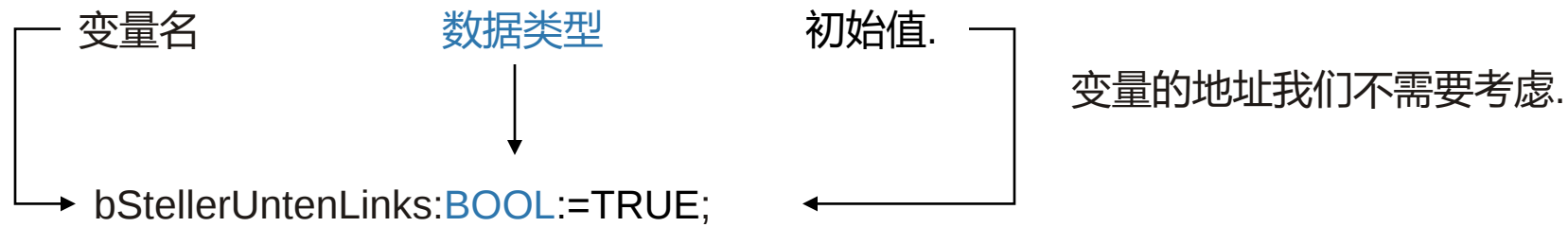


TwinCAT 3: eXtended Automation



一个变量名包含了一个数据的类型.
变量名是所要声明数据的地址.
变量在运行过程中值是可以变化的.



- 变量的首字符可以是
- 字母 (abcd....) 或者下划线 (_)
- 后面可以由
- 数字 (123...) , 字母 (abc...) 和下划线 (_) 组成。
- 变量不区分大小写 (abc与ABC表示同一个变量)
- 变量不可以有特殊字符 (例如: ! , " , \$等等)
- 变量名中不可以存在空格 (a b) , 连续的下划线 (a____b) 。

在IEC61131-3中的关键字不可以做为变量名.

例如:

逻辑运算关键字: AND, OR, NOT...

数据类型关键字: BOOL, INT, REAL...

类型和结构定义关键字: TYPE, STRUCT

块或程序的关键字: FUNCTION, FUNCTION_BLOCK, PROGRAM

不能和功能块名一样

关键字在程序中会自动以蓝色大写出现

变量声明示例:

nInput	and	_inout	1light
Light1	dint	P out	Point_1
net@163	auto_matic	tc_3_PLC	PLC___C

变量声明示例:

nInput

Light1

net@163

and

dint

auto_matic

_inout

P out

tc_3_PLC

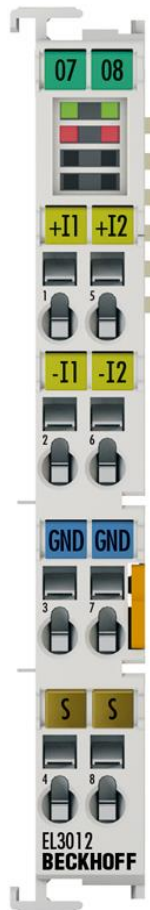
1light

Point_1

PLC___C

Type	Lower	Upper	Size	Prefix
BOOL	FALSE	TRUE	1BIT	x b
BYTE	0	255	8 BIT	n
WORD	0	65535	16 BIT	n
DWORD	0	4294 967 295	32 BIT	n

Type	Lower	Upper	Size	Prefix
SINT	-127	127	8 Bit	n
USINT	0	255	8 BIT	n
INT	-32768	32767	16 BIT	n
UINT	0	65535	16 BIT	n
DINT	-2147 483 648	2147 483 647	32 BIT	n
UDINT	0	4294 967 295	32 BIT	n
LINT	-2^{63}	$2^{63} - 1$	64 BIT	n
ULINT	0	$2^{64} - 1$	64 BIT	n



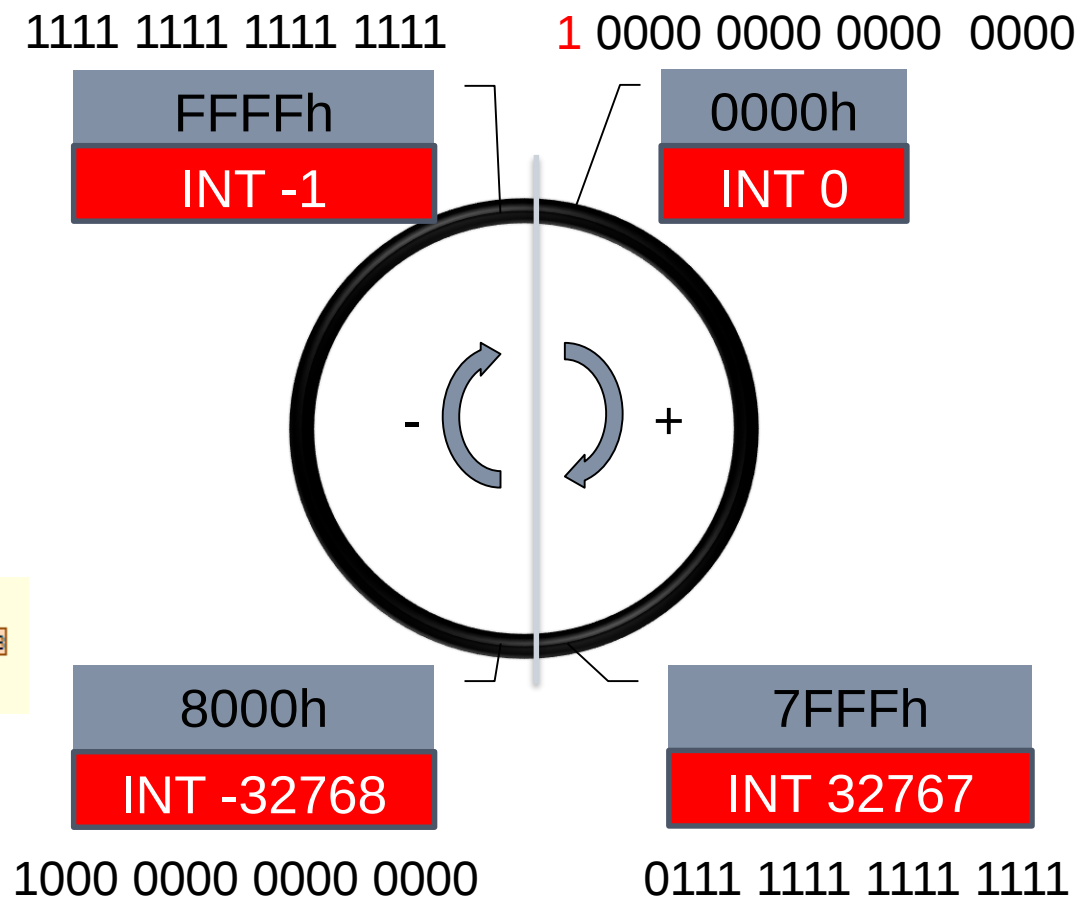
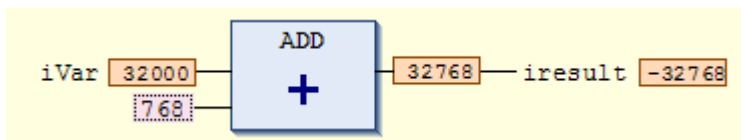
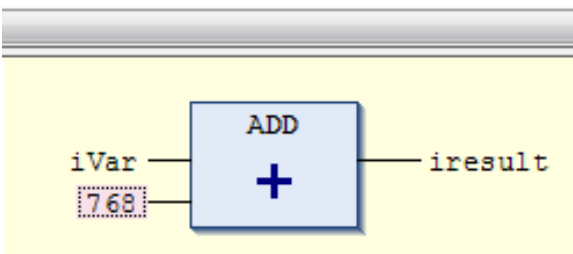
EL3102
2-channel analog input terminals
-10...+10 V,
(模数转换)

根据输入信号值的范围声明为INT型变量.

Input signal = Value	Value	
EL310x	Decimal	Hexadecimal
10 V	32767	0x7FFF
5 V	16383	0x3FFF
0 V	0	0x0000
-5 V	-16383	0xC001
-10 V	-32768	0x8000

进行数据操作的时候注意溢出问题

```
VAR
  iVar: INT:=32000;
  iresult: INT;
END_VAR
```



Type	Lower	Upper	Size	Prefix	
REAL			4 Byte	f	
LREAL			8 Byte	f	

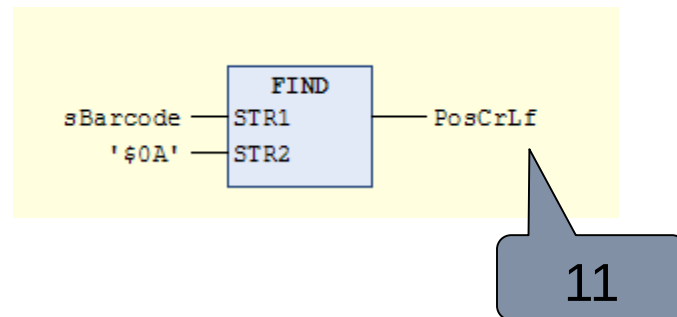
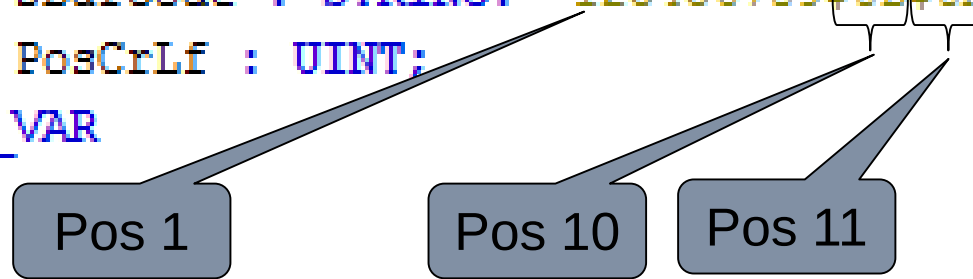
Type	Description	Example	Size	Prefix
STRING	String in ASCII code. Standard length 80 characters. Maximum length 255. Strings are zero-terminated	'1234ABCDE'	80 +1	s
		'ABCDE\$R\$L'		
		'ABCDE\$0D\$0A'		

String length specifications

Example declaration	Assignment	Result SIZEOF	Result LEN
sVar : STRING;	sVar:='ABC';	81	3
sVar1 :STRING(1);	sVar := 'X';	2	1
sVar: STRING(255);	sVar:='ABC';	256	3

Constants		
\$<2 Hex values>		ASCII Code
\$0D		CR
\$R	\$r	CR
\$L	\$l	Line Feed
\$N	\$n	New Line
\$T	\$t	Tab

```
VAR  
    sBarcode : STRING:='123456789$0D$0A';  
    PosCrLf : UINT;  
END_VAR
```



Type	Description	Example	Description	Pre fix
WSTRING	String in Unicode format	“Обучение”	Level 0,Block 0x0400-0x4FFF Cyrillic	ws
		“培训、讲座、研讨会”		
		“Training, seminar”	Level 0 Block 0x0000- 0x007F Basic Latin	

Type	Lower	Upper	Size	Prefix
TIME_OF_DAY	TOD#0:0:0	TOD#23:59:59	32 Bit	tod
DATE	D#1970-01-01	D#2106-02-07	32 Bit	date
DATE_AND_TIME	DT#1970-01-01-00:00:00	DT#2106-02-07-06:28:15	32 Bit	dt
TIME	T#0S	T#49D17H2M47S295MS	32BIT	tim

数据单位表示形式

BECKHOFF

Variable Type		Examples			
BOOL		TRUE	2#1	16#1	1
		FALSE	2#0	16#0	0
WORD, DWORD		2#1010111111111110		16#AFFE	45054
INT		2#10000000000000001		16#8001	-32768
TIME		t#1h		t#60m	t#3600000ms
d	day			t#12h	t#43200000ms
h	hours				
m	min				
s	sec	t#30m18s90ms		t#0.505025h	t#1818090ms
ms	ms				
REAL		0.3333		3.333e-1	

注释是以 (*) 开始,以 *) 结束,可以放在程序中除字符串当中的任何地方.

```
(*The Beginning*)  
bStart AT %IX0.0 :BOOL;(*bool*)
```

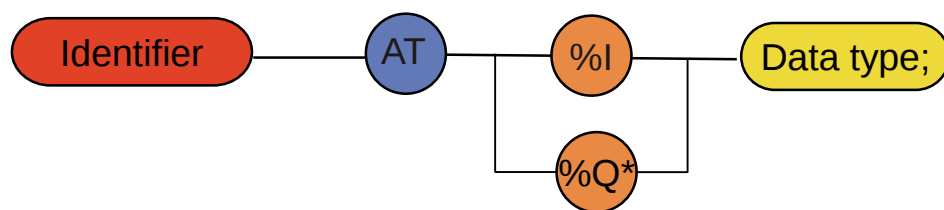
```
(*WORD*) TemK1 AT %IW10 (*Byte 10-11*) :WORD;
```

单行注释可以由//表示.

```
bStart AT %IX0.0 :BOOL; //Single Line
```

变量名在特殊情况下是可以与地址进行连接的

对于确定输入,输出固定地址变量,可以使用`AT%I*`和`AT%Q*`进行变量类型的声明.



`VAR_GLOBAL`

`K12531Velocity AT%Q* :INT;`

`K12531Ctrl AT%Q* :BYTE;`

`K12531ExtCtrl AT%Q* :WORD;`

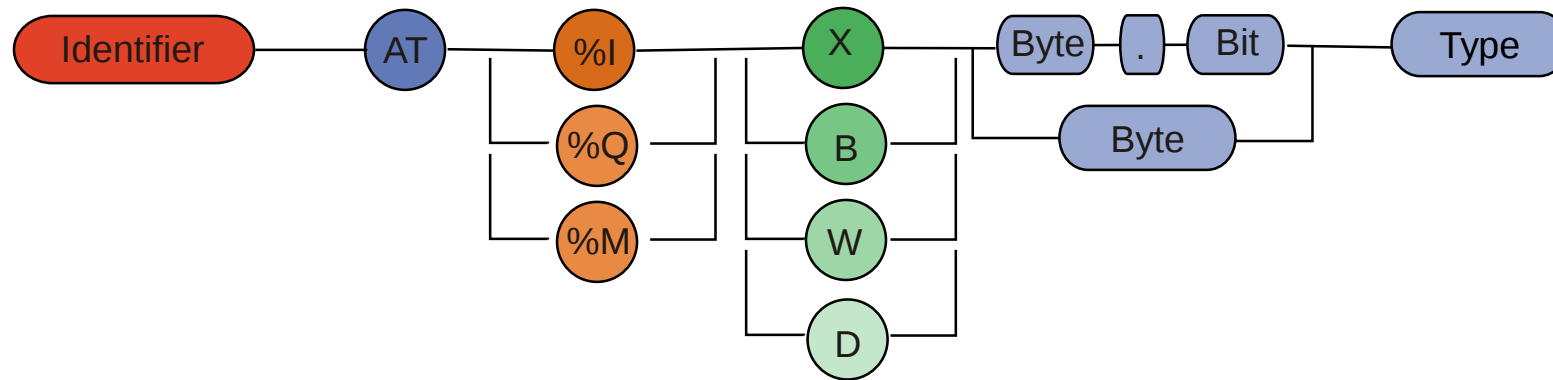
`K12531Position AT%I* :UINT;`

`K12531Status AT%I* :BYTE;`

`K12531ExtStatus AT%I* :WORD;`

`END_VAR`

对于知道完整地址的变量:



StrWord AT %MW10 : WORD;

局部变量只能用在我们声明变量所在的块或程序内.

Keywords
VAR ..
END_VAR
VAR_INPUT ..
END_VAR
VAR_IN_OUT ..
END_VAR
VAR_OUTPUT ..
END_VAR

全局变量可以用在一个项目的多个块中.

Keywords
VAR_GLOBAL ..
END_VAR
VAR_CONFIG ..
END_VAR

Project machine

```
VAR_GLOBAL  
Var1:WORD ;  
END_VAR
```

Example
name:
Gvl1

```
PROGRAM A  
VAR  
  Var1 :WORD;  
END_VAR
```

```
LD Var1  
LD Gvl1.Var1
```

当全局变量与局部变量重名的时候,例如左边这种情况.全局变量中的Var1则会**进行处理**.

对于全局变量的是重复变量访问则通过**命名空间**与变量名的组合进行访问.

例如:左边的全局变量是在Gvl1中的,则访问路径为:Gvl1.Var1.而局部变量则是Var1.

■

■

通过固定地址读取变量.

BECKHOFF

在程序B中声明的固定地址变量.在程序A中可以通过固定地址读取.他们是同一个值.

Project machine

PROGRAM A

VAR

END_VAR

LD %MB2

▪

▪

PROGRAM B

VAR

locVar AT%MB2:WORD;

END_VAR

▪

▪

▪

▪

在**PERSISTENT**中声明的变量在**PLC**关机时会保存其数值,在**PLC**上电后会读取已保存的数值.

```
VAR_GLOBAL PERSISTENT
```

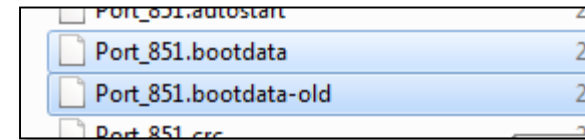
```
    Stueckzaehler: UDINT;
```

```
END_VAR
```

```
VAR PERSISTENT
```

```
    Meldungstext: STRING;
```

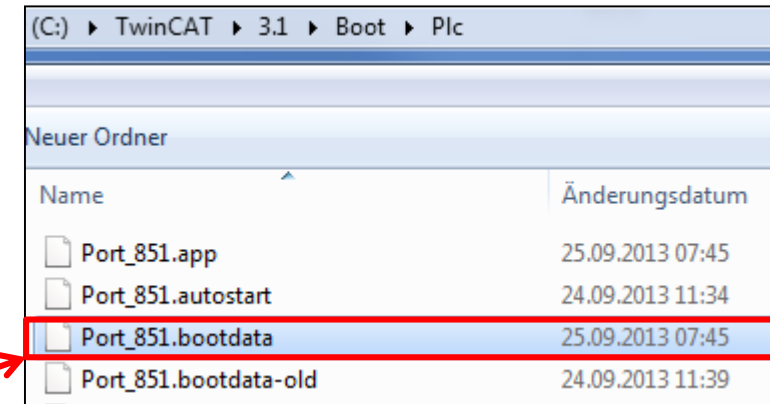
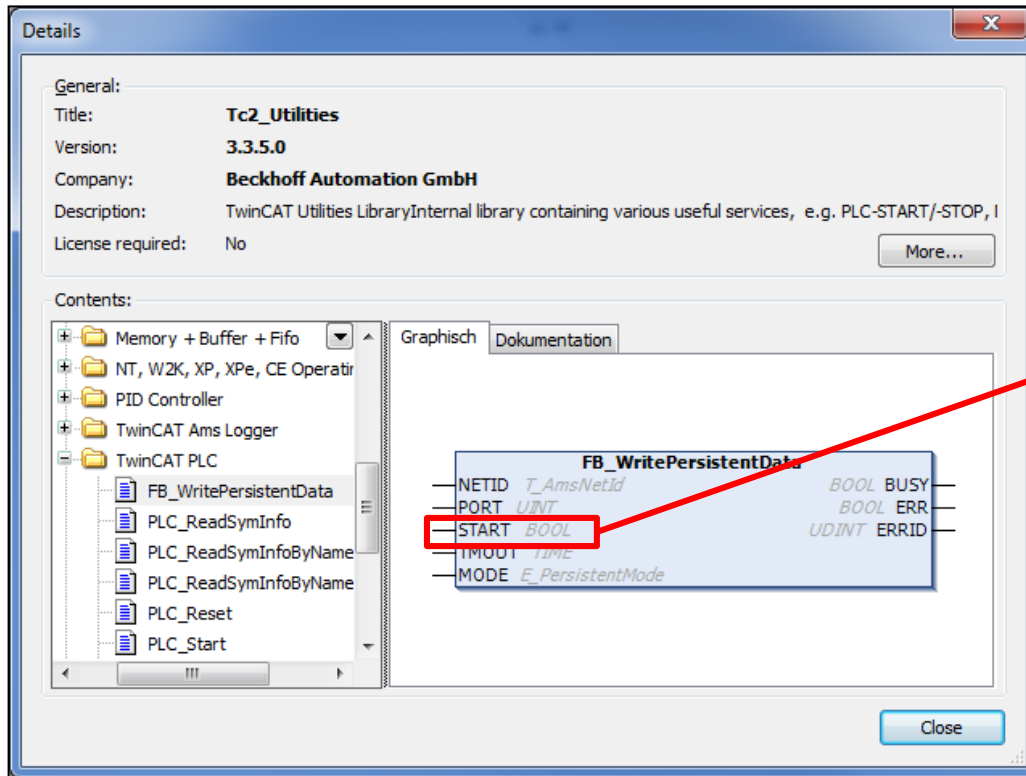
```
END_VAR
```



可以通过功能块对PERSISTENT型变量进行写操作.

- 注意:调用写掉电保持数据功能块的时候, 不要更改变量的值

FB_WritePersistentData
Tc2_Uutilities.LIB



初始化变量值, 在PLC启动和复位时会对变量进行赋予预设定的值.

VAR

AccelerationTime : TIME := T#3s200ms;

END_VAR

只读:

VAR_GLOBAL CONSTANT

pi:REAL:=3.141592654;

Global

常量.

END_VAR

VAR CONSTANT

pi:REAL:=3.141592654;

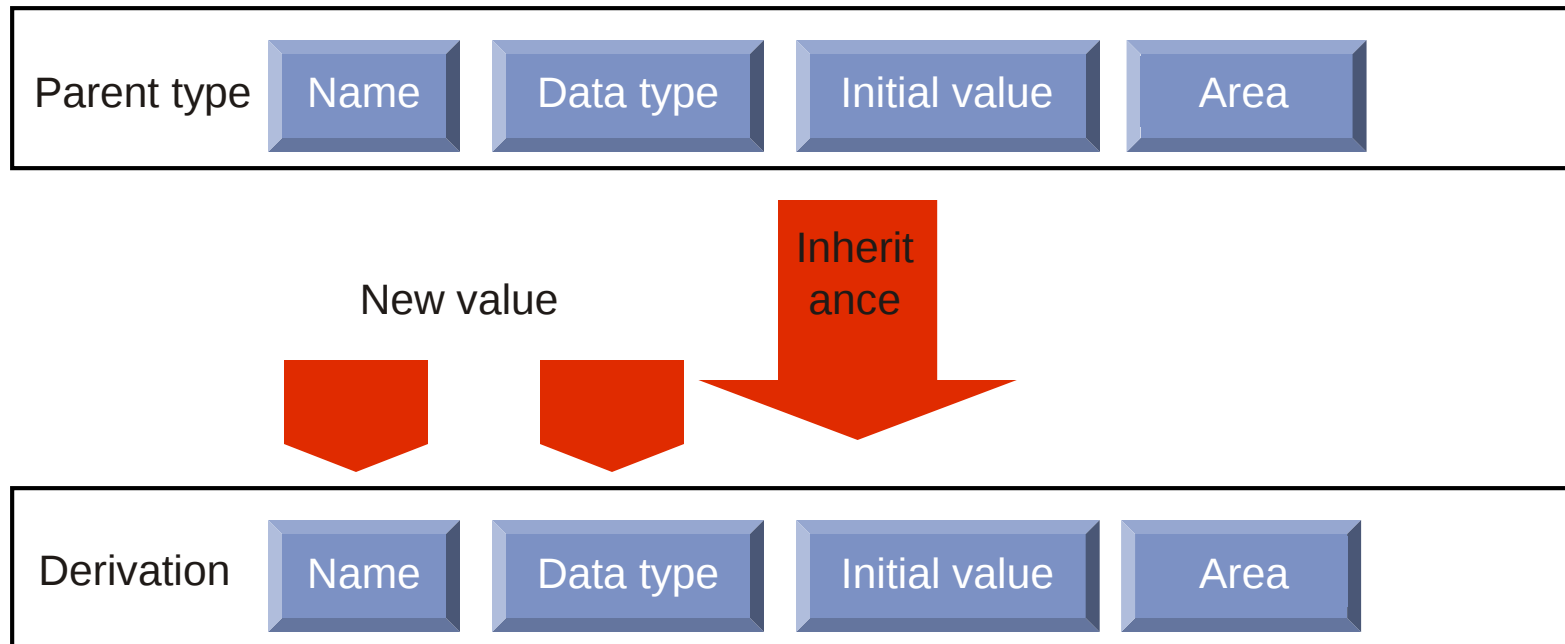
Also locally possible

END_VAR

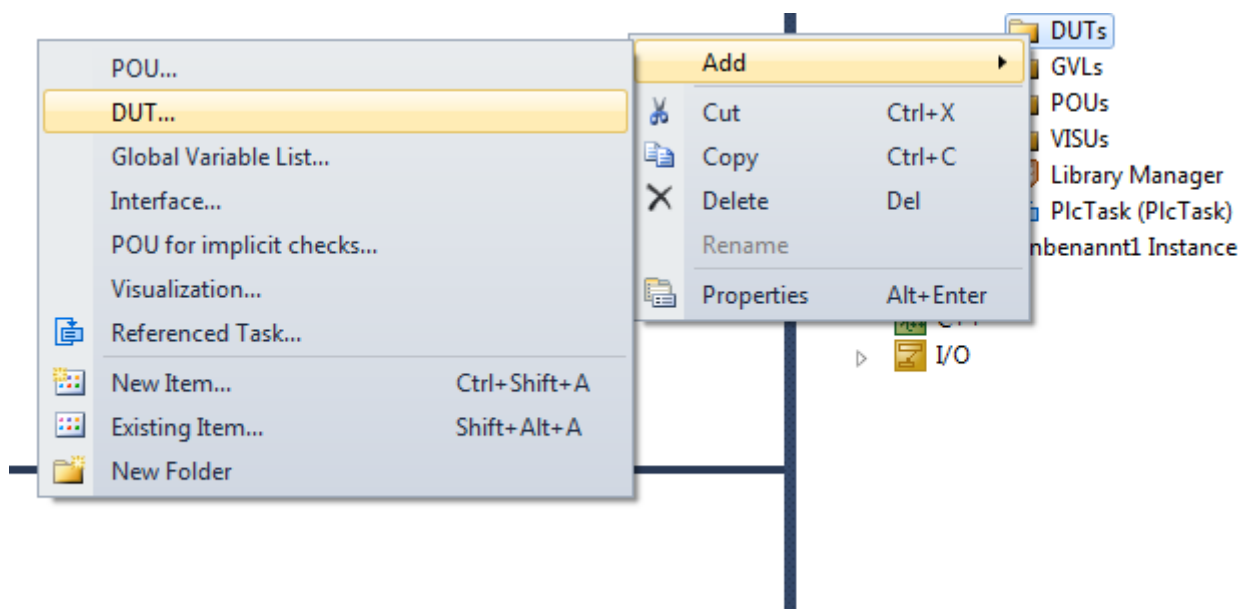
使用者可以通过基本的数据类型或者已经创建的数据类型来定义出自己的类型.新创建的类型是在整个项目中找到的.
自定义类型以**TYPE** 开始以 **END_TYPE** 结束.

TYPE

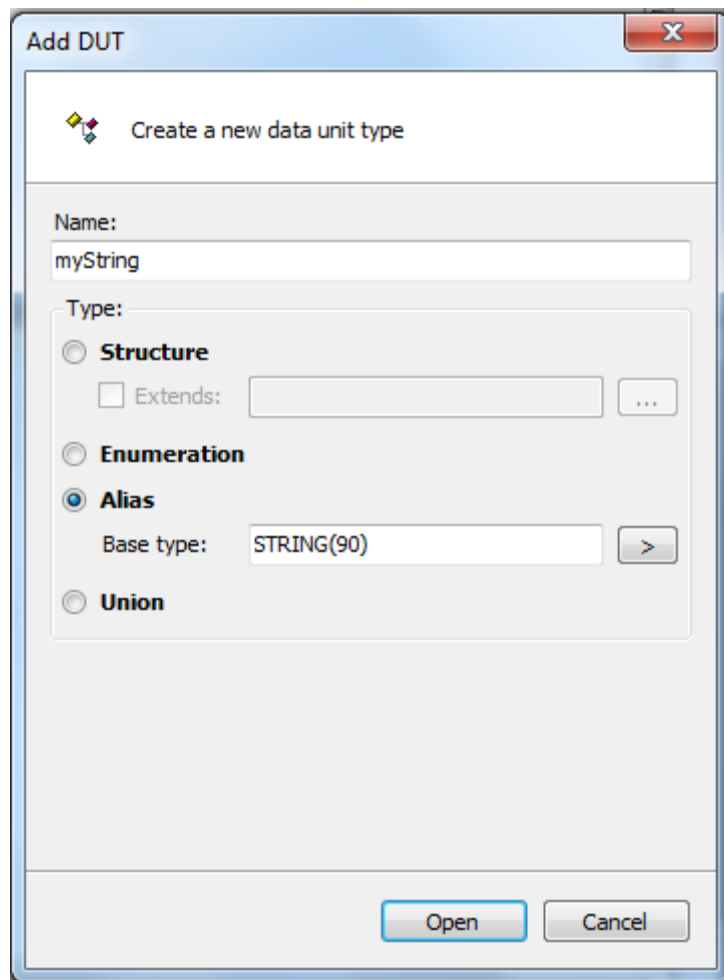
END_TYPE



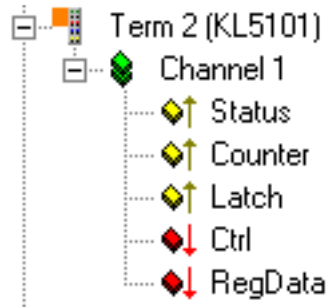
如果创建了一个字符串类型的别名,那么只需修改自定义类型里面的数据长度即可.



如果创建了一个字符串类型的别名,对于相同类型使用只需修改自定义类型里面的数据即可.



Example: KL5101 编码器端子.



结构体是自定义类型.
结构体类型帮助使用者方便的管理过程数据. 并且结构体方便封装数据传送到功能块. 结构体类型可以用于单个变量.

```
TYPE KL5101_IN :  
STRUCT  
    State:UINT;  
    Counter:UINT;  
    Latch:UINT;  
END_STRUCT  
END_TYPE
```

```
TYPE KL5101_OUT :  
STRUCT  
    Ctrl:USINT;  
    RegData:UINT;  
END_STRUCT  
END_TYPE
```

枚举类型是由多个字符串常量组成的自定义数据类型.这些常量是枚举变量的值.枚举量的值在整个项目中都是确定的.对于只有几个确定值的变量最好声明枚举变量.枚举变量声明以关键字 **TYPE** 开始以关键字 **END_TYPE** 结束

Syntax:

```
TYPE <Bezeichner>:(<Enum_0> ,<Enum_1>, ...,<Enum_n>);  
END_TYPE
```

Example:

```
TYPE enmu1:(Mo, Di, Mi, Dn, Fr, Sa, So:=10);(*Mo = 0 Di = 1..  
.. Sa = 6 So = 10*)
```

END_TYPE

```
TYPE enmu2:(Up, Dn);(*Up = 0 Dn = 1*)  
END_TYPE
```



由于枚举名字区别相同的枚举量可以用两次.例如 Example: Woche.Dn
Richtung.Dn

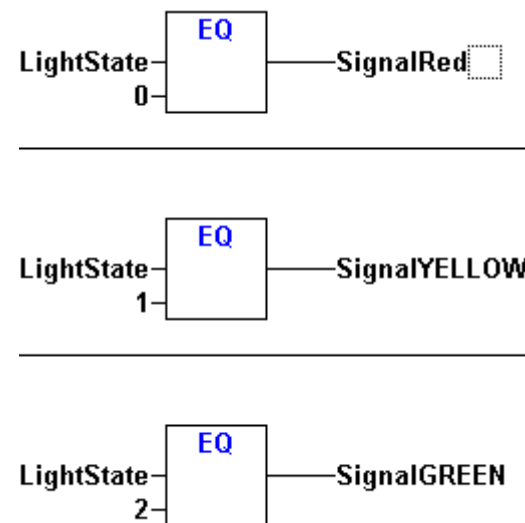
不使枚举类型

Declaration

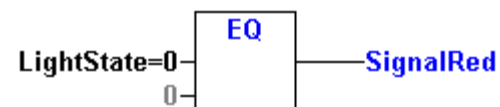
```
VAR
  LightState :INT;    (*0:RED, 1: YELLOW, 2:GREEN*)

  SignalRed AT %Q*: BOOL;
  SignalYELLOW AT %Q*: BOOL;
  SignalGREEN AT %Q*: BOOL;
END_VAR
```

Use



Online



使用枚举类型

Type

```

TYPE TrafficLight:
(
  RED,      (# 0 #)
  YELLOW,   (# 1 #)
  GREEN     (# 2 #)
);
END_TYPE
    
```

Declaration

```

VAR
  LightState :TrafficLight;

  SignalRed  AT %Q*: BOOL;
  SignalYELLOW AT %Q*: BOOL;
  SignalGREEN AT %Q*: BOOL;
END_VAR
    
```

Use

LightState
RED

EQ

SignalRed

LightState
YELLOW

EQ

SignalYELLOW

LightState
GREEN

EQ

SignalGREEN

Online

LightState=RED
RED

EQ

SignalRed

别名的目的声明出变量,常量,功能块的备用名声,方便使用.

以关键字**TYPE** 开始以关键字**END_TYPE** 结束.

Syntax:

TYPE

<Bezeichner>:<Zuweisungsausdruck>;

END_TYPE

Type

NetID

TYPE

NetID:STRING(23);

END_TYPE

Declaration

VAR

NetIDCX1 : NetID:='172.16.17.100.1.1';

NetIDPC1 : NetID:='172.16.17.100.1.2';

NetIDPBX : NetID:='172.16.17.101.1.1';

END_VAR

如果创建了一个字符串类型的别名,对于相同类型使用只需修改自定义类型里面的数据即可.

```
1 TYPE myString : STRING(90); END_TYPE  
2
```

Globale_Variablen

```
0001 VAR_GLOBAL  
0002     StrVar: myString;  
0003 END_VAR
```

fb1 (FB-ST)

```
0001 FUNCTION_BLOCK fb1  
0002 VAR  
0003 END_VAR  
0004 VAR_IN_OUT  
0005     strln: myString;  
0006 END_VAR  
0007  
0001 (* Insert Code... *)
```

fct2 (FUN-ST)

```
0001 FUNCTION fct2 : BOOL  
0002 VAR_INPUT  
0003 END_VAR  
0004 VAR_IN_OUT  
0005     strln: myString;  
0006 END_VAR  
0007  
0001 (* Insert Code*)
```

MAIN (PRG-FUP)

```
0001 PROGRAM MAIN  
0002 VAR  
0003     fb1_1: fb1;  
0004 END_VAR  
0005  
0001  
0002
```

```
graph TD
    fb1_1[fb1_1] -- StrVar-strln --> fct2[fct2]
```

- 共用体,变量共用内存.

```
TYPE unionStringArray :          VAR
UNION
    sVar : STRING;                unionStringKonvert : unionStringArray;
    aVar : ARRAY[0..80] OF BYTE;  END_VAR
END_UNION
END_TYPE
```

unionStringKonvert		unionStringArray	
sVar	STRING		'ABCDE\$R\$N'
aVar	ARRAY [0..80] OF B...		
aVar[0]	BYTE	16#41	← A
aVar[1]	BYTE	16#42	B
aVar[2]	BYTE	16#43	C
aVar[3]	BYTE	16#44	D
aVar[4]	BYTE	16#45	E
aVar[5]	BYTE	16#0D	← \$
aVar[6]	BYTE	16#0A	← R
aVar[7]	BYTE	16#00	
aVar[8]	BYTE	16#00	← N

A
B
C
D
E
\$
R
N

- 使用Union代替%MB定义的便利.

```
VAR
    //----mask String / Array
    sVar AT%MB100 : STRING;
    aVar AT%MB100 :ARRAY[0..80] OF BYTE;
END_VAR
```

可以.

好.使用者不用内存管理地址.

```
TYPE unionStringArray :
    UNION
        sVar : STRING;
        aVar : ARRAY[0..80] OF BYTE;
    END_UNION
END_TYPE

VAR
    unionStringKonvert : unionStringArray;
END_VAR
```

数组表示一系列,或者一个区域的数据. 在数组中所有元素都是同一种类型.
数组也可以由使用者自己的数据类型组成.
IEC支持一维,二维,三维数组.

VAR

Feld_1 : ARRAY[0..9] OF BYTE;

1-维

Feld_2 : ARRAY[0..9, 0..1] OF UINT;

2-维

Feld_3 : ARRAY[0..9, 0..1, 0..1] OF DINT;

3-维

END_VAR

使用者可以直接定义数组的内存地址.

VAR

Feld_1 AT%MB100: ARRAY[1..10] OF BYTE;

END_VAR

访问数组中的元素

Feld_1[2] := 120; (* Expliziter Zugriff*)

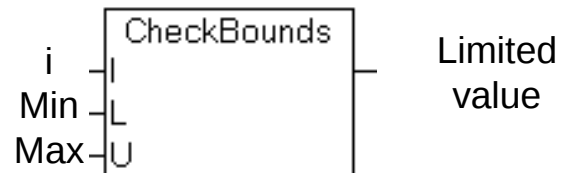
Feld_2[i,j] := EXPT(i,j); (*Indizierter Zugriff*)

```
VAR  
  Feld_1 :ARRAY[1..10] OF BYTE;  
  Feld_2 :ARRAY[1..10, 2..5] OF UINT;  
END_VAR
```

i:= 9		9
Feld_1[i+2] := 120;		
▪		
▪		
Feld_1[9];	0	
Feld_2[1,2];		120

在PLC运行时可以通过功能块进行
监视

这个功能块能够避免下标越界的情
况发生



```
FUNCTION CheckBounds :DINT
```

```
VAR_INPUT
```

```
    I,L,U : DINT;
```

```
END_VAR
```

```
IF I < L THEN
```

```
Error case           CheckBounds := L;
```

```
ELSIF I > U THEN
```

```
Error case           CheckBounds := U;
```

```
ELSE
```

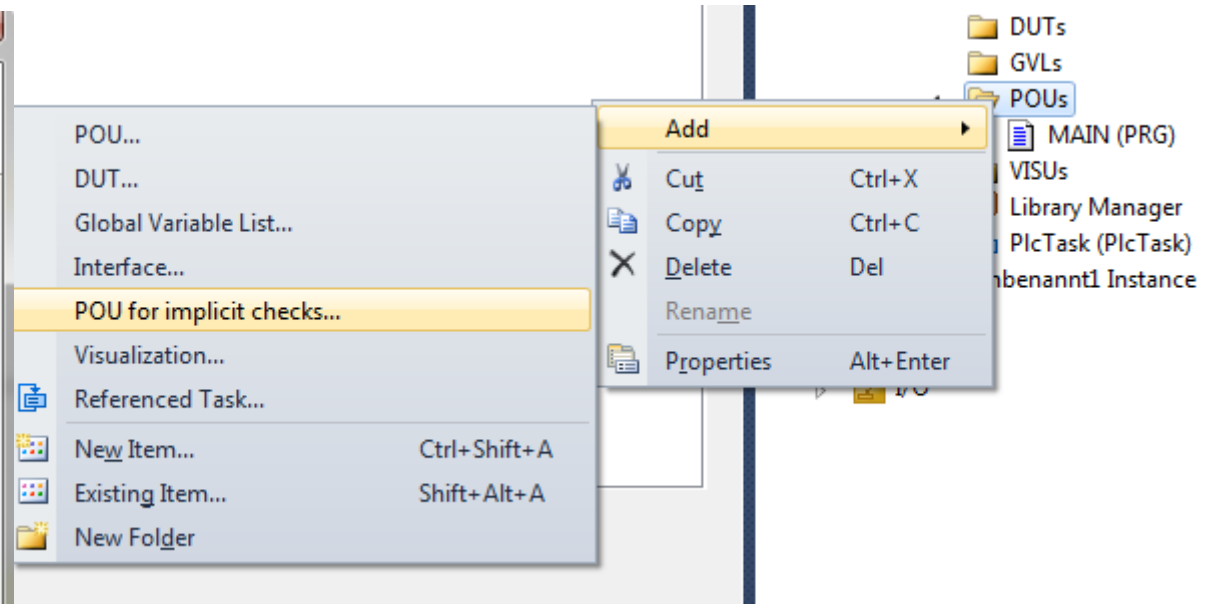
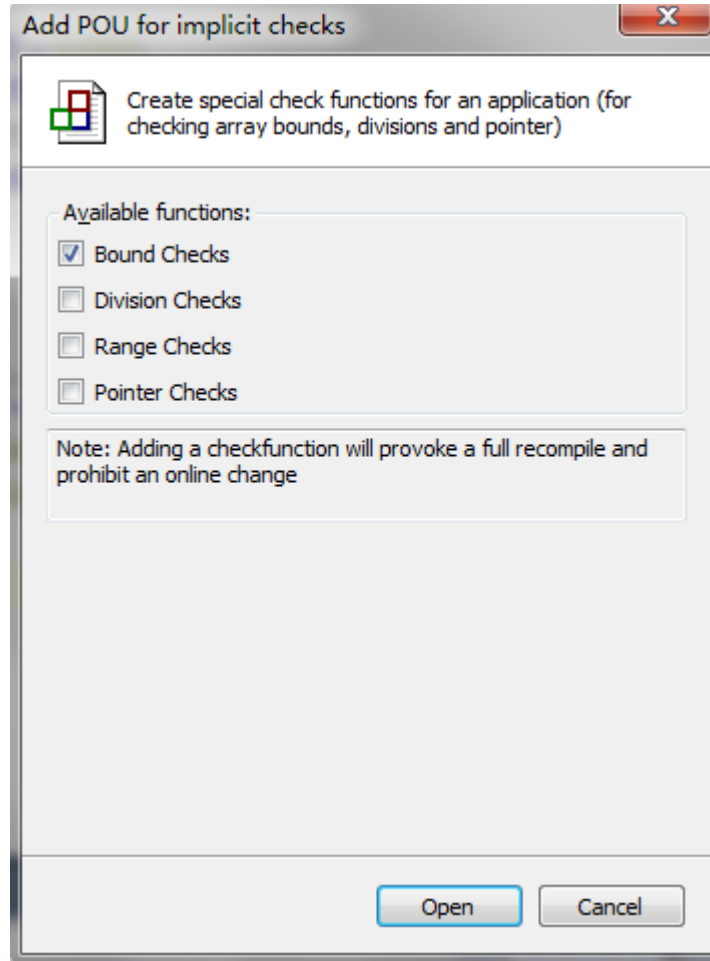
```
"OK" case            CheckBounds := I;
```

```
END_IF
```

添加CheckBounds (FUN)

BECKHOFF

添加CheckBounds (FUN) :



添加CheckBounds (FUN)

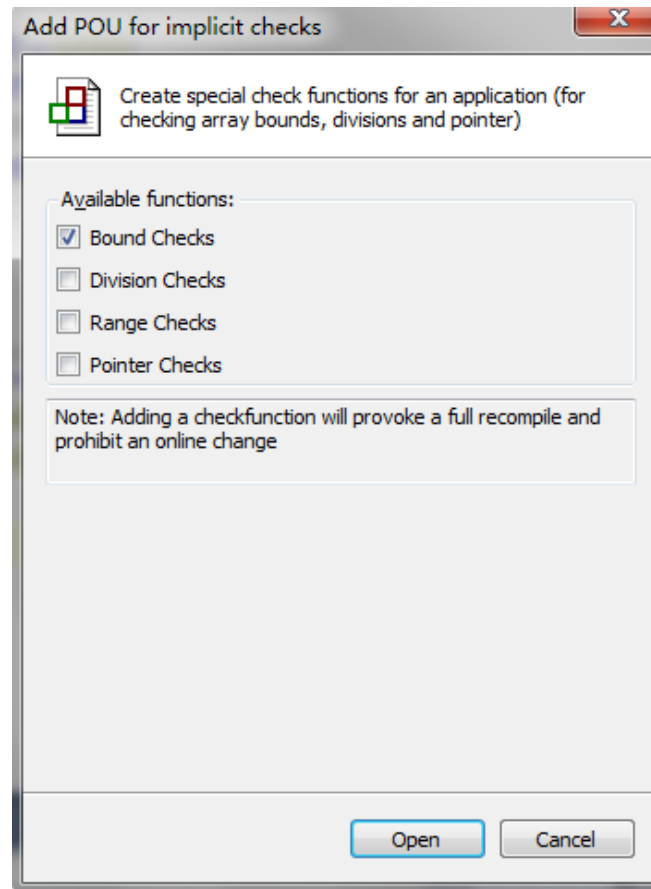
BECKHOFF

CheckBounds

```
1 // Implicitly generated code : DO NOT EDIT
2 FUNCTION CheckBounds : DINT
3 VAR_INPUT
4     index, lower, upper: DINT;
5 END_VAR
6
```

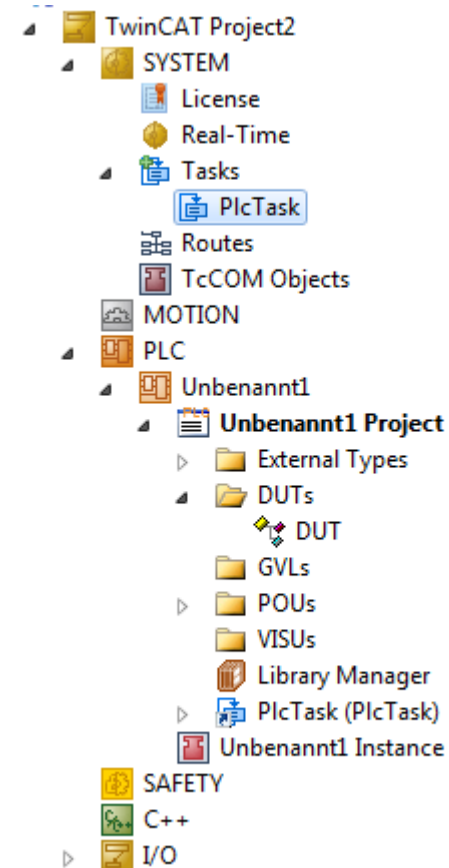
```
1 // Implicitly generated code : Only an Implementation suggestion
2 IF index < lower THEN
3     CheckBounds := lower;
4 ELSIF index > upper THEN
5     CheckBounds := upper;
6 ELSE
7     CheckBounds := index;
8 END_IF
9
```

以下是其它checker functions
TwinCAT 3:



在IEC61131-3 的POU (PROGRAM ORGANISATION UNIT)中包含了三种类型的块:

- Program
- Function Block
- Function



Program PRG

- 由任务调用 (一个program能够调用另一个program)
- 可以调用: FBs, functions, (programs)
- 局部变量: 静态, 即局部变量的数据可以在下个周期使用.
- 输入: 默认是没有输入类型的变量的, 在VAR_INPUT是可以定义输入变量的.
- 输出: 默认是没有输出的变量的, 在VAR_OUTPUT 是可以定义输出变量的.
- 输入输出型变量是可以在VAR_IN_OUT定义的.
- 监视: 在线控制状态下变量的值是实时可见的.

Function block FB

- 由programs或者其它的FBs调用
- 可以调用: FBs, functions,
- 局部变量: 静态, 即局部变量数据可以在下个运行周期使用.
多个功能块中,每个FB都有自己的局部变量.
- 输入: 0,1,2,3...VAR_INPUT
- 输出: 0,1,2,3.. VAR_OUTPUT
- 输入输出: 0,1,2,3.. VAR_IN_OUT
- 监视:在线控制状态下对于每个指定的功能块,局部变量是可见的.

Function: FC

- 由programs, function blocks和其它的functions调用
- 可以调用: Functions
- 局部变量: 临时的, 即局部变量数据仅在当前function执行时可以使用. 之后这个数据区就被用于其它内容.
- 输入: 1,2,3..... VAR_INPUT
- 输出: 只有1个!
- 输入输出: 1,2,3..... VAR_IN_OUT ,
- 监视: 在线状态下仅能看到“???”. 使用断点.

条件语句,满足条件执行内部语句

Keywords:

IF Condition THEN

Instruction block;

ELSIF Condition THEN

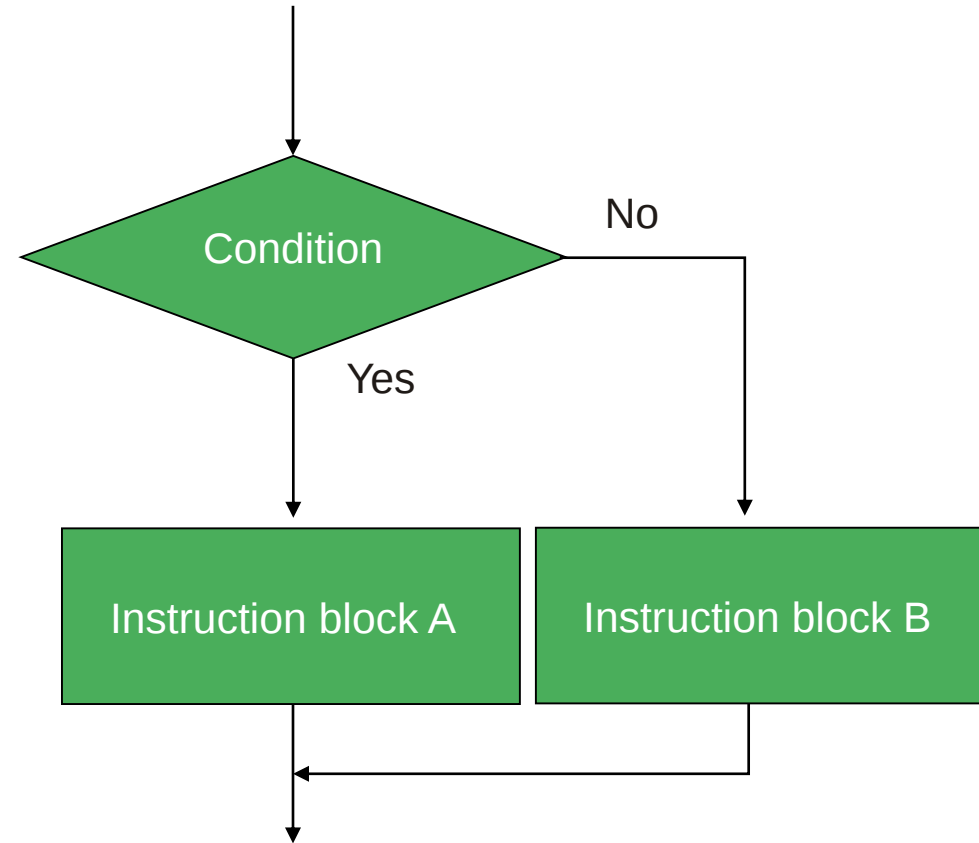
Instruction block;

ELSE

Instruction block;

END_IF

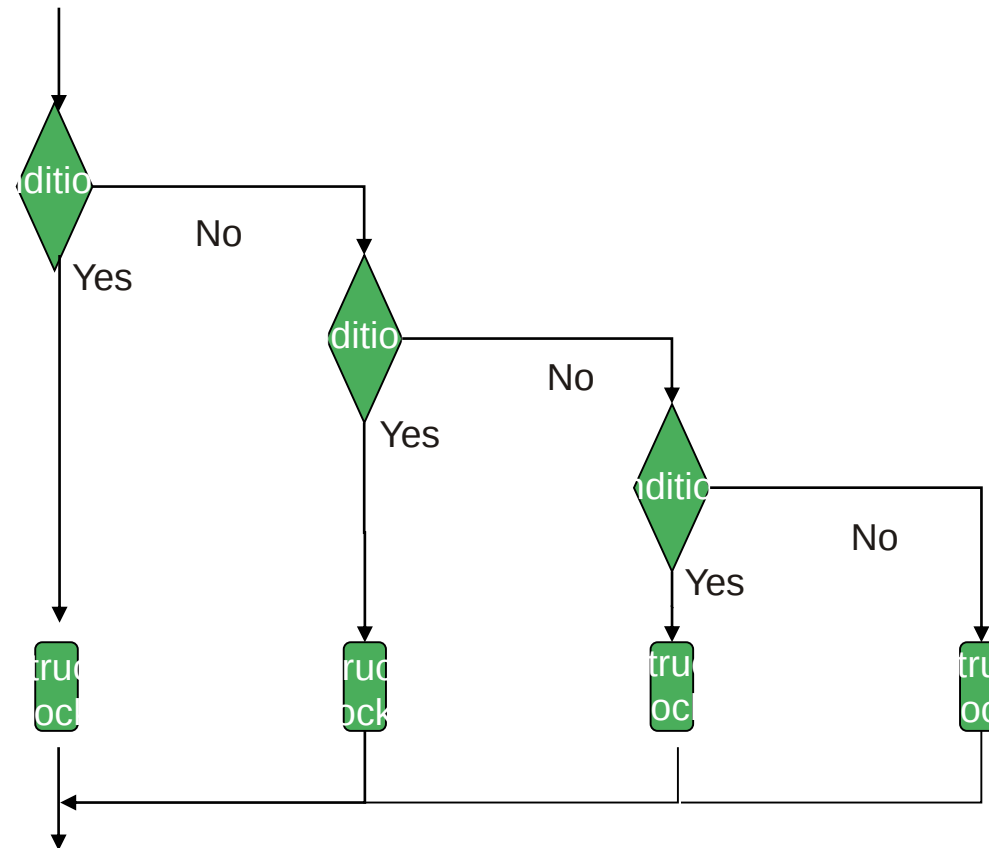
```
IF a>b THEN  
  Instruction block A;  
ELSE  
  Instruction block B;  
END_IF
```



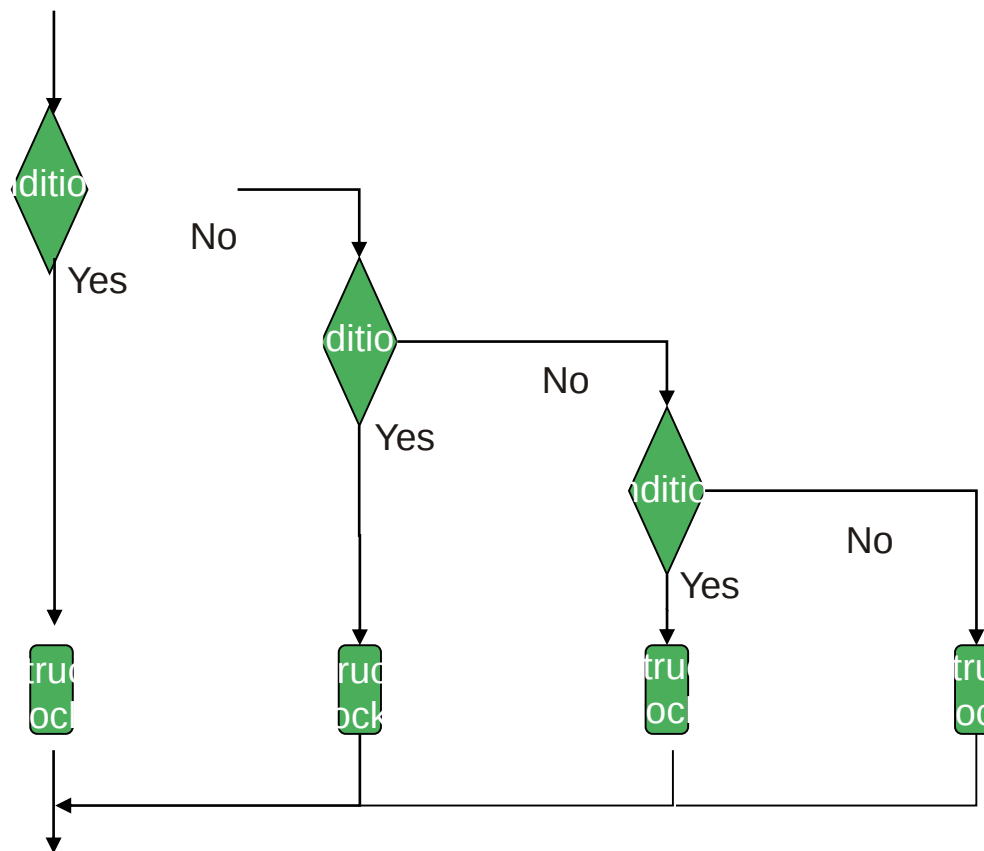
ST: IF instruction

BECKHOFF

```
IF Condition1 THEN  
    Instruction block A;  
ELSE  
    IF Condition2 THEN  
        Instruction block B;  
    ELSE  
        IF Condition3 THEN  
            Instruction block C;  
        ELSE  
            Instruction block D;  
        END_IF  
    END_IF  
END_IF
```



```
IF Condition1 THEN  
    Instruction block A;  
ELSIF Condition2 THEN  
    Instruction block B;  
ELSIF Condition3 THEN  
    Instruction block C;  
ELSE  
    Instruction block D;  
END_IF
```



布尔表达式

条件:

- 布尔类型变量 → IF bVar THEN
.
- 比较 → IF a>b THEN
.
- 功能块的值的判断 → IF LEFT(STR:= strVar, SIZE:=7) = 'TwinCAT'
THEN
.
- 功能块中的值 → IF Ton1.Q THEN
.
- 仅功能块调用 → ~~IF Ton1(IN:=bVar, PT:=T#1s) THEN~~
.

ST CASE Instruction

BECKHOFF

CASE 选择条件

OF

1: **Instruction 1**

2, 4, 6: **Instruction 2**

7..10: **Instruction 3**

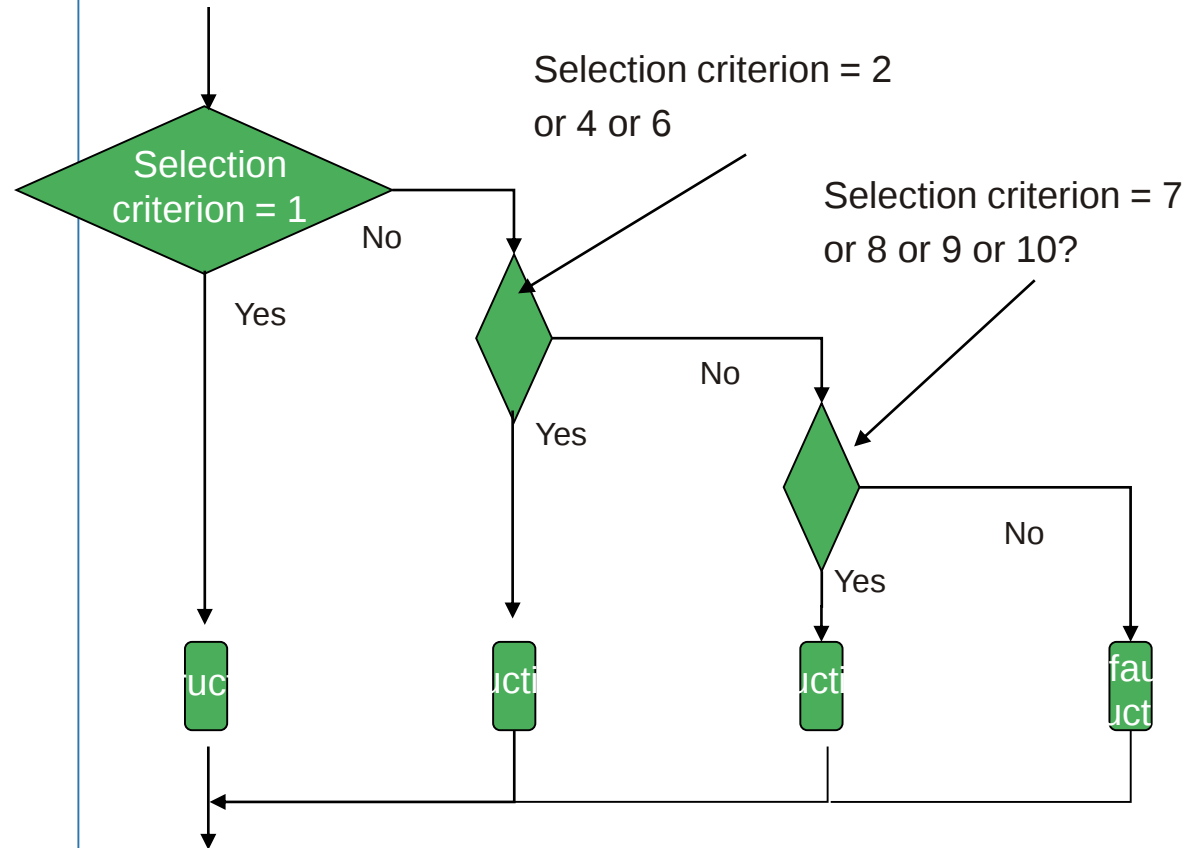
..

ELSE

Default

instructions

END_CASE;



CASE State OF

0: Q0:=TRUE;

IF Transition THEN state := 1; END_IF

1: Q1:=TRUE;

IF Transition THEN state := 2; END_IF

2: Q2:=TRUE;

IF Transition THEN state := 3; END_IF

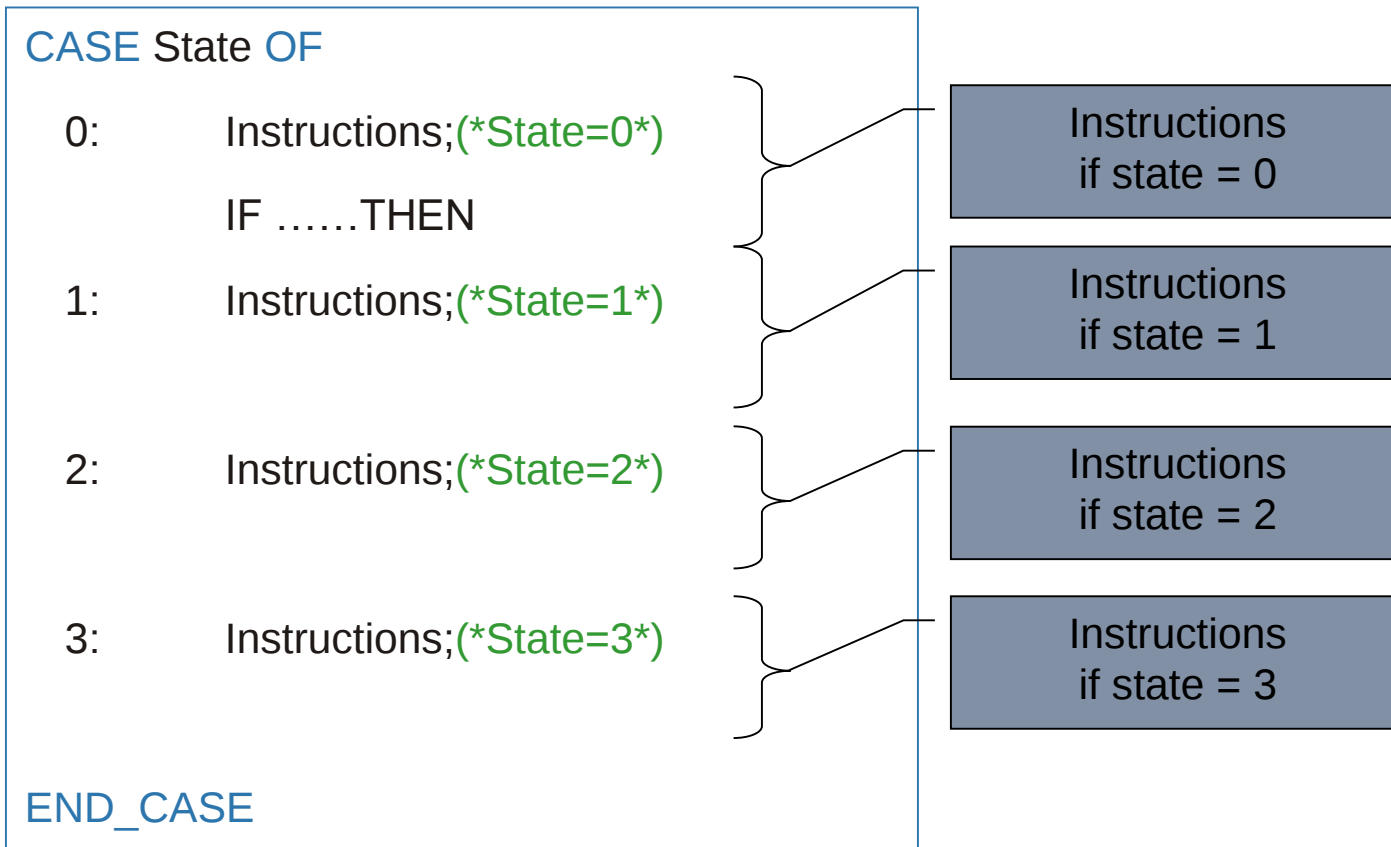
3: Q3:=TRUE;

IF Transition THEN state := 0; END_IF

END_CASE

Instructions for the step
(Actions)

“Step-further condition”
(Transition)



ST: CASE instruction “Integer Selector Value” with Enum types

BECKHOFF

Enum-Typ:

TYPE Schritte :

(INIT:=0, START, AUTOMATIK, ENDE);

END_TYPE

CASE State OF

INIT: Instructions;(*State=0*)

START: Instructions;(*State=1*)

AUTOMATIK: Instructions;(*State=2*)

ENDE: Instructions;(*State=3*)

END_CASE

ST: CASE instruction: suggestion for a step chain / state machine

TYPE Schritte :

(INIT:=0, START, AUTOMATIK, ENDE);

END_TYPE

CASE State **OF**

INIT: Q0:=TRUE;

IF Transition **THEN** state := START; **END_IF**

START: Q1:=TRUE;

IF Transition **THEN** state := AUTOMATIK; **END_IF**

AUTOMATIK: Q2:=TRUE; Step

IF Transition **THEN** state := ENDE; **END_IF**

ENDE: Q3:=TRUE;

IF Transition **THEN** state := INIT; **END_IF**

END_CASE

Instructions for the step
(Actions)

“Step-further condition”
(Transition)

ST: CASE instruction “Integer Selector Value” with constants

BECKHOFF

```
VAR CONSTANT
    Step1 : INT:= 0;
    Step2 : INT:= 1;
    Step3 : INT:= 2;
    Step4 : INT:= 3;
END_VAR
```

```
VAR
    State:INT;
END_VAR
```

CASE State OF

```
    Step1:      Instructions;(*State=0*)
    Step2:      Instructions;(*State=1*)
    Step3..Step4: Instructions;(*State=2 oder 3*)
```

END_CASE

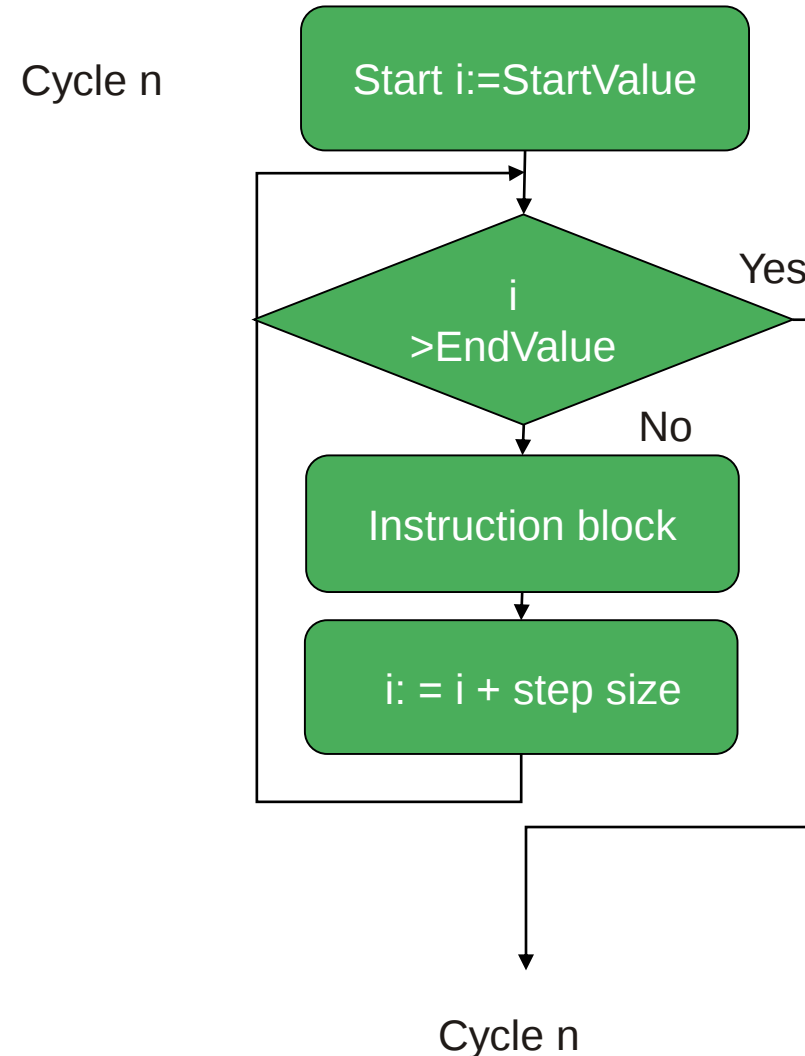
所有循环在遇到关键字EXIT时,便跳出循环.

	Expression	Processing	n cycle fix
FOR	SINT/INT/DINT	Instructions follow condition	Yes
WHILE	BOOL	Instructions follow condition	No
REPEAT	BOOL	Condition follows instructions	No

FOR循环语句如下工作方式为:

1. 首先赋值*i*然后
2. 执行后判断 *i* 是否达到关键字**TO**后的状态,达到则退出,
3. 未达到执行循环内的语句
4. *i* 增加关键字**BY**后的数值并再次执行2-4.

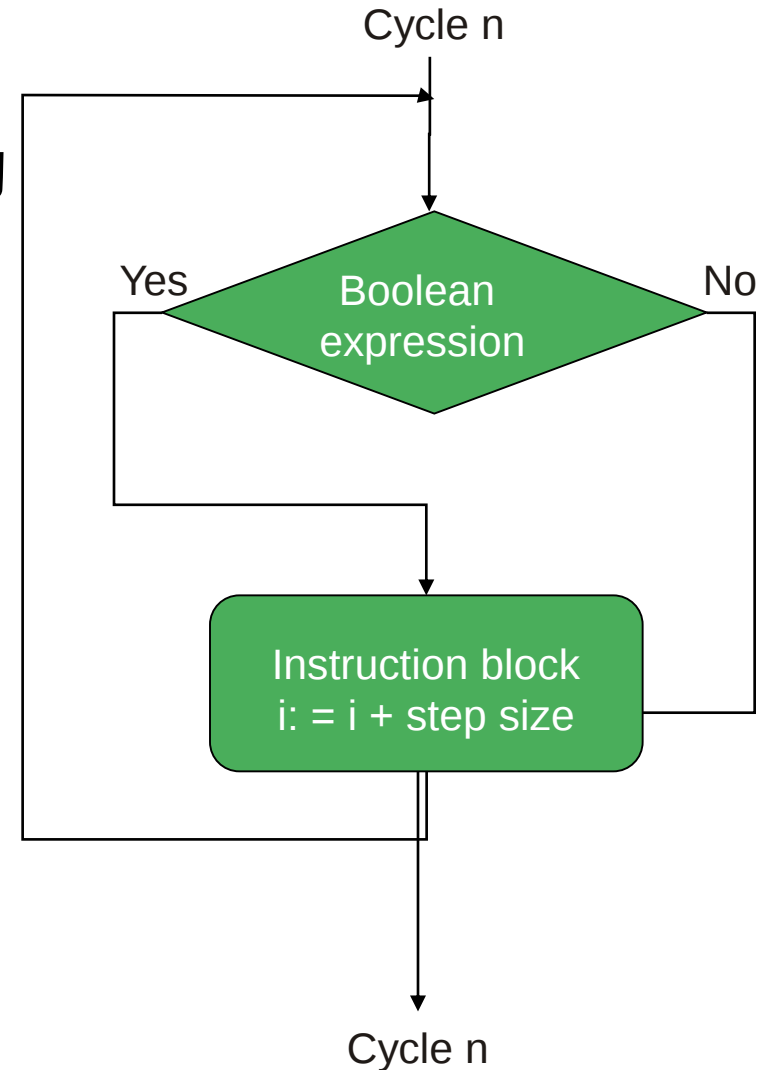
```
FOR i:=1 TO 12 BY 2 DO  
    Feld[i]:=i*2;(*Anweisung*)  
END_FOR
```



WHILE循环

当关键字WHILE后面的布尔表达式的值为TRUE.则一直执行循环.

```
i:=0;  
WHILE i<100 DO  
    Feld[i]:=i*2;(*Anweisung*)  
    i:=i+1;  
END_WHILE
```



REPEAT循环在布尔表达式的值是FALSE一直执行循环,在布尔表达式的值变为TRUE时退出循环.程序至少被执行一次.

```
i:=0;
```

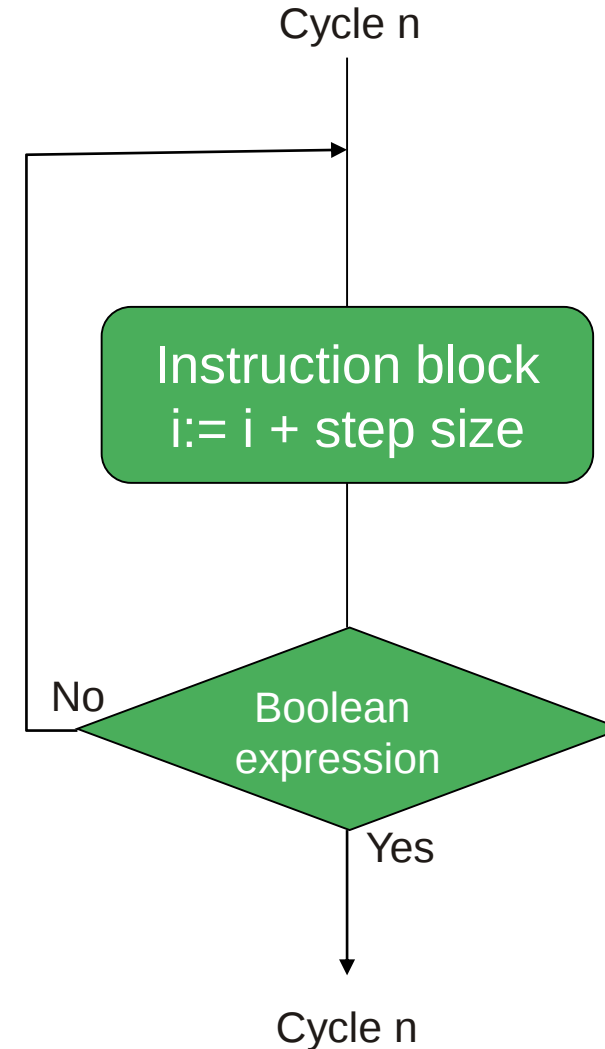
```
REPEAT
```

```
  Feld[i]:=i*2;(*Anweisung*)
```

```
  i:=i+1;
```

```
UNTIL i>100
```

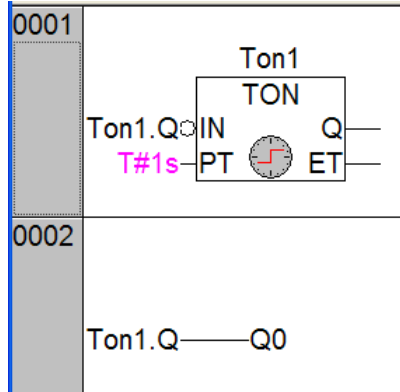
```
END_REPEAT
```



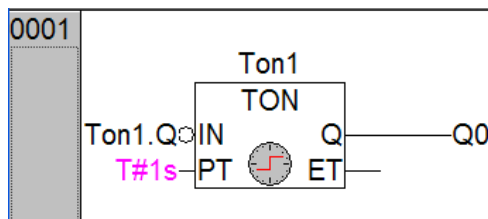
VAR

TON1:TON;

END_VAR



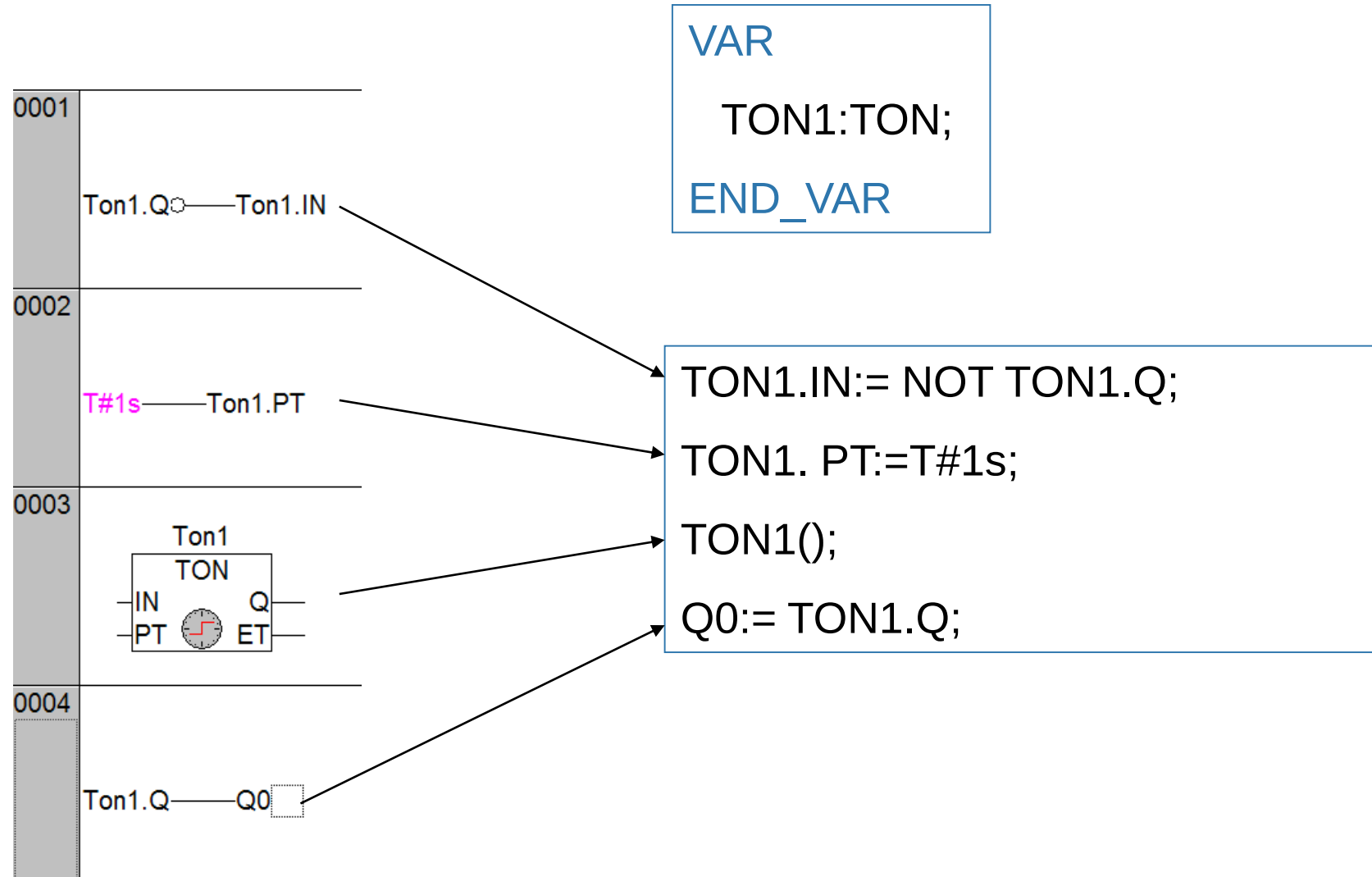
TON1 (IN:= NOT TON1.Q , PT:=T#1s);
Q0:= TON1.Q;



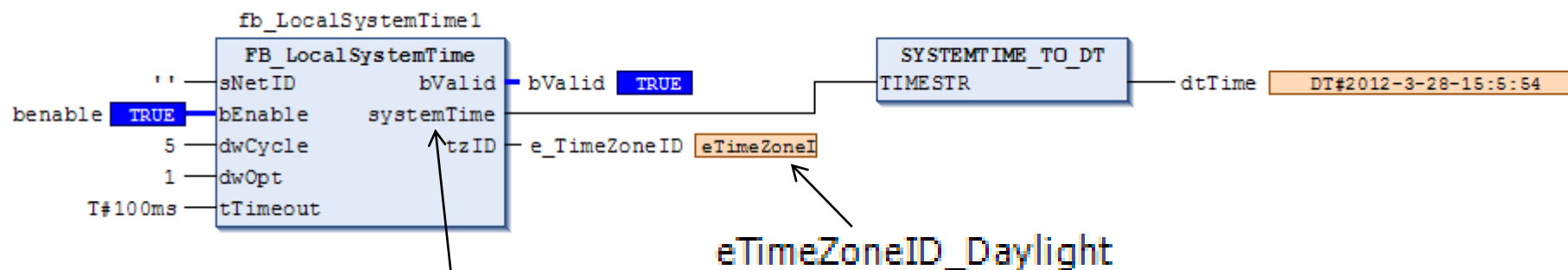
TON1(IN:= NOT TON1.Q, PT:=T#1s , Q=>Q0);

ST: FB calls in ST (alternative)

BECKHOFF

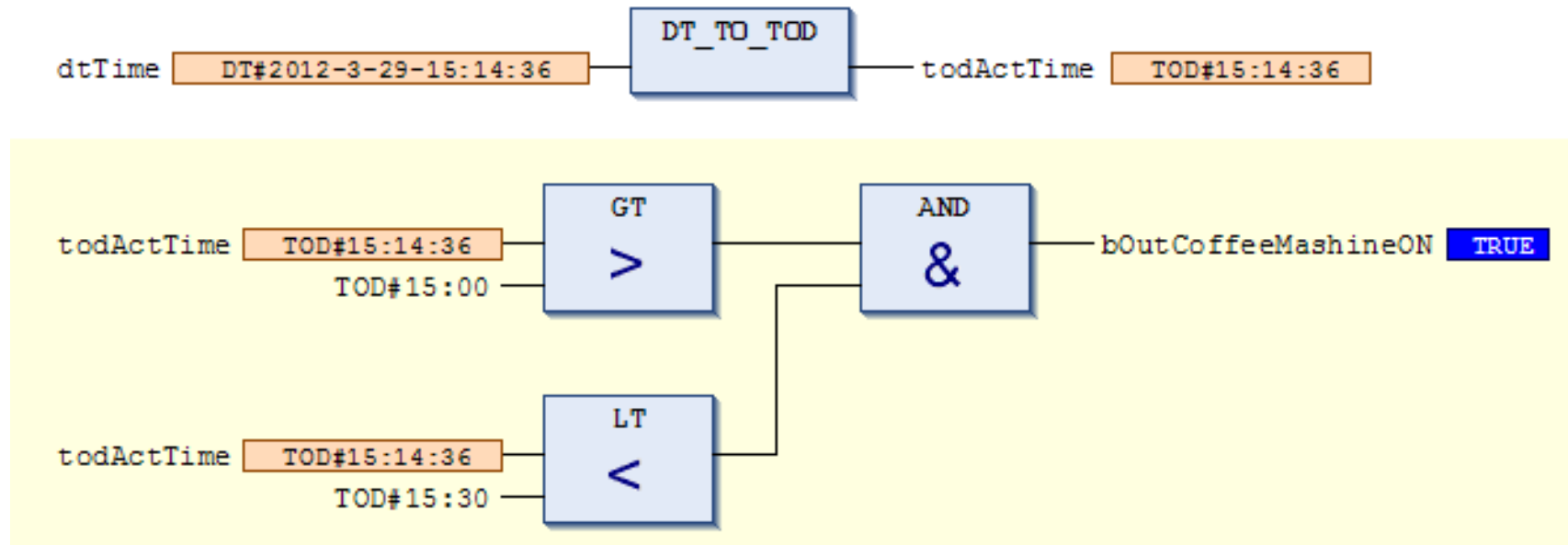


通过功能块FB_LocalSystemTime可以读出系统时间
(功能块在Tc2_Uilities库中)



systemTime	TIMESTRUCT	
wYear	WORD	2012
wMonth	WORD	3
wDayOfWeek	WORD	3
wDay	WORD	28
wHour	WORD	15
wMinute	WORD	7
wSecond	WORD	59
wMilliseconds	WORD	155

DT类型数据可以用在IEC的标准操作中.



德国倍福自动化有限公司

上海总部

上海市汶水路299弄9号（市北智汇园）

电话： 021-6631 2666

传真： 021-6631 5696

E-Mail: support@beckhoff.com.cn

Web: www.beckhoff.com.cn

虚拟学院: <https://tr.beckhoff.com.cn>



扫一扫，关注倍福官方微信！

© 德国倍福自动化有限公司

本 PowerPoint 演示文稿中的所有照片及图片均受版权保护。未经许可，任何用户不得擅自复制、使用、转载或将其提供给任何第三方。

Beckhoff®、TwinCAT®、EtherCAT®、Safety over EtherCAT®、TwinSAFE®、XFC® 和 XTS®是德国倍福自动化有限公司的注册商标。本 PowerPoint 演示文稿中所使用的其它名称可能是商标名称，任何第三方为其自身目的而引用，都可能触犯商标所有者的权利。

本PowerPoint 演示文稿中所包含的信息仅是一般描述或性能特征简介，在实际应用中并不总是与所述完全一致或者可能由于产品的进一步开发而不完全适用。仅在书面认同情况下，才提供相关特性信息。