软　件

KRC1/KRC2/KRC3

参考指导

版本**4.1**

目　录

1.概述

1.1排版惯例

1.2绘图惯例

下面格式用于显示手册中的语法

实例	说明
IF,THEN,TRIGGER,...	关键字和特征用粗体印刷
Signal,Interface_Name, DataType,...	可选择的命令名称用斜体印出
Name, Distance, Time, Priority, ...	必须由程序中指定的信息替换的字符
DELAY=Time , <**ELSE**,...	在方括号中的选择基础
+|-	互相独立的选择用“**OR**”或“|”分开

1.2 图解惯例

下面这些符号将贯穿这本手册

这“范例”符号用于描述和说明实际的例子

交叉参考手册中包含了更多说明和解释的部分和章节。

NOTE图标用于突出某个题目的一般性或辅助性说明，或是对特殊性提出注意。

TIP图标表示包含有助于简化工作的建议的文字段落。

这个标记的意义是：应该注意某个特别的提示。一般来说，遵循这个提示将使进行的工作容易完成。

这个标记的意义是：如果不遵守或严格遵守操作说明、工作指示规定的操作顺序和诸如此类的规定，可能会导致机器人系统的损坏。

这个符号用于设备失灵或不遵守操作说明和工作说明规定的顺序操作，将使你受到伤害或丧命。

2 参考部分

2.1 基础

2.1.1 程序、数据列表和模块

KRC存储程序代码存储在扩展名为**SRC**扩展名的文件中。固定不变的数据保存在扩展名为**DAT**的文件中。在一个模块可以由同名的**SRC**和**DAT**文件组成。

2.1.2 名字和文字

一个文字表示实际的值。例如：符号“1”表示数字“one”。一个名字或名称指定一个数据对象表示包含值(例如：变量)或确定的值 (例如：常量)。

一个常量和变量名有以下的限制：

- 它最多24个字符
- 它用字母(A--Z),数字(0--9)和符号“_”和“$”组成.
- 它不能用数字开头
- 它必须不能是关键字

2.1.3 数据类型

有两个不同组的数据类型：例如：**INT，**它是在系统中预先确定的和用户定义的数据类型，它以 **ENUM**和**STRUC** 类型为转变基础。

不同两类的两个例子:

- 在系统中预先定义的数据类型是有效的，由用户定义的数据类型仅在局部有效，除非关键字**GLOBAL**被用于它们的声明中或它们在$CONFIG.DAT文件中被声明。
- 当声明的数据类型是系统预先给定的时，关键字**DECL**可以被省略

2.1.3.1 简单数据类型

数据类型	关键字	意义	值的范围
Integer	**INT**	整数	-2^{31}--1...2^{31}--1
Real	**REAL**	浮点数	1.1E--38...3.4E+38
Boolean	**BOOL**	逻辑状态	TRUE, FALSE
Character	**CHAR**	字符	ASCII字符

2.1.3.2 固有的类型转换

如果运算的两者都是整数类型，运算的结果是**INT**类型。如果整数除法的结果不是整数，将切除小数。

如果操作数中有一个为实数类型（**REAL**），那么结果为实数类型**REAL**。

操作数	INT	REAL
INT	INT	REAL
REAL	REAL	REAL

2.1.3.3 预先确定的数据类型

下面运动数据是在软件控制器中预先确定的。

STRUC AXIS REALA1,A2,A3,A4,A5,A6

用A1~A6组成的结构AXIS表示机器人1～6轴移动的角度值（转动轴）或移动值（移动轴）。

STRUC E6AXIS REALA1,A2,A3,A4,A5,A6,E1,E2,E3,E4,E5,E6

外部轴的角度值和移动值被存储在附加的E1到E6部分中。

STRUCFRAMEREALX,Y,Z,A,B,C

空间坐标系被存在X,Y,Z中，极坐标系被存在A,B,C中。

STRUCPOSREALX,Y,Z,A,B,C,INTS,T

用附加的**S**(状况)和**T**(转动)明确定义轴的位置　。

STRUCE6POSREALX,Y,Z,A,B,C,E1,E2,E3,E4,E5,E6,INT S,T

2.1.3.4固有的数据类型分配

如果一个变量名没有首先在**KRL**程序中声明，他将自动分配为**POS**类型.

固定数据类型不用特别注意，在编程中很容易遵守。

2.1.4 常量

常量的名字、数据类型和数值不会改变下面的设定值。

常量必须在数据列表中被定义和初始化。

为了可以使用常数，配置文件CONST_KEY必须为TRUE：

CONST_KEY=TRUE

2.1.5 变量

变量名在数值和内存区都可以被改变

2.1.5.1 系统变量

KRL系统变量名起始都要用 “$”；这个特征不能用于用户变量名的开始。

2.1.6 操作

下面是可能用到的数据运算:

2.1.6.1 算术操作

符号	功能
+	加法
—	减法
/	除法
*	乘法

2.1.6.2 逻辑操作

符号	功能
NOT	相反的（逻辑非）
AND	逻辑与
OR	逻辑或
EXOR	异或

2.1.6.3 关系操作

符号	功能
==	等于
<	小于
>	大于
<=	小于等于
>=	大于等于
<>	不等于

2.1.6.4 位操作

符号	功能
B_NOT	逐位取反
B_AND	逐位与
B_OR	逐位或
B_EXOR	逐位异或

2.1.6.5 几何操作

符号	功能
:	在**FRAME**与**POS**数据类型间进行矢量加法（几何加法）

2.1.6.6 操作优先级

在超过一个操作的复杂的表达式中，有些单独的表达式在命令中会被优先执行。

优先级	操作符
1(最高级)	NOT，B_NOT
2	*，/
3	+，--
4	AND，B_AND
5	EXOR，B_EXOR
6	OR，B_OR
7(最低级)	==，<，>，<=，>=，<>

下面这些规则同样适用：

- 相同的运算从头开始.
- 不同的运算按运算符的优先权执行.
- 逻辑运算的左右两边优先执行.

2.1.7 声名

声明指定分配一个名字的数据类型。

2.1.8 初始化

初始化赋值

2.1.9 表达式

数据对象由自己的数据类型和数值所构成

- 如果结果是**INT**或**REAL**，那么是算术表达式.
- 如果结果是**BOOL**，那么是逻辑表达式
- 如果结果是 **POS**, **E6POS**, **AXIS**或**E6AXIS**，那么是几何表达式.

2.1.10 语句

语句是那些不表现自己的确定的值和数据类型的命令。简单的语句只包含一个命令行。而复杂的语句则包含完整的控制结构。

2.1.11 注释

注释是指被程序编辑器忽略的文本。使用“；”字符在程序中把注释和程序区分出来。

2.1.12 运动编程

机器人编程语言的一个显著的特征是编程点与机器人**TCP**(刀具中心点)之间的运动。它有以下两种基本的运动模式：

2.1.12.1 点到点的运动(PTP=点到点)

机器人的主要轴，使用最大的加速度和速比到达目标点。它不保持直线轨迹。

2.1.12.2 连续路径运动(CP=连续轨迹)

机器人手臂从起始点到目标点的轨迹路径为直线（**LIN**）或圆弧（**CIRC**）。这主要是依靠程序对速度和加速度的控制，而且从起始点到目标点之间不使用近似定位。

2.1.13 控制结构

控制语句可以用于影响程序的执行。在程序段中它依靠条件而执行。**IF ELSE**就是一个例子。

2.1.14 子程序

子程序是主程序的分支。当一个子程序被调用时，子程序中的命令立即被重新执行。

除主程序外，更多的子程序被定义在**SRC**文件中。如果它们和程序卡中**SRC**文件的程序名相同，那么主程序是经过认证的。如果**SRC**中的子程序是全局有效的，那么必须用关键字**GLOBAL**。

2.1.15 函数

函数能分配数据类型，函数能像子程序一样被程序调用。

2.1.16 块结构

块由**KRL**程序语言组成。它包括指令、参数和注释。这些语句、声明在系统中逐块的执行。**KRL**块的建立必须遵循一定的规律。它规定块的结构要符合语法描述的说明和指令索引中语句的规定。

在一个块中包含：

- 声明
- 语句
- 注释

空的块也是可以使用。它可以只包含一个块结束的字符。

块的开始行没有任何的标识符。块也可以由一个或多个空格开始，但块的结尾必须键入**RETURN**字符。如果程序块太长，屏幕显示不下时，系统将自动换行。在程序块中每行最长474个字符。

2.1.17 有效性区域

如果常量、变量、子程序、函数和变量是全局的，而且读入的所有**KRL**程序都是经过认证的，那么必须用**GLOBAL**关键字。否则数据对象只是局部有效的，只有一个例外：在CONFIG.DAT文件中被激活的变量也是全局有效的。

局部数据的有效性：

- 在程序中激活的局部变量是合法的，它被置于关键字**DEF**和**ENDDEF**中。
- 在数据列表中被承认的常量是有效的。
- 子程序和函数在参与**SRC**文件的主程序中是有效的。
- 一个局部中断只有在优先级比程序级别低或者相等时，它是被承认的。

如果局部和全局变量名相同，那么编辑器在局部变量名有效的区域内使用局部变量名。

全局变量和常量都在公告在数据列表中。

为了可以使用关键字GLOBAL，在文件“Progress.ini”中的全局选项在INIT选项中必须设置位TRUE:

GLOBAL_KEY=TRUE

2.1.18 关键字

关键字的字母排列有固定的功能。它们在上面的印刷中用粗体描述。

下面是保留和不保留的关键字：

- **保留的关键字:**不能用于与其内在含义不同的场合。多数时候他们不能用于文件名。在下表中列出了一些被编辑器认可但不用于系统的关键字。它们虽然没有被执行，但仍然保留
- **非保留的关键字:**受到上下文的限制。当非保留的关键字出现在文章中时，上下文可以解释他的含义。此外，非保留关键字还可以解释为一个名字。但为了避免混淆，非关键字一般不用作名字。

下表中列出了在**KRL**机器人程序语言中和声明语句中使用了的关键字：

关键字	功能	摘要信息
ANIN	语句	循环读入模拟输入。
ANOUT	语句	操作控制模拟输出。
BRAKE	语句	在中断程序中制动机器人的运动。
CASE	语句	在**SWITCH**语句中开始一个分支。
CCLOSE	语句	关闭通道。
CHANNEL	声明	声明输入和输出的通道名。
CIRC	语句	圆形运动。
CIRC_REL	语句	相对目标坐标系的圆形运动。
CONFIRM	语句	承认确认的消息。
CONTINUE	语句	防止先前的停止运行。
COPEN	语句	操作输入／输出通道。
CREAD	语句	从通道读入数据。
CWRITE	语句	从通道写入数据。
DECL	声明	变量的声明和排列。
DEF	定义	程序和子程序的声明。
DEFAULT	语句	在**SWITCH**语句中开始默认的分支。
DEFDAT	定义	数据声明列表。
DEFFCT	定义	声明函数。
DELAY	参数	在**TRIGGER**和**ANOUT**语句中启动延时。
DIGIN	语句	循环读入数字输入。
DISTANCE	参数	在**TRIER**语句中，启动开关点。
DO	语句	开始在**INTERRUPT**声明中调用中断程序和调用子程序或在**TRIGGER**语句中分配一个值。
ELSE	语句	在ＩＦ语句中开始另一个语句分支。
END	语句	子程序结束（参考 **DEF**）。
ENDDAT	语句	数据列表结束（参考 **DEFDAT**）。
ENDFCT	语句	函数结束（参考 **DEFFCT**）。
ENDFOR	语句	ＦＯＲ循环语句结束。

ENDIF	语句	ＩＦ分支结束。
ENDLOOP	语句	ＬＯＯＰ结束。
ENDSWITCH	语句	结束ＳＥＩＴＣＨ传送。
ENDWHILE	语句	结束ＷＨＩＬＥ环语句。
ENUM	声明	列举类型的声明。
EXIT	语句	无条件退出循环。
EXT	声明	外部子程序的声明。
EXTFCT	声明	外部函数的声明。
FOR	语句	计算循环或以ＷＡＩＴ语句为条件的开始。
GLOBAL	声明	全局声明。
GOTO	语句	无条件的跳跃语句。
HALT	语句	中断程序的执行和暂停。
IF	语句	语句的执行依靠逻辑表达式的结果。
IMPORT	声明	在数据列表中输入变量。
INTERRUPT	语句	中断功能的定义和它的激活和解除。
IS	语句	初始化**IMPORT**声明中的源属性。
LIN	语句	直线运动。
LIN_REL	语句	在相对坐标系中的直线运动。
LOOP	语句	无穷循环。
MAXIMUM	参数	输出模拟量最小值的关键字。
MINIMUM	参数	输出模拟量最大值的关键字。
PRIO	参数	在**TRIGGER**语句中调用子程序时，启动规定的优先权。
PTP	语句	点到点的运动。
PTP_REL	语句	相对坐标系中的点到点的运动。
PULSE	语句	激活脉冲输出。
REPEAT	语句	循环程序总是最少执行一次(不拒绝循环)。中止条件在循环中止时选择。
RESUME	语句	子程序和中断程序的中断。

RETURN	语句	从函数和子程序的返回。
SEC	语句	开始在ＷＡＩＴ语句中规定的时间。
SIGNAL	声明	输入和输出信号名的声明。
SREAD	语句	中止数据进入它的组成部分。
STRUC	声明	结构类型的声明。
SWITCH	语句	在几个语句分支中选择。
SWRITE	语句	组合数据形成数据设置。
THEN	语句	在ＩＦ语句中开始第一个语句。
TO	语句	在**FOR**语句中最初和最后的值和开始在**SIGNAL**语句中指出的数字输入/输出。
TRIGGER	语句	通道触发开关的作用是使机器人的运动同步。
UNTIL	语句	在**REPEAT**循环中开始**“end”**质询。
WAIT	语句	等待继续的条件或指定的周期时间。
WHEN	语句	在**INTERRUPT**声明中开始逻辑表达式和在**TRIGGER**中指出的轨迹距离标准。
WHILE	语句	程序循环；在选择循环开始时选择循环终止的条件（拒绝循环）。

2.2 命令索引

2.2.1 ANIN

2.2.1.1 摘要信息

循环读入模拟输出

2.2.1.2 语法

读入模拟输出：

```
ANIN ON Signal_Value = Factor * Signal_Name ⟨± Offset
```

中止读操作

```
ANIN OFF Signal_Name
```

	类型	解释
Signal_Value	REAL	循环读入的结果存储在信号值中。信号值可以是变量、模拟信号或前面说明过的信号。
Factor	REAL	常量、变量或说明过得信号可以成为要素。
Signal_Name	REAL	信号名指定输入模拟信号的信号名。
Offset	REAL	偏移量可以是变量、常量、模拟信号或前面说明的信号.

在ANIN语句中的所有的变量必须在数据列表中公告。

在特定的时间内最多有3个ANIN ON被激活。

2.2.1.3描述

模拟模块产生能通过定义变量$ANIN[1]到$ANIN[8]的手段读入的模拟界面。模拟输入能在时间周期内使用**ANIN**或使用“＝”操作符的手段，使变量能分配一次**REAL**类型。在用**ANIN**语句的循环读入的例子中，模拟界面在低优先级循环读入（12ms）。同时间内可以有三个**ANIN ON**语句被使用。两个**ANIN**语句能写入相同的信号值或模拟界面。它可以联合其他符合逻辑的操作模拟输入和用**ANIN**语句中可选的算术方法分配信号值。

模拟模块有8个模拟接口，接口有12位的分辨率（4。88mV 非电器隔离）。输入电压从－10V到+10V间变化，但不能超过35V。硬件输入能使用系统参数的手段分配通道数。访问模拟输入可触发先前的运行停止。

如果电压高过35V，模拟模块或它的部件会被损坏。

2.2.1.4实例

路径修正(系统变量$TECHIN[1])依靠传感器$ANIN[2]的输入来校正。传感器输入$ANIN[2]是符合逻辑的变量与符号变量SIGNAL_1的组合。增加变量FACTOR和SIFNAL_1的电流值是附加的变量OFFSET和循环写入路径修正的系统变量$TECHIN[1]的结果。

```
SIGNAL SIGNAL_1 $ANIN[2]
ANIN ON $TECHIN[1] = FACTOR * SIGNAL_1 + OFFSET
```

循环扫描SIGNAL_1使用ANIN OFF指令结束。

```
ANIN OFF SIGNAL_1
```

SIGNAL，ANOUT，DIGIN

2.2.2 ANOUT

2.2.2.1 摘要信息

模拟输出控制

2.2.2.2 语句

开始模拟输出：

```
ANOUT ON Signal_Name = Factor * Control_Element ⟨± Offset
      ⟨DELAY = ±Time
      ⟨MINIMUM = Minimum_Value ⟨MAXIMUM = Maximum_Value
```

模拟输出结束：

```
ANOUT OFF Signal_Name
```

自变量	类型	解释
Signal_Name	REAL	信号名用**SIGNAL**语句访问模拟输出表示的信号定义。 模拟输出$ANOUT[1]不能直接输入。
Factor	REAL	系数可以是变量，信号说明，模拟信号和常量。
Control_ Element	REAL	控制要素可以是变量，信号声明，模拟信号。.
Offset	REAL	偏移量可以是变量，信号声明或常量。
Time	REAL	时间是指定真实的秒数
Minimum_Value, Maximum_Value	REAL	最小值和最大值必须在－0.1～+0.1之间和符合限定的输出。实际的值不能低于或高于指定的最小值和最大值，甚至如果计算出的值不能超出这个范围。 当然，最小值总是比最大值小而且关键字**MINMUN**，**MAXIMUN**的顺序必须保持。

所有用于AUOUT语句的变量必须在数据列表中声明。

2.2.2.3 描述

不像二进制或数字的输出，模拟输出不控制**ANOUT**语句分配的值。信号名定义**SIGNAL**语句的使用。

表达式指定计算用的模拟值是循环计算的。它不能太大，以便表达式可以在一个循环周期内计算出。表达式的结果必须在-1～+1或0～+1的范围内，以对应硬件的配置。如果表达式的结果超过它的限定，输出相关的最后值，那么显示“**LIMIT SIGNAL MAME**”提示信息，直到结果再次降到界限下。可以用关键字**MINIMUM**和**MAXIMUM**表示定义输

出的界限值。

可选择的关键字**DELAY**可用于程序的否定或肯定的延迟。延迟的值使用指定的真实的值表示多少秒。

机器人控制器规定16个模拟输出($ANOUT[1]...$ANOUT[16]) 作为标准。

模拟输出可以读和设置。

2.2.2.4 范例

模拟输出$ANOUT[5]被分配符号名ANALOG_1。当模拟值输出被激活后，产生FACTOR变量和系统变量$TECHVAL[1]，增加OFFSET_1变量的值，循环计算并写入模拟输出$ANOUT[5]中。模拟输出的结束用ANOUT OFF ANALOG_1。

请注意： $TECHVAL[i]提供未滤波信号的通讯技术功能的次序，$TECH_ANA_FLT_OFF[i] 必须设为**TRUE**。

```
SIGNAL ANALOG_1 $ANOUT[5]
ANOUT ON ANALOG_1 = FACTOR * $TECHVAL[1] + OFFSET_1
...
ANOUT OFF ANALOG_1
```

模拟输出$ANOUT[1]被分配符号名ADHESIVE用于控制粘贴速度比率的分配。控制值是由一半轨迹速率计算出的(系统变量 $VEL_ACT；为了获得纯正轨迹速率信号，$VEL_FLT_OFF必须为 TRUE)而且常数增加0.2。循环输出信号使用选择参数DELAY延时0.05秒。模拟输出使用ANOUT OFF ADHESIVE结束。

```
SIGNAL ADHESIVE $ANOUT[1]

ANOUT ON ADHESIVE = 0.5*$VEL_ACT + 0.2 DELAY = 0.05
...
ANOUT OFF ADHESIVE
```

ANIN，SIGNAL

2.2.3 BRAKE

2.2.3.1 摘要信息

在中断程序中打断机器人的运动。

2.2.3.2 语句

```
BRAKE 〈F
```

自变量	类型	解释
F		指定参数**F**（快速制动）将引起机器人强制制动，就像紧急停止。 如果轨迹操作停止用机床数据$EMSTOP_PATH选择操作模式的配置，那么机器人在最佳的时间内制动。

2.2.3.3 描述

BRAKE语句被用于在中断程序中制动机器人的运动，直到它被激活。

制动运动就像用操作减速一样。

如果自变量没有被指定，制动用在编程时**BRAKE**语句时发生。.

BRAKE语句不能用于外部程序中断。不遵守的话将产生故障或导致程序停止。

2.2.3.4 范例

在粘接应用期间，通过硬件执行的非路径保持的急停。如同使用程序停止粘接应用并且在使能（通过输入10）后重新在路径上定位粘接枪。

```
DEF STOP SP( )
; 中断程序
BRAKE F
ADHESIVE = FALSE
WAIT FOR $IN[10]
LIN $POS_RET
; 移动枪到轨迹前往的位置
ADHESIVE = TRUE
END
```

INTERRUPT，DECL，INTERRUPT

2.2.4 CCLOSE

2.2.4.1 摘要信息

先前用“**CHANNEL**”语句承认的通道的输入\输出可以用**CCLOSE**语句关闭。**CCLOSE** 删除所有等待被读入的数据。当取消选定和重新启动程序时，所有打开的通道将被关闭。

2.2.4.2 语句

```
CCLOSE (Handle, State)
```

自变量	类型	解释
Handle	INT	“handle”变量通过“**COPEN**”传送。
State	STATE_T	“STATE_T结构类型最初的组成变量类型是“CMD_STAT”。 “CMD_STAT”的值与“**CCLOSE**”有关： **CMD_OK**命令成功执行。 **CMD_ABORT**命令没有执行成功。

2.2.4.3 描述

被“**CHANNEL**”语句承认的输入输出通道能用“**CCLOSE**”语句关闭。

可能导致“CMD_ABORT”：

通道

--已经关闭；

--HANDLE无效；

--已经开始另一个程序。

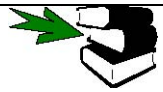
“HANDLE”不再使用在通道语句中已经顺利执行过一次的函数。变量的值无论如何不能改变。“CCLOSE”删除所有等待读入的数据。当取消和重启程序时，所有打开的通道绝对关闭。

通道语句的全部的方式和状态信息都可以在“CHANNEL”章节中找到。

2.2.4.4 例子

使用“HANDLE” 关闭通道。变量“SC_T”返回有关信息。.

CCLOSE(HANDLE,SC_T)



程序全部的例子都能在“CHANNEL”章中找到。

2.2.5 CHANNEL

2.2.5.1 摘要信息

“**CHANNEL**”语句被用于声明输入和输出的信号名。

2.2.5.2 语句

```
CHANNEL :Channel_Name:  Interface_Name  Structure_Variable
```

自变量	类型	解释
Channel_Name		任何符号名。
Interface_Name		预先确定的信号变量 **SER_1** 连续界面1 **SER_2** 连续界面2
Structure_Variable		系统决定结构变量。 赋值不能输出。.

2.2.5.3 描述

机器人控制器包含两类界面：

- 简单程序界面--**signals**
- 逻辑界面--**channels**.

所有界面用符号名表示地址。特殊的界面名(符号名)是预先用**CHANNEL**声明的信号的变量通道符合逻辑的组合。

预先确定的信号变量通道，

- **SER_1**和
- **SER_2**

为了连续界面，和

- **$CMD(例如：“RUN....”)**

为了解释程序命令。

访问通道的过程是一样的。为了可以在命令中能访问通道，通道名必须在“**CHANNEL**”中被公告。

在文件 “$CUSTOM.DAT”(目录“....\PROGRAM FILES\KRC\MADA\STEU”)“:SER_1”和“：SER_2”中，变量是预先确定的。.

通道可以通过“**COPEN**”语句打开。“**CREAD**”语句能用于读通道，“**CWRITE**” 语句用于写通道。

用“**CCLOSE**”语句关闭通道。

通道语句的规定和命令的信息是相同的。

状态信息是预先确定的结构类型“STATE_T”的变量的反馈。

“STATE_T”有下列的定义：

STRUC STATE_T CMD_STAT RET1, INT HITS, INT LENGTH

“CMD_STAT”是被下面预先确定的举例类型的数据：

ENUMCMD_STATCMD_OK，CMD_TIMEOUT，DATA_OK，DATA_BLK，DATA_END，
CMD_ABORT，CMD_REJ，CMD_PART，CMD_SYN，FMT_ERR

这个命令可使用“**CREAD**”和“**CWRITE**”语句产生的可能用到预先确定的列举类型：

ENUMMODUS_TSYNC，ASYNC，ABS，COND，SEQ

在单独的命令中解释规定和命令的意义。只有参数被使用时才被列出。

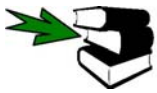
2.2.5.4例子

使用“**CHANNEL**”语句分配实际通道的通道名：
Channel name: **SER_2**

被分配到
physical channel: **SER_2**

在“$CUSTOM.DAT”文件中预先确定的
(地址....\PROGRAMFILES\KRC\MADA\STEU)

CHANNEL:SER_2:SER_2 $PSER_2



SIG NAL，COPEN，CCLOSE，CREAD，CWRITE

2.2.6 CIRC

2.2.6.1 摘要信息

圆形运动编程

2.2.6.2 语法

```
CIRC Auxiliary_Point, Target_Position ⟨,CA Circular_Angle
     ⟨Path_Approximation
```

自变量	类型	解释
Auxiliary_ Point	**POS E6POS, FRAME**	在循环中，几何表达式产生辅助点。 在这里可以使用笛卡儿坐标系。 笛卡儿系统的辅助点在系统变量$BASE中被定义。 对于辅助点方向角度和用S和T规定的角度状态总是可以忽略的。 如果结构中缺少辅助点，那么这些值不能改变现在的位置。
Target_ Position	**POS, E6POS FRAME**	几何表达式指定循环运动的目标点。 在这里可以使用笛卡儿坐标系。 参考系统中的笛卡儿坐标系的目标点由系统变量$BASE决定。 对于**POS**或**E6POS**类型的目标位置，角度位置指示**S**和**T**总是忽略的。 如果结构中缺少目标点部分，那么这些值不能改变点的当前的位置。
!		可以示教的参考点和目标点。如果动作迟顿，一个‘‘**！**’’将在辅助点和目标点的位置编程。

Circular_ **Angle**	REAL	算术式允许关键字 **CA**（圆形角）一起使圆弧延长或缩短。 测量的单位是度。 圆弧的角度没有限制。 超过360°的角度也可以用来编程 如果角度值是正值，机器人运动沿着起始点规定的方向运动。 如果角度值是负值，机器人沿着相反的方向运动。 在指定圆弧角时，程序的目标点通常不是真实的目标点。
Approximate_ Positioning	Keyword	这个选项允许你使用近似定位。 可能的输入： **C_DIS**(默认值) **C_ORI** **C_VEL**

编程TCP路径的速率和加速度:

	变量	数据类型	单位	功能
速度	$VEL.CP	REAL	m/s	移动速度(pathvelocity)
	$VEL.ORI1	REAL	°/s	旋转速度
	$VEL.ORI2	REAL	°/s	转动的速度
加速度	$ACC.CP	REAL	m/ s^2	路径加速度
	$ACC.ORI1	REAL	°/s^2	旋转加速度
	$ACC.ORI2	REAL	°/s^2	转动加速度

关于CIRC运动的刀具控制方向:

变量	作用
$ORI_TYPE=#CONSTANT	在轨迹运动期间，方向保持恒定， 目的点和起始点在编程时可以忽略。
$ORI_TYPE=#VAR	在轨迹运动期间，方向不断从最初的方向改变到目标点的方向。
$CIRC_TYPE=#BASE	有关基坐标系的角度控制($BASE)。
$CIRC_TYPE=#PATH	在圆弧运动中有关基于刀具的运动的角度控制。

近似定位时系统变量的初始定义:

变量	数据类型	单位	意义	命令中的关键字
$APO.CDIS	REAL	mm	转换长度标准	**C_DIS**
$APO.CORI	REAL	°	尺寸方向	**C_ORI**
$APO.CVEL	INT	%	速率标准	**C_VEL**

2.2.6.3描述

在**CIRC**运动的例子中，控制器计算从当前的点经过辅助点到中止点，因此明确的得出目标点或圆弧角。机器人经过中间点运动到中止点执行适当的插补周期间隔。

就像**LIN**运动，速度与加速度和系统变量$TOOL与$BASE也必须在圆弧运动中被编程。可是，速度和加速度不涉及除**TCP**外的各轴电机速度。

系统变量:

- **$VEL** 路径速率
- **$ACC** 路径加速度

可用到的定义。

方向控制

- 空间关系或路径关系的方向

  你可以从空间关系和路径关系定位控制中选择。在空间定位控制的例子中，位置由当前的系统变量($BASE)插补得出；在路径定位控制中，定位由刀具坐标系(基于刀具移动结构)插补得出。定位控制的类型由系统变量$CIRC_TYPE定义：

  -- $CIRC_TYPE=#BASE 空间定位控制

  -- $CIRC_TYPE=#PATH 路径定位控制.

- 变量和常量方向

  你可以选择变量和常量方向。如果你选择变量方向，在开始点和程序目标点间计算改变的方向。它可以执行空间和路径的输入。在常量空间方向的例子中，任何控制点的方向必须绝对与起始点的方向一致。在常量路径方向中，方向是常量。如果方向用于指定目标位置，那么这个方向常量是可以被忽略的。系统变量$ORI_TYPE用于定义方向：:

  --$ORI_TYPE=#VAR 变量定位

  --$ORI_TYPE=#CONSTANT 常量定位

角度状态

在**CIRC**运动的场合中，结束点的角度状态总是与开始点的相同。由于这个原因，目标位置类型为**POS**后**E6POS**的状态**S**和转变方向**T**总是可以忽略的。

为了保证次序确保同样的运动，各轴首先要清楚的定义。程序中的第一个运动指令首先必须使用PTP指定S和T。

近似定位

机器人不是必须到达辅助点。你可以使用下列运动块（**PTP，ＬＩＮ**或**CIRC**）定义到达目标点(所谓的近似定位)的距离。

近似定位在编程中有两步：

- 近似定位的定义范围由系统变量$APO 决定：

--$APO.CDIS：

转变距离标准(激活C_DIS)：

近似定位的轮廓是从目标点到起始点的指定距离(单位[mm])。

--$APO.CORI：

方向距离(激活C_ORI)：当角度(工具的纵轴的旋转或转动)低于距离目标点规定的角度差值时，TCP离开程序块指定的轮廓。

--$APO.CVEL

速率标准(激活C_VEL)：

当$APO.CVEL到达$VEL.CP定义的速度百分比时，近似定位轮廓开始执行。最大近似距离为编程尺寸的一半。

- 运动编程指令中的目标点和近似定位模式：

--CIRC—CIRC或CIRC—LIN 近似定位

在**CIRC—CIRC**和**CIRC—LIN**近似定位的情况下，控制器不能计算出对称的近似定位轮廓。近似定位路径由两段抛物线组成，它们之间由切线过渡，在单独的程序段间也是这样的。定义近似定位的开始和结束，使用关键字C_DIS, C_ORI或 C_VEL三者之一编程。

--CIRC—PTP 近似定位

近似定位的前提是，机器人各轴在**CIRC**段的转动量不能超过180°，位置**S**不能改变。近似定位的开始由变量$APO.CDIS，$APO.CORI和$APO.CVEL之一决定，终点由$APO.CPTP决定。关键字**_DIS**,**C_ORI**和**C_VEL**得用于**CIRC**编程指令中。

为了近似定位，电脑必须在运行前被激活。 如果没有将显示“Approximation not possible”信息。在CIRC—PTP的近似定位情况下，无论如何在运行前设为1 。

- **程序在近似定位段中提前停止运行(用CONTINUE补救)。**
- **速度和加速度越大，动态偏离（跟踪误差）也就越大 。**
- **改变加速度比改变速度的影响要小。**
- **在近似定位中，插补总是以空间基本理论为基础来执行的。**
- **近似定位的最初的方向总是以近似定位点到达时的方向为基础的。**
- **如果两个路径方向的CIRC块被执行，那么近似定位范围中的再定位仍然是空间关系的**

2.2.6.4 范例

圆弧运动的辅助点的目标坐标系。

CIRC ；

穿过辅助的目标坐标的圆弧运动；运动的目标点由-35 ^{0}圆弧角决定

CIRC ! ,CA -35
程序圆弧运动的辅助点的目标坐标系。

CIRC {X 5,Y 0, Z 9.2},{X 12.3,Y 0,Z -5.3,A 9.2,B -5,C 20}
程序圆弧运动的辅助点的目标坐标系；激活近似定位。

CIRC {Z 500,X 123.6},POINT1,CA +260 C_ORI
常量方向的路径控制。

$ORI_TYPE=#CONSTANT
$CIRC_TYPE=#PATH
CIRC AUX_POINT,{X 34, Y 11, Z 0}
用CIRC—PTP从第二点到第三点近似定位。近似定位在第二点前30mm开始。.

```
$APO.CDIS=30
$APO.CPTP=20
PTP POINT1
CIRC AUX_POINT，POINT2，CA　ANGLE
C_DISPTP　POINT3
```

CIRC_REL，PTP，LIN

2.2.7 CIRC_REL

2.2.7.1 摘要信息

相对坐标的圆弧运动

2.2.7.2 语法

```
CIRC_REL   Auxiliary_Point,Target_Position
           ⟨,CA Circular_Angle
           ⟨Approximate_Positioning
```

自变量	类型	解释
Target_ Position	**POS** **E6POS** **FRAME**	用几何表达式指定圆弧运动的目标位置。 在这里也可以用笛卡儿坐标系。 这可以认为是**CIRC**运动相对于当前的位置运动。 距离转换在基本坐标系$BASE的各轴下执行。 如果目标位置包含不明确的结构组织，那么这些值设为0，例如：绝对值保持不变。 预先确定的变量$ROTSYS定义编程方向的效果。 对于**POS**或**E6POS**类型的目标位置，角度位置指示**S**和**T**总是忽略的。

Auxiliary_ Point	**POS** **E6POS** **FRAME**	几何表达式产生圆弧路径上的辅助点。 辅助点在相对坐标系和笛卡儿坐标系中指定。 距离转换在基本坐标系$BASE 的各轴向执行。 如果辅助点包含未定义的结构分量，这些值设为0，例如：绝对值保持不变。 预先确定的变量$ROTSYS定义编程方向的效果。 角度值和角度状态用S和T定义，在表示辅助点时总是被忽略。
Circular_ Angle	REAL	算术式允许关键字 **CA**（圆的角度）一起使圆弧延长或缩短。 测量的单位是度。 圆弧的角度没有限制。 超过360°的角度也可以用来编程 如果角度值是正值，机器人运动沿着起始点，辅助点和目标点规定的方向运动。 如果角度值是负值，机器人沿着相反的方向运动。 在指定圆弧角时，程序的目标点通常不是真实的目地点。这是通过角度指定的。
Approximate_Positioning	Keyword	这个选项允许你使用近似定位。 可能的输入： **C_DIS**(缺省值) **C_ORI** **C_VEL**

	变量	数据类型	单位	功能
速度	$VEL.CP	REAL	m/s	移动速度(路径速度)
	$VEL.ORI1	REAL	/s	旋转速度
	$VEL.ORI2	REAL	/s	转动速度
加速度	$ACC.CP	REAL	m/s²	路径加速度
	$ACC.ORI1	REAL	/s²	旋转加速度
	$ACC.ORI2	REAL	/s²	转动加速度

在CIRC运动中刀具角度控制：

变量	作用
$ORI_TYPE=#CONSTANT	在轨迹运动期间，方向保持恒定， 目的点和起始点在编程时可以忽略。
$ORI_TYPE=#VAR	在轨迹运动期间，方向不断从最初的方向改变到目标点的方向。
$CIRC_TYPE=#BASE	有关基坐标系的角度控制($BASE).
$CIRC_TYPE=#PATH	在圆弧运动中有关基于刀具的运动的角度控制

定义辅助点的起始点的系统变量

变量	数据类型	电位	意义	命令中的关键字
$APO.CDIS	REAL	mm	距离转换标准	**C_DIS**
$APO.CORI	REAL	°	方向角度	**C_ORI**
$APO.CVEL	INT	%	速率标准	**C_VEL**

2.2.7.3描述

相对**CIRC**命令与绝对**CIRC**命令在工作执行时是相同的。目标坐标系仅仅定义相对当前的位置代替辅助位置绝对值的轴坐标系。所有包含绝对**CIRC**指令的应用在这儿。

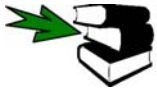
2.2.7.4 例子

在目标坐标系辅助编程的圆弧运动：运动的目标点的范围是圆弧角500°；近似定位被激活。

CIRC_REL {X 100,Y, 3.2,Z -20},{Y 50},CA 500 C_VEL

LIN—CIRC从第一点到第二点的近似定位和CIRC——CIRC从第二点到第三点的近似定位。当速率减少到6m/s(50%of1.2m/s)近似定位从第一点开始。近似定位第二点从第二点前20mm开始。

```
$VEL.CP=1.2
$APO.CVEL=50
$APO.CDIS=20
LIN POINT1 C_VEL
CIRC_REL AUX_POINT,POINT2_REL C_DIS
CIRC AUX_POINT,POINT3
```



CIRC，PTP_REL，LIN_REL，CONTINUE

2.2.8 CONFIRM

2.2.8.1 摘要信息

承认确认信息.

2.2.8.2 语法

```
CONFIRM Management_Number
```

自变量	类型	解释
Management_ Number	**INT**	算术表达式指定被承认的信息数。 指定管理数0意味着所有信息能被承认。

2.2.8.3 描述

在程序控制器承认使用语句**CONFIRM**承认的信息。在信息被成功的承认后，承认的信息不再提出。

2.2.8.4 例子

承认被承认的信息数是27.

CONFIRM 27

承认被承认的信息数为**M_NUMBER**.。

CONFIRM M_NUMBER

承认被承认的信息数为**M_NUMBER** +5。.

CONFIRM M_NUMBER+5

承认所有现有的被承认的信息。

CONFIRM 0

在停止信号(例如：急停)被取消后，确认信息总是被显示。在你使用它前，它必须被承认。在子程序发现和承认这个信息前，那个信息的正确的操作是在它被取消前中止。在机器人程序没有开始时，如果承认的信息被激活，子程序必须在Submit文件中定位。

```
DEF AUTO_QUIT( )
INT M
DECL STOPMESS MESSAGE                    ; 预先确定停止信息的结构类型
IF $STOPMESS AND $EXT THEN               ; 检查停止的信息和操作命令
   M=MBX_REC($STOPMB_ID,MLD)             ; 在MESSAGE中读当前的值
   IF M==0 THEN                          ; 检查被承认的信息
        IF ((MESSAGE.GRO==2) AND (MESSAGE.STATE==1)) THEN
             CONFIRM MLD.CONFNO          ; 承认这个消息
        ENDIF
   ENDIF
ENDIF
END
```

2.2.9CONTINUE

2.2.9.1摘要信息

阻止先前运行的停止。.

2.2.9.2语法

CONTINUE

2.2.9.3描述

你能用系统变量$ADVANCE定义先前控制器执行的中断如何动作。在指令关于外围(例如输入/输出指令)的情况下，计算机先前的运行总是被停止。如果你不想这样的事情发生，**CONTINUE**语句必须在相应的指令前被编程。

CONTINUE语句总是用于下列指令行，甚至是空白行。

2.2.9.4例子

使用$OUT防止运行的停止：

```
CONTINUE
$OUT[1]=TRUE
CONTINUE
$OUT[2]=FALSE
```

ANOUT，HALT，WAIT，PULSE

2.2.10 COPEN

2.2.10.1 摘要信息

已经被“**CHANNEL**”语句承认的输入输出通道能用 “**COPEN**”语句打开。

2.2.10.2 语句

COPEN(Channel_Name, *Handle*)

自变量	类型	解释
Channel_Name		在“**CHANNEL**”语句承认的通道名
Handle	INT	用户定义的变量

2.2.10.3 描述

被**CHANNEL**语句承认的输入输出通道能用 “**COPEN**”语句打开。

“Handle”变量确定下列存取有关的通道。如果系统拒绝打开通道，则返回0。

预先确定的变量“$CMD”是命令执行一般的打开可用到的。

2.2.10.4 例子

开放名为：“SER_2”的通道和“HANDLE”。

COPEN(:SER_2,HANDLE)

程序全部的例子都可以在“CHANNEL”章中找到。

2.2.11 CREAD

2.2.11.1 摘要信息

从通道中读取数据。

应用的例子： 在**KRC1**和外围数据(PC,智能传感器...) 数据交换（读语句）。

“**CREAD**”语句用于从打开的通道中读取数据。在这有两个区别：：

- **主动读入**
  程序请求通过通道进行输入。通道驱动器提出输入请求并将接受到的数据作为结果返回到CREAD语句 。
- **被动读入**
  预先确定的变量(INT$DATA_SER1orINT$DATA_SER2),未被请求的数据到达后通过增加通道驱动器，每个通道都是这样的。当执行热启动或当打开/关闭通道时变量初始化为0。在系统等待读入请求的反馈信号方面也是有差别的：绝对的或有条件的。绝对的意思是系统等待直到通道给出请求的数据。在有条件的情况下，系统检查数据是否有效。

2.2.11.2 语句

```
CREAD (Handle, State, Mode, Timeout, Offset, Format,
       Var1,..., VarN)
```

自变量	类型	解释
Handle	INT	由“**COPEN**”转移变量。注意：变量“$CMD”将被拒绝

State	STATE_T	“CMD_STAT”是“STATE_T”结构类型的“State”变量的列举类型第一个元素。 “CMD_STAT”能有与“**CREAD**”有关的下列值： **CMD_OK** 命令顺利的执行； 在#COND模式中没有可用的数据 **CMD_TIMEOUT**在“ABS”模式指定的时间限制被超出 读入失败。 **DATA_OK** 数据块是被通道承认的。所有的数据被分配与格式描述一致的变量。不论所有的变量是否需要都被描述。（注意也在“HITS”变量下） **DATA_BLK**数据被读入但更多的数据能利用“SEQ”模式准备。 **DATA_END** 数据被读入。数据块被完全的读入。 **CMD_ABORT**读入失败，从通道返回错误信息或在读出数 据时有致命的错误。如果格式规范和变脸类型不一致，读写失败。 **FMT_ERR** 错误的格式规范或不对应的变量。 **CREAD**由其他的状态变量组成。 **HITS** 数字正确地读入格式。 **LENGTH** “%s”或“%r”格式的在出现第一次格式中长度。 所有下列“%s”或“%r”格式的长度没有被指定。如果需要，将使用几个“**CREAD**”语句
Mode	MODUS_T	“MODUS_T”是列举类型，它有与下列“**CREAD**”有关的值： **ABS** 激活读入通道。函数等待到通道产生一个可用的数据块或等待到“Timeout”失败。 **COND** 未被请求的通道读入。 **SEQ** 从以前被请求使用“**ABS**”或“**COND**”或作为结果返回“**CWRITE**”的字节补偿读入完成了的数据块

Timeout	REAL	参数“**TIMOUT**”可被用于指定时间，在等待数据块失败之后。 **Timeout**的值0.0允许无止境的等待。. 一个值大于60或是为负值，则数值是无效的。与系统有关的错误在等待时间内是是固有的。
Offset	INT	变量“**Offset**”用于指定被承认的数据中的字节数，在系统开始读入之前。 如果开始就读入，偏移量t必须设为0（原点）。 “**CREAD**”语句不能分配所有被程序承认的数据变量，偏移量指定已经分配的字符数。
Format	CHAR[ ]	变量“**Format**”的类型“**CHAR[ ]**”(文字讯息)包含产生了的文本格式。 变量结构主要符合“**C**”语言 “**FPRINTF**”功能的格式。
Var		变量符合“**Format**”。

2.2.11.3 说明

“**CREAD**”语句用于读取打开通道的数据。两个例子列在下面：

主动读写
程序经由通道请求输入。通道驱动输入请求和返回经过“**CREAD**”语句承认的数据作为结果。
被动读写
写入数据到通道没有被接受，期望数据被收集。预先确定的变量产生任何可用的通道“SER_1”和“SER_2”。
INT $DATA_SER1 或
INT $DATA_SER2,
在未被请求的数据到达后，这些数据通过通道驱动增加。当系统启动或通道打开或关闭时，变量被初始化为0（原点）。
这与系统等待读请求的反馈信号不同。“**CREAD**”语句可以是有条件的等待或无条件的等待。
绝对的意义是系统等待到通道供给被请求得数据。
在有条件的例子中，系统检查数据是否是可用到的。
通过使用反馈信号，它能决定读语句是否成功。相应的程序在“**MODE**”中定义。

如果操作不是来于“COPEN”语句程序是被传送的“CREAD”语句或通道已经又再关闭，承认信息“INVALIDHANDLE”被显示。

其它方式的规定或非初始化的变量导致一个被变量“**STATUS**”发现错误 。如果读“**ABS**”“**COND**”成功，那么数据块的数据是先前被承认的，就像他们完全的读出。

文本是按照格式规定的程序段返回的。值决定被分配的适当的变量，系统检查值之否在任何区域有效。“变量FOMAT”转换支持由“Kernighan/Ritchie”指定的格式规定（C语言1978），**O，P，N，U**和【**list**】出外。
字符长度规范“**H**”和“**L**”不能使用。

只有9个格式参数可用于指定“**CREAD**”语句。如果几个变量是格式化可用到的，则必须在“＃SEQ”模式下读入。

系统不能区别大小写字符。在发生一个错误后读入失败(不符合格式或无效的值)。

转换字符“**R**”，当读入任何一个字节指定的字符长度次序 (与写入类似，例如“%2.5r”)或这所有字节在结束信息之前，他也同样传入。

不像其它格式，单独字节的读入必须被“%1r”明确规定。.

没有用“%c格式”指定宽度，这样的格式是不合格的。字节能被分配到**INT，REAL，CHAR，BOOL，ENUM**数据类型的变量或这种类型的一维数组。

假定整数数据类型出现在“little endian”格式中，而且是有正负之分的。

Real的数据类型在**IEEE754**标准格式中用32位表示。

- Bit 31 符号,
- Bit 30—23 指数
- Bit 22—0 尾数

格式变量	%d %I %x	%f %e %g	%c	%s	%1r	(3) %1.h WDHir	%2r	(3) %2.h WDHir	%4r	(3) %4.h WDHir	(4)%r	(3)%. hWDHir
(Signal) INT	X		X		X		X		X		X	
INT array						X		X		X	X	X
REAL	X	X							X		X	
REAL array										X	X	X
(Signal) BOOL(1)	X		X		X		X		X		X	
BOOL array						X		X		X	X	X
ENUM(2)	X		X		X		X		X		X	
ENUM array						X		X		X	X	X
CHAR	X		X		X						X	
CHAR array				X		X					X	X

备注：

任何不等于0(zero)的值可以被**TURN**转换。

系统控制值是否被**ENUM**值允许。如果不是，读失败。**ENUM**开始为1。

如果没有足够的格式数据，(例如%2.5r，但只有7位),没有数据从格式读出，**CREAD**语句失败。被忽略的数据依然被**READ**读取。.

变量的读就像变量装入一些字节一样。直到读准备好后重新启动。如果数组足够大但是数字用到的字节不是要素排列的倍数，那么多余的字节列在格式后或在下一个**CREAD**语句的左面读取。

通过“**CREAD**”调用下列更多的读取。字节数“%s”或“%r的格式指定首先读取实际上被状态变量返回的格式行。

所有其他的字长度不是确定的。因此用“%s”或“%r”格式只是一次“**CREAD**”调用格式行和重复。
如果“%s”或“%r”格式不在已经成功读取的格式之中(看“HITS”的变量状态)，“**LENGTH”**的值在语句中不变。

下面列出了所有语句的应用：

--语句总是等待语句完全的执行完成和它返回程序。这是绝对**CREAD**语句读文本通道的一个显著特征。

--不注意这些，这些程序会被中断程序打断。任何访问通道的努力都会被中断子程序阻断。.

语句的全部解释和通道指示命令信息都可以在“**CHANNEL**”中找到。观看2.2.5.部分。

程序完整的范例可以在“CHANNEL”章找到。

2.2.12 CWRITE

2.2.12.1 摘要信息

“**CWRITE**”语句能够使文字写入打开的通道，或者命令被写入命令通道。

应用实例： 数据 (写语句)在**KRC**1和装置间(**PC**,智能传感器...). 转换。

2.2.12.2 语句

```
CWRITE (Handle, State, Mode, Format, Var1, ..., VarN)
```

2.2.12.3 说明

自变量	类型	解释
Handle	INT	“Handle”变量通过 “COPEN”或预先去定的变量“$CMD”传送。
State	STATE_T	“CMD_STAT”是举例类型，它是组成结构类型“STATE_T”状态变量的第一步。 “CMD_STAT”能有下列与**CWRITE**对应的值： **CMD_OK** 命令成功执行； ;**DATA_OK** 命令被成功的执行。数据作为回复准备； **CMD_ABORT** 因为“HANDLE”是无效的，命令没有被执行； **CMD_REJ** **BCC** 错误 **CMD_SYN**在 命令中语句错误。命令的语句错误，命令因此不能被执行。 **FMT_ERRI** 错误的格式规范或不相应的变量。状况变量由 “**CWRITE**”组成另一个重要部分： **HITS** 正确的数字写入格式。.
Mode	MODUS_T	变量类型“MODUS_T”(结构类型)定义些多少通道写入。它有下面的值： **SYNC** 语句直到数据已经发送还没有执行。 **ASYNC** 语句没有执行直到通道驱动已经证实数据已经被承认。

Format	CHAR[ ]	变量“**Format**”包含产生的文本格式。 变量的结构主要符合“C”语言 “FPRINTF” 功能的格式。
Var		“**Format**”相应的变量.

2.2.12.3 说明

语句“**CWRITE**”使文本能够写入打开的通道或命令能够写入命令通道。

“**Mode**”的值写入命令通道的无关。如果“**Mode**”是非初始化的变量，那么语句失败而且在变量“**Status**”中放置错误标记。

如果“**Mode**”有一个除**SYNC**或**ASYNC**以外的值，那么数据在**SYNC**模式下写入通道。

变量格式的转换规定有以下构造：

%FWGU

下面定义的应用：

- **F** 格式字符**+**，**－**，**#**，**etc**，(可选)。
- **W** 宽度，指定输出的最小字符数（可选）
- **G** 精度，它的意义是由字符转化决定’**.**’或’*****’或’整数’可以被使用(可选).
- **U** 允许转换的字符：**c**，**d**，**e**，**f**，**g**，**I**，**s**，**x**和**%**。系统不能区别大小写。
  给上面增加转换字符(相应于”“C”中的“FPRINTF)，字符“**r**”是也可能用到的。

格式变量“%r”使它的变量值不使用ASCII但使用二进制符号。由于“%r”，系统不能检查变量或排列元素是否经过初始化。

依靠输入的宽度(“%2r”)，你可以指定多少字节的值会被延长或压缩。REAL的值出外。

当数值被压缩，高位字节被忽略；值通过在结尾增加原点位来延长 (little endian format) 。

如果宽度没有指定，固有的显示输出：**INTEGER**，**REAL**和**ENUM** 4个字节，**BOOL**和**CHAR** 1个字节。

精度仅仅能被数组指定和循环数解释。与数组元素的相应的数能在开始时就被输出。如果循环的数字比数组排列的更多，那么没有数组元素写入而且输出失败。

“*”不能指定精度。如果值的精度被省略，数组全部写入。

“**Var1”,...,“VarN**”可能不是结构类型或数组的结构类型的变量(包括像 “**POS**”的结构)。类型在下表的运行时间内检查一致性。

如果类型是不兼容的或者系统遇到的第一个值是未初始化的，那么转换失败（除了“%r”的场合）。不输出错误信息。

错误的格式可以从**HITS**的值进行判断。（看下面）

布尔值作为0和1输出，**ENUM**作为数字值

结构变量	**%d %I %x**	**%f %e %g**	**%c**	**%s**	**%1r**	**%1.** <WDH>	**%2r**	**%2.** <WDH>	**%4 r**	**%4.** <WDH> **r**	**%r**	**%.** <WDH> **r**
(Signal) INT	X	X			X		X		X		X	
INT array						X		X		X	X	X
REAL		X							X		X	
REAL array										X	X	X
(Signal) BOOL	X				X		X		X		X	
BOOL array						X		X		X	X	X
ENUM	X				X		X		X		X	
ENUM array						X		X		X	X	X
CHAR	X		X		X						X	
CHAR array				X		X					X	X

“CWRITE”语句被用程序控制或机器人移动控制触发先前停止的运行。

特别的时间输入输出操作在程序执行时起相当大的作用。

语句中的申请：语句总是等待自己彻底的执行完成而返回程序。这是绝对值**CWRITE**语句显著的特征。

如果不注意这些，语句会被打断程序中断。任何访问通道的企图都会被其他的子程序再一次打断。

命令能返回或通过命令通道拒绝几个反馈信号。

通道结构的命令信息和状态定义都可以在**“CHANNEL”**章找到。

2.2.12.4 范例

下面给出了几个“**CWRITE**”语句的范例：

数值的十进制和十六进制的转换

“I”十进制和十六进制ASCII格式的转换：

```
INT I
I=123
%D,I ;发送3位数据: 123
%X,I ;发送2位数据: 7B
```

“**I**”的二进制计数法转换

```
INT I
I=123
%R,I ;发送1位数据:{ (Hex. 7B)
I=123456 %R,I ； 传送3位数据：@â  (Hex.40E201)
```

首先写入数组名R[ ]的5个值

首先写入数组名为R[ ]”的5个值，因为数组还没有被初始化的值而产生任意的值。

```
REAL R[10]
%.5R,R[ ] ； 发送20位二进制的数据
```

输出所有数组元素的值

下面的语句用于输出所有数组元素的值：

```
REAL R[10]
%R,R[ ]
```

输出特定的数组元素

输出数组元素“S”，结尾用第一个为初始化的元素：：

```
CHAR S[100]
%S,S[ ]
```

首先写入50个数组元素S

首先写入数组“**S**”的50个数，忽视初始化信息。

```
CHAR S[100]
%.50R,S[ ]
```

ENUM变量的固有转换值在ASCII中

“ENUM”变量的固有转换值在ASCII 中。相应的数字被输出。

DECL ENUM_TYP E

%D,E

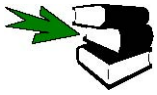
写入两个真实值的名字(定义字符长度)

REAL W1,W2

W1=3.97

W2=-27.3 CWRITE(..,..,..,..,"Value1=%+#07.3F Value2=+#06.2F",W1.W2)

；发送数据: Value1=+03.970 Value2=-27.30



程序全部的范例都可以在"CHANNEL"章中找到。

2.2.13 DECL

2.2.13.1 摘要信息

声明程序和数据列表中的变量和常量。

2.2.13.2 语句

声明程序中的变量:

```
〈DECL Data_Type Variable_Name1 〈, ...,
    Variable_NameN
```

声明程序中的数组

```
〈DECL Data_Type Array_Name1 [Size1 〈, ...,
    Size3 ] 〈, ..., Array_NameN [SizeN1 〈,...,
    SizeN3 ]
```

声明数据列表中的变量:

```
〈DECL〈GLOBAL Data_Type Variable_Name1 〈, ...,
    Variable_NameN
```

或者同时分配值：

```
〈DECL〈GLOBAL Data_Type Variable_Name = Value
```

声明数据列表中的数组:

```
〈DECL〈GLOBAL Data_Type Array_Name [Size1 〈,...,
    Size3 ] 〈, ..., Array_NameN [SizeN1 〈,...,
    SizeN3 ]
```

声明和初始化数据列表中的常量：

```
DECL〈GLOBAL CONST Data_Type Constant_Name = Value
```

声明和初始化数据列表中的数组：

```
DECL〈GLOBAL CONST Data_Type Array_Name [Size1 〈,...,
    Size3 ]

Array_Name[1 〈, 1, 1 ] = Value1
...
Array_Name[Size1 〈, Size2, Size3 ] = ValueN
```

自变量	类型	解释
Data_Type		简单的数据类型： **INT**，**REAL**，**CHAR**，**BOOL** 这些在系统文件能被任何结构和列举的类型，例如： **FRAME**，**POS**，**E6POS**，**AXIS**，**E6AXIS** 能被自己定义的数据类型： 结构类型(**STRUC**)或列举类型(**ENUM**)
Variable_Name, Array_Name, Constant_Name, Object_Name		对象的名字被公告。
Size	INT	没有负号的，正整数的常量定义数组的大小。 常量的数量产生数组的范围。 数组范围最大为3。
Value		被公告的常数数据类型
Array_Index	INT	常量能在1到Size的范围内。把它看作数组元素开始来分配值。

全部的变量和常量能仅仅在数据列表中被公告。

顺序使用关键字GLOBAL，在文件“Progress.ini”中的全球选项，在INIT目录必须设为, TRUE：

GLOBAL_KEY=TRUE

2.2.13.3 说明

在程序中使用的变量必须是在声明中公告的名字和数据类型。简单的、复杂的和随意的、可定义的数据类型是可用的。

声明首先由关键字**DECL**开始，跟随数据类型和变量列表和数组已有的数据类型。当声明预先确定类型的变量和数组，关键字**DECL**可被省略。除简单数据类型**INT**，**REAL**，**CHAR**和**BOOL**外，数据类型**POS**，**EPOS**，**FRAME**，**AXIS**，**EAXIS etc**中都被预先确定。声明能被数据类型为**POS**的变量省略，因为这个数据类型是标准的数据类型而且是默认的分配。关键字**DECL**可能被用户定义结构声明或列举类型忽略。

数组声明

就像变量，任何数据类能用于数组。增加数据类型和数组名，数组大小和尺寸必须也被数组公告。尺寸由指定数组大小的数决定。它最大为3。数组的大小出现在方括号中的数组名逗号分隔之后。任何一个数组的大小是无符号的整数。它必须等于或大于1。

如果数组被作为子程序或函数的参数形式传送，就像变量一样它必须被在这个子程序或函数的定义中公告。数组的大小必须在这个声明中被忽略但是方括号和逗号决定数组的尺寸。当调用子程序和函数时，数组的大小时由分配当前的参数转换决定。.

声明变量与默认的设置

变量能在数据列表中公告和像默认相同的时间一样分配最初的值。声明语句包含的默认设置不能在程序和函数的声明部分使用。

在简单数据变量的情况下，最初的值作为简单的常量指定。通过结构变量，最初的值是一个集合。

声明语句在开始时给变量分配默认的设置，就像简单的声明由关键字**DECL**分配变量名和数据类型以默认设置。

“=”符号和最初的值以常量的形式跟随变量名。当声明默认的设置，你不能在声明语句中列出多个变量。一个单独的声明语句是任何一个变量被分配默认设置的所必须的。当分配默认设置时，关键字**DECL**可被忽略。

“＝”符号右面的常量数据类型必须与左面指定的数据类型兼容但是并不必须完全一致。如果数据类型兼容，系统自动作为常量匹配它们。

声明数组与默认设置

声明语句包含的默认设置不能在程序和功能的声明部分使用。也不能在数据列表声明和初始化中单个排列。

当在数组中声明默认设置，独立的语句必须写入任何一个数组元素。

数组默认设置的声明最少包含两块：

- 第一块包含由DECL.声明的标准数组声明。
- 第二块包含数组元素的说明并通过“＝”号和数组元素的初始值跟随。
- 在默认设置中更多的这个类型块被分配可跟随的其他数组元素。

当分配比数组元素更多的默认设置，元素必须在升序排列的数组索引中被指定。

“＝”符号右面的常量数据类型必须与左面指定的数据类型兼容但是并不必须完全一致。如果数据类型兼容，系统自动作为常量匹配它们。

作为默认分配字符行

如果你想默认分配相同的字符串作为数组元素的类型字符行，你不是必须单独的分配它的每一个数组元素。数组索引的有指针可被省略，而且常量字符串作为默认设置分配到一个完整的行。

变量数据类型“Freely” 声明可定义为结构类型或列举类型。
如果之后数据类型名不是预先确定的系统数据类型，关键字DECL必须在这里被编程。数据类型定义STRUC和ENUM必须总是在DECL声明的这个类型变量之后。

2.2.13.4范例:

声明没有初始化。

```
DECL POS P1
；关键字DECL可被省略
INT A1,A2
REAL VEL[7],ACC[7],B
DECL S_PAR_TYPE S_PAR[3]
```

由默认设置声明数组(只在数据列表中).

```
INT A[7] ；数组的7个整数值
A[1]=27  ；第一个数组元素被分配为27
A[2]=313
A[6]=11
CHAR TEXT1[80]
TEXT1[ ]=”Message Text”
CHAR TEXT2[2,80]
TEXT2[1,]=”First Message Text”
TEXT2[2,]=”Second Message Text”
```

声明变量的初始化(只在数据列表中).

```
FRAME F1={X 123.4, Y -56.7, Z 89.56}
```

STRUC，ENUM，DEFDAT

2.2.14 DEF...END

2.2.14.1 摘要信息

程序和子程序的声明。

2.2.14.2 语句

```
〈GLOBAL  DEF Program_Name(〈Parameter_List )
〈DECL Data_Type Parameter_Name1
...
DECL Data_Type Parameter_NameZ
〈further declarations
〈Statements
END
```

```
〈GLOBAL  DEF Program_Name(〈Parameter_List )
〈DECL Data_Type Parameter_Name1
...
DECL Data_Type Parameter_NameZ
〈further declarations
〈Statements
END
```

自变量	类型	解释
Program_Name		程序名进入这儿被定义。它是一个目标名而且在全球函数情况下的它可能不长于24个字符 长度受控制器的指令系统限制。

Parameter_ List		参数列表包含下列说明： 参数名 输出数组类型参数的情况下（输入参数不能被排列），数组尺寸除数组名字外使用下列符号。 **[ ]**一维空间的数组 **[,]**二维空间的数组 **[,,]**三维空间的数组 参数的各自转换格式： **:IN**输入参数(由值唤醒) **:OUT**输出参数(由参数唤醒)(默认值)

Declaration	只有在声明部分可能出现声明。在这里不能使用程序语句。与在第一个语句定义的声明部分和语句部分接近
Statements	只有在语句部分可能出现语句。在这里不能使用声明。你能使**RETURN**语句推出当前的子程序。没有RETURN语句，**END**语句在语句的最后被执行。

2.2.14.3 说明

通过默认，**SRC**文件中的第一个程序与**SRC**文有同样的名字并认为全局有效，甚至没有关键字**GLOBAL**。

当程序被唤醒，有两种参数传送：传送输入参数和传送输出参数。

输出参数(关键字IN)
变量的值被传送到这儿。直接的参数传送工作就像在子程序中分配默认设置到变量。通常传送的值可以是常量，变量函数，简单或复杂的表达式。

一个值在**IN**参数的情况下不能返回调入的指令(由值唤醒)。它仅仅由于赋一个值到子程序。

I如果当前的数据类型和外部的IN参数不同但兼容，系统自动转化被传送的值的类型。数组不能像输入参数一样被传送 **(IN)**。

输出参数(关键字OUT)
变量名被传送到这儿（有参数唤醒）。变量在调用子程序时必须有一个值。这个值能被子程序用于调用。

参数类型**OUT**能在子程序调用时被分配一个值。由于这个原因当前的数据类型和外部参数必须在传送模式**OUT**中完全一致。

作为一个参数输出传送是默认的设置，例如：**OUT**不需要指定。

END语句
END语句总是的全球或本地子程序的最后程序段。子程序的最后程序段执行任何**RETURN**语句，没有时执行**END**语句。

2.2.14.4 范例:

没有外部参数的程序的声明：

```
DEF PROG( )
...
END
```

声明子程序的外部电流和电压。适当的默认设置，他们输出参数。

```
DEF WELD(电流，电压)
...
END
```

声明子程序的外部电流和电压作为输入参数和RESULT作为输出参数。

```
DEF WELD(电流:IN,电压:IN,结果:OUT)
...
END
```

在CALCULATE子程序中，一些变量经过自动操作。在子程序调用后，在主程序中的A和B跟随值：A＝11；B＝2。

```
DEF PROG( )
INT A,B
A=1
B=2
CALCULATE(A,B)
...
END

DEF CALCULATE(X1:OUT,X2:IN)
INT X1,X2
X1=X1+10
X2=X2+10
END
```

END，DEFFCT，RETURN

2.2.15 DEFDAT...ENDDAT

2.2.15.1 摘要信息

数据列表声明。

2.2.15.2 语句

```
DEFDAT Data_List_Name ⟨PUBLIC
⟨Declarations
ENDDAT
```

自变量	类型	解释
Data_List_ Name		数据列表的名字进入这儿被定义。它是最长为24个字符的对象名。长度通过控制器的管理系统限制。 如果数据列表的名字与指令相同，数据列表分配这个指令作为数据列表声明的结果，同样在指令中相同的名字也适用。 指令和数据列表从指令包中分配。
PUBLIC		通过增加这个关键字，其它指令和数据列表也能访问这个数据列表，而且在这儿被分配的变量，etc..，可用于其它的指令包。它们必须使用关键字**GLOBAL**定义。
Declaration		子程序和函数指令的外部声明在指令中使用。 为了输入变量引入声明。 声明变量 声明信号和通道名。 声明结构和列举类型 在数据列表中声明的变量声明可以包含默认设置。

2.2.15.3 说明

除预先确定的数据列表外，你自己可以定义更多的数据列表。程序列表被特定的程序和高级的声明所使用。变量值可以存储在在被声明的数据列表中。数据列表作为独立的**ENDDAT**语句总是在任何数据列表的最后程序段。

没有语句出现在数据列表中，除了变量和常量的初始化。

2.2.15.4 范例

声明数据列表。

```
DEFDAT WELD
...
ENDDAT
```

为了符合全局声明数据列表。

```
DEFDAT CENDAT PUBLIC
...
ENDDAT
```

指令包PROG_1由指令和被分配的数据列表PROG_1组成。

如果它在数据列表中被声明和被初始化的，那么在主程序作中被省略。如果在主程序的变量OTTO被分配了新的值，那么它也进入数据列表并被存储在这儿。.

“new”值在控制器转换完成而且再次返回后被使用。这是在线修正和和其他程序修改的要素。如果你想总是用相同的值开始主程序，那么想得到的值必须在主程序中被作为默认分配适当的变量。

```
DEFDAT PROG_1
INT OTTO = 0
ENDDAT
```

DEF PROG_1( ) HALT ;OTTO现在是0

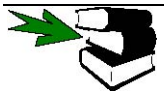
OTTO=25 HALT ;数据列表现在包含INTOTTO=25

END

全局的数据列表：变量OTTO在PROG_1和PROG_2中被验证的。它可能允许外面的主程序访问在数据列表中定义的变量。数据列表必须作为PUBLIC定义而且变量必须作为GLOBAL公告。DEFDAT PROG_1 PUBLIC GLOBAL INT OTTO = 0 ENDDAT

```
DEF PROG_1( )
HALT
OTTO = 25
END

DEF PROG_2( )
OTTO=27
END
```



IMPORT

2.2.16 DEFFCT...ENDFCT

2.2.16.1 摘要信息

声明函数。

2.2.16.2 语句

```
⟨GLOBAL  DEFFCT Data_Type Function_Name (⟨Parameter_List )
⟨DECL Data_Type Parameter_Name1
...
DECL Data_Type Parameter_NameZ
⟨further declarations
⟨Statements
ENDFCT
```

```
Parameter_List = Parameter_Name1
    ⟨ , ..., Parameter_NameI  : IN | ⟨OUT ⟨ , ...,
    ⟨ Parameter_NameN
    ⟨ , ..., Parameter_NameZ  : IN | ⟨OUT
```

自变量	类型	解释
GLOBAL		关键字**GLOBAL**使所有装入**KRL**程序的函数可用。
Data_Type		函数的数据类型
Function_ Name		函数名被定义。函数名是对象名而且字符不能超过24个字符。长度受控制器的目录系统限制。.
Parameter_ List		参数列表包含下列规定： 参数名 在输出情况下的参数类型排列（输入不能被排列），排列的尺寸用下列的符号增加排列名： **[ ]**一维空间排列 **[,]**二维空间排列 **[,,]**三维空间排列 参数各自的转换指令 **:IN** 输入参数（由值唤醒） **:OUT** 输出参数(有参数唤醒)（默认值）

Declarations	只有在声明部分可能出现声明。在这里不能使用程序语句。 与在第一个语句定义的声明部分和语句部分接近
Statements	只有在语句部分可能出现语句。函数的返回值使用**RETURN**语句转化。

2.2.16.3 说明

函数发送返回的值到调入指令。函数是表达式而且函数可以比被分配为变量或其他算术语句使用的。这儿有两种参数传送：通过输入参数传送和通过输出参数传送。

输入参数

变量的值被传送到这儿。直接的参数传送工作就像在子程序中分配默认设置到变量。通常传送的值可以是常量，变量函数，简单或复杂的表达式。

一个值在**IN**参数的情况下不能返回调入的指令(由值唤醒)。它仅仅由于赋一个值到子程序。

如果当前的数据类型和外部的**IN**参数不同但兼容，系统自动转化被传送的值的类型。数组不能像输入参数一样被传送 **(IN)**。

输出参数(关键字OUT)

变量名被传送到这儿（有参数唤醒）。变量在调用子程序时必须有一个值。这个值能被子程序用于调用。

参数类型**OUT**能在子程序调用时被分配一个值。由于这个原因当前的数据类型和外部参数必须在传送模式**OUT**中完全一致。

作为一个参数输出传送是默认的设置，例如：**OUT**不需要指定。

ENDFCT语句

ENDFCT语句总是的全球或本地子程序的最后程序段。子程序的最后程序段总是执行**RETURN**语句。如果注释遇到**ENDFCT**语句同时执行函数，那么出现运行时间错误。

2.2.16.4 范例

说明INT数据类型的函数，用FCT1命名而且传送参数P1和P2。

```
DEFFCT INT FCT1(P1,P2)
...
ENDFCT
```

声明BOOL数据类型的函数，用WELD命名而且输入参数CURRENT和VOLTAGE。

```
DEFFCT BOOL WELD(电流:IN,电压:IN)
...
ENDFCT
```

在函数中，不同的两个变量被计算出并传送到主程序。

```
DEF PROG_1( )
EXTFCT INT DELTA(INT:OUT,INT:OUT)
INT A,B,C
A=1
B=25
C=DELTA(A,B)
END

DEFFCT INT DELTA(X1:OUT,X2:OUT)
INT X1,X2,X3
X3=X2-X1
RETURN(X3)
ENDFCT
```

DEF，RETURN

2.2.17 DIGIN

2.2.17.1 摘要信息

循环读入数字输入。

2.2.17.2 语句

读数字输入：

```
DIGIN ON Signal_Value = Factor * $DIGINX ⟨± Offset
```

中止数字输入：

```
DIGIN OFF $DIGINX
```

自变量	类型	解释
Signal_Value	Real	操作的结果存储在Signal_Value中。他可以是变量或信号名.
Factor	Real	要素可以是常量，变量或信号名。
$DIGINX		***$DIGINX***指出预先确定的信号名： $DIGIN1到$DIGIN6。
Offset	Real	偏移量包含常量，变量或信号名。

所有在DIGIN语句用到的变量必须在数据列表中公告。

访问数字输入触发先前停止的运行。排列索引只在DIGIN ON语句求一次表达式的值。在排列索引已经被循环求得的数值代替之后延长。数字的输入不能用在INTERRUPT语句中使用。

2.2.17.3说明

控制器提供6个数字接口通过预先确定的变量$DIGIN1到$DIGIN读取。任何一个数字输入有32位而且带有选通输出。控制器能说明这些有信号或没有信号的输入。数字输入在文件“$MACHINE.DAT”中配置。

数字输入在使用**DIGINT**周期性的读取。数字输入能通过操作符“＝”分配REAL数据类型的变量。两个**DIGIN ON**语句在相同的时间内使用。两个**DIGIN ON**语句可以读取相同的数字输入。他可能是逻辑结合数字的输入，用其他的操作和使用**DIGIN**语句选择算法分配它们信号值。**DIGIN ON**语句也能访问模拟输入信号。

利用**DIGIN OFF**语句使循环读入数字输入信号名无效。

2.2.17.4 范例

数字输入$DIGIN1被分配二进制的输入信号从$IN[1020]到$IN[1026]。模拟输入t$ANIN[1]被分配符号名FACTOR。系统变量$TECHIN[1]分配到通过变量OFFSET的值增加的模拟输入和数字输入值乘积的结果。DIGIN OFF指令使循环读入数字输入无效。

```
SIGNAL $DIGIN1 $IN[1020] TO $IN[1026]
SIGNAL FACTOR $ANIN[1]
DIGIN ON $TECHIN[1] = FACTOR * $DIGIN1 + OFFSET
...
DIGIN OFF $DIGIN1
```

SIGNAL，ANIN，ANOUT

2.2.18 ENUM

2.2.18.1 摘要信息

声明列举数据类型。.

2.2.18.2 语句

```
GLOBAL ENUM Enumeration_Type_Name Enumeration_Constant1
    ⟨,..., Enumeration_ConstantN
```

自变量	类型	解释
Enumeration_ Type_Name		新列举数据类型的名字。
Enumeration_ Constant		列举的常数是值或列举数据类型能获得的变量。任何常量名必须有明确的数据类型。

在上下文中声明Enum数据类型时，关键字GLOBAL可在数据列表中使用。

2.2.18.3 说明

列举数据类型是有限的证书常量数据类型指令。列举常量表现列举变量能接受的离散的值。常量是自由定义名字和能被用户定义。

列举类型定义可能出现在指令的声明部分或数据列表中。全球的ENUM声明仅仅在数据列表中允许。.

列举常量的符号指定

你可以使用简略或完整的符号名。

如果使用简略的符号名，“#”字符插入在列举常量名之前。t,e.g.#monday.

如果使用完整的符号名，列举的常量由列举类型跟随的“#”字符和列举常量名确定，例如：WEEK_TYPE#MONDAY。这儿有两个列举常量必须使用完整符号名的实例：

1. 如果列举常量作为在子程序和函数的当前参数调用模块外的单独命令。
2. 如果列举常量需要与左边的比较。.

列举类型名以_TYPE结束以便区别列举类型的变量名。

2.2.18.4 范例

由STATE_TYPE和常量S_STAR，S_STOP和S_WAIT的名字声明列举数据类型。

ENUM STATE_TYPE S_STOP, S_START, S_WAIT

列举数据类型由SWITCH_TYPE和常量ON和OFF在后面的程序中公告。它们在程序中使用简略的符号名编址。

```
DEF PROG( )
ENUM SWITCH_TYPE ON, OFF
DECL SWITCH_TYPE ADHESIVE
IF A>10 THEN

        ADHESIVE=#ON
ELSE

        ADHESIVE=#OFF
ENDIF
END
```

DECL，STRUC

2.2.19 EXIT

2.2.19.1 摘要信息

无条件退出循环。

2.2.19.2 语句

```
EXIT
```

2.2.19.3 说明

EXIT出现在语句块循环中。它可以在任何循环中使用。

EXIT能被用于退出当前的循环。程序在**ENDLOOP**语句后继续。.

2.2.19.4 范例

从无穷的循环中退出。

```
LOOP
A=(A+1)*0.5/B
IF A>=13.5 THEN

 EXIT
 ENDIF
 ENDLOOP
```

FOR，WHILE，REPEAT，LOOP

2.2.20 EXT

2.2.20.1 摘要信息

声明外部的子程序。.

2.2.20.2 语句

```
EXT Program_Source(〈Parameter_List )
```

```
Parameter_List = Data_Type1
    〈 , ..., Data_TypeI  : IN | 〈OUT 〈 , ...,
    〈 Data_TypeN
    〈 , ...,Data_TypeZ  : IN | 〈OUT
```

自变量	类型	解释
Program_ Source		识别子程序的名字和路径。.
Parameter_ List		参数列表包含下面的说明： 所有外部子程序的参数的数据类型的定义次序。 在参数数组的情况下，数组的尺维数有下列符号: **[ ]** 一维数组 **[,]** 二维数组 **[,,]**三维数组 各自参数传送指令: **:IN** 输入参数(由值调入) **:OUT** 输出参数(由参数调入)(默认值) 参数列表的次序必须被遵守。单独的参数由逗号区分。

EXT声明不能在子程序中使用。

2.2.20.3 说明

EXT声明用于确定在主程序中可调用的外部子程序。只有主程序可以使用 (程序名与作为**SRC**文件的名子相同)，在相同的文件中没有其他的子程序。另外在**SRC**文件中的子程通常用关键字**GLOBAL**识别。

EXT声明必须被用于产生被调用子程序的名字和路径和编辑器使用的参数。通过制定参数列表，必须的存储空间也是明确定义的。

2.2.20.4 范例

声明在目录R1中名为SP1的外部子程序。

```
EXT /R1/SP1( )
```

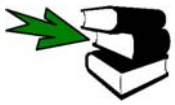
声明名为SP1的外部子程序和声明必要的参数。

```
EXT /SP1(INT,REAL:IN,CHAR[ ],INT[,,]:IN)
```

声明名为SP1的外部子程序和声明必要的参数。.

```
DEF PROG( ) ;主程序
EXT SP1(INT:OUT,REAL:OUT,BOOL:IN)
INT A
REAL B
BOOL C
...
SP1(A,B,C)
...
END

DEF SP1 (X1:OUT,X2:OUT,X3:IN);外部子程序
INT X1
REAL X2
BOOL X3
...
END
```



DEF，EXTFCT

2.2.21 EXTFCT

2.2.21.1 摘要信息

声明外部局部函数。

2.2.21.2 语句

```
EXTFCT Data_Type Program_Source(⟨Parameter_List )
```

```
Parameter_List = Data_Type1
    ⟨ , ..., Data_TypeI  : IN | ⟨OUT ⟨ , ...,
    ⟨ Data_TypeN
    ⟨ , ...,Data_TypeZ  : IN | ⟨OUT
```

自变量	类型	解释
Data_Type		数据类型必须符合函数声明。
Program_ Source		识别函数使用的路径和名字。
Parameter_ List		T数据列表包含下列说明： 所有外部函数的参数的数据类型的定义次序。 在参数数组的情况下，数组的尺维数有下列符号: **[ ]** 一维数组 **[,]** 二维数组 **[,,]**三维数组 各自参数传送指令: **:IN** 输入参数(由值调入) **:OUT** 输出参数(由参数调入)(默认值) 参数列表的次序必须被遵守。单独的参数由逗号区分。

EXTFCT声明不能在子程序中使用。

2.2.21.3 说明

EXTFCT声明用于识别程序中调用的外部函数。只有主程序能使用 (函数名与作为SRC文件的名子相同)，在相同的的文件中没有其它的函数。另外在**SRC**文件中的函数通常使用关键字**GLOBAL**识别。

使用EXT声明使编译器可以认识被调用的子程序的名字和路径及参数。通过参数列表，指定存储空间。

2.2.21.4 范例

声明在目录R1中名为FCT1的外部函数。

```
EXTFCT /R1/FCT1( )
DEF PROG( )
；主程序
EXTFCT INT DELTA(INT:OUT,INT:IN)
INT A,B,C
...
A=5
B=20
C=DELTA(A,B)
；调用函数
...
END
```

声明名为FCT1的外部函数而且传送参数。

```
EXTFCT /FCT1(INT,REAL:IN,CHAR[ ],INT[,,]:IN)
```

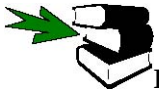
声明名为FCT1的外部函数而且传送参数。

```
EXTFCT FCT1(INT:OUT,REAL:OUT,BOOL:IN)
```

在函数中，两个的变量被计算和传送到主程序是有差别的。变量的值在调用函数之后：:A=15,B=20,C=15.

```
DEF PROG( )
；主程序
EXTFCT INT DELTA(INT:OUT,INT:IN)
INT A,B,C
...
A=5
B=20
C=DELTA(A,B)

；函数调用
...
END
DEFFCT INT DELTA(X1:OUT,X2:IN)
；外部函数
                    INT X1,X2,X3
                    X1=X2-X1
                    X3=X1
                    RETURN(X3)
                    ENDFCT
```



DEFFCT，EXT

2.2.22 FOR...TO...ENDFOR

2.2.22.1 摘要信息

循环计算。

2.2.22.2 语句

```
FOR Counter = Start TO End ⟨STEP Increment
⟨Statements
ENDFOR
```

自变量	类型	解释
Counter	INT	循环计算使用整数变量
Start	INT	表达式指定计算器最初的值。
End	INT	表达式指定计算器最后的值。
Increment	INT	算术表达式的计算器总量由每个执行的循环增加： 增量可能是负数 增量可能是零 增量可能是变量 如果增量被指定，那么默认值为1。

2.2.22.3 说明

指定运行数能真正明确**FOR**循环被程序使用。循环运行由计算器帮助计算。

执行**FOR**的条件:

- 通过正增量:如果计算器的值比最后的值大，那么循环结束。
- 通过负增量：如果计算器的值比最后的值小，那么循环结束。

执行的条件是在任何一个循环运行之前选择。在个别的情况下**FOR**循环不能完全执行。

表达式类型的整数必须给出计算器的最初和最后的值。表达式求一次循环开始的值。计时器被最初的值调整和增加或在循环运行后消耗。

增加可以不是0。如果不指定增加，那么默认的值是1。负值也能被用于增加。

计数器的值能在循环语句的内部和外部使用。在循环内部，作为排列处理的最新的指数，在循环外，计数器保留大部分新近来的值。

为了每个**FOR**语句都必须**ENDFOR**语句对应。在循环执行完成后，程序在 遇见**ENDFOR**后返回第一个指令。使用**EXIT**语句循环可以提前退出。

2.2.22.4 范例

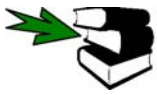
在10循环中变量B每次增加1。

```
FOR A=1 TO 10
B=B+1
ENDFOR
```

在两级FOR循环中,增加计数器A后每次运行通过计数器的值增加变量B。变量B达到10，循环提前退出。

```
FOR A=1 TO 15 STEP 2
        B=B+A
        IF B==10 THEN

            EX
  IT
  ENDIF
  ENDFOR
```



EXIT，SWITCH，REPEAT，WHILE，LOOP

2.2.23 GOTO

2.2.23.1 摘要信息

无条件跳跃语句

2.2.23.2 语句

```
GOTO Marker
```

自变量	类型	解释
Marker		Marker说明跳跃语句的目的地

2.2.23.3 说明

无条件跳跃语句**GOTO**是程序编程指令。在**GOTO**语句使用后，通过**GOTO**语句程序将在指定的位置恢复执行。

作为**GOTO**语句的目的地必须是在同一个子程序和函数中。它通过跟随标记冒号定义。

在IF语句或在循环中跳跃到外面，或从一个CASE语句跳跃到到另一个CASE语句是不可能的。

当使用GOTO时，程序很快变得混乱和无组织的。它比使用分支语句IF或选择语句SWITCH代替要好。

2.2.23.4 范例

无条件的跳跃到程序的MARKER_1位置。

```
GOTO MARKER_1
```

无条件的从IF语句跳跃到程序END的位置.

```
IF X>100 THEN
GOTO END
ELSE
   X=X+1
ENDIF
A=A*X
...
END:
END
```

IF，SWITCH，REPEAT，WHILE，LOOP

2.2.24 HALT

2.2.24.1 摘要信息

中断程序的执行和停止处理。

2.2.24.2 语句

```
HALT
```

2.2.24.3 说明

HALT语句停止程序的执行。最后的运动指令被无论如何被完全执行。

程序只有重新使用开始键才能执行。在**HALT**后的下一个指令被执行。.

在中断程序，在前面的程序完全执行后程序执行停止。在BRAKE语句的情况下，程序立即执行停止。

WAITFOR，WAITSEC，BRAKE

2.2.25 IF...THEN...ENDIF

2.2.25.1 摘要信息

依靠逻辑表达式的结果执行语句。

2.2.25.2 语句

```
IF Condition THEN
      Statements
⟨ELSE
      Statements
ENDIF
```

自变量	类型	解释
Condition	BOOL	逻辑表达式能包含布尔变量、布尔函数或布尔结果的逻辑操作。

2.2.25.3说明

分支语句**IF**是程序执行指令。它依靠第一语句块（THEN black）或第二的语句块(ELSE block)来执行。程序由语句后跟随的**ENDIF**延续。

语句块中的包含的语句数是没有限制的。几个IF语句可以互相嵌套。

关键字**ELSE**和第二语句块可以被省略。如果条件没有满足，程序继续直接立即运行到**ENDIF**。

在任何IF中必须有**ENDIF**。

2.2.25.4 范例

没有第二语句块IF循环。.

```
IF A==17 THEN
B=1

ENDIF
```

有第二语句块IF循环。

```
IF $IN[1] THEN
$OUT[17]=TRUE
ELSE
$OUT[17]=FALSE
ENDIF
```

2.2.26 IMPORT...IS.....

2.2.26.1 摘要信息

从数据列表输入数据。

2.2.26.2 语句

```
IMPORT Data_Type Import_Name IS Data_Source..Data_Name
```

自变量	类型	解释
Data_Type		数据必须是在外部数据列表中公告过的数据类型。
Import_Name		引入数据能被分配一个与外部数据列表中的数据名不同的名字。
Data_Source		数据的来源看作被调入的数据在数据列表中的路径和名字。
Data_Name		数据名与变量名分配在外部数据列表中的数据符合。

2.2.26.3 说明

IMPORT语句允许外部的数据列表影响访问“**PUBLIC**”的属性。**IMPOR**语句允许变量，完整的排列或排列元素从外部的数据列表定义你自己的程序活数据列表。任何变量的调入需要**IMPORY**语句承认。已经被调入的变量不能使用其它**IMPORT**语句分配。变量必须从最初创建的数据列表中调入。

数据必须由外部数据列表公告的类型相同的数据调入。直到连线操作执行，系统才检查正确的数据被选择。

数据能在你自己的数据列表或有外部数据列表的程序有不同的名字。数据列表的目录和名字由路径指定。数据名与外部数据列表中的名字相应。

数据来源和数据名在两个周期内互相连接。在两个周期内没有空白。

2.2.26.4 范例

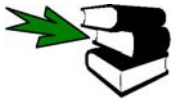
从数据列表DATA中调用整数变量VALUE的值。

IMPORT INT VALUE IS /DATA..VALUE
简化格式

IMPORT INT VALUE

在数据列表R1/POSITION中调用POS排列POS_EX。

IMPORT POS POS1[ ] IS /R1/POSITION..POS_EX



DEFDAT

2.2.27 INTERRUPTDECL...WHEN...DO

2.2.27.1 摘要信息

声明中断。.

2.2.27.2 语句

```
GLOBAL INTERRUPT DECL Prio WHEN Event DO Subprogram
```

自变量	类型	解释
GLOBAL		关键字**GLOBAL**用于识别中断包括子程序以上级别的被公告的中断。 如果中断在子程序中公告，那么在主程序调用时是经过验证的。.
Prio	INT	算术表达式指定先前的中断。先前的级别1到128可用，但是系统自动优先分配40到80范围保留。.
Event	BOOL	逻辑表达式定义中断的结果。下面是允许的： 布尔常量 布尔变量 信号名 比较 简单的逻辑操作：**NOT**，**OR**，**AND**或**EXORT** 不能用于： ◆结构部分
Subprogram		子程序的名字和参数(中断程序)，它在中断出现时执行。

如果程序被延伸到HAL语句，中断依然被检测和执行(包括运动指令)。在指令被执行后，程序在HALT语句暂停一次。

中断声明是指令。它不是必须在声明部分定位。

在第一次公告时，中断被解除和失效。

运行时间变量可能不作为中断程序的参数被传送和在数据列表中区分变量公告。

同时最多公告32个中断。

2.2.27.3 说明

中断功能允许用户在程序执行的不同时发生事件使用程序语句来反映。这种事件可以是紧急停止，错误信息和输入信号。中断可能的原因和系统对对中断声明的各自的定义的反映。中断可以被分配优先权，事件和中断程序可以被调用。在相同的时间内可以调用32个中断。声明可以在任何时候被另一个声明覆盖。

如果下面所有的四个条件都满足，定义中断触发一个反映：：

1. 中断必须被触发(中断打开).
2. 中断必须被允许(中断使能).
3. 中断必须是最高级。
4. 有关的事件必须发生。事件发生由边沿触发。(边沿触发)

如果几个中断同时出现，最高级的中断首先执行，然后按优先级高低顺序执行。中断直到声明级别后才发现。高级别的编程尽管中断被激活，中断仍不能识别。换句话说，在子程序识别中断不能被主程序识别。为了识别主程序的中断和级别，它必须作为全球的公告。

在事件被发现后，机器人存储当前的位置和调用中断程序。中断可以被本地子程序和外部子程序使用。中断在结束时使用**RENTURN**和**END**语句是惯例。中断程序是在随后中断出现的位置恢复(除了**RESUME**的情况)。

在一般程序中触发先前运行中止语句，不能在子程序中这样做。中断程序在命令等级上运行，它是逐段的按次序执行。

程序变量$EM_STOP和$STOPMESS的中断在错误的情况下执行，比如**INTERRUPT**语句不论机器人是否停止都执行 (忽视运动指令).。

任何被公告和激活的中断能在操作停止期间发现一次。在系统重新开始后，中断按时中断执行的优先权的次序发现（如果有使能）。程序随后继续。

在调用中断程序时，参数正确地传送就像调用一般的子程序一样。

2.2.27.4 范例

定义中断优先级5在变量$STOPMESS为真时调用子程序STOPSP。

```
INTERRUPT DECL 5 WHEN $STOPMESS DO STOPSP( )
```

定义中断优先级23在$IN[12]为真时调用子程序SP1使用参数20a和VALUE。

```
INTERRUPT DECL 23 WHEN $IN[12]==TRUE DO SP1(20,VALUE)
```

两个物体能被两个连接到输入6和7传感器检测且位于预先编程的路径上。机器人移动到后面的这两个位置。

```
DEF PROG( )

        ;主程序
INTERRUPT DECL 10 WHEN $IN[6]==TRUE DO SP1( )
INTERRUPT DECL 20 WHEN $IN[7]==TRUE DO SP2( )
...
LIN START_POINT
INTERRUPT ON
INTERRUPT ENABLE
LIN END_POINT
INTERRUPT OFF
                ;移动到参考点
LIN POINT1
LIN POINT2
...
END
DEF SP1( )

        ;本地中断程序1
POINT1=$POS_INT
END
DEF SP2( )

        ; 本地中断程序2
POINT2=$POS_INT
END
```

INTERRUPT，BRAKE，RESUME，TRIGGER

2.2.28 INTERRUPT

2.2.28.1 摘要信息

激活和取消中断。

2.2.28.2 语句

```
INTERRUPT Action ⟨Priority
```

自变量	类型	解释
Action	关键字	有关的关键字 **ON** 激活 **OFF** 取消 **ENABLE** 使能 **DISABLE** 禁止 公告中断
Priority	INT	算术表达式指定你想要使能或禁止的中断的优先级。 如果这个参数被遗漏，,激活和禁止的中断语句指的是所有被公告的中断。 而且使能和无效的中断语句指的是所有被激活的中断

2.2.28.3 说明

语句被用于中断的激活、取消、使能和无效的执行。

如果公告的中断被激活，那么它将循环监控。当事件被发现，那么发生的事件和当前实际的位置被存储。无效的中断不能被执行。

被激活的中断能被使能或禁止。禁止语句允许程序部分反对中断。无效的中断被验证和保存但没有执行。中断一激活就按优先级执行中断。如果中断在激活前被禁止，被保存的事件没有更多的作用。如果中断出现在被禁止时，那么在使能后中断仅执行一次。.

当出现中断且在它执行时所有优先级比它低的中断无效。，当从中断程序返回后，中断再次被使能。当前的中断应用于再次直接调用，例如信号保持到中断程序被执行后。如果你想停止开始再次调用中断，那么中断必须被取消或禁止。中断程序总是完全彻底的被执行，不考虑中断是否被禁止或取消。

在中断程序中的第一个语句由高优先级级的中断后，中断能被自己再一次中断。编程员这样才能防止在最初的指令中禁止或取消一个或多个中断。在高优先级的一个中断结束后，被中断的中断程序在他被中断的位置重新开始。

同时可以有**16**个中断被激活，在中断激活的情况下，需要特别注意

2.2.28.4 范例

声明中断优先级2被激活.

INTERRUPT ON 2

声明中断优先级5被取消.

INTERRUPT OFF 5

声明所有中断被激活

INTERRUPT ON

声明所有中断被取消.

INTERRUPT OFF

被激活的中断优先级3被使能

INTERRUPT ENABLE 3

被激活的中断优先级2被禁止

INTERRUPT DISABLE 2

所有被激活的中断被使能

INTERRUPT ENABLE

所有被激活的中断被禁止.

INTERRUPT DISABLE

通过硬件在粘贴应用期间执行非路径保持急停，你应该使用程序停止该粘贴应用程序而且在使能后重新配置粘贴枪到轨迹（通过输入10）。

```
DEF PROG( )
                              ; 主程序
...
INTERRUPT DECL 1 WHEN $STOPMESS DO STOPSP( )
...
LIN POINT1
INTERRUPT ON
INTERRUPT ENABLE
LIN POINT2
INTERRUPT OFF
...
END
```

```
DEF STOPSP( )
                                    ;中断程序
BRAKE F
                                  ;快速中止运动
ADHESIVE=FALSE
WAIT FOR $IN[10]
LIN $POS_RET
                                ;在轨迹左面位置
ADHESIVE=TRUE
END
```

INTERRUPT DECL, BRAKE, RESUME, TRIGGER

2.2.29 LIN

2.2.29.1 摘要信息

直线运动

2.2.29.2 语句

```
LIN Target_Position ⟨Approximate_Positioning
```

自变量	类型	解释
Target_ Position	**POS, E6POS FRAME**	几何表达式指定直线运动的目标点。 在这里只能使用笛卡儿坐标系 笛卡儿目标位置的参考系统由系统变量$BASE定义。 角度状态用**S**和**T**相对于目标点的位置类型**POS**或**E6POS**在总是被忽略。 如果目标位置包含不明确的结构成分， 那么这些值不从当前位置变化。 如果目标点是被示教的后 ，“**!**”可以作为目标点被编程。
Approximate_Positioning	Keyword	选项允许你使用近似定位。可能是： **C_DIS**(默认值) **C_ORI** **C_VEL**

编程路径在TCP的数度和加速度::

	变量	数据类型	单位	功能
速度	$VEL.CP	REAL	m/s	移动速度(路径速度)
	$VEL.ORI1	REAL	°/s	旋转速度
	$VEL.ORI2	REAL	°/s	转动速度
加速度	$ACC.CP	REAL	m/s²	路径加速度
	$ACC.ORI1	REAL	°/s²	旋转加速度
	$ACC.ORI2	REAL	°/s²	转动加速度

用LIN运动控制刀具的方向:

变量	作用
$ORI_TYPE=#CONSTANT	在路径运动方向保持常量期间；编程的方向使用开始点的方向而忽略目标点的方向。
$ORI_TYPE=#VAR	在路径运动期间方向不断从最初的方向变化到目地方向。

系统变量定义近似开始的位置：

变量	数据类型	单位	意义	命令中的关键字
$APO.CDIS	REAL	mm	移动距离标准	**C_DIS**
$APO.CORI	REAL	0	方向尺寸	**C_ORI**
$APO.CVEL	INT	%	速度标准	**C_VEL**

2.2.29.3 说明

在**LIN**运动的情况下，控制器计算直线方程从当前的位置到达在**LIN**指令中指定的目标位置。机器人通过计算出的辅助点和执行一定间隔的插补循环到达中止点。

就像**PTP**运动，速度与加速度和系统变量$TOOL与$BASE两者都必须被用于直线运动编程。速度和加速度不再与各轴电机速度有关但是与**TPC**有关。系统变量

$VEL　路径速度

$ACC　路径加速度

角度状态

在**LIN**运动的情况下，终点的角度状态总是看作与起始点相同。由于这个原因，目标点的数据类型**POS**和**EPOS**指定的**S**和**T**总是可以被忽略。

为了总是保证同样的运动顺序，轴的坐标必须首先被明确定义。程序首先的运动指令必须总是被PTP指令指定的S和T。

方向

你能由系统变量$ORI_TYPE选择常量或变量方向:

- **$ORI_TYPE=#VAR**
  在直线运动期间，方向变化一律由开始方向到目标方向。(默认设置).
- **$ORI_TYPE=#CONSTANT**
  在直线运动期间的方向为常量，由于程序使用结束点和起始点所以编程的方向被忽视。

如果轴的角度在LIN或CIRC运动期间变换3或5度，如果轴加速无限快速的运动，那么方向被保持。这是不可能的，当超过电机限定值时，控制器将中止运动并输出错误信息，或减小速度继续运动。

系统变量$CP_VEL_TYPE用于定义操作在速度超过轴的限定值时降低速度方式。变量可以采用3种值:

#VAR_T1　　在T1方式中速度减小.
#VAR_ALL　　在所有方式中速度减小。
#CONSTANT　　这个功能没有使能。

用户能定义操作方式，在此操作方式中通过信息窗口的信息来指示速度减小：

$CpVelRedMeld=1　　仅在T1和T2的方式中指示速度减小。

$CpVelRedMeld=100　　在所有的方式中指示速度减小。

近似定位

机器人准确定位到辅助点是要耗费时间的，这是不必要的。因此你能定义过渡到下一个运动程序段(**PTP**，**LIN**或**CIRC**)距离目标点的距离(所谓的近似定位)。 最大的近似距离是编程距离的一半。

近似定位的编程有两步：
使用系统变量$APO定义近似定位的范围：

--$APO.CDI

移动距离的标准(由C_DIS激活)::近似定位的轮廓从距离目标点的规定距离(单位[mm])处开始。

--$APO.CORI

方向距离(activatedbyC_ORI)：当支配的角小于距离目标点规定的距离时，**TCP**离开单独的段轮廓。

--$APO.CVEL

速度标准(由C_VEL激活)：当在$VEL.C 中定义的速度百分比$APO.CVEL 到达时，定义的近似定位轮廓开始。

由目标位置和近似定位模式规划运动指令：

--LIN--LINorLIN—CIR近似定位

为了**LIN—LIN**近似定位，控制器计算抛物线路径。在**LIN－CIRC**近似定位的场合下，对称的近似定位轮廓不能被计算出。近似定位路径由两段抛物线组成，也由彼此的切线和个别的块转变。近似定位由被编程的关键字**C_DIS**，**C_ORI**或**C_VEL** 开始。

--LIN—PTP近似定位

提前处理的近似定位不能在LIN块中使机器人的轴旋转超过180^0，而且位置S不变。近似定位的开始点由变量$APO.CDIS，$APO.CORI和$APO.CVEL定义，中止点由变量$APO.CPTP定义。关键字中的 C_DIS, C_ORI 和 C_VEL的一个在LIN指令中被编程。

为了近似定位，计算机预先运行必须是在被使能后。如果不是，信息“Approximation not possible”将被显示。在LIN－PTP近似定位的场合中，提前的运行被限制为1。

注意：

近似定位的程序段之间不能出现停止提前运行的程序语句。(用CONTINUE补救).

速度和加速度越大背离轨迹的动态偏差越大。 (跟踪误差).

变化加速度比变换速度的在路径轮廓效果小。

2.2.29.4 范例

目标坐标系的直线运动。

LIN !

直线运动的目标坐标系编程：激活近似定位.

LIN POINT1 C_DIS

在基本(笛卡儿)坐标系中指定目标位置。

LIN {X 12.3,Y 100.0,Z -505.3,A 9.2,B -50.5,C 20}

指定目标位置的两个值。旧的分配被保留的值保留。

LIN {Z 500,X 123.6}

借助于几何操作指定目标位置：由BASE坐标系定义的，在TOOL坐标系系统中X方向减少30.5毫米，而且在Z方向增加20毫米到达位置1。

LIN POINT1:{X -30.5,Z 20}

LIN—PTP从第2点到第3点近似定位。近似定位在第2点前30mm开始。.

```
$APO.CDIS=30
$APO.CPTP=20
PTP POINT1
LIN POINT2 C_DIS
PTP POINT3
```

LIN_REL，PTP，CIRC，CONTINUE

2.2.30 LIN_REL

2.2.30.1 摘要信息

在关系坐标系中的直线运动。

2.2.30.2 语句

```
LIN_REL Target_Position ⟨Approximate_Positioning
```

自变量	类型	解释
Target_Position	**POS** **E6POS** **FRAME**	几何表达式指定直线运动的目标点。在这里最好使用笛卡儿坐标系来解释当前的位置关系。 目标位置不能示教。 编译的距离在基本坐标系$BASE的轴的方向上执行。 如果目标位置包含未定义的组成部分，那么这些值设为0， 例如绝对值保持不变。 预先确定的变量$ROTSYS定义方向部分被编程的结果。 目标点的数据类型**POS**或**E6POS**的角度状况**S**和**T**总是被忽略。
Approximate_Positioning	Keyword	这个选项允许你使用近似定位。选项可能是： **C_DIS**(默认值) **C_ORI** **C_VEL**

TCP的编程路径的速度和加速度:

	变量	数据类型	单位	功能
速度	$VEL.CP	REAL	m/s	移动速度(路径速度)
	$VEL.ORI1	REAL	^{0}/s	旋转速度
	$VEL.ORI2	REAL	^{0}/s	转动速度
加速度	$ACC.CP	REAL	m/s^0	路径加速度
	$ACC.ORI1	REAL	/s^0	旋转加速度
	$ACC.ORI2	REAL	/s^0	转动加速度

用LIN运动控制刀具方向

变量	作用
$ORI_TYPE=#CONSTANT	在路径运动期间角度为常量；编程时的方向不理睬目的点，而使用起始点的方向。
$ORI_TYPE=#VAR	在路径运动期间，方向从最初的方向变化为目的点的方向。

系统变量定义近似定位的开始

变量	数据类型	单位	信息	命令中的关键字
$APO.CDIS	REAL	mm	移动距离标准	**C_DIS**
$APO.CORI	REAL	0	角度距离	**C_ORI**
$APO.CVEL	INT	%	速度标准	**C_VEL**

2.2.30.3 说明

相对的**LIN**指令基本上与绝对**LIN**指令的作用相同。目标坐标系仅指定绝对空间坐标系或轴坐标系相对于当前的位置关系。

除了这些，绝对**LIN**指令方面包含所有的信息应用在这儿。

2.2.30.4 范例

机器人从当前的位置在X方向移动100mm而且在Z方向移动200mmY，A，B，C，S保持不变而且T由运动决定。

LIN_REL {X 100,Z -200}

LIN—LIN近似定位从1点到2点，**LIN—CIR**近似定位从2点到3点。当速度减少到0.3m/s(30%of0.9m/s)时，近似定位从1点开始。近似定位轮廓在2点前20mm开始。

```
$VEL.CP=0.9
$APO.CVEL
=30
$APO.CDIS=
20
LIN POINT1 C_VEL
LIN_REL POINT2_REL C_DIS
CIRC AUX_POINT,POINT3
```

LIN，PTP_REL，CIRC_REL，CONTINUE

2.2.31 LOOP...ENDLOOP

2.2.31.1 摘要信息

无穷循环的编程。

2.2.31.2 语句

```
LOOP
Statements
ENDLOOP
```

2.2.31.3 说明

循环执行能被使用**LOOP**编程。在**LOOP**中语句块被不断的重复执行。如果你想中止重复语句块的执行，你必须使用**EXIT**语句。

2.2.31.4 范例

无穷循环

```
 LOOP
A=A+1
IF A==65 THEN
        EXIT
        ENDIF
        ENDLOOP
```

EXIT，SWITCH，FOR，REPEAT，WHILE

2.2.32 PTP

2.2.32.1 摘要信息

点到点的运动

2.2.32.2 语句

```
PTP Target_Position ⟨C_PTP ⟨Approximate_Positioning
```

自变量	类型	解释
Target_ Position	**POS E6POSAXIS, E6AXIS, FRAME**	几何表达式指定运动的目标点。在这里可以使用笛卡儿和轴坐标系 笛卡儿参考系统的目标位置由系统变量$BASE定义。 如果目标位置包含不明确的结构成分，那么这些值不从当前位置变化。 如果“**!**”被作为目标位置被编程随后使用，目标位置是能被示教的，
C_PTP	Keyword	这个选项导致机器人以近似定位到达PTP运动的目标点。 这个规定对于**PTP－PTP**近似定位是充分的。 近似定位选项必须被后来的**CP**块中的近似定位指定。
Approximate_ Positioning	Keyword	这个选项用于指定输入后来的**CP**运动块的近似定位的标准。 它也可以用于与**C_PTP**联合的选项。 可能的是 **C_DIS** 距离标准(默认值) **C_ORI** 方向标准 **C_VEL** 速度标准

2.2.32.3 说明

PTP运动是机器人手臂快速的从当前位置移动到变编程的目标位置的方法。轴是同步移动的，例如所有的轴同时开始和结束。控制器计算各轴的速度，因此至少一个轴在预先确定的速度和加速度的界限中移动。最大速度和最大加速度必须被各轴单独编程。系统变量：

$VEL_AXIS[*No*] 指定轴的速度

$ACC_AXIS[*No*] 指定轴的加速度

如果这两个系统变量在首次运动指令前没有被编程，那么在程序执行是将显示错误信息。如果目标点在笛卡儿坐标系中指定，这也同样适用于系统变量$TOOL和BASE。.

角度状况

由于运动学的特征，机器人可以由不同角度的轴到达相同的空间位置。轴的角度位置能由几何表达式中的。S(状况)和**T**(转动) 明确的定义。两者都需要由二进制的整数输入。位有下面的意义：

状况：

- Bit1：根部关节位置t(0基本区,1头顶区)
- Bit2：轴3的角度位置(0负,1正)
- Bit3：轴5的角度位置(0正,1负)。角度位置是各情况下的相对各轴的确定的原点间的关系。

转动：

- Bitx：轴x的角度位置(0正，1负)

角度位置是各情况下的相对于各轴的确定原点间的关系。

如果Turn T在PTP运动中省略，机器人将总是移动最短的角度路径。如果状况S被省略，先前点的状况被保留。为了保证运动次序是同样的，程序的首次运动必须总是被PTP指令指定S和T。

近似定位

机器人精确的到达辅助点是不必要的和耗费时间的。因此你能定义过渡到下一个运动程序段(**PTP,LIN**或**CIRC**)距离目标点的距离(所谓的近似定位)。 最大的近似距离是编程距离的一半。

近似定位被编程的两步：

定义系统变量$APO的近似定位范围：

--$APO.CPTP轴向近似定位标准(由C_PTP激活)：当在$APO_DIS_PTP[*No*]中定义的主要的轴是$APO.CPTP角度最大百分比远离近似定位点时，近似定位开始。

--$APO.CDIS移动距离标准(由C_DIS激活)：近似定位轮廓在从近似定位点指定的距离处开始。.

--$APO.CORI方向距离(由C_ORI激活)：当可支配的角度小于从近似定位点到指定距离处的角度时，**TCP**允许单的独块轮廓。

--$APO.CVEL速度标准(由C_VEL激活)：当在$VEL.CP指定的 $APO.CVEL速度百分比被到达，近似定位开始。

由目标点编程的运动指令和近似定位方式：

--PTP—PTP近似定位

在**PTP**指令中的程序关键字**C_PTP**包含机器人近似定位的目标位置。近似定位范围是由变量$APO.CPTP 分配的值来定义。当轴最后下降到低于目标位置的指定角度时，近似定位开始。近似定位轮廓描述一个空间抛物线。这是由控制器计算出的，编程员对此没有影响。可是近似定位的开始可以被编程。

--PTP—LIN或PTP—CIRC近似定位

提前处理的近似定位不能使机器人在**LIN**或**CIRC**块中使轴的转动超过180^0，而且位置**S**不变。近似定位的开始由变量$APO.CPTP 定义。近似定位的结束由变量APO.CORI或$APO.CVEL中的一个定义。现在**PTP**语句中的程序关键字**C_PTP** 和**C_DIS**(默认值),**C_ORI**或**C_VEL**中德一个定义近似定位的开始。

由于近似定位，电脑提前运行必须被使能。如果不是，信息“Approximationnotpossible”将被显示。

- $APO.CPTP越小，近似定位范围越小。
- 在两个近似定位点之间不能出现A$TOOL语句。
- 速度加速度越大，路径的动态误差越大。 (跟随误差).
- 改变加速度比改变速度对于路径轮廓的效果小。
- 如果机器人使用$APO.CPTP=0近似到达近似定位点,机器人实际上准确到达近似定位点，但运行的时间并没有减少。 (跟随误差没有被排除).

2.2.32.4 范例

PTP运动到目标坐标示教

PTP !

PTP运动到编程目标坐标；激活近似定位。

PTP POINT1 C_PTP

说明目标点的基本坐标(笛卡儿)。

PTP {X 12.3,Y 100.0,Z 50,A 9.2,B 50,C 0,S ’B010’,T ’B1010’}

说明目标点的特定轴坐标。

PTP {A1 10,A2 -80.6,A3 -50,A4 0,A5 14.2, A6 0}

说明仅有两个值的目标位置。旧的分配之辈保留。

PTP {Z 500,X 123.6}

使用几何运算说明目标位置：在工具坐标系X方向延长100毫米到达基本坐标系的1点。.

PTP POINT1:{X 100}

PTP—LIN近似从2点到3点。当在主要的轴到达2点的路途中，主要的轴剩余的移动的角度小于在$APO_DIS_PTP[*No*]中规定的最大值的20%时，近似定位开始。

```
$APO.CORI=10
$APO.CPTP=20
PTP POINT1
PTP POINT2 C_PTP C_ORI
LIN POINT3
```

PTP_REL，LIN，CIRC，CONTINUE

2.2.33 PTP_REL

2.2.33.1 摘要信息

在关系坐标的点到点运动

2.2.33.2 语句

```
PTP_REL Target_Position ⟨C_PTP ⟨Approximate_Positioning
```

自变量	类型	解释
Target_ Position	POS E6POSAXIS, E6AXIS, FRAME	几何表达式指定运动的目标点。在这里可以使用笛卡儿和轴坐标系说明相对当前点的关系。 笛卡儿参考系统的目标位置由系统变量$BASE定义。 如果目标位置包含不明确的结构成分，那么这些值 不从当前位置变化。 如果“!”被作为目标位置被编程随后使用，目标位置是能被示教的。
C_PTP	Keyword	这个选项将导致机器人以近似定位到达**PTP**运动的目标点。 这个规定对于**PTP－PTP**近似定位是充分的。 近似定位选项必须被后来的**CP**块中的近似定位指定。
Approximate_ Positioning	Keyword	这个选项用于指定输入后来的CP运动块的近似定位的标准。 P它也可以用于与C_PTP联合的选项。 可能的是 **C_DIS** 距离标准(默认值) **C_ORI** 方向标准 **C_VEL** 速度标准

2.2.33.3 说明

PTP指令与绝对**PTP**指令的工作方法是一样的。目标坐标是由绝对空间或轴坐标代替当前位置。因此你能由指定机器人指定的空间坐标的角度或步骤移动各轴。

这个出外，包括指定的绝对**PTP**指令所有信息的应用在这儿：

2.2.33.4 范例

轴2负方向移动30度。其他轴不动。

PTP_REL {A2 -30}

机器人从当前点在X方向移动100mm；在Z负方向移动200mm。.Y,A,B,C,S保持不变而且T是由最短路径计算出的。

PTP_REL {X 100,Z -200}

PTP—PTP近似定位从1点到2点而且PTP—CIRC近似定位从2点到3点。当主要的轴在剩余的角度低于在$APO_DIS_PTP[*No*]中规定的角度最大值的40%时，近似定位开始，路径位分别到达1点和2点。

```
$APO.CDIS=
30
$APO.CPTP=
40
PTP POINT1 C_PTP
PTP_REL POINT2_REL C_PTP;C_DISissetbydefault
CIRC AUX_POINT,POINT3
```

PTP，LIN_REL，CIRC_REL，CONTINUE

2.2.34 PULSE

2.2.34.1 摘要信息

激活脉冲输出

2.2.34.2 语句

```
PULSE (Signal, Level, Pulse_Duration)
```

自变量	类型	解释
Signal	BOOL	输出到馈给的脉冲。下面是允许的： **UT[***No***]** 信号变量
Level	BOOL	逻辑表达式: **TRUE**表示正的脉冲输出(高) **FALSE**表示负脉冲输出(低)
Pulse_ Duration	REAL	算术表达式指定脉冲周期。周期值范围从1——3秒。脉冲间隔0.1秒。例如脉冲周期是上升或下降。.

2.2.34.3 说明

PULSE语句用于激活脉冲输出。当程序被执行后，二进制输出的格式设置定义电平的时间周期。在脉冲周期消失后，系统自动复位输出信号。输出信号的设置和复位与前面输出信号的长度无关。

如果在一个脉冲期间相反的被激活的电平输出，脉冲缩短。如果在下降沿前再一次激活脉冲，脉冲复位。在脉冲输出正电平的情况下，二进制输出为**TURE**；在脉冲输出负电平情况下，二进制输出为**FALSE**。

因为**PULSE**语句由控制器执在低时钟频率时执行，脉冲间隔产生的公差（0.1秒）。时间偏离平均在1%－2%左右。非常短的脉冲的偏离在13%左右。

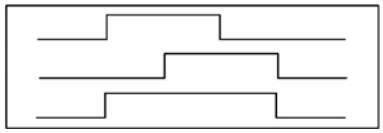
在程序运行时，仅检测允许的外部的脉冲周期的时间间隔；它引发程序停止.
最多可以有16个脉冲被同时编程。
在程序停止时，编程的脉冲延续到消失。
就RESET和CANCEL的来说，另一方面，脉冲被停止。
激活的脉冲能被中断影响。
PULSE语句触发前面的运行停止。只有在TRIGGER语句中与机器人的动作联合执行。
在脉冲激活期间，如果程序延伸到END语句，脉冲没有被停止。
如果脉冲输出被首次运动前编程，如果再次释放Start键而且机器人还没由到达路径，脉冲宽度也消失。 (BCO).

在紧急停止，操作停止或错误停止时脉冲没有被中止。

2.2.34.4 范例

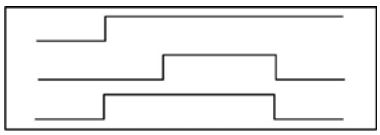
如果脉冲输出在下降沿前被激活，脉冲重启。

```
PULSE($OUT[50],TRUE,0.5)
PULSE($OUT[50],TRUE,0.5)
Output 50
```



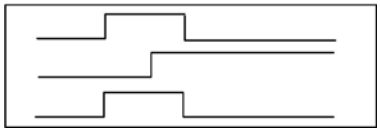
如果输出已经设置，那么它将由脉冲的下降沿重新复位。

```
$OUT[50]=TRUE
PULSE($OUT[50],TRUE,0.5)
Output 50
```



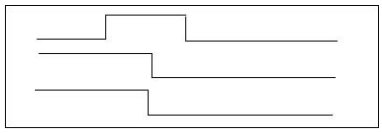
如果在脉冲持续期间设置同样的输出，那么他将由脉冲输出的下降沿复位。

```
PULSE($OUT[50],TRUE,0.5)
$OUT[50]=TRUE
Output 50
```



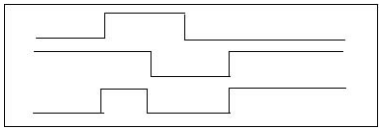
如果在脉冲持续时间输出50复位，脉冲持续时间被减少。

```
PULSE($OUT[50],TRUE,0.5)
$OUT[50] = FALSE
Output 50
```



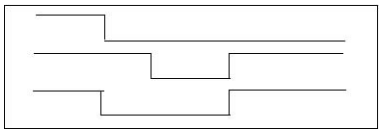
如果在脉冲的正脉冲期间负脉冲应用于同样的输出，正电平设为低和在正脉冲消失后复位为高。

```
PULSE($OUT[50],TRUE,0.5)
PULSE($OUT[50],FALSE,0.5)
Output 50
```



如果负脉冲被用于低电平输出，输出保持低电平直到脉冲结束和被设为高电平。

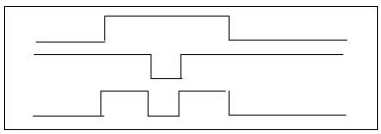
$OUT[50] = FALSE
PULSE($OUT[50],FALSE,0.5)
Output 50



如果在脉冲期间输出复位和再次设置，输出在脉冲结束时再次复位。

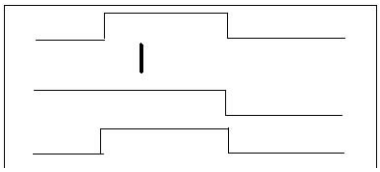
PULSE($OUT[50],TRUE,0.8)

Output manipulation
Output 50



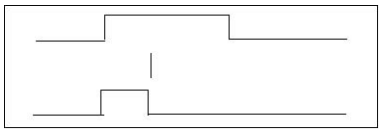
如果在END语句前脉冲被编程，程序执行期间增加。

PULSE($OUT[50],TRUE,0.8)
END statement
Program active
Output 50



如果程序执行复位或流产(复位/取消)激活脉冲输出,脉冲立即复位。

PULSE($OUT[50],TRUE,0.8)
RESET or CANCEL
Output 50



2.2.35 REPEAT...UNTIL

2.2.35.1 摘要信息

程序最少执行一次循环(不拒绝循环)。在循环结束检查中止条件。

2.2.35.2 语句

```
REPEAT
Statements
UNTIL Termination_Condition
```

自变量	类型	解释
Termination －Condition	BOOL	逻辑表达式能包含布尔变量，调用布尔函数或 布尔结果的逻辑操作

2.2.35.3 说明

REPEAT循环是依靠用户指定的条件重复。

在任何一个循环执行后，检查中止条件。语句块最少执行一次。

如果逻辑条件有**FALSE**值，语句块被重复。如果逻辑条件有**TURE**值，在条件满足后程序在下一个语句重新开始。.

2.2.35.4 范例

循环执行100次，在最后一次循环执行后R值为101。

```
R=1
REPEAT
R=R+1
UNTIL R>100
```

循环执行到$IN[1]为真时。

```
REPEAT
Statements
UNTIL $IN[1]==TRUE
```

循环执行一次，甚至中止条件在循环执行前已经满足。在循环结束前不检查中止条件。在循环执行前，R的值为102。

```
R=101
REPEAT
 R=R+1
 UNTIL R>100
```

EXIT, SWITCH, FOR, WHILE, LOOP

2.2.36 RESUME

2.2.36.1 摘要信息

子程序和中断的异常中断。

2.2.36.2 语句

```
RESUME
```

2.2.36.3 说明

在处理中断期间，RESUME语句单独执行。因此他能在中断开始模式中单独执行。所有激活的中断程序和子程序直到在当前被公告的中断中，由于RESUME流产。

在RESUME语句时被延伸：

- **变量$ADVANCE必须等于0(不预先运行)。**
- **预先运行指针必须低于被公告的中断的优先级；它最少比中断低一个优先级 （否则程序停止或重新启动）**

- **在RESUME后面的运动不能使圆形运动(圆弧)因为开始点在任何时间不同。(不同的园)。**
- **在中断程序中改变变量$BASE有这样的结果。**
- **在中断程序中， $ADVANCE分配不能使用。**

2.2.36.4 范例

机器人在编程路径上搜索。零件由连接输入15传感器而发现。在找到零件后，机器人不再继续到路径结束点但返回中断位置，拾起零件放到放下点。

```
DEF PROG( )                    ；主程序
...
INTERRUPT DECL 1 WHEN $IN[15] DO FOUND( )
...
PTP HOMEPOS
...
SEARCH( )                      ；查找路径必须在子程序被编程
LIN SETDOWN POINT
...
END

DEF SEARCH( )                  ；子程序查找零件
LIN START_POINT C_DIS
LIN TARGET_POINT
$ADVANCE=0                     ；不允许提前运行
END

DEF FOUND( )                   ；中断程序
INTERRUPT OFF                  ；因此中断程序没有执行两次
BRAKE
LIN POS_INT                    ；返回中断发生的点
...                            ；拾起零件
RESUME                         ；搜索路径异常中断
END
```

INTERRUPTDECL，INTERRUPT，BRAKE，RETURN

2.2.37 RETURN

2.2.37.1 摘要信息

从函数和子程序返回

2.2.37.2 语句

函数

```
RETURN Function_Value
```

子程序

```
RETURN
```

自变量	类型	解释
Function_ Value	数据类型必须符合函数类型	函数值是函数退出时传送的数值。.

2.2.37.3 说明

在函数和子程序中使用**RETURN**语句。它结束函数和子程序的执行并导致返回调用模式。

函数中的RETURN语句

函数的执行必须由**RETURN**语句包含的函数值来决定。函数值可以作为常数，变量或表达式来定义。数据类型必须与**DEFFCT**声明中定义的函数数据类型一致。

在子程序中的RETURN语句

RETURN语句可以在程序中仅由关键字**RETURN**组成。它不包含表达式。函数值不能被转移。

2.2.37.4 范例

从函数调用模式返回并传送函数值0

```
RETURN 0
```

从函数调用模式返回并传送函数值(X*3.1415)/360。

```
RETURN (X*3.1415)/360
```

从函数调用模式返回并传送函数值X。

```
DEFFCT INT X( )
        INT XRET
        XRET=10
        RETURN XRET
ENDFCT
```

从子程序调用模式返回。

```
DEF PROG_2( )
        Declarations
        Statements
RETURN
END
```

DEFFCT，DEF

2.2.38 SIGNAL

2.2.38.1 摘要信息

声明输入输出信号的信号名。

中断外围系统传送。

2.2.38.2 语句

声明输入输出信号的信号名：

```
SIGNAL Signal_Name Interface_Name ⟨TO Interface_Name
```

中断外围系统传送：

```
SIGNAL System_Signal_Name FALSE
```

自变量	类型	解释
Signal_Name		任何符号名
Interface_Name		预先确定的信号变量类型。可选择下列类型： **$IN[***No***]**二进制输入 **$OUT[***No***]**二进制输出 **$DIGIN[***No***]**数字输入 **$ANIN[***No***]**模拟输入 **$ANOUT[***No***]**模拟输出 不涉及控制器的输入输出
System_ Signal_Name		预先确定的二进制系统输出名。例如.$T1.
FALSE		系统状况不传送到外围。. 不能用选项TRUE.

2.2.38.3 说明

机器人控制器有两种界面

1.简单程序界面(信号)
2.逻辑界面(通道)

所有界面使用符号名地址。***Interface_Names***(符号名)理论上由**SIGNAL**指令预先确定的信号变量组成。**SIGNAL**声明必须出现在声明区。输出出现在在几个**SIGNAL**语句中。信号的个数符合控制器的输入和输出个数。作为能被发现的预先定义的变量名，在二进制和数字式的输入或输出由更多的差别。在二进制的输入或输出情况下，输入或输出被单独编址。在数字式信号的情况下，几个输入或输出被组合。

几个连续的二进制输入或输出能被一个数字式的输入或输出使用**SIGNAL**指令和**TO**选项组合。用这个方法组合的信号能随意是十进制名，十六制名（前缀**H**）或位的形式名（前缀**B**）的编址。它也能被布尔操作处理。最多32个二进制信号能与一个数字式信号组合。联合的信号名与预先确定的信号都名必须是二进制的输入或输出并且在它们的索引中连续的上升次序。最多32个输入或输出能被组合。

机器人控制器能装配提供32个输入和32个输出的输入输出模块。输出1－28额定负荷100mA，输出29－32额定负荷2A。不使用的输出能被作为标记使用。在二进制输入的情况下，信号名作为**BOOL**类型的开始在内部公告。

2.2.38.4 范例

二进制输出$OUT[7]被分配位符号名开关。开关($OUT[7])设置。

SIGNAL SWITCH $OUT[7]

Switch = TRUE

二进制输入$IN[1]到$IN[8]在符号名INWORD中与一个数字式的输入组合。

SIGNAL INWORD $IN[1] TO $IN[8]

二进制输出$OUT[1]到$OUT[8]在符号名OUTWORD中与一个数字式的输出组合。输出$OUT[3]，$OUT[4],$OUT[5]和$OUT[7]由数字式的$OUTWORD.设置。

SIGNAL OUTWORD $ OUT[1] TO $ OUT[8]

OUTWORD = ‘B01011100’

ANIN，ANOUT，DIGIN，CHANNEL

2.2.39 SREAD

2.2.39.1 摘要信息

“SREAD”语句中止数据组（文本行）进入它们的组成部分。

2.2.39.2 语句

```
SREAD (String1, State, Offset, Format, String2, Value, Var)
```

自变量	类型	解释
String1	CHAR[ ]	从字符列表读取要处理传送的字符串2.
State	STATE_T	这是返回的核心系统用户能求值的信息。 **STATE.MSG_NO**如果在命令执行期间发生错误，那么这个变量包含错误数字。 **CMD_OK**命令成功执行。 **CMD_ABORT**命令没有成功执行。 **FMT_ERRI**错误的格式或没有相应的变量。 **HITS**正确的数字写入格式。 **LENGTH**在首次格式中出现“的%s”格式长度。
Offset	INT	指定复制的字符串1替换字符串2。
Format	CHAR[ ]	变量“Format”包含产生的格式文本。
String2	CHAR[ ]	复制串1进入字符数组。
VALUE	INTREALBOOL	粘贴的数据使用指定的格式从串1进入这个变量。 布尔值输出0或1。ENUM作为数字值。
Var		相应“Format”.变量

2.2.39.3 说明

“SREAD”命令为了处理字符串而使用。不同于**“CREAD”**，不从打开的通道读取而是从变量读取。

转换指定变量的格式有下列结构：

%FWGU

F 格式符+,--,#,etc.(可选)。

W 宽度，指定输出最大字节数。(可选)

G 精度，它的意义依靠转换的字符。’.’or’，*’，或’integer’能被使用(可选)。

U 允许转换的字符：d,e,f,g,i,s,x和%。系统不能区别大小写字母。

由输入的宽度你能指定值有多少字节被压缩或延伸。**REAL**值除外。

在压缩值时，忽视值的高等级的字节；值被延伸时，在结尾增加零字节。 (little end ian format).

如果没有指定字符宽度，固有的输出表示：**INTEGER**，**REAL**和**ENUM** 4位，**BOOL**和**CHAR** 1位.

错误的格式能由值的HITS发现。(看下面)

在运行时选择的类型和值：

Format Variable	**%d %I %x**	**%f %e %g**	**%s**	**%c**	(3) **%1.** <WDH> r	(3) **%2.** <WDH> r	(3) **%4.** <WDH> r	(3) **%.** <WDH> r
(Signal)INT	**X**			**X**				
INTarray					**X**	**X**	**X**	**X**
REAL	**X**	**X**						
REALarray							**X**	**X**
(Signal)BOOL(1)	**X**			**X**				
BOOLarray					**X**	**X**	**X**	**X**
ENUM(2)	**X**			**X**				
ENUMarray					**X**	**X**	**X**	**X**
CHAR	**X**			**X**				
CHARarray			**X**		**X**			**X**

任何不等于0的值表示真

- 系统检查**ENUM**值是否被允许。如果不允许，读失败。**ENUM**开始为1。
- 如果没有格式要求的足够数据，不进行读取而且SREAD语句失败。被忽视的数据仍然为读取准备。
- 只有一些字节到了能进入变量的时候读入。等待到读入准备好。如果实际的排列足够大但可用的字节数不是排列元素的若干倍，多余的字节在后面格式或下一个SREAD语句左边读取。

2.2.39.4 范例

读入满足变量HUGO格式的字符。

```
INTOFFSET
    DECLSTATE_TSTATE
    DECLCHARHUGO[20]
OFFSET=0
    HUGO[ ]=”1234567890”
    SREAD(HUGO[ ],STATE,OFFSET,%01d%02d,VAR1,VAR2)

;结果:VAR1=1;VAR2=23
```

当使用“SREAD”读入时，它必须定义“Format”。

在例子中符合: %01d 在这里读取的字符数是一个，因此在VAR1中第一个在HUGO中的数字被读取。例如：1。

%02d 在这里读取的字符数是两个，因此在VAR2中第二个和第三个在HUGO中的数字被读取。例如：2和3。

2.2.40 STRUC

2.2.40.1 摘要信息

声明结构数据类型.

2.2.40.2 语句

```
⟨GLOBAL  STRUC Structure_Type_Name
    Data_Type1 Component_Name1 ⟨⟨,..., Component_NameM
    ,...,
    Data_TypeN Component_NameN ⟨,..., Component_NameZ
```

自变量	类型	解释
GLOBAL		关键字**GLOBAL**用于识别，包括外部程序和 可在数据类表中使用的结构。
Structure_ Type_Name		结构数据类型的名字
Data_Type	任何数 据类型	结构类型可以由任何数据类型组成。 结构类型也作为结构类型的组成： 这是调用嵌套结构类型。如果数组的类型是**CHAR**或一维的， 那么数字能被用于组成结构类型。在这种情况下，数组限制 下面方括号中数组的名字定义结构类型。
Component_ Name		结构类型的单独元素由结构元件调用。 数据类型和名字定义各自结构元件

在声明结构数据类型中，关键字GLOBAL 可以仅在数据列表中使用。

2.2.40.3 说明

结构类型是由几个同样或不同的数据类型组成的复合数据类型。**AXIS**，**FRAME**，**POS**，**E6POS**和**E6AXIS**是重要的预先确定的结构类型。结构类型可以有由用户定义的这些名字。

作为用户，你能自由的使用结构类型声明**STRUC**定义更多的结构类型。**STRUC**为了自由定义结构类型定义必须发生在声明这个类型的变量前。必须遵守下面次序：首次**STRUC**定义，那么声明变量。

应用于：

在数据列表定位模式前，预先确定的数据列表$CONFIG 被定位。

数据列表

访问结构变量的成分

结构变量的部分能被单独处理。为了识别它们由结构部分的名字的更多变量名跟随。为了能在嵌套结构的情况下访问内在结构部分，一串变量名，外部结构部分和内部结构部分共同的名子，依靠句号辨别它们。

分配值到结构变量

数值能够利用被分配的值单独的分配到各自的变量结构组成部分。值使用集合同时分配到几个或所有结构变量部分。当在数据列表中声明结构变量时，你能作为默认初始值分配集合。

结构类型名能结束in_TYPE，要把他们从变量命中区别出来

2.2.40.4 范例

声明由CURRENT,VOLTAGEandFEED的数据类REAL组成的W1_TYPE结构类型。.

```
STRUC W1_TYPE REAL CURRENT, VOLTAGE, FEED
```

由CURRENT的数据类型REAL组成和排列组成的TEXT[80]数据类型CHAR声明的W2_TYPE结构类型。

```
STRUC W2_TYPE REAL CURRENT, CHAR TEXT[80]
```

下列信息传送到子程序的变量用于圆弧焊接：

- 走丝速度
- 特征
- 在模拟中有没有圆弧

```
DEF PROG( )
STRUC W_TYPE REAL WIRE,INT CHARAC,BOOL ARC
DECL W_TYPE W_PARAMETER
W_PARAMETER.WIRE=10.2   ；由单独的初始值组成
W_PARAMETER.CHARAC=60
W_PARAMETER.ARC=TRUE
W_UP(W_PARAMETER)       ；调用焊接子程序...
                        ；借助于集合设定初值
W_PARAMETER={WIRE 7.3, CHARAC 50, ARC TRUE}
W_UP(W_PARAMETER)   ；另一个调用焊接子程序...
END
```

ENUM, DECL

2.2.41 SWITCH...CASE...ENDSWITCH

2.2.41.1 摘要信息

在几个语句分支中选择。

2.2.41.2 语句

```
SWITCH Selection_Criterion
CASE Block_Identifier1 〈,Block_Identifier2,...
Statements
...
〈CASE Block_IdentifierN 〈,Block_IdentifierM,...
Statements
〈DEFAULT
Default_Statements
ENDSWITCH
```

自变量	类型	解释
Selection_ Criterion	INT，CHAR, 列举类型	选择标准能被变量，函数调用或指定表达似的数据类型。.
Block _Identifier	INT，CHAR, 列举类型	与CASE块有关的标识符可以是整数字符或常数表达式。 常数的数据类型必须与选择的标准一致。这儿最少存在一个块标识符。 你可以像指定块标识符一样指定你想要的一个程序分支。 如果同一个块标识符被重复使用，仅有第一个分支持有标识符。保留程序分支是被忽略。几个块标识符由逗号从其他标识符分开。

2.2.41.3 说明

SWITCH语句是为了选择程序中的变量。选择标准由前面**SWIT**CH语句确定的值分配。如果选择符合块标识符，那么执行相应的分支后程序直接跳跃到**ENDSWITCH**语句。如果没有块标识符符合选择标准，那么如果有**DEFAUL**语句块的话执行；否则程序在语句**ENDSWITCH**后重新开始。

几个块标识符能分配在一个程序分支中。另一方面，它几次使用一个块标识符是不明智的，作为首个分支有相应的经过考虑的标识符。

数据类型的选择标准和块标识符必须符合。**SWITCH**语句必须最少包含一个**CASE**语句；它必须确保没有空白行或注释出现在**SWITC**指令和第一个**CASE**语句之间。

DEFAULT语句可以被忽略。在**SWITCH**语句中默认的语句可能只出现一次。

SWITCH语句不能使用**EXIT**语句提前的退出。

2.2.41.4 范例

选择标准和整数类型的块标识符。DEFAULT语句在这儿用于输出错误信息。

```
SWITCH VERSION
CASE 1
        SP_1( )     ；调用子程序SP_1
CASE 2,3
        SP_2( )     ；调用子程序SP_2
        SP_3( )     ；调用子程序SP_3
        SP_3A( )    ；调用子程序SP_3a
DEFAULT
        ERROR_SP( )；调用子程序ERROR_SP
ENDSWITCH
```

选择标准和字符类型的块标识符。语句SP_5( )从不执行，因为块标识符“JOHN”出现了两次。

```
 SWITCH NAME
 CASE ”ALFRED”
                                SP_1( )；调用子程序SP_1
CASE ”BERT”,”JOHN”
                                SP_2( )；调用子程序SP_2
                                SP_3( )；调用子程序SP_3
CASE ”JOHN” SP_5( );调用子程序SP_5ENDSWITCH
```

EXIT，FOR，REPEAT，WHILE，LOOP

2.2.42 SWRITE

2.2.42.1 摘要信息

“SWRITE”语句使得它能组合几个数据到数据组。

2.2.42.2 语句

```
SWRITE (String1, State, OFFSET, Format, String2, VALUE)
```

自变量	类型	解释
String1	CHAR[ ]	操作串2写入到字符数组中。
State	STATE_T	这结构返回的关于用户估计核心系统状况得信息。 **STATE.MSG_NO**如果在命令执行期间出现错误， 那么这个变量包含错误数字。 **CMD_OK** 命令成功执行。 **CMD_ABORT** 命令没有成功执行。 状况变量： **HITS** 恰当的数字写入格式。
OFFSET	INT	复制字符串2到串1的指定位置。
Format	CHAR [ ]	变量“Format”包含产生的文本格式。
String2	CHAR [ ]	这个字符组被拷贝到字符组行1。串2可以包含 粘贴在这个指定格式位置中的满足变量“VALUE”的字符格式.
VALUE	INT REAL BOOL	满足在串2中有指定格式粘贴的变量。 布尔值作为0或1输出。ENUM值作为数字。.

2.2.42.3 说明

“SWRITE”命令用以处理字符行。不同于**“CWRITE”**，数据不是写在打开的通道而是变量。

变量格式的转换规格有下列结构：

%FWGU

下列指定的应用：

F 格式字符+,--,#,etc.(可选).

W 宽度，指定用于输出的最大字节数 (可选)。

G 精度，它的意义使依靠转化字符，使用’.’ or ’.*’ or ’.integer’ (可选).

U 可允许的转换字符： c, d, e, f, g, i, s, x 和 %.
系统不能区别字符的大小写。.

由于输入的宽度，你能指定多少字节被延长或压缩。在这儿**REAL**值除外。

当压缩一个值，高位字节被忽略；当值被延长时，在结尾增加零字节(little endian format)。

如果没有指定宽度，固有的输出表示：**INTEGER**，**REAL**和**ENUM** 4字节，**BOOL**和**CHAR** 1字节。

错误的格式可以由HITS的值推断。(看下面)。

Format Variable	**% d %I %x**	**%f %e %g**	**%s**	**%c**	**%1.** 《WDH》 r	**%2.** 《WDH》 r	**%4.** 《WDH》 r	**%.** 《WDH》 r
(Signal) INT	X	X						
INT array					X	X	X	X
REAL		X						
REAL array							X	X
(Signal) BOOL	X							
BOOL array					X	X	X	X
ENUM	X							
ENUM array					X	X	X	X
CHAR	X			X				
CHAR array			X		X			X

“SWRITE”语句能在程序中用于控制或触发前面的运行停止。

2.2.42.4 范例

复制变量HUGO的内容到变量BERTA

```
INTOFFSET
DECLSTATE_TSTATE
DECLCHARHUGO[20]
DECLCHARBERTA[20]

OFFSET=0
HUGO[ ]="TEST"
BERTA[ ]=""
SWRITE(BERTA[ ],STATE,OFFSET,HUGO[ ])
                    ;结果:BERTA[ ]="TEST"
                    ; 由参数调用作为"OFFSET的变量"，变量现在为值4
                    ;重复相同的命令
        SWRITE(BERTA[ ],STATE,OFFSET,HUGO[ ])
                    ;结果:BERTA[ ]="TESTTEST"
        OFFSET=OFFSET+1
        SWRITE(BERTA[ ],STATE,OFFSET,HUGO[ ])
                    ;结果:BERTA[ ]="TESTTESTTEST"
```

利用格式化字符

```
        INTOFFSET
        INTNO
        DECLSTATE_TSTAT
        DECLCHARHUGO[20]
        DECLCHARBERTA[20]


        NO=1
        OFFSET=0
        HUGO[ ]="TEST%d"
        BERTA[ ]=""
        SWRITE(BERTA[ ],STATE,OFFSET,HUGO[ ],NO)
                    ;结果:BERTA[ ]="TEST1"

        OFFSET=OFFSET+1
        NO=22
        SWRITE(BERTA[ ];STATE,OFFSET,HUGO[ ],NO)
                    ;结果:BERTA[ ]="TEST1TEST22"
```

2.2.43 TRIGGER WHEN DISTANCE...DO

2.2.43.1 摘要信息

由轨迹触发相关机器人的开关相应的动作。

2.2.43.2 语句

```
TRIGGER    WHEN DISTANCE=Distance DELAY=Time DO Statement
           (PRIO=Priority
```

自变量	类型	解释
Distance	INT	变量或常量指定开关操作的发生： **DISTANCE=0** 在开 **DISTANCE=1** 在关 仅有这两个值可以分配！
Time	INT	变量或常量能延迟或超前使用开关操作：如果值是： 肯定的　语句执行延迟 否定的　语句执行超前. 单位是毫秒。
Statement		指令能是 分配值到变量或PULSE语句或调用的子程序 子程序像执行中断指令一样被执行。 想得到的优先权必须依靠操作PRIO指定。
Priority	INT	变量或常量指定中断的优先权触发调用子程序的 语句被分配优先权。优先权1－39级和81－128级 是可用到的。优先权40－80级由系统保留而且如 果输入优先权－1值系统会自动分配。1级中断有最高 优先权。

2.2.43.3 说明

TRIGGER语句能被用于子程序的执行或距离规标准下分配一个数值到机器人动作所依靠的变量。

指定在运动块的开始或依靠限定距离的结束的开关动作定义轨迹触发：

在单独的块中， **DISTANCE=0** 指出开始点
DISTANCE=1 目标点

在近似定位的情况下，**DISTANCE=1**意味着目标点在其后的近似定位的中间。如果以前的块已经是近似定位块，**DISTANCE=0**意味着目标点在以前的近似定位的目标点

由在指定的一段时间中使用的**DELAY**选项可能延迟或提前语句的执行。开关点能被延迟或超前，只要它仍然保留在有关的块中。因此在单独块或**DISTANCE=1**的情况下，开关点不能被移动穿过目标点和由正的**DELAY**值指定的更远的轨迹。指定负值**DELAY**，保证开关操作在到达目标点前已经发生。但开关点不能被提前到比开始点运动更远的运动。

2.2.43.4 范例

开关操作在下一个动作前130毫秒；设置信号

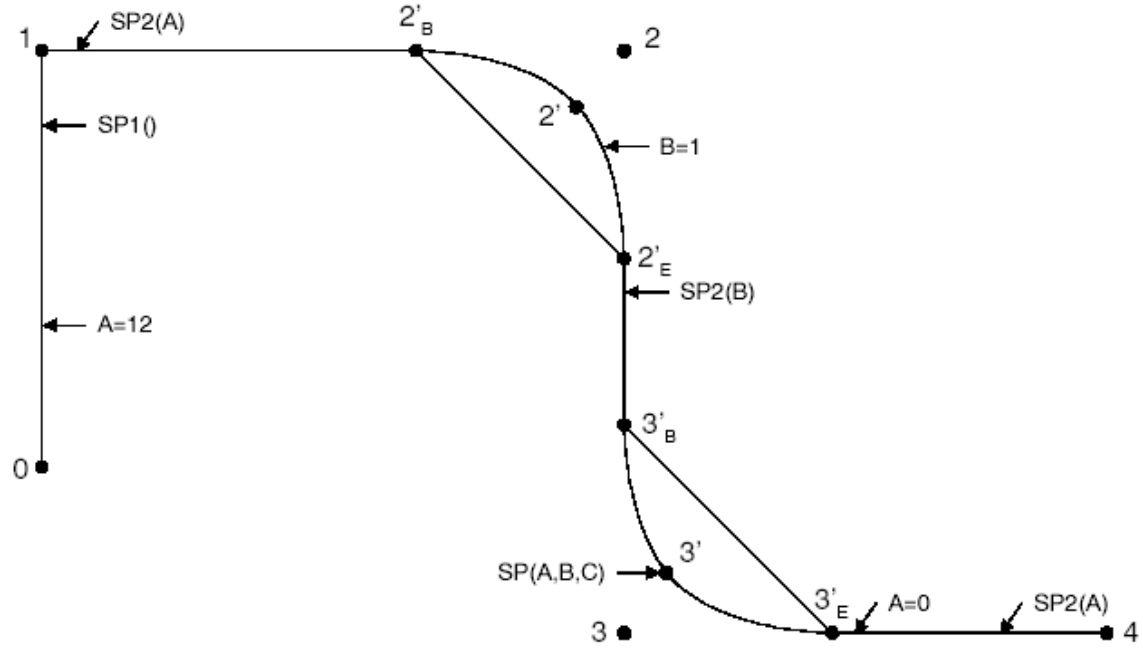
TRIGGER WHEN DISTANCE=0 DELAY=130 DO $OUT[8]=TRUE

开关在下一个动作结束时;调用子程序优先级为5.

TRIGGER WHEN DISTANCE=1 DO QUOTIENT(DIVIDEND,DIVISOR)PRIO=5

使用指定的动作次序和DELAY选项定义开关的范围。

```
DEF PROG( )
...
PTP POINT0
TRIGGER WHEN DISTANCE=0 DELAY=40 DO
A=12
                              ;开关范围:0--1
 TRIGGER WHEN DISTANCE=1 DELAY=-20 DO SP1( ) PRIO=10
                              ;开关范围:0--1
LIN POINT1
TRIGGER WHEN DISTANCE=0 DELAY=10 DO SP2(A) PRIO=5
                             ;开关范围:1--2'B
TRIGGER WHEN DISTANCE=1 DELAY=15 DO B=1
                            ;开关范围:2'B--2'E
LIN POINT2 C_DIS
TRIGGER WHEN DISTANCE=0 DELAY=10 DO SP2(B) PRIO=12
                            ; 开关范围:2'E--3'B
TRIGGER WHEN DISTANCE=1 DO SP(A,B,C) PRIO=6
                            ; 开关范围:3'B--3'E
LIN POINT3 C_DIS
TRIGGER WHEN DISTANCE=0 DELAY=50 DO SP2(A) PRIO=4
                            ; 开关范围:3'E--4
TRIGGER WHEN DISTANCE=1 DELAY=-80 DO A=0
                            ; 开关范围:3'E--4
LIN POINT4
...
END
```



INTERRUPTDECL， INTERRUPT ，PULSE

2.2.44 TRIGGER WHEN PATH...DO

2.2.44.1 摘要信息

轨迹延迟触发机器人相应动作的开关。.

2.2.44.2 语句

```
TRIGGER    WHEN PATH=Distance DELAY=Time DO Statement
           〈PRIO=Priority
```

自变量	类型	解释
Distance	INT	用毫米指定相应开关动作被延迟的与下一个运动的目标点的距离。 如果被延迟的开关动作在到达目标点前，则Distance为负值。 如果Distance为正值，被延迟的开关动作在到达目标点后。 如果目标点是近似定位点，那么Distance是近似定位运动的中心点。 由于正值Distance，它可以转化开关点远到在触发后被编程的的精确点。 如果开始点是近似定位点，开关点能被替换远到近似定位范围的开始。
Time	INT	规定的“Time”用于轨迹触发的开关点的延迟或超前的 由时间的数量定义的标准。 开关点仅能被上面指定的范围替换。 单位是毫秒。

Statement		指令能分配一个值到变量或一个PULSE 语句或调用一个子程序 子程序可以像中断程序一样被执行。想得到的优先权必须由操作PRIO的方法指定。
Priority	INT	变量或常量指定中断的优先权触发调用子程序的语句被分配优先权。优先权1－39级和81－128级是可用到的。优先权40－80级由系统保留而且如果输入优先权－1值系统会自动分配。1级中断有最高优先权。

2.2.44.3 说明

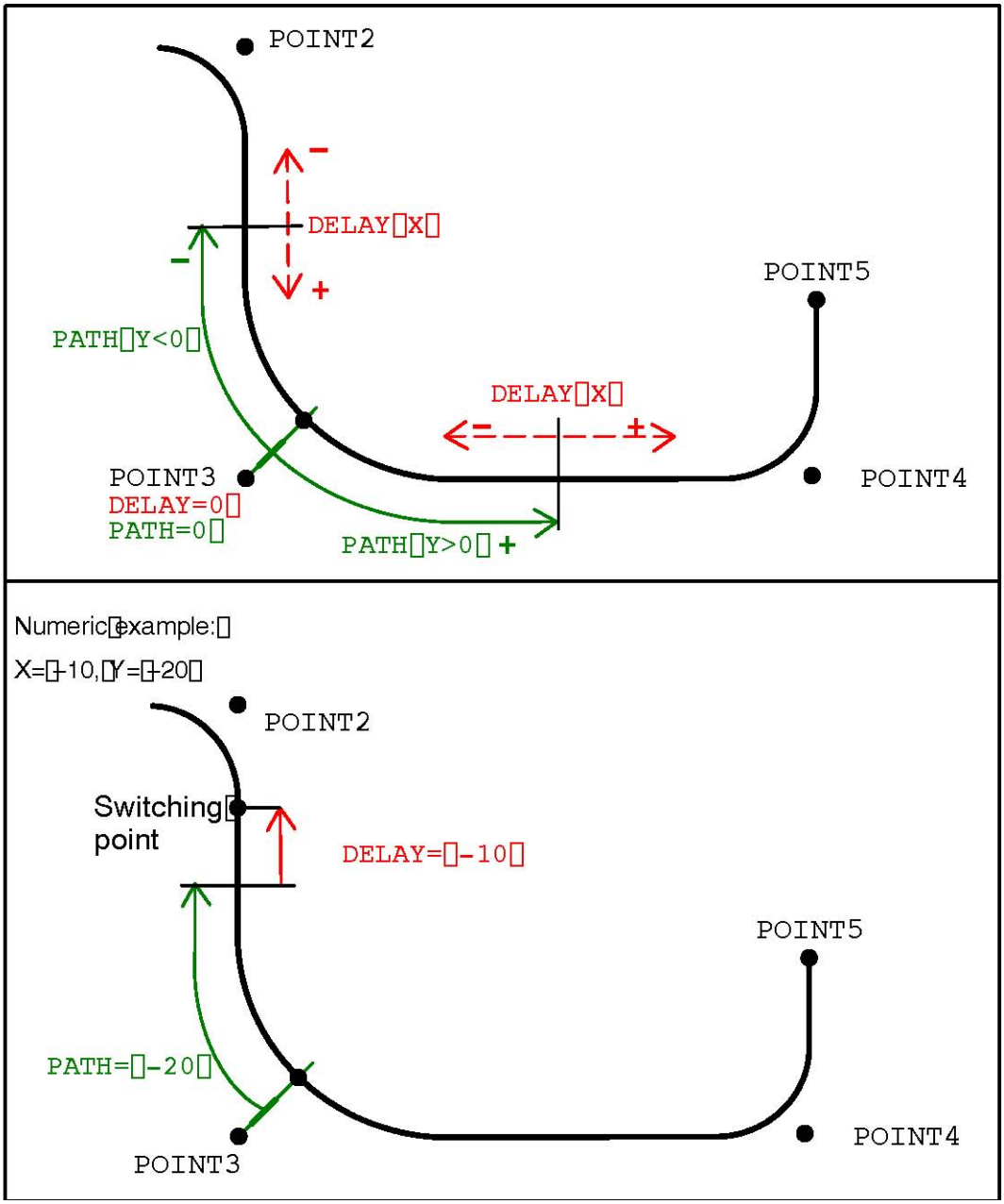
如果你使用**TRIGGER**语句，你能在任何沿轨迹触指定距离的位置触发发开关动作。使用“Trigger when Distance”语句他能再一次增加开关点的提前或延迟。

以被计算出的标准距离的轨迹为条件控制位置，由近似定位决定的运动，在台面上的显示。标准时间总是由轨迹计算出来的。开关点的范围依靠在近似定位中在台面上的显示而决定。

指令次序：

：
LIN POINT2 C_DIS
TRIGGER WHEN PATH = Y DELAY= X DO
$OUT[2]=TRUE
LIN POINT3 C_DIS
LIN POINT4 C_DIS
LIN POINT5
：

自从开关点能被编程的前的运动点替换后，所有近似定位点要直到到达下一个精确位置点，开关点可能被近似定位的开始点**POINT2**到**POINT5**替换。如果**POINT2**在这个指令中不是近似定位点，那么开关点能仅被精确位置点**POINT2**替换。



INTERRUPTDECL，INTERRUPT，PULSE

2.2.45 WAITFOR

2.2.45.1 摘要信息

等待继续的条件。

2.2.45.2 语句

```
WAIT FOR Continue_Condition
```

自变量	类型	解释
Continue_ Condition	BOOL	当程序被继续时，逻辑表达式被用于指定继续的条件。 如果逻辑表达式为TRUE，那么在调用WAIT时程序的执行没有被停止。 如果逻辑表达式为FALSE，那么程序的执行被停止，直到表达式的值为TRUE后程序才能继续。

.

2.2.45.3 说明

WAIT语句可以停止程序的执行和在指定的等待时间后继续执行程序。等待的时间长度由由编程决定。

如果由于错误的表达式永远得不到编辑器认可的TRUE值,在这种条情况下,因为程序等待不能实现的条件，程序将被永远的停止。

2.2.45.4 范例

程序执行中断，直到$IN[17]为TRUE。

WAIT FOR $IN[17]

程序执行中断，直到BIT1为FALSE。

WAIT FOR BIT1 == FALSE

WAIT SEC

2.2.46 WAIT SEC

2.2.46.1 摘要信息

等待时间

2.2.46.2 语句

```
WAIT SEC Wait_Time
```

自变量	类型	解释
Wait_Time	INT ,REAL	算术表达式被用于指定程序执行被中断的秒数。 如果值为负数，那么程序不执行中断。 很少的等待时间的精确性由插补循环的倍数指定。

2.2.46.3 说明

WAIT语句可以停止程序的执行而且可以在指定的等待时间后继续程序的执行。等待时间的长度用秒指定。

2.2.46.4 范例

中断程序执行17.156秒。

WAIT SEC 17.156

中断程序执行的时间由用秒指定的变量V_WAIT的值决定。
WAIT SEC V_WAIT

WAIT FOR

2.2.47 WHILE...END WHILE

2.2.47.1 摘要信息

程序循环；在循环开始选择中止条件(拒绝循环)。

2.2.47.2 语句

```
WHILE Repetition_Condition
Statements
ENDWHILE
```

自变量	类型	解释
Repetition_ Condition	BOOL	逻辑表达式能包含Boolean变量，调用布尔函数或逻辑操作布尔结果。例如：比较。

2.2.47.3说明

WHILE循环依靠用户指定的条件重复。如果重复条件没有预先完成，重复条件在各自的循环执行前选择。

如果逻辑条件的值为**TRUE**，那么执行语句块。例如：履行重复条件。如果逻辑表达式的值为**FALSE**，那么程序重新执行在**ENDWHILE**后的下一个指令。**Each WHILE**语句必须由**ENDWHILE**语句结束。.

2.2.47.4 范例

程序循环执行99次。W在循环结束后的值为100。

```
W=1
WHILE W<100
W=W+1
ENDWHILE
```

程序循环执行到$IN[1]的值为true。

```
WHILE $IN[1]==TRUE
    Statements
ENDWHILE
```

由于在循环执行前重复条件没有满足，循环没有执行。在退出后W的值为100。

```
W=100
WHILE W<100
W=W+1
ENDWHILE
```

EXIT, SWITCH, FOR, REPEAT, LOOP

2.3.1 VARSTATE( )

2.3.1.1 摘要信息

变量状态 VARSTATE ( ) 是“VAR_STATE”类型的返回值:

ENUM VAR_STATE DECLARED, INITIALIZED, UNKNOWN

VARSTATE的声明：
VAR_STATE VARSTATE(CHAR VAR_STR[80]:IN)

2.3.1.2 语法

```
DECL VAR_STATE Variable_Name
Variable_Name = VARSTATE(”Variable”)
```

自变量	类型	解释
Variable_ Name	**VAR_STATE**	任何变量名
Variable	任何	有状态决定的变量名

2.3.1.3 说明

函数**VARSTATE**被用于决定变量的状态。函数的返回值可以是Enum常数，DECLARED，**INITIALIZED**或**UNKNOWN**。

2.3.1.4 范例

```
DEF PROG1( )
 INT A,B
 CHAR STR[5]
 A=99
 STR[ ]="A"

 IF VARSTATE("A")==#DECLARED THEN
 ; 这个A没有被公告但已被初始化的IF条件是错误的。
 $OUT[1]=TRUE
 ENDIF
 IF VARSTATE("A")==#INITIALIZED THEN
 ; 这个IF条件是正确的.
 $OUT[2]=TRUE
 ENDIF

 IF VARSTATE("A")==#UNKNOWN THEN
 ; 这个IF条件是错误的
 $OUT[3]=TRUE
 ENDIF

 IF VARSTATE("B")==#DECLARED THEN
 ; 这个IF条件是正确的.
 $OUT[4]=TRUE
 ENDIF

 IF VARSTATE("NOTHING")==#UNKNOWN THEN
 ; 这个IF条件是正确的，设想在$CONFIG.DAT 中没有变量"NOTHING"。
 $OUT[5]=TRUE
 ENDIF
 IF VARSTATE(STR[ ])==#INITIALIZED THEN;
 ; 这个A已被初始化的IF条件是正确地。
 $OUT[6]=TRUE
 ENDIF
 END
```

Symbols

A

B

C

D

E

F

T

V

W