_XML_INVALID_ELEMENT
0x9000012C	AMI_TASK_NOT_EXIST
0x9000012D	AMI_FORK_PROCESS_FAILED
0x9000012E	AMI_POPEN_FILE_FAILED
0x9000012F	AMI_STILL_RUNNING
0x90000130	AMI_NOT_IN_IDLE
0x90000131	AMI_GET_NO_ERROR
0x90000132	AMI_GET_NO_INFO
0x90000133	AMI_GET_MSG_NOTFINISHED
0x90000134	AMI_CREAT_PIPE_FAILED
0x90000136	AMI_RUN_FAILED
0x90000137	AMI_STOP_FAILED
0x90000138	AMI_NOT_RUNNING
0x90000140	AMI_DB_INIT_FAILED
0x90000141	AMI_DB_COMPILE_FAILED
0x90000142	AMI_DB_BREAKPOINT_FAILED
0x90000143	AMI_DB_CLEARPOINT_FAILED
0x90000144	AMI_DB_DELETEPOINTS_FAILED
0x90000145	AMI_DB_RUN_FAILED
0x90000146	AMI_DB_CONTINUE_FAILED
0x90000147	AMI_DB_NEXT_FAILED
0x90000148	AMI_DB_PROGRAM_NOT_RUN
0x90000149	AMI_DB_STOP_FAILED
0x9000014A	AMI_DB_OUT_OF_RANGE
0x9000014B	AMI_DB_EXIT_NOMARLLY

0x9000014C	AMI_DB_GET_LOCAL_VAR_FAILED
0x9000014D	AMI_DB_NOT_READY
0x9000014E	AMI_DB_ALREADY_PAUSED
0x9000014F	AMI_GET_RUNNING_TASKLIST_FAILED
0x90000190	AMI_MB_ILLEGAL_FUNCTION
0x90000191	AMI_MB_CRC_FAILED
0x90000192	AMI_MB_ILLEGAL_LENGTH
0x900001F4	AMI_MEM_UPDATE_VR_MBADDR_ERROR
0x900001F5	AMI_MEM_UPDATE_TABLE_MBADDR_ERROR
0x900001F6	AMI_MEM_UPDATE_DO_INIT_VALUE_ERROR
0x90000258	AMI_BASIC_RESET_ERROR
0x90000259	AMI_BASIC_INITIAL_ERROR
0x9000025A	AMI_BASIC_REFRESH_ERROR
0x9000025B	AMI_BASIC_GET_OFFSET_VALUE_FAILED
0xb0000000	AMI_GetSharedMemFailed
0xb0000001	AMI_GetTaskNameFailed
0xb0000002	AMI_IsNotInitialized
0xb0000003	AMI_IsAlreadyInitialized
0xb0000004	AMI_LoadXMLFailed
0xb0000005	AMI_ParseXMLFailed
0xb0000006	AMI_CreateDevListFailed
0xb0000007	AMI_InitializeDeviceFailed
0xb0000008	AMI_InitializeSharedMemFailed
0xb0000009	AMI_RefreshSharedMemFailed
0xb000000a	AMI_SetDeviceCfgFailed

0xb000000b	AMI_IncorrectCommand
0xb000000c	AMI_DeviceLargerList
0xb000000d	AMI_SerialPortError
0xb000000e	AMI_EthernetError
0xb000000f	AMI_LogOpenFailed
0xb0000010	AMI_StartTaskFailed
0xb0000011	AMI_StopTaskFailed
0xb0000012	AMI_CreatEventFailed
0xb0000013	AMI_CreatEThreadsFailed
0xb0000014	AMI_AllocatePMotionInfoFiled
0xb0000015	AMI_FAILEDTOCHECKEVENT
0xb0000016	AMI_FailedCloseCheckingEventThread
0xb0001000	AMI_CardsNotFound
0xb0001001	AMI_MotionBoardIDNotFound
0xb0001002	AMI_AxesOrGroupCountNotFound
0xb0001003	AMI_AxisIDorPhyIDNotFound
0xb0001004	AMI_GroupNotFound
0xb0001005	AMI_AxisInfoError
0xb0001006	AMI_MotionDeviceCountError
0xb0001007	AMI_InputBoardIDNotFound
0xb0001008	AMI_InputDeviceCountError
0xb0001009	AMI_InDAQdeviceCountError
0xb000100a	AMI_OutputBoardIDNotFound
0xb000100b	AMI_OutputDeviceCountError
0xb000100c	AMI_OutDAQdeviceCountError
0xb000100d	AMI_ActualDeviceCountError
0xb0002000	AMI_WrongAxisIndex

0xb0002001	AMI_WrongDoIndex
0xb0002002	AMI_WrongDiIndex
0xb0002003	AMI_NoAxis
0xb0002004	AMI_AxisInDifferentDevice
0xb0002005	AMI_WaitModeNotMatch
0xb0002006	AMI_MasterAxisIndexError
0xb0002007	AMI_GANTRYAxisNotInSameDev
0xb0002008	AMI_GEARAxisNotInSameDev
0xb0002009	AMI_AddPathAxisCntError
0xb000200a	AMI_AddPathHELIX3PnotSupport
0xb0003000	AMI_EthernetModeError
0xb0003001	AMI_EthernetOpened
0xb0003002	AMI_EthernetOpenFailed
0xb0003003	AMI_EthernetCloseFailed
0xb0003004	AMI_EthernetWrongNum
0xb0003005	AMI_EthernetNotOpen
0xb0003006	AMI_EthernetReadFailed
0xb0003007	AMI_EthernetResetFailed
0xb0003008	AMI_EthernetWriteFailed
0xb0003009	AMI_EthernetReadVRFailed
0xb000300a	AMI_EthernetWriteVRFailed
0xb0003100	AMI_SerialPortWrongID
0xb0003101	AMI_SerialPortOpenFailed
0xb0003102	AMI_SerialPortCloseFailed
0xb0003103	AMI_SerialPortNotOpen
0xb0003104	AMI_SerialPortWrongCfg
0xb0003105	AMI_SerialPortSetCfgFailed
0xb0003106	AMI_SerialPortWriteFailed

0xb0003107	AMI_SerialPortReadFailed
0xb0003108	AMI_SerialPortResetFailed
0xb0003109	AMI_SerialPortWriteVRFailed
0xb000310a	AMI_SerialPortReadVRFailed

3.2 MAS 控制器运动功能支持列表

项目	说明	PCI-1245L-MAS	PCI-1245-MAS	PCI-1285-MAS	MVP-3245/85-MAS
单轴运动	单轴点位运动	✓	✓	✓	✓
	单轴定速运动	✓	✓	✓	✓
	变位运动	✓	✓	✓	✓
	变速运动	✓	✓	✓	✓
	背隙补偿	✓	✓	✓	✓
	T&S 形速度曲线	✓	✓	✓	✓
	运动中重设轴的加速度和减速度	✓	✓	✓	✓
	同步运动	不支持	✓	✓	✓
	迭加运动	不支持	不支持	不支持	不支持
	JOG 运动	✓	✓	✓	✓
	手轮运动	✓	✓	✓	✓
	回原点运动(16 种模式)	✓	✓	✓	✓
	DI 触发停止	✓	✓	✓	✓
插补功能	直线插补	2 轴	2~3 轴	2~3 轴	2~3 轴
	直接线性插补	2 轴	2~4 轴	2~8 轴	2~4 轴
	2 轴圆弧插补	不支持	✓	✓	✓
	3 轴圆弧插补	不支持	不支持	不支持	不支持
	螺旋插补	不支持	✓	✓	✓
	运动中改变组的运行速度	不支持	✓	✓	✓
同步运动	电子齿轮	不支持	✓	✓	✓
	电子凸轮	不支持	✓	不支持	不支持
	龙门	不支持	✓	✓	✓
	CAM DO	不支持	✓	✓	✓

	切向跟随	不支持	✓	✓	✓
	PathLink	不支持	✓	不支持	不支持
路径运动	加载路径功能	不支持	支援最多 10000 个点	支援最多 7000 个点	支援最多 10000 个点
	添加直线运动	不支持	✓	✓	✓
	添加圆弧运动	不支持	✓	✓	✓
	添加螺旋运动	不支持	✓	✓	✓
	延迟功能	不支持	✓	✓	✓
	添加 DO 开关运动	不支持	✓	✓	✓
	路径速度交接功能	不支持	✓	✓	✓
	路径速度前瞻功能	不支持	不支持	不支持	不支持
比较触发功能	单点比较	不支持	✓	✓	✓
	等距线性比较	不支持	✓	✓	✓
	不等距随机点比较	不支持	✓	✓	✓
	多轴比较触发	不支持	不支持	不支持	✓
等待事件	单轴/插补运动的停止运行功能	✓	✓	✓	✓
	轴的比较	不支持	✓	✓	✓
	轴的锁存	不支持	✓	✓	✓
	轴的锁存缓存	不支持	✓	✓	✓
输入/输出功能	DI	16	16	32	32
	DO	16	16	32	32
	AI	不支持	不支持	不支持	不支持
	AO	不支持	不支持	不支持	不支持

3.3 数据类型及类型转换指令

Motion BASIC 常用数据类型说明

数据类型	说明
BOOLEAN	布尔数据类型，True 或者 False
BYTE	长度为 8 位的整数数据类型
UBYTE	长度为 8 位的无符号整数数据类型
DOUBLE	长度为 64 位的双精度浮点数据类型
INTEGER	长度为 32 位或 64 位的整数数据类型
UINTEGER	长度为 32 位或 64 位的无符号整数数据类型
LONG	长度为 32 位的整数数据类型
ULONG	长度为 32 位的无符号整数数据类型
SHORT	长度为 16 位的整数数据类型
USHORT	长度为 16 位的无符号整数数据类型
UNSIGNED	整数类型有无符号的修饰符
STRING	字符串类型
SINGLE	长度为 32 位的单精度浮点数据类型

数据类型转换指令

指令	说明
CBYTE	将数字或字符串的表达式转换成字节类型数据
CDBL	将数字或字符串的表达式转换成双精度浮点数
CHR	返回用 ASCII 码表达的值对应的字符
CINT	将数字或字符串表达式转换成整型数据
CLNG	将数字或字符串表达式转换成长整型数据
CLNGINT	将数字或字符串表达式转换成 64 位长整型数据
CSNG	将数字或字符串的表达式转换成单精度浮点数
CUBYTE	将数字或字符串表达式转换为无符号字节类型数据
CUINT	将数字或字符串表达式转换为无符号整型数据
CULNG	将数字或字符串表达式转换为无符号长整型数据

CULNGINT	将数字或字符串表达式转换为无符号 64 位长整型数据
CUNSG	将一个表达式转换成对应的无符号类型
CUSHORT	将数字或字符串表达式转换为无符号短整型数据
VALINT	将一个字符串转换为一个 INTEGER 类型数据
VAL	将一个字符串转换为一个浮点数
HEX	将一个数用十六进制数返回
OCT	将一个数用八进制数返回
VALLNG	将一个字符串转换为一个 LONG 类型数据
VALUINT	将一个字符串转换为一个 UINTEGER 类型数据
VALULNG	将一个字符串转换为一个 ULONG 类型数据
ASC	返回字符串中字符的 ASCII 码

3.4 MAS 控制器中的控制单元

MAS 控制器中的运动控制硬件是由一个个控制单元组成的，一个 MAS 控制器中会有一个控制单元或多个控制单元。通常一个控制单元是指一张 MAS 运动控制卡，目前 MAS 控制器支持的控制单元如下列表：

序号	MAS 控制器支持的控制单元	轴数	轴上的 DI/DO 数	设备的 DI/DO 数
1	PCI-1245L-MAS	4	16DI/16DO	0
2	PCI-1245-MAS	4	16DI/16DO	0
3	PCI-1285-MAS	4	32DI/32DO	0
4	MVP-3245-MAS	4	16DI/16DO	16DI/16DO
5	MVP-3265-MAS	6	16DI/16DO	8DI/8DO
6	MVP-3285-MAS	8	16DI/16DO	0
7	PCI-1203-06MAS	6	0	8DI/4DO
8	PCI-1203-10MAS	10	0	8DI/4DO
9	PCI-1203-16MAS	16	0	8DI/4DO
10	PCI-1203-32MAS	32	0	8DI/4DO
11	AMAX-3285	8	0	16DI/16DO

控制单元 DI/DO 说明：

轴上的 DIO：上表中序号 1~6 的控制单元属于直接脉冲输出控制的控制单元，这种控制单元上的有些 DIO 是跟轴控制相关的，比如可以将某些 DO 启用“位置比较输出功能”，我们称为“轴上的 DIO”。通常情况下每个轴有 4 个 DI,4 个 DO 是“轴上的 DIO”。

设备的 DIO：设备的 DIO 是指控制单元中除“轴上的 DIO”外，其它的通用 DIO 接口。

3.5 MAS 控制器支持的 I/O 卡（研华 DAQ 系列）

MAS 控制器中需要插入研华 DAQ 系列的 I/O 卡来扩展 I/O 时，分两大类：

1. MAS 控制器直接能识别并映射出 I/O 指令。支持的卡片如下表格：

I/O 卡型号	插槽类型	DI 数量	DO 数量	AI 数量	AO 数量
PCI-1750	PCI	16	16	0	0
PCI-1756	PCI	32	32	0	0
PCIE-1730	PCIE	16	16	0	0
PCIE-1752	PCIE	0	64	0	0
PCIE-1754	PCIE	64	0	0	0
PCIE-1756	PCIE	32	32	0	0
PCIE-1758DI	PCIE	128	0	0	0
PCIE-1758DO	PCIE	0	128	0	0
PCIE-1758DIO	PCIE	64	64	0	0

2. MAS 控制器不能直接识别，需使用 DAQ 类控制指令来编程控制。支持的卡片为：除上述第 1 类支持表格以外的其它研华 DAQ 系列 I/O 卡。