埃夫特智能装备股份有限公司

C30 指令手册

版本号：v0.1

目录

1 数据类型

1.1 变量声明的数据类型

1.1.1 BOOL 布尔

此类数据用于逻辑值。BOOL 值可以是 true 或 false。初始化后的默认值为 false。
例子:

```
BOOL firstRun
...
IF firstRun = false THEN
firstRun:= true ;
...
END_IF ;
```

当 firstRun 为 false 时，执行后，被置为 true。

1.1.2 DINT 双精度整数

此类数据用于整数值，可以是正数或负数。
整数值用 32 位值表示。
DINT 值可以在-2147483648 和 2147483647 之间。
初始化后的默认值为 0。
例子:

```
DINT counter
DINT diff
...
counter := counter + 1 ;
...
```

1.1.3 UDINT 无符号双精度整数

此类数据用于仅为正数的整数值。
整数值用 32 位值表示。
UDINT 值可以在 0 到 4294967295 之间。
初始化后的默认值为 0。
例子：

```
DINT counter
```

...

counter := counter + 1 ;

1.1.4 LREAL 长实数

此类数据用于具有双精度的十进制数。
该值是 64 位有符号值。
LREAL 值可以在-1.7976931348623158e + 308 和 1.7976931348623158e + 308 之间。
初始化后的默认值为 0.0。
例子:

```
LREAL width
...
width := 88.506 ;
...
```

1.1.5 VECT3 三维实向量

这种类型的数据用于表示三维向量。
数据使用 3 个 LREAL 值。
数据具有 LREAL 类型的 x，y，z 组件。
要使用 3 LREAL 值组合此类数据，可以使用函数 VECT3。
初始化后的默认值是一个向量，其中所有分量 x，y，z 都设置为 0.0。
例子:

```
VECT3 pose
...
pose := VECT3( 100, 200, 300) ;
```

1.1.6 POINTC 笛卡尔空间位姿

这种类型的数据用于表示笛卡尔点。数据具有 LREAL 类型的 x，y，z，a，b，c 分量。分量 x，y，z 表示位置，a，b，c 表示具有 Euler 约定的方向。分量 a 是指 z 轴上的方向，b 是指 y 轴上的方向，c 是指 x 轴。要使用 6 个 LREAL 值组合此类数据才可以使用 POINTC 功能。POINTC 可用于笛卡尔和关节运动。初始化后的默认值是无效点（因此，如果在移动指令中使用则发出错误），所有分量 x，y，z，a，b，c 都设置为 0.0。
例子:

```
POINTC target
...
target := POINTC( 100, 200, 300, 10, 20, 30) ;
MLIN (target, v250, fine, tool0, wobj0) ;
```

1.1.7 ROBOT 机器人

此类数据用于与系统中定义的轴组进行交互。例如，可以设置轴组的参考系。要设置此类型的数据，必须使用函数 ROBOT，该函数需要作为参数的字符串，其名称为 axis group。初始化后的默认值是无效的 ROBOT。

例子:

```
ROBOT conveyor
REFSYS cvywobj
POINTC cvyOrigin
...
conveyor := ROBOT(“conveyor”) ;
cvywobj:= SETROBOTWOBJ(conveyor, “”, cvyOrigin) ;
...
```

在这个例子中，它设置了传送带的参考系统。

完成此设置是为了在程序中指定传送带的参考系统，例如可以在跟踪应用程序中使用。

1.1.8 REFSYS 参考坐标系

这种类型的数据用于定义笛卡尔运动的坐标参考系。如果在移动指令中使用，则位置将基于指定坐标系。REFSYS 可以基于分层链，可以是固定的或可移动的。例如，可以使用需要父 REFSYS 和六个 LREAL 值的 REFSYS 函数。初始化后的默认值是 REFSYS，其中所有数据都设置为 0.0，与世界参考系统相同。

例子:

```
REFSYS wobj
POINTC p0
...
wobj:= REFSYS( wobj0, 100, 200, 50, 0, 0, 0) ;
p0 := POINTC(0, 0, 0, 0, 0, 0) ;
MLIN (p0, v500, fine, tool0, wobj) ;
```

在此示例中，机器人将移动到参考系统 wobj 的原点。

1.1.9 POINTJ 关节位置类型

这种类型的数据用于确定机器人关节的位置。数据具有 LREAL 类型的 j1，j2，j3，j4，j5，j6 组件。要设置此类数据，可以使用 6 个 LREAL 值的 POINTJ 函数。POINTJ 用于 MJOINT 指令，将机器人移动到关节位置定义的特定位置。POINTJ 不能用作笛卡儿运动的目标。

初始化后的默认值是无效点（因此，如果在移动指令中使用则发出错误），所有组件 j1，j2，j3，j4，j5，j6 都设置为 0.0。

例子:

```
POINTJ startpos
...
startpos:= POINTJ( 0, 0, 0, 0, -90, 0) ;
```

MJOINT(startpos, v500, fine, tool0) ;

1.1.10 TRIGGER 触发类型

TRIGGER 数据类型用于存储与移动指令相关的事件中涉及的数据。通过指令，TRIGSET 可以将数据存储在 TRIGGER 变量中，指令 TRIGON 可以在执行运动指令之前激活触发器。初始化后的默认值是无效触发器。

例子:

```
TRIGGER trig
...
trig := TRIGSET when Distance is 20 do (io.output[5] := true) ;
...
TRIGON (trig) ;
MLIN (target, v500, fine, tool1, wobj1) ;
```

在此示例中，当机器人在目标之前 20 mm 时，io.output [5]设置为 true。

1.1.11 STRING 字符串

此类数据用于存储字符串。STRING 最多可包含 128 个字符。初始化后的默认值为空字符串。

例子:

```
STRING str
...
str := “Hello world” ;
MESSAGE (“%1”, str) ;
```

在用户日志中包含内容为“Hello world”的字符串 str。

1.1.12 SPEED 速度类型

SPEED 数据类型用于指定移动指令中的目标速度。它包含以 mm / s 表示的切向速度和以度/ s 表示的定向速度。要将数据存储在 SPEED 类型的变量中，可以使用 SSPEED 函数。初始化后的默认值为 SPEED，所有参数均设置为 0.0。

例子:

```
SPEED vel
...
vel := SSPEED( 250, 5) ;
MLIN (target, vel, fine, tool0) ;
```

在此示例中，机器人将以 250 mm / s 和 5 度/ s 的目标速度移动。

例子:

```
SPEED vel
...
vel := v250 ;
```

...

MLIN (target1, vel, fine, tool0) ;

MLIN (target2, vel, fine, tool0) ;

MLIN (target3, vel, fine, tool0) ;

此示例显示如何使用变量 vel 中定义的预定义速度设置为一组运动定义目标速度。

1.1.13 ZONE 过渡混成类型

ZONE 数据类型用于指定两个连续移动指令之间的混合参数。它包含以 mm 为单位的线性距离和以度为单位的重定向角距离。要设置此类型的数据，可以使用 SZONE 功能。初始化后的默认值为 ZONE，所有参数均设置为 0.0。

例子:

ZONE blend

...

blend := SZONE(100, 5) ;

...

MLIN (target1, v250, blend, tool0) ;

MLIN (target2, v250, fine, tool0) ;

在此示例中，target2 的移动在机器人到达目标 1 之前从 100 mm 开始。

1.1.14 TOOL 工具类型

此类数据用于描述工具的参数。TOOL 数据类型用于移动指令。要设置此类数据，可以使用 STOOL 函数。初始化后的默认值是 TOOL，所有参数都设置为 0.0。

例子:

TOOL gun

...

gun := STOOL(0, 0, 100, 0, 0, 0) ;

...

MLIN (target, v250, fine, tool0) ;

MLIN (target, v250, fine, gun) ;

此示例可用于使用相同目标和两个不同工具检查机器人位置的不同。

1.1.15 INTR 中断处理类型

这种类型的数据用于中断处理。要将此类型的变量与特定中断例程绑定，必须使用 INTRSET 指令。可以使用指令 INTRCOND 或 INTRERRNO 指定导致中断的条件。请注意，此类数据必须在程序执行时只设置一次，否则将发出错误。有关其他信息，请参阅以 INTR 开头的指令并指导“如何使用中断”。初始化后的默认值是无效的 INTR。

例子:

INTR intr1

BOOL firstRun

```
...
(* set interrupt only once *)
IF NOT firstRun THEN
firstRun := true ;
INTRSET (intr1, intHnd()) ;
INTRCOND (intr1, io.input[8]) ;
END_IF ;
...
```

此示例显示如何绑定中断以及如何设置触发该中断的条件。

1.1.16 CLOCK 时间测量时钟类型

CLOCK 数据类型用于时间测量。时间测量以秒表示。无法直接设置或读取此类数据。可以使用 CLOCKRESET 指令重置时间测量。要开始测量必须使用指令 CLOCKSTART 和 CLOCKSTOP 来停止测量。对于读取必须使用 CLOCKREAD 函数返回 LREAL 值。初始化后的默认值是 0.0 秒的测量值。

例子:

```
CLOCK clk1
...
CLOCKRESET (clk1) ;
CLOCKSTART (clk1) ;
...
CLOCKSTOP (clk1) ;
MESSAGE ("Time elapsed %1", CLOCKREAD(clk1)) ;
```

此示例显示如何测量执行指令期间所经过的时间。

1.1.17 TRACKING

TRACKING 数据类型用于存储跟踪应用程序的数据。要设置此类数据必须使用函数 TRACKING。有关参数的说明，请参阅跟踪应用说明文档。初始化后的默认值是无效的跟踪。

例子:

```
TRACKING cvy
REFSYS wobj
REFSYS rsPhoto
...
cvy := TRACKING(1000, 700, 100, 1000, Linear, 0.1) ;
...
wobj := GETTRKWOBJ(rsPhoto, cvy) ;
...
MLIN (target, v500, fine, tool1, wobj) ;
...
```

此示例显示如何为跟踪应用程序设置跟踪参数。

1.2 来源于机器人控制环境中可供使用的变量数据类型

1.2.1 POINT_L 库点

此类数据用于访问系统中加载的库点。数据可用于移动指令。
例子:

```
POINT_L p0
...
MLIN (p0, v500, fine, tool0) ;
```

在此示例中，机器人将移动到系统中加载的库中定义的点 p0。

1.2.2 PATH_L 库路径

此类数据用于访问系统中加载的库路径。数据可与指令 EPATH 一起使用。
例子:

```
PATH_L path0
...
EPATH (path0, v500, fine, tool0) ;
```

在此示例中，机器人将执行在系统中加载的库中定义的轨迹 path0。

2 变量

2.1 变量的行为

2.1.1 VAR

这是变量的默认行为。可以在程序中设置变量，并在重新启动程序时丢失其值。

2.1.2 CONST

具有此行为的变量无法在程序中更改，必须使用 Init 值进行设置。

2.1.3 RETAIN

当程序从内存中卸载时重新启动并存储程序时，将保留变量的值。对于 LOCAL 和 TASK 变量，该值保存在程序中。仅当 Usage 字段为 Module 时，才能选择此行为。

2.2 变量的可见性

2.2.1 Routine 局部

此类型的变量只能在其定义的程序和子程序中可见和使用，不能被其他程序和子程序可见和使用。可以在多个子程序中定义同一个变量名的变量，子程序中定义的变量只能被自己使用。

2.2.2 Routine 输入

输入变量用于定义子程序的输入参数。输入变量的使用类似局部变量但是其初始值来源于调用的程序。输入变量定义顺序与调用指令传递参数顺序一致。主程序中不能够使用此类变量。

2.2.3 Routine 输出

输出变量用于定义子程序的输出参数。输入变量的使用类似局部变量但是其初始值来源于调用的程序。输出变量定义顺序应与调用指令设置的参数顺序一致。主程序中不能够使用此类变量。

2.2.4 Module 局部

此类变量可以被所有程序和子程序使用。Module 局部变量可以在一个子程序中设置，然后通过另外一个子程序读取。不能使用相同的名字定义多个 Module 局部变量。这些变量对于其他 Module 不可见。

2.2.5 Module 公共

同 Module 局部相似，但是这个变量可以从其他 Module 中看到。在其他模块中，可以在模块名称前面使用这种变量（例如 moduleName.variableName）。

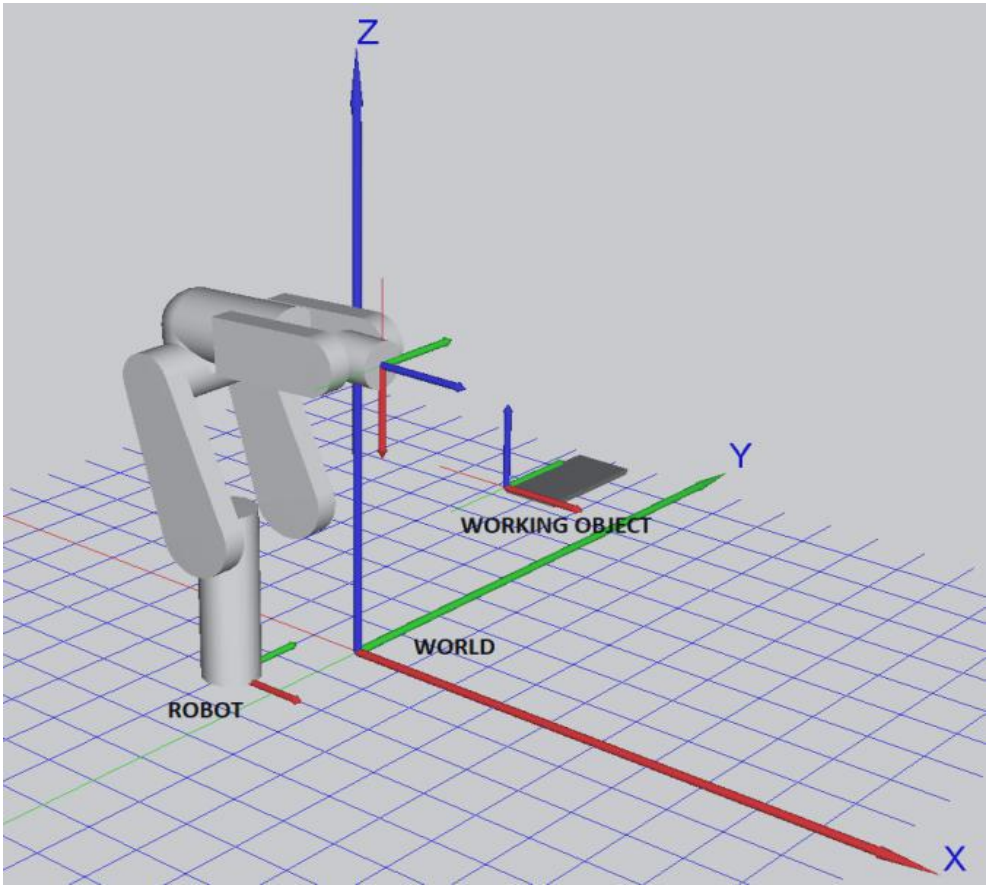
2.2.6 Task 任务

类似 Module 公共变量，在其他模块中，可以使用这种变量，而无需使用模块名称（例如 variableName）。

2.2.7 Global

这种变量对于所有 TASK 系统都是通用的。在不同的 TASK 之间共享数据很有用。如果对同一 GLOBAL 有不同的定义，则报告错误。使用 GLOBAL 变量时，Module 名称前面没有该变量

2.2.8 参考坐标系



参考坐标系用于简化程序位移。如果改变机器人的位置和/或工件的位置，则可以仅改变参考坐标系的值来重复使用程序。

2.2.9 世界坐标系

世界坐标系是一个固定坐标系，用于引用系统中使用的所有其他参考。

2.2.10 机器人坐标系

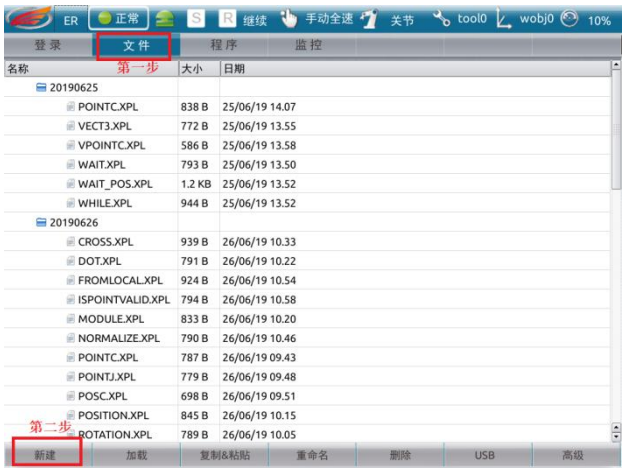
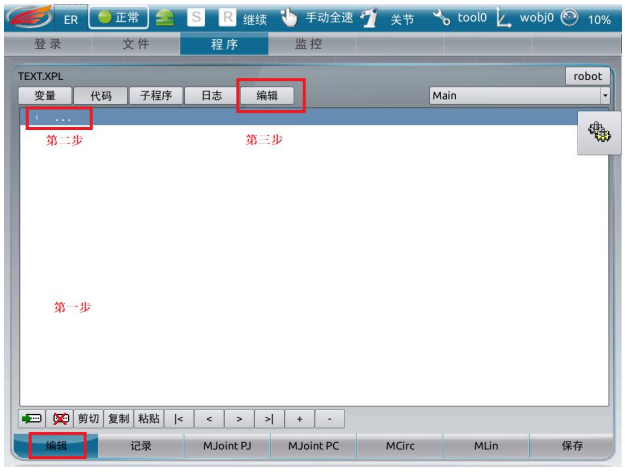
机器人坐标系识别机器人在系统中的位置。通常，机器人坐标系设置与世界参考相同。

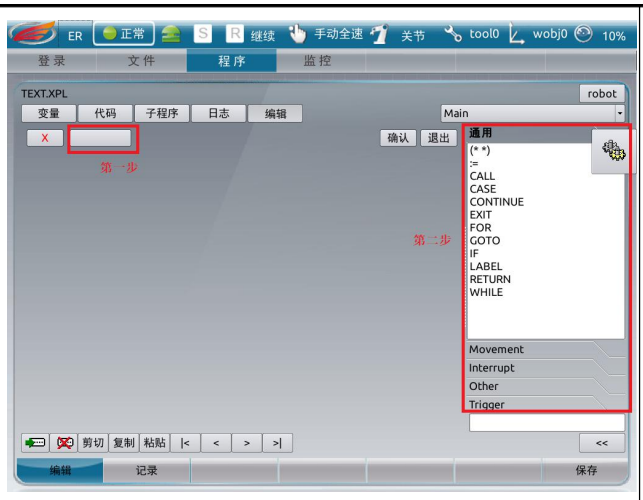
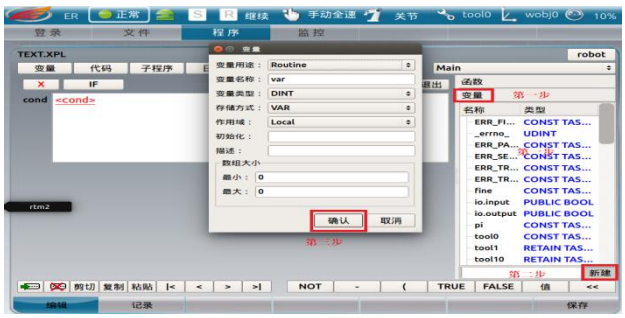
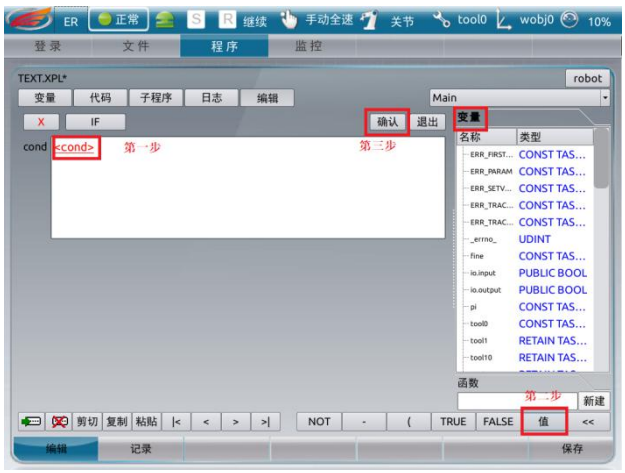
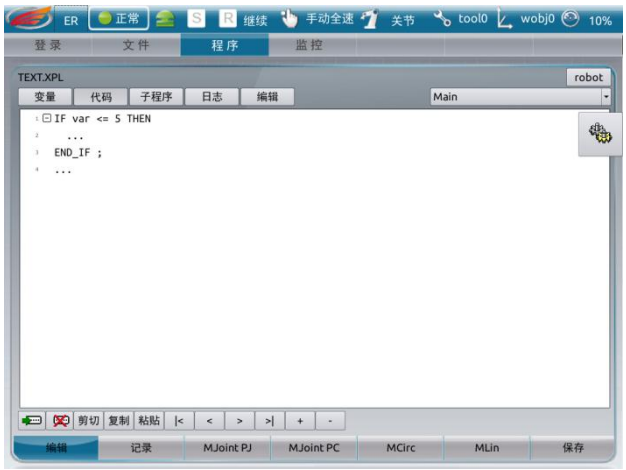
2.2.11 工作对象坐标系

工作对象坐标系识别工件的位置。

3 RPL 指令

3.1 添加指令操作

步骤	图片	描述
1. 登录。		① 点击标题栏“登录”后输入密码(999999)。 ② 点击“登录”。
2. 创建文件夹或文件。		① 在界面中选择文件选项。 ② 点击新建，新建文件夹或文件。 ③ 确定文件夹或文件名称。 ④ 如需加载文件，选中文件后，点击下方加载即可。
3 .编写程序准备。		① 点击“编辑”进入编辑功能。 ② 选中需要编辑的行。 ③ 点击上方编辑。

4. 选择函数或方法以及创建变量。	 	① 选择白框后即可在右侧选择函数或方法。 ② 选中变量点击“新建”即可创建各个类型的变量，最后点击“确认”即可。
5. 程序确认。	 	① 选中需要填写的参数(这里的函数选择为 IF)。 ② 在变量范围内选择刚刚创建的变量。 ③ 对变量进行相应的设置和应用。 ④ 点击“确认”，创建出完整代码。（下图）。

3.2指令详解

3.2.1 (* *) 注释指令

表示在示教器界面的注释。使用此指令，可以在代码中输入一个注释。
例子:

```
(* this is an annotation *)
```

3.2.2 IF 语句

表示条件判断，如果 IF 条件为真，Then 后面的指令则会执行。
例子：

```
IF i = 2 THEN
    CONTINUE ;
END_IF ;
```

3.2.3 := 赋值指令

通过此指令，可以为变量赋值。
格式：变量 := 表达式;
例子：

```
i := 5 ;
```

该例中，变量 i 被赋值为 5。
注意：此指令不能用于将一个数组直接赋值给另外一个数组。如果想要把一个数组的值复制到另外一个数组，必须针对数组中的每个元素使用此指令。

3.2.4 ALIAS 变量别名

通过一个变量绑定另一个变量。
例子：

```
ALIAS (out_1, out_2) ;
```

通过 ALISA 语句使 out_1 变量可以当作 out_2 使用。

3.2.5 CALL 调用

用于执行用户定义的子程序。

3.2.6 CLEARMOVE

此指令可打断所有等待运动指令，在调用此指令前需要保证机器人处于暂停或运动停止状态。如果机器人在运动中执行 CLEARNOVE 指令将会报错。

例子：

```
STOPMOVE () ;
CLEARMOVE ;
```

该例中先暂停当前运动，然后打断所有等待运动。

3.2.7 CLOCKRESET

重置用于计时的 CLOCK 数据类型的变量。

格式：CLOCKRESET(变量名);

例子：

```
CLOCKRESET (time) ;
```

注意：在开始重新计时之前，需要调用 CLOCKRESET 指令。

3.2.8 CLOCKSTART

开始计时

格式：CLOCKSTART(变量名)

例子：

```
CLOCKRESET (time) ;
CLOCKSTART (time) ;
CLOCKSTOP (time) ;
```

3.2.9 CLOCKSTOP

停止计时。

例子：

```
CLOCKRESET (time) ;
CLOCKSTART (time) ;
CLOCKSTOP (time) ;
```

此例中，展示了计算运动开始到结束的时间。

3.2.10 CONTINUE 继续

中断 WHILE 或 FOR 循环，并强制重新评估目前的循环条件。

例子：

```
FOR i := 1 TO 5 BY 1 DO
  IF i = 2 THEN
    CONTINUE ;
  END_IF ;
END_FOR ;
```

该例中，当 i 等于 2 时，跳出 IF，返回 FOR 循环重新评估条件。

3.2.11 DWELL 时间等待

在执行后面的指令前等待指定的参数时间（单位为秒）。
格式：DWELL(单位为秒的时间参数)
例子:

```
FOR i := 0 TO 5 BY 1 DO
  IF i = 2 THEN
    DWELL (20) ;
  END_IF ;
END_FOR ;
```

该例中，当 i 等于 2 时，将会等待 20 秒。

3.2.12 ENDPROG

终止当前程序的执行。等待移动将在终止当前程序之前完成。只有重启请求或设置程序指针的请求才能恢复程序执行。
例子:
IF unexpected=TRUE
ENDPROG ;
END_IF
如果变量 unexpected 为真，则程序执行终止。

3.2.13 EPATH 路径执行

用于启动一条路径的执行。
格式：EPATH(PATH_L variable, speed, tool, [refsys]);
注意：如果想要执行路径指令，必须在键入此指令之前导入期望的路径，并将其作为外部变量。如果省略 refsys 参数，则使用世界坐标系进行移动。
注释：[refsys]为可选参数(则可以为空)，以下出现[ ]的变量，皆为可选参数。

3.2.14 ERROR 错误代码

设置 expr 值为系统变量 error 的错误代码。然后错误消息将会在系统中的用户日志文件打印显示，程序的执行将会被中断，错误处理代码将会执行。默认的错误处理是停止目前程序的执行。

格式：ERROR(expr,[expr1],[expr2]);

例子：

```
ERROR (1) ;
```

3.2.15 EXEC 子程序执行

执行在加载指令 LOAD 的模块中定义的子程序。子程序仅在运行时进行评估。 在编译期间，不检查子例程名称，因为它驻留在运行时加载的模块中。

格式：EXEC [result =] subroutine ([parameters]) ;

例子:

LOAD “parprog.xpl” ;

…

EXEC “parprog.work” ;

…

EXEC value = “parprog.calc” (a, b) ;

在此示例中，将执行子例程工作和子例程计算。

Example:

LOAD “sub_p01.xpl” ;

…

EXEC “sub_p01.Main” ;

此示例显示如何从另一个程序运行完整的程序。

要从另一个程序运行程序，需要运行 Main 子例程。

3.2.16 EXIT 结束循环

终止 WHILE 或 FOR 循环。

例子：

```
FOR i := 1 TO 5 BY 1 DO
  IF i = 2 THEN
    EXIT ;
  END_IF ;
END_FOR ;
```

注意：EXIT 指令也可以表示停止执行程序。执行 EXIT 指令之后，只能重新启动程序或者将 PC 设置到特定位置。

3.2.17 FOR 循环体

循环执行某块代码若干次首先需要设置使用初始值表达式 init expression 来初始化变量 variable，在代码块执行每次执行后，变量 variable 将会以增量表达式 increment expression 的值更新；如果更新后的变量 variable 值在初始值 init expression 和终止值 end expression 之间指定范围内，代码块将会再次执行，否则执行 FOR 循环体外的后续指令。若增量表达式不进行任何指定,默认变量 variable 的增量为 1。FOR 循环体重可以使用 EXIT 和 CONTINUE 指令。

格式：

FOR variable := 初始值表达式 TO 终止值表达式 BY [增量表达式]DO：

...

END_FOR;

例子：

```
FOR i := 0 TO 5 BY 1 DO
    MLIN (*, v500, fine, tool0) ;
    MLIN (*, v500, fine, tool0) ;
END_FOR ;
```

3.2.18 GOTO 跳转指令

跳到某一个特定的标签，并执行标签后的程序指令。

格式：GOTO(标签名称)

例子：

```
LABEL txt :
FOR i := 0 TO 5 BY 1 DO
    MLIN (*, v500, fine, tool0) ;
    MLIN (*, v500, fine, tool0) ;
END_FOR ;
GOTO txt ;
```

3.2.19 IF THEN ELSE 条件语句

IF 条件为真，则执行 IF 后的指令语句；否则执行 ELSE 后的指令语句。

格式：

IF 条件语句 THEN ...

ELSE...

END_IF...

例子：

```
IF i = 5 THEN
    a := true ;
ELSE
    a := false ;
END_IF ;
```

注意：当键入一个 IF 时，同时会自动添加 END_IF。如果想要插入 ELSE 语句，必须选中 END_IF 语句，然后在示教器显示的编辑栏进行添加。

3.2.20 INTRCOND 中断触发条件

INTRCOND 指令通过设置条件控制触发与 INTR 变量相关联的中断。
格式：INTRCOND(INTR varibale,Bool expression);
例子：

```
INTRCOND (var, expression) ;
```

注意：Expression 表达式必须为 Bool 类型，当条件满足时，相关联的中断被触发，中断触发为上升沿。

3.2.21 INTRALLOW 中断允许

启用先前被 INTRDENY 指令禁用的特定中断指令。
格式：INTRALLOW(INTR variable);
例子：

```
INTRALLOW (expression) ;
```

3.2.22 INTRDENY 中断否决

用于禁用特定中断触发的指令。此指令之后该指定的任何触发条件都将失效。
格式：INTRDENY(INTR variable);
例子：

```
INTRDENY (expression) ;
```

3.2.23 INTRDIS 中断禁用

禁用所有中断或某个中断的指令。
格式：INTRDIS([INTR variable]);
例子：

```
INTRDIS (expression) ;
```

注意：如果省略了可选参数则所有中断都将被禁用。

3.2.24 INTRENA 中断触发

启用所有中断或某个中断的指令。

格式：INTRENA([INTR variable]);

例子：

```
INTRENA (expression) ;
```

注意：如果省略了可选参数则所有中断都将被启用。

3.2.25 INTRERRNO 错误编号触发中断

设置触发中断时与变量相关联的条件的指令变量_errno_被设置为一个特定值（如果缺省值不是整形数值），或非零值。当变量_errno_被设置时，中断将被触发。

格式：INTRERRNO (INTR variable, [integer value]) ;

例子：

```
INTRERRNO (expression, aaa) ;
```

3.2.26 INTRSET 中断设置

将一个 INTR 变量与子程序连接的指令。

格式：INTRSET(INTR variable,subroutine);

例子：

```
INTRSET (expression, sonapplication()) ;
```

3.2.27 JOIN 等待线程执行结束

等待线程的执行结束，并且如果设置了超时，则可选中断等待。如果线程在 超 时 之前未结束，或线程终止时可选参数不等于零，则可选参数的结果将被 设为 0。超时及参数的结果是可选的。

格式：JOIN(pid_thread, [time] , [result]);

3.2.28 KMAXJ

用于设置主轴在关节空间运动的最大工作参数。此指令是一个模态设置。

格式：KMAXJ([speed] , [acceleration] , [deceleration] , [jerk]);

参数以百分数表示，并参照配置在机器人中的轴组的 MAX_SPE_J,MAX_ACC_J 和 MAX_JER_J 参数。

例子：

```
KMAXJ (10 ) ;
MJOINT (*, v1000, fine, tool1) ;
...
KMAXJ (10 ) ;
MJOINT (*, v100, fine, tool1) ;
```

在上面的设置下，maximum speed 被改变为所有后续关节移动的最大速度的 10%。如果 TANYSEP 为 2000 mm/s，关节速度为 10%，即 10%至 50%间的最小值。

在下面的设置中，maximum speed 被改变为后续关节移动的最大速度的 10%。如果 TANYSEP 为 2000 mm/s，关节速度为 5%即 5%至 10%间的最小值。

注意：例子中，“ * ”代表坐标。以下同义。

3.2.29 KMAXT

用于设置主轴在笛卡尔空间运动的最大工作参数。此指令是一个模态设置。

格式： KMAXT([speed], [acceleration], [deceleration], [jerk]);

每个参数的单位取决于机器配置，速度通常为毫米/秒。Speed 参数用于限制所有的笛卡尔空间内的运动。如果在一个运动中设置一个大于 KABST 值的速度值，则使用的值被缩小为 KABST 的值。

注意：每个参数不能超过为机器中的轴组配置的参数。如果设置过大的值，则将该值设置为机器中配置的最大值。

默认情况下，这些参数（TANYSPE，TANYAACC，TANYJER）由 RPE 计算 为主轴的最大笛卡尔参数之间的最小值(MAX_SPE_C*1…3+，MAX_ACC_C*13+ ，MAX_JER_C*1…3+)。

例子：

```
KMAXT (50, 15) ;
MLIN (*, v500, fine, tool0) ;
```

该例中，速度被设置为 50 毫米/秒，加速度被设置为 15 毫米/秒^2，用于后面所有的笛卡尔移动，减速度以及 jerk 值保持机器设置不变。

3.2.30 KMAXW

用于设置所有笛卡尔运动的姿态最大运动工作参数。这是一个模态设置。

格式：**KMAXW([speed],[acceleration],[deceleration],[jerk]);**

speed 参数被用作限制所有笛卡尔空间运动。

参数以百分比表示，并参照 M_SPE_C,MAX_ACC_C,MAX_JER_C（轴组中的 4、 5、6 轴）参数

例子：

```
KMAXW (50, 15) ;
MLIN (*, v500, fine, tool0) ;
```

在这个例子中，速度被改变为 50 度/秒，并且加速度被改变到 15 度/秒，用于所有后

续笛卡尔空间的运动；减速度和 jerk 保持为机器人配置中设置。

如果 V100 姿态速度重载低于 50，则得到的速度是 V100 指定的值。

3.2.31 LABEL 标签指令

在程序代码中输入一个标签名称，用于 GOTO 指令跳转到代码的特殊部分。

格式：LABEL(标签名称)

例子：

```
LABEL name :
MLIN (*, v500, fine, tool0) ;
MLIN (*, v500, fine, tool0) ;
GOTO name ;
```

3.2.32 LOAD 程序加载

将程序从文件加载到内存中作为当前程序的模块。加载后可以执行子程序并访问模块中定义的数据。

格式：LOAD (filename);

例子:

LOAD (“parprog.xpl”) ;

EXEC “parprog.work” ;

在此示例文件中，“parprog.xpl”作为模块加载。在执行加载模块中定义的子例程工作之后。

3.2.33 MCIRC 圆弧运动

开始执行 TCP 从最后一个位置到目标位置的循环（笛卡尔）移动，通过中间点。此语句的第一次出现设置中间点，下面的 MCIRC 指令设置目标点。如果不使用 MBREAK 或任何其他移动指令中断 MCIRC 的序列，则所有后续的 MCIRC 指令都将使用前两个点对目标点执行圆弧以定义圆。如果需要改变姿态，方向轴（A、B 和 C）将被插补（开始移动并同时到达目标位置）到主轴（X、Y 和 Z）。

注意：如果圆弧运动的起始点和终点是同一个点，圆弧运动将是不可知的。

格式：MCIRC(intermediate point,target point,speed,zone,tool,[refsys])

例子：

```
MLIN (*, v500, fine, tool0) ;
MCIRC (*, *, v500, fine, tool0) ;
```

该例中我们假设第一个点（*）为 a，后面为 b，c，则按照 a，b，c 的顺序进行圆弧运动。

3.2.34 MESSAGE 消息指令

在系统和用户的日志文件中打印显示消息。在文本表达式中使用%1 和%2 作为表达式 1 和表达式 2 的形式参数。

格式：MESSAGE(文本表达式，表达式 1，表达式 2);

例子：

```
MESSAGE ("this num is 1%", 50) ;
```

该例执行之后，打印字符串为“this num is 50”。

3.2.35 MJOINT 关节运动

开始关节运动，从上个点到达此目标点。所有轴开始运动并同时到达目标点。

TCP 运动是机器人运动各轴的组合。如果 refsys 坐标系未指定，默认在世界坐标系下运动。

格式：MJOINT(目标点,速度,overlap 过度参数,工具,参考坐标系);

在此指令中速度(speed)参数指用于计算关节速度的百分比。

3.2.36 MLIN 直线运动

开始执行从最后一个位置到目标位置的关节（点对点）移动。所有轴将开始移动并同时到达目标位置。TCP 移动将是单轴移动的组合。如果省略 refsys 参数，则使用世界坐标系进行移动。

格式：MLIN(point,speed,zone,tool,[refsys]);

例子：

```
MLIN (*, v500, fine, tool0) ;
```

该例中按照直线移动到目标点。

3.2.37 PULSE

为一个布尔变量设置一个预先定义的时间量的特定值。时间用秒表示。经过一段时间后，布尔变量设置为相反的值。通过将可选参数“advance”设置为“真”，可以选择性地推进连续步骤的执行。

格式：PULSE(Boolean var,Value to set,time,[advance]);

例子：

```
PULSE (bool, true, 2, true) ;
PULSE (bool, true, 2, false) ;
```

第一个 PULSE 语句，2 秒之后 bool 的 true 被改为 false，再过 2 秒，又被设置为 true，再过 2 秒，又被设为 false，如此循环。

第二个 PULSE 语句，2 秒之后 bool 的 true 被设置为 false。

3.2.38 RESTART

从主子程序的第一条指令重新开始执行程序的指令。在执行该指令后，没有用初始化值设置变量。

格式：RESTART;

例子：

```
RESTART ;
```

3.2.39 RETURN

中断 RETURN 所在的程序或子程序的执行。

例子：

```
IF bool = true THEN
    RETURN ;
END_IF ;
```

当变量 bool 为 true 时，RETURN 所在的程序段会被中断。

3.2.40 CASE 分支判断

根据表达式的值，运行多个语句序列中的一个。

格式：

CASE test_expression OF
expression_1,[expression_2],[expression_3]:
sequence of statement
[expression from]...[expression to]:
sequence of statement
ELSE
sequence of statement
ELSE_CASE;

与 CASE 表达式匹配的 CASE 指令将会被执行。

ELSE 指令必须在放置在 CASE 块中的最后，如果 CASE 中没有匹配的值，ELSE 将会被执行。

例子：

```
CASE i OF
  1:
    MLIN (*, v500, fine, tool0) ;
  EXIT ;
  2:
    MCIRC (*, *, v500, fine, tool0) ;
  EXIT ;
  3:
    MJOINT (*, v500, fine, tool0) ;
  EXIT ;
END_CASE ;
```

该例中根据 i 取值的不同，执行不同的运动指令。

3.2.41 STARTMOVE

恢复前面被暂停的运动，恢复运动时间参数以秒为单位。

格式：STARMOVE(unhold time);

例子：

```
STARTMOVE (100) ;
```

在 100 秒内恢复运动。

3.2.42 STOPPROG

所有待处理的移动完成后停止程序执行。程序执行可以恢复。执行后，程序指针处于下一条指令。

格式：STOPPROG

例子：

STOPPROG ;

MLIN (target1, v100, fine, tool1) ;

STOPPROG 程序指针将在 MLIN 指令后。有一个新的开始请求运动将被执行。

3.2.43 STOPMOVE

暂停运动，参数以秒为单位设置暂停时间。

格式：STOPMOVE(hold time);

例子：

```
STOPMOVE (20) ;
```

运动被暂停 20 秒。

3.2.44 THREAD

与程序的正常流并行启动子程序的执行。指令的第一个参数必须是数字，并且在执行 THREAD 线程指令后包含子程序的数值引用。此数字引用可用于检　查子例程的执行状态，例如使用 JOIN 指令。

注意：THREAD 启动的子程序不能包含：

---运动指令
---可能会修改 ROBOT 参数的指令

3.2.45 TRIGCALL

设置触发器类型的变量。当触发条件时，调用特定的程序。
格式：TRIGCALL trigger_var, type, ref_value, subroutinetocall();
例子：

```
TRIGCALL trig, Distance, 150, OpenGrimper() ;
```

当 trig 被触发时，将会执行子程序 QpenGrimper。

3.2.46 TRIGON

在下一个运动中激活触发变量。使用 TRIGON 指令最多可以设置 4 个事件。如果多次调用 TRIGON，所有的触发变量将会在运动前的最后一个 TRIGON 指令激活。调用运动指令后，将没有触发变量被设置，因此必须使用多个 TRIGON 指令在下一个运动中来激活触发变量。激活变量只能为 MLIN 和 MCIRC 运动指令使用。其他运动指令 MJOINT 或者 EPATH 不能用于处理事件。
格式：TRIGON (trigger1, [trigger2], [trigger3], [trigger4]) ;
例子：

```
trig := TRIGSET when Distance is 150 do (var := true) ;
TRIGON (trig) ;
MLIN (*, v500, fine, tool0) ;
```

变量 var 在距离*点 150mm 处被设置为 true。

3.2.47 TRIGSET

设置触发器类型的变量。当触发触发器的条件时，将使用表达式的值设置指定的变量。执行 TRIGSET 指令时计算表达式的值。
格式：trigger_var := TRIGSET when type isref_value do (var_to_set := expr_to_set)
触发的类型可以是：
--- Distance 距离：与目标点之间的距离。
---TimeToEnd 时间：到达目标点的时间。
触发变量的数值可以是（与触发变量类型有关）

---如果是距离触发，单位为毫米。

---如果是时间触发，单位为秒。

var_to_set:必须是数值变量。

expr_to_set:当事件被触发时，表达式的值将会被设定。

例子：

```
trig := TRIGSET when Distance is 150 do (var := true) ;
```

当变量 var 为 true 时，trig 被触发。

3.2.48 WAIT 事件等待

等待条件的执行，如果设置了超时，则可以选择中断等待。如果等待正确终止，则可选参数的结果将为 0；如果超时，则不等于 0。timeout 和 result 参数是可选的。

格式：WAIT (condition, [timeout], [result]);

例子：

```
WAIT (io.output[12]) ;
MLIN (*, v500, fine, tool0) ;
```

3.2.49 UNLOAD

从先前加载指令 LOAD 的存储器中卸载模块。 如果是可选参数 save 被定义为 true，加载模块后模块被更改将被保存在卸载之前。

格式：UNLOAD (filename, save);

例子:

LOAD (“parprog.xpl) ;

...

UNLOAD (“parprog.xpl”, true) ;

在此示例中，从“parprog.xpl”加载的模块将从内存中卸载。如果是模块加载后更改模块将被保存。

3.2.50 WHILE 循环

如果 WHILE 条件为真执行下一代码块。

格式：

WHILE condition DO

...

END_WHILE;

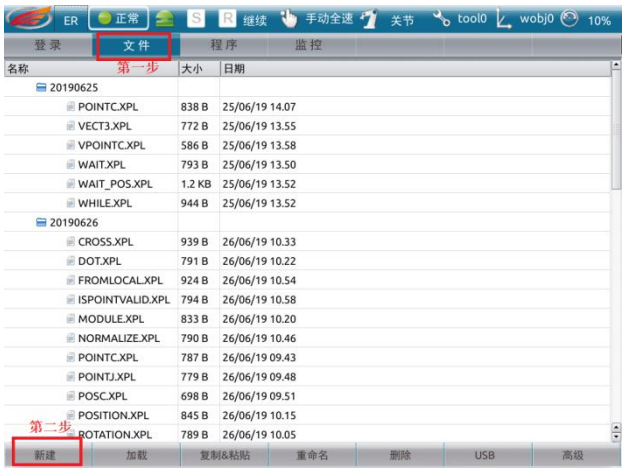
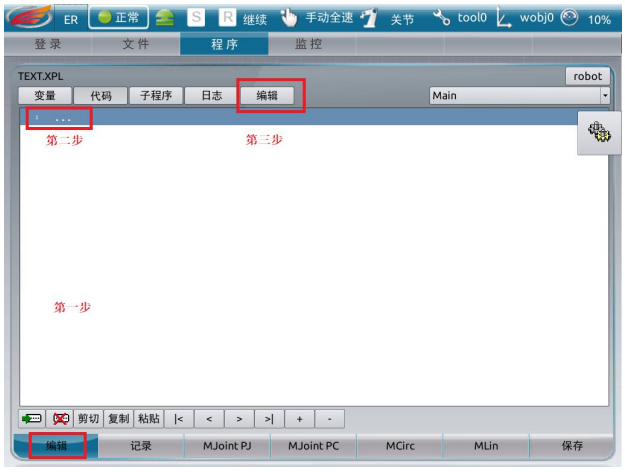
例子：

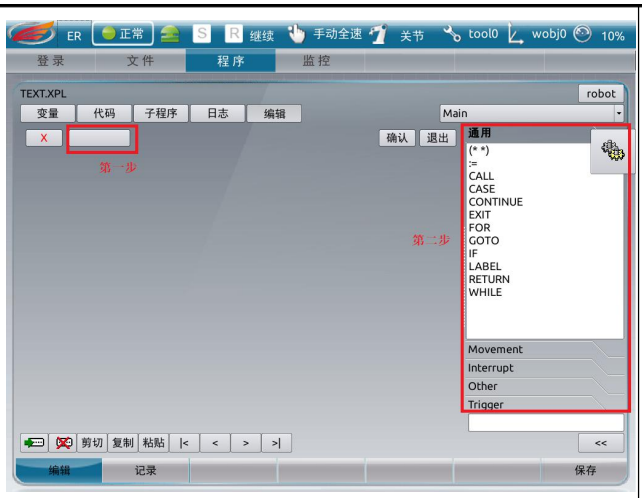
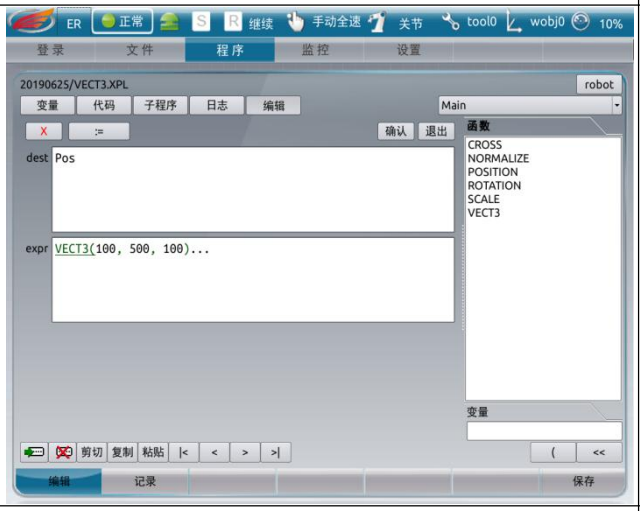
```
WHILE i = 2 DO
    MLIN (*, v500, fine, tool0) ;
END_WHILE ;
```

当 i 等于 2 的时候，执行 MLIN 指令。

4 函数

4.1 函数添加步骤

步骤	图片	描述
1. 登录。		① 点击标题栏“登录”后输入密码(999999)。 ② 点击“登录”。
2. 创建文件夹或文件。		① 在界面中选择文件选项。 ② 点击新建，新建文件夹或文件。 ③ 确定文件夹或文件名称。 ④ 如需加载文件，选中文件后，点击下方加载即可。
3 .编写程序准备。		① 点击“编辑”进入编辑功能。 ② 选中需要编辑的行。 ③ 点击上方编辑。

4. 选择赋值函数 :=		① 选择赋值函数后添加一个变量和其对应的值 ② 选中变量点击“新建”即可创建各个类型的变量，最后点击“确认”即可。
5. 程序确认。		① 选中需要填写的参数。 ② 在变量范围内选择刚刚创建的变量。 ③ 对变量进行相应的设置和应用。 ④ 点击“确认”，创建出坐标系。（下图）。
		① 在该函数中可继续添加子函数，如 DOT 函数，继续设置他对应的参数和值即可完成设置。

说明：其他函数添加方法与该表类似，具体方法根据指令的不同而稍有不同。

4.2 表达式函数

4.2.1 ABS (值)

返回值的绝对值。
例子：i := ABS(-20) ;
在该示例中，值 20 被分配给变量 i。

4.2.2 ASIN (值)

Arcsin 三角函数，结果以弧度表示。
例子：val := ASIN(1) / pi ;
变量 val 值将为 0.5。

4.2.3 ACOS (值)

Arccos 三角函数，结果以弧度表示。
例子：val := ACOS(-1) / pi ;
变量 val 值将为 1

4.2.4 ATAN2 (值 1，值 2)

Arctan 三角函数，结果以弧度表示。
例子：val := ATAN2(SQRT(2), SQRT(2)) / pi ;
例子：valB := ATAN2(SQRT(3), 1) / pi * 180 ;
变量 val 值将为 0.25，变量 valB 将为 60。

4.2.5 COS (弧度)

余弦三角函数。
例子：val := COS(pi) ;
变量 val 将为-1。

4.2.6 LIMIT (值，最小值，最大值)

限制指定范围内的值。
例子：
valueA := LIMIT(45, 10, 100) ;
valueB := LIMIT(800, 10, 100) ;

valueC := LIMIT(-20, 10, 100) ;
变量 valueA 将为 45，valueB 将为 100，valueC 将为 10。

4.2.7 MAX (值 a，值 b)

返回 a，b 之间的最大值。
例子：
valueA := MAX(3, 7) ;
valueB := MAX(10, 2) ;
变量 valueA 将为 7，valueB 将为 10。

4.2.8 MIN (值 a，值 b)

返回 a，b 之间的最小值。
例子：
valueA := MIN(3, 7) ;
valueB := MIN(10, 2) ;
变量 valueA 将为 3，valueB 将为 2。

4.2.9 MOD (值，除数)

给出除法的余数。
例子：i := MOD(10, 3) ;
此例中变量 i 为 1。

4.2.10 PI ()

返回圆周率（3.1415…）。
例子：len := 2 * PI() * r ;
变量 len 为半径为 r 的圆的周长。

4.2.11 RAND (最小值，最大值)

返回最小值和最大值之间的随机数。
例子：val := RAND(10, 100) ;
变量 val 将被赋值为 10~100 之间的随机数，例如 20.25。

4.2.12 RANGE (值，最小值，最大值)

用于检验值是否在最小值最大值之间，如果在该范围内，将返回 true。

例子：

```
testA := RANGE(45, 10, 100) ;
testB := RANGE(800, 10, 100) ;
```

变量 testA 将为 true，testB 将为 false。

4.2.13 ROUND (值)

返回最接近值的整数。

例子：

```
temp := 4.29 ;
i := ROUND(temp) ;
temp := 5.5 ;
a := ROUND(temp) ;
```

变量 i 将为 4，a 为 6。

4.2.14 SIGN (值)

如果值为负，则返回-1，否则返回+1。

例子：absValue := SIGN(-24.5) * (-24.5) ;

变量 absValue 将为 24.5。

4.2.15 SIN (radiant)

正弦三角函数。

例子：val := SIN(pi / 2) ;

变量 val 将为 1。

4.2.16 SQRT (值)

平方根。

例子：val := SQRT(16) ;

变量 val 将为 4。

4.2.17 TAN (弧度)

正切三角函数。

例子：val := TAN(pi / 4) ;

变量 val 将为 1。

4.2.18 TFB ()

返回从引导开始经过的时间（以秒为单位）。也可用于时间测量。
例子：

```
timeA := TFB() ;
...
timeB := TFB() ;
elapsed := timeB – timeA ;
```

4.2.19 TRUNC (值)

返回值的整数部分。
例子：integerValue := TRUNC(3.75) ;
变量 integerValue 将为 3。

4.3 位姿向量和空间点变量函数

4.3.1 VECT3 (x, y, z)

返回具有 3 个实数（浮点）值的向量变量。
格式：Pos := VECT3(x, y, z) ;

4.3.2 VPOINTC (位置向量，旋转向量)

从两个向量返回一个点：position（x，y，z）和 rotation（a，b，c）。 其中 a 是旋转绕轴 z，b 围绕轴 y，c 围绕轴 x。
格式：Point := VPOINTC(VECT3(x, y, z), VECT3(a, b, c)) ;

4.3.3 VEPOINTC (位置向量，旋转向量,eaux point)

返回由两个矢量位置（x，y，z），旋转（rotX，rotY，rotZ）和存储外部辅助轴关节位置的点组成的点。
例子：

```
pos := VECT3(400, 300, 100) ;
rot := VECT3(0, 180, 90) ;
eaux := POINTJ(1, 2, 3, 4, 5, 6) ;
Point := VEPOINTC(pos, rot, eaux) ;
```

4.3.4 POINTC (x, y, z, a, b, c)

返回具有 6 个实数值的点。 a 是绕轴 z 的旋转，b 围绕轴 y，c 绕轴 x 旋转。
格式: Point := POINTC(x, y, z, a, b, c) ;

4.3.5 POINTJ (j1, j2, j3, j4, j5, j6)

返回具有 6 个实数值的点。 j1，j2，j3，j4，j5，j6 是关节位置。
格式: Point := POINTJ(0, 90, 0, 0, 0, 0) ;

4.3.6 EPOINTC (x, y, z, a, b, c, ea1, ea2, ea3, ea4, ea5, ea6)

返回由 12 个实数值组成的扩展点。 a 是绕轴 z 的旋转，围绕轴 y，c 绕轴 x 旋转。 值 ea1，ea2，ea3，ea4，ea5，ea6 是外部辅助轴关节位置。
例子：Point := EPOINTC(40, 300, 100, 0, 0, 0, 1, 2, 3, 4, 5, 6) ;

4.3.7 EPOINTJ (j1, j2, j3, j4, j5, j6, ea1, ea2, ea3, ea4, ea5, ea6)

返回由 12 个实数值组成的扩展点。 值 j1，j2，j3，j4，j5，j6 是关节位置，值 ea1，ea2，ea3，ea4，ea5，ea6 是外部辅助轴关节位置。
例子：Point := EPOINTJ(0, 90, 0, 0, 0, 0, 1, 2, 3, 4, 5, 6) ;

4.3.8 AJEPOINTC (笛卡尔点，轴关节位置点)

返回由笛卡尔点和存储外部辅助轴关节位置的点组成的扩展点。
例子：

```
pos_cart := POINTC(100, 200, 300, 0, 180, 0) ;
eaux := POINTJ(1, 2, 3, 4, 5, 6) ;
Point := AJEPOINTC(pos_cart, eaux) ;
```

4.3.9 AJEPOINTJ (关节点，轴关节位置点)

返回由关节点和存储外部辅助轴关节位置的点组成的扩展点。
例子：

```
pos_joint := POINTJ(10, 20, 30, 1, 2, 3) ;
eaux := POINTJ(4, 5, 6, 7, 8, 9) ;
Point := AJEPOINTJ(pos_joint, eaux) ;
```

4.3.10 POSC (参考坐标系)

将工具实际中心位置（TCP）返回为 POINTC（相对于参考系统）。
例子：

```
lostPiecePosition := POSC(wobj) ;
...
MLIN (servicePosition, v100, fine, tool, wobj0) ;
...
MLIN (lostPiecePosition, v100, fine, tool, wobj) ;
```

在上面的例子中，实际的笛卡尔位置存储在变量 lostPiecePosition 中，然后在一些其他指令机器人将移回到这一点之后执行。

4.3.11 POSJ ()

返回实际的关节位置作为 POINTJ。
例子：

```
retryPosition := POSJ() ;
...
MJOINT (retryPosition, v100, fine, tool) ;
```

4.3.12 POSITION (笛卡尔空间位姿点)

从笛卡尔点提取位置向量（x，y，z）。
例子：

```
Point := POINTC(400, 300, 100, 0, 180, 90) ;
Pos := POSITION(Point) ;
Variable Pos will be (400, 300, 100).
```

4.3.13 ROTATION (笛卡尔空间位姿点)

从笛卡尔点提取旋转矢量（rotX，rotY，rotZ）。
例子：

```
Point := POINTC(400, 300, 100, 0, 180, 90) ;
Rot := ROTATION(Point) ;
Variable Rot will be (90, 180, 0).
```

4.3.14 MODULE (向量)

返回向量的大小（长度）。
例子：

vect := VECT3(SQRT(3), SQRT(3), SQRT(3)) ;
len := MODULE(vect) ;
Variable len will be 3.

4.3.15 DOT (向量 a, 向量 b)

返回两个向量的标量积。
例子：
a := VECT3(100, 0, 0) ;
b := VECT3(-100, 0, 0) ;
cosang := DOT(a, b) / (MODULE(a) * MODULE(b)) ;
变量 cosang 将为-1。

4.3.16 CROSS (向量 a, 向量 b)

返回矢量积。
例子：
a := VECT3(100, 0, 0) ;
b := VECT3(0, 100, 0) ;
c := NORMALIZE(CROSS(a, b)) ;
变量 c 将为(0,0,1)。

4.3.17 NORMALIZE (向量)

返回规范化的向量。 归一化向量是模块为 1 的向量。
例子：
a := VECT3(100, 0, 0) ;
Versor := NORMALIZE(a) ;
Variable versor will be (1, 0, 0).

4.3.18 TOPOS (点, 参考坐标系)

从与 refsys 坐标系相关的点返回世界坐标系中表示的点。
例子：
wobj := REFSYS(wobj0, 100, 200, 50, 0, 0, 0) ;
pointA := POINTC(200, 300, 100, 0, 0, 0) ;
pointB := TOPOS(pointA, wobj) ;
Variable pointB will be (300, 500, 150, 0, 0, 0).

4.3.19 TOLOCALPOS (点，参考坐标系)

从与世界坐标系相关的点返回 refsys 坐标系中表示的点。
例子：
wobj := REFSYS(wobj0, 100, 200, 50, 0, 0, 0) ;
pointA := POINTC(300, 500, 150, 0, 0, 0) ;
pointB := TOLOCALPOS(pointA, wobj) ;
Variable pointB will be (200, 300, 100, 0, 0, 0).

4.3.20 CALCPOSJ (点，参考坐标系)

在实际机器人关节位置转换与 refsys 坐标系相关的笛卡尔点。
返回点是 POINTJ，如果可以转换并且在关节限制内，则该 POINTJ 有效。
例子：
jointP := CALCPOSJ(pointFromCamera, refsysCamera) ;
IF ISPOINTVALID(jointP) THEN
MJOINT(jointP, v100, fine, tool0) ;
ELSE
MESSAGE(“Piece not reachable”) ;
END_IF ;
在上面的示例中，仅当 pointFromCamera 可达时，机器人才会进行关节运动。

4.3.21 ISPOINTVALID (点)

返回一个布尔值，指示某个点是否有效。

4.3.22 OFFSET (笛卡尔空间的点，x, y, z)

用于在 x，y，z 方向上向笛卡尔点添加偏移的函数。
例子：
point_low := POINTC(100,200,0,0,0,0) ;
point_high := OFFSET(point_low, 10, 20, 30) ;
In example above point_high will be (110, 220, 30, 0, 0, 0).

4.3.23 DISTANCE (笛卡尔点，笛卡尔点)

用于计算两个笛卡尔点之间的线性距离的函数。该值始终为正值。
例子：
p1 := POINTC(0,0,0,0,0,0) ;
p2 := POINTC(100,100,100,0,0,0) ;

dist := DISTANCE(p1, p2) ;
在上面的例子中，dist 将是 173.205。

4.3.24 OFFSETTOOL (笛卡尔点, x, y, z, a, b, c)

用于将在活动 TOOL 坐标系中表示的偏移添加到笛卡尔点的函数。
例子：
tool := STOOL(0, 0, 100, 0, 45, 0) ;
point := POINTC(300, 0, 400, 180, -45, 180) ;
point_s := OFFSETTOOL(point, 0, 0, 100, 0, 0, 0) ;
在上面的例子中，point_s 将是（229.289,0,329.289,180,-45,180）。

4.4 参考坐标系的函数

4.4.1 REFSYS (父参考系, x, y, z, a, b, c)

从父系统创建参考系统和六个 LREAL 值。
例子：wobj := REFSYS(wobj0,100,200,300,0,0,0) ;

4.4.2 VREFSYS (父参考系，位置向量，旋转向量)

从父系统创建参考系统和两个向量，第一个为位置向量，第二个为旋转向量。
例子：
pos := VECT3(100,200,300) ;
rot := VECT3(0,0,0) ;
wobj := VREFSYS(wobj0,pos,rot) ;

4.4.3 REFSYS3P (父参考系，起始点, x 方向点, y 方向点)

从父系统创建参考系统和 3 个点，第一个点为起始点，第二个点为 x 方向点，第三个为 y 方向点。
例子：
p0 := POINTC(100,200,300,0,0,0) ;
p1 := POINTC(200,200,300,0,0,0) ;
p2 := POINTC(100,300,300,0,0,0) ;
wobj := REFSYS3P(wobj0,p0,p1,p2) ;

4.4.4 CWOBJ ()

返回当前活动的参考系统。

格式：wobj := CWOBJ() ;

4.4.5 RWOBJ ()

返回活动的机器人参考系统。
格式：wobj := RWOBJ() ;

4.4.6 GETWOBJ (base/holder flag, robot)

返回特定机器人的基础/持有者参考系统。如果 flag 为 FALSE 则返回机器人参考系统，否则返回持有者参考系统。基准参考系统是固定参考系统。保持器参考系统可以是可移动参考系统。例如，输送机具有固定的参考系统，该系统描述了输送机在系统中的位置。
输送机的支架参考系统描述了一个与输送带一起移动的参考系统。

例子：

```
wobj := GETWOBJ(false, cvy) ;
p0 := POINTC(0,0,0,0,0,0) ;
MLIN(p0,spe,ovl,tool,wobj) ;
In this example robot will move at origin of cvy.
wobjM := GETWOBJ(true, cvy) ;
rsPhoto := REFSYS(wobjM, x, y, z, a, b, c) ;
wobj := GETTRKWOBJ(rsPhoto, cvyTrackData) ;
```

在这个例子中，我们制作了一个可移动的参考系统来跟踪传送带上的物体。

4.4.7 GETTRKWOBJ (参考系统，跟踪数据)

设置特定参考系统的跟踪参数并返回可移动参考系统。
格式：wobj := GETTRKWOBJ(wobjcvy, cvyTrackdata) ;

4.4.8 SETROBOTWOBJ (robot，父参考系,参照点)

设置并返回特定机器人的基础 refsys。
例子：cvy := ROBOT(“conveyor”) ;

```
p0 := POINTC(200,-400,200,0,0,0) ;
wobj := SETROBOTWOBJ(cvy,wobj0,p0) ;
```

4.5 设置复杂数据的函数

4.5.1 ROBOT (axesgroup name)

返回从具有指定轴组名称的系统检索的 ROBOT 数据。

当您需要在跟踪应用程序中访问不同的应用程序时，通常会使用此功能。
例子：conveyor := ROBOT("conveyor_belt") ;
在这个例子中，我们将输送机变量与轴组 conveyor_belt 连接起来。

4.5.2 TRACKING(maxspace,startspace,maxspe,maxacc,type,par1)

用于设置跟踪数据。耦合斜坡期间的最大跟踪速度，maxacc 定义耦合斜坡期间的最大跟踪加速度，类型定 义跟踪是线性还是旋转。最后一个参数 par1 用于在传送带位置上进行过滤，值为 0 表示没有过滤器。 如果外部 编码器读取传送带位置，则此参数可能很有用。起始值可以是 0.1。
例子：cvyTrackData := TRACKING(1000, 300, 200, 2000, Linear, 0.1) ;
在这个例子中，我们用一些参数设置 cvyTrackData。

4.5.3 SSPEED (切向速度，定向速度)

此功能可用于设置 SPEED 数据类型。
切向速度以 mm / s 表示，定向速度以度/ s 表示。
例子：spe := SSPEED(500, 5) ;
在这个例子中，我们将 spe 设定为切向速度为 500 mm / s，定向速度为 5 度/ s。

4.5.4 SZONE (线性距离，重定向角距离)

此功能可用于设置 ZONE 数据类型。
线性距离以 mm 表示，重新定向角度距离以度表示。
例子：ovl := SZONE(100, 3) ;
在这个例子中，我们将 ovl 设置为 100 mm 作为线性距离，3 度作为角距离。

4.5.5 STOOL (x, y, z, a, b, c)

此函数可用于设置 TOOL 数据类型。
例子：tool := STOOL(0, 0, 150, 0, 180, 0) ;
在此示例中，我们使用一些参数设置工具。

4.6 字符串相关函数

4.6.1 BOOL_TO_STRING (BOOL val)

转换 BOOL 为 STRING。

例子：

```
BOOL bool_val
STRING strTrue
STRING strFalse
...
bool_val := true ;
strTrue := BOOL_TO_STRING(bool_val) ;
bool_val:= false ;
strFalse := BOOL_TO_STRING(bool_val) ;
```

执行后 strTrue 将为“true”，strFalse 将为“false”。

4.6.2 DINT_TO_STRING (DINT val)

转换 DINT 为 STRING。

例子：

```
DINT dint_val
...
dint_val := -120 ;
strVal := DINT_TO_STRING(dint_val) ;
```

执行后 strVal 为”-120”。

4.6.3 LREAL_TO_STRING (LREAL val)

转换 LREAL 为 STRING。

例子：

```
LREAL lreal_val
...
lreal_val := 3.14 ;
strVal := LREAL_TO_STRING(lreal_val) ;
```

执行后 strVal 为”3.140000”。

4.6.4 UDINT_TO_STRING (UDINT val)

转换 UDINT 为 STRING。

例子：

```
UDINT udint_val
...
udint_val := 456 ;
strVal := UDINT_TO_STRING(udint_val) ;
```

执行后 strVal 为”456”。

4.6.5 LEN (STRING str)

返回字符串的长度。
例子：
strA := “value” ;
n := LEN(strA) ;
执行后 n 赋值为 5。

4.6.6 LEFT (STRING str, pos)

返回字符串 str 最左边 pos 位的字符组成的字符串。
例子：
strA := “Hello world” ;
strB := LEFT(strA, 5) ;
执行后的字符串 strB 将为“Hello”。

4.6.7 RIGHT (STRING str, pos)

返回字符串 str 最右边 pos 位的字符组成的字符串。
例子：
strA := “Hello world” ;
strB := RIGHT(strA, 5) ;
执行后的字符串 strB 将为“world”。

4.6.8 MID (STRING str, len, pos)

返回字符串 str 第 len 位到 pos 位的字符组成的字符串。
例子：
strA := “Time is over” ;
strB := MID(strA, 2, 6) ;
执行后的字符串 strB 将为“is”。

4.6.9 CONCAT (STRING str1, STRING str2)

返回一个由字符串 str1 和字符串 str2 组成的字符串。
例子：
strA := “air” ;
strB := “plane” ;
strC := CONCAT(strA, strB) ;
执行后的字符串 strC 将是“airplane”。

4.6.10 INSERT (STRING str1, STRING str2, pos)

返回在字符串 str1 第 pos 位插入字符串 str2 组成的字符串。
例子：
strA := “I go to office” ;
strB := “post ” ;
strC := INSERT(strA, strB, 8) ;
执行后的字符串 strC 将是”I go to post office”。

4.6.11 FIND (STRING str1, STRING str2)

返回字符串 str2 在字符串 str1 出现的位置。
例子：
strA := “policeman” ;
strB := “man” ;
n := FIND(strA, strB) ;
执行后 n 将是 7。

4.6.12 DELETE (STRING str, len, pos)

返回通过从位置 pos 开始从字符串 str 中删除 len 个字符而获得的字符串。
例子：
strA := “policeman” ;
strB := DELETE(strA, 3, 7) ;
String strB after execution will be “police”.

4.6.13 REPLACE (STRING str1, STRING str2, len, pos)

返回一个字符串，该字符串是将字符串 str1 的位置 pos 处的 len 个字符替换为字符串 str2 的结果。
例子：
strA := “grandmother” ;
strB := “fa” ;
strC := REPLACE(strA, strB, 2, 6) ;
执行后的字符串 strC 将是“granfather”。

4.6.14 STRING_TO_LREAL (STRING str)

转换 string 为 LREAL。
例子：

```
STRING strVal
LREAL lreal_val
...
strVal := “1.5” ;
lreal_val := STRING_TO_LREAL(strVal) ;
```

执行后 lreal_val 为 1.5。

4.6.15 STRING_TO_UDINT (STRING str)

转换 string 为 UDINT。

例子：

```
STRING strVal
UDINT udint_val
...
strVal := “124” ;
udint_val := STRING_TO_UDINT(strVal) ;
```

执行后 udint_val 为 124。

4.6.16 STRING_TO_DINT (STRING str)

转换 string 为 DINT。

例子：

```
STRING strVal
DINT dint_val
...
strVal := “-78” ;
dint_val := STRING_TO_DINT(strVal) ;
```

执行后 dint_val 为-78。

4.6.17 STRING_TO_BOOL (STRING str)

转换 string 为 BOOl。

例子：

```
STRING strVal
BOOL bool_val
...
strVal := “true” ;
bool_val := STRING_TO_BOOL(strVal) ;
```

执行后 bool_val 为 true。

4.7其他函数

4.7.1 RANDOM ()

返回 0 到 32767 之间的随机正整数值。
例子：val := RANDOM() ;
变量 val 可以具有 0 到 32767 之间的值，例如 val 是 27236。

4.7.2 R_AND (UDINT a, UDINT b)

返回两个值的二进制与。
例子：
binA := 7 ;
binB := 5 ;
binC := R_AND(binA, binB) ;
变量 binC 为 5。

4.7.3 R_OR (UDINT a, UDINT b)

返回两个值的二进制的或。
例子：
binA := 7 ;
binB := 5 ;
binC := R_OR(binA, binB) ;
变量 binC 为 7。

4.7.4 R_XOR (UDINT a, UDINT b)

返回两个整数值的二进制异或。
例子：
binA := 7 ;
binB := 5 ;
binC := R_XOR(binA, binB) ;
变量 binC 为 2。

4.7.5 R_NOT (UDINT)

返回整数值的二进制反转。
例子：
binA := 13 ;

binB := 255 ;
binC := R_AND(R_NOT(binA), binB) ;
变量 binC 为 242。

4.7.6 ABS_MOD (值，除数)

返回值除以除数后的余数，如果值为负，则将除数添加到结果中以使其为正。
例子：
valA := ABS_MOD(42, 5) ;
valB := ABS_MOD(-42, 5) ;
变量 valA 为 2，valB 为 3。

4.7.7 CLOCKREAD (时间变量)

返回 CLOCK 变量中包含的已用时间。 时间以秒表示。
例子：

```
    CLOCKRESET(clk) ;
CLOCKSTART(clk) ;
...
CLOCKSTOP(clk) ;
MESSAGE (“Cycle time is %1”, CLOCKREAD(clk)) ;
```

在上面的示例中，可能的消息可以是“Cycle time is 3.234”。