BECKHOFF

TwinCAT 3_PLC_OOP教程

Version 3.2

毕孚自动化设备贸易（上海）有限公司

2020 年 7 月

前言

TwinCAT3 是基于 PC 的控制软件并且它开启了一个新的时代，是倍福公司历史上又一个里程碑。

特别是在高效的工程领域中 TwinCAT3 将模块化思想以及其灵活的软件架构，融入到整个平台。

几乎每一种控制应用程序都能在 TwinCAT3 中实现。从印刷设备、木工设备、塑料机械或门窗设备、风力发电机和实验台，亦或是楼宇，诸如剧院，以及运动场，一切都可以通过 TwinCAT3 实现自动化。

用户可以选择不同的编程语言来实现这些应用。除了经典的 PLC 编程语言的 IEC 61131-3，用户现在也可以用高级语言 C 或 C++，以及 MATLAB®/ Simulink®。

整合了运动功能从而简化了工程项目，以及全新的安全应用编辑更加人性化。

这些以及更多的特性都证明了为什么 TwinCAT3 也名为扩展的自动化。

本书针对任何想要学习倍福 TwinCAT3 软件如何实现基于 PC 控制编程的读者，阅读本书需要预先具备 IEC61131-3，C/C++或 MATLAB®/ Simulink®中至少一种编程语言的知识。

本书内容主要介绍了 IEC61131-3 新标准中扩展的 OOP（面向对象）功能的学习，了解到许多新概念：方法，属性，扩展，接口等等，以及如何通过新的关键词来实现这些概念，从中可以感受到 OOP 编程带来的便利。

本书所有的内容都会不间断更新，如果想获取更新的教材可以通过访问 FTP 获取到，当然本书所有配套的案例程序也会在此 FTP 中供所有读者免费获取。

FTP 地址：ftp://ftp.beckhoff.com.cn/TwinCAT3/TC3_training/

欢迎对本书的结构、内容提出意见和建议，请发邮件至：
y.yang@beckhoff.com.cn

杨煜敏
2015 年 12 月 1 日

目录

一、 TwinCAT3-OOP 编程

本章主要分成三个部分，第一部分是关于 IEC61131-3 编程标准第三版中新增的数据类型和功能；第二部分针对 TwinCAT3 中 OOP 的关键字做一个用法的介绍；最后以一个完整的例程给大家讲一下如何用面向对象的思想去写一个程序。

1. 新增的数据类型和功能

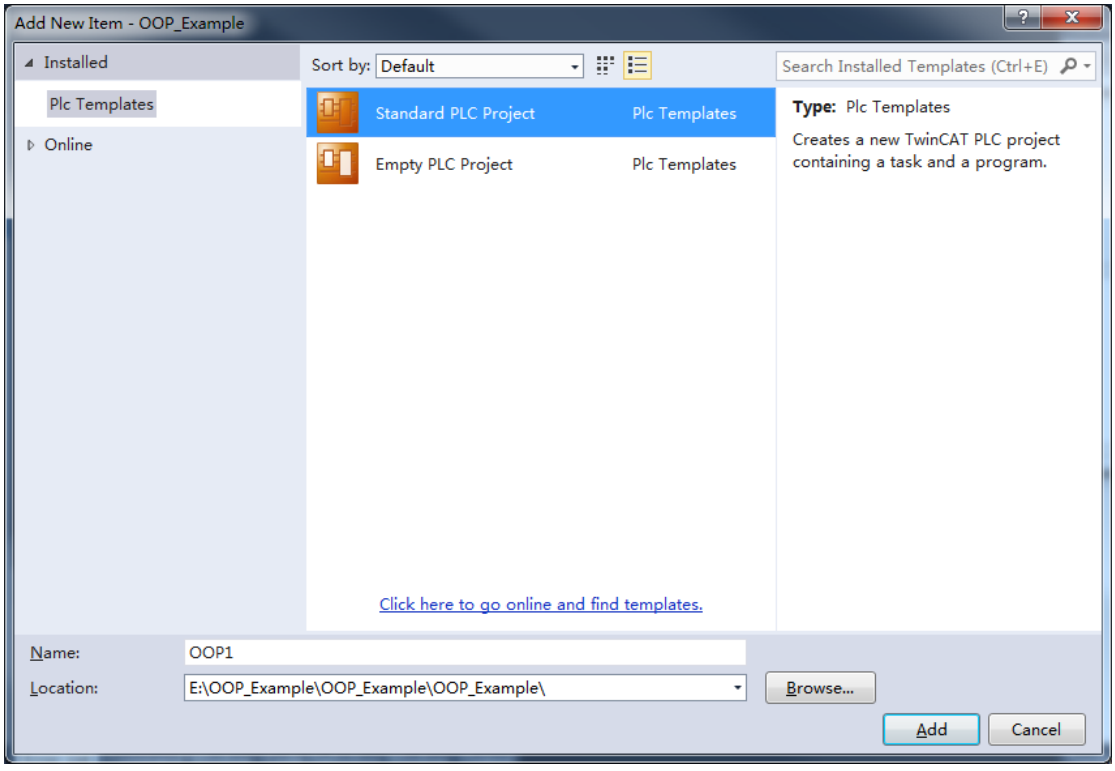
IEC61131-3 第三版的新增数据类型，主要是针对 64 位操作系统的，例如：LINT，ULINT，LWORD，和 LTIME, LDATE, LDATE_AND_TIME, LTIME_OF_DAY 等时间类型的变量，还包括 UNION 共用体以及 WSTRING 这样的数据类型。

除此之外，TwinCAT3 还新增了两个作用域，分别是 VAR_STAT 和 VAR_TEMP，在这两个作用域之下，我们可以创建静态变量和临时变量。

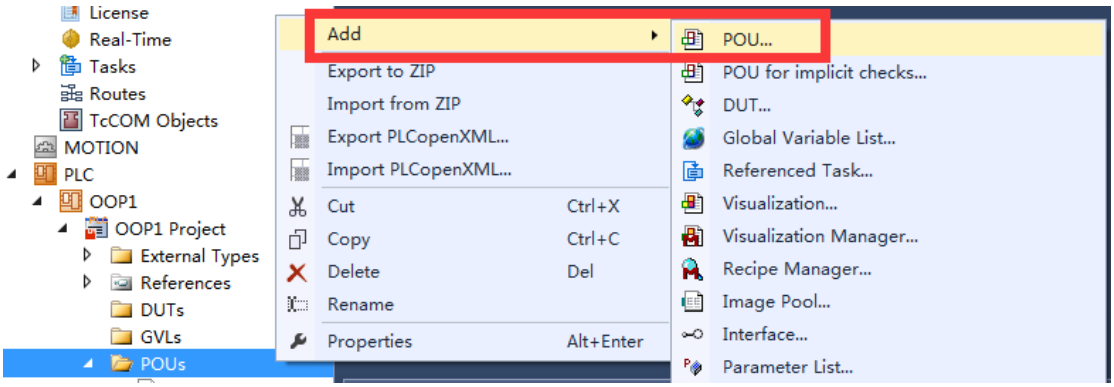
1.1 VAR_STAT

众所周知，所有 Method 和 Function 里的内部变量默认都是临时变量，也就是我们申明在 VAR 关键字下的变量默认都是临时变量。他们每一次被调用后都会被初始化。这里给大家做一个演示。

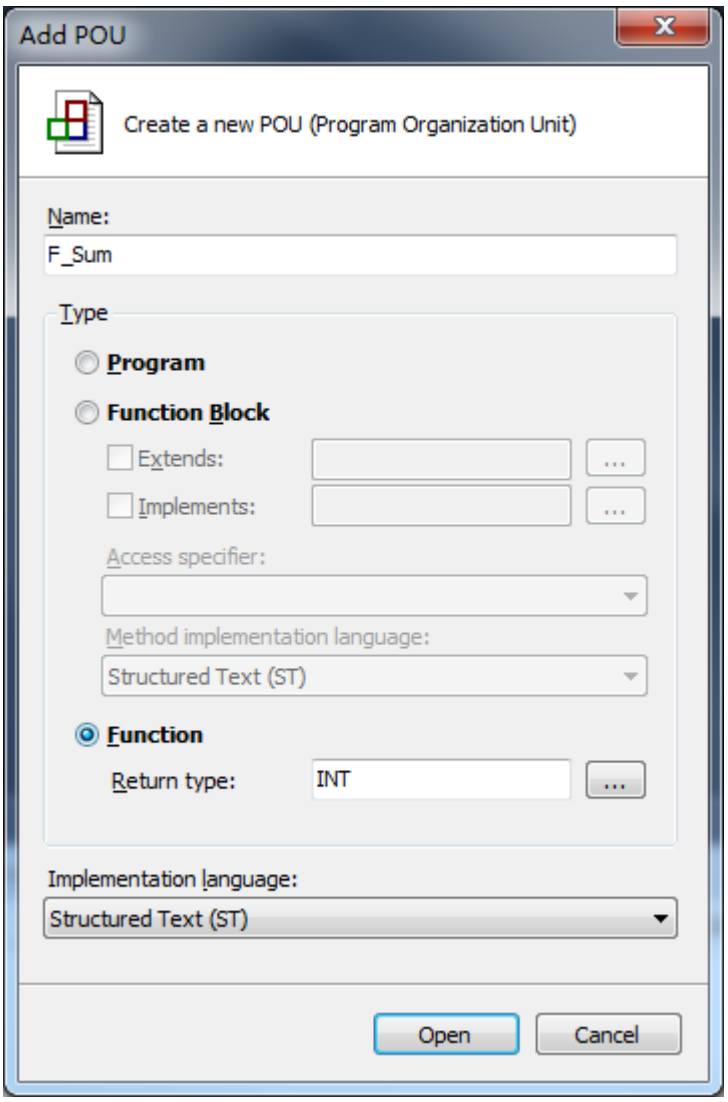
（1）首先还是新建 PLC 程序，并且取名为 OOP1。



（2）右键 POUs 添加 POU。



（3）创建一个 Function，返回类型选择 INT 型。



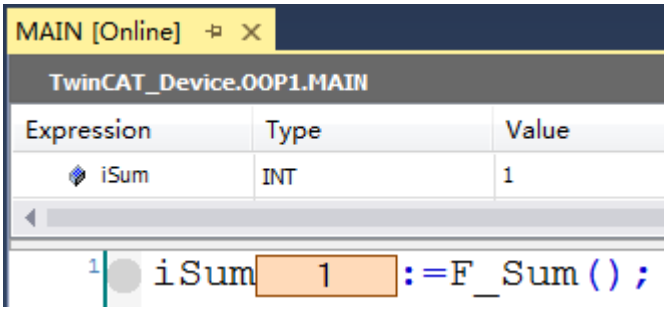
（4）在这个 Function 中编写程序，并在主程序中调用 Function。

```
F_Sum*
1  FUNCTION F_Sum : INT
2  VAR
3      ia: INT;
4  END_VAR

1  ia:=ia+1;
2  F_Sum:=ia;
```

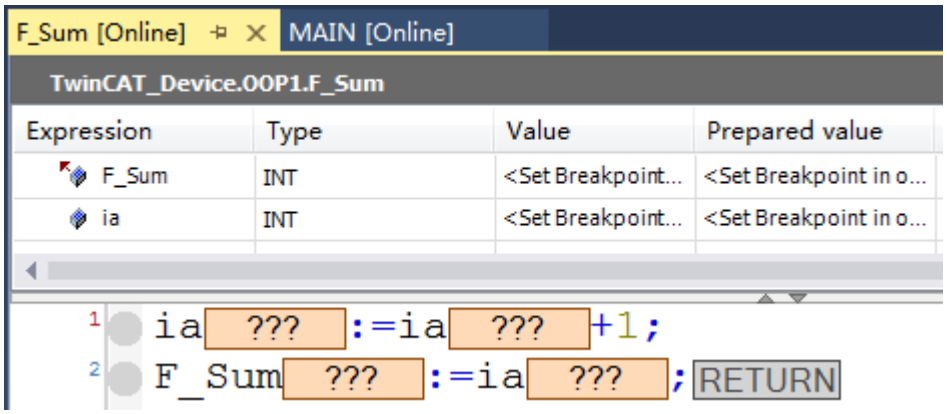
```
MAIN*
1  PROGRAM MAIN
2  VAR
3      iSum: INT;
4  END_VAR

1  iSum:=F_Sum();
```

（5）激活并 Login 之后的运行效果如下图。

主程序中读到的 F_Sum 的返回值是 1，且它始终为 1。

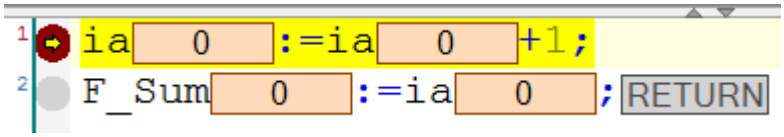


而在 Function 里的变量显示都是问号，并且我们可以发现在 Value 下也没办法看到实际值，它提示我们<Set Breakpoint in order to watch this variable>。

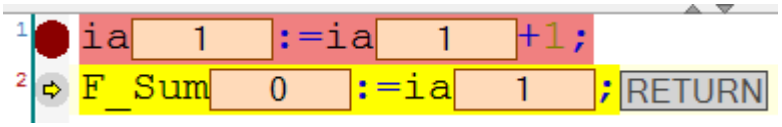


（6）添加断点，单步操作查看 Function 的变量值。（快捷键 F9 设置断点，F11 单步操作）

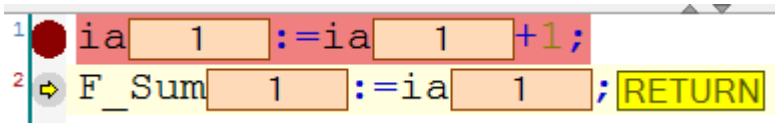
开始时，ia 的值是 0



下一步，ia 进行自加一

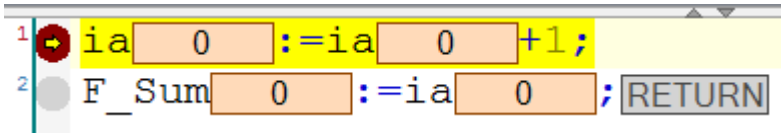


下一步，把 ia 作为返回值赋给这个 Function

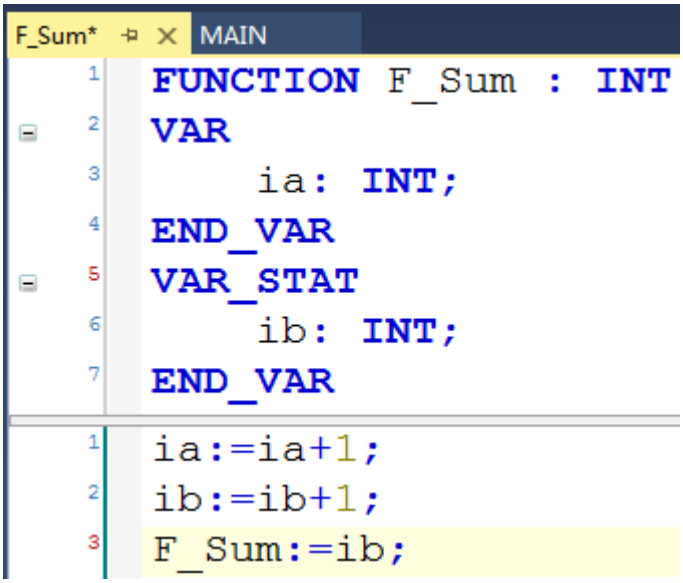


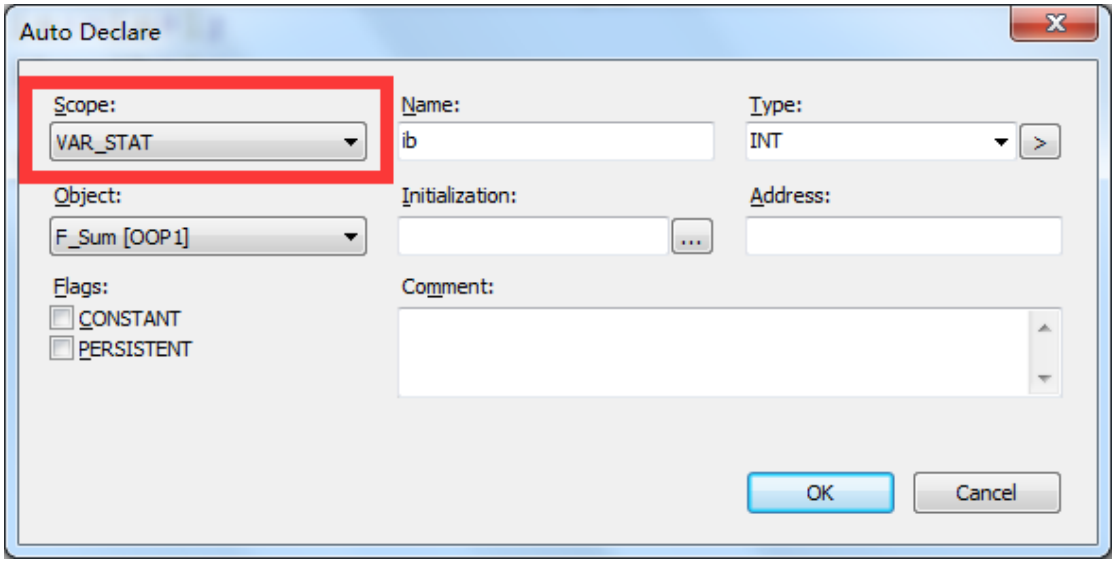
下一步，回到主程序，这个程序把自己的返回值赋给 iSum 变量，使其值为 1。

那么这个 ia 是不是会继续累加呢？我们可以继续单步执行来看一下结果。可以发现 F_Sum 这个 Function 中的 ia 变量已经变回 0 了。所以我们之前看到的始终值为 1 的 iSum 变量就是这么来的。

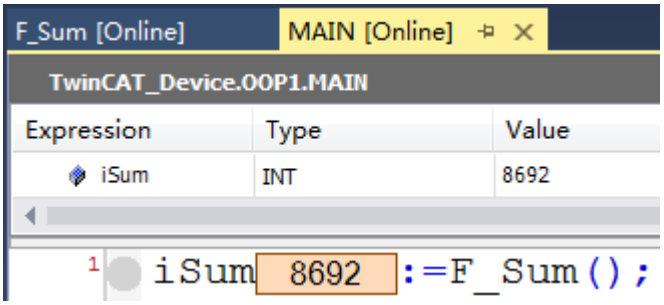


（7）当然在 Function 里也可以声明静态变量，这就要用到我们之前说的 VAR_STAT 关键字。在这个 Function 中编写程序。注意把 ib 变量申明在 VAR_STAT 关键字下。注意 Function 的返回值也要修改。

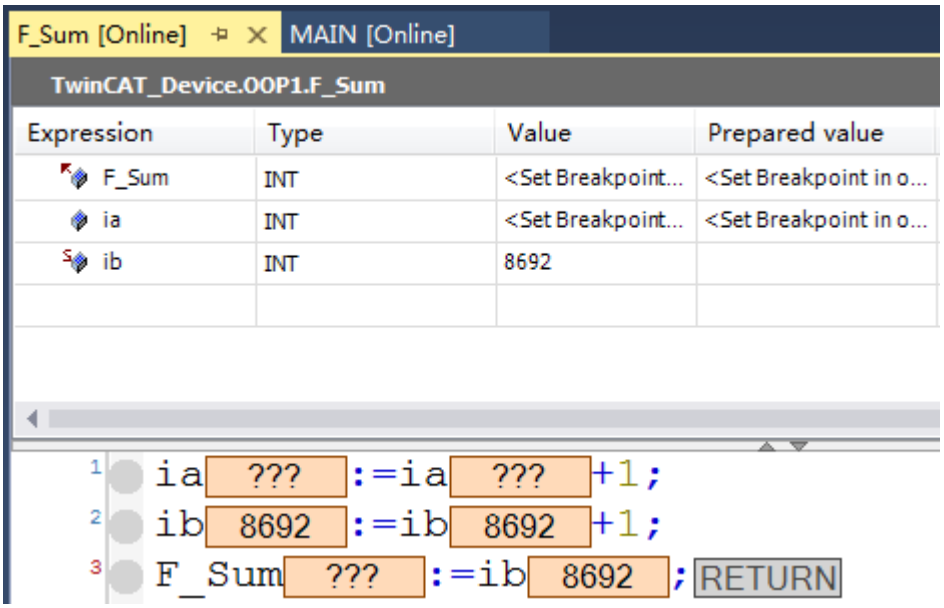




（8）效果如下，我们可以看到主程序里 iSum 的值一直在累加



到 Function 里来看，可以发现 ib 是有显示在进行累加的。我们单步操作的时候也可以发现 ib 的数值一直在累加，并没有像 ia 一样被初始化成 0。因为这里我们的 ib 是一个静态变量。

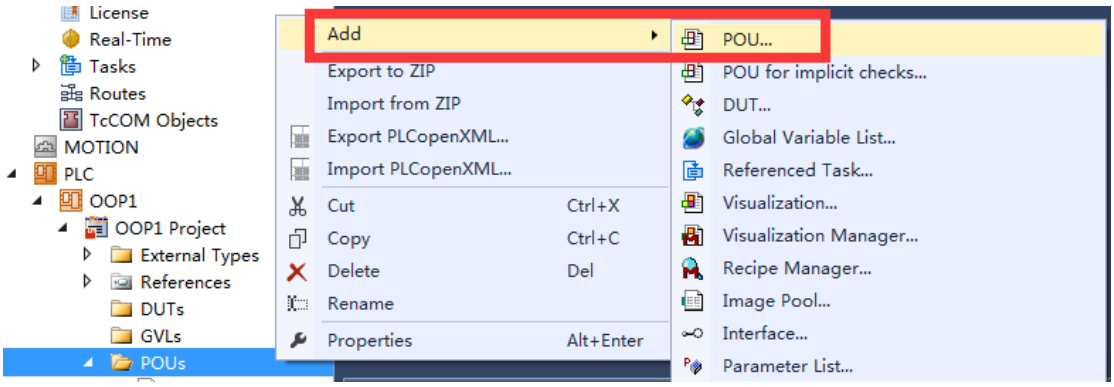


1.2 VAR_TEMP

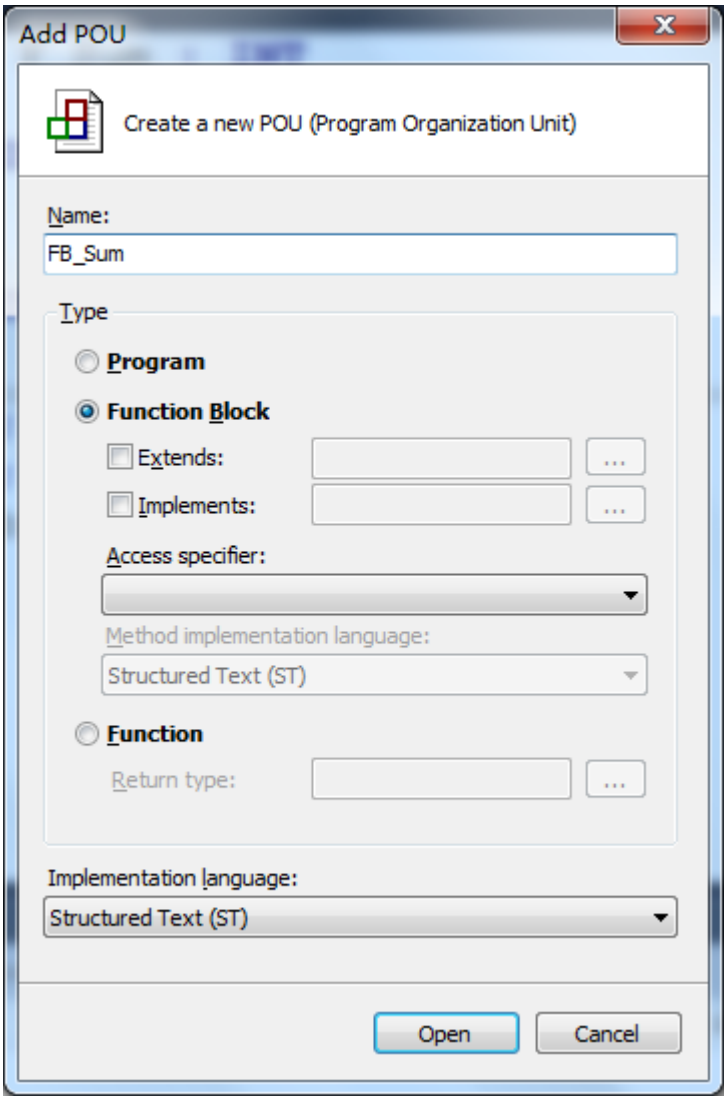
我们都知道，所有程序和功能块中的默认申明的变量都是静态变量（就是我们申

明在 VAR 关键字下的变量）。与上面类似的，我们也可以通过在 VAR_TEMP 关键字下申明变量来使之成为临时变量。下面，就把默认情况和通过 VAR_TEMP 关键字来申明的情况都做一个演示。

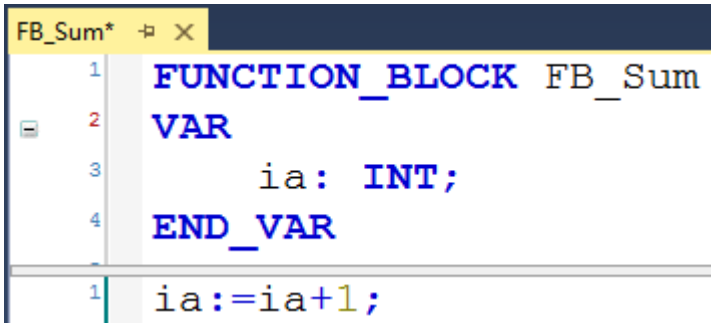
（1）右键 POUs 添加 POU。

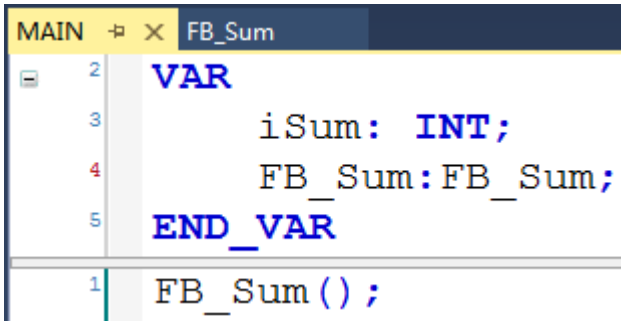


（2）创建一个 Function Block。

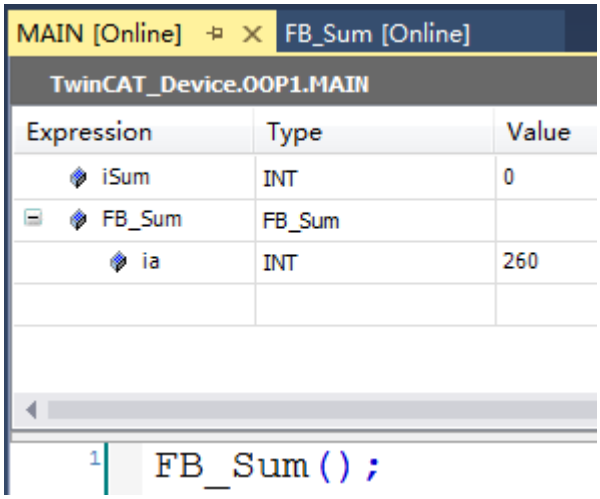


（3）在 Function Block 中编写程序，在主程序中实例化功能块并调用功能块。

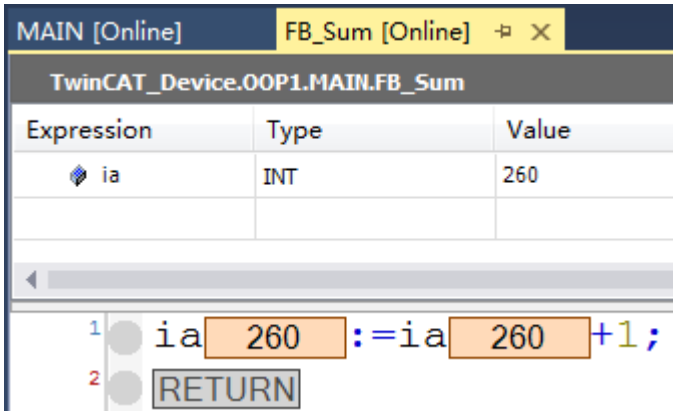




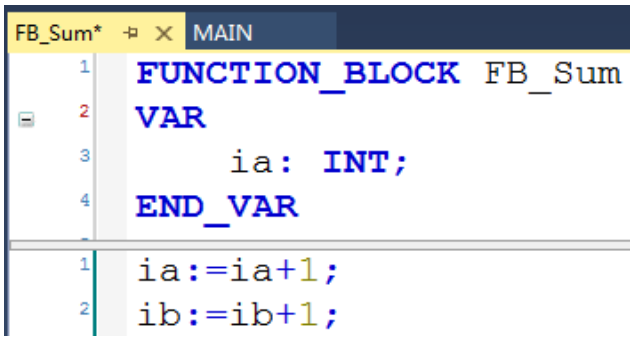
（4）激活并 Login 之后，效果如下。我们可以发现 ia 这个变量一直在做累加。

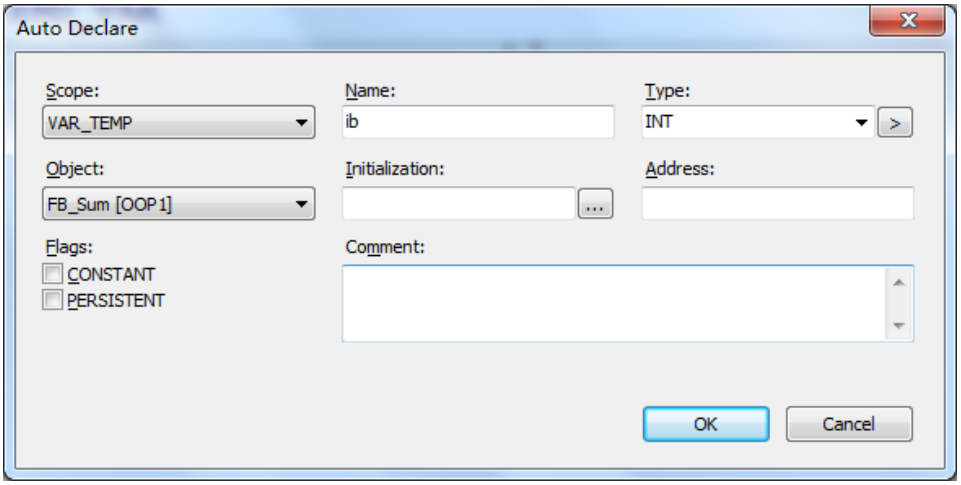


同样的在 Function Block 中也可以看到 ia 在做累加。

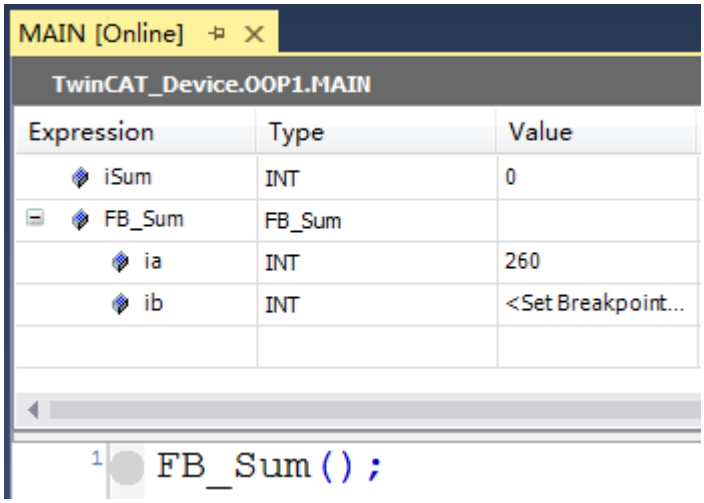


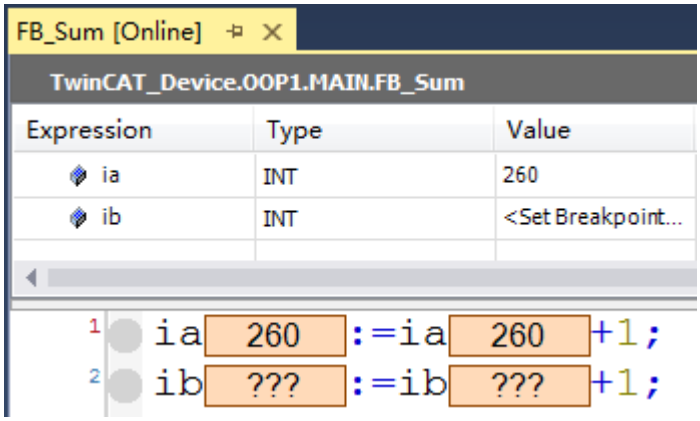
（5）同样，Function Block 也可以建立临时变量，在功能块中编写程序如下，变量注意申明在 VAR_TEMP 关键字下。



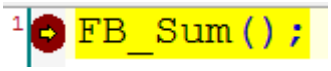


（6）激活并 Login 之后，效果如下。可以发现 ib 这个变量的值是看不见的，同样提示我们要添加断点才可以看到。

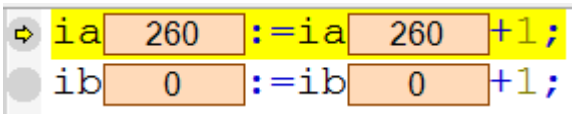




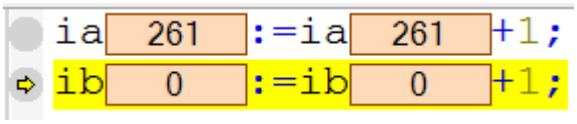
在添加断点之后我们单步操作来看一下 ib 的自加一过程。从主程序的功能块调用开始。



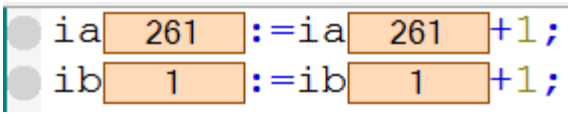
下一步，ia 值为 260，ib 值为 0。



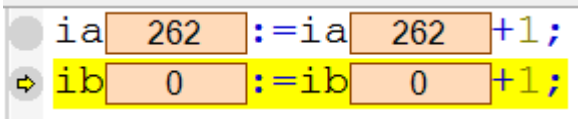
下一步，ia 作自加一，ib 值还是为 0。



下一步，ib 作自加一，那么下个循环 ib 是不是会继续累加呢？可以继续单步执行看一下。



可以发现下一个循环，ib 的初始值还是 0，因为它是一个临时变量，每次调用前都会被初始化，所以它不会像 ia 一样值一直累加上去。



1.3 编译指令

编译编译指令主要分为三种，消息编译、条件编译、属性编译。

1.3.1 消息编译：消息编译可以用于在我们对工程进行编译的时候，在消息栏强制输出一些信息。

消息编译分为四种，text、information、warning、error。我们在编译的时候可以附加消息，警告或是报错。

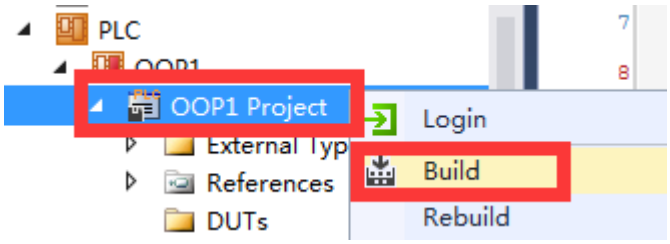
```
{text 'Hello World!'}
{info 'Information!'}
{warning 'Warning!'}
{error 'Error!'}
```

（1）来试一下下面这个例子。

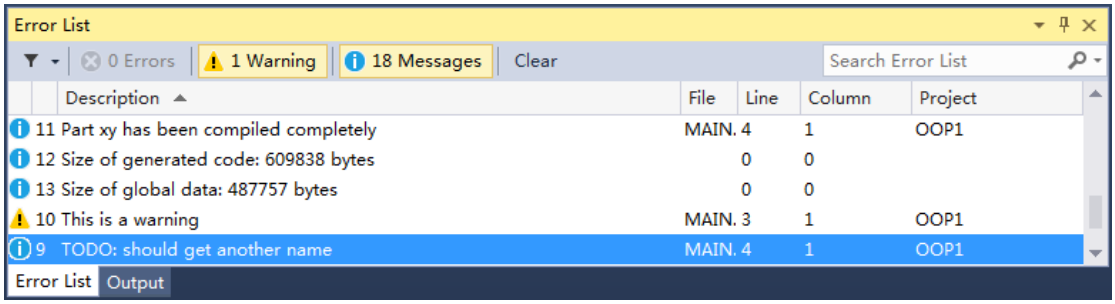
```
MAIN*
PROGRAM MAIN
VAR
    nVar : INT;
    {info 'TODO: should get another name'}
    bVar : BOOL;
    arrTest: ARRAY[0..10] OF INT;
    aTest : ARRAY [0..10] OF INT;
    nIdx : INT;
END_VAR
```

```
arrTest[nIdx] := aTest[nIdx] + 1;
nVar := nVar + 1;
{warning 'This is a warning'}
{text 'Part xy has been compiled completely'}
```

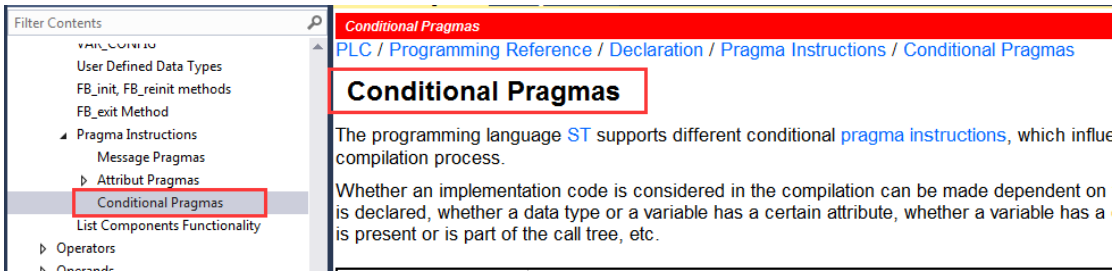
（2）右键 OOP1 Project，点击 Build 进行编译



（3）可以在消息栏找到对应的三条信息，分别地告诉我们已经编译完成、有一个警告要注意、以及显示的变量申明区的信息等等。



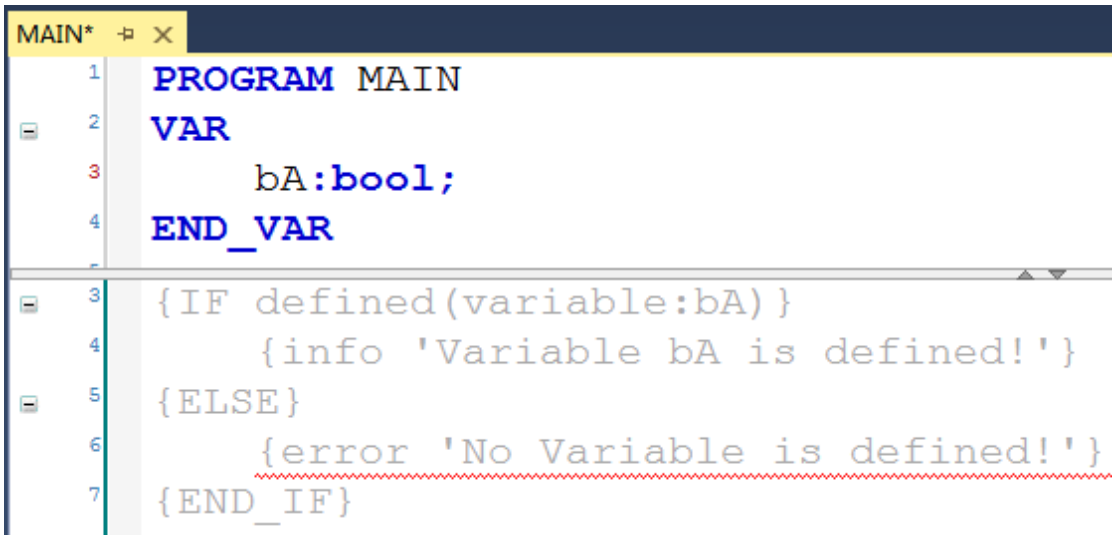
1.3.2 条件编译：我们的 ST 语言支持不同的条件编译指令，这些编译指令会影响预编译或编译过程中代码的生成。条件编译有很多类型，可对是否定义了某一变量、是否具有某一属性、是否具有某一类型、是否具有某一个值等情况判断，然后再选择性地编译。条件编译可配合消息编译一起使用。我们挑几个条件编译指令来演示一下。



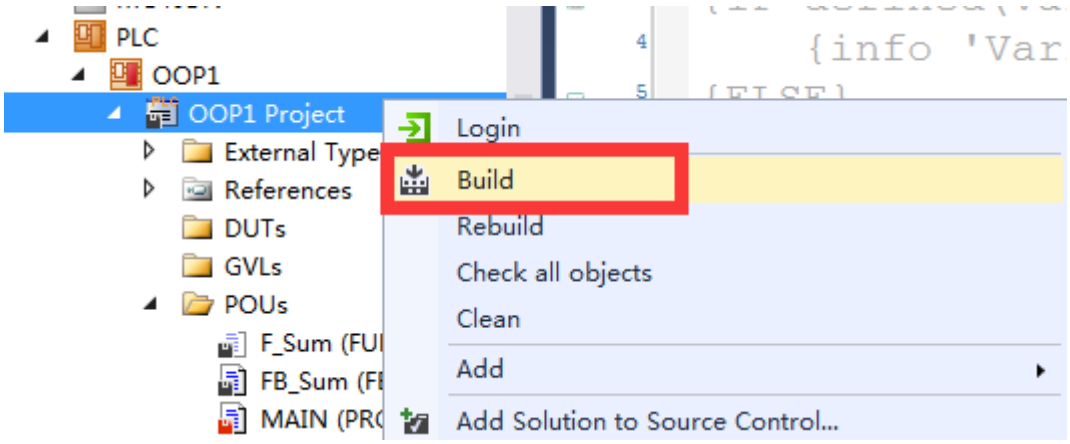
defined (variable:variable)

该指令是对是否定义了某一变量进行判断，如果该变量被定义则表达式值为 TRUE，如果没有定义则该表达式值为 FALSE。

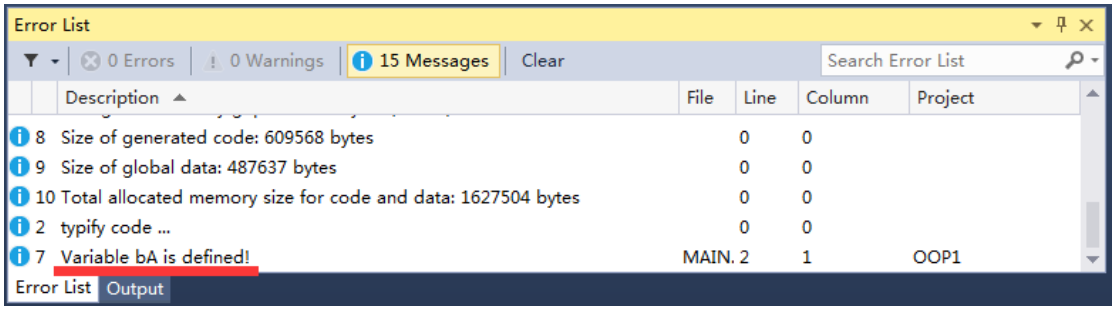
（1）可编写条件编译指令如下



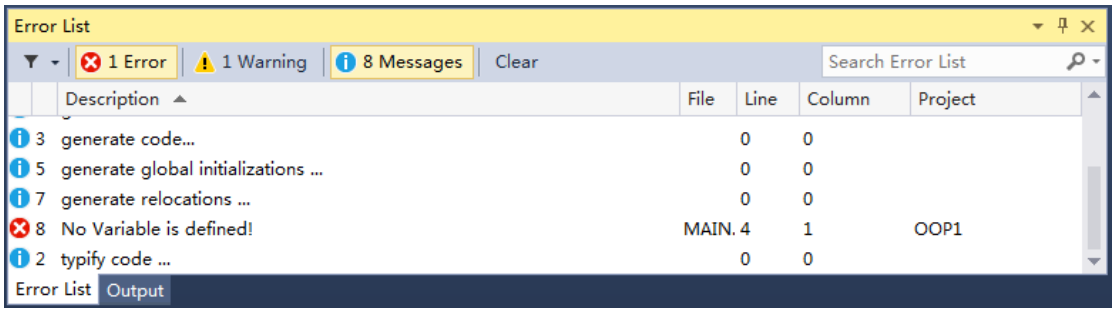
（2）右键 OOP1 Project，选择 Build 编译。



（3）编译完成后可以在消息栏看到自定义的编译信息’Variable bA is defined!’



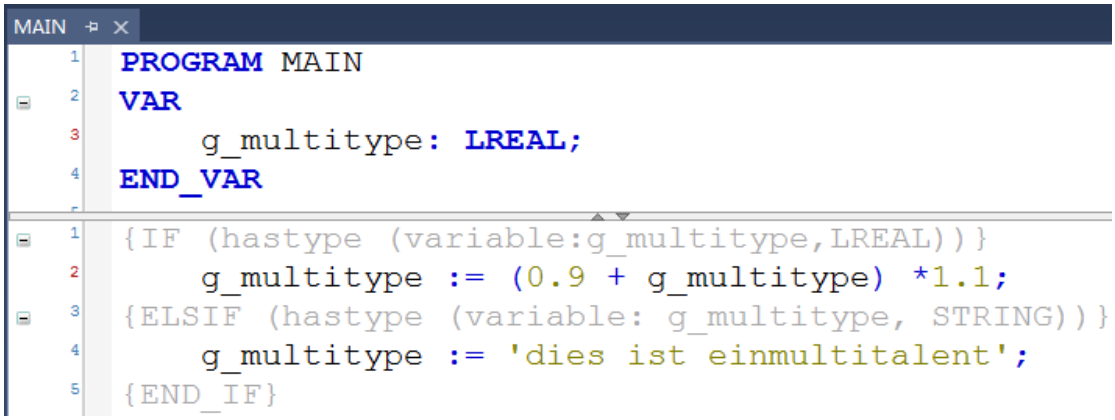
（4）如果变量申明区不申明 bA 变量，则编译后可找到报错信息’No Variable is defined!’



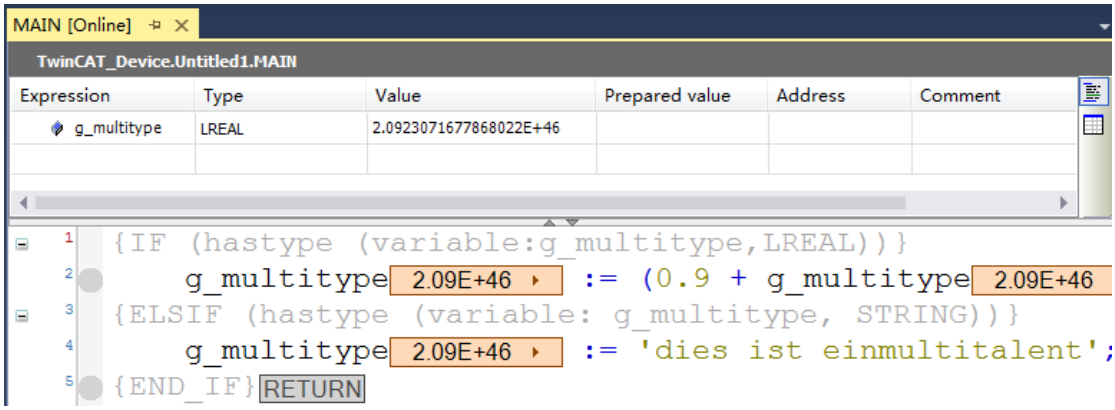
hastype (variable: variable, type-spec)

该指令是对是否定义了某一变量类型进行判断，如果该变量被定义为指定类型则表达式值为 TRUE，如果没有则该表达式值为 FALSE。

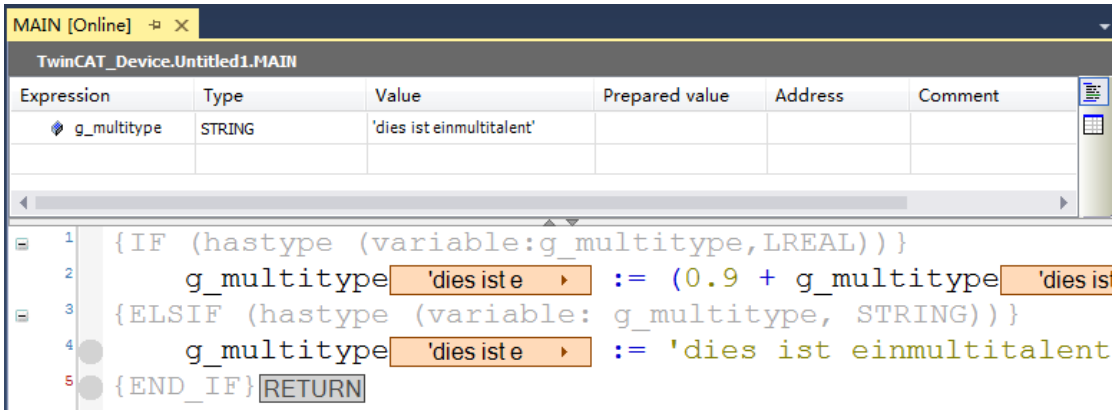
（1）可编写条件编译指令如下



（2）此时 Activate Configuration，Login 并 Run，就可以观察到下面的结果。很明显，这里经过条件编译的判断，只执行了第一个语句，所以 g_multitype 的值一直在按公式增加。



（3）而如果我们把这个变量的类型修改成 STRING 类型，再 Login 并 Run，就会看到它的值变成了下面图中显示的这一串字符串。

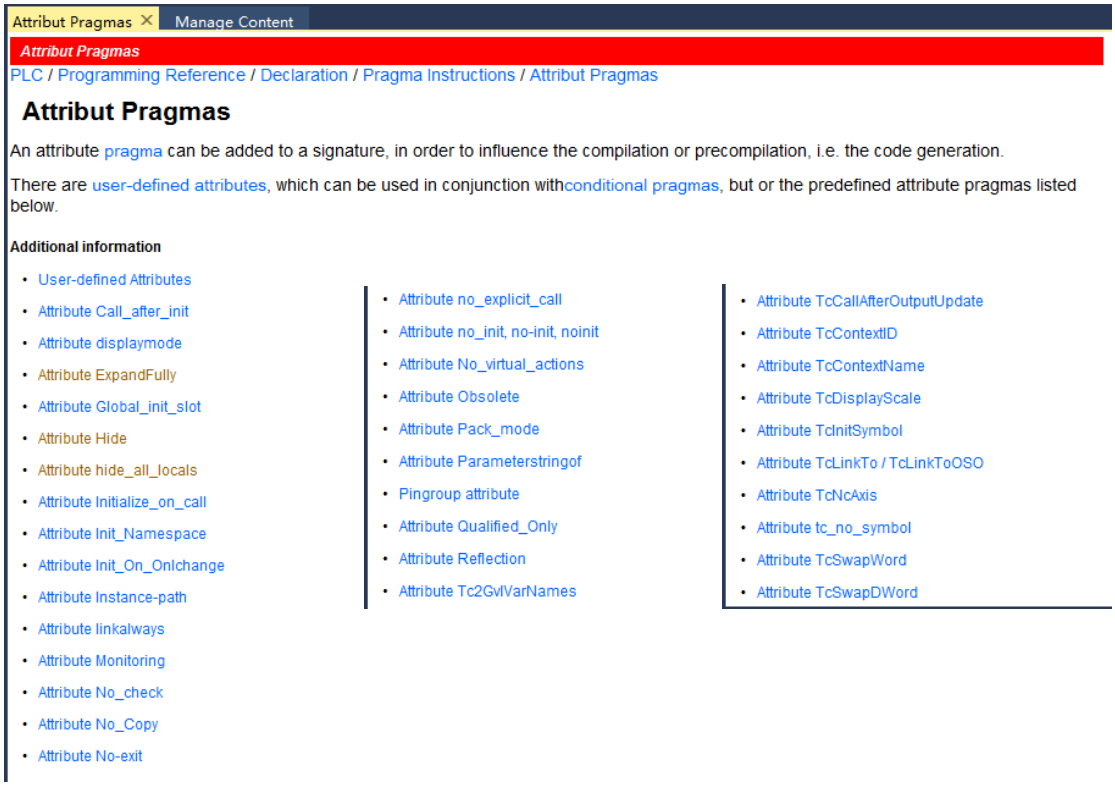


1.3.3 属性编译

属性编译需要添加属性，其种类可以在帮助文档搜索 Pragmas 找到，属性编译的种类非常之多。可以在变量申明区添加属性，会影响编译或预编译的结果。

也可以有自定义属性，一般配合条件编译一起使用。下面这张图是我们所有预编译好的属性编译。

属性一般适用于改变所声明或定义的函数或数据的特性。在本教材中我们选取了几个比较具有代表性的属性来做演示。



（1）Hide 属性：{attribute 'hide'}

新建功能块，在功能块的 VAR_INPUT 关键字下申明三个变量，在 d 变量的上一行添加 Hide 属性，在 VAR_OUTPUT 和 VAR 关键字下各申明一个变量，在变量前一行添加 Hide 属性。

FB_Pragmas

```
FUNCTION_BLOCK FB_Pragmas
VAR_INPUT
    a   : INT;
    {attribute 'hide'}
    d   : BOOL;
    b : BOOL;
END_VAR
VAR_OUTPUT
    c   : INT;
END_VAR
VAR
    {attribute 'hide'}
    e   : BOOL;
END_VAR
```

在主程序对功能块实例化，Activate Configuration 之后，Login

MAIN*

```
PROGRAM MAIN
VAR
    FB_Pragmas:FB_Pragmas;
END_VAR
```

我们只能看到 a、b、c 这三个变量，而被隐藏的 d 和 e 及其值是无法被看到的。

MAIN [Online]

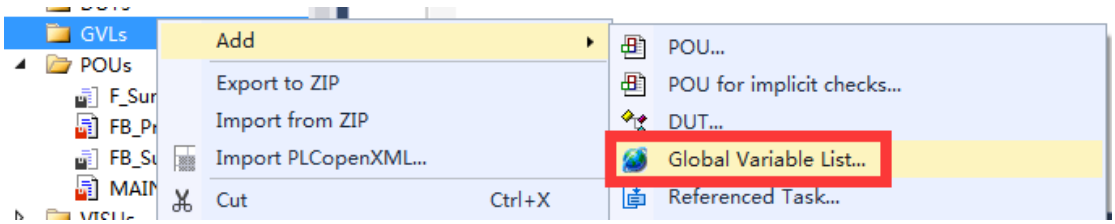
TwinCAT_Device.Untitled1.MAIN

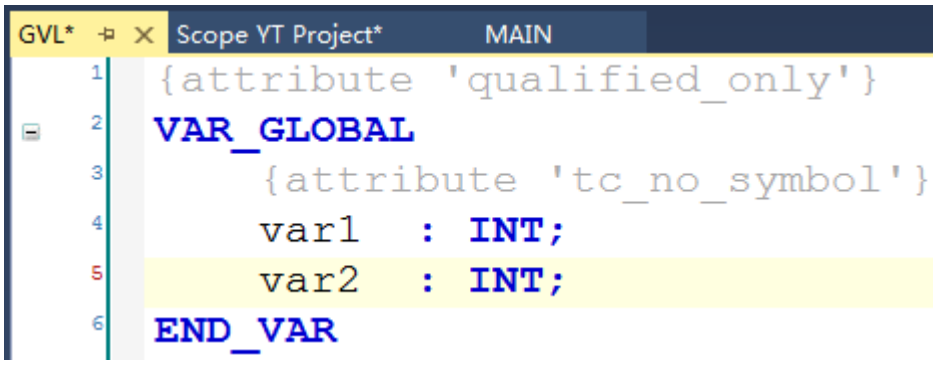
Expression	Type	Value
FB_Pragmas	FB_Pragmas	
a	INT	0
b	BOOL	FALSE
c	INT	0

隐藏属性可用于防止变量在列表组件中、点索引时或是在线的变量申明区域可见的情况。当然，它是作用域的，只会对它的下一行变量有效。

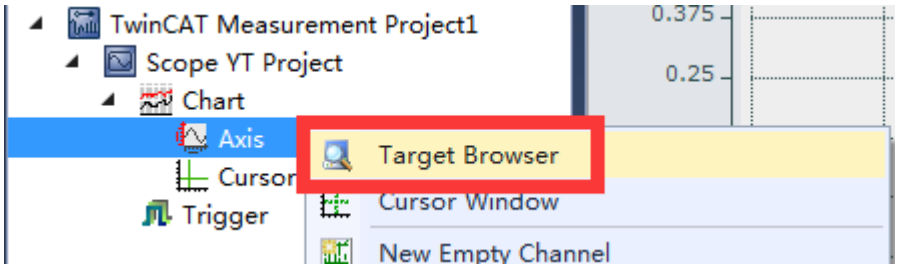
（2）No Symbol 属性：{attribute 'tc_no_symbol'}

新建一个 Global Variable List，在 VAR_GLOBAL 关键字下写两个变量，在其中一个变量前一行加上{attribute 'tc_no_symbol'}属性

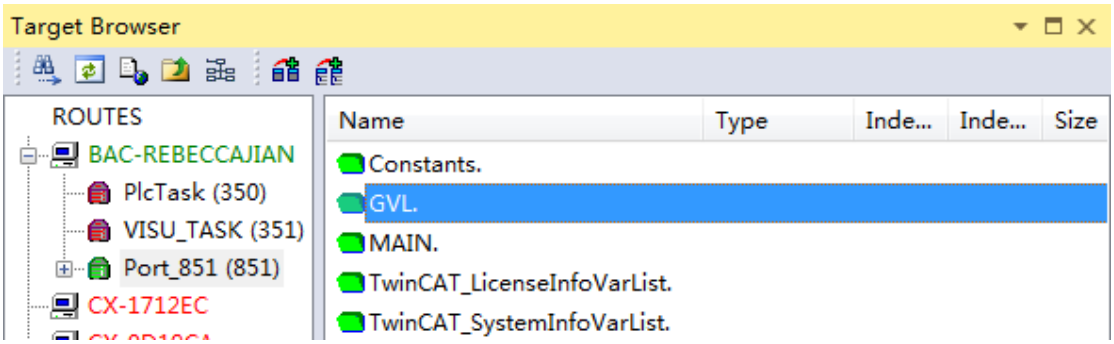




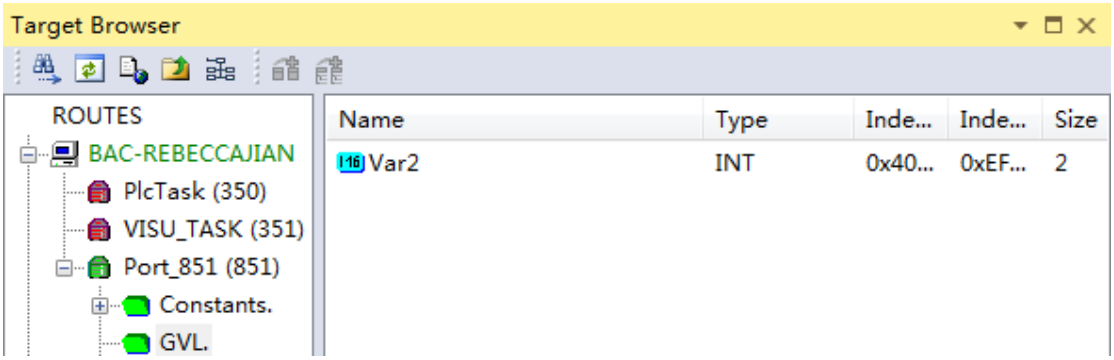
激活并运行，再新建一个 Measurement Project，右键 Axis 选择 Target Browser



选择对应的控制器名和端口，找到 GVL 文件夹



双击打开后可以看到，只有一个 Var2 变量，另一个变量被隐藏了

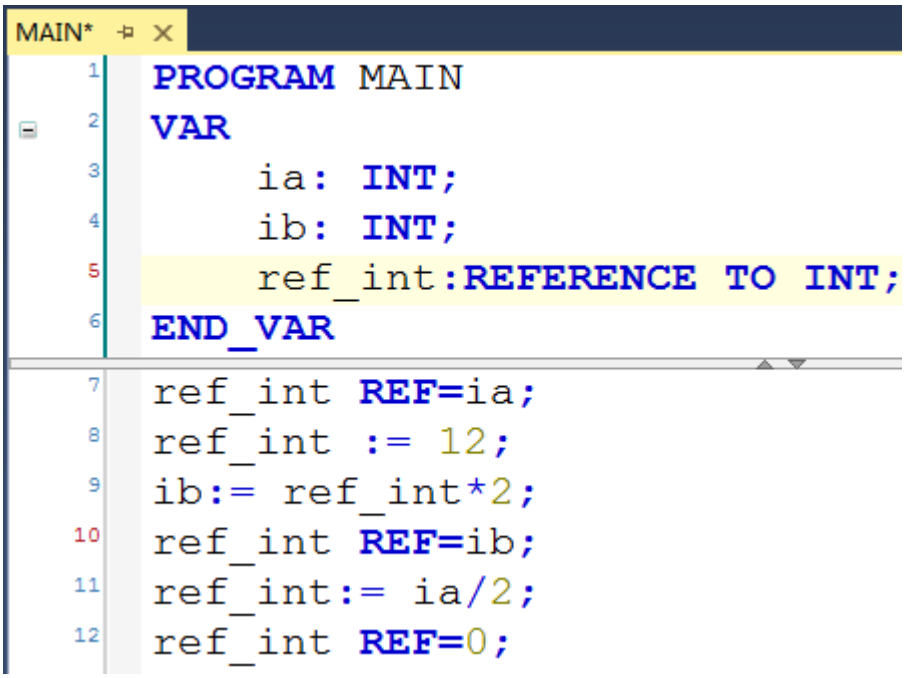


之所以看到这样的情况是因为我们的数据传输都是通过 ADS 通讯进行的，我们的 TwinCAT3 把数据送出去以后，Scope View 通过 ADS 获取变量和变量值。而{attribute 'tc_no_symbol'}这个属性会使得它无法找到这个变量。

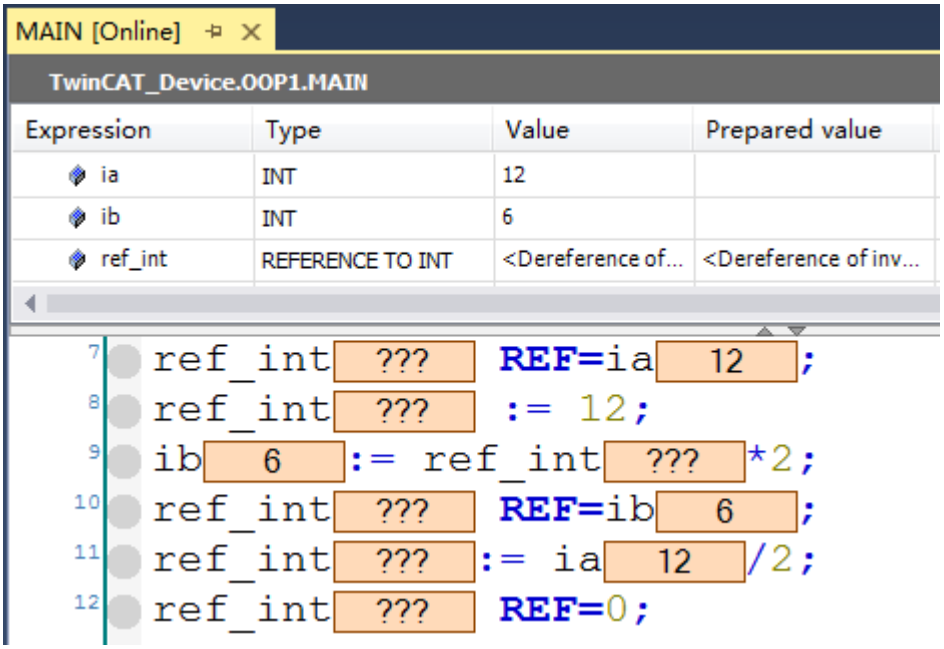
我们程序写好封装好以后，如果部分变量不想让第三方软件通过 ADS 通讯获取到，我们就可以用这个属性来完成。

1.4 Reference 引用：引用类型的变量可以看做是一个对象的别名。别名可以通过标识符来读写。而引用的作用同指针型变量，可代替指针型变量，更为安全。与指针的不同之处在于，赋值给引用类型变量，值会直接作用到它引用的变量。且引用变量的地址必须通过独立的赋值操作来设置。它的语法结构也很简单，<标识符> : REFERENCE TO <数据类型>，这样的定义使我们的这个变量可以去引用同一空间名下所有该类型的变量。

（1）编写程序如下



（2）激活并 Login 之后效果如下，可以看到 ia 和 ib 变量值分别为 12 和 6。指针引用的值我们也只能通过添加断点的方式来查看。



（3）通过添加断点的方式来查看引用和赋值过程，把它复位就可以看到下图的

情况。

```
ref_int ???  REF=ia 0 ;
ref_int ???  := 12;
ib 0 := ref_int ??? *2;
ref_int ???  REF=ib 0 ;
ref_int ??? := ia 0 /2;
ref_int ???  REF=0;
```

第一步：ref_int 变量指向 ia 变量，此时 ia=0，ib=0，ref_int=0。

```
ref_int 0  REF=ia 0 ;
ref_int 0  := 12;
ib 0 := ref_int 0 *2;
ref_int 0  REF=ib 0 ;
ref_int 0 := ia 0 /2;
ref_int 0  REF=0;
```

第二步：给 ref_int 赋值 12，此时 ia=12，ib=0，ref_int=12。

```
ref_int 12  REF=ia 12 ;
ref_int 12  := 12;
ib 0 := ref_int 12 *2;
ref_int 12  REF=ib 0 ;
ref_int 12 := ia 12 /2;
ref_int 12  REF=0;
```

第三步： ref_int 变量乘以 2 的值赋给 ib 变量，此时 ia=12，ib=24，ref_int=12。

```
ref_int 12  REF=ia 12 ;
ref_int 12  := 12;
ib 24 := ref_int 12 *2;
ref_int 12  REF=ib 24 ;
ref_int 12 := ia 12 /2;
ref_int 12  REF=0;
```

第四步： 把变量 ib 的值赋给 ref_int 变量，此时 ia=12，ib=24，ref_int=24。

```
ref_int 24  REF=ia 12 ;
ref_int 24  := 12;
ib 24 := ref_int 24 *2;
ref_int 24  REF=ib 24 ;
ref_int 24 := ia 12 /2;
ref_int 24  REF=0;
```

第五步：把 ia 变量除以 2 的值赋给 ref_int 变量，此时 ia=12，ib=6，ref_int=6。

```
ref_int 6  REF=ia 12 ;
ref_int 6  := 12;
ib 6 := ref_int 6 *2;
ref_int 6  REF=ib 6 ;
ref_int 6 := ia 12 /2;
ref_int 6  REF=0;
```

如果继续单步执行，就会看到 ref_int 清零。

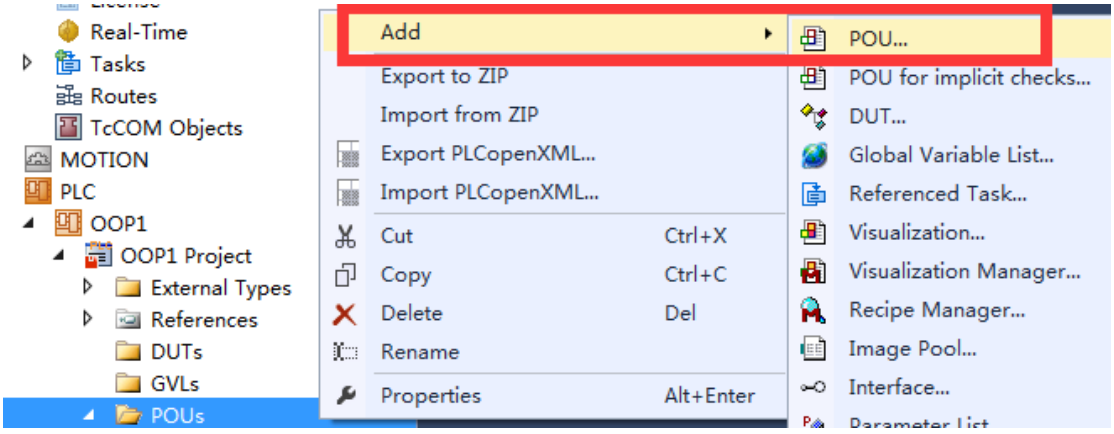
2. OOP 关键词的介绍与用法

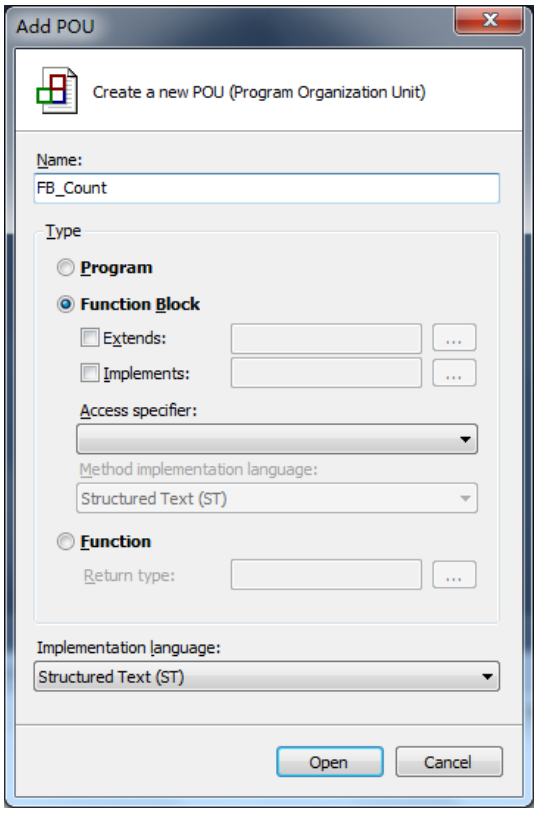
概念说明：
Method：方法是带自定义变量并且依附于功能块，描述了一系列的动作指令。
Property：属性提供了 Get 和 Set 方法对内部变量的状态或者值进行读取和设置。
This：一种指针，引用当前功能块的方法、变量或属性。
Super：一种指针，引用父类功能块的方法、变量或属性。
Extends：继承，获得继承对象的方法，变量和属性等，继承的对象可以是功能块或者结构类型变量。
Interface：接口，一种复杂的结构类型，包含变量，方法，属性。（类似于没有代码的功能块）
Implements：功能块通过 implements 对一个接口类型进行实现。

这里做一个计数的例子来演示所有的关键词用法

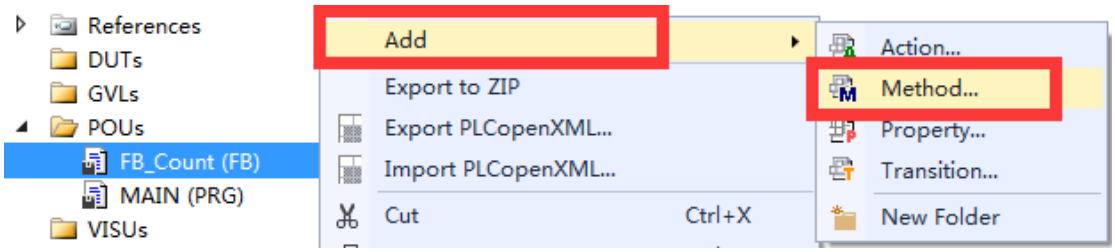
2.1 Method 方法

（1）方法不能独立存在，它要依附于功能块，所以先创建一个功能块，取名 FB_Count。

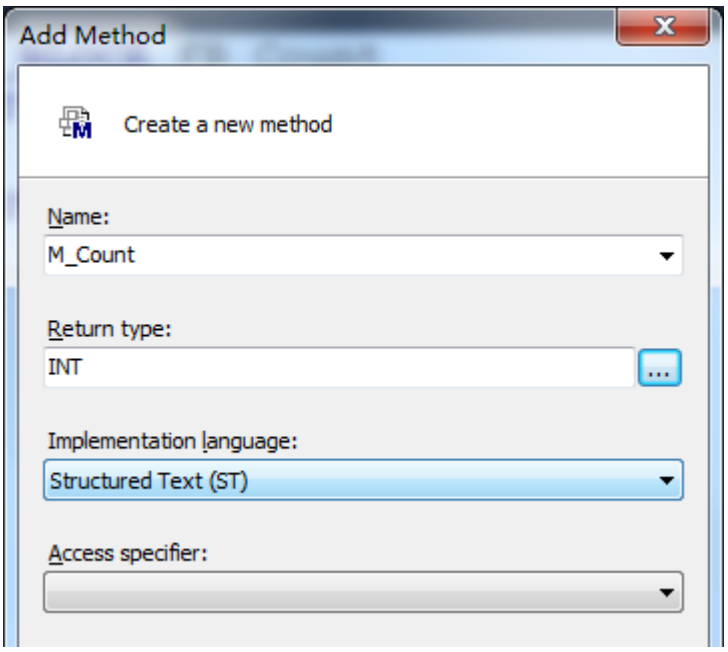




（2）右键功能块 Add Method



（3）命名其为 M_Count，并且选择返回类型为 INT。



（4）在 M_Count 这个 Method 中编写程序。并且注意把 bEnable 申明在 Method 中的 VAR_INPUT 关键字下，而把 i_Wert 和 k_wert 申明在 VAR_OUTPUT 关键字下，同时注意把 i、k、max_k 和 max_i 变量申明在 Function Block 中，作为整个功能块的内部变量。（如果是自动申明变量记得在 Scope 和 Object 的下拉菜单里选择对应项）

FB_Count.M_Count

```
{attribute 'object_name' := 'M_Count'}
METHOD M_Count
VAR_INPUT
    bEnable : BOOL;
END_VAR
VAR_OUTPUT
    i_Wert  : INT;
    k_Wert  : INT;
END_VAR
```

```
IF bEnable THEN
    i:=i+1;
    IF i>=max_i THEN
        i:=0;
    END_IF
    k:=k+1;
    IF k>=max_k THEN
        k:=0;
    END_IF
    i_Wert:=i;
    k_Wert:=k;
END_IF
```

FB_Count

```
FUNCTION_BLOCK FB_Count IMPLEMENTS I_Count
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    i,k: INT;
    max_i:INT:=755;
    max_k: INT:=2265;
END_VAR
```

（5）在主程序里把功能块实例化，然后调用这个 Method，编写程序如下

```
MAIN*
1  PROGRAM MAIN
2  VAR
3      FB_Count:FB_Count;
4      StartCount: BOOL;
5      i: INT;
6      k: INT;
7  END_VAR
```

```
1  FB_Count.M_Count
2      (bEnable:=StartCount ,
3      i_Wert=>i ,
4      k_Wert=>k );
```

（6）Activate Configuration 之后，Login 并 run，给 StartCount 赋值 TRUE

MAIN [Online]

TwinCAT_Device.OOP1.MAIN

Expression	Type	Value	Prepared value	Address
FB_Count	FB_Count			
StartCount	BOOL	FALSE	TRUE	
i	INT	0		
k	INT	0		

```
1  FB_Count.M_Count
2      (bEnable:=StartCount FALSE <TRUE> ,
3      i_Wert=>i 0 ,
4      k_Wert=>k 0 );RETURN
```

（7）可以看到 i 和 k 一直在做累加，i 的值从 0 到 755，而 k 的值从 0 到 2265。

MAIN [Online]

TwinCAT_Device.OOP1.MAIN

Expression	Type	Value	Prepared value
FB_Count	FB_Count		
StartCount	BOOL	TRUE	
i	INT	698	
k	INT	2208	

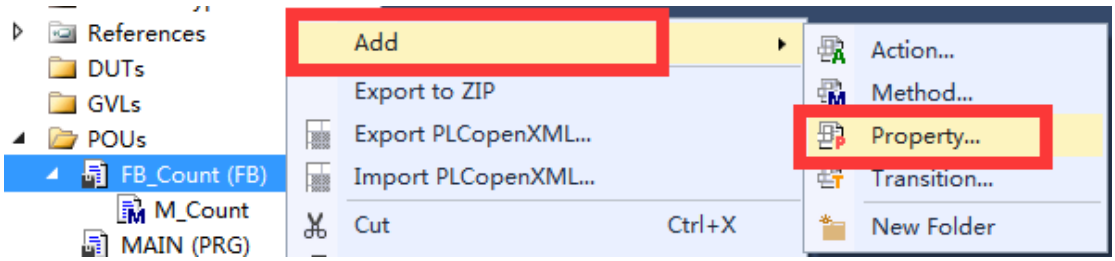
```
1  FB_Count.M_Count
2      (bEnable:=StartCount TRUE ,
3      i_Wert=>i 698 ,
4      k_Wert=>k 2208 );RETURN
```

2.2 Property 属性

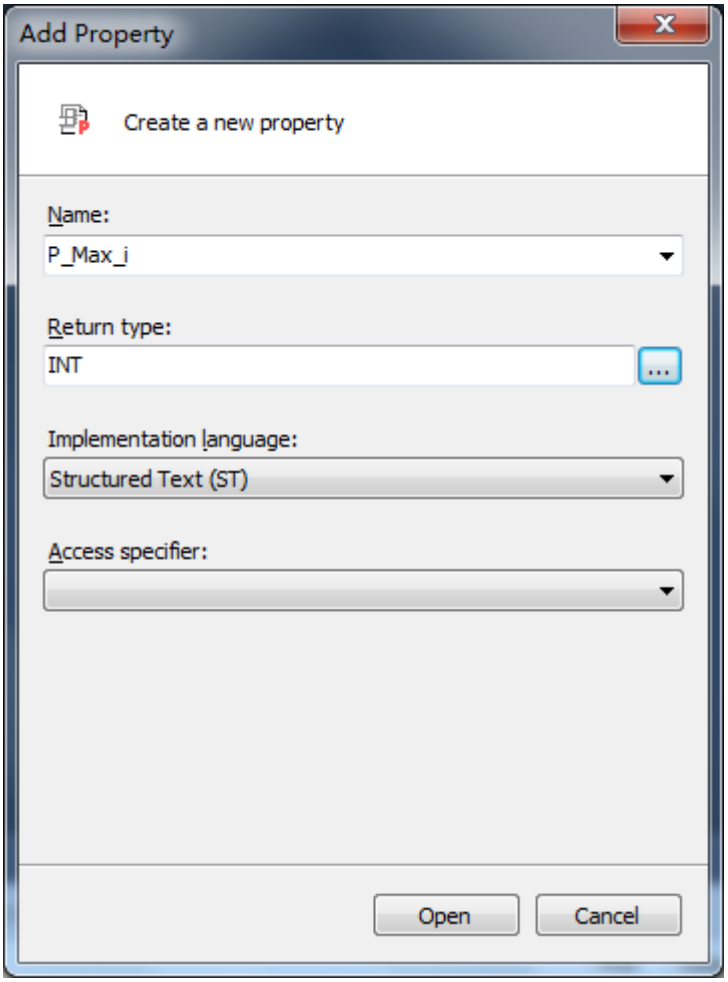
在刚刚那个 Method 中，我们把 bEnable 作为输入使能，把 i_Wert 和 k_Wert 作为输出返回，它们都是需要随时改动数值或者需要实时显示数值的一类变量。而作为 Function Block 的内部变量是不是就无法在线修改了呢？

当然可以，这就要用到 Property 属性。

（1）右键功能块 Add Property。



（2）修改 property 名为 P_Max_i，返回类型设置为 INT 型。



（3）分别对应 property 中 2 个 method 写入程序， Get 可以对 property 所对应的变量进行读取。

FB_Count.P_Max_i.Get*

```
VAR
END_VAR
```

```
P_Max_i:=max_i;
```

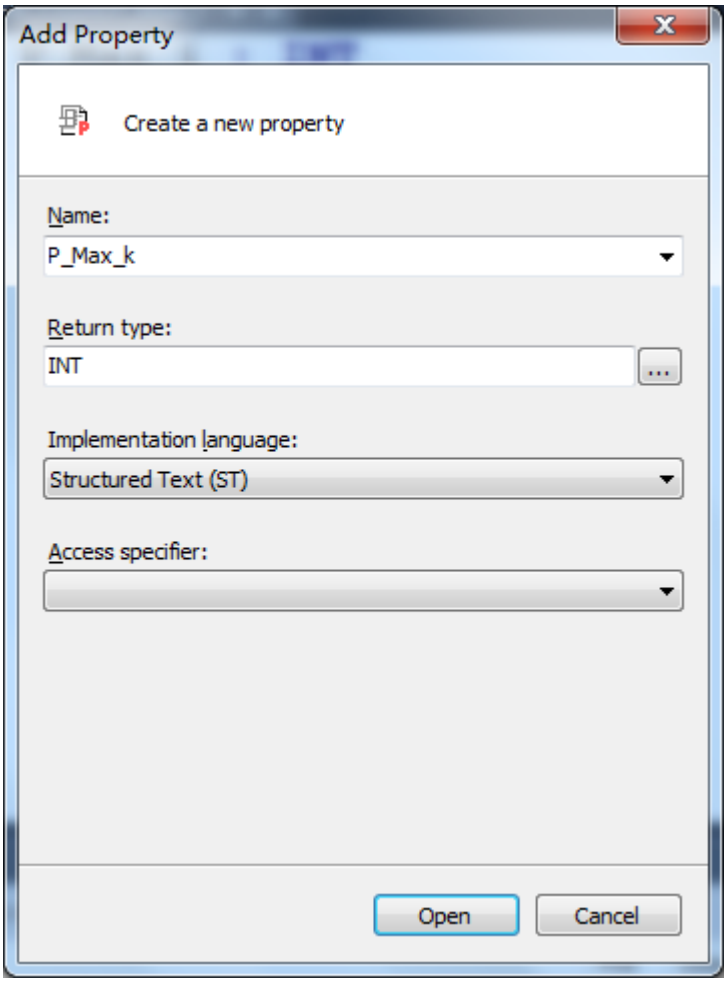
（4）Set 可以对 property 所对应的变量进行写入。

FB_Count.P_Max_i.Set*

```
VAR
END_VAR
```

```
max_i:=P_Max_i;
```

（5）同样我们再建立一个 Property，同样在 Get 和 Set 中实现对 max_k 这个变量的读写。



FB_Count.P_Max_k.Get*

```
VAR
END_VAR
```

```
P_Max_k:=max_k;
```

FB_Count.P_Max_k.Set*

```
VAR
END_VAR
```

```
max_k:=P_Max_k;
```

（6）只要在主程序调用这两个属性，就可以实现读写操作了

```
MAIN*
7       max_i: INT;
8       max_k: INT;
9   END_VAR
10

5
6   FB_Count.P_Max_i:=1000;
7   max_i:=FB_Count.P_Max_i;
8
9   FB_Count.P_Max_k:=2000;
10  max_k:=FB_Count.P_Max_k;
```

（7）Activate Configuration 之后，Login 并 Run，可以看到 i 和 k 一直在做累加，i 的值从 0 到 1000，而 k 的值从 0 到 2000。

MAIN [Online]

TwinCAT_Device.OOP1.MAIN

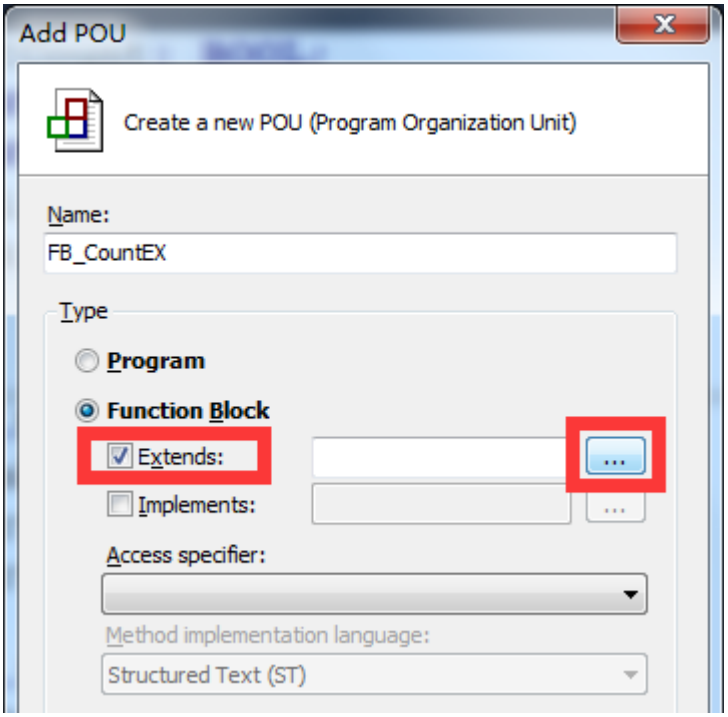
Expression	Type	Value	Prepared value
FB_Count	FB_Count		
StartCount	BOOL	TRUE	
i	INT	707	
k	INT	1707	
max_i	INT	1000	
max_k	INT	2000	

```
1   FB_Count.M_Count
2       (bEnable:=StartCount TRUE ,
3       i_Wert=>i 707 ,
4       k_Wert=>k 1707 );
5
6   FB_Count.P_Max_i:=1000;
7   max_i 1000 :=FB_Count.P_Max_i;
8
9   FB_Count.P_Max_k:=2000;
10  max_k 2000 :=FB_Count.P_Max_k;
```

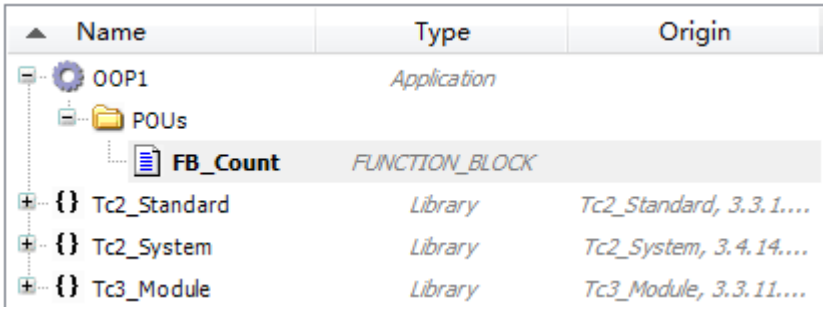
2.3 Extends 继承

现在想在此基础上增加一个检测 i 与 k 两变量是否相等的功能，但是仍然希望保留原来的功能块，那么可以用到 Extends 来完成任务。

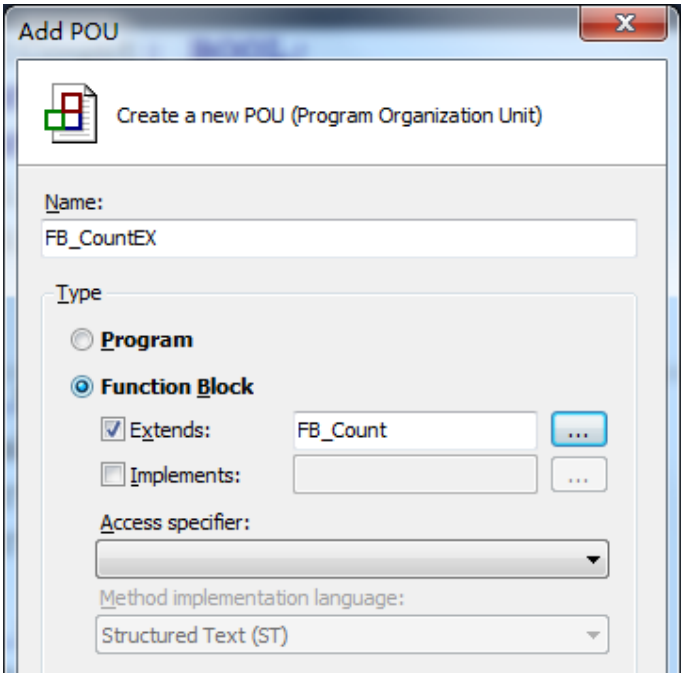
（1）新建功能块，取名为 FB_CountEX，勾选 Extends，点击选项框。



（2）在选项框中找到被扩展的功能块 FB_Count。

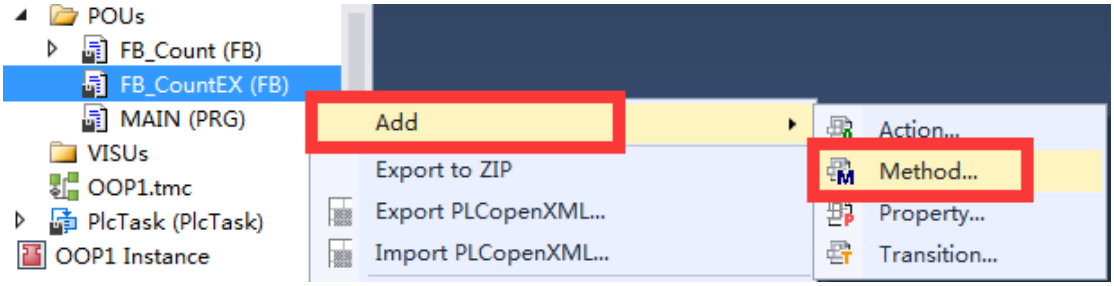


（3）选中后点击 OK 可以发现被扩展功能块名出现在 Extends 框中，点击 Open。

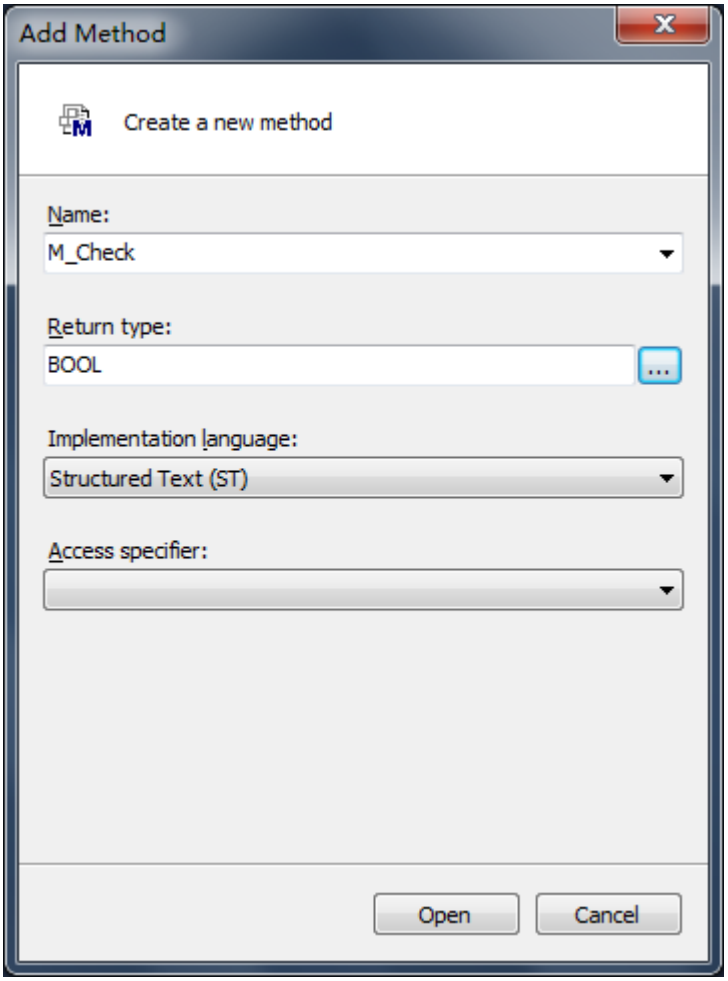


（4）功能块 FB_CountEX 已经继承了 FB_Count 所有变量，方法和属性，我们可以对于 FB_CountEX 功能块进行添加新的方法和属性进行扩展，当然也可以对于所继承的原有的方法和属性进行重写，接下来就演示如何进行扩展

右键 FB_CountEX 新建 Method。



（5）命名此 Method 为 M_Check，并且返回类型选择 BOOL。



（6）在 M_Check 这个 Method 中编写代码，是的 i 与 k 两变量值相等时，返回 TRUE；不等时返回 FALSE。

FB_CountEX.M_Check

```
METHOD M_Check : BOOL
VAR_INPUT
    bCheckValue: BOOL;
END_VAR
```

```
IF bCheckValue THEN
    IF i=k THEN
        M_Check:=TRUE;
    ELSE
        M_Check:=FALSE;
    END_IF
END_IF
```

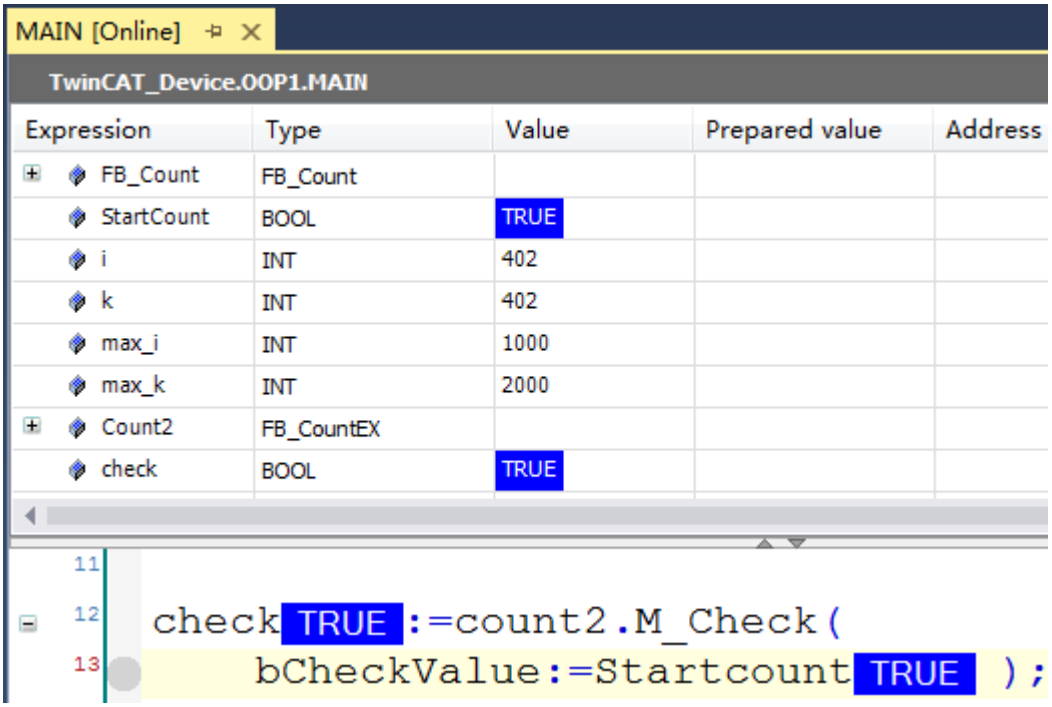
（7）也是在主程序区实例化功能块并调用这个方法

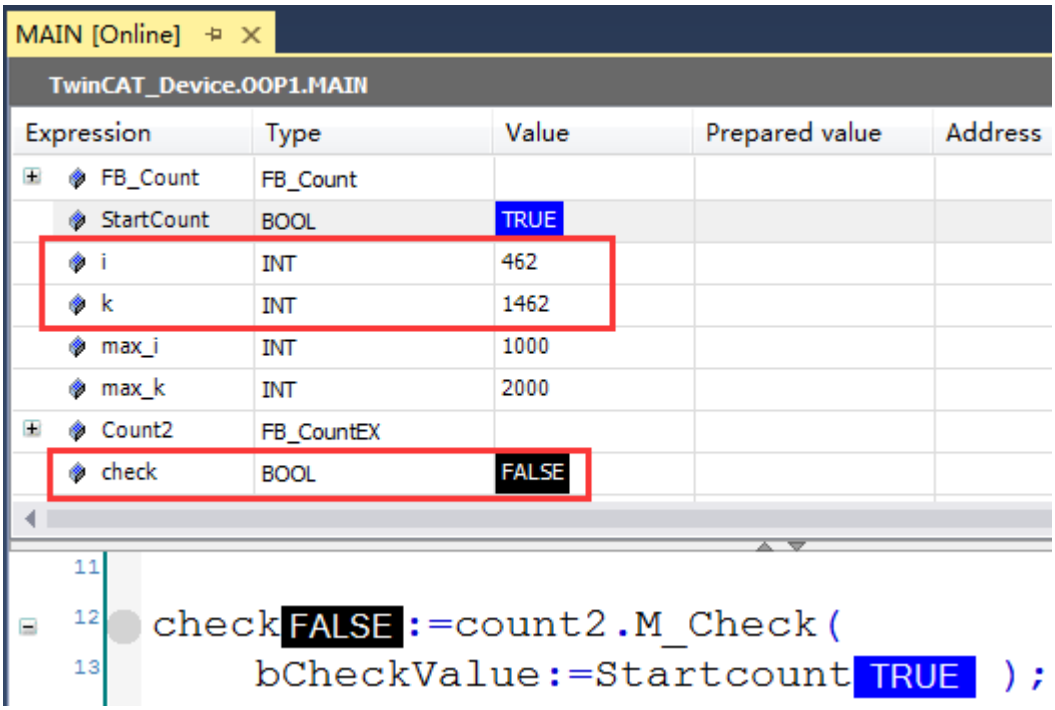
MAIN

```
    Count2:FB_CountEX;
    check: BOOL;
END_VAR
```

```
Count2.M_Count
    (bEnable:=StartCount ,
    i_Wert=>i ,
    k_Wert=>k );

Count2.P_Max_i:=1000;
max_i:=Count2.P_Max_i;

Count2.P_Max_k:=2000;
max_k:=Count2.P_Max_k;

check:=count2.M_Check(
    bCheckValue:=Startcount );
```

（8）Activate Configuration 之后，Login 并 Run，可以发现当 i 与 k 相等时候，check 的值是 TRUE；而当两者不等时，check 的值是 FALSE。

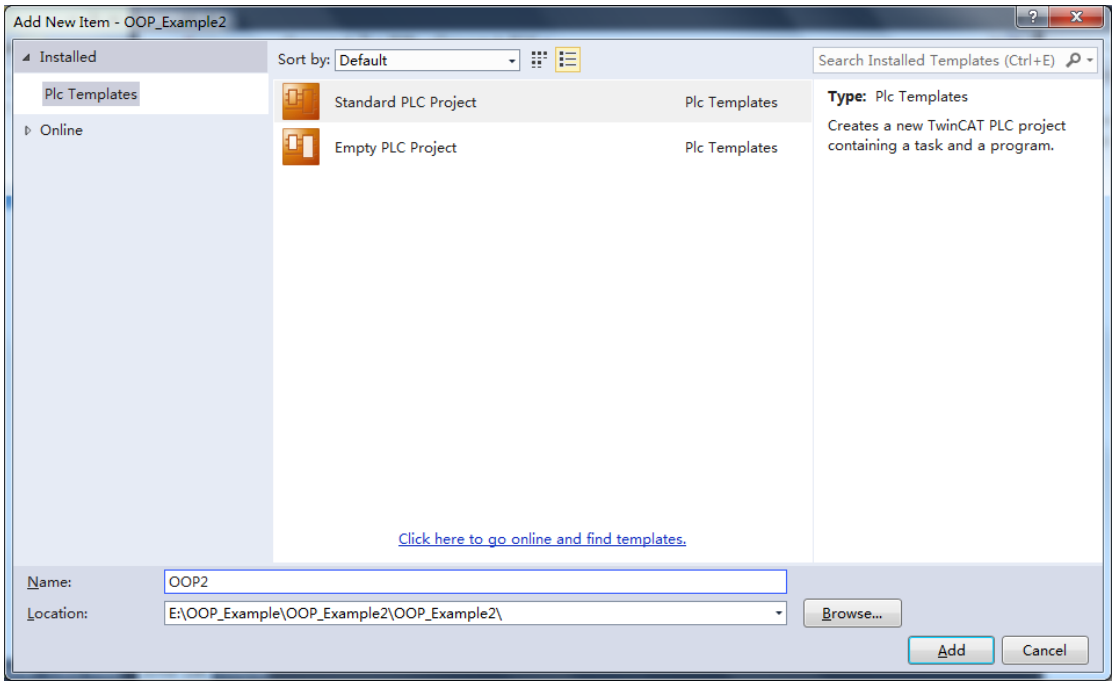




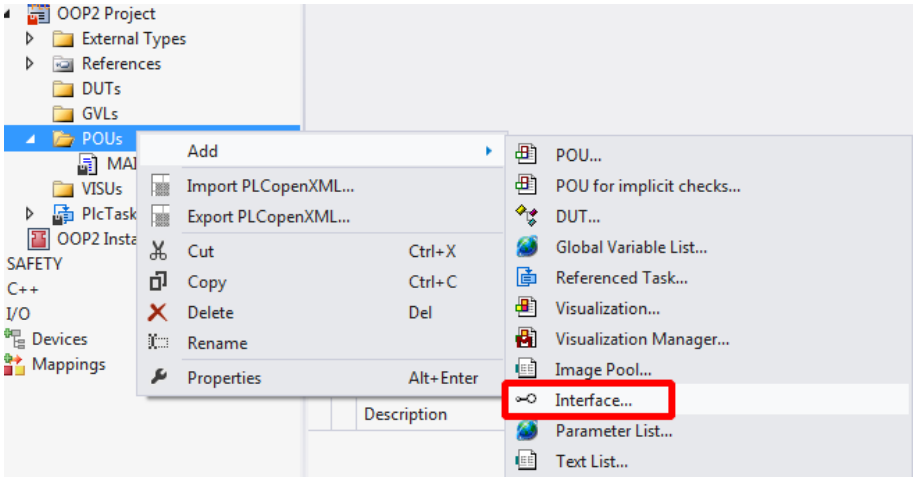
2.4 Interface 接口

当然，像上面这样的调用方式有些麻烦。尤其是当我想要在两个功能块之间切换的时候就需要去修改程序，那么有没有更好的方法呢？接口很大程度上为我们提供了这种便利。

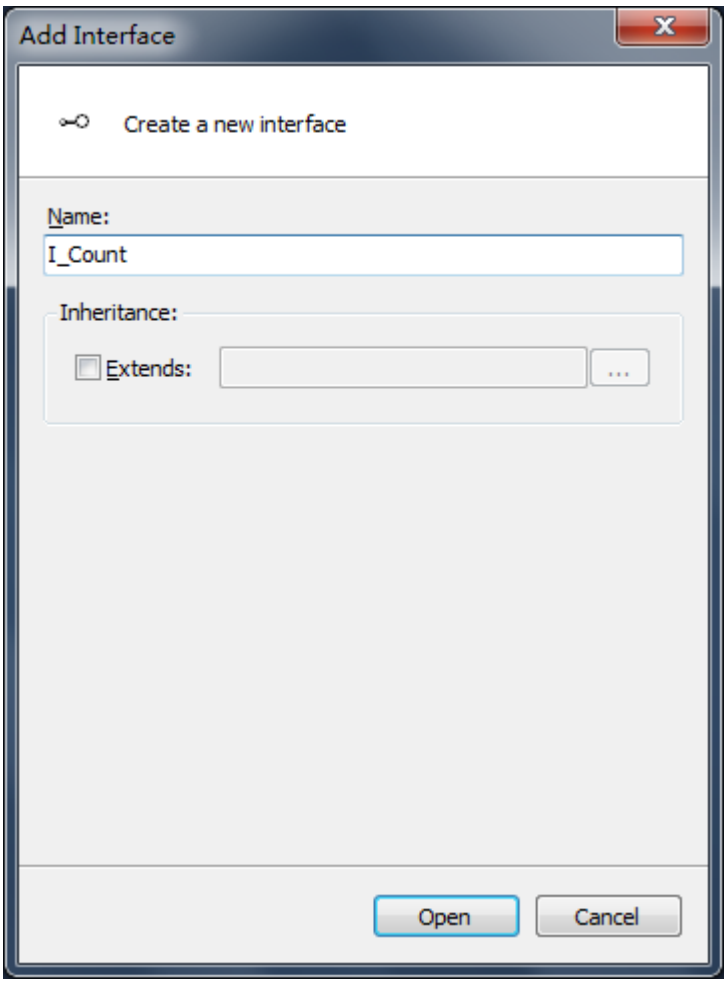
（1）新建 PLC 项目，取名为 OOP2。



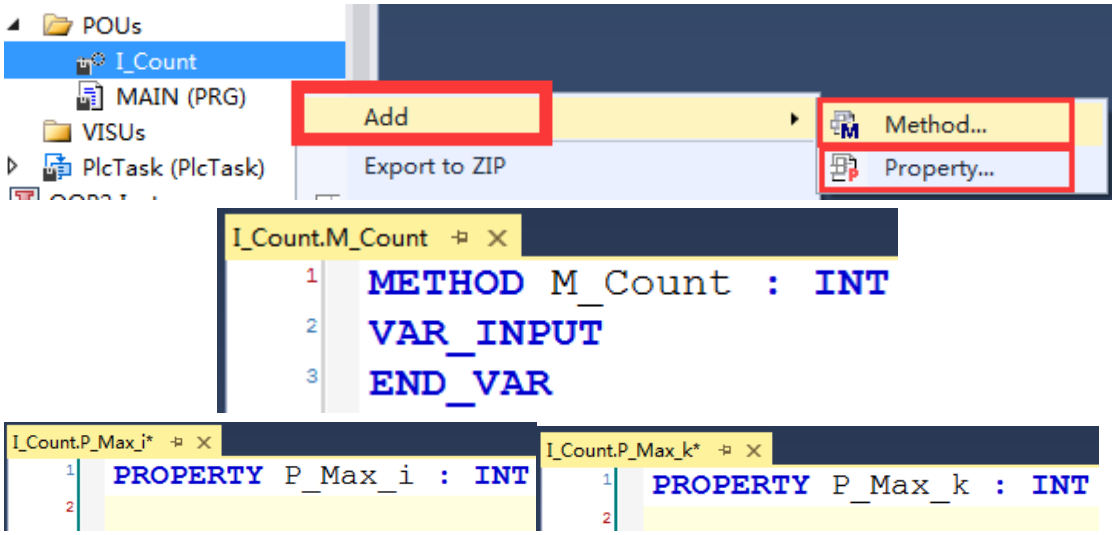
（2）右键 POUs 添加 Interface。



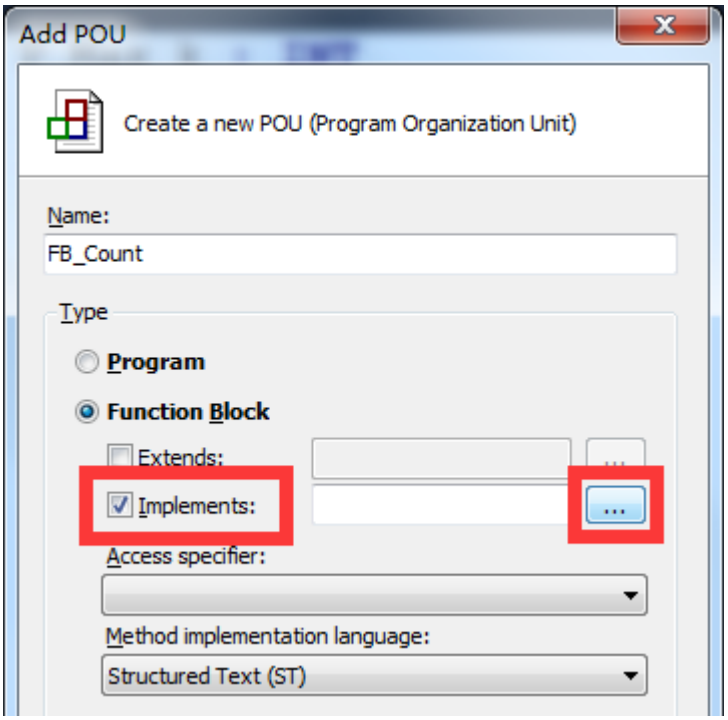
（3）取名为 I_Count 并且点击 OK 进行创建。

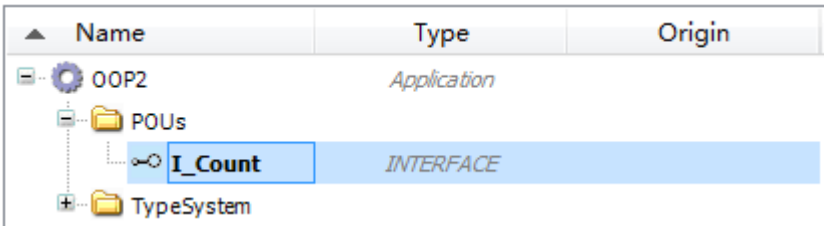


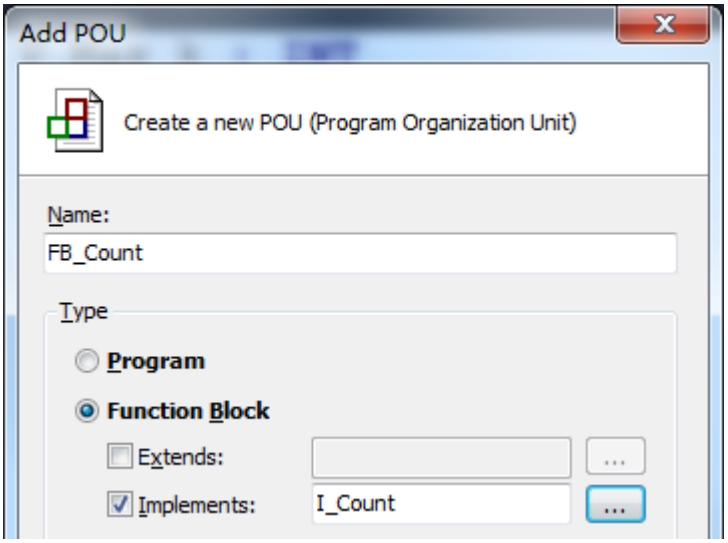
（4）右键 I_Count 添加 1 个方法 M_Count，返回类型选择 INT 型。再添加两个属性 P_Max_i 和 P_Max_k。



（5）接口创建好了后开始创建功能块实现这些接口并且编写实现代码，右键 POUs 新建功能块，取名功能块为 FB_Count，并且勾选 Implements，在选项框选择之前创建好的接口 I_ Count。







（6）完成实现接口后就可以在其功能块下的 Count 方法中编写实现代码，属性中也需要添加代码。两者的代码与之前讲 method，property，extends 的用法时编写的代码一致。

（7）接下来再建一个功能块 FB_CountEX 继承功能块 FB_Count，代码与之前讲 method，property， extends 的用法时候编写的程序一致。

（8）随后在程序中就可以利用接口的实例化实现多个对象的切换。

MAIN*

```
PROGRAM MAIN
VAR
    count1: FB_Count;
    count2: FB_CountEX;
    icount: I_Count;
    i,k :INT;
    Startcount:BOOL;
    check: BOOL;
    max_i: INT;
    max_k: INT;
    change: BOOL:=TRUE;
END_VAR
```

```
IF change THEN
    icount:=count1;
ELSE
    icount:=count2;
    check:=count2.M_Check(bCheckValue:=Startcount );
END_IF
icount.M_Count( bEnable:=Startcount ,
                i_Wert=>i,
                k_Wert=>k );
icount.P_Max_i:=1000;
max_i:=icount.P_Max_i;
icount.P_Max_k:=2000;
max_k:=icount.P_Max_k;
```

（9）Activate Configuration 之后，Login 并 Run

可以观察到当 change 置 TRUE 时，当 i 与 k 相同，Check 的值为 False。

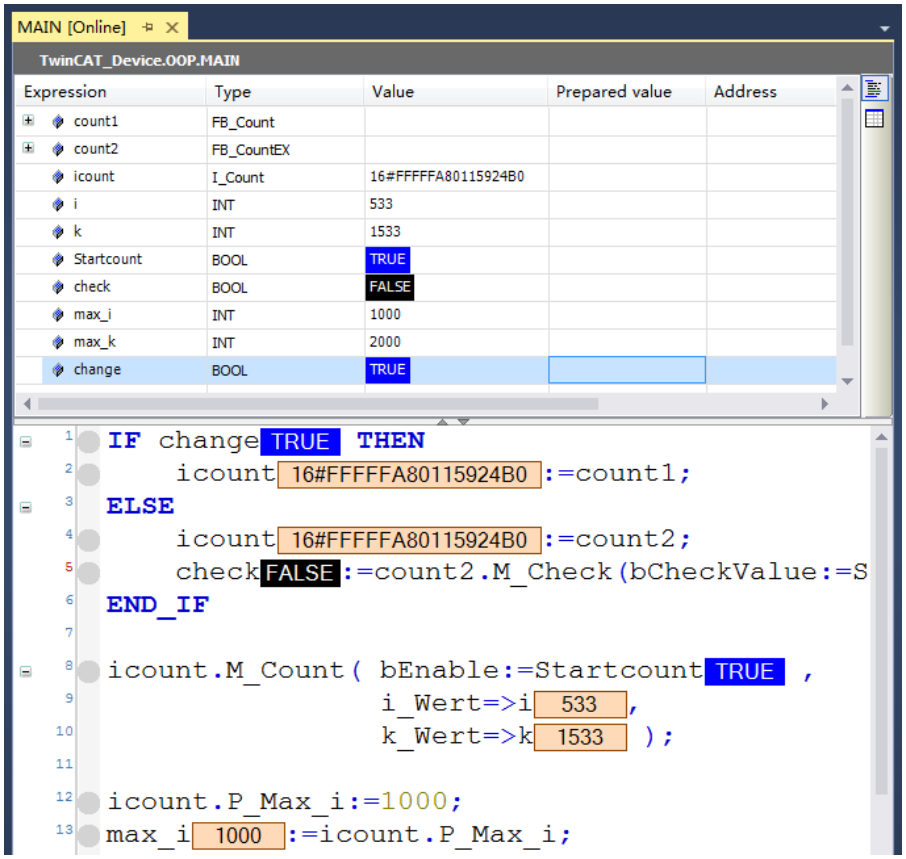
MAIN [Online]

TwinCAT_Device.OOP.MAIN

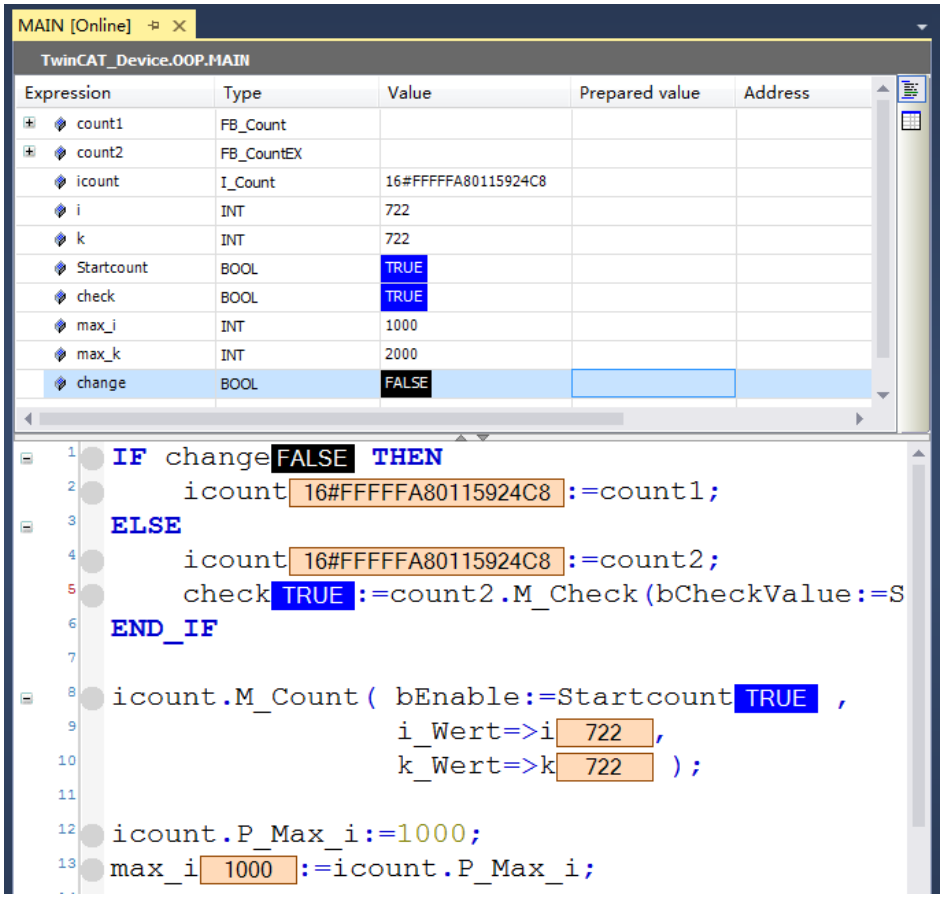
Expression	Type	Value	Prepared value	Address
count1	FB_Count			
count2	FB_CountEX			
icount	I_Count	16#FFFFFA80115924B0		
i	INT	303		
k	INT	303		
Startcount	BOOL	TRUE		
check	BOOL	FALSE		
max_i	INT	1000		
max_k	INT	2000		
change	BOOL	TRUE		

```
IF change TRUE THEN
    icount 16#FFFFFA80115924B0 :=count1;
ELSE
    icount 16#FFFFFA80115924B0 :=count2;
    check FALSE :=count2.M_Check(bCheckValue:=S
END_IF

icount.M_Count( bEnable:=Startcount TRUE ,
                i_Wert=>i 303 ,
                k_Wert=>k 303 );

icount.P_Max_i:=1000;
max_i 1000 :=icount.P_Max_i;
```

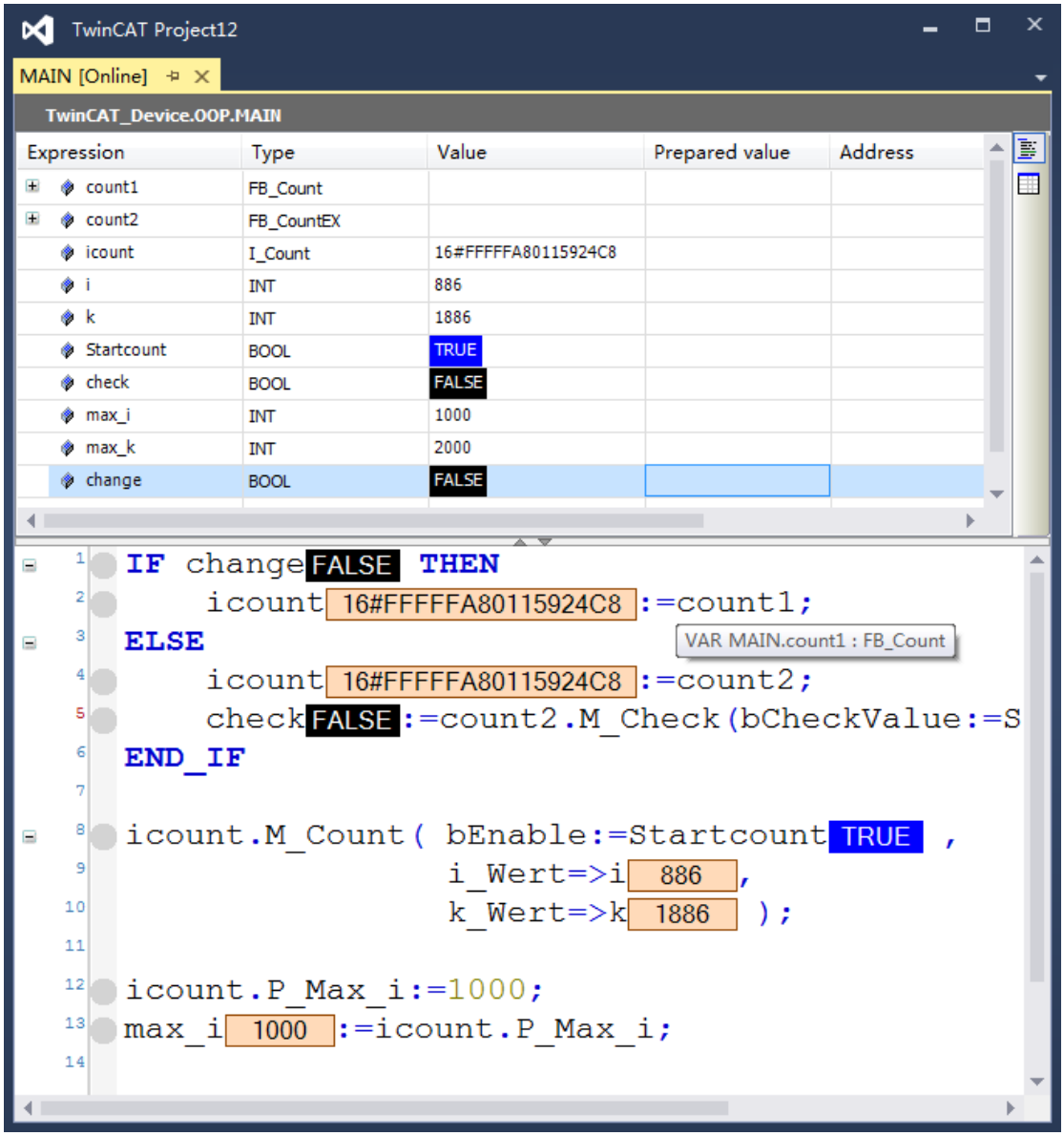
i 与 k 不相同，Check 的值为 False。



而当 change 置 FALSE 时， i 与 k 相同时，Check 的值为 TRUE。



而 i 与 k 不同，Check 的值为 FALSE。



3. OOP 小应用——信号发生器

第三部分，我们要以面向对象的思想完整的来编写一个例子，在这个例子中我们还要学习一下 Property 的进一步用法（具体的介绍两种应用方式），以及刚刚我们没有涉及到的 SUPER 指针和 THIS 指针的用法。

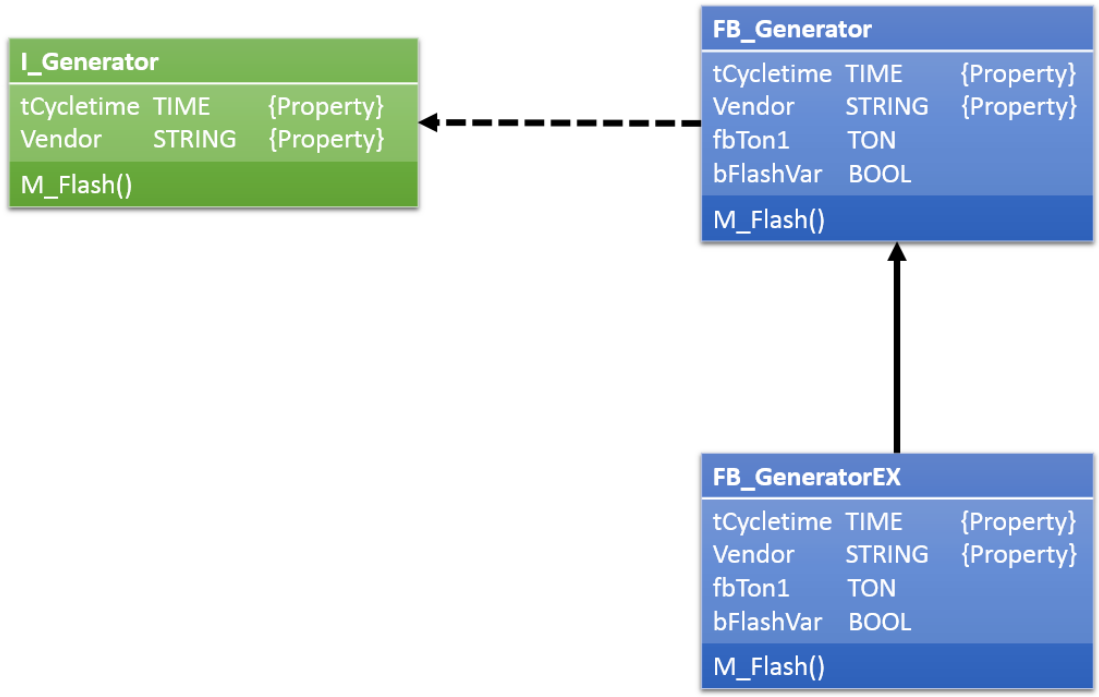
我们设计这样一个场景：
1.现在需要一个信号发生器，初步要求输出一个周期为 1 秒的方波，并且要求这个周期可改。同时标识是由公司 A 开发的此信号发生器。
2.原有方波占空比是 50%，现将其改为 33.3%，同时标识是由公司 A 和公司 B 共同开发。
3.两种波形输出方式要求可在线切换。

先进行一下结构梳理。这样的场景下对象肯定是信号发生器，对于要求方波输出的功能，可以写成这个对象的一个方法。而对于周期可改动的特性，我们不难发现可以把这个周期变量用 Property 来实现，并且我们还可以对这个变量的输入进行一些限定。另外一个标识的功能，也只要通过 Property 的 Get 方法就可以轻易的实现。

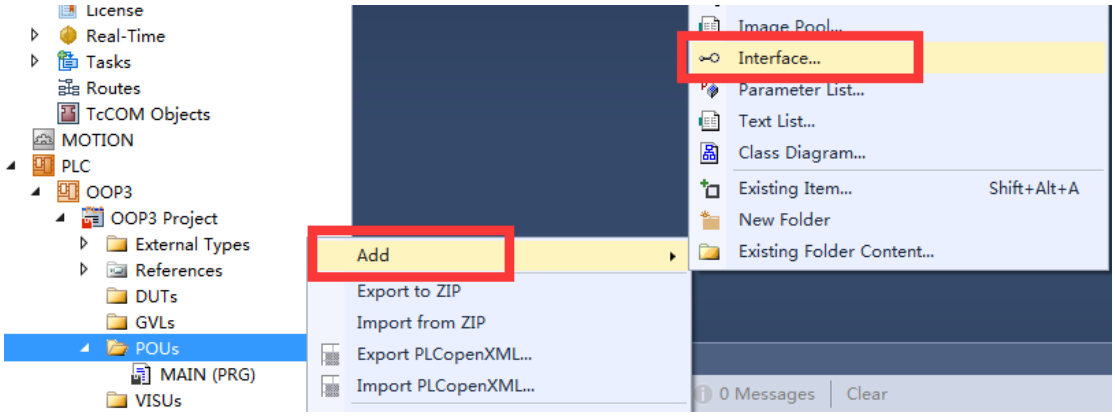
而对于需要改变占空比，我们可以继承原有类，再重写它的方法。

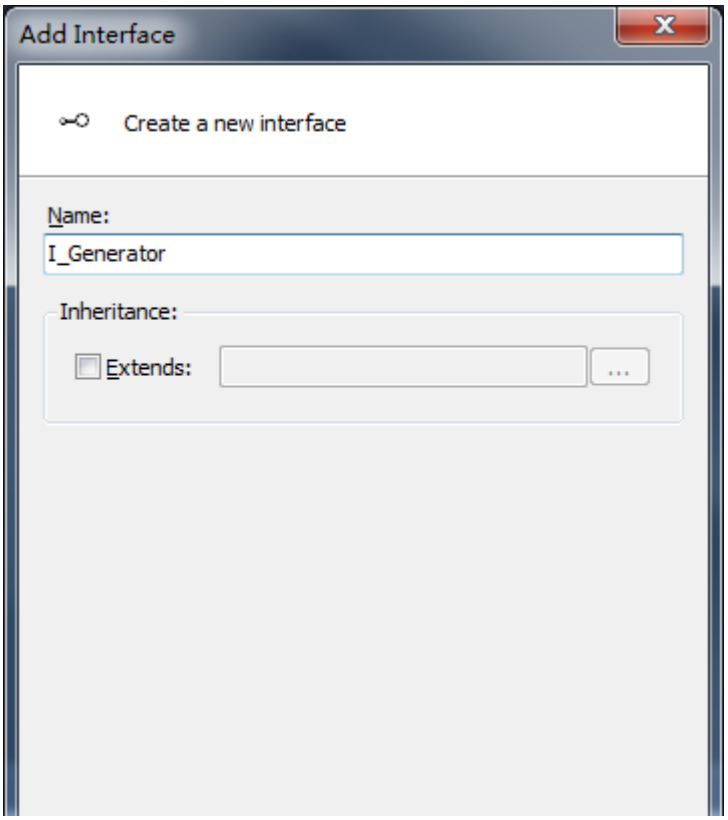
至于要求两种输出方式均保留并可在线切换，可用到接口方便切换。接口的架构通过刚刚对于对象的 Method 和 Property 分析，我们已经能明确了。

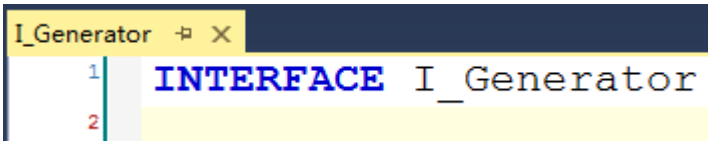
结构如下图：



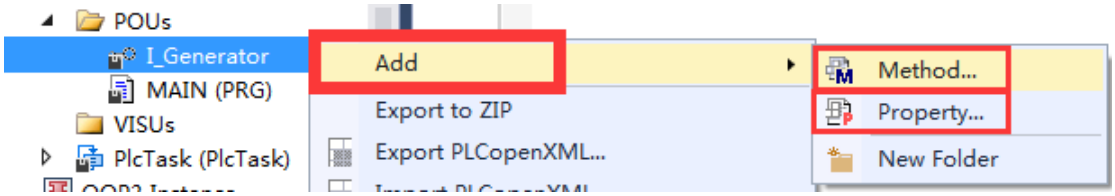
（1）首先要建立接口，搭建一个架构。命名其为 I_Generator



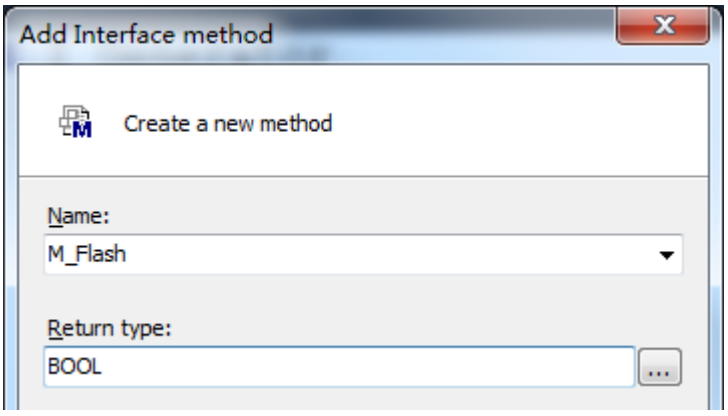




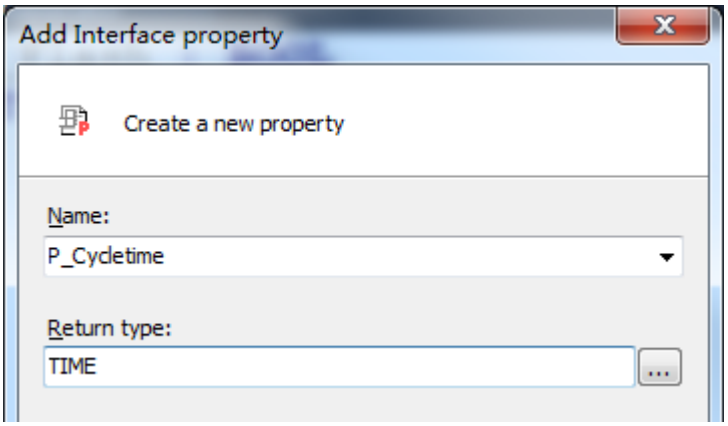
（2）这里就建立基本架构，一个 Method，两个 Property。



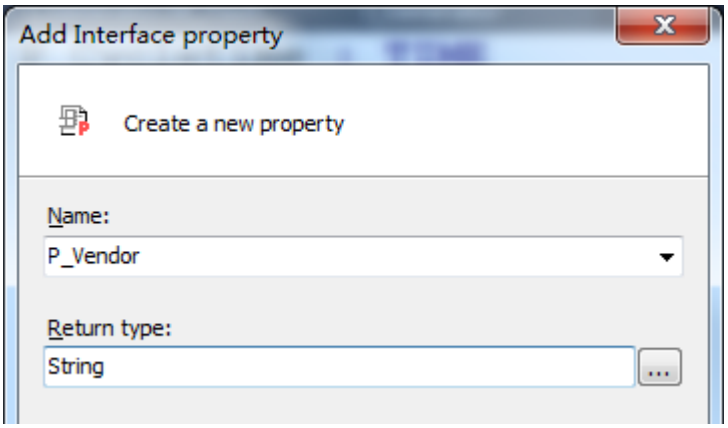
（3）M_Flash 返回类型选择 BOOL，且 M_Flash 这个 Method 中我们不需要输入输出变量

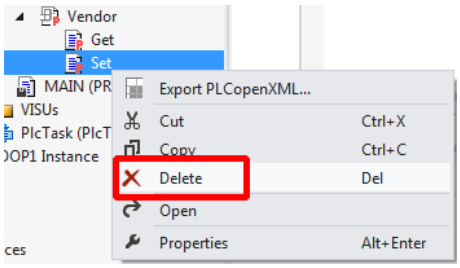


（4）P_Cycletime 这个 property 的返回类型选择 TIME 类型

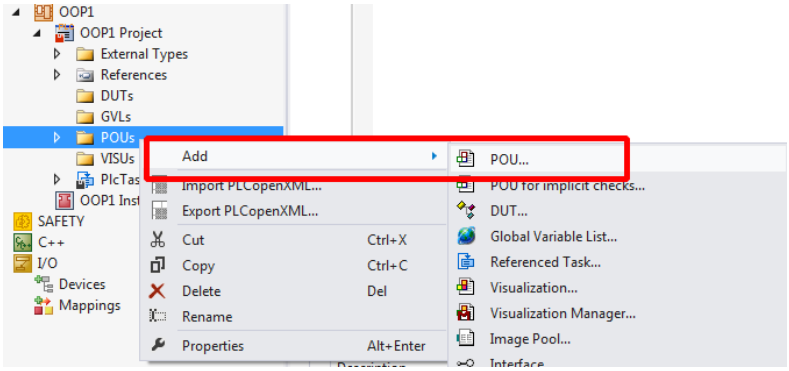


（5）P_Vendor 这个 property 的返回类型选择 STRING 类型，且这里我们把它作为一个标签，所以只读不可写，因此把 Set 属性删掉。这样 P_Vendor 这个属性就不支持写。

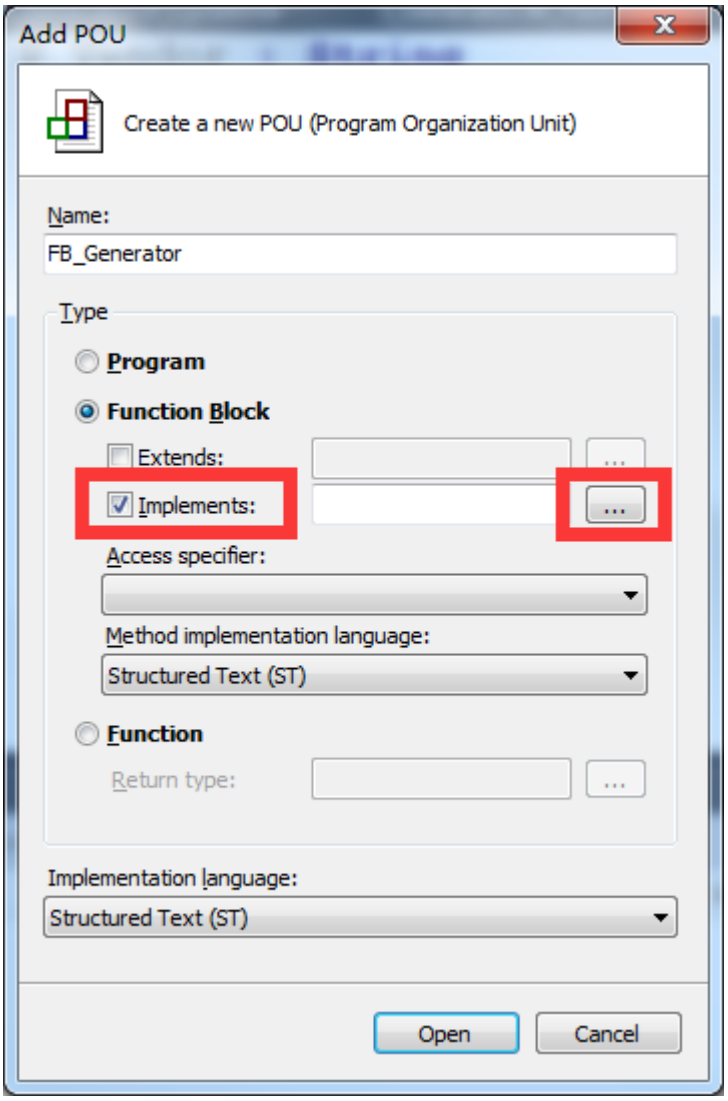




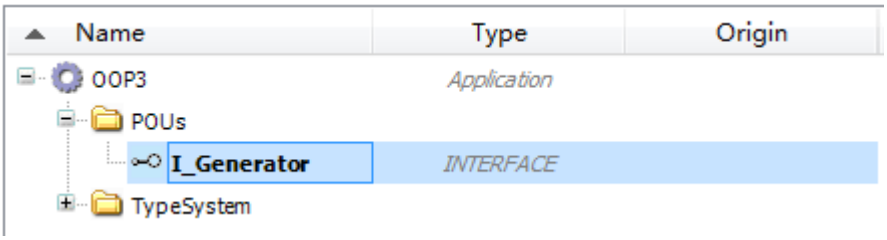
（6）右键 POUs 添加 POU。

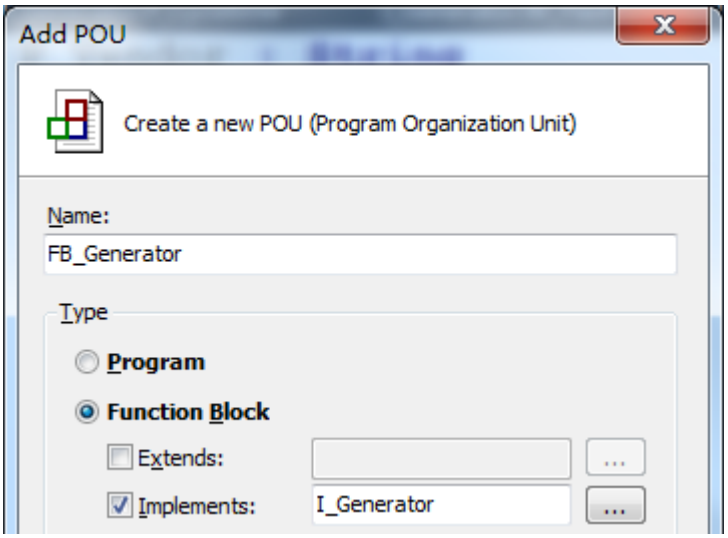


（7）新建一个功能块实现这个接口，命名为 FB_Generator，并勾选 Implements，点击旁边的小按钮

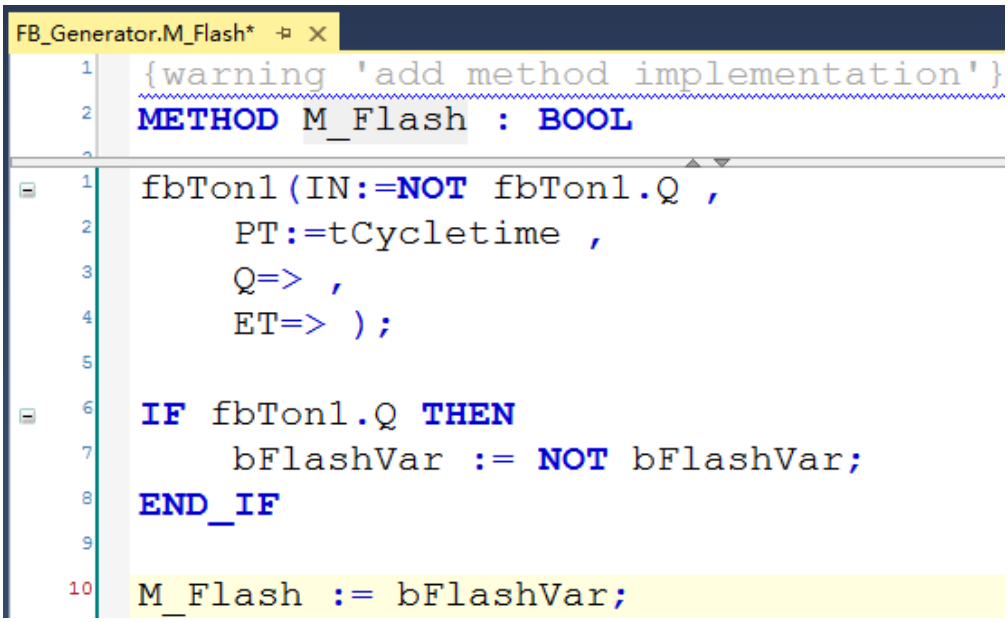


（8）选择建立好的接口 I_Generator。

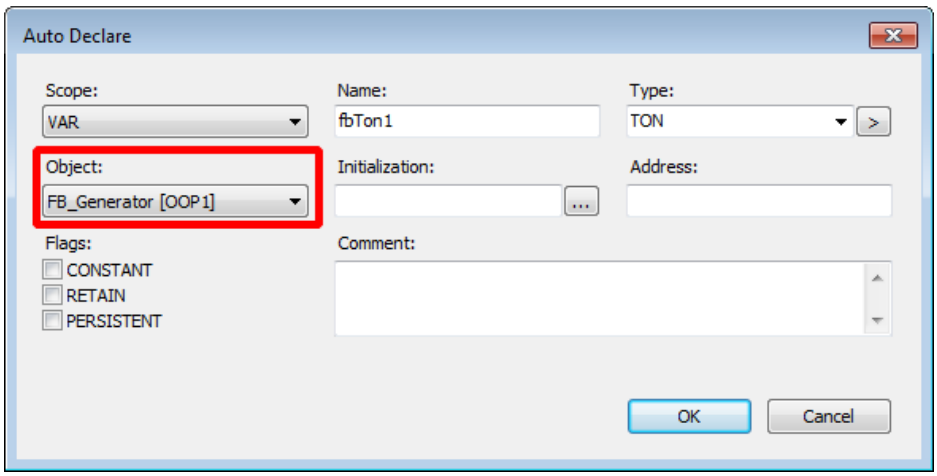




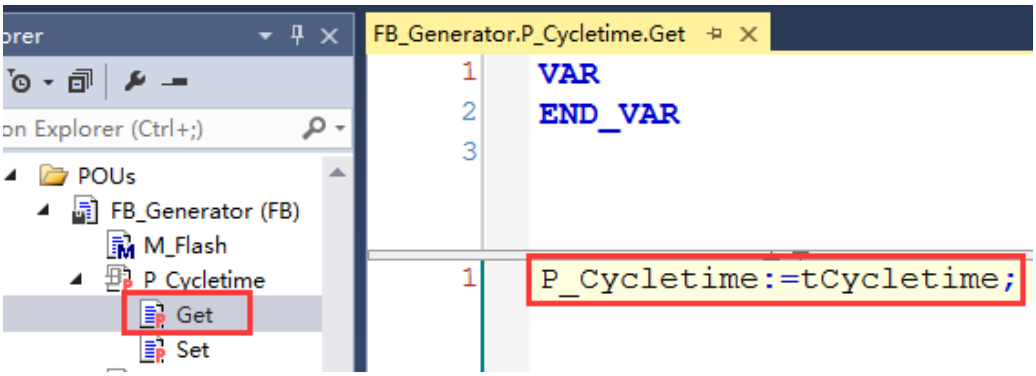
（9）在 M_Flash 这个 Method 中编写程序



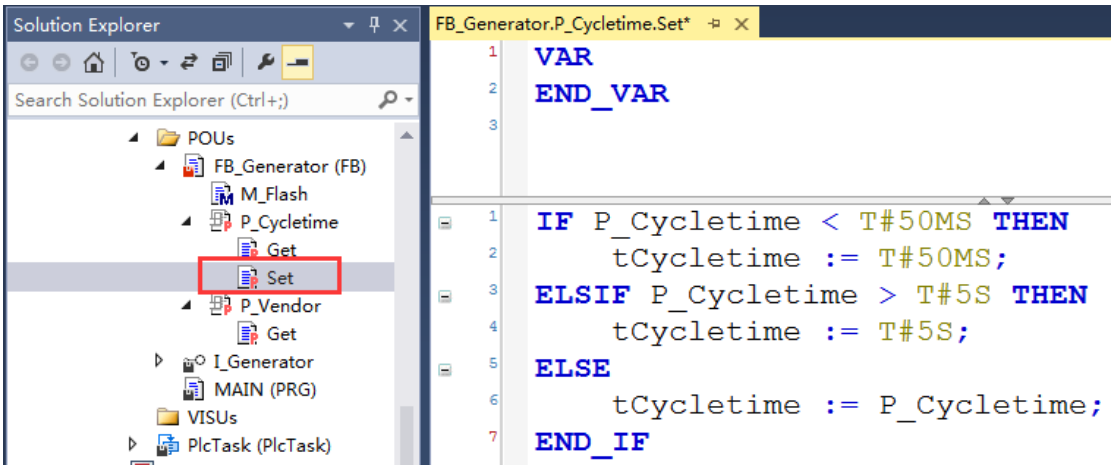
（10）并且注意所有的变量声明都声明在 FB_Generator 中，而不是 M_Flash 中，如下图，Object 都选择 FB_Generator。



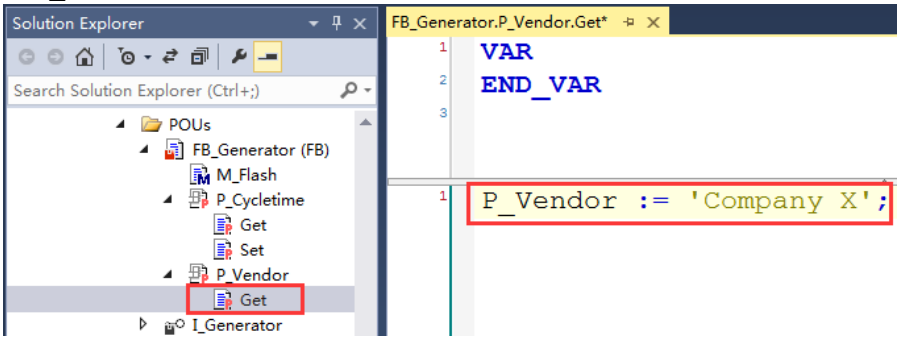
（11）分别对应 P_Cycletime 中 2 个 method，Set 和 Get 进行编程使得 property 所对应的变量可读可写。



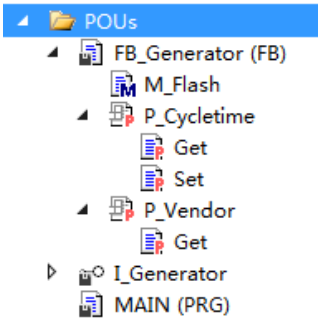
（12）Set 可以通过程序定义为有条件写入。这是我们关于 Property 的第一个应用，即对于输入或输出值进行范围限定。



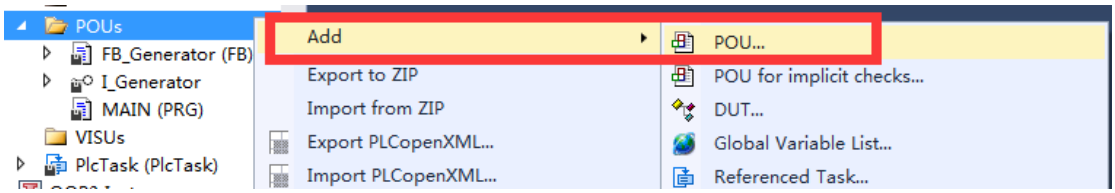
（13）在 P_Vendor 这个标签的 Get 里面编写代码。

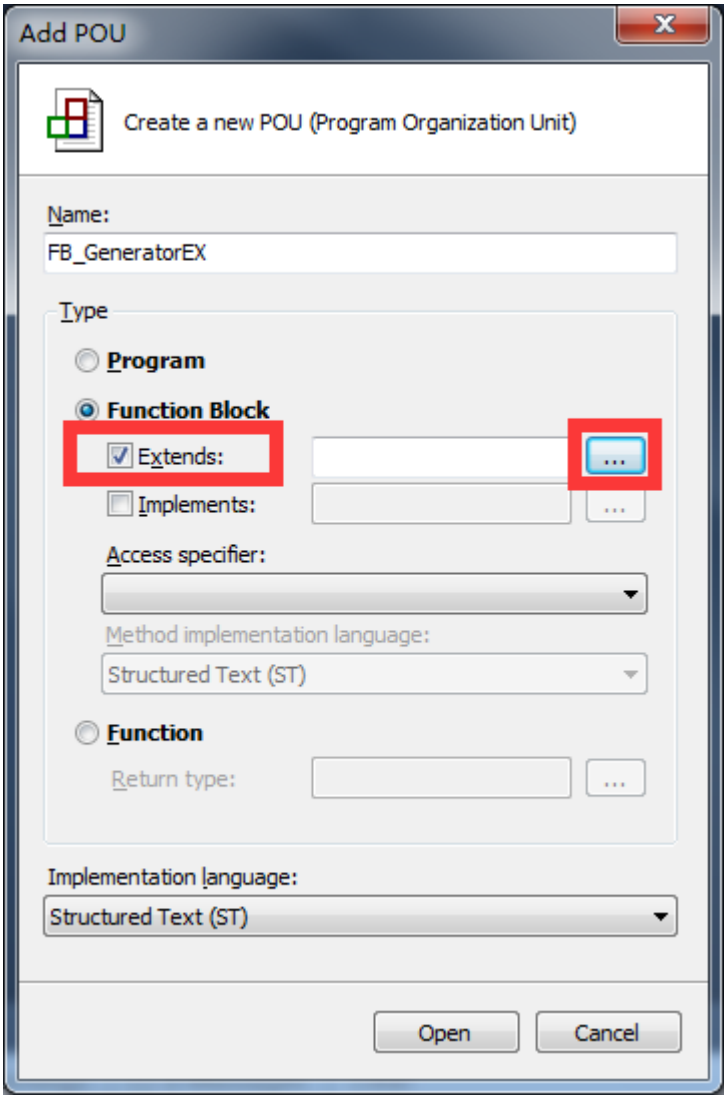


（14）这样一个简单的类就创建好了，里面包含 1 个 method 和 2 个 property。

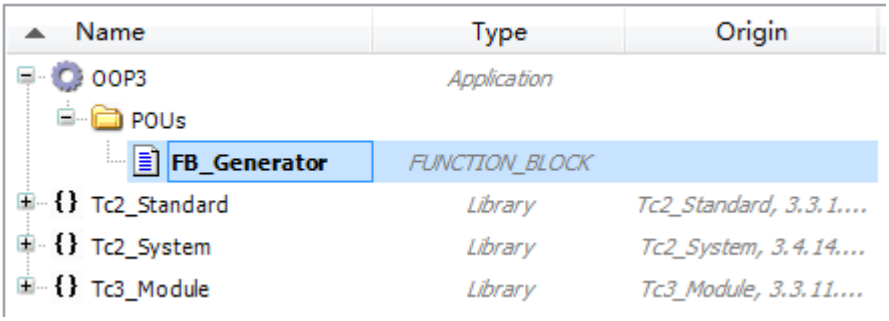


（15）再创建一个 Function Block 用于扩展原有的功能，命名 FB_GeneratorEX，勾选 Ectends，点击旁边的小按钮

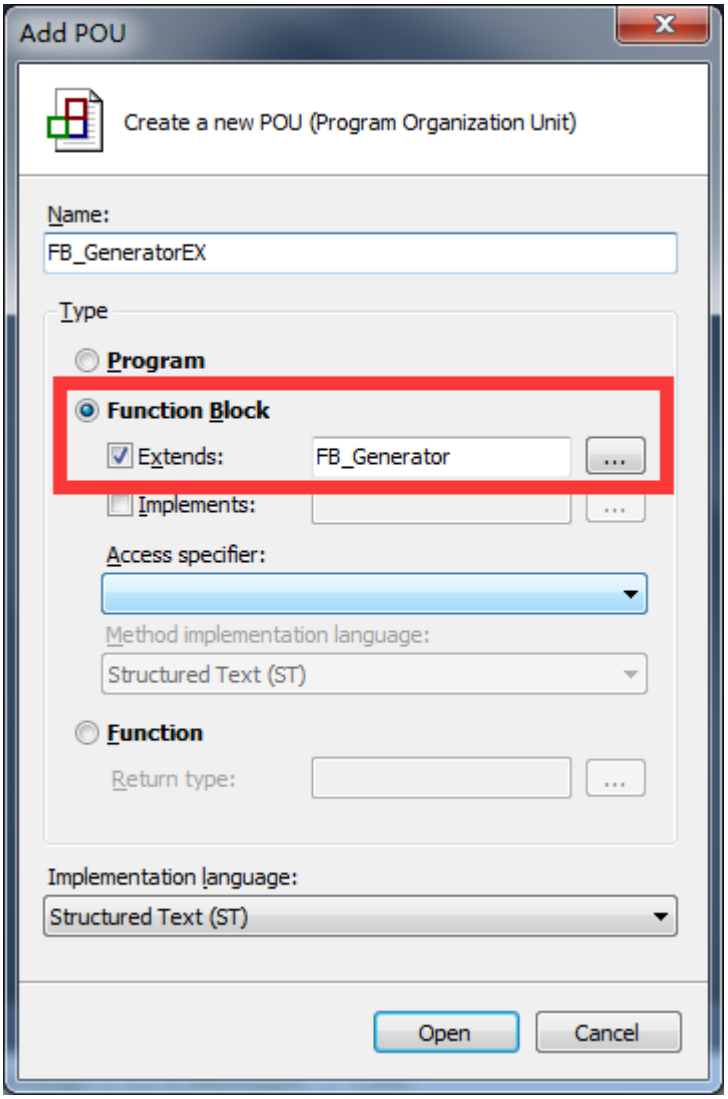




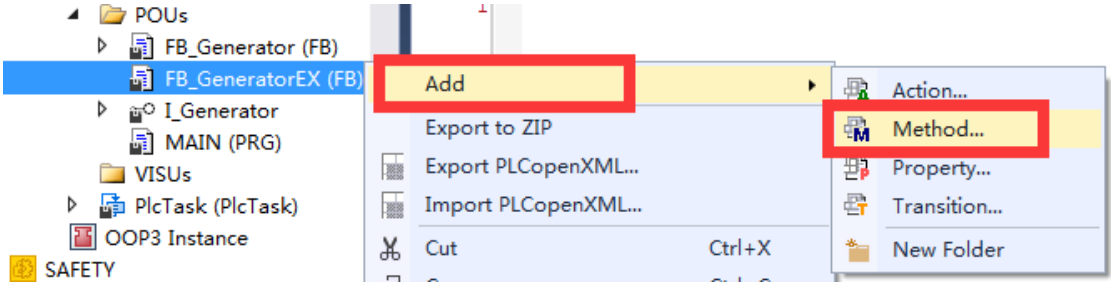
（16）选择建立好的 FB_Generator 这个类作为继承对象。



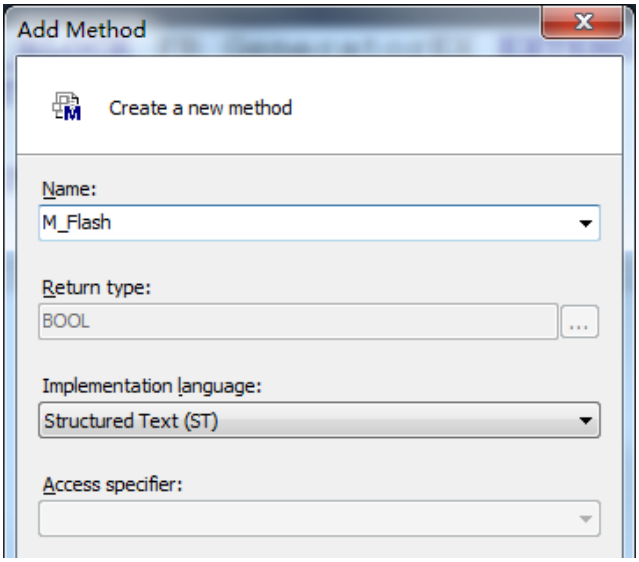
（17）选中后点击 OK 可以发现被扩展功能块名出现在 Extends 框中，点击 Open。



（18）功能块 FB_GeneratorEX 已经继承了 FB_Generator 所有变量，方法和属性，我们可以对于 FB_GeneratorEX 功能块进行添加新的方法和属性进行扩展，当然也可以对于所继承的原有的方法和属性进行重写，接下来就演示如何进行重写右键 FB_GeneratorEX 新建 Method。



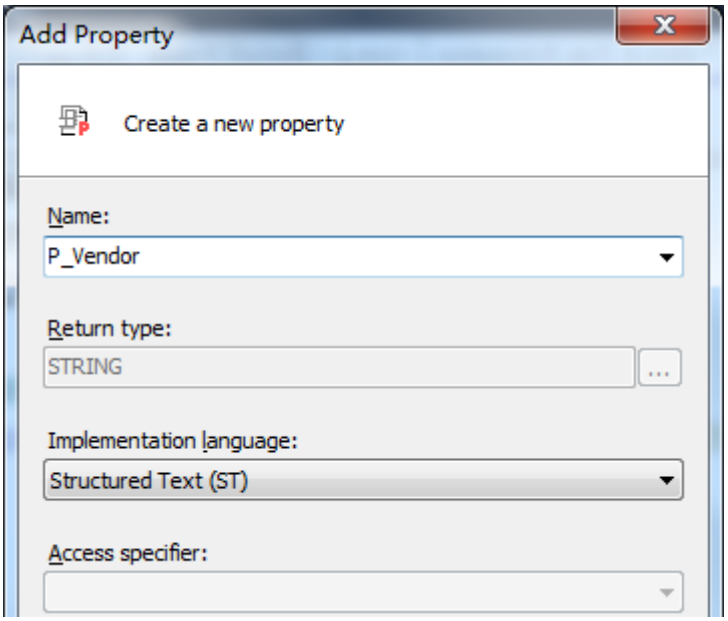
（19）修改此 Method 名也为 M_Flash，返回类型它会默认 BOOL 且为灰色不可修改。

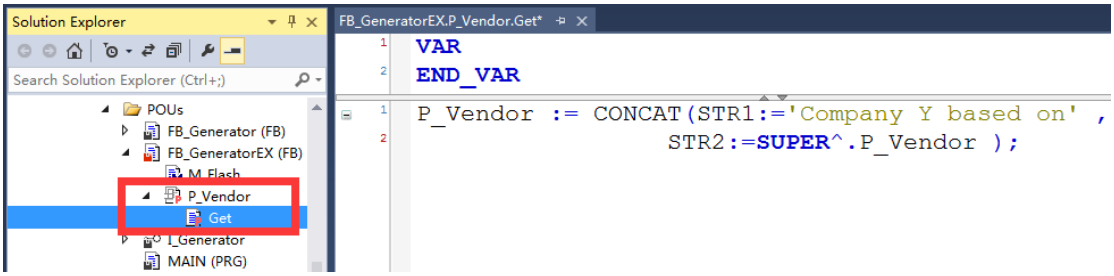


（20）随后重写编写 Flash 代码，实现的方法变为 1/3 为 true，2/3 为 false。

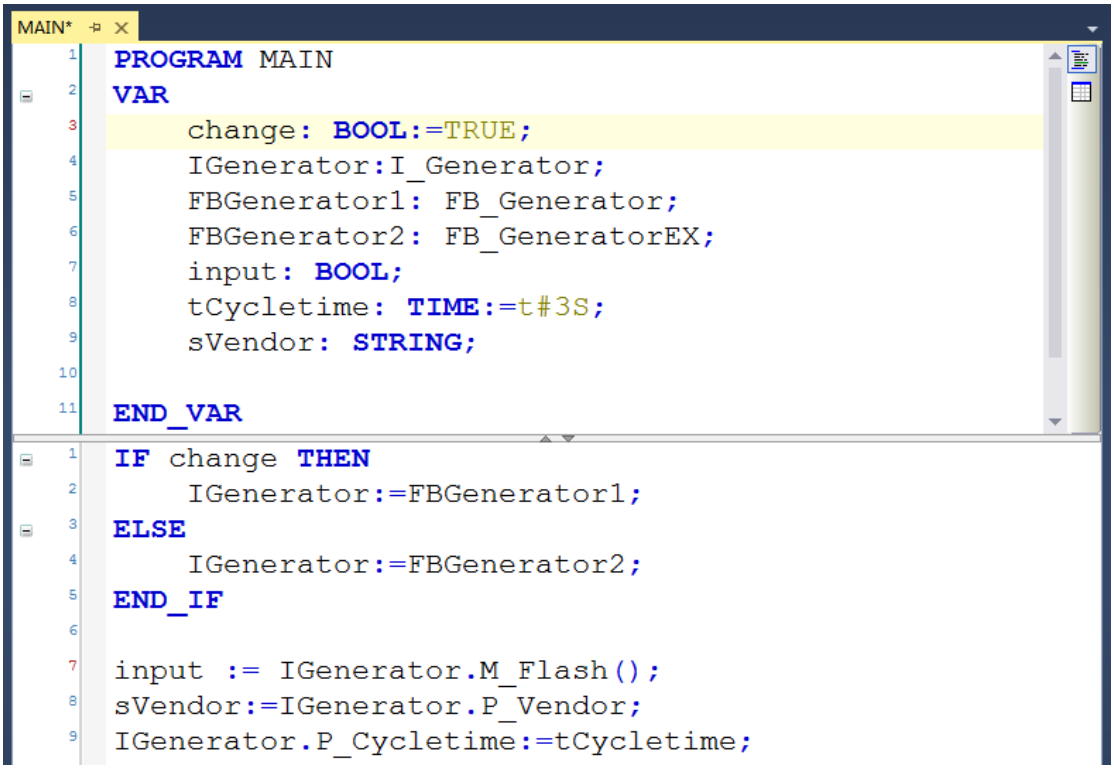
```
FB_GeneratorEX.M_Flash*
{warning 'add method implementation'}
METHOD M_Flash : BOOL

fbTon1(IN:=NOT fbTon1.Q ,
       PT:=tCycletime ,
       Q=> , ET=> );

// 33% ON 66% OFF
M_Flash := fbTon1.ET < fbTon1.PT / 3 ;
```

（21）再重写 vendor 这个属性，重写 vendor.get，其中使用到 super 可以直接访问到被扩展功能块 FB_Generator 中的 Vendor，并且使用 CONCAT 函数进行字符串合并。

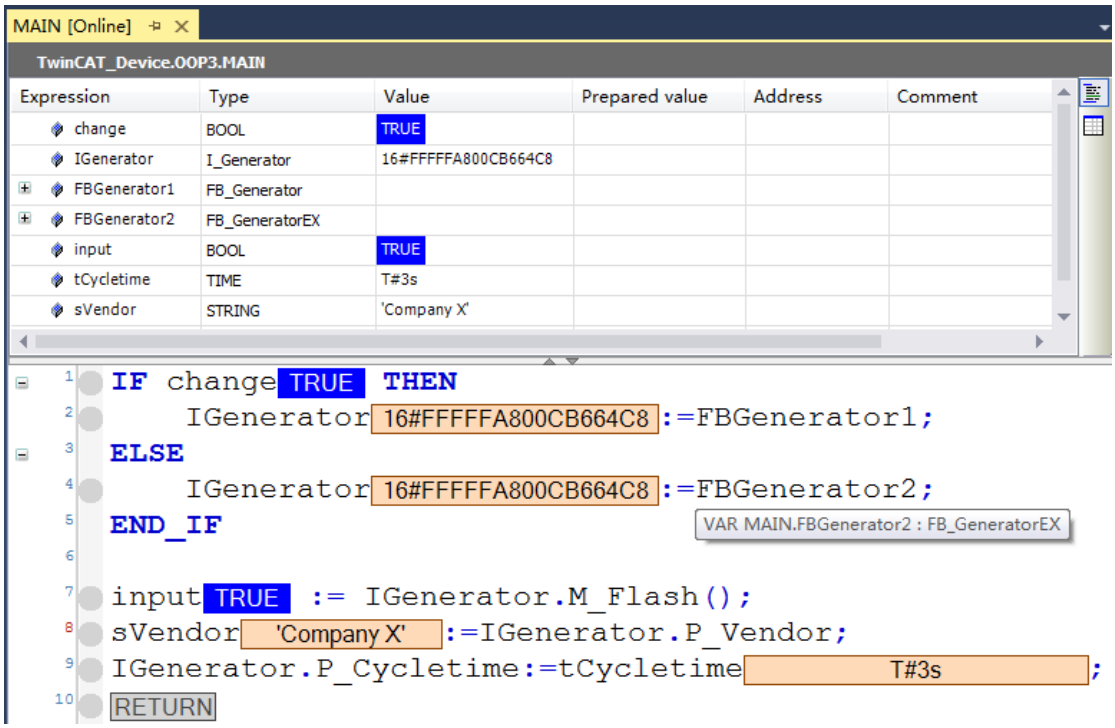




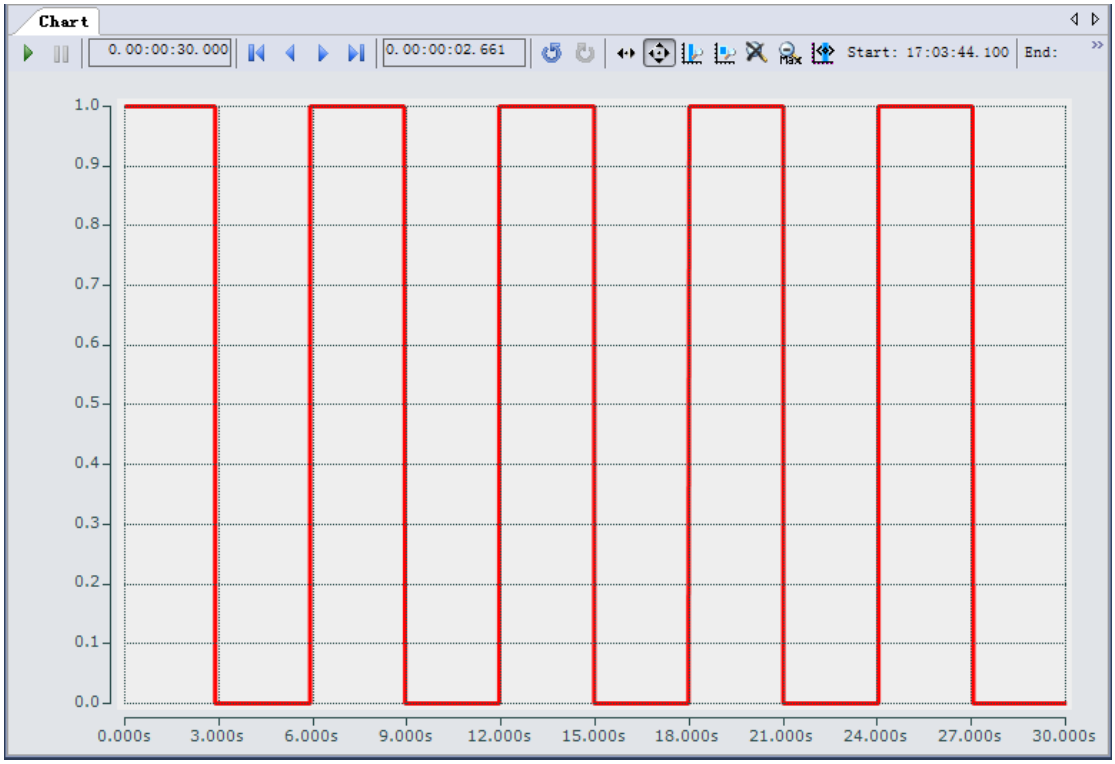
（22）在 MAIN 程序中对接口和功能块进行实例化，并且编写代码完成对两个功能块的调用和切换



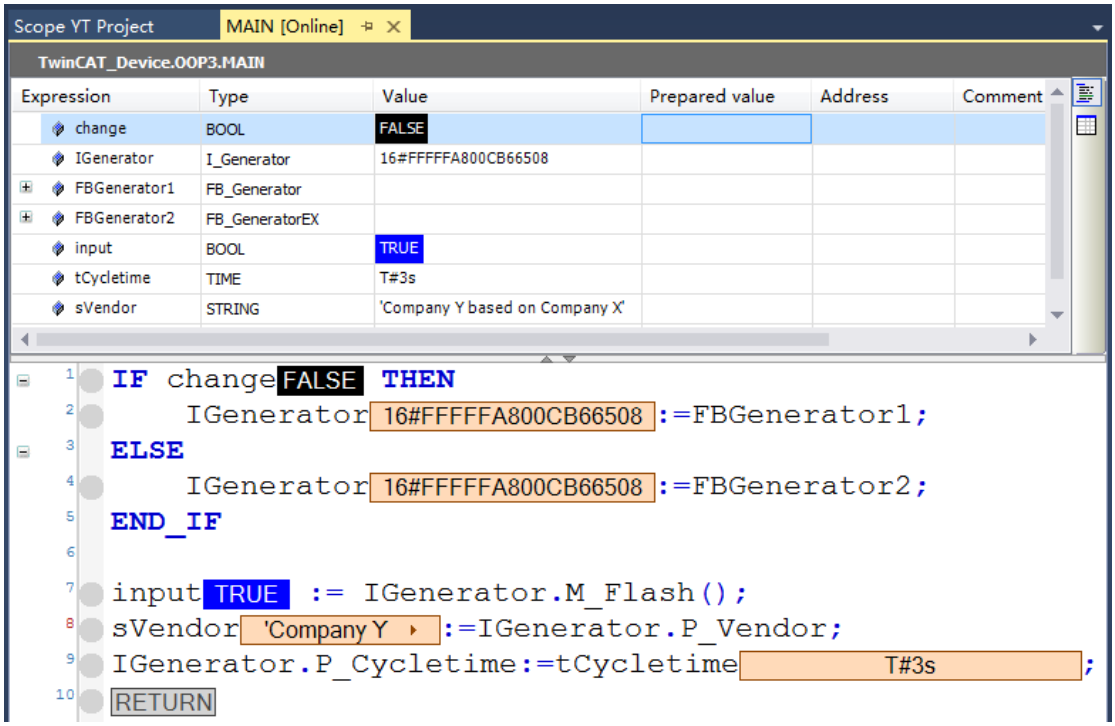
（23）Activate Configuration 之后，Login 并 Run。我们不难发现，在 change 变量为 TRUE 的状态下，采用占空比为 50%的波形输出方式，sVendor 读到的字符串为 Company X。



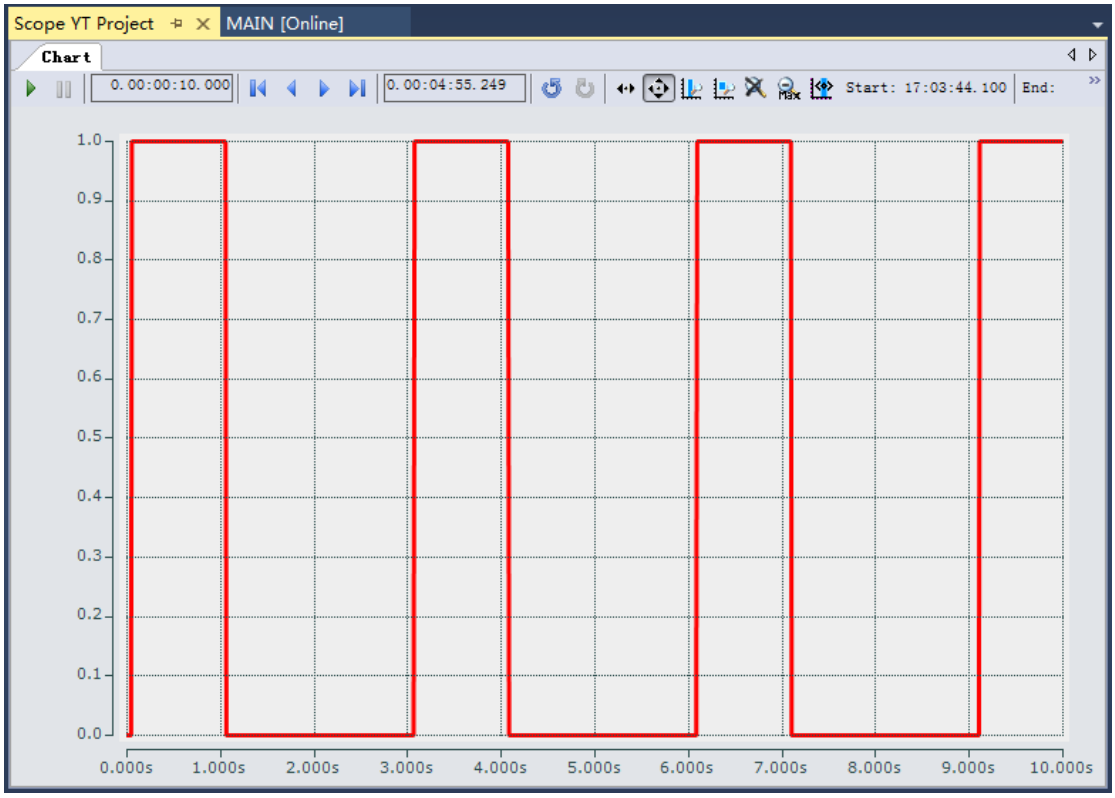
（24）并且通过 scope view 观察到 input 为 3 秒 true，3 秒 false。



（25）在 change 变量为 FALSE 的状态下，采用占空比为 33.3%的波形输出方式，sVendor 读到的字符串为 Company Y based on Company X。



（26）并且通过 scope view 观察到 input 为 1 秒 true，2 秒 false。



（27）单独说明一下 this 的用途。this 是指针关键词，可以指向当前功能块中所有成员（变量，方法，属性）。

例如在 Function Block 和 Method 中同时声明 bool 型变量 bFlashVar，那么单写 bFlashVar 表示的是 Method 中的变量，而使用 THIS^.bFlashVar 表示的是 Function Block 中的变量。

FB_Generator* | FB_Generator.Flash*

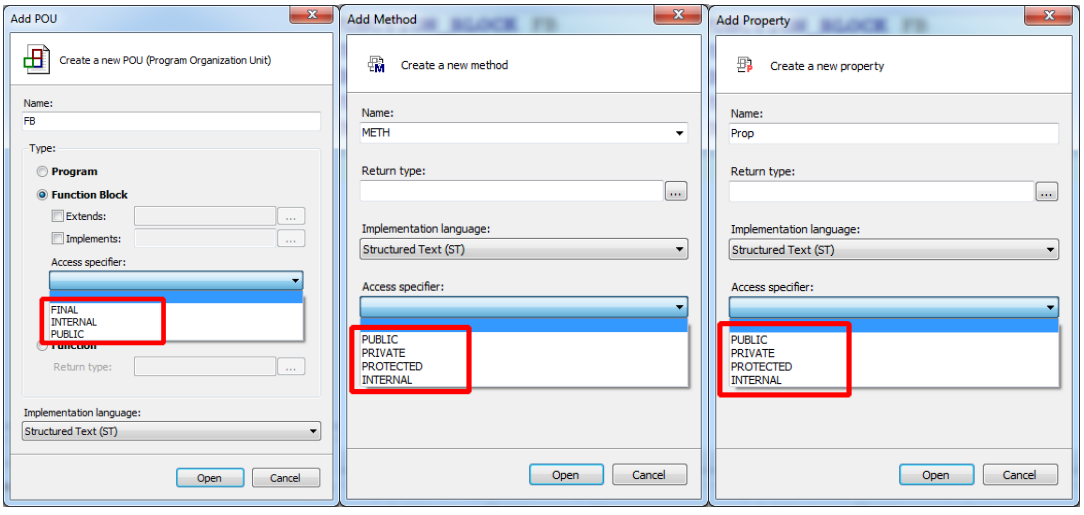
```
FUNCTION_BLOCK FB_Generator IMPLEMENTS I_Generator
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    fbTon1 :TON;
    _tCycletime : TIME:= T#1S;
    bFlashVar: BOOL;
END_VAR
```

FB_Generator* | FB_Generator.Flash*

```
{attribute 'object_name' := 'Flash'}
METHOD Flash : BOOL
VAR
    bFlashVar:BOOL;
END_VAR
```

```
fbTon1(IN:=NOT fbTon1.Q , PT:=_tCycletime , Q=> , ET=> );

IF fbTon1.Q THEN
    THIS^.bFlashVar := NOT THIS^.bFlashVar;
END_IF

Flash := THIS^.bFlashVar;
```

（28）针对于 FB，method，property 在添加的时候都有 acess specifier（访问说明符）的设置，下面就说明一下这些设置的含义。



出现在 FB 中：

Internal：只允许当前项目命名空间才可以调用，也就是说一旦封装成库，外部（MAIN）就无法调用此功能块

Final：此功能块无法被扩展

出现在 method 和 property 中：
Private：只限于功能块这个命名空间才可以被互相调用或者嵌套，并且此方法或者属性无法被继承到子类中
Protected：只限于功能块这个命名空间才可以被互相调用或者嵌套，此方法或者属性可以被继承到子类中，但外部（MAIN）无法调用此方法或者属性
Internal：只允许当前项目命名空间才可以调用，也就是说一旦封装成库，外部（MAIN）就无法调用此方法或属性
Final：此方法或者属性不允许重写

德国倍福自动化有限公司各地办事处联系方式：

宁波办事处

宁波市解放南路 9 号天元大厦2010室（315010）

电话：0574-87203335　　传真：0574-87203336

武汉办事处

武汉市武昌区中南路 7号中商广场写字楼 A座 1803 室

电话：027-87711992/3/5　传真：027-87711916

杭州办事处

浙江省杭州市江干区钱江路1366号华润大厦A座1007室

电话：0571-87652786-8008　　传真：0571-87652785

合肥办事处

安徽省合肥市怀宁路 288 号置地广场 D座 606 室

电话：0551-65543513-8001　　传真：0551-65543713

无锡办事处

无锡市滨湖区梁溪路 51 号万达广场 A 区写字楼 2010 室（214062）

电话：0510-85819306　　传真：0510-85809306

南京办事处

南京市中山南路 49 号商茂世纪广场 22 楼 A4、A5

电话：025-85862271/3　　传真：025-85862272

苏州办事处

苏州工业园区苏雅路 388 号新天翔广场 A 座 1207 室

电话：0512-62852207　　传真：0512-62852156

青岛办事处

青岛市郑州路 43 号橡胶谷 A 栋 349 室

电话：0532-55663857/8-80　　传真：0532-55663857/8-85

沈阳办事处

沈阳市沈河区惠工街 10 号卓越大厦 1803 号（110013）

电话：024-22788896/98-0　　传真：024-22789845

西安办事处

西安市南二环西段 88 号老三届世纪星大厦 16C

电话：029-88499908　　传真：029-81773323

天津办事处

天津市河东区六纬路 99 号津东大厦 B303 室

电话：暂无

郑州办事处

郑州市金水区经三路 68 号平安保险大厦 1311 室

电话：0371-61732582　　传真：0371-61732580

深圳办事处

深圳市福田区深南大道 6023 号创建大厦 2410 室（518040）

电话：0755-23603232　　传真：0755-23603233

长沙办事处

长沙市芙蓉区五一大道 766 号中天广场写字楼 9043A（410005）

电话：0731-89608950　　传真：0731-89608951

重庆办事处

重庆市北部新区黄山大道中段三号水星科技大厦北翼 2-13

电话：023-67398175　　传真：023-67398175-608

昆明办事处

昆明市北京路 155 号附 1 号红塔大厦 1202 室

电话：0871-63550636　　传真：0871-63211889

上海（ 中国区总部）
德国倍福自动化有限公司
上海市闸北区江场三路 163 号（市北工业园区）5 楼
电话：021-66312666　　传真：021-66315696　　邮编：200436

北京分公司
德国倍福自动化有限公司
北京市西城区西直门外大街 1 号西环广场 T3 写字楼 1801 - 1803 室
电话：010-58301236/7　　传真：010-58301286　　邮编：100044

广州分公司
德国倍福自动化有限公司
广州市天河区珠江新城珠江东路16号高德置地G2603室
电话：020-38010300/1/2　传真：020-38010303　　邮编：510623

成都分公司
德国倍福自动化有限公司
成都市锦江区东御街18号 百扬大厦2305 房
电话：028-86202581　　传真：028-86202582　　邮编：610016

扫一扫，关注
倍福官方微信

倍福中文官网：
https://www.beckhoff.com.cn/

倍福虚拟学院：
https://tr.beckhoff.com.cn/

招贤纳士：job@beckhoff.com.cn
技术支持：support@beckhoff.com.cn
产品维修：service@beckhoff.com.cn
方案咨询：sales@beckhoff.com.cn