结构文本(ST)
TM246

贝加莱工业自动化
Perfection in Automation
www.br - automation.com

简介

前提

培训模块：	TM210 – Automation Studio基础 TM211 – Automation Studio在线通讯 TM213 – 自动化运行（Runtime）系统 TM223 – Automation Studio诊断
软件：	无
硬件：	无

目录

1、简介

结构文本是一种高级语言，如果你知道如何使用高级语言来编程，像：Basic、PACSAL或C，那么你会很轻松的掌握Structured Text（ST）编程；如果不知道，你会看到ST有着简单、标准的结构，保证程序高效、快速运行并简单易懂。

图. 1 书本印刷: 过去和现在

在下一章里，你会学习到ST的命令、关键字、语法和其它的主题。所有的这些你都可以做练习，我们有许多帮助你理解ST的简单例程。

简介

1.1 目的

课程参与者将熟悉使用结构文本给自动化目标编程。
课程参与者将学到每个独立的命令组和它们之间如何工作。
课程参与者将了解结构文本预留的关键字。

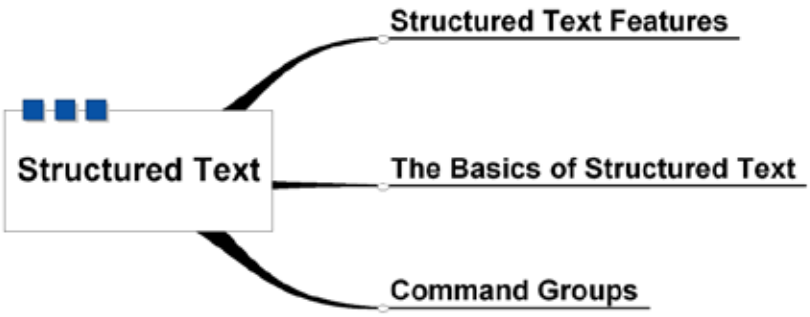


图. 2 综述

2、结构化文本的特点

2.1 概述

ST是针对自动化系统的高级文本编程语言。简单的标准结构确保快速、高效的编程。ST使用了高级语言的许多传统特性，包括：变量、操作符和控制流程语句。ST还能与其它的PLC编程语言一起工作。

那么什么是结构文本呢？"结构"是指高水平的结构化编程能力，象一个"结构化的编程"；"文本"是指应用文本而不是梯形图和顺序函功能表的能力。

ST语言不能代替其它的语言，每种语言都有它自己的优点和缺点。ST主要的一个优点就是能简化复杂的数学方程。

2.2 特点

结构化文本有以下特点:

高级文本编程语言

结构化的编程

简单的标准结构

快速高效的编程

使用直观灵活

与PASCAL类似

有计算机编程经验的人可以很容易地使用它

符合IEC 61131-3标准

2.3 可能性

Automation Studio提供以下功能:

数字量和模拟量I/O

逻辑操作

逻辑比较表达式

算术运算

判断语句

机器的状态语句

循环语句

功能块

可选用的动态变量

诊断工具

3、结构文本基础

3.1 表达式

表达式是指返回变量评估值的结构。表达式由操作符和操作数组成。操作数可以是常量,变量,调用函数或其它表达式。

例子:

```
b + c

(a - b + c)**COS(b)

SIN(a) * COS(b)
```

3.2 赋值操作符

通过一个表达式和一个值来给变量赋值。赋值语句包括位于左边的变量，赋值操作符":="，及后边需要计算的表达式。所有的语句，包括赋值语句，必须要以分号";"结尾。

例子:

```
Var1 := Var2 * 2; (*  Var1 <-- (Var2 * 2) *)
```

图 4 Assignment

当这行程序执行后，变量"Var1"的值是变量"Var2"的两倍。

3.3 注释

虽然注释经常被删掉，但它们是源代码中非常重要的一部分。它们解释了一部分代码，使程序更易读懂。注释帮助你或其他人读你的程序，即使过去了很长时间。注释不被编译，因此不会影响程序的执行。注释应该用一对星号和小括号括起来"(*comment*)"。

例子:

```
(* This is one line comment *)
```

图. 6 单行注释

```
(* This
is more
lines
comment *)
```

3.4 操作符优先级

如果在一个表达式中使用几个操作符，就会出现优先级的问题（执行的顺序）。操作符按优先级的顺序来执行。

在任何一个表达式中，首先执行最高级别的操作符，接着执行低一级的操作符，等等，直到执行完所有的操作符。具有相同级别的操作符按照书写顺序从左至右依次执行。

操作符	符号 / 语法:	
括号	()	最高优先级
函数调用		
例子	Call argument(s)	
LN(A), MAX(X), 等.		
注释	**	
取反	NOT	
乘		
除		
取模 (取除法的余数)	*	
/		
MOD		
加		
减	+	
-		
比较	<, >, <=, >=	
等于		
不等于	=	
<>		
逻辑与	AND	
逻辑异或	XOR	
逻辑或	OR	最低优先级

执行顺序:

例 1:

```
Result := 6 + 7 * 5 - 3;     (*The multiplication first; higher precendence *)
Result := 6 +   35  - 3;     (*The addition; rule from left to right *)
Result := 41 - 3;            (*Substraction at the end *)
Result := 38;
```

图. 7 例 1: 执行顺序

首先做乘法，然后是加法，最后是减法。
使用小括号（最高优先级），可以得到你想要的执行顺序。看下面的例子。

例 2:

如下所示，将操作符放到小括号里可能影响执行的顺序。

```
Result := (6 + 7) * (5 - 3);(*operations inside the parentheses first *)
Result :=    13   *    2;   (*then the multiplication *)
Result :=    26;
```

图. 8 例 2: 执行顺序

表达式从左至右执行。先执行小括号里的操作，接着是乘法。因为小括号的优先级高于乘法的优先级。可以看出，这两个例子看起来很相似，但结果不同。

4、命令组

ST有下面的命令组:
布尔逻辑操作
算术操作
比较操作
判断
Case语句

4.1 布尔逻辑操作

操作数不需要是BOOL类型。
布尔逻辑操作:

符号	逻辑操作	例子
NOT	取反	a := NOT b;
AND	逻辑与	a := b AND c;
OR	逻辑或	a := b OR c;
XOR	异或	a := b XOR c;

真值表:

输入		AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

这些操作符可以形成一个逻辑表达式和条件语句，结果是真（TRUE）或假（FALSE）。

命令组

例 1:

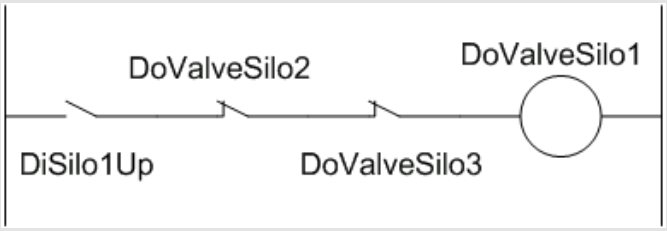


图. 10 电气图

ST编辑器允许任何数的分枝。

例 2:

```
IF (Level >= MaxLevel) OR (E_Stop = 1) THEN
  Pump := 0;
END_IF
```

图. 10 电气图

练习:

当按下"BtnLigntOn"开关后，输出"DoLight"应该亮起，直到"BtnLightOff"按下后才关闭。
使用布尔逻辑操作编写该任务。

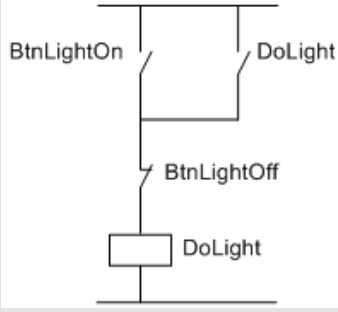


图. 11 例子, 逻辑操作

4.2 算术运算

使用高级语言的决定性因素是看它处理算术运算的简单程度。

4.2.1 基本的算术运算

ST为应用程序提供了以下基本的算术运算:

符号	算术操作	例子
:=	赋值	a := b;
+	加	a := b + c;
-	减	a := b - c;
*	乘	a := b * c;
/	除	a := b / c;
MOD	取模 (显示余数)	a := b mod c;

数据类型是非常重要的参数。看下面的表格。:

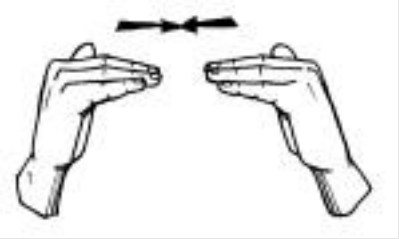
语法	数据类型	结果		
	Res	Op 1	Op 2	
Res := 8 / 3;	INT	INT	INT	2
Res := 8 / 3;	REAL	INT	INT	2.0
Res := 8.0 / 3;	REAL	REAL	INT	2.6667
Res := 8.0 / 3;	INT	REAL	INT	Error

* 编译器出错信息: Type mismatch: Cannot convert REAL to INT.

你可以看到，结果也依赖于语法和数据类型。
表达式左边的数据类型应该等同于（或大于）右边的数据类型。

备注:

左面数据类型 := 右面数据类型;



4.2.2 隐性数据类型转换

该类型的转换由编译器完成。编译器将表达式中低的数据类型转换成高的数据类型。如果有两种或多个类型的变量参与运算，那么必须将它们转换成相同的类型以便执行运算。

Data type	BOOL	SINT	INT	DINT	USINT	UINT	UDINT	REAL
BOOL	BOOL	x	x	x	x	x	x	x
SINT	x		INT	DINT	USINT	UINT	UDINT	REAL
INT	x	INT		DINT	INT	UINT	UDINT	REAL
DINT	x	DINT	DINT		DINT	UDINT	UDINT	REAL
USINT	x	USINT	INT	DINT		UINT	UDINT	REAL
UINT	x	UINT	UINT	DINT	UINT		UDINT	REAL
UDINT	x	UDINT	UDINT	UDINT	UDINT	UDINT		REAL
REAL	x	REAL	REAL	REAL	REAL	REAL	REAL	

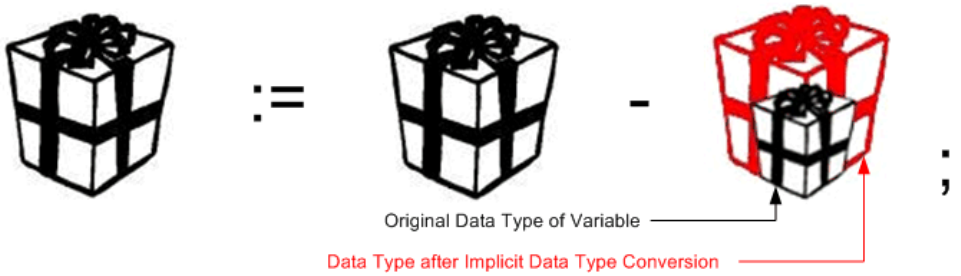


图. 12 隐性数据类型转换

例子:

```
INT_Result := INT_Var1 + SINT_Var2
(* [INT]        [INT]      [SINT] *)
```

图. 13 固有数据类型转换例子

SINT_Var2 首先转换成INT类型。

4.2.3 显性数据类型转换

显性数据类型转换也是数据类型转换问题。我们知道，表达式的左右两边要有相同的数据类型，但还需注意…

例子:

```
INT_TotalWeight := INT_Weight1 + INT_Weight2
(*   [INT]              [INT]          [INT]  *)
```

第一眼看上去好像没什么问题，但和（INT_Weight1+INT_Weight2）超过了INT的取值范围。在这种情况下，必须使用显性数据类型转换。

例子:

```
DINT_TotalWeight := INT_TO_DINT(INT_Weight1) + INT_Weight2
(*   [DINT]                        [INT]          [INT]  *)
```

变量DINT_TotalWeight应该为DINT类型，右边的变量中至少有一个应转换成DINT类型。这种转换用的是OPERATOR库中的函数。

练习:

在两个不同的地方检测玻璃钢的温度，编写程序来计算平均温度，并以模拟量显示输出。
注意模拟量输入和输出必须是INT 类型。

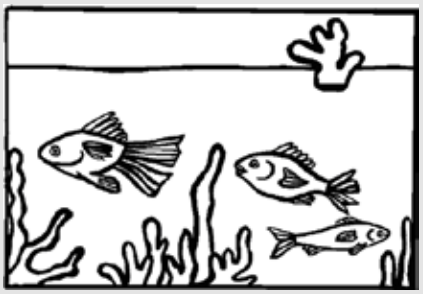


图. 14 玻璃钢

4.3 比较操作

高级编程语言ST或以允许比较操作的简单结构分枝。比较的结果是真（TRUE）或假（FALSE）。

符号	逻辑比较含义	例子
=	等于	IF a = b THEN
<>	不等于	IF a <> b THEN
>	大于	IF a > b THEN
>=	大于等于	IF a >= b THEN
<	小于	IF a < b THEN
<=	小于等于	IF a <= b THEN

比较操作作为一个逻辑条件用在IF, ELSE, WHILE 和UNTIL语句中。

4.4 判断

用IF语句表示判断，这里还要用到比较操作。判断分三部分:

简单IF语句

IF – ELSE语句

IF – ELSIF语句

嵌套的IF

判断	语法	描述
IF THEN	**IF a > b THEN**	**1. 比较**
	Result := 1;	**1. 语句(s)**
ELSIF THEN	**ELSIF a > c THEN**	**2. 比较 (可选)**
	Result := 2;	**2. 语句(s)**
ELSE	**ELSE**	**前面IF语句都不满足(可选)**
	Result : = 3;	**3. 语句(s)**
END_IF	**END_IF**	**判断结束**

4.4.1 IF

最简单的IF判断语句。

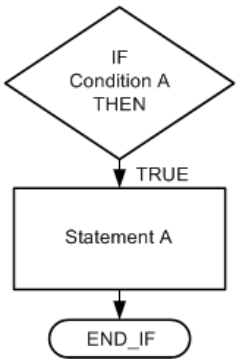


图. 15 简单 IF 语句流程图

```
(*********************************)
(******    Nested  IF     ********)
(*********************************)
IF V1 > V2 THEN
  IF V2 > V4 THEN
    V3 := 99;          (* comment *)
  END_IF
END_IF
(*********************************)
```

例子:

```
IF (Level >= MaxLevel) OR (E_Stop = 1) THEN
  Pump := 0;
END_IF
```

图. 16 简单 IF 语句程序

基本上，如果条件表达式的结果为TRUE就执行语句。如果条件表达式的结果是FALSE，程序就执行END_IF后面的语句。条件表达式可以是简单关联的语句或用运算符（and,or等）连接的复合语句。

4.4.2 ELSE

它是简单IF语句的扩展。在IF结构中应该只有一个ELSE。

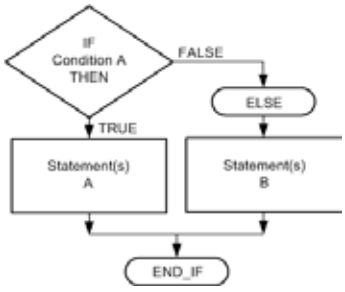


图. 17 IF- ELSE 语句

```
(*******************************)
(******     IF ELSE      ********)
(*******************************)
IF V1 > V2 THEN
  V3 := 99;          (* comment *)
ELSE
  V4 := 66;          (* comment *)
END_IF
(*******************************)
```

图. 18 IF – ELSE 程序

如果条件为TRUE，执行语句A。如果条件为FALSE，执行语句B。

4.4.3 ELSIF

运用一个或多个ELSE_IF语句可以实现多个不同的条件，而不用使用多个简单的IF语句创建复杂的程序逻辑。

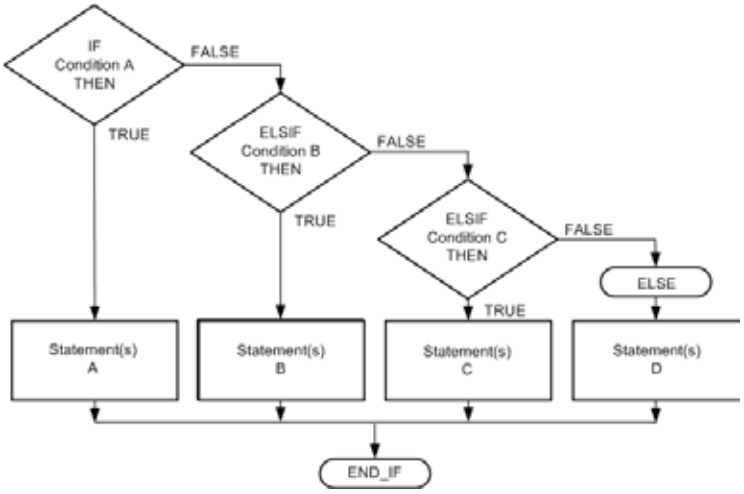


图. 19 IF-ELSIF-ELSE 流程图

```
(*****************************)
(****** IF, ELSIF, ELSE *******)
(*****************************)
IF V1 > V2 THEN
  V3 := 99;          (* comment *)
ELSIF V1 > V4 THEN
  V5 := 88;          (* comment *)
ELSIF V1 > V6 THEN
  V7 := 77;          (* comment *)
ELSE
  V8 := 66;          (* comment *)
END_IF
(*****************************)
```

图. 20 IF-ELSIF-ELSE 程序

处理器自上而下地执行判定。如果条件的结果为TRUE，那么就执行属于这个条件的指令和命令，之后处理器就跳到判断语句的结尾（END_IF）。在程序的一次循环中，无论下一个条件是否为TRUE，只有上面属于第一个条件TRUE的语句被执行。如果IF或ELSIF条件都不为TRUE，那么就执行属于ELSE下的指令。

任务: 气象站 – 第 I部分

用温度计来测量室外的温度，温度通过模拟量读取(1°=10)，并且要以文本的形式在房间里显示。

- 当温度在18°C以下，显示"cold"（冷）。
- 当温度是在18°C到25°C之间，显示"opt"（最佳）。
- 当温度是25°C以上，显示"hot"（热）。

用IF，ELSIF和ELSE语句实现这个任务。

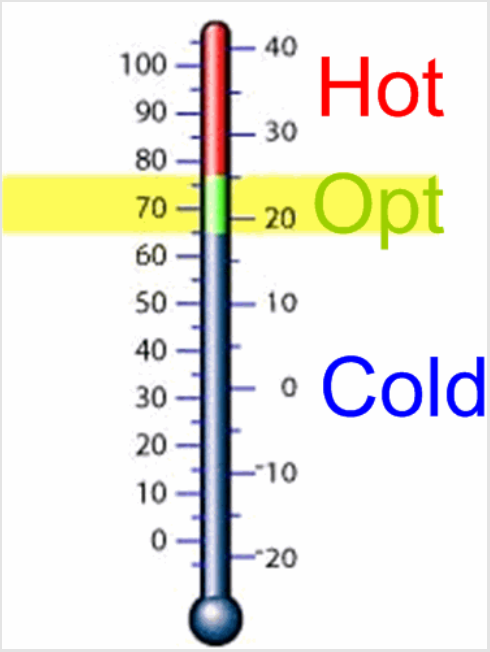


图. 21 温度计, 例子, IF

备注:

在ST中, 按如下指定一个字符串文本:
StringVar := ’ COLD’

4.4.4 嵌套的IF语句

嵌套对于依赖其它条件的条件测试很有用处。一个嵌套的IF语句其优先级低于上一层的优先级，它的执行取决于上一层IF条件的结果。使用嵌套时必须注意每个IF要和END_IF相匹配，否则会导致错误的执行顺序。

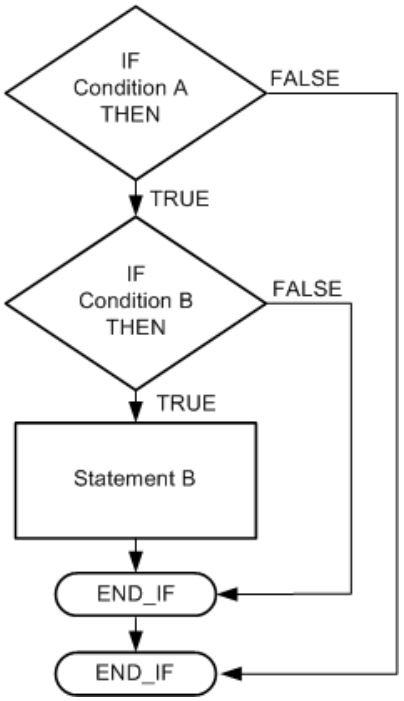


图. 22 嵌套 IF 语句

```
(*********************************)
(******    Nested  IF     ********)
(*********************************)
IF V1 > V2 THEN
  IF V2 > V4 THEN
    V3 := 99;          (* comment *)
  END_IF
END_IF
(*********************************)
```

图. 23 嵌套 IF 语句程序

建议在每个嵌套的IF 语句和它的执行语句中使用缩排。IF语句可以嵌套在你想要的深度，但是三十或四十级以后，编辑器会用完内存，所以这是一种非常糟糕的编程格式！这时你完全丢失了代码的总貌。
当使用嵌套多于2级或3级时，你应该重组程序代码。

任务: 气象站 – 第 II部分

检测温度和湿度。
只在当湿度介于40和75%之间并且温度在18 和25° C之间显示文本"OPT"。否则显示"Temp. OK"。
运用嵌套的IF 语句完成该任务。

用两个但并不嵌套的IF语句也可以达到同样的功能，就像嵌套的语句一样。可以在多个语句中运用标识变量或一个标志。第一个IF语句描述该标志，其他的IF语句利用该变量。

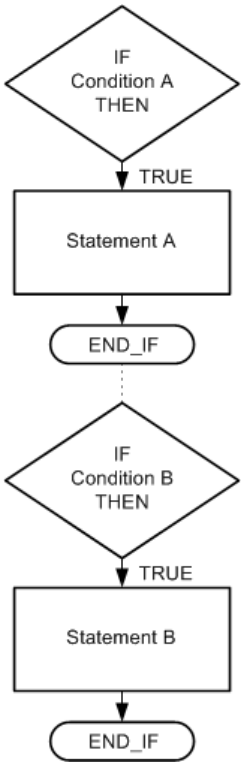


图. 24 两个 IF 语句

```
(********************)
(* 1. IF Statement *)
(********************)
IF V1 > V2 THEN
  V3 := 99;
END_IF
(********************)
(********************)
(* 2. IF Statement *)
(********************)
IF V3 = 99 THEN
  Burner := ON;
END_IF
(********************)
```

Fig. 25 两个 IF 语句

在这种情况下，IF语句有相同的优先级，第二个IF语句和第一个IF语句一样每次都执行。不用相同的变量，你可以有两个独立的IF语句。
当发生以下情况时，推荐使用Case语句代替IF语句:
IF结构有过多分层
使用过多（三个或更多）的ELSE_IF

在这种情况下Case语句更容易读懂。
CASE与IF结构相比较还具有另一个优点：CASE语句中只做一次计算，并能创建更有效的代码。

4.5 Case 语句

在CASE语句中，控制变量与几个值作比较，如果表达式的结果与其中一个值相同，那么就执行相应的语句。如果表达式的结果与任何一个值都不相同，那么就执行象IF 语句一样的ELSE分支。

语句执行完后，继续执行END_CASE后的程序。

关键字	语法	描述
CASE OF	CASE step variable OF	CASE开始
	1,5: Display := MATERIAL	从 1 到 5
	2: Display := TEMP	2
	3,4,6..10: Display := OPERATION	3,4,6,7,8,9,10
END_CASE	END_CASE	CASE结束

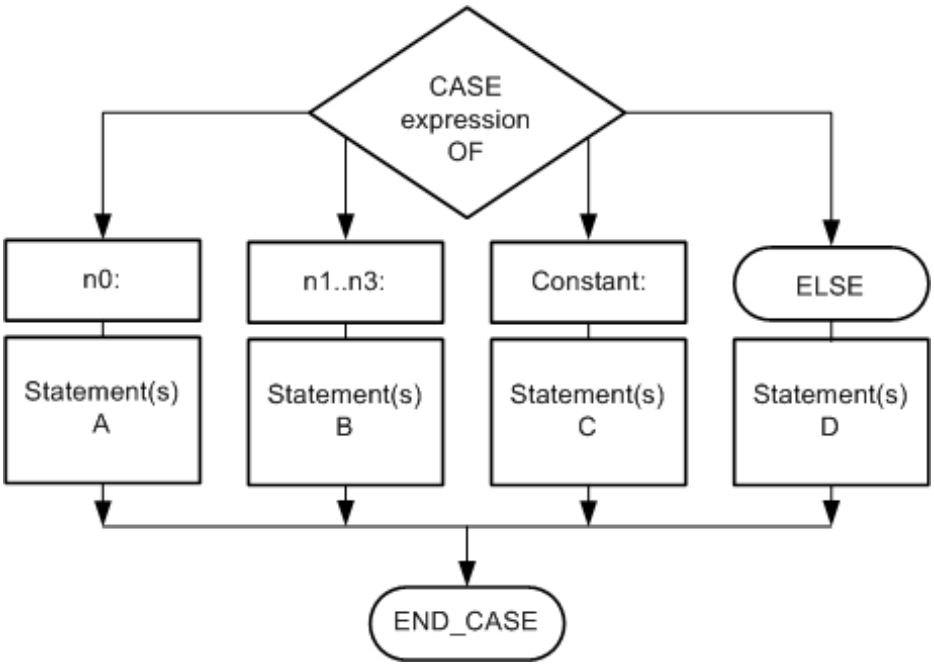


图. 26 CASE流程图

在程序的一次循环中，只执行case的一个子句。

```
(*****************************************)
(**************** CASE ******************)
(*****************************************)
CASE NumbSelectItem OF
    0:      heat:= LOW;      (*Commands A*)
            fan:= LOW;

    1..3:   heat:= MEDIUM;   (*Commands B*)
            fan:= MEDIUM;

    SELECTIONHIGH:
            heat:= HIGH;     (*Commands C*)
            fan:= HIGH;


ELSE
    heat:= OFF;              (*Commands D*)
    fan:= OFF;
END_CASE
```

图. 27 CASE语句程序

> 备注:
>
> 在CASE 语句步骤中可以用常量来代替数字。这样可以更加容易读程序。

CASE语句的语法:
CASE语句以CASE开始，以END_CASE结束，并且各自单独占一行。
在CASE和OF之间的变量必须是UINT类型。
在CASE的子句中，只能使用正整数，不允许使用变量名或表达式。
数字不能重叠使或在几个区域内使用。

任务: 酿造槽

酿造槽的填充程度由low, ok,和high表示。分别运用输出表示low, ok,和high等级。
通过读取模拟量输入内部转换成0-100%来表示槽中液体的水平面。如果容量低于1%,将会出发一个警告声。
运用CASE语句创建一个方案。

Fig. 30 酿造槽

4.6 循环语句

在很多应用程序中，需要多次执行某些步骤，这就是重复执行代码的原因，这个过程叫循环。循环程序的设计中需要建立这样一种程序使它能够循环返回并循环执行自身程序。

循环语句使源代码简短并一目了然。
循环语句可以嵌套在其它语句中。
循环程序编写时很容易进入死循环，一直重复执行自身程序，引起控制器启动看门狗限制，并产生一个严重错误阻碍程序的执行。
因此，通常使用一些方法来跳出循环：设定循环次数或条件改变时停止循环。
在ST中有几种不同的循环方式:
有限制的
FOR
无限制的
WHILE
REPEAT

4.6.1 FOR

如果提前可以确定循环的次数就用FOR语句，否则就用WHILE 或REPEAT。

关键字	语法	描述
FOR TO BY DO	FOR i:=StartVal TO StopVal {BY Step} DO	{}中的部分是可选的
	Res := value + 1;	循环语句段
END_FOR	END_FOR	FOR语句结束

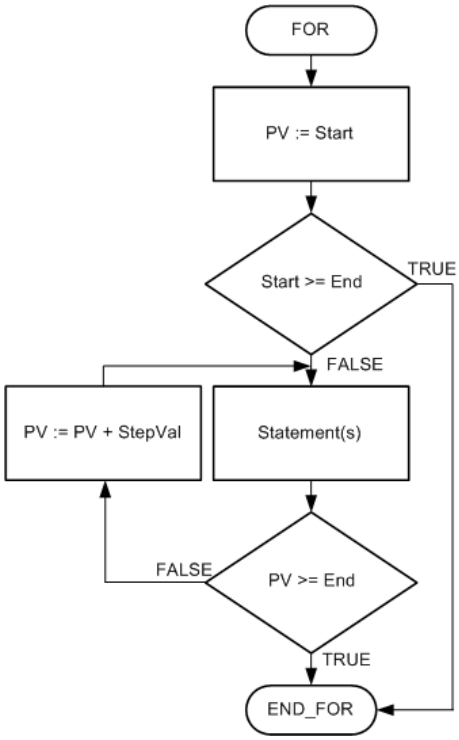


图. 29 FOR 流程图

```
(*********************************************)
(*************       FOR       ***************)
(*********************************************)
FOR i := StartValue TO EndValue BY StepValue DO
  VarFor := VarFor + 1;
END_FOR
(*********************************************)
```

图. 30 FOR语句程序

FOR 循环中的语句反复被执行，而递增值分配给了FOR语句的循环控制变量(i)。每次循环后，变量"StepVal"会自动加1。"StartVal"和"EndVal" 是循环控制变量的起始值和终止值。当控制变量达到终止值时，程序就执行END_FOR 语句。循环控制变量、起始值和终止值都必须是相同的数据类型((U)SINT, (U)INT, (U)DINT)，并且不能由循环语句改变。

FOR语句可以递增或递减循环控制变量的值，使它从起始值到达终止值。递增或递减的默认值为1。

每次循环之前都会检测终止条件，如果循环控制变量的值超过终止值时，就不再执行语句段了。

任务: 起重机

5个独立的负载悬挂在起重机上。为了得到总负载，您需要累加每个独立的负载。

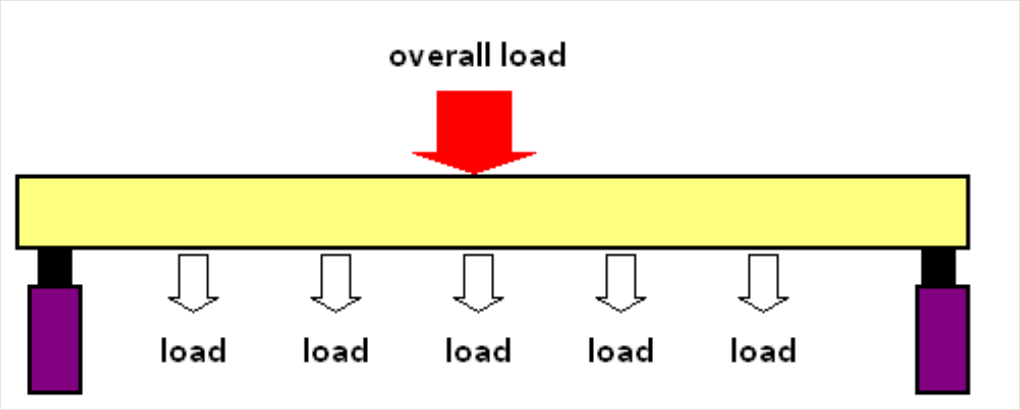


图. 32 Crane

使用FOR 循环语句创建任务解决方案。

4.6.2 WHILE

WHILE循环除了条件可以是任意的布尔表达式以外其他和FOR循环用处一样。当条件满足时,执行循环语句。当逻辑条件为TRUE就重复调用语句时，使用WHILE语句。

关键字	语法	描述
WHILE DO	WHILE i<4 DO	布尔条件
	Res := value + 1;	语句
	i := i + 1;	语句
END_ WHILE	END_ WHILE	WHILE结束

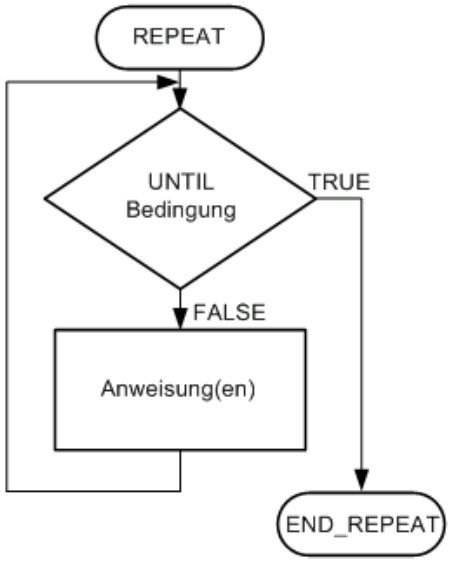


图. 32 WHILE流程图

```
(*************************************)
(**********    WHILE     *************)
(*************************************)
WHILE (indexWhile < EndIndexWhile)DO
  VarWhile := VarWhile + 1;
  indexWhile := VarWhile;
END_WHILE
(*************************************)
```

图. 33 WHILE语句程序

WHILE语句执行时首先检测条件。如果条件为TRUE就执行语句段。当执行完语句段后，再次检测条件，如果条件仍为TRUE，那么就再次执行语句段，直到条件不为TRUE。如果条件一开始就为FALSE，那么就不会执行语句段。

备注:

如果条件永远达不到FALSE,语句将无休止的执行,造成运行错误。

4.6.3 REPEAT

REPEAT语句与WHILE语句不同，它在循环执行后检测条件。也就是说不管有没达到终止条件循环至少执行一次。

关键字	语法	描述
REPEAT	REPEAT	开始循环
	Res := value + 1;	语句
	i := i + 1;	语句
UNTIL	UNTIL i > 4	退出条件
END_ REPEAT	END_ REPEAT	循环结束

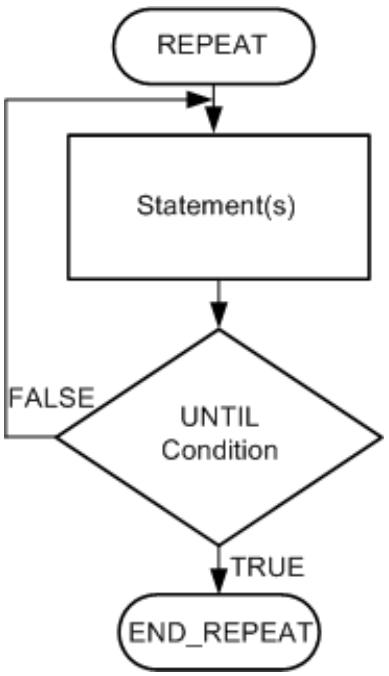


图. 34 REPEAT流程图

```
(*****************************)
(*******    REPEAT    ********)
(*****************************)
REPEAT
  UNTIL VarRepeat > 3
  VarRepeat := VarRepeat + 1;
END_REPEAT
(*****************************)
```

图. 35 REPEAT语句程序

语句一直执行直到UNTIL条件为TRUE。如果UNTIL条件在第一次执行时为TRUE，那么语句只执行一次。

备注:

如果UNTIL条件永远不为TRUE，那么程序进入死循环，并产生一个运行错误。

4.6.4 EXIT

EXIT语句是当终止条件满足时，退出所有的循环语句。

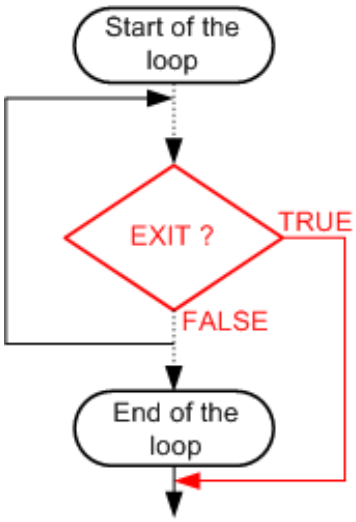


图. 36 EXIT流程图

```
(*****************************)
(*****  REPEAT & EXIT    *****)
(*****************************)
REPEAT
  VarRepeat := VarRepeat + 1;
  UNTIL VarRepeat > 3
  IF VarRepeat = VarExit THEN
    EXIT;  (*** EXIT loop ***)
  END_IF
END_REPEAT
(*****************************)
```

图. 37 EXIT语句程序

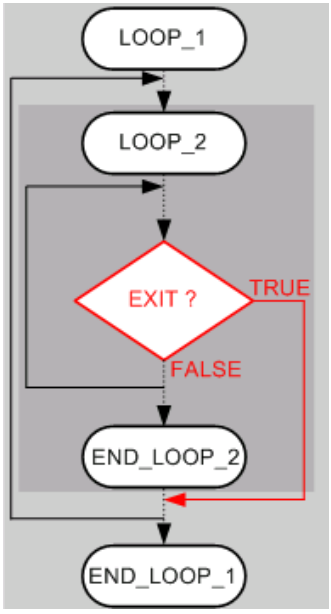


图. 38 嵌套循环中的EXIT流程图

```
(***********************************)
(******* EXIT NESTED LOOP **********)
(***********************************)
WHILE (indexWhile < EndIndexWhile)DO
  VarWhile := VarWhile + 1;
  FOR i := StartValue TO EndValue DO
    VarFor_1 := VarFor_1 + 1;
    IF VarFor_1 = VarExit THEN
      EXIT;(*** <<<-- EXIT loop  ***)
    END_IF
  END_FOR
  indexWhile := VarWhile;
END_WHILE
(***********************************)
```

图. 39 FOR 嵌套语句中的EXIT语句程序

当EXIT 语句用在一个嵌套的循环中时，退出只是在EXIT所在的循环中执行。循环结束后继续执行结束语句后的程序。

4.7 调用功能块

ST中调用功能块的方法：写出功能块的名称，并在括号中给的需的输入参数分配变量名。

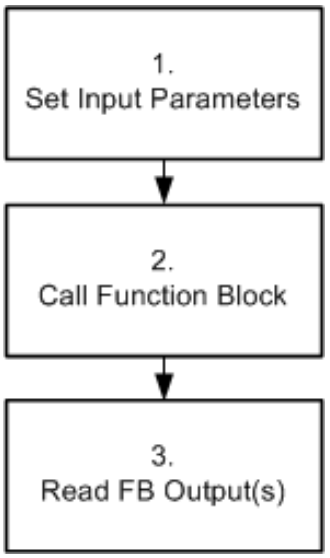


图. 40 调用功能块流程图

```
(*************************************)
(****  Calling of  Function Block   ****)
(*************************************)
PresetTime := T#3s;
TON_0(IN:= DiSensConv1, PT:= PresetTime);
DoConv1 := TON_0.Q;
(*************************************)
```

图. 41 调用功能块程序

在调用功能块之前，需要给输入参数分配所需的值。功能块调用放在一行里，以分号结束。在调用功能块之后才能读取FB输出值。

Function block call in detail:

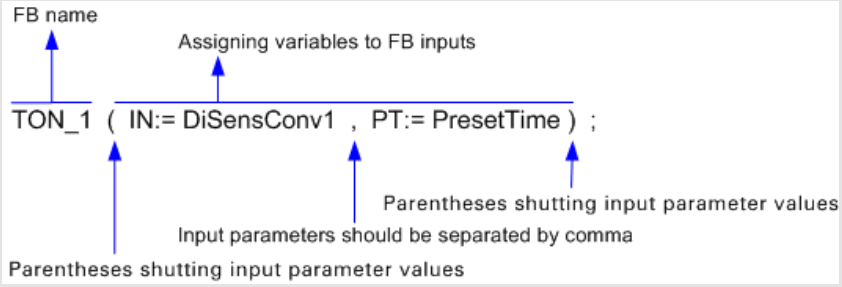


图. 42 功能块调用的明细

首先是功能块的名称，接着是位于括号里的输入参数。输入参数用逗号隔开。功能块调用以分号结束。

命令组

任务: 瓶记数

创建一个程序来计算传送带上瓶子的数量。运用STANDARD 库中的CTU (上升沿计数器)功能块。

图. 43 瓶记数

备注:

在调用功能块是可以参考Automation Studio在线帮助。

4.8 指针和动态变量

B&R为您提供指针以扩展ST的功能。使用这些是可选的。

在运行时，系统给动态变量PV分配了一个内存地址，这个过程叫做动态变量的寻参或初始化。

当一个动态变量被初始化后，根据数据类型它可以获得它所指向的存储器的内容。

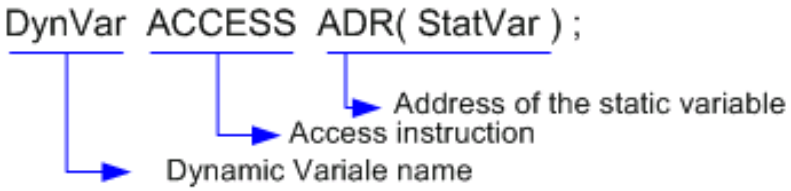


图. 43 定位动态变量

用ADR()操作符，返回的是小括号中变量的地址，为UDINT类型。这行语句应该以分号“；”结束。

5、小结

结构文本是一种高级语言，并能提供非常广泛的指令。ST的优势在于它的简洁性，它比ANSI C简单易学，比梯形图或指令表效率更高。

图. 44 书本印刷: 过去和现在

当学习完这个培训模块后，您就可以编写自己的ST任务了。您也可以把该培训模块当作参考。

该编程语言尤其在数学功能和数学公式的运算时十分有用。

6、练习

任务: 起重箱

两个传送带(doConvTop, doConvBottom)把箱子传输到起重机。
如果光电池(diConvTop, diConvBottom)激活, 相应的传送带停止并调用起重机。
如果起重机没有被调用,它返回到适当的位置(doLiftTop, doLiftBottom)。
当起重机到达正确位置(diLiftTop, diLiftBottom),起重机传送带(doConvLift)启动直到箱子完全位于起重机上。
然后起重机运动到卸载位置(doLiftUnload)。当它到达该位置(diLiftUnload), 箱子移到卸载带上。
知道箱子离开了起重机,起重机才能空闲接受下一个请求。

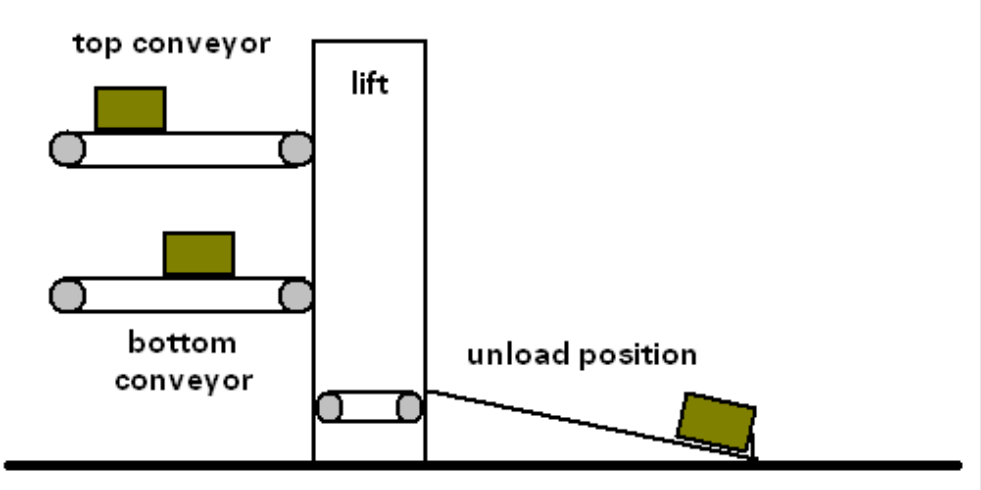


图. 45 起重箱

7、附录

7.1 关键字

关键字是ST可用的语句，但不能作为变量名。Automation Studio编辑器用蓝色来显示它们。

关键字	描述
ACCESS	使用动态变量。
BIT_CLR	A := BIT_CLR(IN, POS) A为变量IN在POS 位清0后的值。IN 值保持不变。
BIT_SET	A := BIT_CLR(IN, POS) A为变量IN在POS 位置1后的值。IN 值保持不变。
BIT_TST	A := BIT_TST(IN, POS) 位的状态值。 A为IN变量在POS位的状态值。
BY	参考FOR 语句。
CASE	参考CASE语句。
DO	参考 WHILE语句。
EDGE	检测上升沿和下降沿信号。
EDGENEG	检测上升沿信号。
EDGEPOS	检测下降沿信号。
ELSE	参考IF语句。
ELSIF	参考 IF语句。
END_CASE	参考 CASE语句。
END_FOR	参考 FOR语句。
END_IF	参考 IF语句。
END_REPEAT	参考 REPEAT语句。
END_WHILE	参考 WHILE语句。
EXIT	参考 EXIT语句。
FOR	参考 FOR语句。
IF	参考 IF语句。
REPEAT	参考 REPEAT语句。
RETURN	可以通过例子中的条件来结束函数。
THEN	参考 IF语句。
TO	参考 FOR语句。
UNTIL	参考 REPEAT语句。
WHILE	参考 WHILE语句。

7.2 函数

在结构文本中使用有些函数不需插入库函数。Automation Studio编辑器用蓝色显示这些函数。

函数	例子
ABS	返回数的绝对值。 ABS(-2) 返回 2。
ACOS	返回数的arc cosine值。(cosine的反函数)。
ADR	返回变量的地址。
AND	位操作逻辑与。
ASIN	返回数的arc sine值。(sine的反函数)。
ASR	操作数算术右移: A := ASR (IN, N) IN 右移N位, 左边由符号位填充。
ATAN	返回数的arc tangent值。(tangent的反函数)。
COS	返回数的cosine值。
EXP	指数函数: A := EXP (IN).
EXPT	一个操作数为另一个操作数的幂: A := EXPT (IN1, IN2).
LIMIT	A = LIMIT (MIN, IN, MAX) MIN 是结果的下限, MAX是结果的上限。如果IN小于MIN,则返回结果是MIN。如果IN大于MAX,则返回结果是MAX。否则,返回结果为IN。
LN	返回数的自然对数。
LOG	返回以10为底的数的对数。
MAX	最大值函数。返回两个数中较大的数。
MIN	最小值函数。返回两个数中较小的数。
MOD	将USINT, SINT, INT, UINT, UDINT, DINT 变量取模除另一个这些类型的变量。
MOVE	把输入变量复制到输出变量。:=符号用有赋值语句中。 "A := B;" 和 "A := MOVE (B)相同;"
MUX	选择: A = MUX (CHOICE, IN1, IN2, ... INX); CHOICE指定返回哪个操作数(IN1, IN2, ... INX)。
NOT	位取反。
OR	位操作逻辑或。
ROL	循环左移操作数: A := ROL(IN, N); IN中每位左移N次,最左面的位移到右边。
ROR	循环右移操作数: A := ROL(IN, N); IN中每位右移N次,最右面的位移到左边。
SEL	二进制选择: A := SEL (CHOICE, IN1, IN2) CHOICE必须是BOOL类型。如果CHOICE为FALSE,那返回IN1，否则返回IN2。
SHL	操作数的位左移: A := SHL (IN, N); IN 左移 N 位, 右边的位以零填充。

SHR	操作数的位右移: A := SHR (IN, N); IN 右移 N 位, 左边的位以零填充。
SIN	返回数的sine值。
sizeof	以字节数返回指定变量的长度。
SQRT	返回数的平方根。
TAN	返回数的tangent值。
TRUNC	返回数的整数部分。
XOR	位操作逻辑异或

Notes

培训模块综述

TM200 – 贝加莱B&R 公司介绍**
TM201 – 贝加莱B&R 产品系列**
TM210 – Automation Studio™ 基础
TM211 – Automation Studio™ 在线通信
TM212 – 自动化对象（Target）**
TM213 – 自动化运行（Runtime）系统
TM220 – 维护信息*
TM221 – 自动化组件和出错信息查询*
TM223 – Automation Studio™ 诊断
TM230 – 结构化软件编程
TM231 – 面向机器设备的Automation Studio™ *
TM240 – 梯形图(LAD)
TM241 – 功能块图 (FBD)*
TM242 – 连续功能图 (CFC)*
TM243 – 顺序功能图 (SFC)*
TM245 – 指令表 (IL)*
TM246 – 结构文本 (ST)
TM247 – Automation Basic (AB)*
TM248 – ANSI C
TM250 – 内存管理和数据存贮
TM260 – Automation Studio™ 函数库I
TM261 – Automation Studio™ 函数库 II*
TM264 – 定时处理单元（TPU）*

TM400 – 运动控制的基础
TM401 – 贝加莱B&R 运动控制产品**
TM402 – 运动控制系统的计算*
TM410 – ASiM 的基础
TM440 – ASiM的基本功能
TM441 – ASiM多轴运动功能
TM445 – ACOPOS ACP10 软件
TM446 – 电子凸轮*
TM447 – ACOPOS 智能过程技术（SPT）*
TM450 – ACOPOS 控制理念和控制器设置
TM460 – 启动B&R 电机*
TM461 – 启动第三方电机*
TM470 – CNC*

TM600 – 图文显示的基础
TM601 – 贝加莱人机界面产品**
TM610 – ASiV 的基础
TM620 –ASiV 的维护*
TM630 – 图文显示的编程规则
TM640 –ASiV报警系统
TM650 –ASiV的国际化操作
TM660 –ASiV 的远程操作
TM670 –ASiV 高级应用

TM700 – Automation Net PVI
TM701 – PVI 通信*
TM710 – PVI DLL 编程
TM711 – PVI的服务
TM712 – PVIControl.NET
TM720 – PVI 维护和诊断*
TM730 – PVI OPC

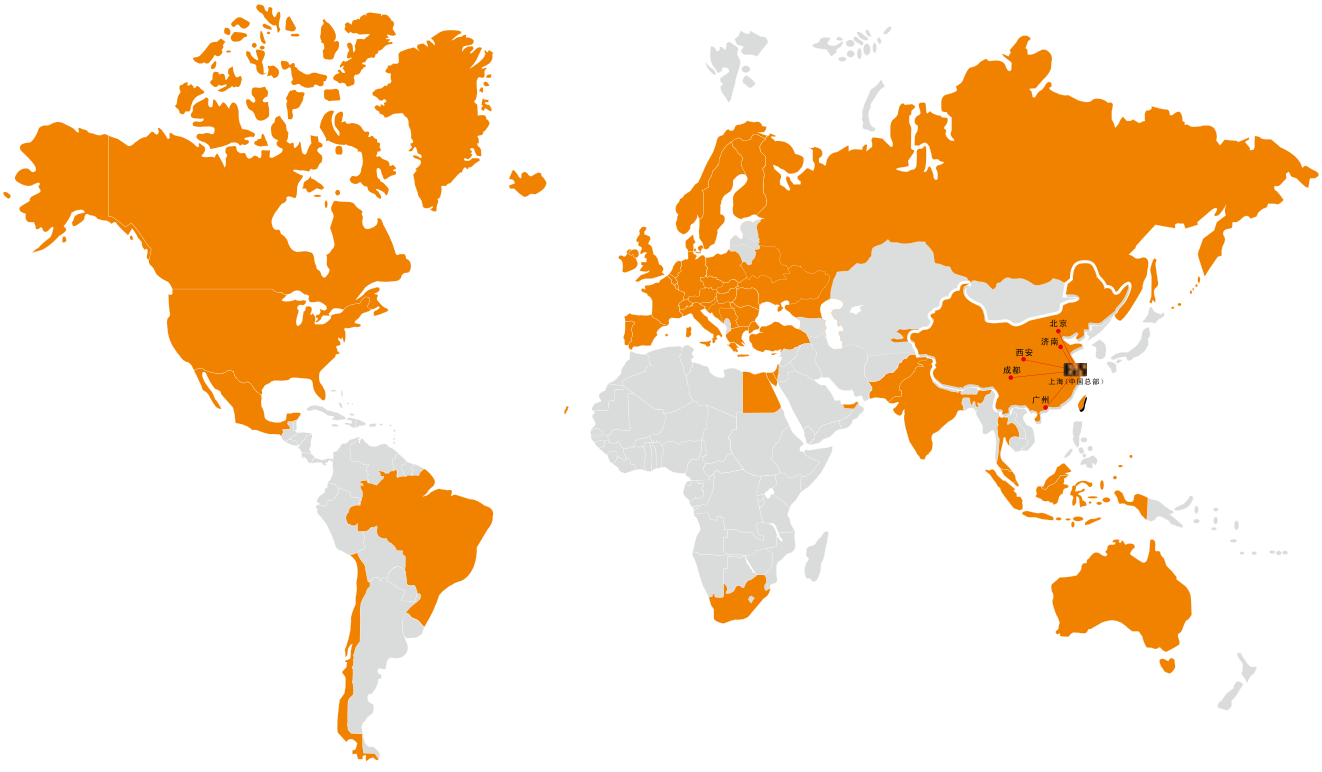
TM800 – APROL 系统概念
TM801 – APROL 工程设计基础
TM810 – APROL 安装，配置和恢复*
TM811 – APROL运行(Runtime)系统*
TM812 – APROL 操作员管理
TM813 – APROL XML 查询*
TM814 – APROL 审计追踪*
TM820 – APROL 维护*
TM830 – APROL 项目工程设计
TM840 – APROL 参数管理和配方
TM850 – APROL 控制器配置和INA 通讯
TM860 – APROL 库设计
TM861 – APROL 通讯互联*
TM865 – APROL 库指导手册
TM870 – APROL Python编程*
TM880 – APROL 报表*

** 查看产品目录
* 即将出版

全球总部
Bernecker+Rainer Industrie-Elektronik Ges.m.b.H.
B&R Straße 1
A-5142 Eggelsberg　奥地利
Tel.：+43(0)7748/6586-0
Fax：+43(0)7748/6586-26
info@br-automation.com
www.br-automation.com

中国总部
贝加莱工业自动化（上海）有限公司
上海市漕宝路70号光大会展中心C座16楼
Tel.：+86/(0)21/6432 6000
Fax：+86/(0)21/6432 6108
info.cn@br-automation.com
www.br-automation.cn

全球50多个国家超过120个分支机构 **www.br-automation.com/contact**



中国总部　● 中国办事处

Austria · Australia · Belgium · Belarus · Brazil · Bulgaria · Canada · Chile · China · Croatia · Cyprus · Czech Republic
Denmark · Egypt · Emirates · Finland · France · Germany · Greece · Hungary · India · Indonesia · Ireland · Israel · Italy · Korea
Kyrgyzstan · Malaysia · Mexico · The Netherlands · Norway · Pakistan · Poland · Portugal · Romania · Russia · Singapore
Slovakia · Slovenia · South Africa · Spain · Sweden · Switzerland · Thailand · Turkey · Ukraine · United Kingdom · USA