CoDeSys 软件编程

用户指令手册 V1.0

ABB(中国)有限公司 武汉分公司
地址：中国湖北省武汉市武昌中南路 7 号 3408 室
电话：(027)87259222 传真：(027)87259233
手机：15927508200 直拨：(027)87259230

目录

1. 标准数据类型

1.1 BOOL

BOOL 类型变量可取值 TRUE 和 FALSE。保留 8 位内存空间。

1.2 整型数据类型

所有的整型数据类型为：

- BYTE 字节
- WORD 字
- DWORD 双字
- SINT 短整型
- USINT 无符号短整型
- INT 整型
- UINT 无符号整型
- DINT 双精度整型
- UDINT 无符号双精度整型

各个不同的数据类型有不同的值范围。下表为整型数据的值范围和占用的内存空间

类型	下限	上限	内存空间
BYTE	0	255	8 位
WORD	0	65535	16 位
DWORD	0	4294967295	32 位
SINT	-128	127	8 位
USINT	0	255	8 位
INT	-32768	32767	16 位
UINT	0	65535	16 位
DINT	-2147483648	2147483647	32 位
UDINT	0	4294967295	32 位

当用大类型转换为小类型时，将导致丢失信息。

1.3 REAL / LREAL

REAL 和 LREAL 被称为浮点数类型。用于有理数表示。REAL 占用 32 位内存空间，LREAL 占用 64 位。

1.4 STRING

STRING 类型变量可以是包含任何字符的字符串。其容量大小在声明变量时说明，如果不对容量大小进行说明，其缺省值为 80 个字符。

字符串变量声明示例：

str : STRING(35) := ‘This is a String’;

1.5 时间、日期类型

TIME、TIME_OF_DAY (缩写 TOD)、DATE 和 DATE_AND_TIME (缩写 DT) 数据类型在内部作为 DWORD 处理。TIME 和 TOD 中的时间用毫秒表示，TOD 中的时间从 12:00 AM 开始。 DATE 和 DT 中的时间用秒表示，并从 1970 年 1 月 1 日 12:00 AM 开始。时间数据的格式在常数一节中说明。

1.6 常数

1.6.1 BOOL – 常数

BOOL – 常数为逻辑值 TRUE 和 FALSE。

1.6.2 TIME – 常数

TIME 常数可以在 TwinCAT PLC 控制中声明。主要用于标准库中定时器的操作，格式如下：
T# xx d xx h xx m xx s xx ms
其中：T 表示时间常数起始，# 数值符号，d 天，h 小时，m 分，s 秒，ms 毫秒。
下面是 ST 中分配的正确的 TIME 常数示例：

```
TIME1 := T#14ms;
TIME1 := T#100s12ms; (* 单位最大的成员允许超过其极限 *)
TIME1 := t#12h34m15s;
```

不正确的 TIME 常数示例：

```
TIME1 := t#5m68s; (*单位较小的成员超过其极限 *)
TIME1 := 15ms; (* 遗漏 T# *)
TIME1 := t#4ms13d; (* 单位顺序错 *)
```

1.6.3 DATE – 常数

该常数用于输入日期。DATE 常数的声明用 d、D、DATE 或 date，后接 # 构成，可以输入格式为 年-月-日的任何日期。
示例：

```
DATE#1996-05-06
d#1972-03-29
```

1.6.4 TIME_OF_DAY 常数

该常数主要用于存储一天中的时间。TIME_OF_DAY 声明用 tod#,TOD#,TIME_OF_DAY# 或 time_of_day# 后接一个时间格式：小时：分：秒。秒可以用实数表示。
示例：

```
TIME_OF_DAY#15:36:30.123
tod#00:00:00
```

1.6.5 DATE_AND_TIME - 常数

日期常数和时间常数可以组合成所谓的 DATE_AND_TIME 常数。DATE_AND_TIME 常数用 dt#, DT#, DATE_AND_TIME# 或 date_and_time# 开始，后接日期和时间，日期和时间之间用 – 连接。
示例：
DATE_AND_TIME#1996-05-06-15:36:30
dt#1972-03-29-00:00:00

1.6.6 数值常数

数值可以用二进制、八进制、十进制和十六进制数表示。
示例：
14 (十进制数)
2#1001_0011 (二进制数)
8#67 (八进制数)
16#A (十六进制数)
这些数值可以是 BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL 或 LREAL 类型，不允许直接从“大类型”向“小类型”转换。例如，DINT 变量不能简单地以 INT 型变量使用。可以用标准库中的转换功能实现其转换。

1.6.7 REAL / LREAL - 常数

REAL 和 LREAL 常数可以用尾数和指数表示，并使用美国标准。
示例：
7.4 取代 7,4
1.64e+009 取代 1,64e+009

1.6.8 STRING - 常数

字符串是由字符组成的序列。STRING 常数使用单引号对区分。一些特殊的符号可用下表表示：

字符	说明
$$	美元符号
$L 或 $l	行给进
$N 或 $n	新行
$P 或 $p	页给进
$R 或 $r	行结束
$T 或 $t	制表
$’	单引号

示例：

'Your Name'
'Susi and Claus'
':-) $''

2. 变量类型转换功能

不能直接从"大类型"向"小类型"变量转换。(例如：从 INT 到 BYTE，或从 DINT 到 WORD)。要完成此功能，可以使用特殊功能块实现。作为一种规则，你可以用此功能将一种类型的变量转换成任何需要的类型变量。

句法：

<elem.Tpy1>_TO_<elem.Typ2>

2.1 BOOL_TO_变换

BOOL 类型变量到不同类型的变换：对于数值类型变量，操作数为 TRUE 时，结果为 1；操作数为 FALSE 时，结果为 0。对字符串类型变量，其结果分别为 'TRUE' 和 'FALSE'。

ST 中的示例：

```
i:=      BOOL_TO_INT(TRUE);          (*结果为 1 *)
str:=    BOOL_TO_STRING(TRUE);       (*结果为 'TRUE' *)
t:=      BOOL_TO_TIME(TRUE);         (*结果为 T#1ms *)
tof:=    BOOL_TO_TOD(TRUE);          (*结果为 TOD#00:00:00.001 *)
dat:=    BOOL_TO_DATE(FALSE);        (*结果为 D#1970-01-01 *)
dandt:=  BOOL_TO_DT(TRUE);           (*结果为 DT#1970-01-01-00:00:01 *)
```

2.2 TO_BOOL – 变换

其它类型变量到 BOOL 的转换：当操作数不为零时，结果为 TRUE，当操作数为零时，结果为 FALSE；当字符串变量的操作数为'TRUE'时，结果为真，否则，结果为假。

ST 中的示例：

```
b := BYTE_TO_BOOL(2#11010101);  (*结果为 TRUE *)
b := INT_TO_BOOL(0);            (*结果为 FALSE *)
b := TIME_TO_BOOL(T#5ms);       (*结果为 TRUE *)
b := STRING_TO_BOOL('TRUE');    (*结果为 TRUE *)
```

2.3 整型数类型之间的转换

整型数值类型到其它数值类型的转换：当从大类型向小类型转换时，存在丢失信息的危险。如果转换的数值超过其极限；则该数的第一个字节将被忽略。

ST 中的示例：

```
si := INT_TO_SINT(4223);  (* 结果为 127 *)
```

如果你将整数 4223 (十六进制为 16#107f) 作为 SINT 变量保存，其结果为 127 (十六进制为 16#7f)。

IL 中的示例：

```
LD 2
INT_TO_REAL
```

MUL 3.5

2.4 REAL_TO - / LREAL_TO – 转换

REAL 或 LREAL 类型到其它数值类型的转换：数值将向上或向下取整并转换成新的数据类型。但变量类型 STRING,BOOL,REAL 和 LREAL 除外。当从大类型向小类型转换时，存在丢失信息的危险。

请注意：当向字符串变量转换时，保留 16 个数据，如果(L)REAL 数据有更多的数，则第十六个数将被取整。如果字符串的长度定义为短型，则从右端开始截取。

ST 中的示例：

```
i := REAL_TO_INT(1.5);     (* 结果为 2 *)
j := REAL_TO_INT(1.4);     (* 结果为 1 *)
```

IL 中的示例：

```
LD 2.7
REAL_TO_INT
GE %MW8
```

2.5 TIME_TO - / TIME_OF_DAY – 转换

TIME 或 TIME_OF_DAY 类型到其它类型的转换：时间在内部以毫秒单位及 DWORD 方式处理(对于 TIME_OF_DAY 变量，用 12:00 AM 起始)。该值将被转换。当从大类型向小类型转换时，存在丢失信息的危险。对于字符串类型变量，其结果为时间常数。

ST 中的示例：

```
str :=TIME_TO_STRING(T#12ms);          (*结果为 'T#12ms' *)
dw:=TIME_TO_DWORD(T#5m);               (*结果为 300000 *)
si:=TOD_TO_SINT(TOD#00:00:00.012);     (*结果为 12 *)
```

2.6 DATE_TO - / DT_TO – 转换

DATE 或 DATE_AND_TIME 类型到其它类型的转换：时间在内部以 1970.01.01 开始所经过的时间，并以秒为单位及 DWORD 方式处理。该值将被转换。当从大类型向小类型转换时，存在丢失信息的危险。对于字符串类型变量，其结果为日期常数。

ST 中的示例：

```
b :=DATE_TO_BOOL(D#1970-01-01);                (*结果为 FALSE *)
i :=DATE_TO_INT(D#1970-01-15);                 (*结果为 29952 *)
byt :=DT_TO_BYTE(DT#1970-01-15-05:05:05);      (*结果为 129 *)
str:=DT_TO_STRING(DT#1998-02-13-14:20);        (*结果为 'DT#1998-02-13-14:20' *)
```

2.7 STRING_TO – 转换

STRING 类型到其它类型的转换：字符串类型变量中必须包含有效的目标变量类型值，否则其转换结果为零。

ST 中的示例：

```
b :=STRING_TO_BOOL('TRUE');     (* 结果为 TRUE *)
```

w :=STRING_TO_WORD('abc34');　(* 结果为 0 *)
t :=STRING_TO_TIME('T#127ms');　(* 结果为 T#127ms *)

2.8 TRUNC (取整)

从 REAL 到 INT 类型转换。数值的所有部分都将被使用。当从大类型向小类型转换时，存在丢失信息的危险。

ST 中的示例：

i:=TRUNC(1.9);　(* 结果为 1 *).
i:=TRUNC(-1.4);　(* 结果为 1 *).

IL 中的示例：

LD 2.7
TRUNC
GE %MW8

3. 用户定义的数据类型

3.1 数组

支持一维、二维和三维数组的成员数据类型。数组可在 POU 的声明部分和全局变量表中定义。

语法：

<数组名> : ARRAY [<ll1>..<ul1>,<ll2>..<ul2>] OF <elem. Type>

ll1，ll2 为数组维数的下限标识，ul1 和 ul2 为数组维数的上限标识。数值范围必须为整数。

示例：

Card_game: ARRAY [1..13, 1..4] OF INT;

数组的初始化：

可以对数组中的所有元素进行初始化，或不进行初始化。

数组初始化示例：

arr1 : ARRAY [1..5] OF INT := 1,2,3,4,5;
arr2 : ARRAY [1..2,3..4] OF INT := 1,3(7); (* 等同 1,7,7,7 *)
arr3 : ARRAY [1..2,2..3,3..4] OF INT := 2(0),4(4),2,3; (* 等同 0,0,4,4,4,4,2,3 *)

结构化中的数组初始化示例：

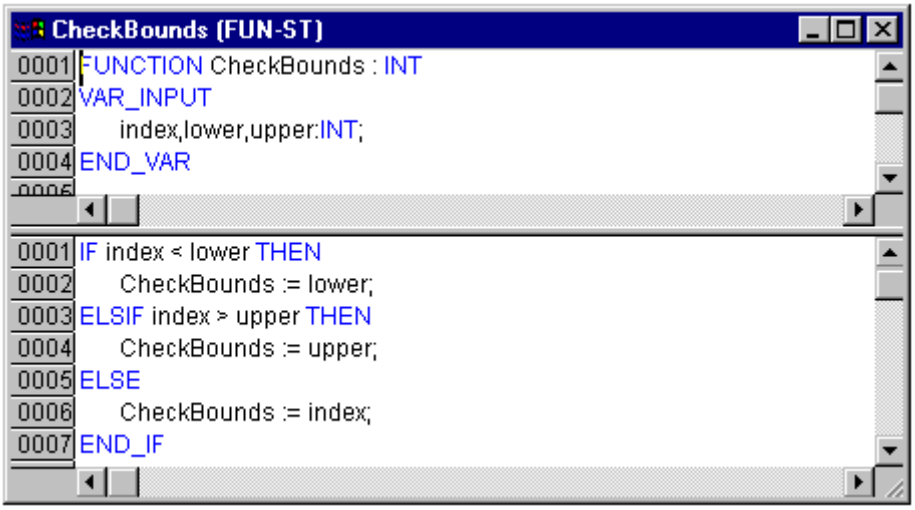
TYPE STRUCT1
STRUCT
p1:int;
p2:int;
p3:dword;
END_STRUCT

ARRAY[1..3] OF STRUCT1:= (p1:=1;p2:=10;p3:=4723),(p1:=2;p2:=0;p3:=299),

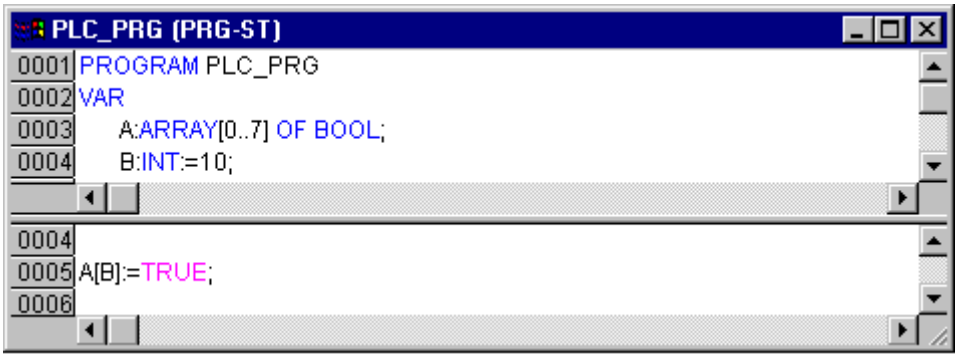
(p1:=14;p2:=5;p3:=112);

数组部分元素初始化示例：
arr1 : ARRAY [1..10] OF INT := 1,2;
数组中的元素如果没有初始化值，则用基本类型的缺省值初始化其值。在上例中，元素 arr1[3] 到元素 arr1[10] 均被初始化为 0。
二维数组的元素存取，使用下列语法：
<数组名>[Index1,Index2]
示例
Card_game[9,2]

注：
如果你在项目中定义了一个名为 CheckBounds 的功能，则可以自动检查数组的上下限超限错误！下图中给出了如何实现该功能的示例。



下面的示例用 CheckBounds 功能测试数组的超限问题。CheckBounds 功能允许 A[0] 到 A[7] 元素分配值 TRUE，而不会给 A[10] 分配值，这样可以避免对数组元素的错误操作。



3.2 指针

当程序运行时，变量或功能块地址保存在指针中。指针声明为如下句法形式：
<指针名> : POINTER TO <数据类型 / 功能块>;
指针可指向任何数据类型、功能块和用户定义的数据类型。对地址操作的 ADR 功能，可将变量或功能块的地址指向指针。指针后加内容操作符"^"可取出指针中的数据。
示例：

```
pt:POINTER TO INT;
var_int1:INT := 5;
var_int2:INT;
pt := ADR(var_int1);
var_int2:= pt^;              (* var_int2 的值为 5 *)
```

3.3 牧举

牧举为用户定义的数据类型，并由一组字符串常数组成。这些常数被视为牧举值。牧举值在项目中为全局使用的变量，即使它们在 POU 中为本地声明的变量。创建牧举变量的最好方法是在数据类型对象组织下创建。用关键字 TYPE 开始，END_TYPE 结束。
句法：
TYPE <牧举变量>:(<Enum_0> ,<Enum_1>, ...,<Enum_n>);END_TYPE
牧举变量可以取牧举值中的任何一个值。缺省情况下，第一个牧举值为零，其后依次递增。

示例：

```
TRAFFIC_SIGNAL: (Red, Yellow, Green:=10); (*每个颜色的初始值为 red 0, yellow 1, green 10 *)
TRAFFIC_SIGNAL:=0; (* 交通信号值为 red*)
FOR i:= Red TO Green DO
    i := i + 1;
END_FOR;
```

不能对同一个牧举值多次使用。
示例：

```
TRAFFIC_SIGNAL: (red, yellow, green);
COLOR: (blue, white, red);
```

错误：red 不能对 TRAFFIC_SIGNAL 和 COLOR 变量同时使用。

3.4 结构

结构作为对象在数据类型页中创建。使用 TYPE 关键字开始，END_TYPE 关键字结束。结构声明的句法如下：

```
TYPE <结构名>:
  STRUCT
  <声明变量 1>
   .
```

.
<声明变量 n>
END_STRUCT
END_TYPE
<结构名>是一种类型，在项目中为全程识别，并且可作为标准数据类型使用。允许内嵌结构。唯一的限制是变量不能带地址(不允许用 AT 声明!)。
下例为多边形的结构示例：

```
TYPE Polygonline:
  STRUCT
   Start:      ARRAY [1..2] OF INT;
   Point1:     ARRAY [1..2] OF INT;
   Point2:     ARRAY [1..2] OF INT;
   Point3:     ARRAY [1..2] OF INT;
   Point4:     ARRAY [1..2] OF INT;
   End:        ARRAY [1..2] OF INT;
   END_STRUCT
END_TYPE
```

可以使用下面的句法存取结构中的成员。
<结构_名>.<成员名>
例如：结构名为 “Week”，其中包含一个成员 “Monday”，可以用 Week.Monday 获取该值。

3.5 参考 (别名类型)

可以使用用户定义的参考数据类型，创建已经更名的变量、常数或功能块。在数据类型页中创建参考对象。使用 TYPE 关键字开始，END_TYPE 关键字结束。
句法：

```
TYPE <标识符>: <分配 项>;
END_TYPE
```

示例：

```
TYPE message:STRING[50];
END_TYPE;
```

3.6 替代范围类型

替代范围类型，是对其基本数据类型重新设置范围的一种数据类型。声明可以在数据类型页中进行，但变量也可直接用子范围类型声明：
在数据类型页中声明的句法如下：

```
TYPE <Name> : <Inttype> (<ug>..<og>) END_TYPE;
```

类型	说明
<Name>	必须为有效的 IEC 标识符
<Inttype>	数据类型中的一种。如 SINT,USINT,INT,DINT,UDINT,BYTE,WORD, DWORD(LINT,UINT,LWORD).

<ug>	常数，必须为基本类型，设定的下边界在其类型范围之内。
<og>	常数，必须为基本类型，设定的上边界在其类型范围之内。

示例：

```
TYPE
SubInt : INT (-4095..4095);
END_TYPE
```

用子范围类型直接声明的变量：

```
VAR
    i1 : INT (-4095..4095);
    i2 : INT (5...10):=5;
    ui : UINT (0..10000);
END_VAR
```

如果常数被分配为一个子范围类型(在声明或实现段中)，但其值没有落在该范围之内(例如 i := 5000)，系统将会发出错误信息。

为了在运行期间检查边界范围，推荐使用功能 CheckRangeSigned 或 CheckRangeUnsigned。这样，边界有效性验证可通过合适的方法和手段捕获(例：数值可以截取或设置错误标志)。

示例：

当变量属于有符号子范围类型时(如上例中的 i)，则功能 CheckRangeSigned 被调用；可以通过编程的方法使其值在允许范围之内。

```
FUNCTION CheckRangeSigned : DINT
VAR_INPUT
     value, lower, upper: DINT;
END_VAR

IF (value < lower) THEN
CheckRangeSigned := lower;
ELSIF(value > upper) THEN
CheckRangeSigned := upper;
ELSE
CheckRangeSigned := value;
END_IF
```

为了自动调用功能，功能名 CheckRangeSigned 被指定，并且接口也被指定：返回值和三个 DINT 类型的参数。

当调用时，功能参数如下：

值	分配给范围类型的值
下限	下限边界范围
上限	上限边界范围
返回值	实际分配给范围类型的值

对 i := 10 * y 进行边界有效性验证的示例：

```
i := CheckRangeSigned(10 * y, -4095, 4095);
```

示例中，y 即使是 1000，i 经过上例赋值后其值仍然为 4095。

同样，功能 CheckRangeUnsigned 过程同上：功能名和接口必须正确。

```
FUNCTION CheckRangeUnsigned : UDINT
VAR_INPUT
    value, lower, upper: UDINT;
END_VAR
```

注意：

如果没有 CheckRangeSigned 和 CheckRangeUnsigned，则运行时，没有子类型的类型检验发生，变量 i 可以在 –32768 和 32767 之间取任何值。

注意：

如果功能 CheckRangeSigned 和 CheckRangeUnsigned 按照上例实现，则在 FOR 循环中可对子范围类型连续使用循环。

示例：

```
VAR
   ui : UINT (0..10000);
END_VAR
FOR ui:=0 TO 10000 DO
...
END_FOR
```

FOR 循环不会剩余，因为 ui 不会大于 10000。

象 CheckRange 功能内容一样，当在 FOR 循环中使用增量值时，也应考虑这些问题。

4. 编程方式

4.1 指令表 IL

指令表(IL)由一系列指令组成。每条指令都由一个新行开始，包含一个操作符以及和操作符类型相关的一个或多个操作数，并用逗号分开。在指令前可以有标号，后接一个冒号。

注解必须在一行的最后，指令之间可以插入空行。

示例

标号	操作符	操作数	注解
	LD	17	
	ST	lint	(* comment *)
	GE	5	
	JMPC	next	
	LD	idword	
	EQ	istruct.sdword	
	STN	test	
next:			

在 IL 语言中，可以使用下面的操作符和修饰符。

修饰符：

- JMP、CAL、RET 中带 C：　　指令在预置表达式结果为 TRUE 时执行。
- JMPC、CALC、RETC 中带 N：指令在预置表达式结果为 FALSE 时执行。
- 其它指令中带 N：　　操作数取反 (不是累加器)。

下表为 IL 中全部的操作符及可能的修饰符和相关的意义：

操作符	修饰符	意义
LD	N	使当前结果等于操作数
ST	N	在操作数位置保存当前结果
S		如果当前结果为 TRUE，置位布尔操作数为 TRUE
R		如果当前结果为 TRUE，复位布尔操作数为 FALSE
AND	N, (	位与
OR	N, (	位或
XOR	(	位异或
ADD	(	加
SUB	(	减
MUL	(	乘
DIV	(	除
GT	(	>
EQ	(	=
NE	(	<>
LE	(	<=
LT	(	<
JMP	CN	跳转到标号
CAL	CN	调用功能块
RET	CN	从调用的功能块返回
)		评估括号操作

IL 是一种面向行的语言。

标号	:	操作符/功能	操作数(表)	注释
跳转标号	分隔符	IL 操作符或功能名	用于操作符的零个，一个或多个常数、变量，或用于功能的输入参数，由逗号分隔。	在(*…*)中的注释，可选

通过不同的操作符组修改 CR

影响 CR 数据类型的操作符组	缩写	操作符示例
Create (建立)	C	LD
Process (处理)	P	GT
Leave unchanged (保持不变)	U	ST: JMPC
Set to undefined (设置为未定义的)	-	CAL = 功能块的无条件调用，

带布尔操作数(BOOL 类型)的操作符

操作符		操作符组	描述
LD	LDN	C	装入操作数(操作数的反值)到 CR
AND AND(	ANDN ANDN(	P	操作数(操作数的反值)和 CR 的布尔 AND (“与”运算)
OR OR(	ORN ORN(	P	操作数(操作数的反值)和 CR 的布尔 OR (“或”运算)
XOR XOR(	XORN XORN(	P	操作数(操作数的反值)和 CR 的布尔 XOR (“异或”运算)
ST	STN	U	将 CR 存到操作数
S		U	若 CR = 1，则将操作数设置为 TRUE
R		U	若 CR = 1，则将操作数设置为 FALSE
)		U	结束括号：对递延操作求值

用于类属数据类型(类型 ANY) 操作数的操作符

操作符		操作符组	描述
LD		C	操作数装入 CR
ST		U	将 CR 存储到操作数
ADD	ADD(	P	加操作数，结果存入 CR
SUB	SUB(	P	从 CR 减去操作数，结果存入 CR
MUL	MUL(	P	操作数乘以 CR
DIV	DIV(	P	CR 除以操作数
GT	GT(	P	CR > 操作数 (大于)
GE	GE(	P	CR >= 操作数 (大于或等于)
EQ	EQ(	P	CR = 操作数 (等于)
NE	NE(	P	CR <> 操作数 (不等于)
LE	LE(	P	CR <= 操作数 (小于或等于)
LT	LT(	P	CR < 操作数 (小于)
)		U	结束括号级

跳转或调用

操作符		操作符组	描述
JMP		- 或 U	(无)/有条件跳转到一个跳转标号
JMPC	JMPCN	U	
CAL		- 或 U	(无)/有条件调用一个功能块
CALC	CALCN	U	
RET		- 或 U	(无)/有条件从一个功能或功能块返回
RETC	RETCN	U	
功能名		P	功能调用

使用功能和功能块

A. 调用一个功能

在 IL 语言中，调用一个功能只是简单地写入该功能名即可。随后的实际参数用逗号分隔。这种语法和带有几个操作数的操作符的语法相同。

功能的第一个参数是当前结果(CR)。因此必须正好在功能调用之前将该值装入 CR 中。用于功能调用的第一个操作数实际上是功能的第二个参数，并依次类推。

B. 调用一个功能块

操作符 CAL (或条件调用 CALC 和条件取反调用 CALCN)可以激活一个功能块。

IEC61131-3 描述 IL 语言中给一个 FB 传送参数的三种方法：

1). 使用一个调用，它包括在括号内的实际输入和输出参数的一个列表

2). 在调用 FB 前，装载和保存输入参数

3). 用输入参数作为操作符“隐性地”调用

第三种方法只对标准 FB 有效，不适合用户定义的 FB。

4.2 结构化文本 ST

ST 语言的优点 (与 IL 语言相比较)：

- 编程任务高度压缩化的表达格式，
- 在语句块中清晰的程序结构，
- 控制命令流的强有力结构

这些优点亦带来其本身的缺陷：

- 由于它借助于编译程序自动地执行程序，因此用户不能直接影响其翻译成机器码。
- 高度抽象导致效率降低(通常，编译程序的时间更长且执行速度更慢)

ST 语句

关键字	说明	示例	说明
:=	赋值	d := 10	将右边的一个供计算的数值赋值给左边的标识符
	调用 FB	FB Name(Par1 := 10, Par2 := 20);	调用另一个类型为 FB 的 POU，包括其参数
RETURN	返回	RETURN	脱离当前的 POU 和返回到调用

			POU
IF	选择	IF d < e THEN f :=1; ELSEIF d = e THEN f := 2; ELSE f := 3; END_IF	通过布尔表达式选择替代值
CASE	多重选择	CASE f OF 1: g := 11; 2: g := 12; ELSE g := FunName(); END_CASE	根据表达式”f”的值选择一个语句块
FOR	跌代 (1)	FOR h:=1 TO 10 BY 2 DO F[h/2] := h; END_FOR	一个多循环语句块，带有起始和结束条件以及一个增量值
WHILE	跌代 (2)	WHILE m > 1 DO N := n / 2; END_WHILE	一个多循环语句块，具有在开始端的结束条件
REPEAT	跌代 (3)		一个多循环语句块，具有在结束端的结束条件
EXIT	循环的结束	EXIT;	一个跌代语句的结束条件。
;	空白语句	;;	

ST 语言不包括跳转指令 (GOTO)。

4.3 功能块图 FBD

功能块图(FBD)语言起源于信号处理领域，对信号处理而言，整数与/或浮点数是很重要的。

使用图形化语言 FBD 或 LD 的 POU 表达式包括的部分与文本化语言相同。

1). POU 的引导部分和结束部分
2). 说明部分
3). 代码部分

代码部分 分为若干个网络。网络有助于构造 POU 的控制流。

一个网络包括

1). 网络标号
2). 网络注释
3). 网络图形

4.4 梯形图 LD

梯形图语言 (LD) 源自机电一体化的继电器系统的应用领域，它描述一个 POU 的网络自左至右的能量流。编程语言主要是设计用于处理布尔信号。

梯形图 LD 接点分类：

常开接点	常闭接点	上升沿接点	下降沿接点

梯形图 LD 线圈分类：

线圈 --( )--	线圈的取反 --(/)--	置位 (锁存) 线圈 --(S)--	复位(解除锁存)线圈 --(R)--
保持 (记忆) 线圈 --(M)--	置位保持(记忆)线圈 --(SM)--	复位保持(记忆)线圈 --(RM)--	
上升沿线圈 --(P)--		下降沿线圈 --(N)--	

梯形图 LD 执行控制分类：

无条件返回	条件返回	无条件跳转	条件跳转

调用功能和功能块

5. CoDeSys 中全部运算符及功能名

ST 中操作符	IL 中操作符	IL 中的修饰符	意义
‘			字符串分界符(例如：’string1’)
[..]			数组大小范围(例如：ARRAY[0..3] OF INT
:			操作数和类型声明之间的分界符(例如：var1 : INT;)
^			指针引用 (例如：pointer1^)
	LD var1	N	装入 var1 值到缓冲器中
:=	ST var1	N	存入实际结果到 var1 中
	S boolvar		当实际结果为 TRUE 时，设置布尔变量 boolvar 为 TRUE
	R boolvar		当实际结果为 TRUE 时，设置布尔变量 boolvar 为 FALSE
	JMP marke	CN	跳转到标号
<程序名>	CAL prog1	CN	调用程序 prog1
<句柄名>	CAL inst1	CN	调用功能块句柄 inst1
<功能名>(vx,vy,..)	<功能名>(vx,vy,..)	CN	调用功能 fctname 并传送变量 vx,vy
RETURN	RET	CN	离开 POU 并返回到调用者
	(		括号之后的值作为操作数处理，不执行括号之前的运算。
	)		执行括号返回的操作运算
AND	AND	N, (	位与
OR	OR	N, (	位或
XOR	XOR	N, (	位异或
NOT	NOT		位取反
+	ADD	(	加
-	SUB	(	减
*	MUL	(	乘
/	DIV	(	除
>	GT	(	大于
>=	GE	(	大于或等于
=	EQ	(	等于
<	LT	(	小于
<>	NE	(	不等于
<=	LE	(	小于或等于
MOD(in)	MOD		取模除
INDEXOF(in)	INDEXOF		POU 内部索引 in1; [INT]
SIZEOF(in)	SIZEOF		数据类型 in 所需字节数
SHL(K,in)	SHL		in 数据向左位移 K 位
SHR(K,in)	SHR		in 数据向右位移 K 位
ROL(K,in)	ROL		in 数据向左循环位移 K 位
ROR(K,in)	ROR		in 数据向右循环位移 K 位
SEL(G,in0,in1)	SEL		选择器，G 为 FALSE 选 in0 G 为 TRUE 选 in1

MAX(in0,in1)	MAX		取极大值
MIN(in0,in1)	MIN		取极小值
LIMIT(Min,in,Max)	LIMIT		取限幅值，当 in 超过限幅值时，取 Min 或 Max 值
MUX(K,in0,..in_n)	MUX		多值选择器 (in0,..in_n)
ADR(in)	ADR		取操作数的地址到 [DWORD] 中
BOOL_TO_<type>(in)	BOOL_TO_<type>		布尔操作数类型转换
<type>_TO_BOOL(in)	<type>_TO_BOOL		类型转换到布尔值
INT_TO_<type>(in)	INT_TO_<type>		INT 转换为其他成员类型
REAL_TO_<type>(in)	REAL_TO_<type>		REAL 转换为其他成员类型
LREAL_TO_<type>(in)	LREAL_TO_<type>		LREAL 转换位其他成员类型
TIME_TO_<type>(in)	TIME_TO_<type>		TIME 转换为其他成员类型
TOD_TO_<type>(in)	TOD_TO_<type>		TOD 转换为其他成员类型
DATE_TO_<type>(in)	DATE_TO_<type>		DATE 转换为其他成员类型
DT_TO_<type>(in)	DT_TO_<type>		DT 转换为其他成员类型
STRING_TO_<type>(in)	STRING_TO_<type>		STRING 转换为其他成员类型
TRUNC(in)	TRUNC		REAL 向 INT 转换
ABS(in)	ABS		in 操作数取绝对值
SQRT(in)	SQRT		in 操作数取平方根
LN(in)	LN		in 操作数取自然对数
LOG(in)	LOG		in 操作数取底数为 10 的对数
EXP(in)	EXP		in 操作数进行指数运算 (e^x)
SIN(in)	SIN		in 操作数进行正弦运算
COS(in)	COS		in 操作数进行余弦运算
TAN(in)	TAN		in 操作数进行正切运算
ASIN(in)	ASIN		in 操作数进行反正弦运算
ACOS(in)	ACOS		in 操作数进行反余弦运算
ATAN(in)	ATAN		in 操作数进行反正切运算
EXPT(in,expt)	EXPT expt		in 为底数，expt 为指数运算
LEN(in)	LEN		in 操作数取字符串长度
LEFT(str,size)	LEFT		从 str 左边取 size 个字符串
RIGHT(str,size)	RIGHT		从 str 右边取 size 个字符串
MID(str,size,pos)	MID		从 str 的 pos 位置取 size 个字符串
CONCAT(str1,str2)	CONCAT		合并 str1 和 str2 两个字符串
INSERT(str1,str2,pos)	INSERT		在 str2 的 pos 处插入 str1 字符串
DELETE(str1,len,pos)	DELETE		在 str1 的 pos 处删除 len 个字符串
REPLACE(str1,str2 len,pos)	REPLACE		在 str1 的 pos 处替换 str2 中的 len 个

			字符串
FIND(str1,str2)	FIND		在 str1 中寻找 str2 字符串
SR	SR		置位优先的 SR 触发器
RS	RS		复位优先的 RS 触发器
SEMA	SEMA		软件信号器(可中断)
R_TRIG	R_TRIG		上升沿检测
F_TRIG	F_TRIG		下降沿检测
CTU	CTU		向上计数
CTD	CTD		向下计数
CTUD	CTUD		向上和向下计数
TP	TP		脉冲计时器
TON	TON		延时开计时器
TOF	TOF		延时断计时器