BECKHOFF


# TwinCAT 3 程序框架教材

Version 1.0

毕孚自动化设备贸易（上海）有限公司

2018 年 6

# 前言

TwinCAT3 软件作为倍福 PC 控制技术的核心，已经被广泛应用于各行各业。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。

本书适用于使用倍福TwinCAT3软件或者从事电气开发的技术工程师。本书作为一本中级TwinCAT3软件学习教材，深入浅出地介绍了适应快速编写的程序构架，帮助电气工程师更加规范，更加快速的编写控制程序。

本书组成部分：

第一章介绍了控制理论，分别从控制模式，协调机制，重构能力和面向对象及面向过程的有机结合等方面阐述了程序架构的必要性；

第二章介绍了快速进行I/O链接的方法；

第三章、第四章分别介绍了运动控制和执行器的程序架构，帮助工程师快速编写控制程序；

第五章、第六章介绍了以码垛机课程设计为背景，通过面向过程的方法来实现逻辑控制和动画功能。

第七章到第十章分别介绍了辅助程序的用户等级功能，配方功能，事件功能和系统诊断功能，保证了控制系统的完整性、高效性和可靠性。

本书的内容会不定期更新，最新教材读者可以通过FTP免费获取

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

最后特别要感谢原倍福工程师陈颂文，本文中涉及到的以码垛机课程设计为背景的程序架构是陈工结合自己10多年的工作经验而编写的；本文的内容是通过陈工的内部分享改编得来。

由于编者水平有限，误漏欠妥之处在所难免，竭诚欢迎各位同行和读者批评指正。

联系邮件：yy.xu@beckhoff.com.cn

徐樱樱

2018年6月1日

# 目录

# 第一章:引言、理论

自从20世纪90年代以后，我国制造业进入了一个高速发展的阶段，在国民经济中所发挥的作用也更加重要。然而，随着信息技术发展的突飞猛进，经济全球化一体化趋势的日趋缩短，产品的种类日趋个性化和多样化，制造业所面临的市场竞争变得更加激烈，且充满了日益频繁且无法预知的市场变化。这就使得制造系统的生产组织模式必须由单纯的面向产品生产转变为面向市场需求和客户服务，不仅要降低生产制造的各种成本、提高生产效率、保证产品质量，还需要能够针对外部动态多变的市场行情作出及时有效的快速响应。这就对设备开发的周期、效率等提出了更高的要求。为适应这种需求，保证控制系统搭建的快速性、高效性、可靠性，本文档介绍了一种适应快速编写的程序构架。

任何架构都有其优越性、局限性及适用性。本构架比较适合电子行业的设备开发。本程序框架适用于需要快速完成程序开发工作的场合。对其它行业也有借鉴意义。

## 控制模式：

日益激烈的全球化市场竞争迫使企业必须缩短产品的交货期、提高质量、控制成本和增强制造系统的柔性。近几十年中，制造控制系统的结构主要经历了集中式控制、递阶式控制、分布式控制和混合式控制等四种体系结构形式的变迁，如图所示。

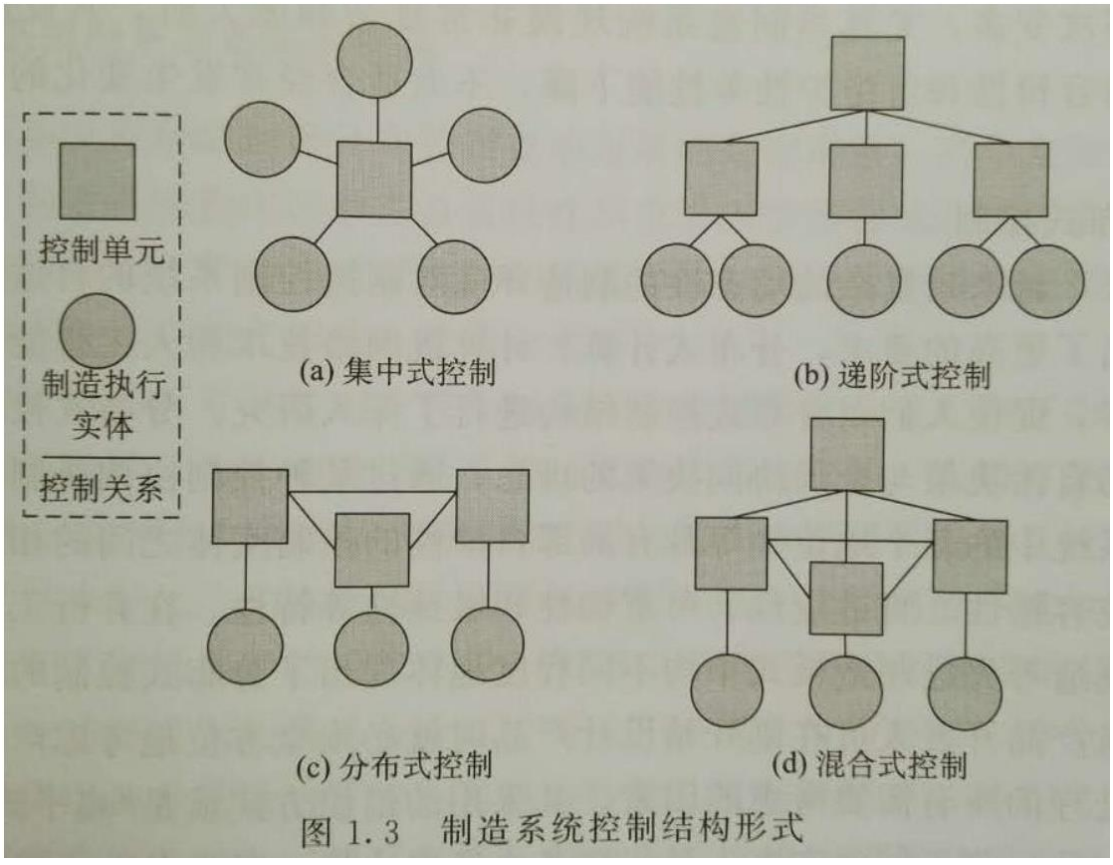


图 1.3　制造系统控制结构形式

从控制结构的发展趋势可以看出，随着系统规模的变大和复杂程度的增

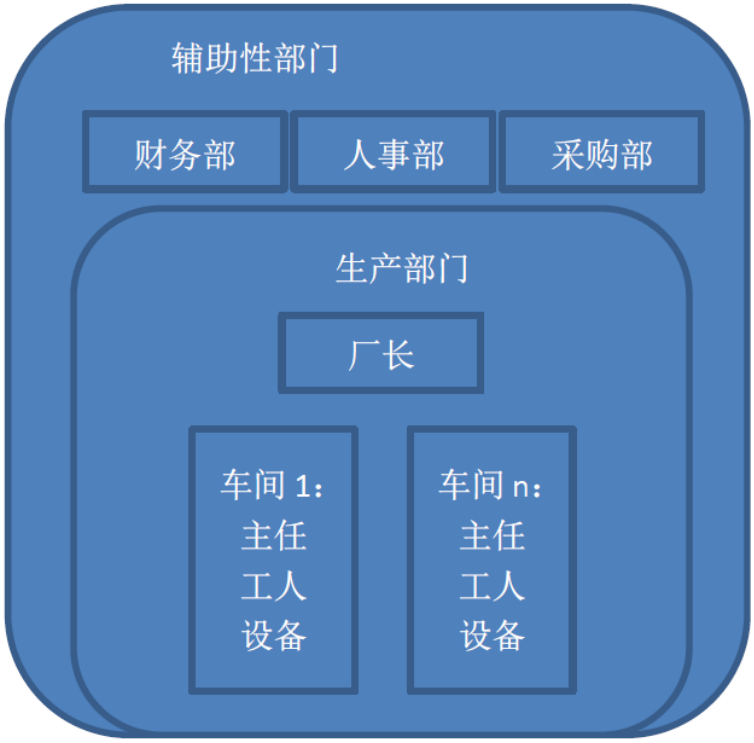
强，相互之间的主从关系趋弱，更多的倾向于协同工作。另外，控制系统的优化目标也从最终的全局优化趋向于局部优化，试图通过对控制单元实体的自治性、智能性的加强及合理的分布与协调各个局部控制单元的决策职责，来实现开放、灵活和具有自组织性的制造系统控制体系。

为了处理控制系统的复杂和适应性，人们将最初的集中控制结构逐渐往“扁平化”的分布控制结构发展，更多地强调了控制系统功能的模块化、智能化与分布化。在稳定的运行环境或者优先变化的制造环境中，传统的递进控制系统可以取得良好的控制性能，但当系统遇到各种不确定的偶发事件时，其刚性的系统结构将无法满足系统实时响应、快速重构的要求。而分布式控制由于控制实体往往具有较强的智能性和自治性，在系统遇到各种偶发性事件时会表现出快速的响应性和良好的可靠性，但由于缺乏集中式优化的全局统筹决策，其全局优化目标和系统局部行为之间难于协调控制，因此，其系统的稳定性和可预测性较差。而混合型控制系统结合分布式控制与递阶式控制的优点，具有较大的发展空间，可以根据结构中分布与递阶程度的不同，组成多种结构形式，大大增加了制造系统控制系统体系的灵活性。

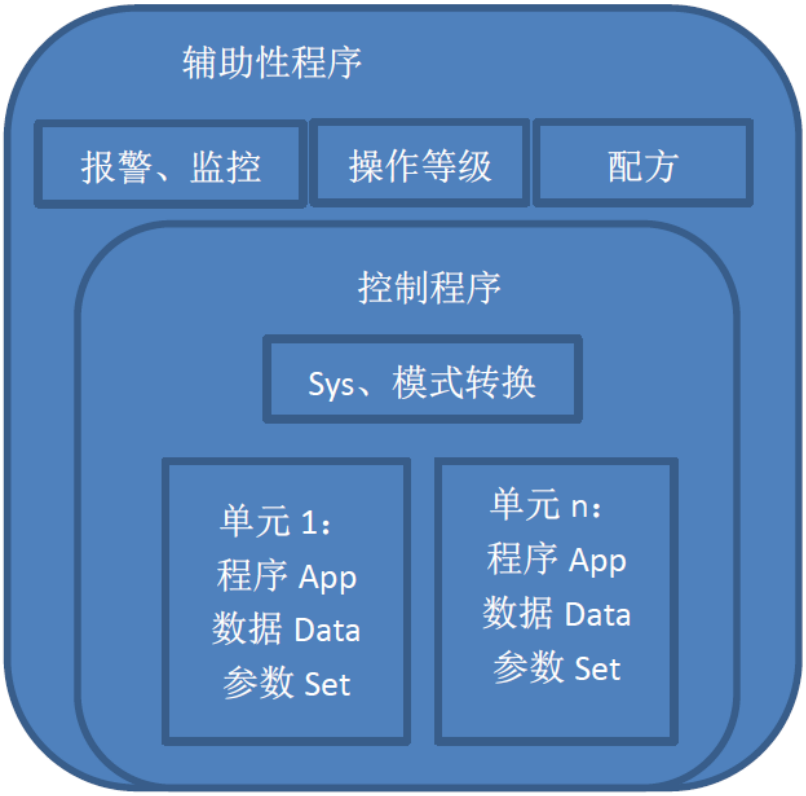
本文档采用了混合式控制的结构。在本结构中，外围的辅助性功能，如配方、用户等级、监控等，独立于控制程序。控制功能中，“90_Sys”和“04_模式转换”做为总控功能，对各个单元(Unit)发布任务：开机、停止、暂停、回原点等指令。而各个单元(Unit)之间是相互独立，互相平等的关系，各个单元具有其独立的数据结构和设置参数。

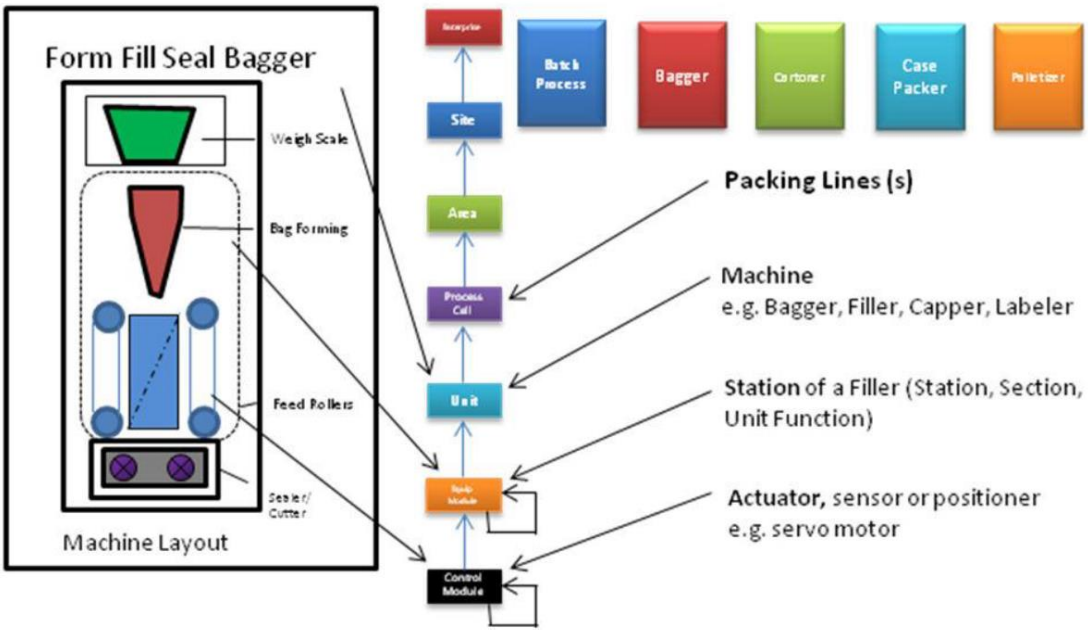

本文档的程序结构，类似于一个企业的组织运行构架。

企业结构：


辅助性部门
财务部
人事部
采购部
生产部门
厂长
车间 1：
主任
工人
设备
车间 n：
主任
工人
设备

程序结构：


辅助性程序
报警、监控
操作等级
配方
控制程序
Sys、模式转换
单元 1：
程序 App
数据 Data
参数 Set
单元 n：
程序 App
数据 Data
参数 Set

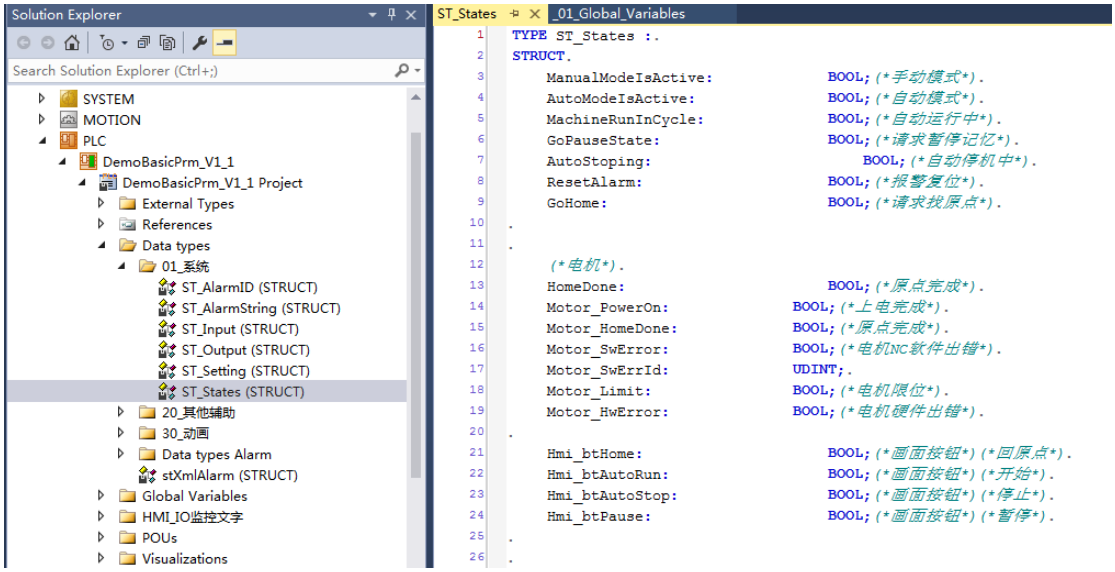



## 协调机制：

从广义上来说，所谓协调就是多个事物对相互之间依赖关系的管理，对于群体活动的有序进行具有主要作用。

由于各个子问题均是由全局问题分解而来的，但对独立又相互关联，这也导致了各个单元的子问题求解活动之间必然存在着相互依赖的关系，从而有机地构成了全局问题的求解过程。因此，对于大规模问题的求解，协调活动是不可或缺的。

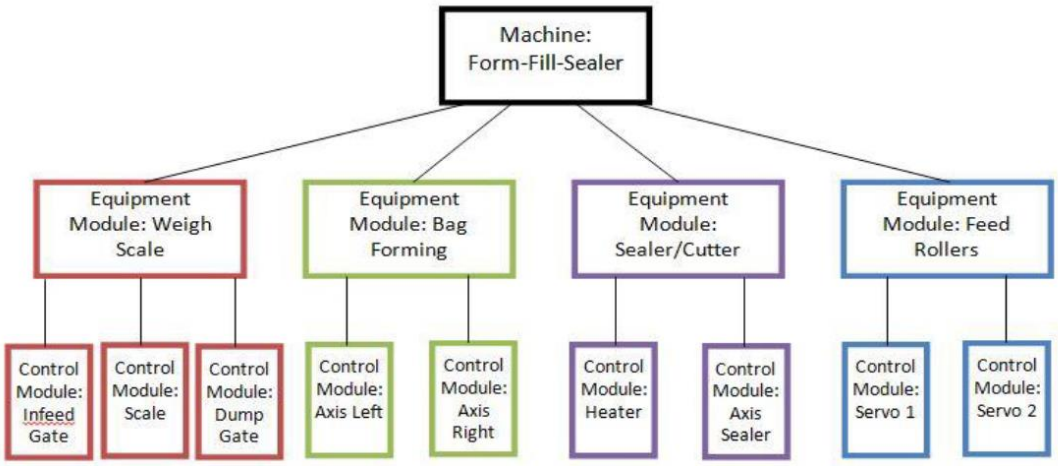
本文档采用了显式协调机制。在显式协调机制的工作过程中，控制实体之间必须要进行交互，因此，其首先必须具有通信模块。




如上图中，每个单元都具有”Data”数据结构。Data 里面存储了当前单元的状态信息。而其它单元可以通过 Data 去读取该单元的状态，也可以直接对 Data 进行写操作，修改状态。

本文对“**FunctionBlock**”功能块**(FB)**的使用方法，和传统教材上介绍的方法

不太一样，构思巧妙，用法独特：利用了 FB 的特性，在某些场合把 FB 当作了数据结构来使用。例如伺服轴“Axis_PTP_CoE”采用了类似的思维。读者就可以简单地把它们当作数据结构来简单看待。



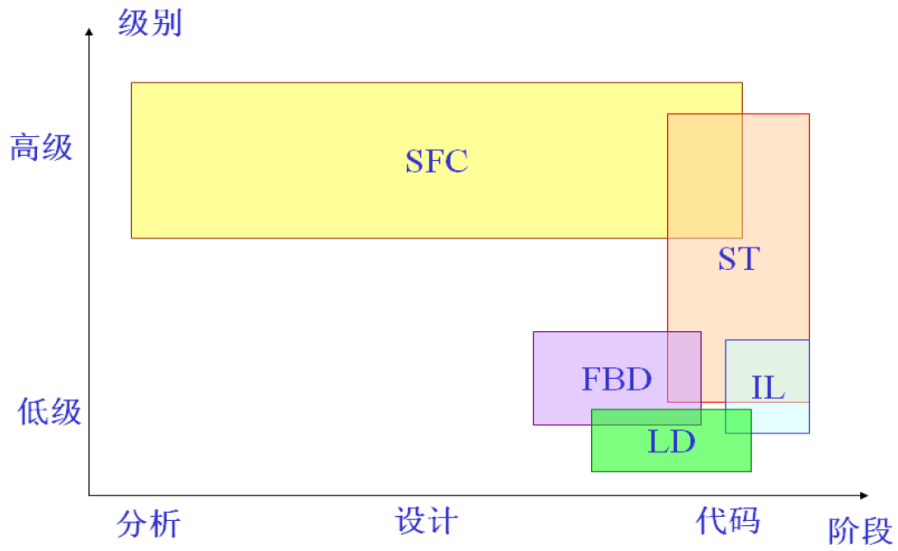

本程序的协调机理如下：
各个单元之间：通过“Data”数据进行交互。

## 重够能力：

制造系统做为一个复杂的大系统，它的开发与维护日趋变得异常困难，而重构不但可以提高任务的完成质量，而且可以节约系统的开发成本。所以，重构能力是衡量程序构架的重要指标。

本构架充分考虑了重够的便利性。基于模块化的思想，可以对模块单元程序进行复用。每个单元模块都有其独立的程序区、数据区和设置区。在复用该单元时，直接复制整个单元模块，就能把该单元相关的逻辑、接口、数据、参数等等一同复制过来，简单便捷。外围辅助程序的编写，也已经可以即拿即用。对伺服轴的处理封装，已经在实际机台运行测试完成。

## 选择 ST 语言：




其中可见，只有 ST 语言适合低级到高级的运用跨度。所以，本程序框架使用了 ST 语言做为编写的语言。

## 面向对象/面向过程的有机结合

面向过程很好理解，指的是程序员接到需求，会把它拆成一个一个的命令，然后串起来交给计算机去执行。举个例子，客户说要把大象装进冰箱里。程序员列了几个步骤：

把冰箱门儿打开。

把大象装进去。

把冰箱门儿关上。

上面每一个步骤，程序员都会用一个「函数」来实现。「函数」是一些代码的集合体，每个函数可以实现一个功能。比如我要定义一个打开冰箱门的函数：

所有函数定义好了之后，依次调用就可以了：

```
openTheDoor();
pushElephant();
closeTheDoor();
```

需求完成，顺利交工。但是你以为这样就结束了？Naive。客户说才刚刚开始呢。

「我要把大象装微波炉里」

「我要把狮子也装冰箱里」

「我要把大象装冰箱，但是门别关，敞着就行」

如果还是用面向过程的方法来应付，每次需求的变更，程序员就要把整个系统通读一遍，找出可用的函数（如果没有就再定义一个），最后依次调用它们。最后系统越来越杂乱无章难以管理，程序员不堪重负。

面向对象从另一个角度来解决这个问题。它抛弃了函数，把「对象」作为程序的基本单元。那么对象到底是个什么东西呢？对象就是对事物的一种抽象描述。人们发现，现实世界中的事物，都可以用「数据」和「能力」来描述。比如我要描述一个人，「数据」就是他的年龄、性别、身高体重，「能力」就

是他能做什么工作，承担什么样的责任。

例如你可以让「狗」这个对象「吃狗粮」，就可以把「吃狗粮」的命令发给「狗」让其执行，然后我们就实现了「狗吃狗粮」的需求。

现在对象有了，如何进行面向对象的编程呢？很简单，依次向不同的对象发送命令就可以了。回到上面的例子，用面向对象来实现，我们会先定义一个「冰箱」对象，它的「数据」就是当前的冷冻温度，或者该冰箱已经有了多少头大象，「能力」就是开门、关门。还有一个「大象」对象，它的「数据」可以是大象的智商、体积，「能力」就是「自己跑到冰箱里去」。然后我们依次：

向冰箱下达「开门」的命令。

向大象下达「进冰箱」的命令。

向冰箱下达「关门」的命令。

总结为：

“面向过程”是做一件事，“面向对象”是造一堆东西。

# 第二章：IO 链接

通过本章节的学习，学员将了解：

- ✓ 利用 EXECEL 快速建立多个 I/O 变量
- ✓ 快速进行变量链接
- ✓ 通过 HMI 画面实现对所有 I/O 的监控

## I/O 链接步骤详解：

1） 新建一个 EXECEL 表格，把本次项目中用到的变量写入 EXECEL 表格中，定义如下：

输入点：

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ST_Input. | bisnAirPressOK | (* | I0.0 | 真空源检测 | *) | := | EL1008[1 ,1] | ; |
| ST_Input. | biSnBackDoorIsClose | (* | I0.1 | 后门关 | *) | := | EL1008[1 ,2] | ; |
| ST_Input. | biSwEmgStop | (* | I0.2 | 急停按钮 | *) | := | EL1008[1 ,3] | ; |
| ST_Input. | biBtstart | (* | I0.3 | 开始按钮 | *) | := | EL1008[1 ,4] | ; |
| ST_Input. | biBtstop | (* | I0.4 | 停止按钮 | *) | := | EL1008[1 ,5] | ; |
| ST_Input. | biBtHold | (* | I0.5 | 暂停按钮 | *) | := | EL1008[1 ,6] | ; |
| ST_Input. | biSnVacuuSor | (* | I0.6 | 取料真空检测 | *) | := | EL1008[1 ,7] | ; |
| ST_Input. | biSnConvBoxIn | (* | I0.7 | 检测到进料口有箱子进来 | *) | := | EL1008[1 ,8] | ; |
| ST_Input. | biSnConvBoxIn | (* | I0.8 | 检测到出料口有箱子出来 | *) | := | El1008[2 ,1] | ; |
| ST_Input. | biSnBoxCapper | (* | I0.9 | 封箱机检测到有箱子进来 | *) | := | EL1008[2 ,2] | ; |

输出点：

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| EL2008[1 ,1] | := | ST_Output. | bqBoxCapperDw | (* | Q0.0 | 压箱气缸 | *) | ; |
| EL2008[1 ,2] | := | ST_Output. | bqBoxCapperCut | (* | Q0.1 | 剪断气缸 | *) | ; |
| EL2008[1 ,3] | := | ST_Output. | bqClamperSuck | (* | Q0.2 | 机械手吸真空 | *) | ; |
| EL2008[1 ,4] | := | ST_Output. | bqClamperZBrake | (* | Q0.3 | Z伺服刹车 | *) | ; |
| EL2008[1 ,5] | := | ST_Output. | bqLightGreen | (* | Q0.4 | 绿灯 | *) | ; |
| EL2008[1 ,6] | := | ST_Output. | bqLightYellow | (* | Q0.5 | 黄灯 | *) | ; |
| EL2008[1 ,7] | := | ST_Output. | bqLightRed | (* | Q0.6 | 红灯 | *) | ; |
| EL2008[1 ,8] | := | ST_Output. | bqBuzzer | (* | Q0.7 | 蜂鸣器 | *) | ; |

在建立 I/O 点表时，建议充分使用 EXECEL 表格的特点，通过复制、粘贴、下拉，可以快速建立表格。

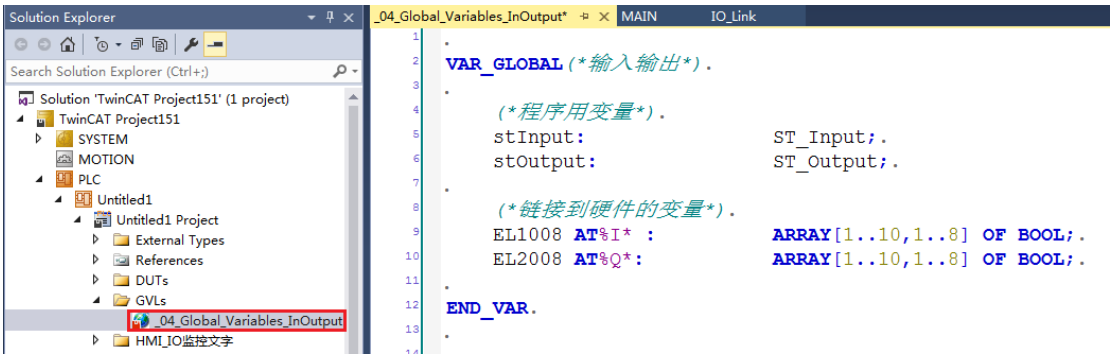
2） 定义输入输出变量“ST_Input”，“ST_Output”

把程序中的输入变量和输出变量分别整合在ST_Input 和ST_Output 的结构体中。其中，在表格里面复制相关的结构定义，粘贴到程序里面，就完成了输入点/输出点的定义，非常快捷方便。

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ST_Input. | bisnAirPressOK | (* | I0.0 | 真空源检测 | *) | := | EL1008[1 ,1] | ; |
| ST_Input. | biSnBackDoorIsClose | (* | I0.1 | 后门关 | *) | := | EL1008[1 ,2] | ; |
| ST_Input. | biSwEmgStop | (* | I0.2 | 急停按钮 | *) | := | EL1008[1 ,3] | ; |
| ST_Input. | biBtstart | (* | I0.3 | 开始按钮 | *) | := | EL1008[1 ,4] | ; |
| ST_Input. | biBtstop | (* | I0.4 | 停止按钮 | *) | := | EL1008[1 ,5] | ; |
| ST_Input. | biBtHold | (* | I0.5 | 暂停按钮 | *) | := | EL1008[1 ,6] | ; |
| ST_Input. | biSnVacuuSor | (* | I0.6 | 取料真空检测 | *) | := | EL1008[1 ,7] | ; |
| ST_Input. | biSnConvBoxIn | (* | I0.7 | 检测到进料口有箱子进来 | *) | := | EL1008[1 ,8] | ; |
| ST_Input. | biSnConvBoxIn | (* | I0.8 | 检测到出料口有箱子出来 | *) | := | El1008[2 ,1] | ; |
| ST_Input. | biSnBoxCapper | (* | I0.9 | 封箱机检测到有箱子进来 | *) | := | EL1008[2 ,2] | ; |

```
TYPE ST_Input :.
STRUCT.
    biSnAirPressOk(*压缩气源正常*),
    biSnBackDoorIsClose(*后门关*),.
    biSwEmgStop(*急停按钮*),.
    biBtStart(*开始按钮*),.
    biBtStop(*停止按钮*),.
    biBtHold(*暂停按钮*),.
    biSnVacuuSor(*取料真空检测*) ,.
    biSnConvBoxIn(*检测到进料口有箱子进来了*),.
    biSnConvBoxOut(*检测到出料口有箱子需要出来*),
    biSnBoxCapper(*封箱机检测到有箱子进来*).
:BOOL;.
END_STRUCT.
END_TYPE.
```

3） 定义全局变量，并定义为输入、输出类型

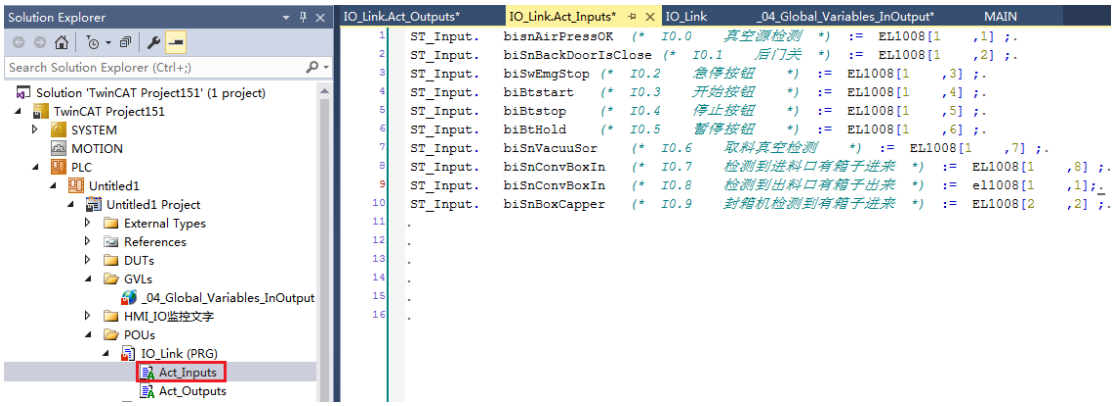



其中 1..10 是说有 10 个模块，我们可以定义多一点没有关系，留多点备用也是很好的。所以 10 个模块基本不用改。建议根据硬件类型（EL1004，EL1008，EL1809 等）来确认变量的名字和二维数组的第二维元素的大小。

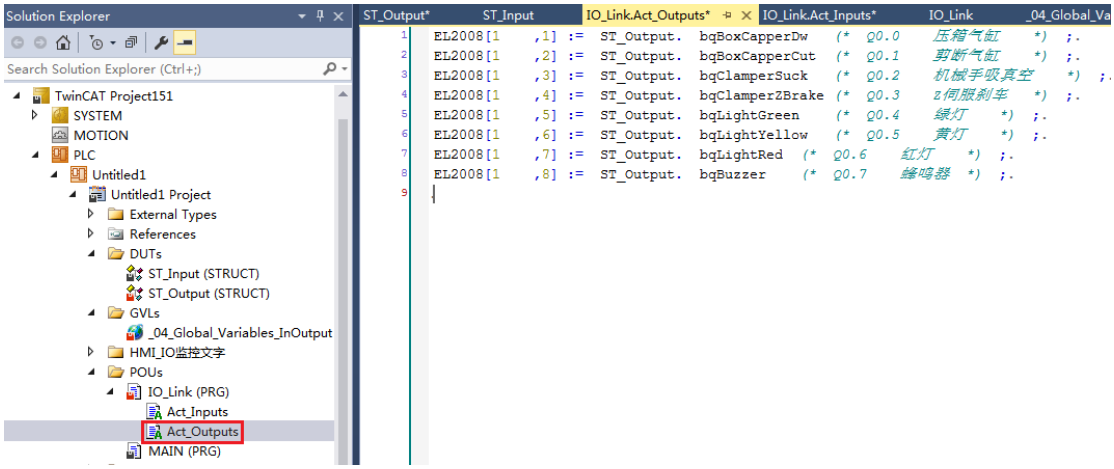
4） 建立相关 POU，编写输入输出点的程序

新建 IO_LINK 的 POU，在相应的 POU 下面添加两个 Action，分别是 Act_Inputs 和 Act_Outputs；在 Act_Inputs 里面写入相关的程序。为了编程的快速性，我们利用 EXCEL 表格的特点，把 EXECEL 表格中的输入和输出分别复制到 Act_Inputs 和 Act_Outputs。

Act_Inputs：

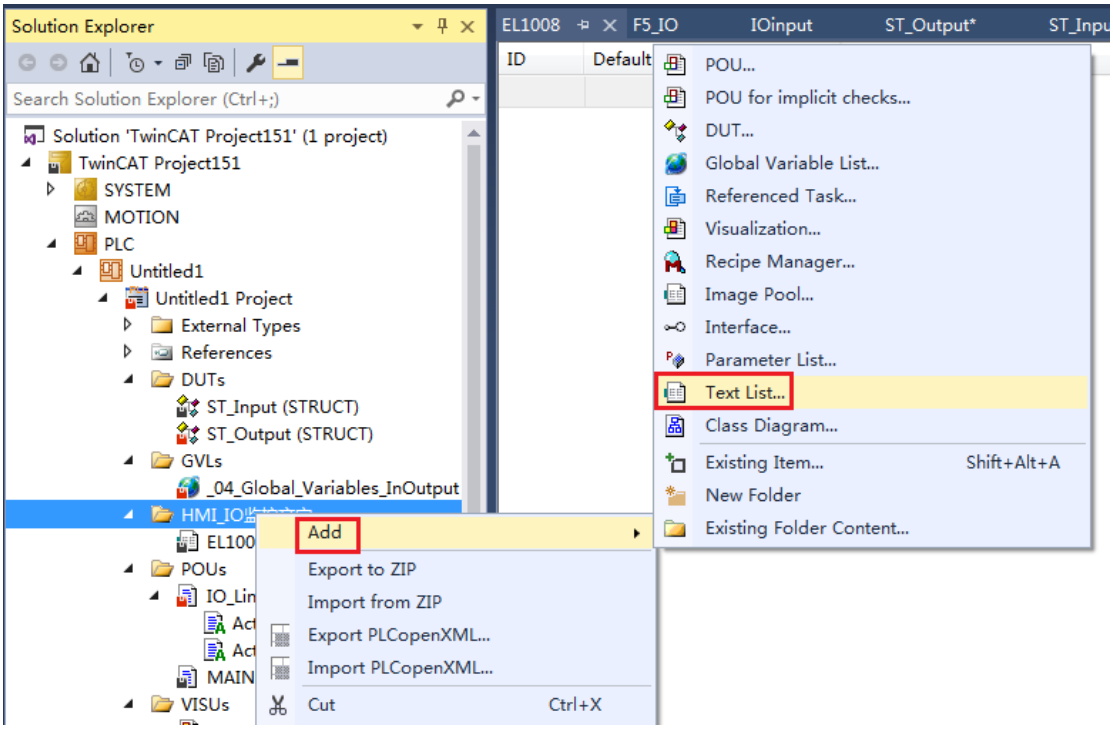


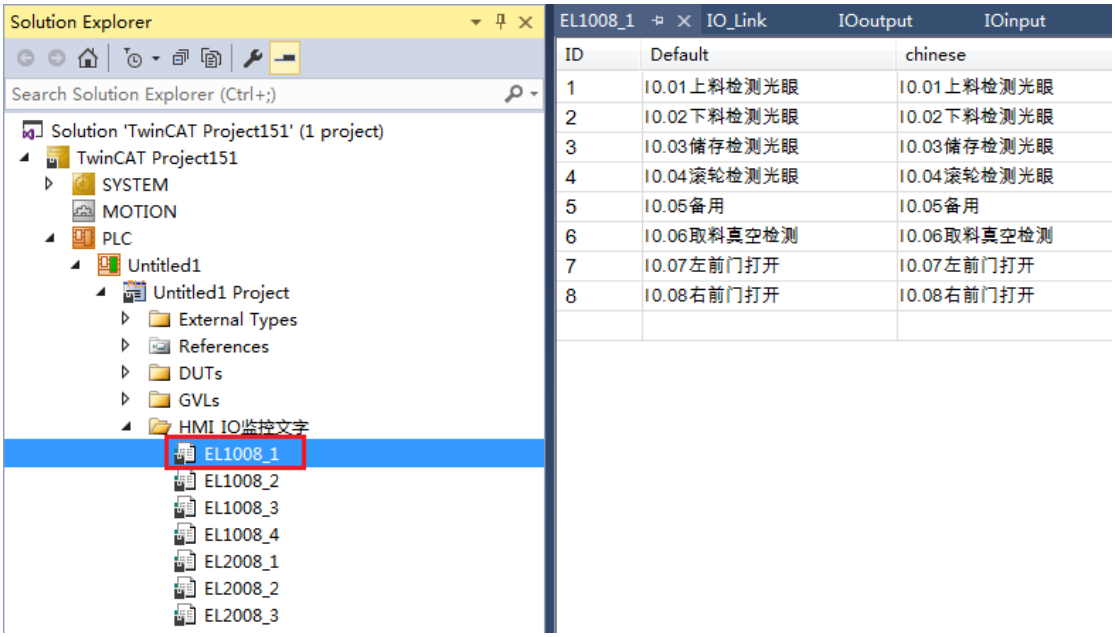

Act_Outputs:




5） 新建 Text list，让可视化界面上可以显示 IO 监控的文字

第一步：新建 text list，总共设置 3 个，分别取名 EL1008_1，EL1008_2 和 EL2008_3。



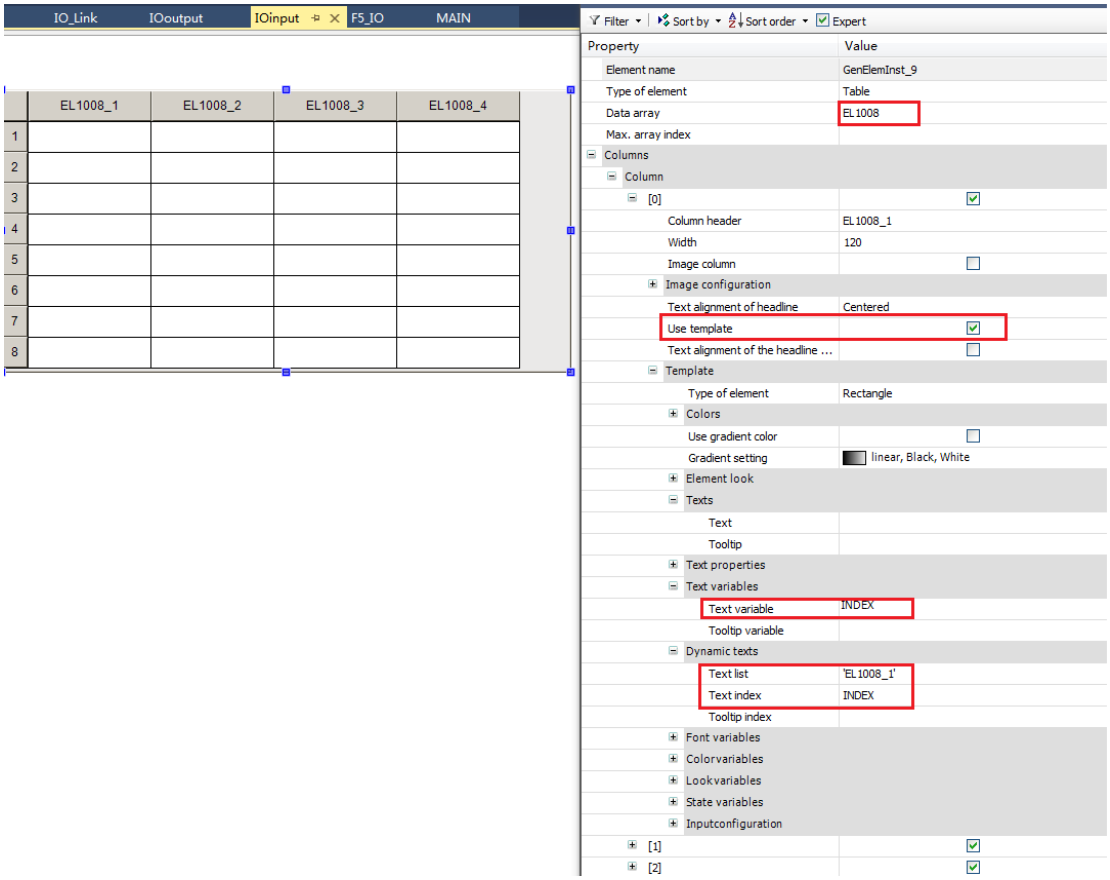

第二步：制作 HMI 画面

本次课程设计中 I/O 链接总共有 3 张 HMI 画面，分别为 F5_IO,Input 和 Output。

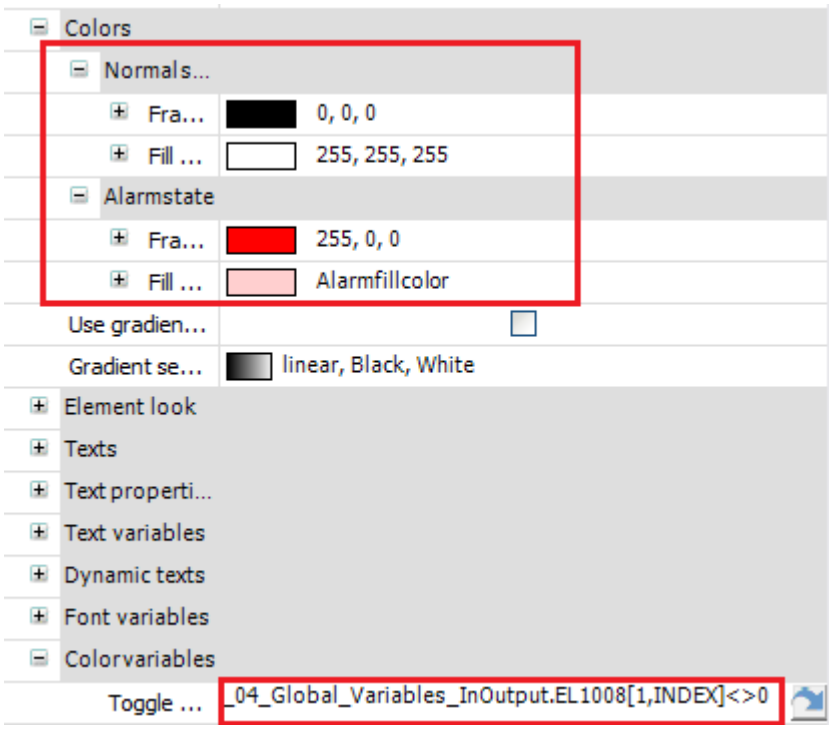
**Input 画面：**

Input 画面需要在表格中显示 EL1008 模块的每个输入点的含义，以及通过 table 控件中单元颜色的变化来确认 EL1008 模块是否接受到输入信号。

Input 画面就包括一个 table，具体设置如下图，下图中的设置是让表格显示 EL1008 模块每个输入点的含义



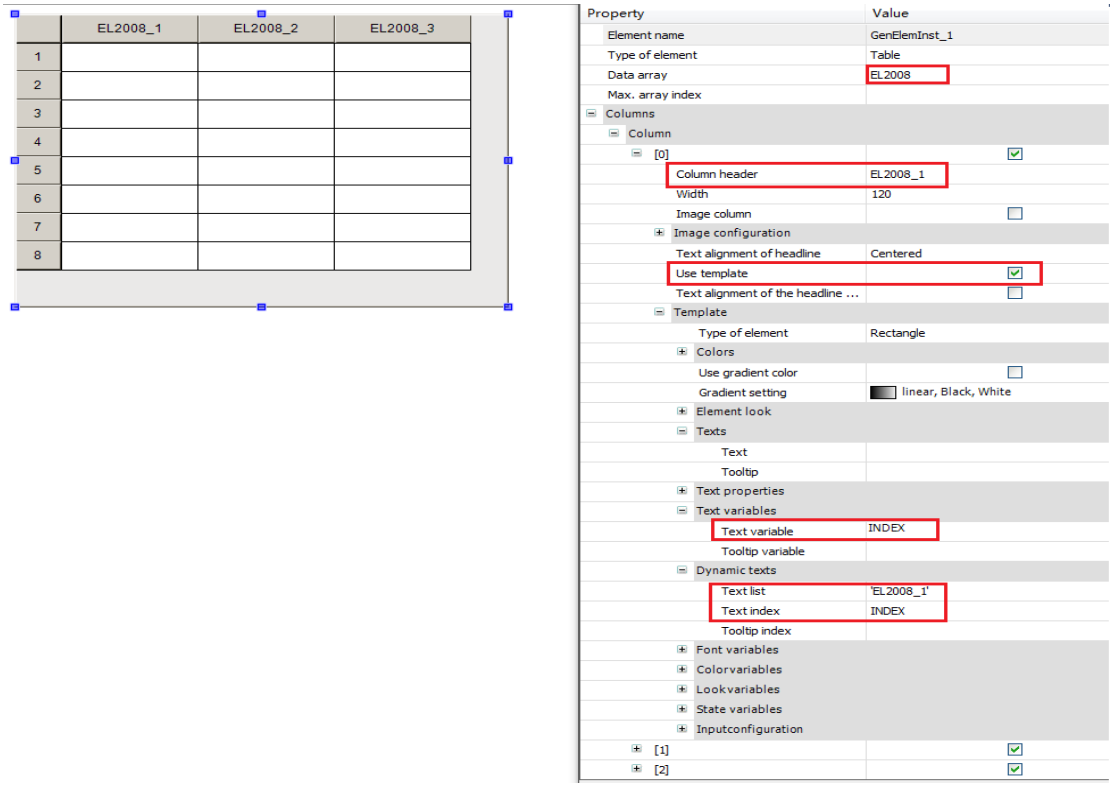

让 table 控件中的单元颜色变化来确认 EL1008 模块的输入信号的设置如下图



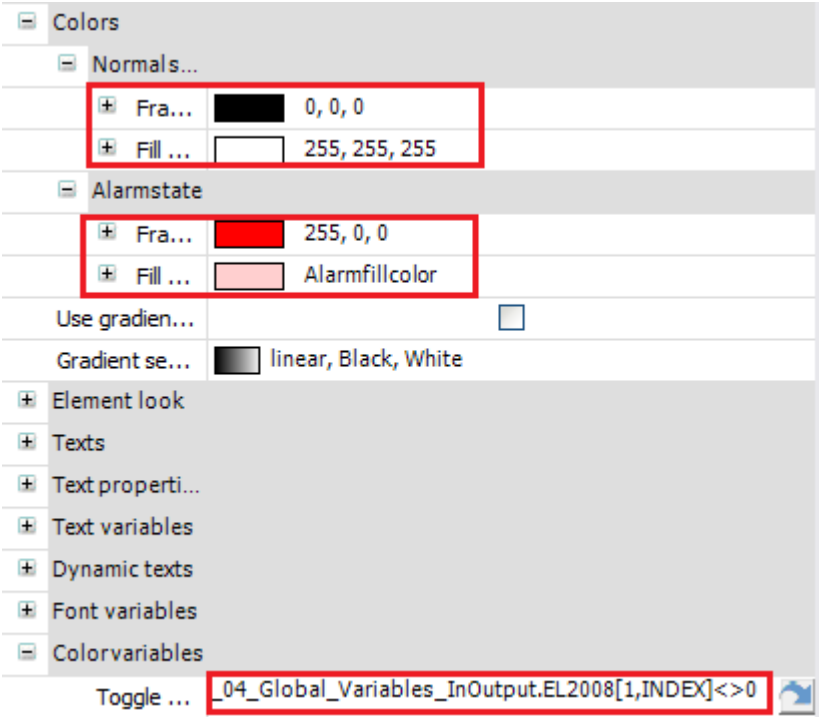

**Output 画面：**

Output 画面需要在表格中显示 EL2008 模块的每个输出点的含义，以及通过 table 控件中单元颜色的变化来确认 EL2008 模块每个输出点是否有输出。

Output 画面就包括一个 table，具体设置如下图，下图中的设置是让表格显示 EL2008 模块每个输入点的含义



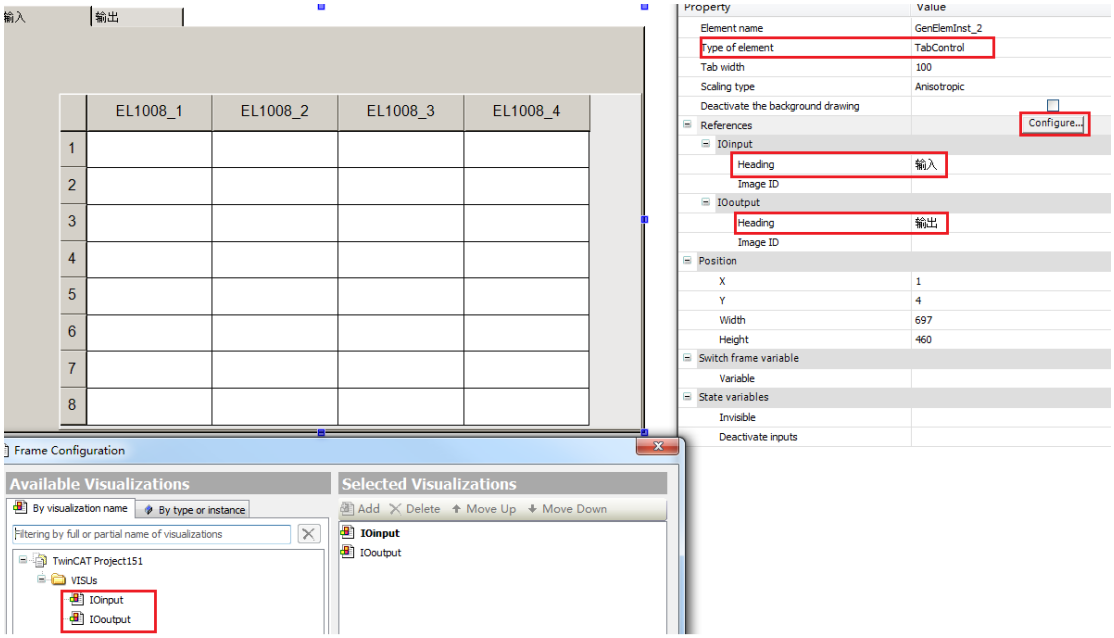

EL2008 模块的信号的输出通过 table 控件中单元的颜色变化来体现，设置如下图：




**F5_I0 画面：**

F5_I0 画面调用了 TabControl 控件，通过 TabControl 控件，可以在一个画面中监

控输入输出变量。


输入
输出
EL1008_1
EL1008_2
EL1008_3
EL1008_4
Property
Value
Element name
GenElemInst_2
Type of element
TabControl
Tab width
100
Scaling type
Anisotropic
Deactivate the background drawing
References
Configure...
IOinput
Heading
输入
Image ID
IOoutput
Heading
输出
Image ID
Position
X
1
Y
4
Width
697
Height
460
Switch frame variable
Variable
State variables
Invisible
Deactivate inputs
Frame Configuration
Available Visualizations
By visualization name
By type or instance
Filtering by full or partial name of visualizations
TwinCAT Project151
VISUs
IOinput
IOoutput
Selected Visualizations
Add
Delete
Move Up
Move Down
IOinput
IOoutput

# 第三章：运动控制

通过本章节的学习，学员将了解：

✓ MC 功能块的封装

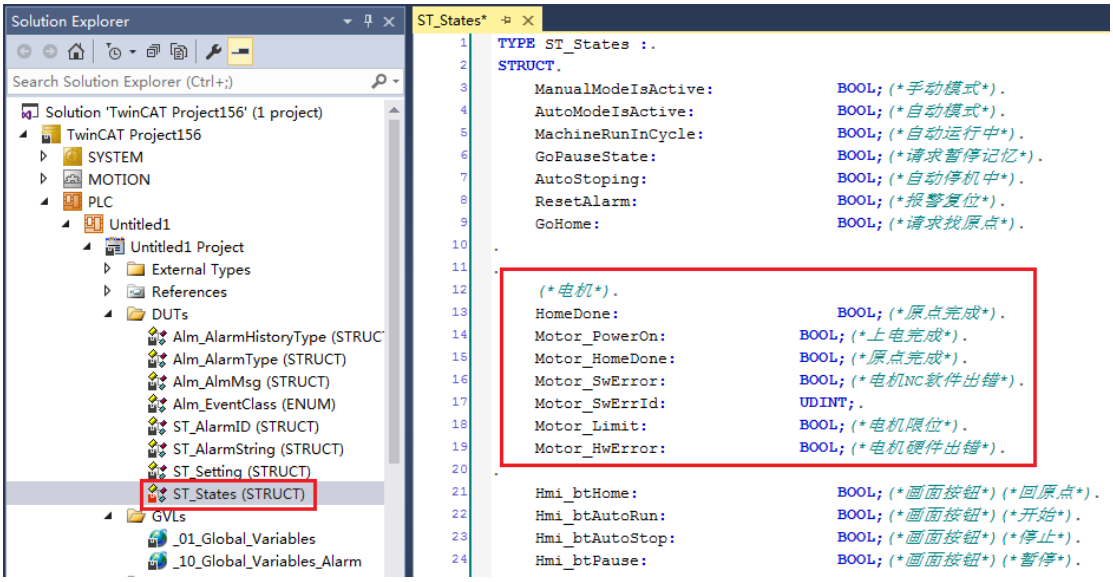
✓ 通过面向对象方法来编写运动轴控制程序

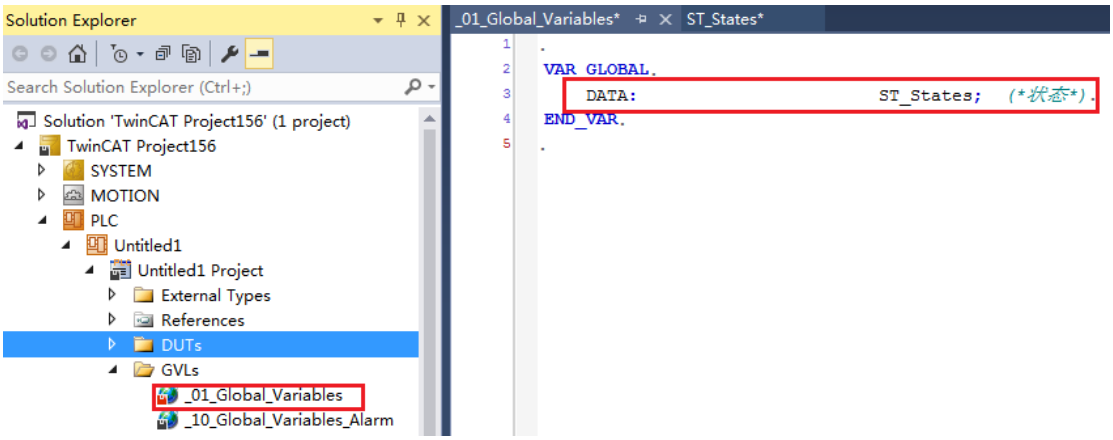
✓ 灵活运用运动控制模板，实现快速编程

本次课程设计中我们通过面向对象的方法来编写伺服轴，我们把每个伺服轴都当作是一个对象，一个轴的使能、绝对运动和相对运动等都是轴的一个能力。其中运动控制轴这个模板程序中包括了共享的状态机，封装的各种功能块和 HMI 的画面。

## 1、 伺服轴的共享状态机设置

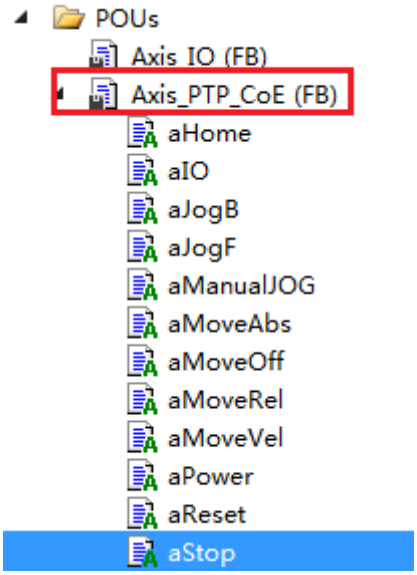
把伺服的一些状态信息设置在 ST_states 结构体中，可以让其他子程序共享运动控制的状态机。本次课程设计中，把所有子程序的状态机都放在了 ST_States。




## 2、 MC 功能块的再次封装

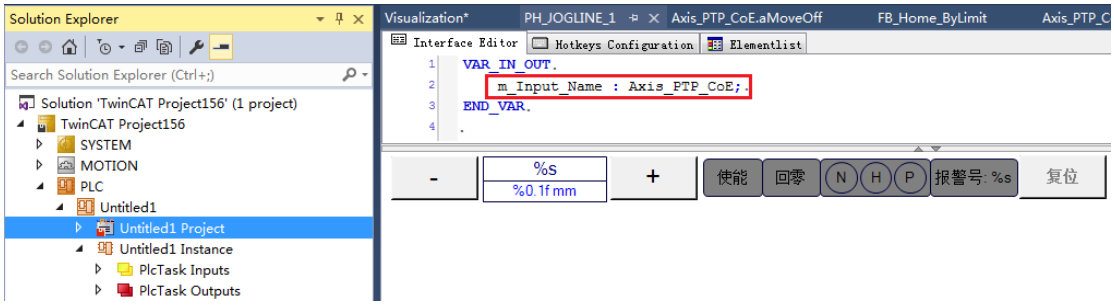
本次课程设计对运动控制的功能块又进行了封装，其中包括了 4 个功能块，分别是 Axis_IO，Axis_PTP_CoE，FB_Home_ByLimit 和 ServoDriver_DI。
ServoDriver_DI:获取伺服驱动上的原点信号，正向限位和负向限位
Axis_IO: 是对 ServoDriver_DI 继承和丰富，汇总了运动控制用到的所有硬件输入信号
FB_Home_ByLimit:轴在任意位置回零功能块，无需考虑回零方向问题。该功能块先负方向找原点，如果碰到负极限则改为正方向找原点
Axis_PTP_CoE:里面封装了 Axis 轴、MC 功能块、Action 动作和 IO 状态。该功能块包括了 12 个 Action，每个 Action 就相当于 NC 轴的一个能力。




Axis_PTP_COE.ahome:实现原点搜索的功能
Axis_PTP_COE.aIO：获取原点信号，正向限位和负向限位，并且把伺服驱动和 NC 报错信息发布给状态机中。
Axis_PTP_COE.aJogB:反向点动
Axis_PTP_COE.aJogF:正向点动
Axis_PTP_COE.aManualJOG:外部硬件控制伺服点动
Axis_PTP_COE.aMoveAbs:绝对运动
Axis_PTP_COE.aMoveOff:复位所有功能块执行标志位及输出
Axis_PTP_COE.aMoveRel:相对运动
Axis_PTP_COE.aMoveVel:速度控制
Axis_PTP_COE.aPower:伺服轴的使能
Axis_PTP_COE.aReset:刷新 IO，读取伺服轴的状态，如果有伺服报错，进行错误复位
Axis_PTP_COE.aStop:伺服轴的停止

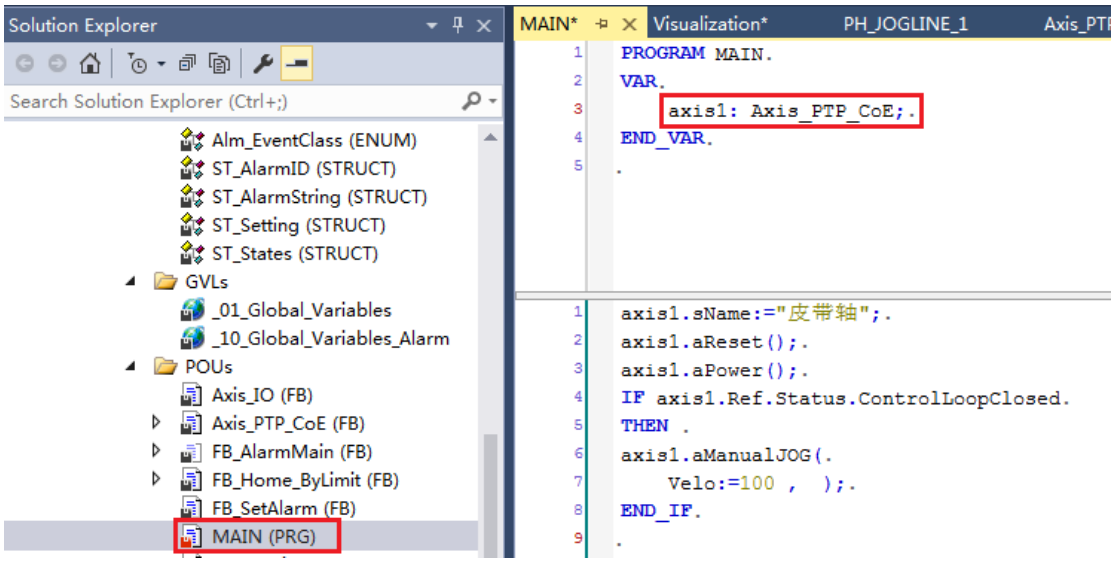
### 3、 HMI 模板制作

运动控制使用的模板是 PH_JOGLINE，该模板中包括了伺服的正反点动按钮，伺服轴名称，伺服轴运动的距离，还有一些伺服的状态位（使能完成的标志位，寻参完成的标志位，正向限位，负向限位，原点信号和 NC 的报错信息）和报错复位按钮。




## 4、 运动控制模板的使用

第一步：程序中声明轴变量

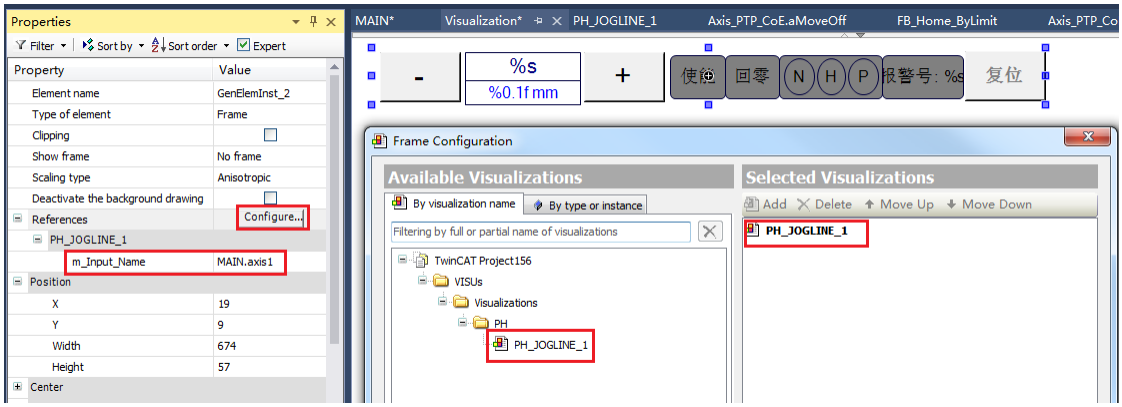



第二步：编写程序

```
axis1.sName:="皮带轴";                  //对伺服轴命名，用于区分
axis1.aReset();                        //调用相关 Action，刷新 IO，读取伺服轴的状态
axis1.aPower();                        //调用相关 Action，对伺服轴使能
IF axis1.Ref.Status.ControlLoopClosed //判断是否完成使能
THEN
axis1.aManualJOG(                      //调用相关 Action，实现点动功能
    Velo:=100 ,  );
END_IF
```

## 5、 HMI 模板的调用

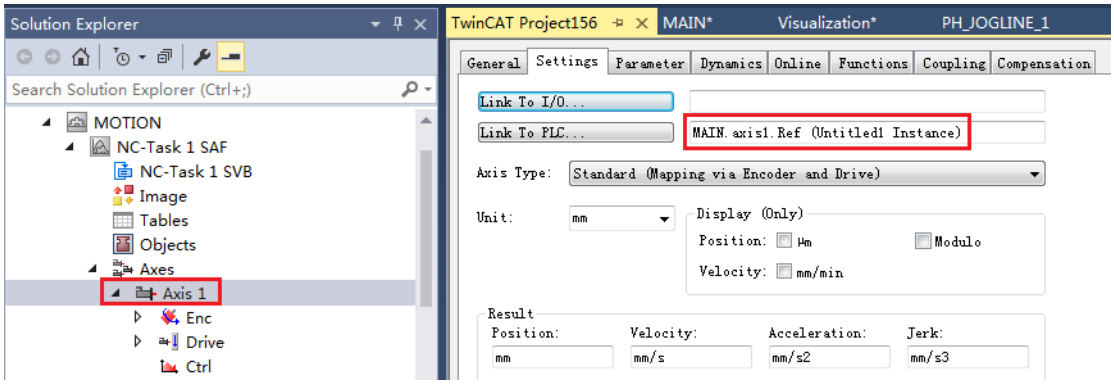
通过 HMI 中的 Frame 控件来调用 HMI 模板




调用 Frame 控件之后，进入 Frame 属性里面设置相关参数：

1) 设置调用 HMI 控件，选择 congfigure，添加 PH_JOGLINE_1 作为模板。
2) 通过绑定变量 Main.axis1，实现控制器上的程序变量和 HMI 上的控件的对应。

## 6、 创建 NC 轴，链接控制器上的轴变量



# 第四章：执行器

通过本章节的学习，学员将了解：

✓ 气缸的工作原理

✓ 根据气缸的工作原理，学会使用气缸功能块

✓ 灵活运用执行器程序模板，实现快速编程

## 1、 气缸的理论介绍

执行器，是对可以来回动作的机构的抽象。例如：气缸上下、伺服左右运行、吸气吹气等等，具有两个状态的逻辑对象。我们的气缸一般有两个状态，工作状态和非工作状态，我们一般把非工作状态叫做基本位，工作状态叫做基工作位。

气缸使用电磁阀来切换气路，通常电磁阀的作用是通过打开和关闭气路来实现对气缸伸出和缩回的动作控制。一般气缸分为单电磁阀气缸和双电磁阀气缸。

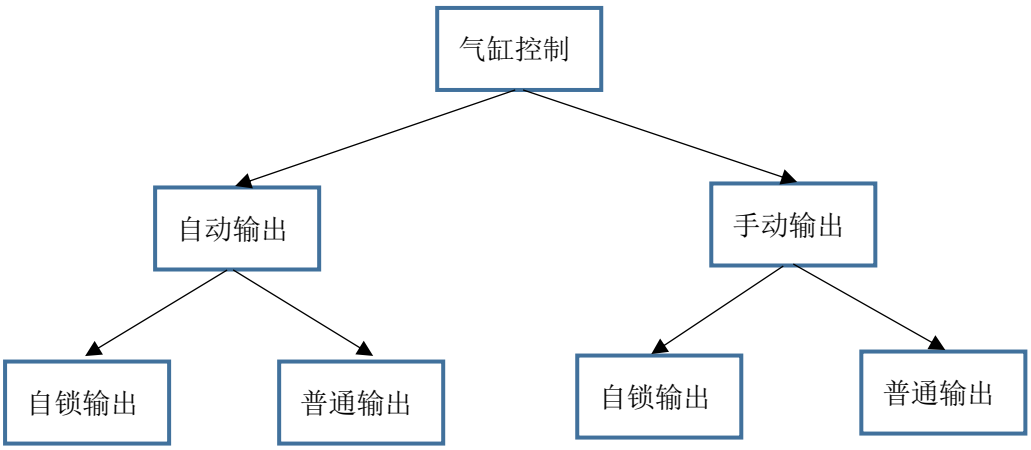
**单电磁阀：**通过单个电磁阀来控制气缸的正反动作，用磁性开关是无法使气缸停到中间某个位置的。

**双电磁阀：**通过两个电磁阀分别控制气缸的正反向运行，例如三位五通的电磁阀执行器的。

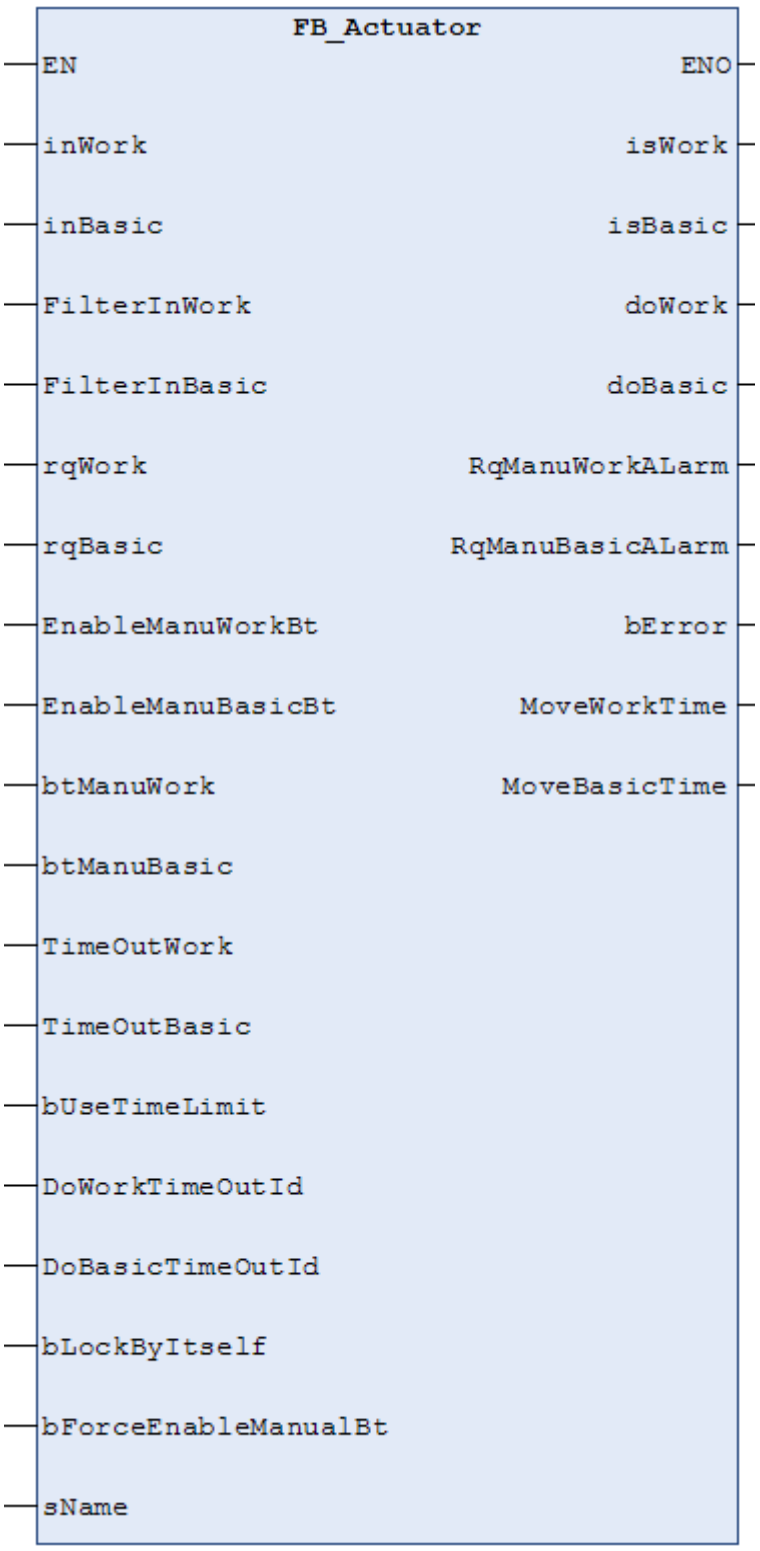
检测气缸的到位信号一般有两种方法，第一种通过时间限位来实现：根据气缸的移动速度，计算出到达工作位的时间，通过这个时间来确实是否到位。第二种通过气缸上的实际到位信号来判断是否到位。一般气缸带有磁性开关（一般 2 个，检测两端到位），用来检测气缸到位情况 （这个也是 PLC 的输入信号）

## 2、 气缸功能块的介绍

本次课程设计中使用的气缸功能块 FB_Actuator 适用于各种气缸（例如单/双电磁阀气缸，时间限位/硬件反馈气缸等），首先了解一下 FB_Actuator 控制原理。




1） FB_Actuator 功能块输入输出管脚的介绍




**输入管脚：**

| | |
|---|---|
| inWork: | 在工作位的实际信号输入 |
| inBasic: | 在基本位的实际信号输入 |
| FilterInWork: | 在工作位滤波时间 |
| FilterInBasic: | 在基本位滤波时间 |

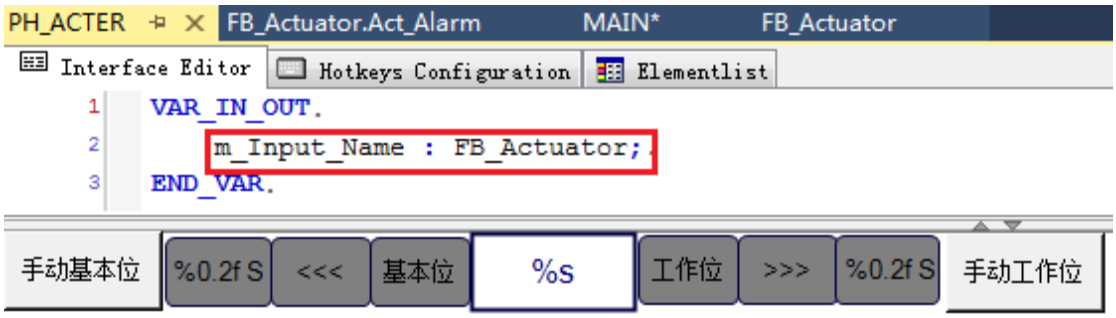
| | |
|---|---|
| rqWork: | 到工作位指令 |
| rqBasic: | 到基本位指令 |
| EnableManuWorkBt: | 使能到工作位按钮 |
| EnableManuBasicBt: | 使能到基本位按钮 |
| btManuWork: | 手动按钮 |
| btManuBasic: | 手动按钮 |
| TimeOutWork: | 到工作位超时报警时间，单位 s |
| TimeOutBasic: | 到基本位超时报警时间，单位 s |
| bUseTimeLimit: | 利用时间做限位到位开关，不使用实际的反馈点 |
| DoWorkTimeOutId: | 超时报警的 ID 号 |
| DoBasicTimeOutId: | 超时报警的 ID 号 |
| bLockByItself: | 是否自锁 |
| bForceEnableManualBt: | 强制使用手动按钮功能,不需要转换到手动模式 |
| sName: | 执行器名字 |

**输出管脚：**

| | |
|---|---|
| IsWork: | 已经到了工作位 |
| IsBasic: | 已经到了基本位 |
| doWork: | 开始到工作位输出信号 |
| doBasic: | 开始到基本位输出信号 |
| RqManuWorkALarm: | 手动，不再工作位条件不满足报警 |
| RqManuBasicALarm: | 手动，不再基本位条件不满足报警 |
| bError: | 错误 |
| MoveWorkTime: | 到工作位时间计时，单位 s |
| MoveBasicTime: | 到基本位时间计时，单位 s |

2) 执行器的 HMI 模板介绍

执行器使用的模板是 PH_ACTER，该模板中包括了手动控制执行机构的控件，显示从基本位（工作位）到工作位（基本位）移动的时间控件，到达基本位，工作位的状态显示控件和执行器的名字。




3) 模板的调用

本次课程设计执行器子程序中，是单电磁阀的气缸，并且有实际信号的反馈。具体调用步骤如下：

**第一步：**新建变量，实例化功能块

```
PROGRAM MAIN.
VAR.
fbActuator:FB_Actuator;.
FBisManualActive:BOOL;.
rqwork: BOOL;.
END_VAR.
```

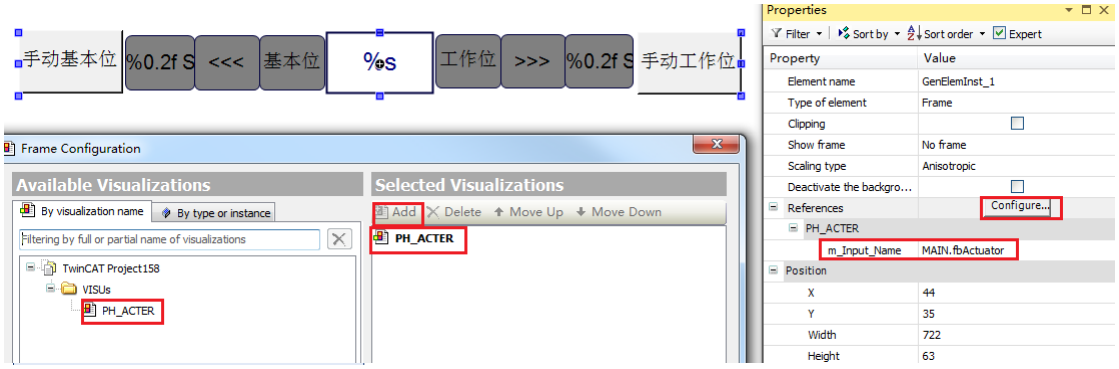
其中 FBisManualActive 是自动使能，rqwork 是要求执行器从基本位运行到工作位。

**第二步：调用相关功能块**

```
RqSysAutoModeIsActive:=true;                    //切换到自动模式
fbActuator.sName:="吸嘴";                        //对执行器命名，用于区分
fbActuator(
    inWork:=stInput.biSnVacuuSor,               //执行器反馈到位信号
    FilterInWork:=t#20ms ,                      //反馈信号的滤波
    rqWork:=rqwork ,                            //要求执行器动作
    sName:= ,
    isWork=> ,
    isBasic=> ,
    doWork=>stOutput.bqClamperSuck,             //用输出模块控制执行器的动作
    );
```

第三步：HMI 模板调用

通过 HMI 中的 Frame 控件来调用 HMI 模板




调用 Frame 控件之后，进入 Frame 属性里面设置相关参数：

1) 设置调用 HMI 控件，选择 congfigure，添加 PH_ACTER 作为模板。
2) 通过绑定变量 Main.fbActuator，实现控制器上的程序变量和 HMI 上的控件的对应。

# 第五章：逻辑控制

通过本章节的学习，学员将了解：

- ✓ 码垛机的工作流程
- ✓ 通过面向过程编程方法，实现逻辑控制

## 1、 码垛机流程介绍

逻辑控制主要采用面向过程的编程方法，把所有的分解动作串联在一起。分为两部分，本次课程设计中第一部分是控制传送带和剪断机，第二部分是码垛。第一部分中分别用一个 NC 轴来模拟传送带，剪断机中包括两个气缸，压箱气缸和剪断气缸。除此之外，还用到两个传感器，分别检测箱子进入传送带和箱子进入剪断机。第二部分是码垛，分别用 3 个 NC 轴来模拟机械手的 X 方向,Y 方向和 Z 方向，其中还包括一个吸嘴气缸，除此之外，还包括一个传感器用于检测是否到达码垛位置。

第一部分的动作流程介绍：

第一个传感器检测到箱子进入到传送带----开启传送带，让传送带匀速运动----第二个传感器检测到箱子进入到剪断机，让压箱气缸动作----当第二个传感器检测不到箱子时，让剪断气缸动作----等待设定时间后，复位压箱气缸和剪断气缸----等待下次循环

第二部分的动作流程介绍：

当传感器检测到箱子在码垛位置----Z 轴放下固定距离----让吸嘴动作----当吸嘴到达工作位，Z 轴上升到固定位置----X,Y 轴移动到对应的码垛位置----Z 轴下降----复位吸嘴气缸----Z 轴上升---XY 轴回到初始位置----等待下次循环

## 2) 逻辑控制程序介绍

本次课程设计中，通过 APP_Conv 功能块，实现第一部分的流程。APP_Conv 里面调用了一个 Axis_PTP_COE 功能块和两个 FB_Actuator 功能块，Axis_PTP_COE 用于控制传送带，FB_Actuator 用于控制压箱气缸和剪断气缸。

```
FUNCTION_BLOCK APP_Conv(*逻辑动作*)(*传送带*).
VAR.
    SysStep:                DINT;.
    AutoStep:               DINT;.
    HomeStep:               DINT;.
.
    ACTR:                   ARRAY[1..2] OF FB_Actuator;
    Axis:                   Axis_PTP_CoE;(*运动轴*).
    HomeDone:               BOOL;.
    TON_Cut:                TON;.
    FBisManualActive:       BOOL;.
    FBisAutoActive:         BOOL;.
END_VAR.
VAR CONSTANT(*局部常量，大写*).
    _UD:                    INT:=1;(*上下压箱气缸*).
    _CT:                    INT:=2;(*剪断气缸*).
END_VAR.
```

APP_Conv 包括 5 个 Action,每个 Action 功能说明：
_00_Init:用于系统变量的初始化，以及对伺服轴和气缸命名
_10_AUTO:通过 Case 语句编写动作流程
_20_Manual:手动控制传送带轴
30_FB:状态监控及调用气缸功能块
40_Home:让执行器和伺服轴回到初始位置
APP_Conv 功能块通过 case 语句来调用这 5 个 Action,首先进行系统初始化和硬件初始化。接着根据全局变量中的 RqSysAutoModeIsActive 变量的值，确定到底执行自动或者手动的程序。
具体程序如下：

```
CASE SysStep OF
    0:  (*Init*)
        _00_Init();
        SysStep:=100;

    100:(*Home*)
        _40_Home();
        IF HomeDone THEN
            HomeStep:=0;
            SysStep:=200;
        END_IF

    200:(*Manual*)
        IF NOT RqSysAutoModeIsActive THEN
            _20_Manual();
        ELSE
            Axis.aMoveOff();(*停止伺服电机的运动*)
```

```
            SysStep:=300;
        END_IF

    300:(*Auto*)
        IF RqSysAutoModeIsActive
        AND RqSysMachineRunInCycle
        AND NOT RqSysGoPauseState
        AND NOT gbAlarm_NeedPause
        AND NOT gbAlarm_StopAlarm THEN
            _10_AUTO();
        ELSE
            Axis.aMoveOff();(*停止伺服电机的运动*)
            IF NOT RqSysAutoModeIsActive THEN
                SysStep:=200;
            END_IF
        END_IF
END_CASE
_30_FB();
```

通过 APP_Clamp 实现第二部分的码垛。
APP_Clamp 里面调用了三个 Axis_PTP_COE 功能块和一个 FB_Actuator 功能块，Axis_PTP_COE 用于控制机械手的 X 方向，Y 方向和 Z 方向，FB_Actuator 用于控制吸嘴。

```
FUNCTION_BLOCK APP_Clamp(*逻辑动作*)(*下料仓机构*).
VAR.
    SysStep:                DINT;.
    AutoStep:               DINT;.
    HomeStep:               DINT;.
.
    ACTR:                   ARRAY[1..1] OF FB_Actuator;.
    AxisX:                  Axis_PTP_CoE;(*上下运动轴*).
    AxisY:                  Axis_PTP_CoE;(*上下运动轴*).
    AxisZ:                  Axis_PTP_CoE;(*上下运动轴*).
    HomeDone:               BOOL;.
.
    ClampBoxID:             DINT;.
    PlacePosX:              REAL;.
    PlacePosY:              REAL;.
    PlacePosZ:              REAL;.
    FBisManualActive:       BOOL;.
    FBisAutoActive:         BOOL;.
END_VAR.
VAR CONSTANT(*局部常量，大写*).
    _SK:                    INT:=1;(*吸气嘴*).
END_VAR.
```

APP_Clamp 包括 5 个 Action,每个 Action 功能说明：
_00_Init:用于系统变量的初始化，以及对伺服轴和气缸命名
_10_AUTO：通过 Case 语句编写动作流程
_20_Manual:手动控制机械手
30_FB: 状态监控及调用气缸功能块
40_Home:让执行器和伺服轴回到初始位置
APP_Clamp 功能块通过 case 语句来调用这 5 个 Action,首先进行系统初始化和硬件初始化。接着根据全局变量中的 RqSysAutoModeIsActive 变量的值，确定到底执行自动或者手动的程序。

```
CASE SysStep OF
    0:  (*Init*)
        _00_Init();
        SysStep:=100;

    100:(*Home*)
        _40_Home();
        IF HomeDone THEN
            HomeStep:=0;
            SysStep:=200;
        END_IF

    200:(*Manual*)
        IF NOT RqSysAutoModeIsActive THEN
            _20_Manual();
        ELSE
            AxisX.aMoveOff();(*停止伺服电机的运动*)
            AxisY.aMoveOff();(*停止伺服电机的运动*)
            AxisZ.aMoveOff();(*停止伺服电机的运动*)
            SysStep:=300;
        END_IF

    300:(*Auto*)
        IF RqSysAutoModeIsActive
        AND RqSysMachineRunInCycle
        AND NOT RqSysGoPauseState
        AND NOT gbAlarm_NeedPause
        AND NOT gbAlarm_StopAlarm THEN
            _10_AUTO();
        ELSE
            AxisX.aMoveOff();(*停止伺服电机的运动*)
            AxisY.aMoveOff();(*停止伺服电机的运动*)
            AxisZ.aMoveOff();(*停止伺服电机的运动*)
            IF NOT RqSysAutoModeIsActive THEN
                SysStep:=200;
```

```
                END_IF
            END_IF
END_CASE
_30_FB();
```

# 第六章：动画说明

通过本章节的学习，学员将了解：

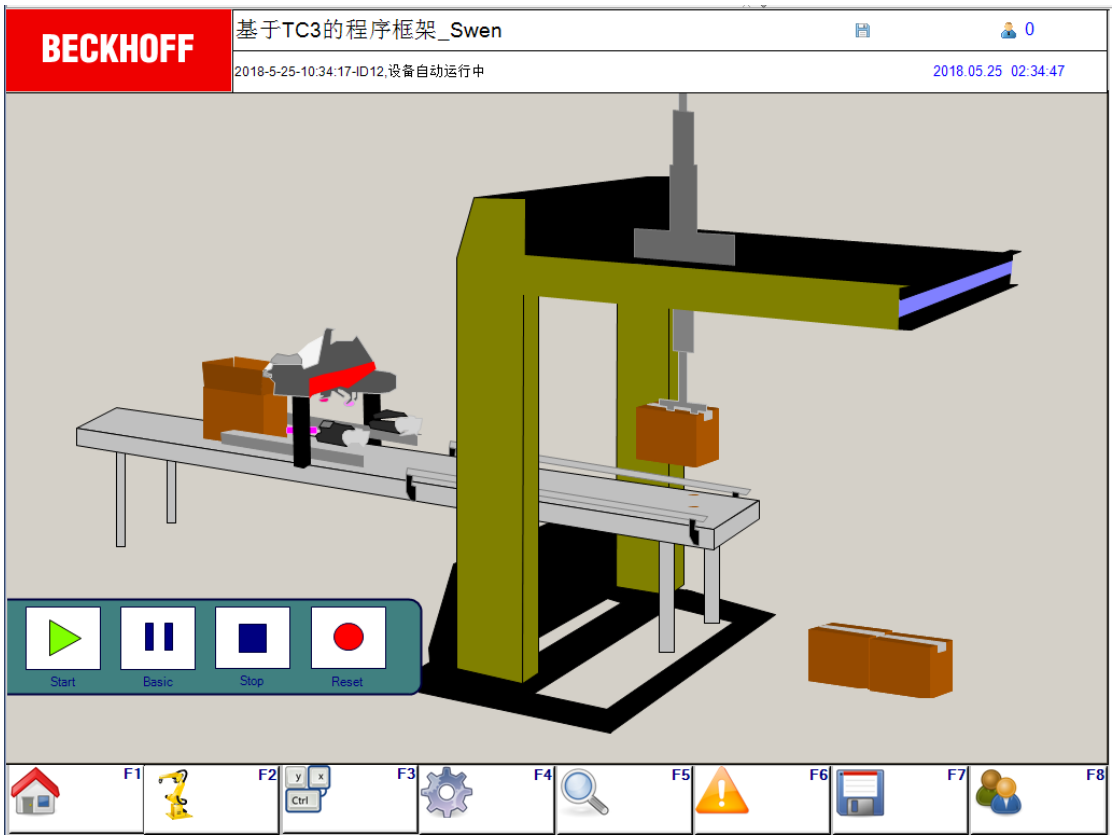
✓ 码垛机 HMI 动画的实现

✓ 多语言切换功能

## 1、 HMI 画面动画介绍

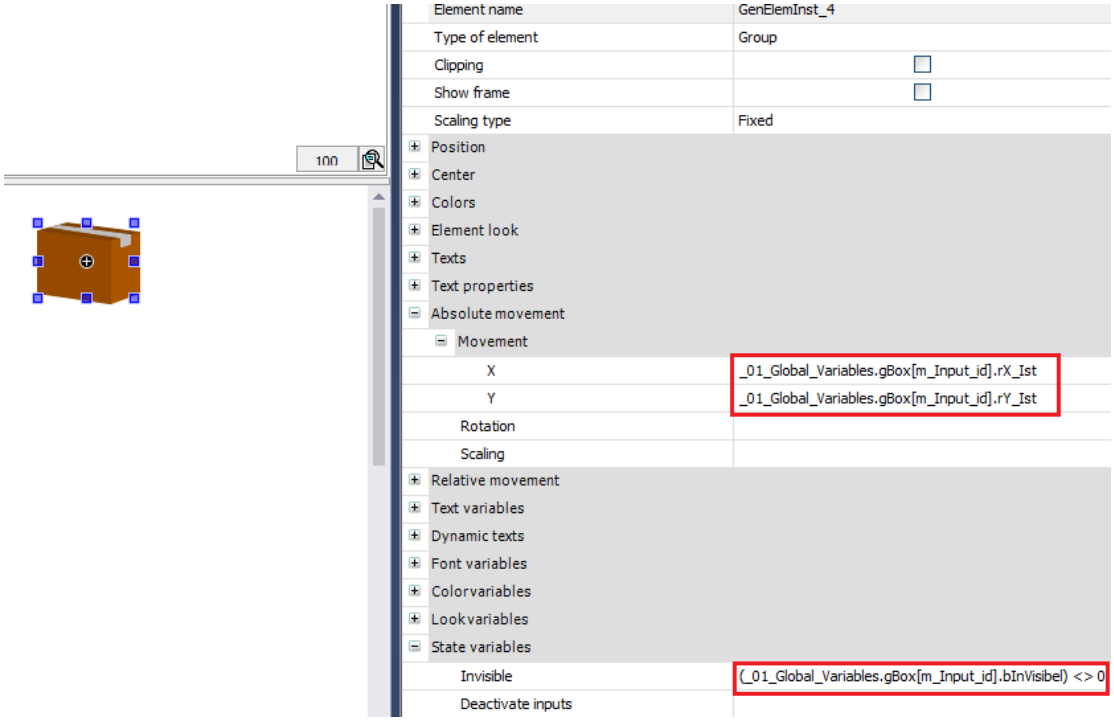
本次课程设计中，用到了很多动画，其中涉及到的动画的控件有箱子，吸头，小连杆和 Y 滑台。

HMI 画面的动画说明：

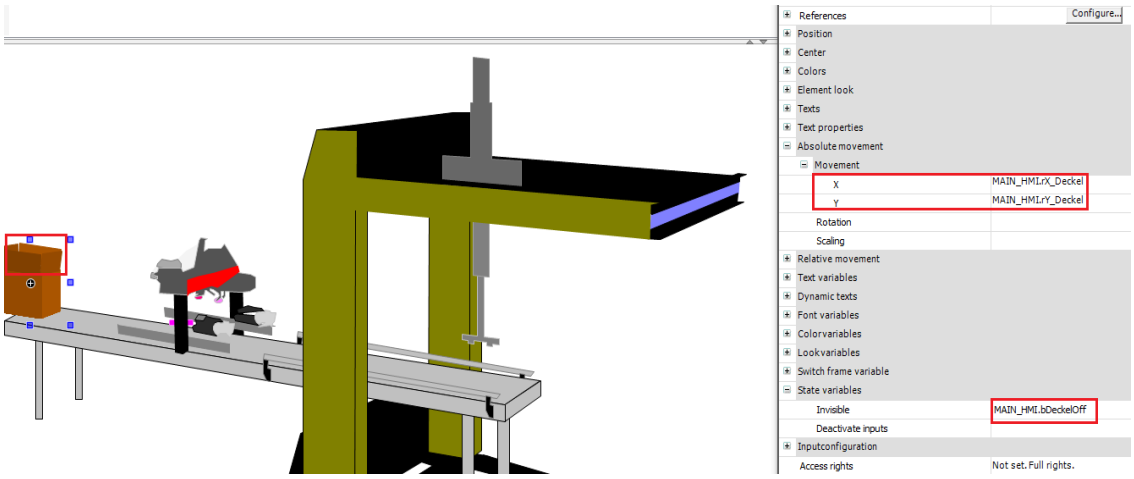



当箱子放在传送带上，箱子的本体和箱子的上盖都要移动和显示，当进入剪断机之后，仅仅显示箱子的本体。当到达码垛时，箱子，吸头，小连杆和 Y 滑台都需要移动。

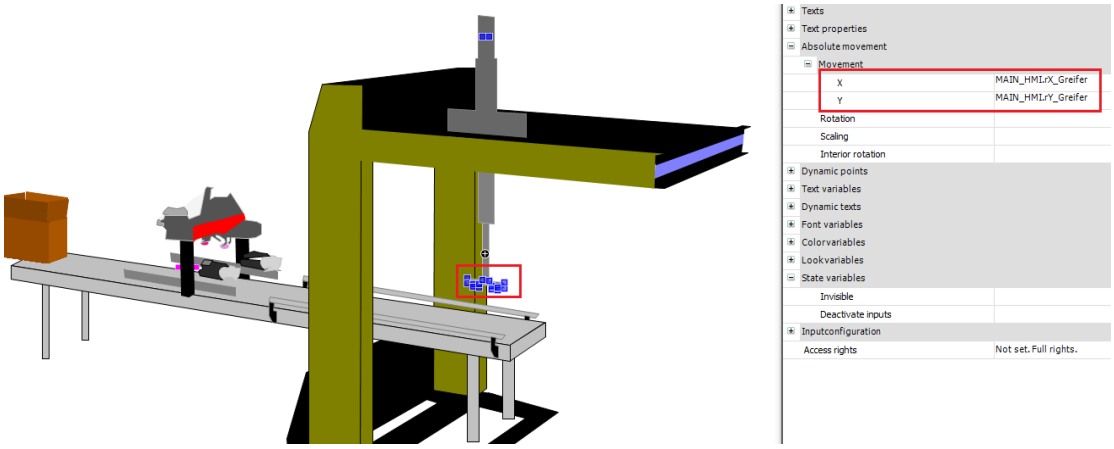
箱子本体的 HMI 属性设置：



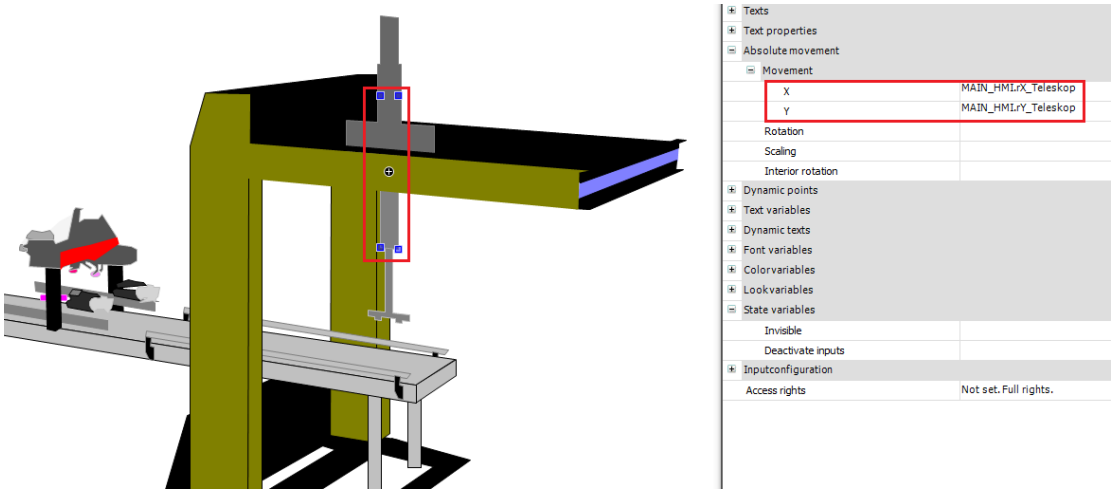

箱子上盖的 HMI 设置：



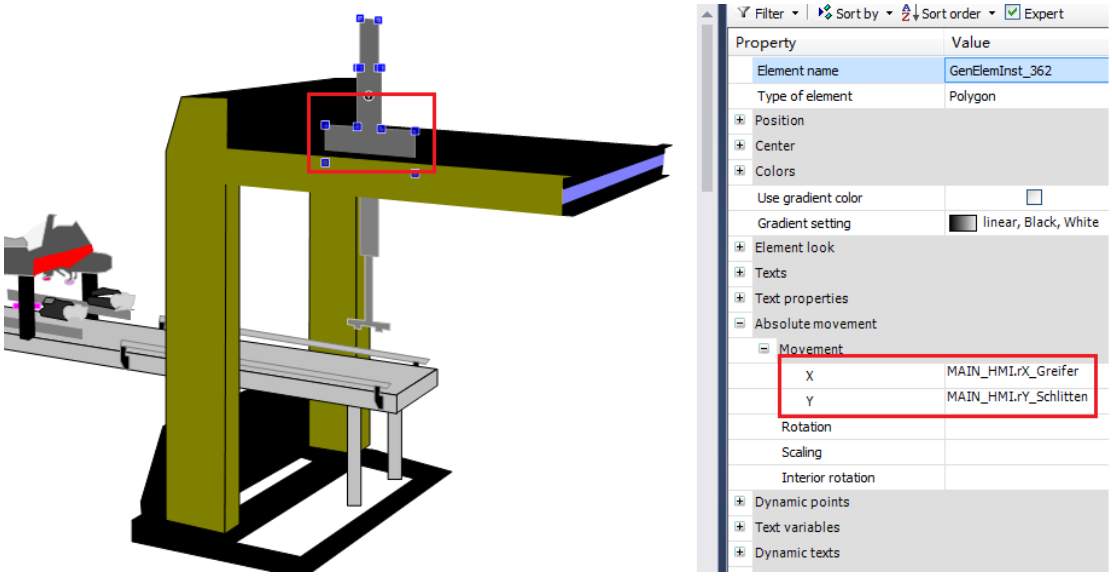

吸头的 HMI 设置：




小连杆的 HMI 设置：




Y 滑台的 HMI 设置：




## 2、 HMI 动画程序介绍

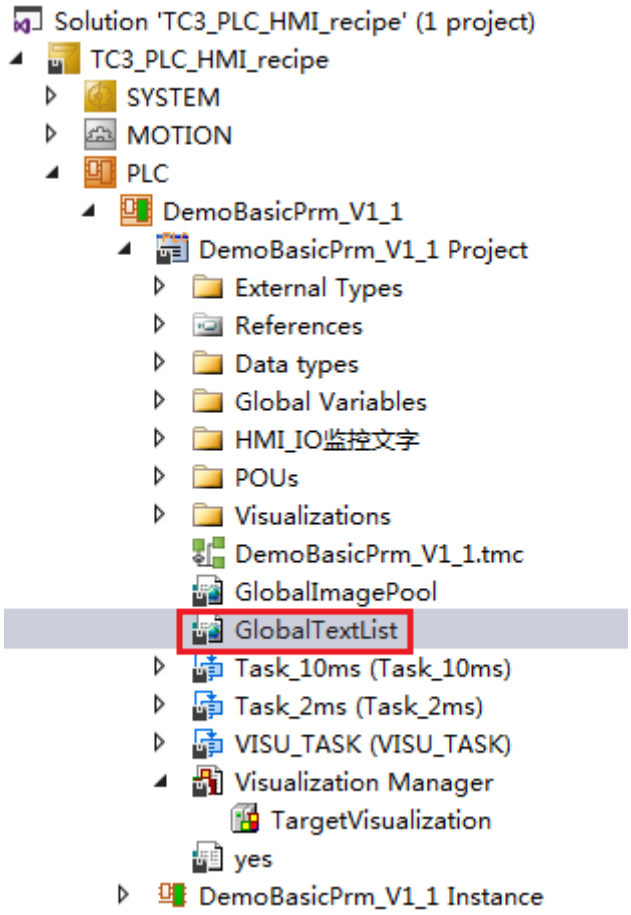
HMI 动画中包括了一个自定义 Function（fnGetHMIpos）,一个主程序（MAIN_HMI）

包括三个子程序（all_Init,act_ShowBoxPos,act_ShowClampPos）。
fnGetHMIpos:把实际的物理移动距离转化为 HMI 画面上移动距离的比例换算
act_init:初始化
act_ShowBoxpos:计算出箱子在 HMI 画面上移动的距离
箱子总共有三种状态：在传送带上，在抓手上和在码垛上。需要分别根据这三种状态，计算出相应箱子状态的移动距离。
act_ShowClampPos：计算出抓手上的吸头，小连杆和 Y 滑台在 HMI 画面上移动的距离。

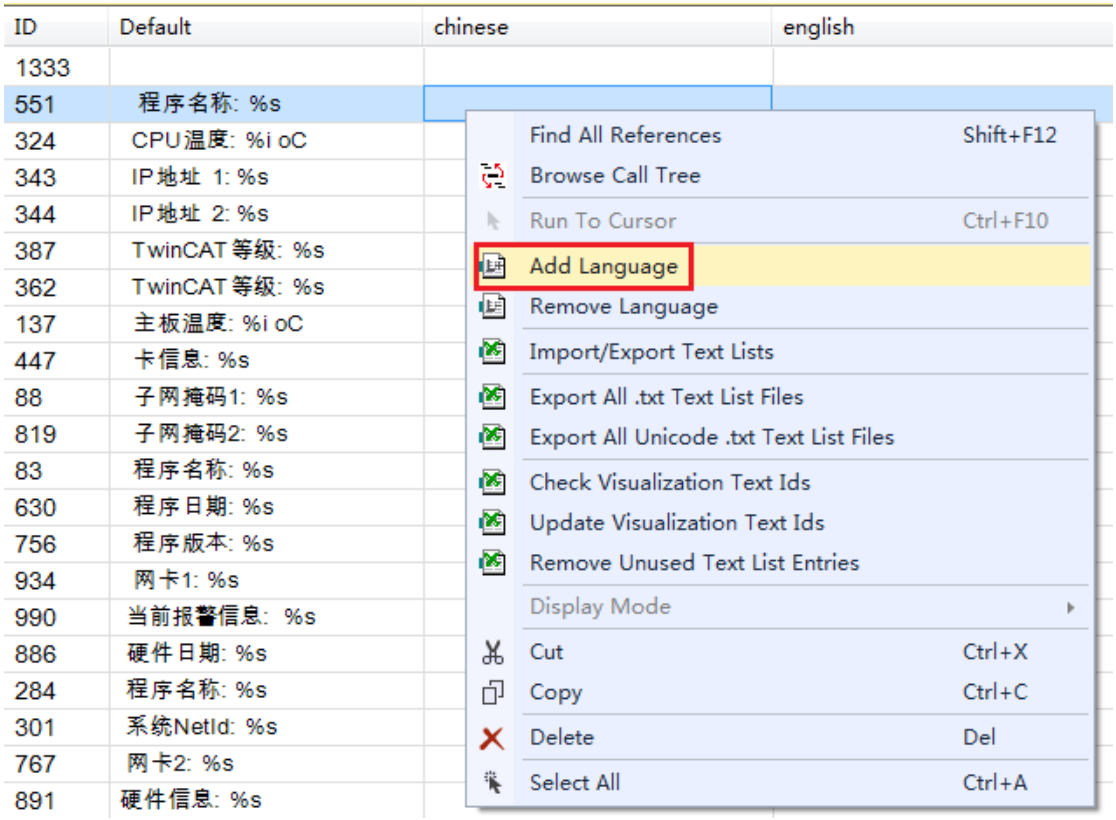
### 3、 多语言切换的介绍

多语言切换同样是通过 InputConfiguration 来实现，在 TwinCAT3 中提供 GlobalTextList，包含在界面中出现的所有的语言文字，并且可以在 GlobalTextList 手动添加语言，对界面中使用的文本进行翻译。

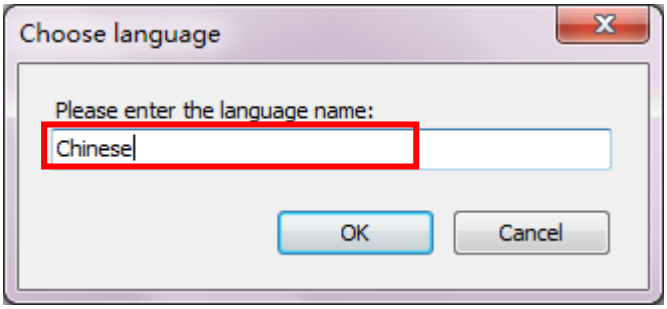
（1） 在左侧管理窗口 VISUs 文件夹下找到 GlobalTextList 这个文件




（2） 双击该文件后会出现之前在 Visualization 中已经编辑过的文字，在空白处右键，在菜单中找到 Add Language，单击后在弹出的对话框中添加语言，点击 OK




输入添加语言名称




在新建语言 Chinese 下对界面中文本进行翻译

| | | |
|---|---|---|
| 11 | Year | 年 |
| 12 | AlarmHistory | 报警历史 |
| 13 | AlarmActive | 当前报警 |
| 14 | ClearHistory | 清除历史 |
| 15 | Start | 启动 |
| 16 | Stop | 停止 |
| 17 | Reset | 复位 |
| 18 | SaveRecipe | 保存配方 |
| 19 | LoadRecipe | 调用所选配方 |
| 20 | flashlist | 刷新列表 |
| 21 | yes | 是 |
| 22 | No | 否 |
| 23 | delete | 删除 |
| 24 | UserLogOut | 用户注销 |
| 25 | UserLogIn | 登录 |

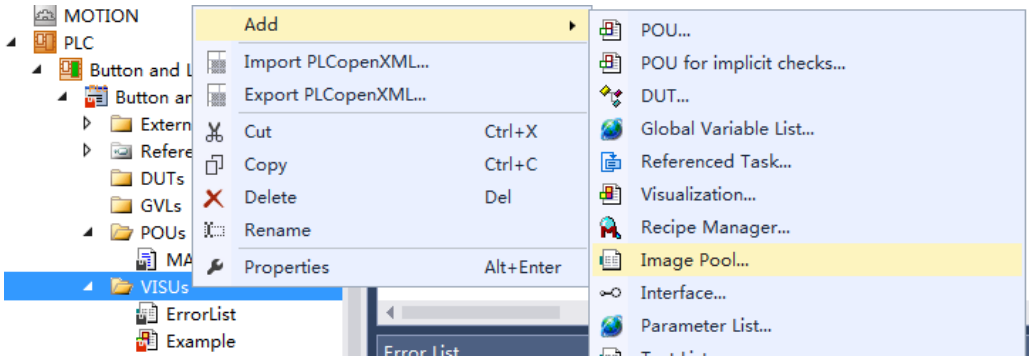
再新建一个语言 English，输入对应的文本

| | | | |
|---|---|---|---|
| 11 | Year | 年 | Year |
| 12 | AlarmHistory | 报警历史 | AlarmHistory |
| 13 | AlarmActive | 当前报警 | AlarmActive |
| 14 | ClearHistory | 清除历史 | ClearHistory |
| 15 | Start | 启动 | Start |
| 16 | Stop | 停止 | Stop |
| 17 | Reset | 复位 | Reset |
| 18 | SaveRecipe | 保存配方 | SaveRecipe |
| 19 | LoadRecipe | 调用所选配方 | LoadRecipe |
| 20 | flashlist | 刷新列表 | Flash |
| 21 | yes | 是 | yes |
| 22 | No | 否 | no |
| 23 | delete | 删除 | Delete |
| 24 | UserLogOut | 用户注销 | UserLogOut |
| 25 | UserLogIn | 登录 | UserLogIn |
| 26 | ChangeLvPass | 修改左边等级的密码 | ChangeLvPass |
| 27 | ChangeLv | 修改的等级 | ChangeLv |
| 28 | Moniter | 输入输出 | Moniter |
| 29 | Moniter | 步进 | Moniter |

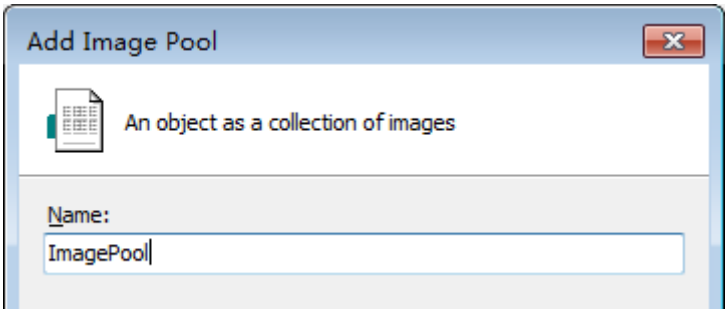
（3） 完成以上操作后需要在界面中通过按钮或者图片进行语言切换，以图片作为语言切换的提示，需要使用 TwinCAT3 HMI 中的 Image Pool

（4） TwinCAT3 中 Image Pool 的使用

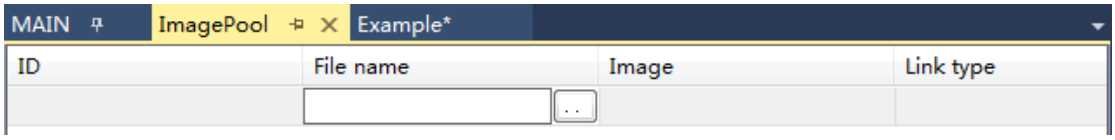
在左侧配置管理窗口，找到 VISUs 文件夹，右键 Add，在菜单中选择 Image Pool



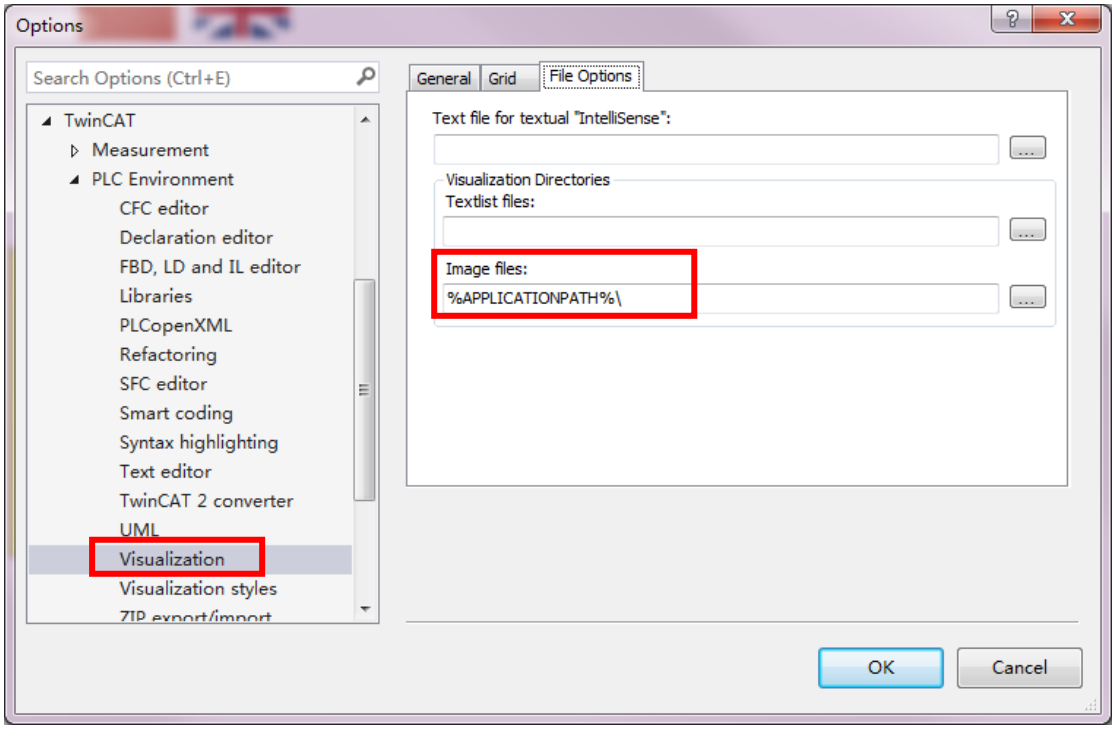

在弹出的对话框中命名其为 Image Pool，点击 Open

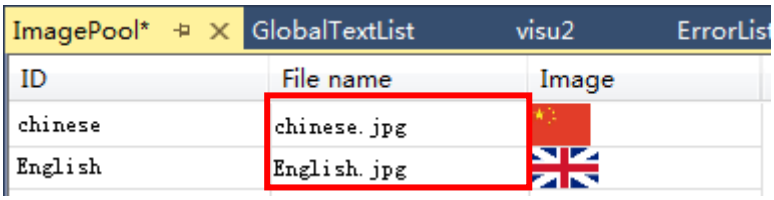



（5） 双击添加好的 ImagePool，在右边的配置窗口中的 File name 一列通过右边的 .. 添加在界面中会使用到的图标，

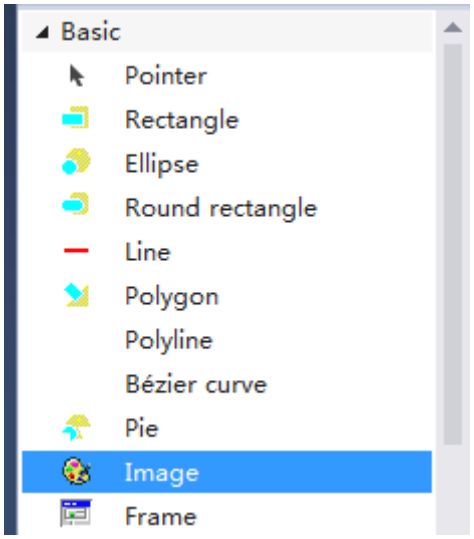



如果将图片添加到 TwinCAT3 自动创建的项目文件夹中，并且确保菜单栏 Tools→options 中的 Visualization 右边 File Options 下的 Image files 路径为%APPLICATION%，就可以看到添加的图片名称不再带有路径，否则如果改变导入的图片路径会导致画面中的图片无法显示

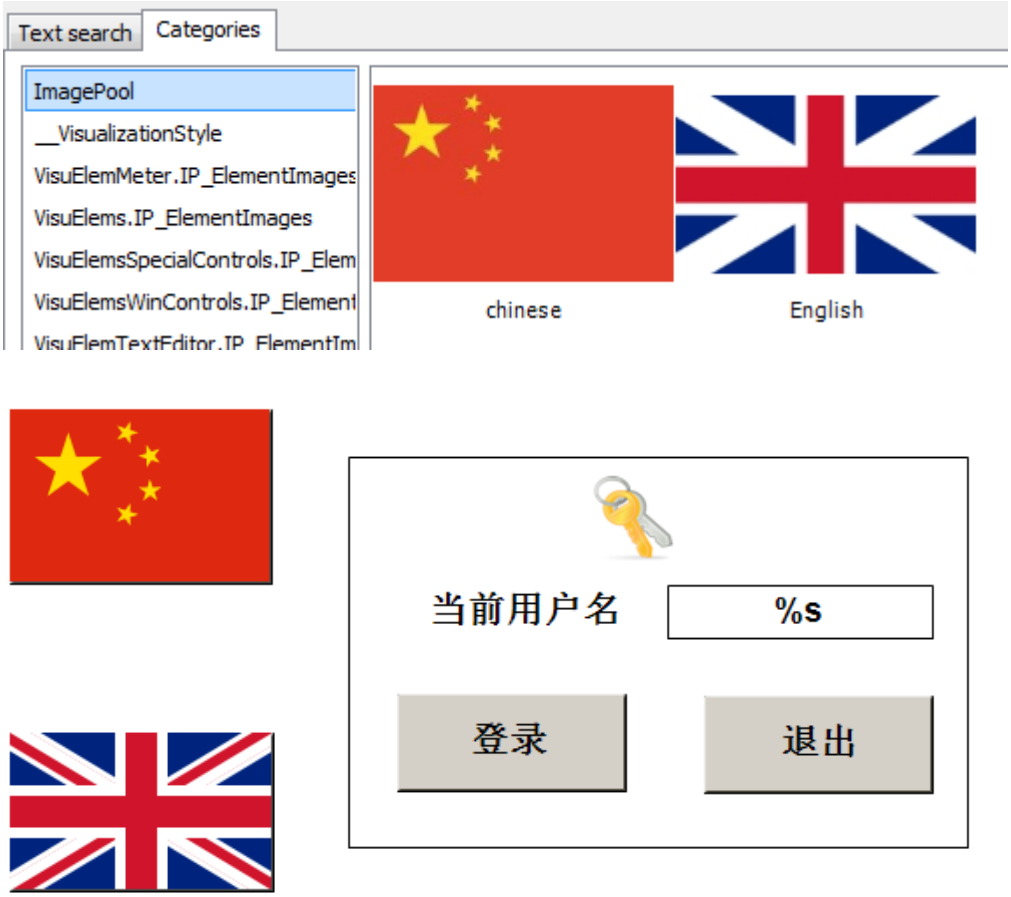


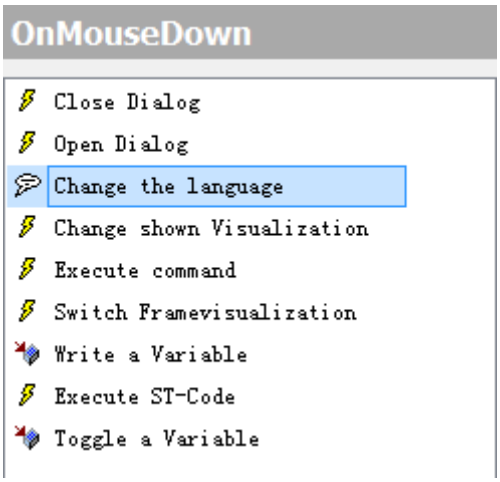

（6） 在画面中添加 Toolbox→Basic→Image 显示图片




将 Image 拖到画面中会自动弹出对话框，在 Categories 选项卡下选择刚刚生成的图库 ImagePool，右侧小窗口会显示图库中的图片，选择需要的图片，点击 OK，在画面上将其调节到合适的大小。

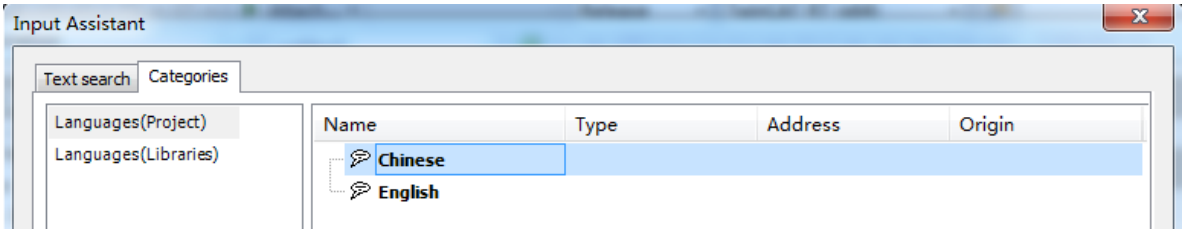



（7） 选中显示中国国旗，在右侧属性栏里将 Inputconfiguration 展开，选择 On MouseClick，单击 Configure…后在弹出的对话框左侧选项中双击 Change the language




在右侧属性栏点击 Language 下空白处右边的小按钮

在弹出的对话框中选择之前创建的语言，选中 Chinese，点击确定




（8） 同理，选中显示英国国旗，在右侧属性栏里将 Inputconfiguration 展开，单击 Configure 后在弹出的对话框左侧选项中双击 Change the language，在右侧属性栏点击 Language 下空白处右边的小按钮，在弹出的对话框里选中 English 点击确定

（9） 上述操作完成后，激活并运行程序，效果如下：


点击英国国旗，切换到英文


current user name
login
logout

# 第七章：TwinCAT3 HMI 用户管理

通过本章节的学习，学员将了解：

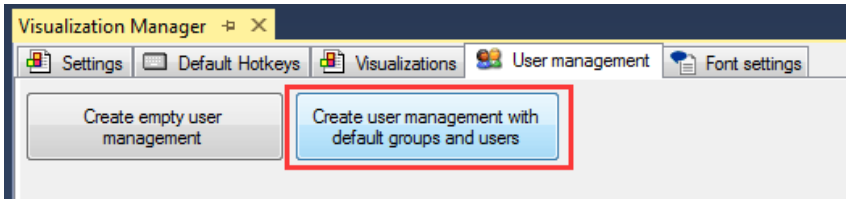
✓ 用户管理的作用

✓ HMI 用户管理的使用

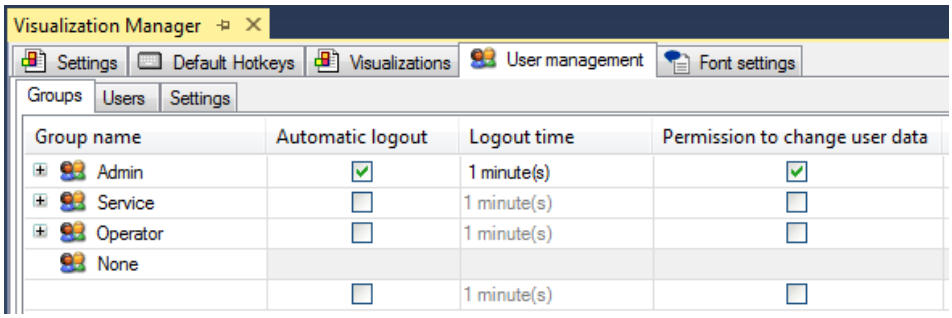
每台倍福控制器的 HMI 画面上可以创建多个不同的用户，用户等级是用来设定不同用户的访问权限，可大大提高应用工程的安全性。不同用户所对应的控件操作权限可以进行自定义配置，大大提高了应用工程的安全性。

## 创建 HMI 用户管理的步骤：

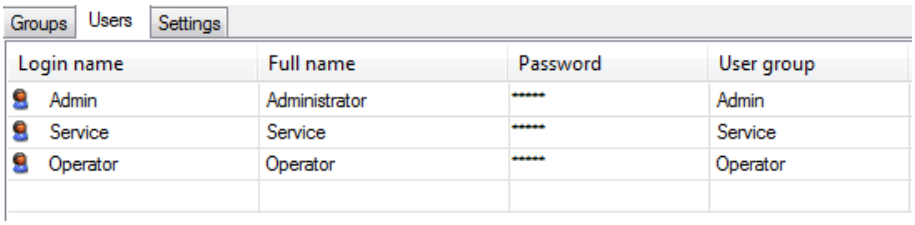
（1） 在 Visualization Manager 中创建一个默认的用户管理



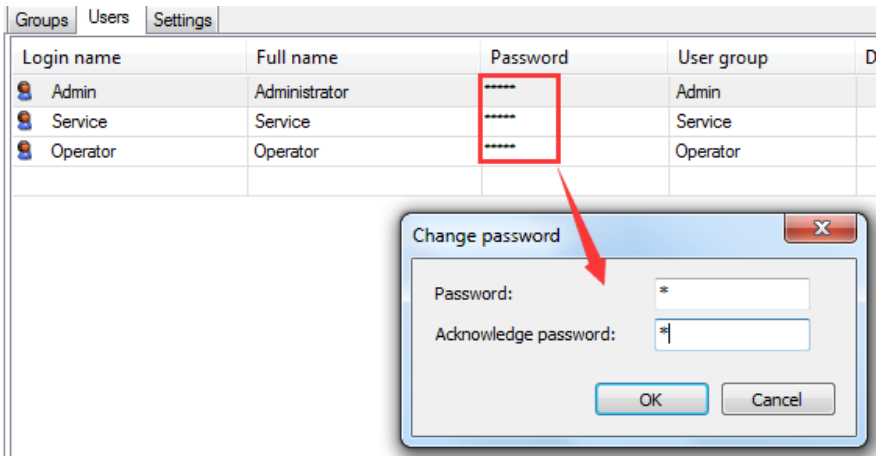

（2）创建好后，可以发现此默认的用户管理中已经创建了 3 个用户组



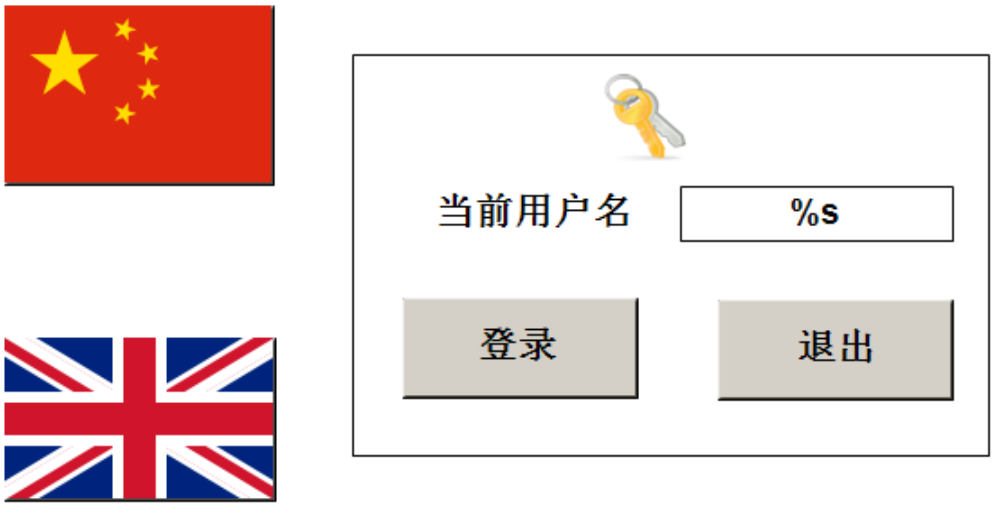

并且在 Users 中可以看到同时也已经创建了 3 个用户，并且对应到了不同组中



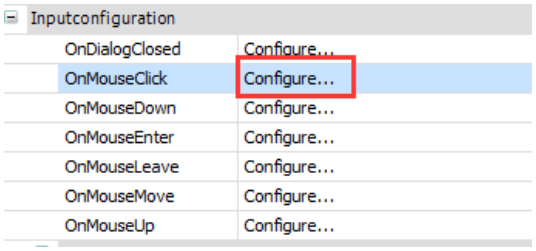

（3）在这里你可以新建用户组或者用户，并且修改登录名和密码，比如我把自带的 3 个用户密码都设置为 1，只需要点击 Password 中对应的一栏即可



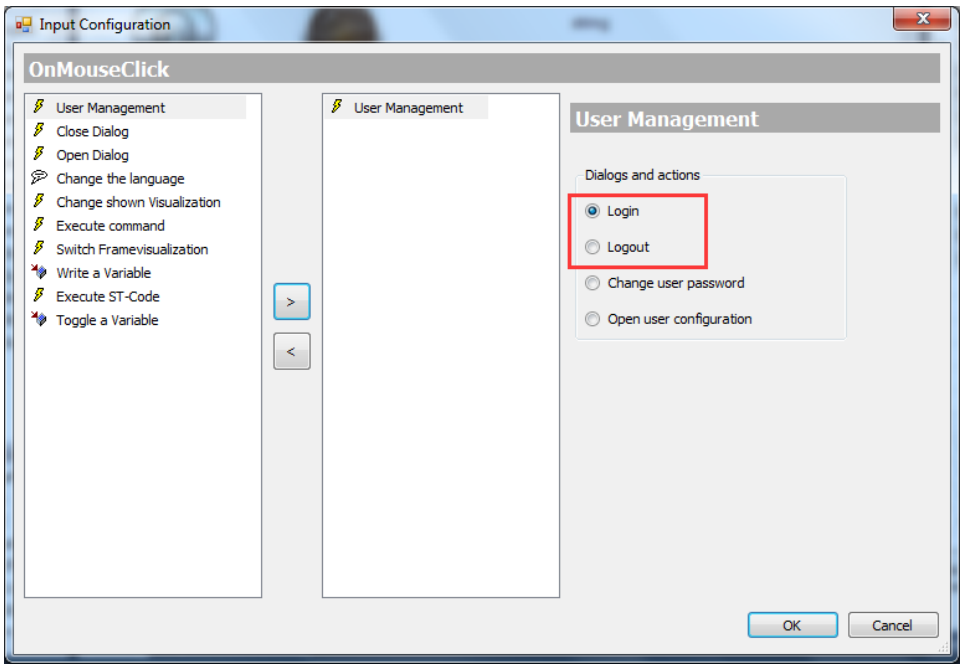

（4）在 F8_USERLEVEL HMI 画面中编写如下的画面



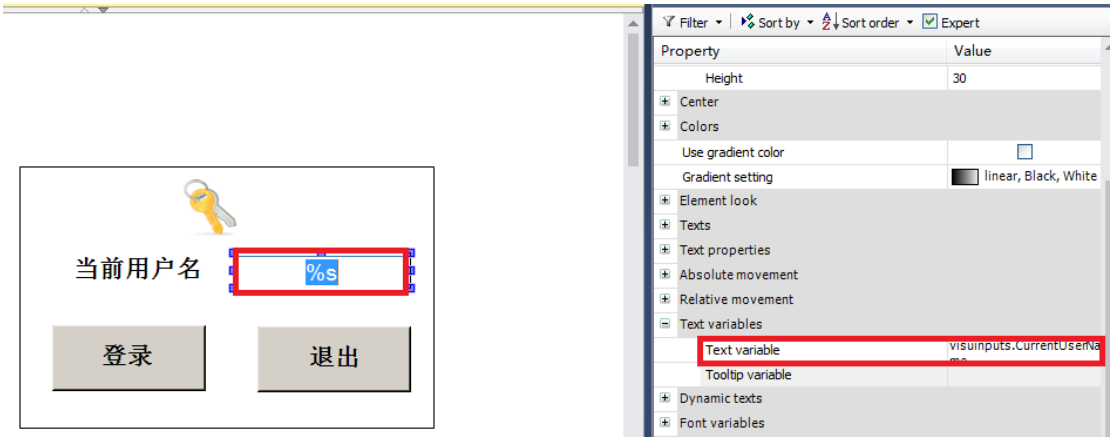

（5）分别配置这 2 个 Button 登录和退出，在属性界面里找到 inputconfiguration→OnMouseClick




（6）选择 User Managerment 后，分别把 2 个 Button 配置成 Login 和 Logout



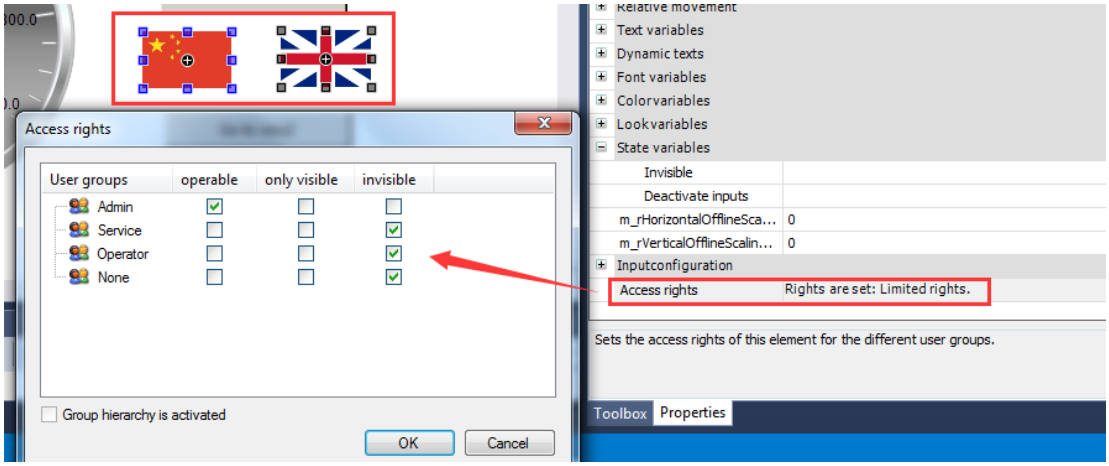

（7）配置显示控件，让其显示当前的用户名，text variables----visuinputs.CurrentUserName



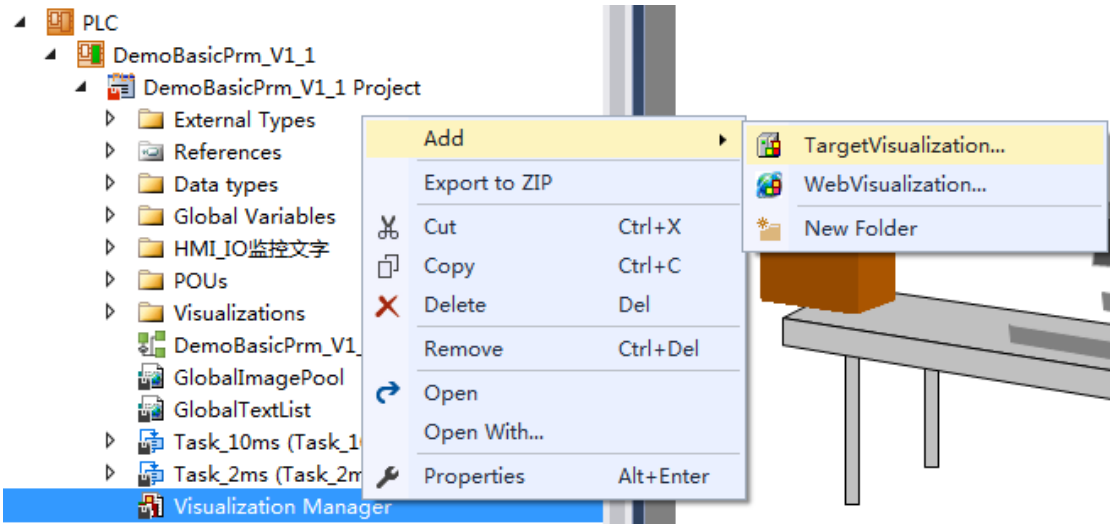

（8）接下来就可以针对不同控件配置不同的访问权限了，比如按钮控件，如果你希望此控件只有 Admin 用户可以操作，其他用户只是可见，那可以点击此控件，在属性栏中选择 Background→Access rights

| Property | Value |
| --- | --- |
| Elementname | GenElemInst_1 |
| Type of element | Dip switch |
| Position | |
| X | 42 |
| Y | 26 |
| Width | 70 |
| Height | 70 |
| Variable | MAIN.input |
| Image settings | |
| Element behavior | Image toggler |
| Texts | |
| Tooltip | |
| State variables | |
| Invisible | |
| Deactivate inputs | |
| Background | |
| Access rights | Not set. Full rights. |

（9）同时如果希望中英文切换只有 Admin 可以操作，其他用户都不可见，可以选中中英文切换的控件，勾选 Admin 的 operable，其他都勾选 invisible

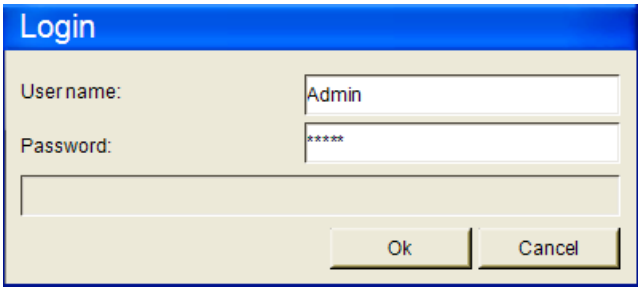



（10）选中 Visualization Manager 添加 TargetVisualization




（11）等全部配置好后，重新激活配置并且全屏显示，可以看到如果没有进行任何用户登录，中英文切换按钮不可见


（10） 此时点击 login 后输入用户名和密码，比如 Admin 和 1




（12）随后就可以看见中英文切换按钮可见并且可以操作


当前用户名
Admin
登录
退出

# 第八章：断电保持及配方功能

通过本章节的学习，学员将了解：

✓ 断电保持功能

✓ 配方功能

✓ 灵活调用配方功能模板

## 1、 配方概念

配方功能是把不同的数据通过触 HMI 批量写入到控制器的连续地址中，或者从控制器写入 HMI 中。

配方通俗的讲就是一组数据的集合，比如：你的生产线需要生产 500 种商品，那么输入这些商品的参数是一个很耗时的工作。把这些数据输入到配方中，那么当你更换需要生产的产品种类时，只需要更改配方号即可，不需要实时的输入数据，这样做的另外一个好处就是数据在现场对于操作人员来说不可见的，从而保护了您的数据安全。

配方就是一组数据的集合，那么任何寄存器或者说任何继续的存储单元都可以作为配方的使用来应用，在一些控制器控制系统中存储的容量有限，而对于掉电保持寄存器更有限，为了解决控制器的掉电存储器的容量有限，为了解决控制器掉电存储的有限继而出现了配方功能，在使用上配方较控制器的掉电保持更简单，所以得到广泛的应用。

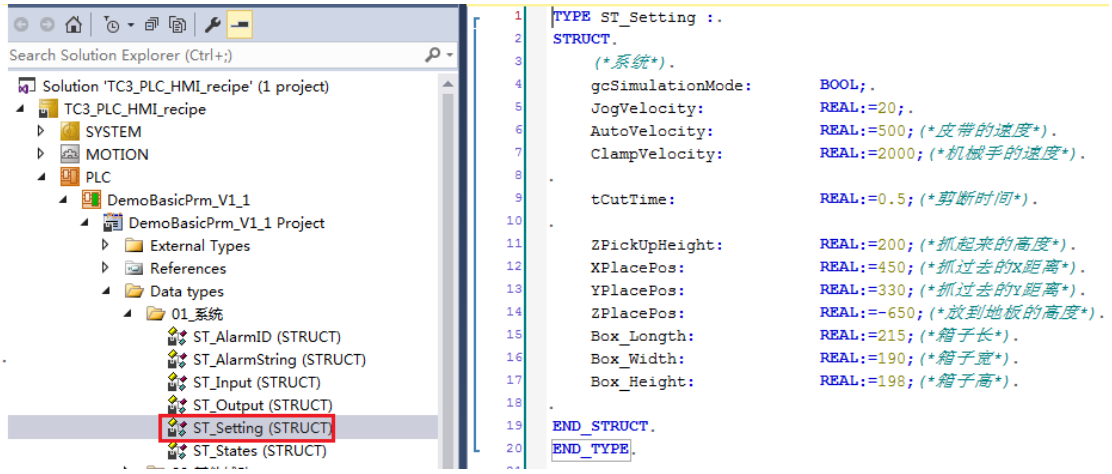
## 2、 断电保持

倍福的断电保持一般分为两种，一种是通过UPS实现断电保持（配置UPS功能的IPC,CX51**等），还有一种是通过NOV/RAM来实现断电保持（CX9020等）。本次的课程设计中的断电保持是通过UPS功能实现的，而且用到的硬件设备是配置UPS功能的IPC。

**设置断电保持型变量步骤：**

把需要断电保持的数据都全部封装在ST_Setting，并且在变量申明区设置为断电保持变量，设置好之后，通过UPS软件配置好UPS功能即可。

```
VAR_GLOBAL.
    DATA:                       ST_States;   (*状态*).
    gBox:                       ARRAY[0..23] OF ST_Box;(*箱子的状态*)
    CLAMP:                      APP_Clamp;.
    CONV:                       APP_Conv;.
.
    RqSysManualModeIsActive,.
    RqSysAutoModeIsActive,.
    RqSysGoHome:                BOOL;.
    RqSysMachineRunInCycle:     BOOL;.
    RqSysGoPauseState:          BOOL;.
END_VAR.
VAR_GLOBAL PERSISTENT(*断电保存型变量*).
    SET:                              ST_Setting;  (*当前设定值*).
END_VAR.
```



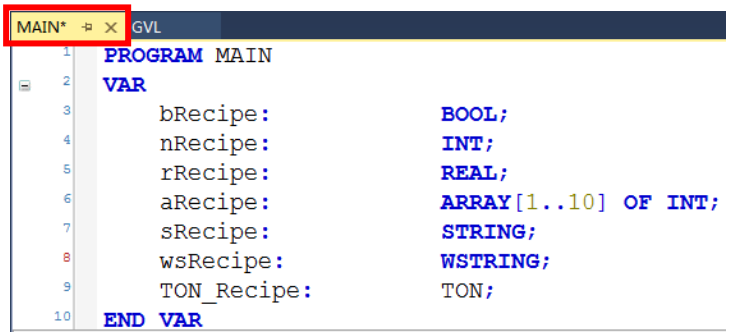

### 3、 配方功能

配方功能的模板我们已经写好，如果客户需要用到配方程序，直接调用即可。

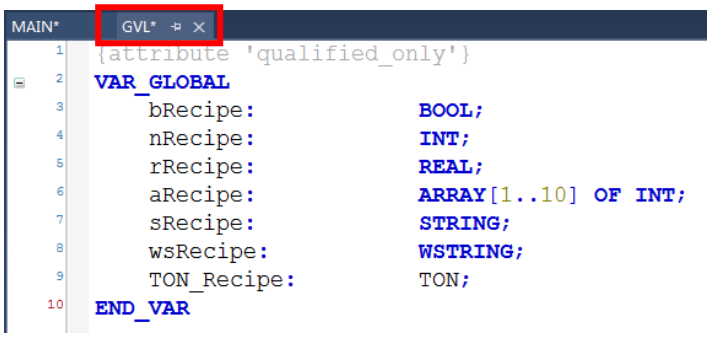
**配方配置步骤：**

假设我们定义如下全局变量：

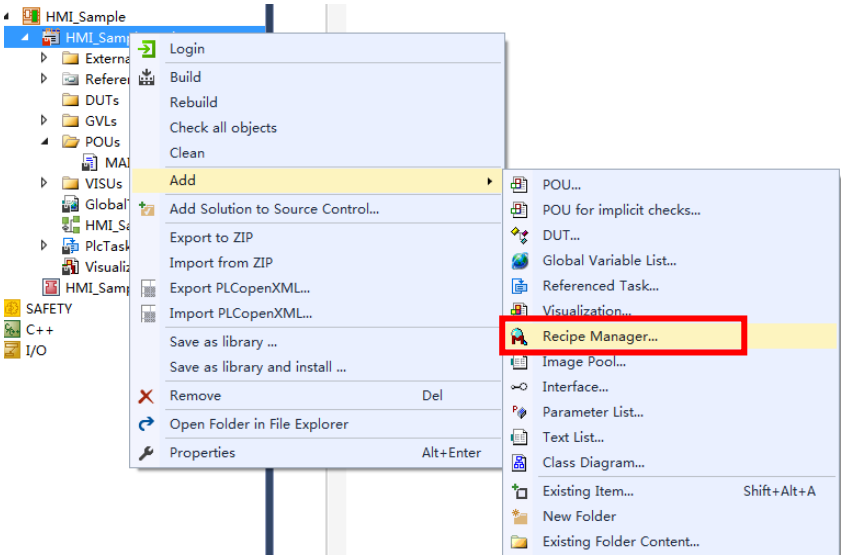
首先编写示例程序，为了测试 TwinCAT3 中的 Recipe 支持的数据类型，以及是否对变量的作用域有要求，我们分别创建多种数据类型的全局变量和局部变量，在 Main 程序中定义局部变量如下：



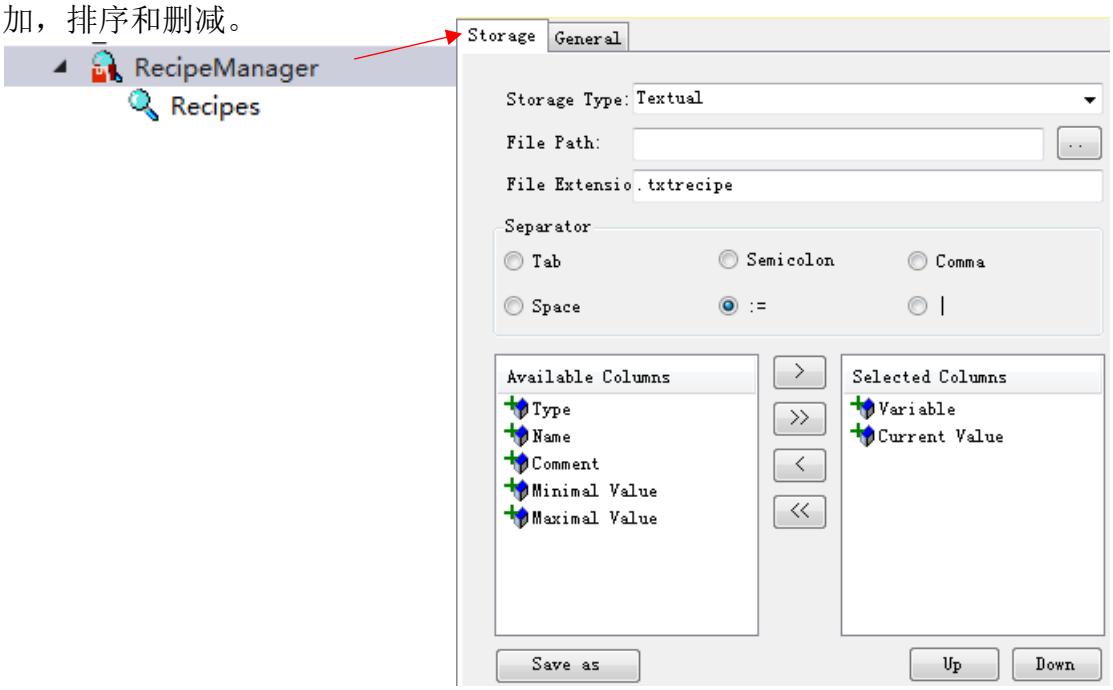

在 GVL 文件夹右键→Add→Global Variable List→OK，使用默认 GVL 创建，在新建的 GVL 中定义全局变量如下如下：



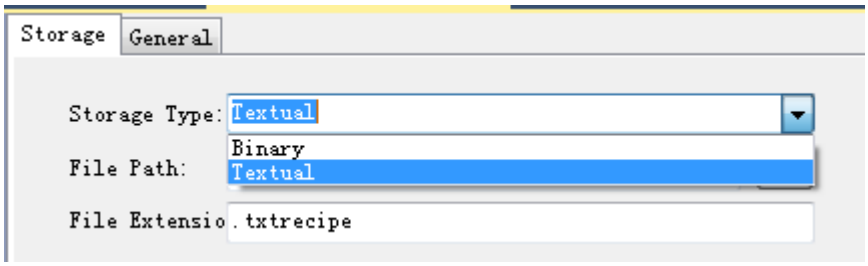

1） 创建 Recipe Manager，在 PLC 项目上面右键→Add→Recipe Manager，使用默认的名称，点击 open 创建



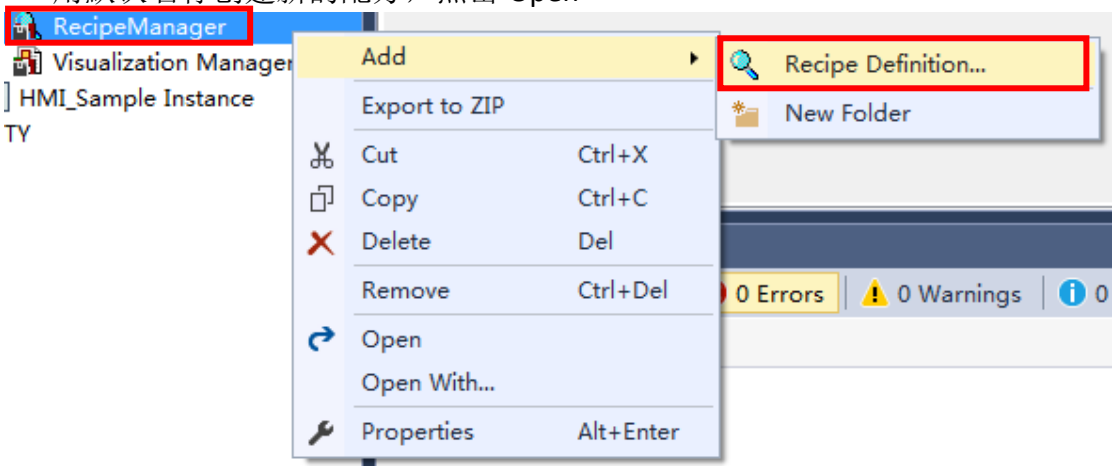

双击创建好的 Recipe Manager 在右边的配置窗口中可以对配方文件的保存格式、路径，文件扩展名进行设置，同时对于配方表格中需要显示的列进行添加，排序和删减。



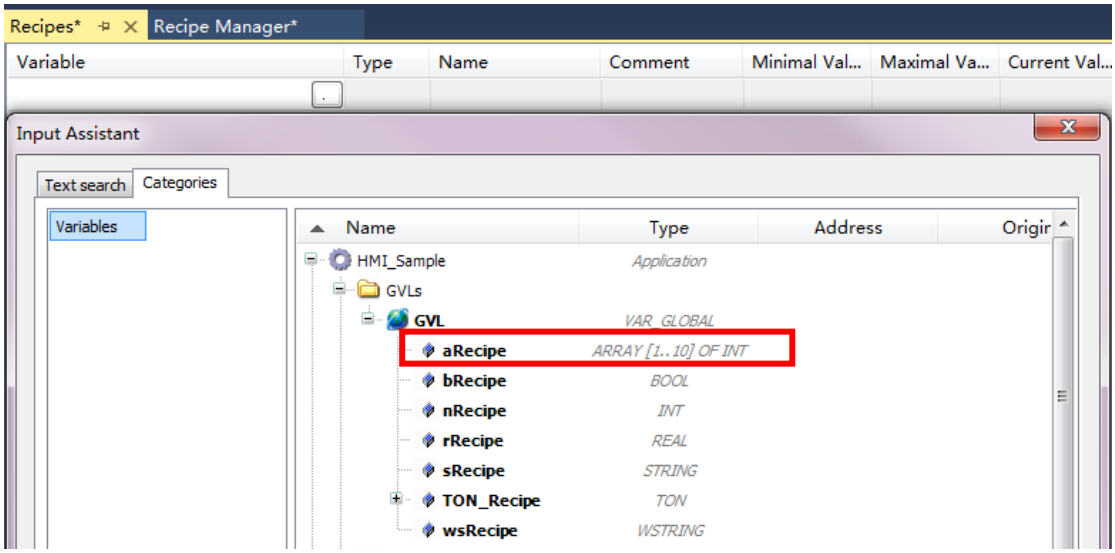

配方文件的保存格式有两种:使用默认的 txt 格式，方便以后对配方文件进行读取、修改等操作；使用 bin 格式，则无法使用其他软件读取、修改保存的配方文件。




2） 配置完成后，在新建的 Recipe Manager 上右键→Add→Recipe Definition…使用默认名称创建新的配方，点击 Open




3） 选择新建的 Recipes，在右边的配置窗口中添加配方中所包含的变量。双击表格中 Variable 一列下单元格，通过 F2 或者调用程序中 [...] 变量，选择全局变量中的 aRecipe。




添加完成后，可以在 Recipes 看到全局变量中添加进来的数组变量；根据配方功能继续添加局部变量，由图可见，局部变量也可以添加，功能块，结构体等数据类型都可以添加到配方中，非常方便。

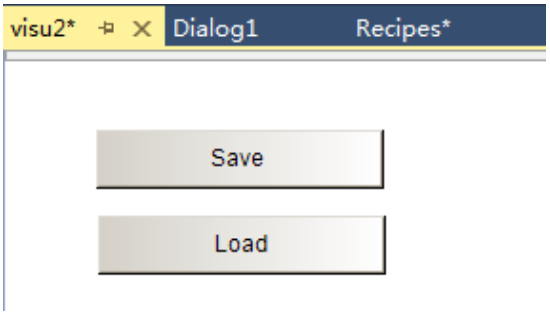
Recipes* Recipe Manager*

| Variable | Type |
|---|---|
| GVL.aRecipe[1] | INT |
| GVL.aRecipe[2] | INT |
| GVL.aRecipe[3] | INT |
| GVL.aRecipe[4] | INT |
| GVL.aRecipe[5] | INT |
| GVL.aRecipe[6] | INT |
| GVL.aRecipe[7] | INT |
| GVL.aRecipe[8] | INT |
| GVL.aRecipe[9] | INT |
| GVL.aRecipe[10] | INT |

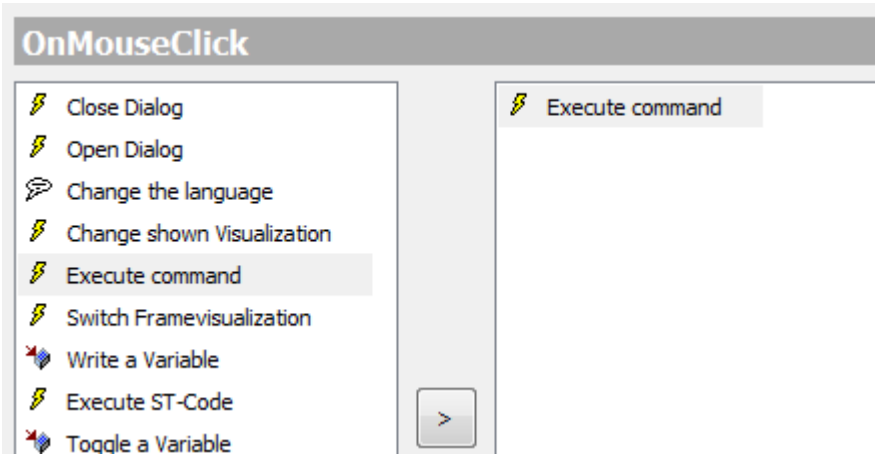
| | |
|---|---|
| MAIN.bRecipe | BOOL |
| MAIN.nRecipe | INT |
| MAIN.rRecipe | REAL |
| MAIN.sRecipe | STRING |
| MAIN.TON_Recipe.IN | BOOL |
| MAIN.TON_Recipe.PT | TIME |
| MAIN.TON_Recipe.Q | BOOL |
| MAIN.TON_Recipe.ET | TIME |
| MAIN.TON_Recipe.M | BOOL |
| MAIN.TON_Recipe.StartTime | TIME |
| MAIN.wsRecipe | WSTRING |

4）完成变量的添加后，创建新的 PLC HMI

5）画面 visu2，在 Toolbox 中找到 Common Controls→Button，添加两个 Button，一个实现 Recipe 的创建和保存，一个实现 Recipe 的调用，选中 Button，在右边属性选项卡中找到 Texts，分别输入 Save 和 Load

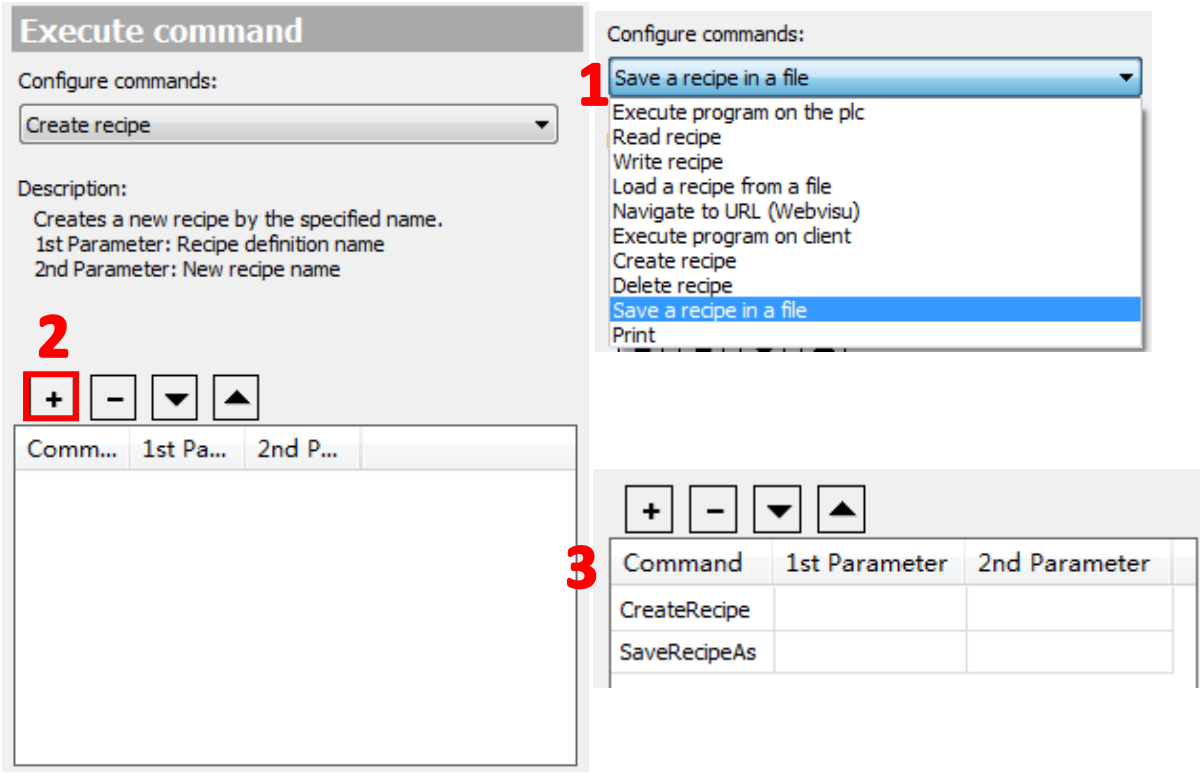



6）选中 Save 按钮在属性菜单中找到 Inputconfiguration→OnMouseClick，单击 Configure…对 HMI 中的 Recipe 功能进行配置，在弹出对话框左侧双击添加 Execute command

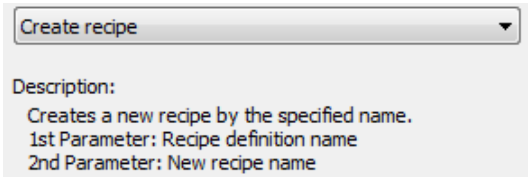



选中 Execute command 在弹出窗口的右边 configure command 下拉选项中选择 Creat recipe，点击下方的 + 添加这条指令，同样的添加 Save a Recipe in a file

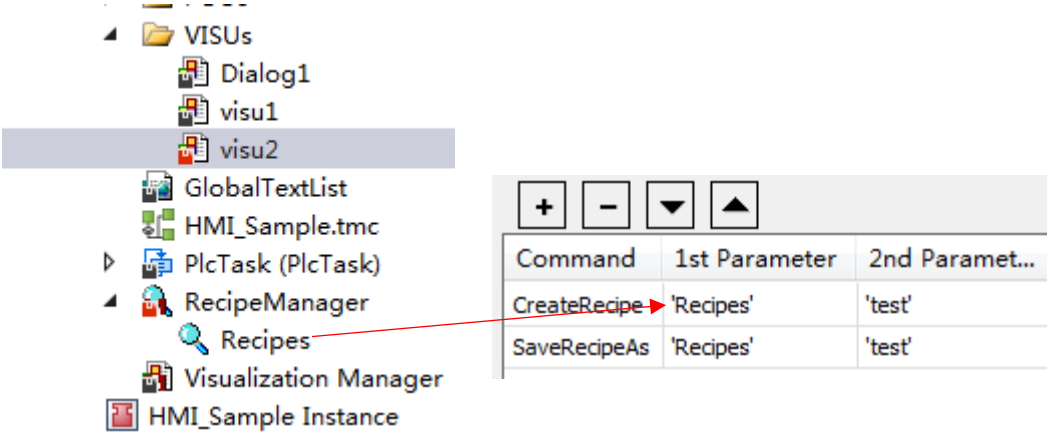
这条指令



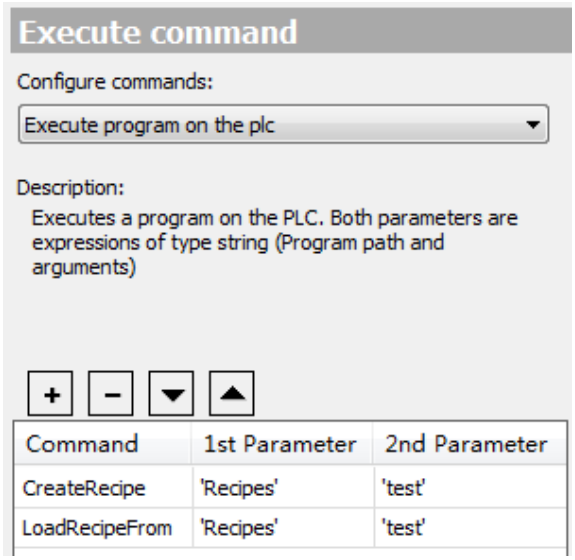

添加完成指令后，需要在 1st Parameter 和 2nd Parameter 中分别填写 Recipe Definition 和 Recipe 的名字，在添加指令的时候同样可以看到相关注释说明



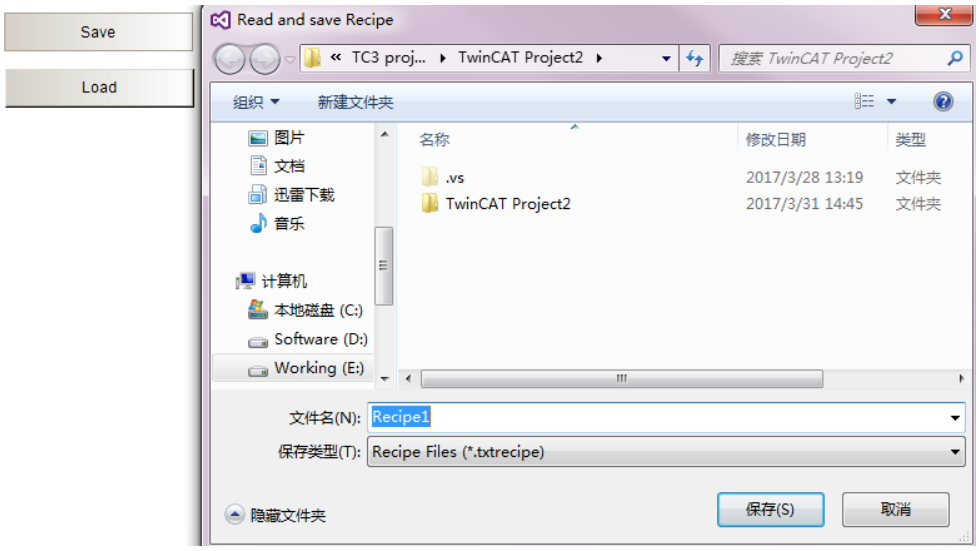

填入之前添加的 Recipe Definition 的名称’Recipes’以及自定义的配方名称’test’，记得在为填写的名称加上单引号



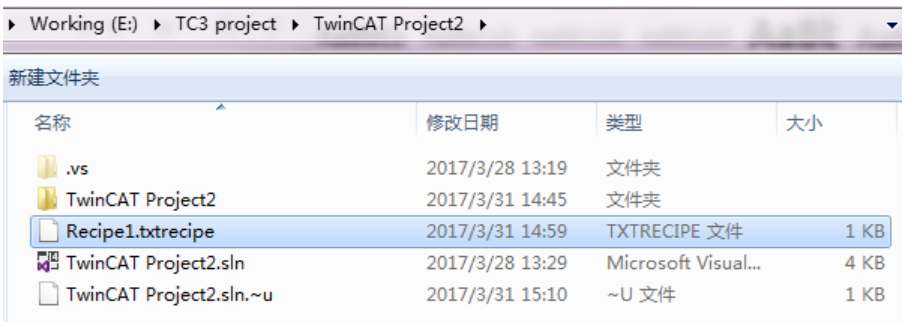

同样的，对 Load 按钮进行类似配置，添加 Creat Recipe 和 Load a Recipe from a file 两个指令，并完成配置，点击 OK 应用



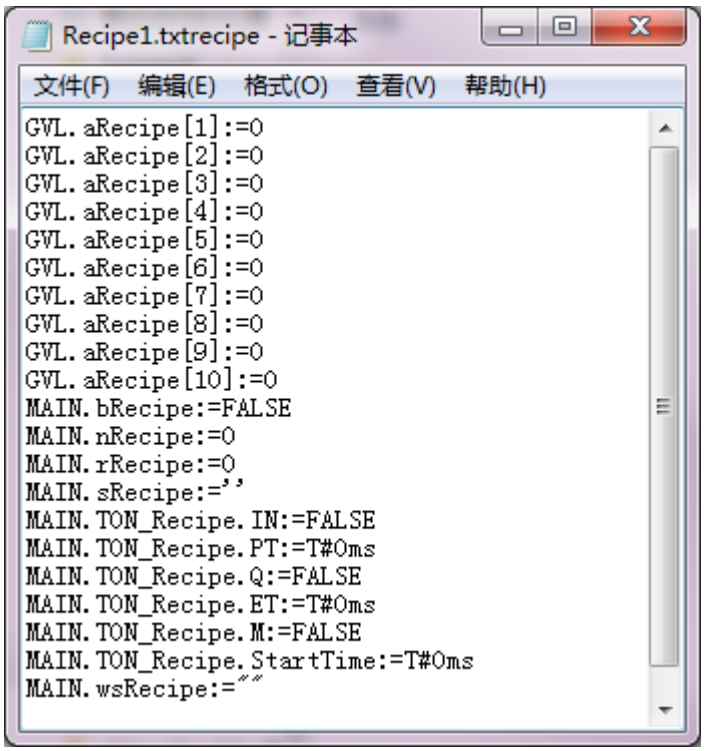

7） 激活并运行程序，点击画面中的 Save 按钮，可以看到画面中会弹出配方的保存对话框，选择保存路径，在文件名中填入 Recipe1 后点击保存



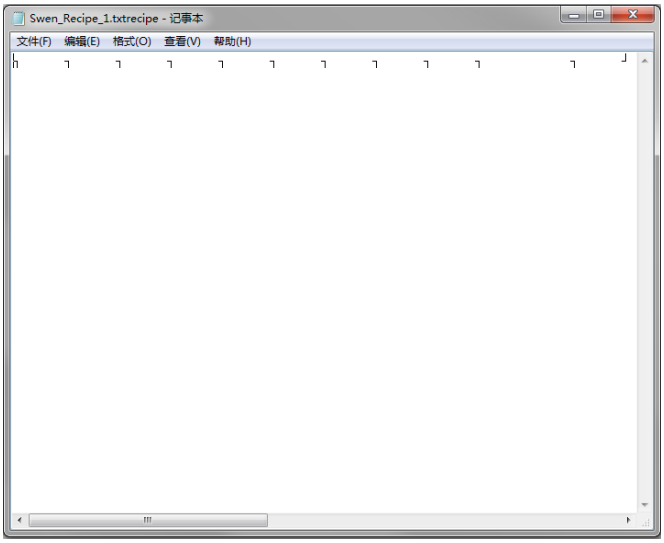

可以得到配方保存的文件：




这个文件可以用记事本打开，并且可以查看里面的数据

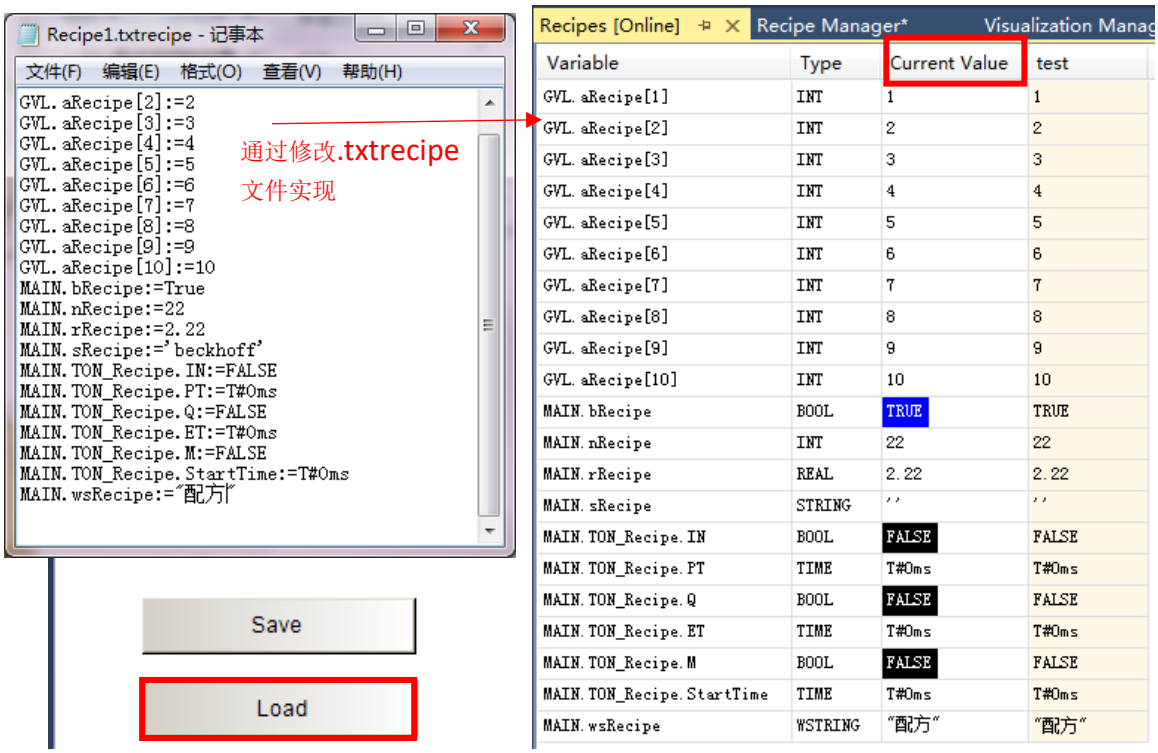



如果在选择保存文件格式的时候选择 bin 的方式，则不能查看配方中的数据，显示为乱码，如下：

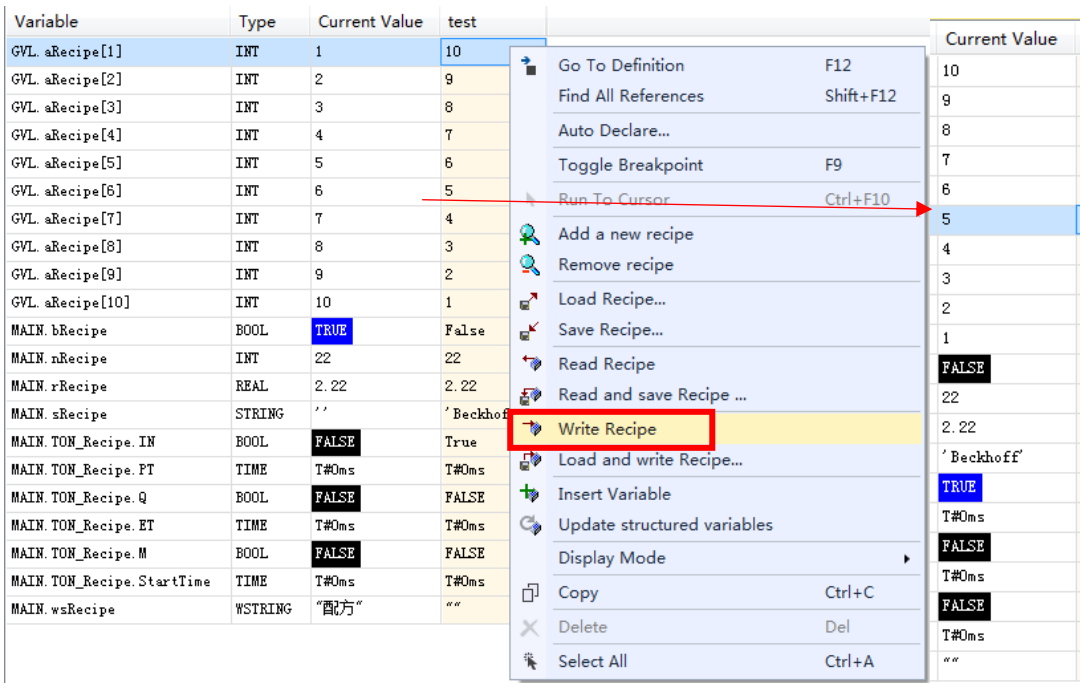



8）如果需要对配方进行更改有两种方式，通过修改保存的.txtrecipe 文件，或者直接在 Recipe Definition 中进行

首先第一种，通过记事本修改.txtrecipe 后缀的文件，点击 HMI 界面中的 Load 按钮上载配方，就可以在 Recipe Manager 下的 Recipes 中对当前配方进行查看，其中 Current Value 是在线值




第二种方法就是直接在 RecipeManager 下的 Recipes 中进行，Recipe Definition 不仅可以监控当前配方的值，也可以在这个界面中直接对配方相关参数进行修改




## 4. 配方模板介绍

**配方模板程序介绍：**

FB_FileCopy:文件的复制

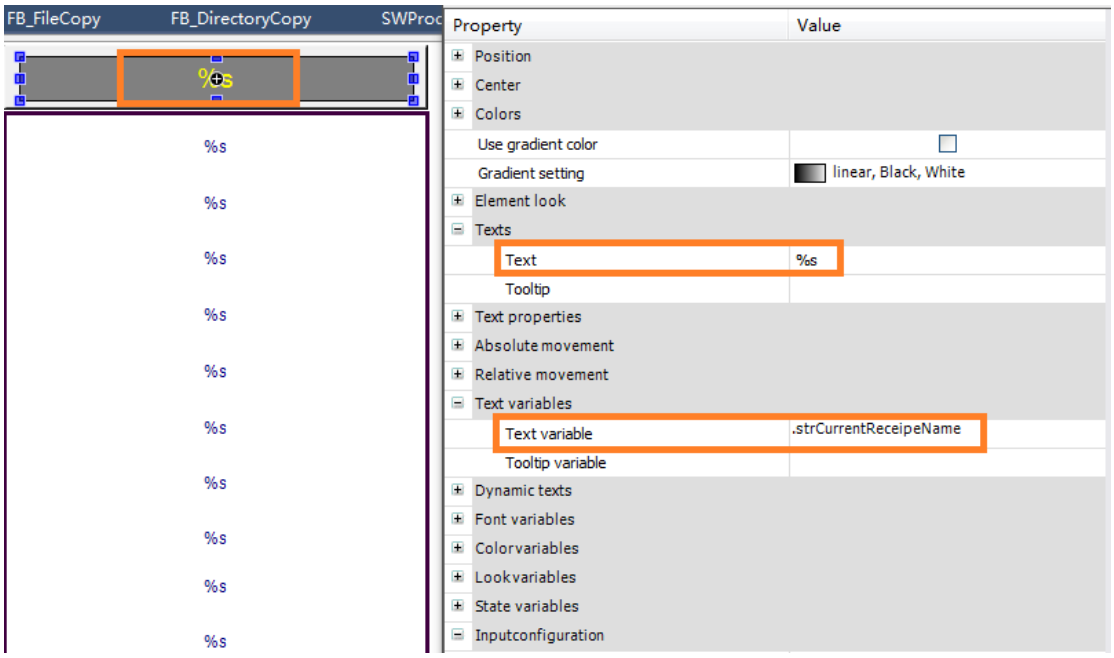
FB_DirectoryCopy:该功能块调用了 FB_FileCopy，主要实现文件查找及复制

SWProduct 子程序里面包括了 9 个 Action,每个 Action 里面包括了一种功能。
SWProduct.ButtonDelet:删除配方文件
SWProduct.ButtonLoad:读取配方文件
SWProduct.ButtonLoadFromUSB:从 USB 中读取配方文件
SWProduct.ButtonSave:保存配方文件
SWProduct.ButtonSaveToUSB:保存配方文件到 USB
SWProduct.HMISelect:配方文件的选择
SWProduct.Init:初始化
SWProduct.ReflashFileList:刷新列表

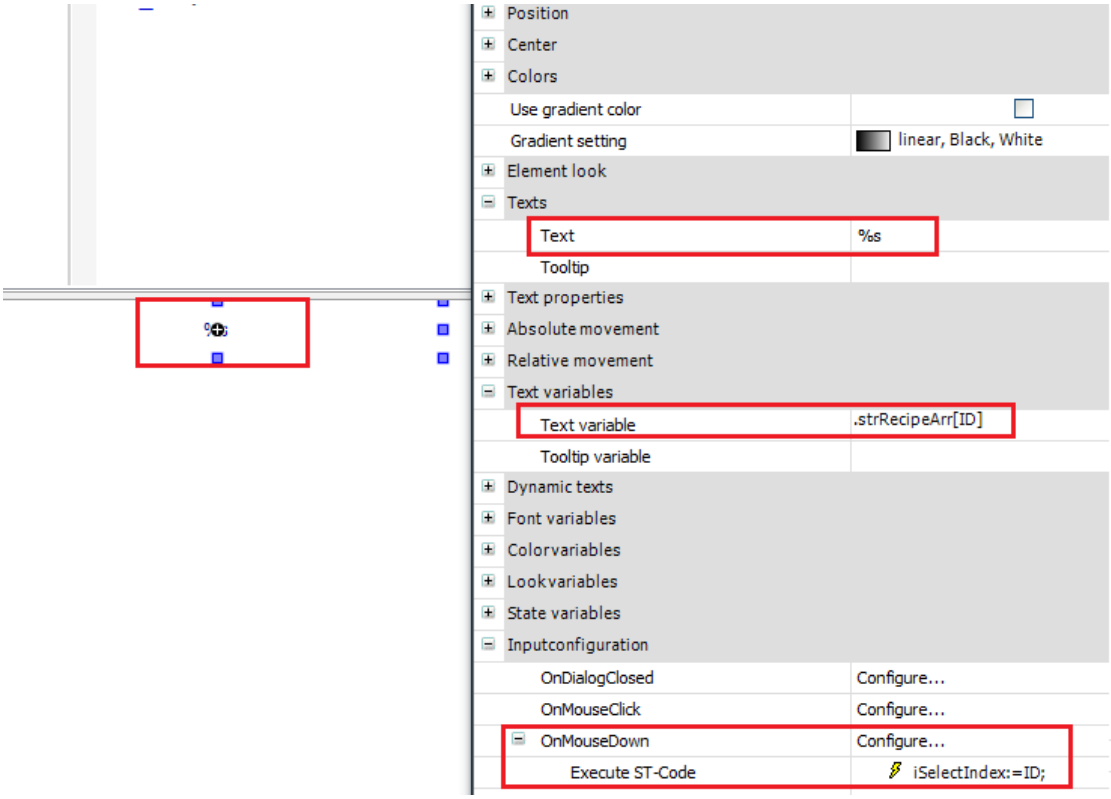
**配方模板的 HMI 画面介绍：**

配方模板总共有 6 个 Button,每个 button 其实和 SWProduct 的子程序的 Action 一一对应。
读取控件：SWProduct.ButtonLoad
保存控件：SWProduct.ButtonSave
删除控件：SWProduct.ButtonDelet
从 USB 读取控件：SWProduct.ButtonLoadFromUSB
下载到 USB 控件：SWProduct.ButtonSaveToUSB
刷新列表控件：SWProduct.ReflashFileList

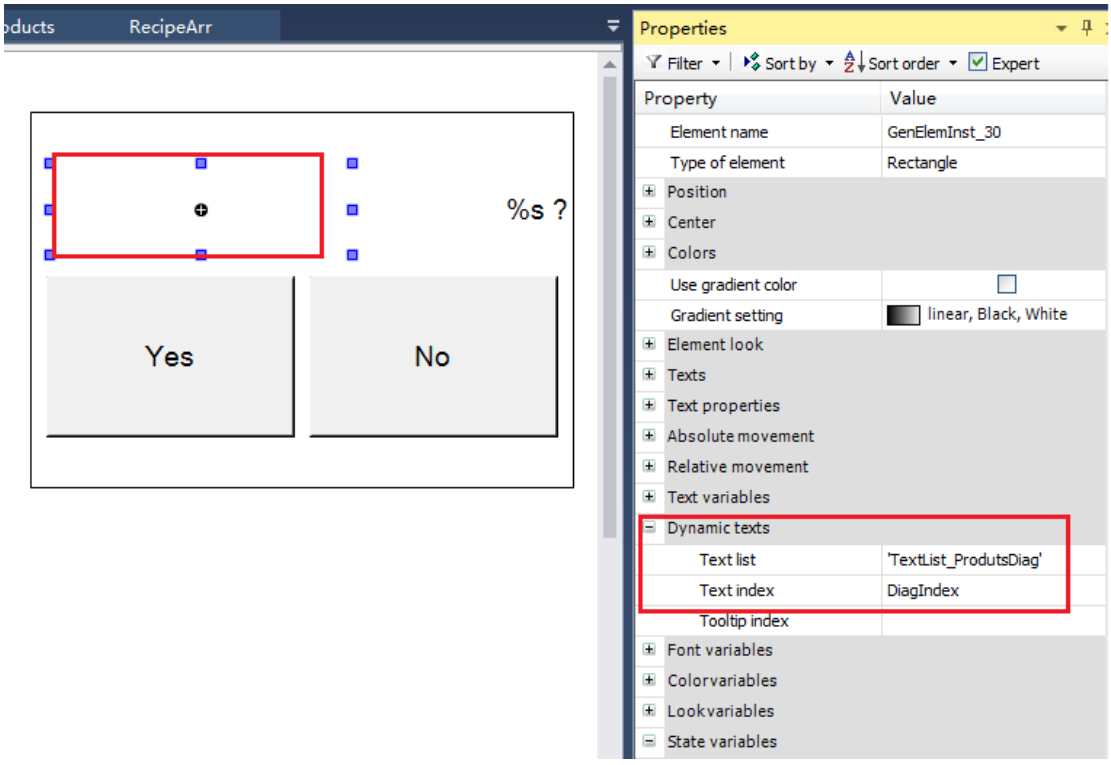
显示当前配方文件的控件设置：



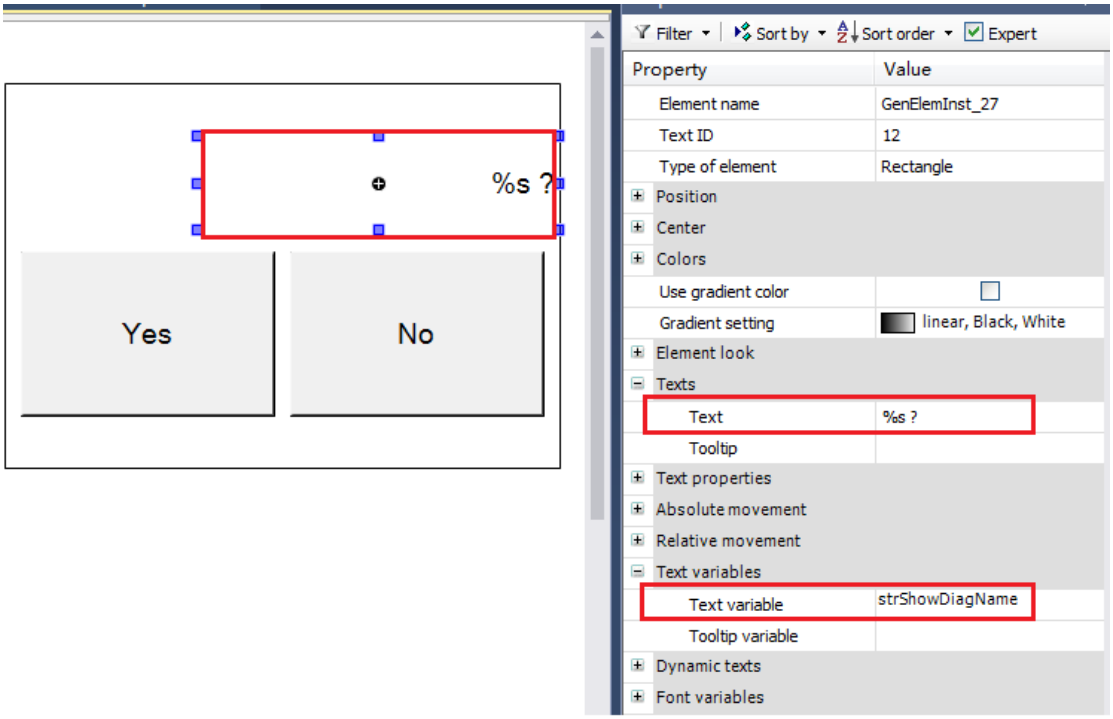

显示配方文件的控件设置：
该配方模板中可以显示 12 个配方文件，通过点击相应的配方文件，就可以实现删除，读取等功能。这个文件配方的显示控件是通过 Frame 控件调用 RecipeArr 来实现的，RecipeArr 具体设置如下：



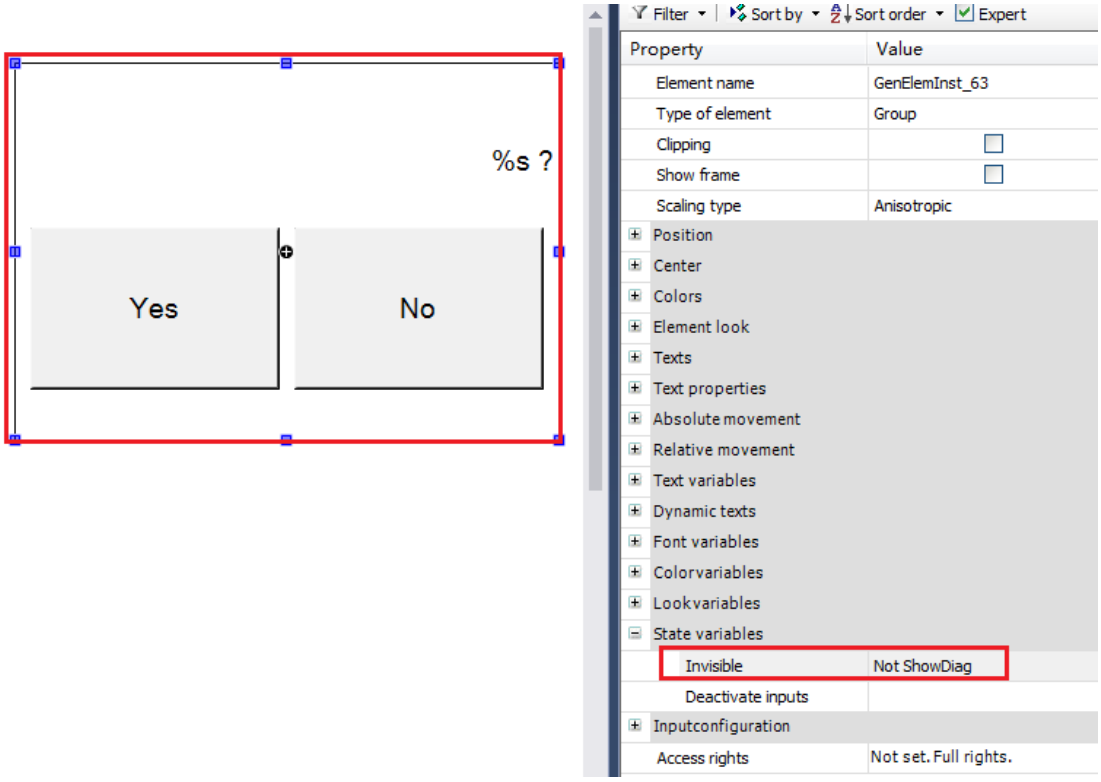

配方的弹出画面设置：
弹出画面总共包括 5 个控件组合而来的。
第一个控件：
通过 Rectangle 控件实现 TextList 显示



第二个控件：
通过 Rectangle 控件显示要处理的配方文件名




整体弹出界面的设置：
把五个控件组成一组，用控件显示来实现弹屏



# 第九章：事件记录

通过本章节的学习，学员将了解：

✓ 事件记录概念
✓ 灵活调用事件记录的模板
✓ 事件记录的文件保存功能

## 1、 事件/报警概念

当设备出现运行故障的时候，设备能通知到工作人员，从而及时处理。通过查看报警产生的历史记录，从而更加清楚的了解设备的运行情况，往往不同的工作情况，需要各种各样报警，所以报警成为了工业设备的必备条件。

事件代表正常的系统状态消息。通常事件在发生某种系统条件时触发，如操作员登录到倍福应用系统程序中，操作员是不必确认事件。

报警是用来指示控制系统中出现的事件或操作状态，可以用报警信息对系统进行诊断。有的资料或手册将报警消息简称为信息、消息或报文。

报警有两种形式，一种是自定义报警，另一种是系统报警。自定义报警是用户组态的报警，用来在 HMI 设备上显示过程状态，或者测量和报告从 PLC 接收到的过程数据。系统报警用来显示 HMI 设备或 PLC 中特定的系统状态，系统报警是在这些设备中预定义的。自定义报警和系统报警都可以由 HMI 设备或 PLC 来触发，在 HMI 设备上显示。

事件（或者报警）优先级（EventClass）：

可以给事件指定一个优先级或者说是严重度。例如，锅炉温度超出极限时，要求发出高优先级的报警，需要引起立即关注。报警的优先级通常取决于环境，工厂应用，设备性质、安全型等。

## 2、 事件记录模板程序介绍

1） 数据类型介绍：

Alm_EventClass：报警的优先级

```
TYPE Alm_EventClass :.
(.
    NoAlarm                         :=0,(*无警报*).
    Message_1                       :=1,(*信息*).
    Event_2                         :=2,(*事件*).
    non_3                           :=3,(*暂无*).
    non_4                           :=4,(*暂无*).
    non_5                           :=5,(*暂无*).
    Alarm_6                         :=6,(*不停机警报*).
    Stop_7                          :=7(*停机警报*).
);.
END_TYPE.
```

Alm_AlmMsg：报警信息，该报警信息包括中英文

```
TYPE Alm_AlmMsg :.
STRUCT.
    CN:     WSTRING:="中文";.
    EN:     STRING:='En';    .
END_STRUCT.
END_TYPE.
```

Alm_AlarmType：当前报警种类

```
TYPE Alm_AlarmType :.
STRUCT.
      (*InPut*).
    EventId:                    INT;.
    EventMessage:           WSTRING;.
    EventTime:                    DT;.
    EventActive:                BOOL;.
    QuitRequired:               BOOL;.
    QuitEvent:              BOOL;.
    EventClass:             Alm_EventClass;.
    AlarmMsg:               Alm_AlmMsg;.
    AlarmlampOn:            BOOL;(*报警灯*).
    AutoRunOff:             BOOL;(*退出自动循环*).
    (* event actions OutPut *).
    AlarmAction1:               BOOL;.
    AlarmAction2:               BOOL;.
    AlarmAction3:               BOOL;.
    AlarmAction4:               BOOL;.
    AlarmAction5:               BOOL;.
    AlarmAction6:               BOOL;.
    AlarmAction7:               BOOL;.
    AlarmAction8:               BOOL;.
.
       (*OutPut*).
    EventAlive:                 BOOL;.
    ShowlampOn:                 BOOL;.
    InUse:                      BOOL;.
END_STRUCT.
END_TYPE;.
```

Alm_AlarmHistoryType：历史报警种类

```
TYPE Alm_AlarmHistoryType :.
STRUCT.
    EventMessage:                WSTRING;.
    EventId:                     INT;.
    EventTime:                   DT;.
END_STRUCT.
END_TYPE.
```

2) 事件登录模板的功能块介绍：

FB_AlarmMain 功能块用于显示报警信息，里面包括了三个 Action,每个 Action 的具体功能如下：

FB_AlarmMain. AlarmAlive:用于显示当前报警列表

FB_AlarmMain. AlarmHMI:清除历史报警

FB_AlarmMain.AlarmShow：所有报警信息交替显示

FB_SetAlarm 功能块用于设置报警以及显示历史报警

FB_SetAlarm 的输入管脚的介绍：

| | | | | |
|---|---|---|---|---|
| VAR_INPUT | **InUse** | BOOL | | 是否使用 |
| VAR_INPUT | **EventId** | INT | | 报警号 |
| VAR_INPUT | **EventMessa...** | STRING | | 报警信息 |
| VAR_INPUT | **EventActive** | BOOL | | 触发信号 |
| VAR_INPUT | **FilterInUse** | BOOL | FALSE | 是否使用滤波器*)(*对触发信号进行滤波，就是触发信号超过多少时间后才报警 |
| VAR_INPUT | **FilterTime** | TIME | | 滤波时间 |
| VAR_INPUT | **QuitRequired** | BOOL | | 是否自动复位：true=复位信号为1时复位报警信号;false=触发信号为0时复位报警信号 |
| VAR_INPUT | **QuitEvent** | BOOL | | 复位信号 |
| VAR_INPUT | **EventClass** | Alm_Eve... | | 报警等级 |
| VAR_INPUT | **ToTheLogFile** | BOOL | | 是否记录到历史记录文件 |
| VAR_INPUT | **ToTheHistory** | BOOL | | 是否记录到报警历史记录 |
| VAR_INPUT | **AlarmlampOn** | BOOL | | 是否需要报警灯亮 |
| VAR_INPUT | **AutoRunOff** | BOOL | | 退出自动循环 |
| VAR_INPUT | **AlarmAction1** | BOOL | | 用户定义报警动作*)(*例如定义为黄灯亮,则该报警号有触发时,主报警程序的该输出信号为1 |
| VAR_INPUT | **AlarmAction2** | BOOL | | 用户定义报警动作 |
| VAR_INPUT | **AlarmAction3** | BOOL | | 用户定义报警动作 |
| VAR_INPUT | **AlarmAction4** | BOOL | | 用户定义报警动作 |
| VAR_INPUT | **AlarmAction5** | BOOL | | 用户定义报警动作 |
| VAR_INPUT | **AlarmAction6** | BOOL | | 用户定义报警动作 |
| VAR_INPUT | **AlarmAction7** | BOOL | | 用户定义报警动作 |
| VAR_INPUT | **AlarmAction8** | BOOL | | 用户定义报警动作 |

FB_SetAlarm 的输出管脚的介绍：

| | | | |
|---|---|---|---|
| VAR_OUTPUT | **EventAlive** | BOOL | 当前报警正在执行 |
| VAR_OUTPUT | **ShowlampOn** | BOOL | 当前报警灯亮 |

SW_FormatWstring 功能块实现如下功能：

把中文的字符串“ETHERCAT 总线错误,数量%D,在第%D 个,名称为%S”中的%D，%D 和%S 替代掉

FB_FileCopy：文件复制

FB_DirectoryCopy：调用 FB_FileCopy 实现文件复制功能

FB_DiagDataLogging_S 包括了 4 个 Action,每个 Action 的具体功能如下：

FB_DiagDataLogging_S. act_AutoDelete:自动删除旧文件
FB_DiagDataLogging_S. act_AutoWrite：自动写入
FB_DiagDataLogging_S. act_CheckCom：命令处理
FB_DiagDataLogging_S. act_Copy2USB：复制到其他文件夹

### 3、 模板调用

1) 定义报警 ID 号及其中英文显示的信息

```
AlarmActionArray[1].AlarmMsg.CN :=  "ETHERCAT总线错误,数量%D,在第%D个,名称为%S";.
AlarmActionArray[2].AlarmMsg.CN :=  "急停按钮!";.
AlarmActionArray[3].AlarmMsg.CN :=  "安全门开";.
AlarmActionArray[4].AlarmMsg.CN :=  "报警4";.
AlarmActionArray[5].AlarmMsg.CN :=  "报警5";.
AlarmActionArray[6].AlarmMsg.CN :=  "报警6";.
AlarmActionArray[7].AlarmMsg.CN :=  "报警7";.
AlarmActionArray[8].AlarmMsg.CN :=  "报警8";.
AlarmActionArray[9].AlarmMsg.CN :=  "报警9";.

AlarmActionArray[1].AlarmMsg.EN :=  'Alarm1';.
AlarmActionArray[2].AlarmMsg.EN :=  'Alarm2';.
AlarmActionArray[3].AlarmMsg.EN :=  'Alarm3';.
AlarmActionArray[4].AlarmMsg.EN :=  'Alarm4';.
AlarmActionArray[5].AlarmMsg.EN :=  'Alarm5';.
AlarmActionArray[6].AlarmMsg.EN :=  'Alarm6';.
AlarmActionArray[7].AlarmMsg.EN :=  'Alarm7';.
AlarmActionArray[8].AlarmMsg.EN :=  'Alarm8';.
AlarmActionArray[9].AlarmMsg.EN :=  'Alarm9';.

.
.
FOR i:=cnAlarmFirst TO cnAlarmLast DO.
    fbSetAlarm[i].EventId := i;.
END_FOR .
bInidDone:=TRUE;.
.
```

2) 在 MAIN_Alarm. Act_OtherAlarm 里面编写报警的触发条件

```
fbSetAlarm[1](.
    InUse:=          TRUE, .
    QuitRequired:=   TRUE, .
    QuitEvent:=      Alm_bBtReset, .
    ToTheLogFile:=   TRUE,.
    ToTheHistory:=   TRUE, .
    D1:=             D1,.
    D2:=             D2,.
    S1:=             ModleName,.
    );.
    .
fbSetAlarm[2](.
    InUse:=          TRUE, .
    QuitRequired:=   TRUE, .
    QuitEvent:=      Alm_bBtReset, .
    ToTheLogFile:=   TRUE,.
    ToTheHistory:=   TRUE, .
    );.
```

功能块 FB_SetAlarm 的输入引脚 ToTheLogFile 为 TRUE 的时候，则记录到全局变量 gsWriteToLog，则写入到文件。当引脚 ToTheLogFile 为空的时候，则不记录到文件。当需要定制化修改报警显示的内容时，可以使用新的引脚：D1,D2,S1(需要注意大小写)

例如：

我们定义了报警的中文内容如下：

AlarmActionArray[1].AlarmMsg.CN:="ETHERCAT总线错误,数量%D,在第%D个,名称为%S";

当我们产生了总线报警的时候，我们不单单需要给出报警，还有提示用户是哪个模块发生了错误导致的总线错误。这时的报警就是定制化的可修改报警了。

在“ETHERCAT 总线错误,数量%D,在第%D 个,名称为%S”这个报警内容中，包含了字符两个“%D”和一个“%S”。通过报警功能块的新引脚 D1,D2,S1,把报警内容的字符替换成引脚内容，实现定制化报警的功能。

例如:

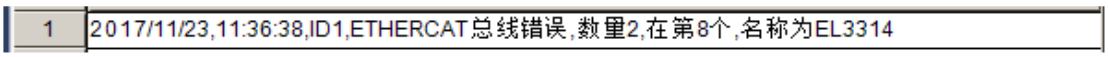
报警定义：ETHERCAT总线错误,数量%D,在第%D个,名称为%S(其中%D ,%S需要注意为大写)

D1:=2

D2:=8

S1:=’EL3314’

那么,报警输出显示为：THERCAT 总线错误,数量 2,在第 8 个,名称为 EL3314




上面实现了功能块增加定制字符的功能。

其实，如果需要更彻底的修改显示内容，只需要每次触发报警之前，改变一

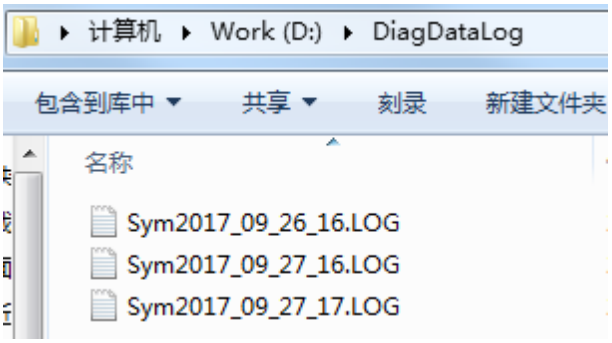
下 AlarmActionArray[AlarmID].AlarmMsg.CN 的内容就可以了。其中报警功能块在全局变量里面已经定义了，不需要重复定义。

MAIN_Alarm:主程序中调用 fbAlarmMAIN 功能块实现 HMI 画面报警显示，调用 FB_DiagDataLogging_S 功能块实现报警的历史记录文件。

3) 事件的历史记录文件

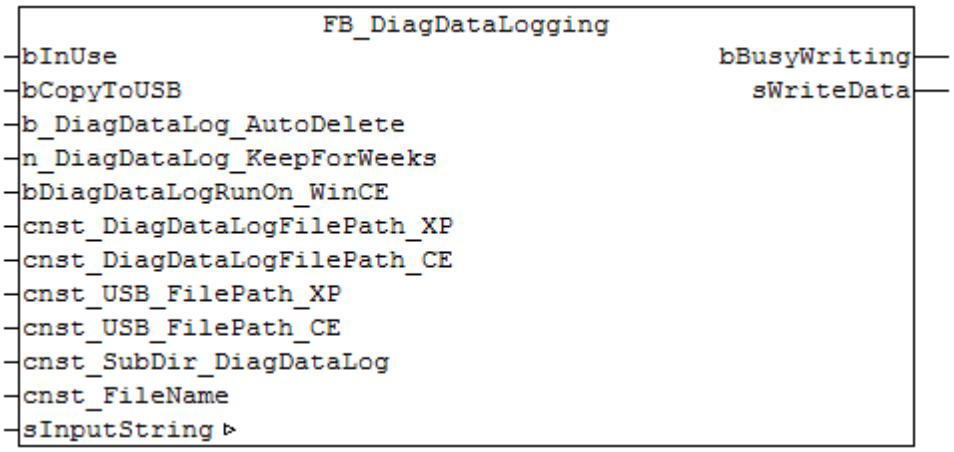
例如，在控制器里面保存历史报警的LOG文档。工程师开发方便的把文档拷贝出来做分析。如果工程师不在车间现场，客户也可以把该文件夹发送给工程师，帮助寻找并解决问题。

如下图，历史记录文件，以“年月日时”为名称。也就是说，每小时产生一个新的文件，以防止文件过大。



。

后台记录功能块的使用




提供了功能块FB_DiagDataLogging作为数据处理的后台程序。

例如在例子程序里面的“MAIN_Alarm”程序里面就定义并调用了该功能块。同时在全局变量里面，定义了string类型的”gsWriteToLog”变量。并作为输入输出类型，链接到了FB_DiagDataLogging的sInputString引脚。

工作流程原理：

1：检测到有报警，则把报警需要记录的字符串赋值给”gsWriteToLog”变量

2：FB_DiagDataLogging 检测到”gsWriteToLog”变量不为零，则记录下需要写的字符串。记录值放到[1..10] OF STRING 的中间变量内。做中间存储变量的目的，是当文件没有写完的时候，又有报警记录进来，则可以缓存一下。

3：FB_DiagDataLogging把”gsWriteToLog”变量置零，等待其它报警的触发。

4：FB_DiagDataLogging检测到[1..10] OF STRING的中间变量不为零，则开始写文件。

5：写文件结束。

TC3事件记录文件注意点：

1：TC3能用中文字符，但是请先转换成**UTF8**的格式。可以使用“string_to_utf8”功能。

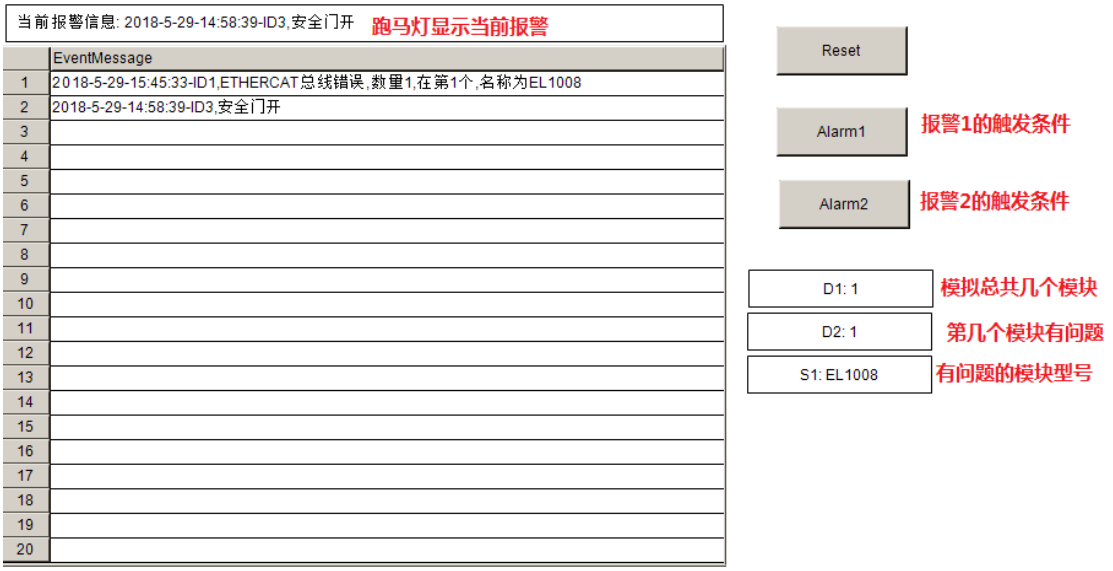
2：当同一个PLC周期有几个报警同时触发时，只能记录到一个最后刷新数据。

解决方法：把几个报警的字符串合并成一个字符串。例如：

gsWriteToLog:= CONCAT(gsWriteToLog, sNewLog);

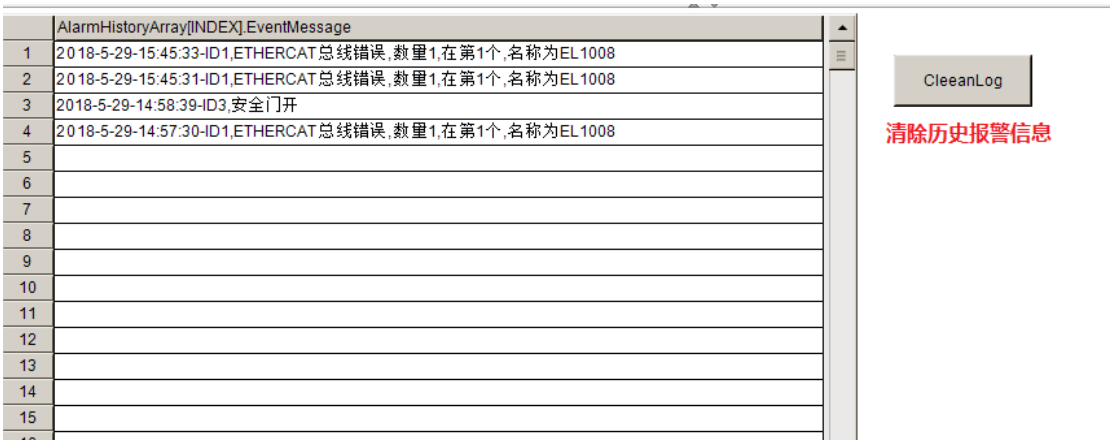
## 4、事件模板 HMI 画面介绍

当前报警显示：




历史报警显示：

历史事件显示会把所有的事件信息显示出来



# 第十章:系统诊断

通过本章节的学习，学员将了解：

- ✓ EtherCAT 诊断的模板
- ✓ 控制器本身诊断的功能块
- ✓ 灵活调用系统诊断模板

倍福的控制器的的系统诊断常见的有两种，一种是我们控制器本身的诊断，比如是温度，使用率和网卡信息等，还有一种是总线信息（EtherCAT）的诊断。

**1、 系统诊断模板 POU 介绍**

MDP_CPU:读取 CPU 的温度，频率和使用率。

MDP_Fan:读取风扇的速度

MDP_Mainboard:读取主板信息，具体信息如下图：

| SubIndex | Type | Name | Type | Access |
|---|---|---|---|---|
| 00 | VAR | Len | UNSIGNED16 | read-only |
| 01 | VAR | Mainboard Type | VISIBLE STRING | read-only |
| 02 | VAR | Serial Number | VISIBLE STRING | read-only |
| 03 | VAR | Production Date | VISIBLE STRING | read-only |
| 04 | VAR | Boot Count | UNSIGNED32 | read-only |
| 05 | VAR | Operating Time in Minutes | UNSIGNED32 | read-only |
| 06 | VAR | Min Board Temperature (°C)[1] | SIGNED32 | read-only |
| 07 | VAR | Max Board Temperature (°C)[1] | SIGNED32 | read-only |
| 08 | VAR | Min Input Voltage (mV)[1] | SIGNED32 | read-only |
| 09 | VAR | Max Input Voltage (mV)[1] | SIGNED32 | read-only |
| 10 | VAR | Mainboard Temperature (°C)[1] | SIGNED16 | read-only |

MDP_RaidSimple:读取 RAID 的信息

MDP_SiliconDrives:读取驱动的信息，具体信息如下图：

| SubIndex | Type | Name | Type |
|---|---|---|---|
| 00 | VAR | Len | UNSIGNED8 |
| 01 | VAR | Total EraseCounts | UNSIGNED64 |
| 02 | VAR | Drive Usage (%) | UNSIGNED16 |
| 03 | VAR | Number of Spares | UNSIGNED16 |
| 04 | VAR | Spares Used | UNSIGNED16 |

MDP_UPS：读取 UPS 状态，具体信息如下图：

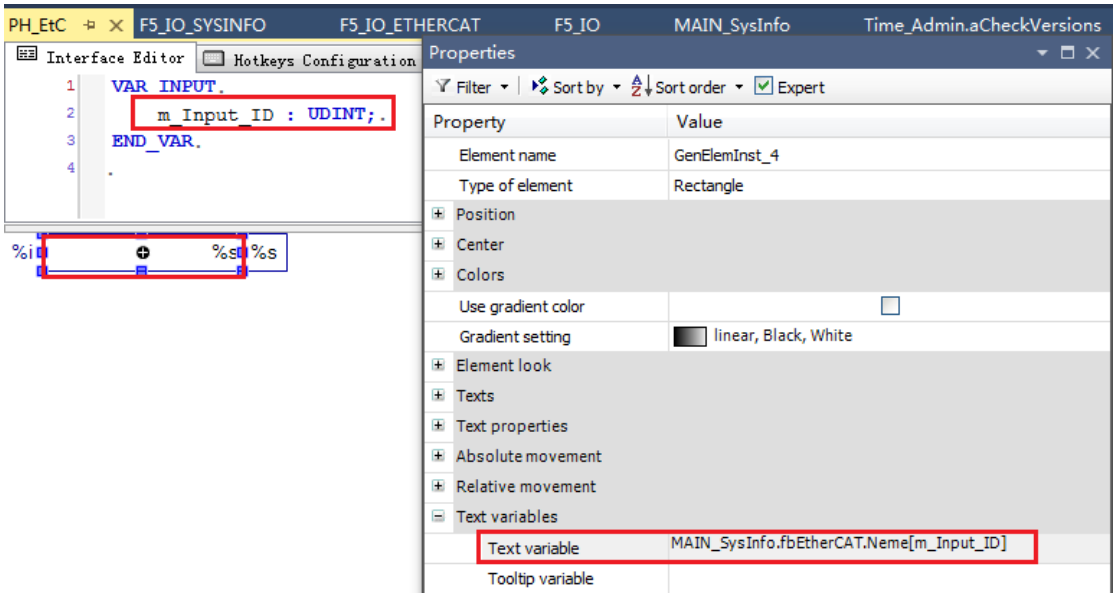
**0x8nn1 - UPS Information**

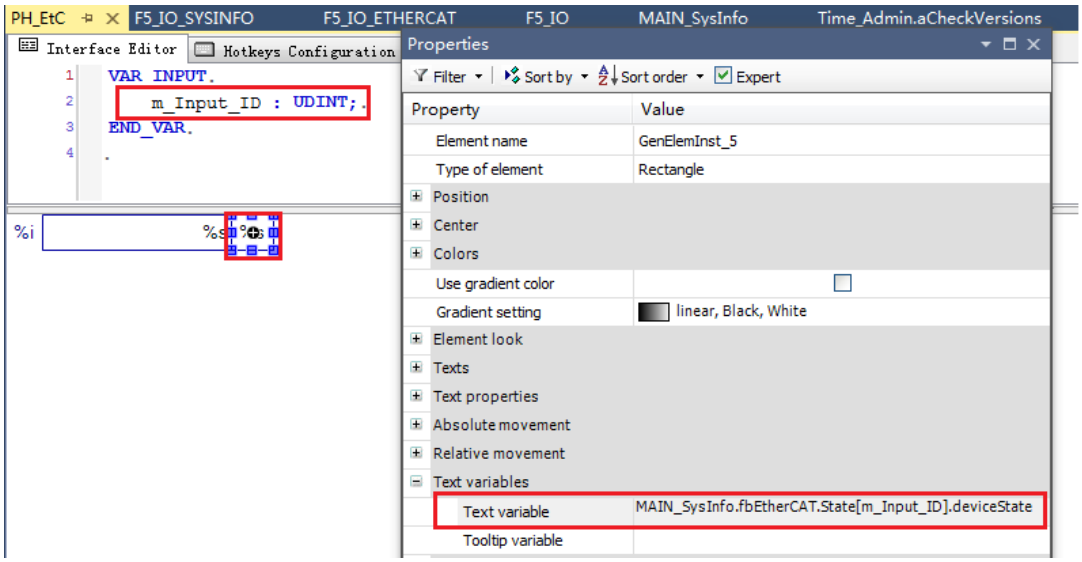
| SubIndex | Type | Name | Type | Access |
|---|---|---|---|---|
| 00 | VAR | Len | UNSIGNED16 | read-only |
| 01 | VAR | UPS Model | VISIBLE STRING | read-only |
| 02 | VAR | Vendor Name | VISIBLE STRING | read-only |
| 03 | VAR | Version | UNSIGNED8 | read-only |
| 04 | VAR | Revision | UNSIGNED8 | read-only |
| 05 | VAR | Build | UNSIGNED16 | read-only |
| 06 | VAR | Serial Number | VISIBLE STRING | read-only |
| 07 | VAR | Power Status | UNSIGNED8 | read-only |
| 08 | VAR | Communication Status | UNSIGNED8 | read-only |
| 09 | VAR | Battery Status | UNSIGNED8 | read-only |
| 10 | VAR | Battery Capacity (in %) | UNSIGNED8 | read-only |
| 11 | VAR | Battery Runtime (in seconds) | UNSIGNED32 | read-only |
| 12 | VAR | Persistent Power Fail Count | BOOLEAN | read-only |
| 13 | VAR | Power Fail Counter | UNSIGNED32 | read-only |
| 14 | VAR | Fan Error | BOOLEAN | read-only |
| 15 | VAR | No Battery | BOOLEAN | read-only |
| 16 | VAR | Test Capacity | BOOLEAN | write-only |
| 17 | VAR | Battery Replace Date | VISIBLE STRING | read-only |
| 18 | VAR | Interval Service Status | BOOLEAN | read-only |

Time_Admin：获取版本信息及修改时间的功能
FB_GetNGSlaverName_v2：读取所有 EtherCAT 从站的名字，地址和状态信息
FB_GetEtherCatAlarm：当从站报错之后，程序里面进行复位，切到 OP 模式

## 2、 系统诊断模板 HMI 的设置

HMI 设置的难点就是 EtherCAT 诊断的画面，其中 EtherCAT 诊断调用了 PH_EtC 模板。
PH_EtC 设置如下：

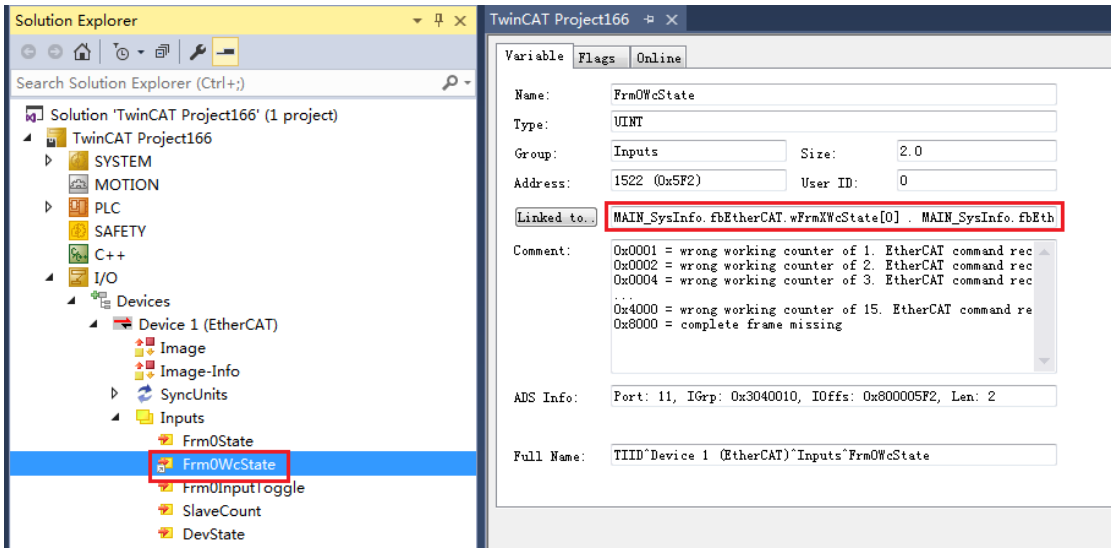



### 3、 调用系统诊断模板

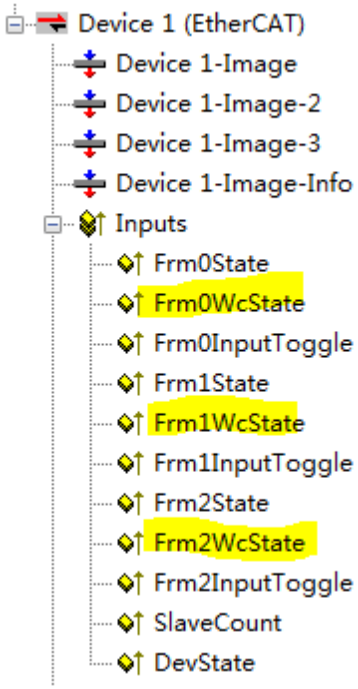
主程序 Main_SysInfo 里面主要调用了 MDP_CPU 功能块，获取 CPU 信息；调用了 MDP_MAINBOARD，获取主板信息；调用了 FB_GetAdaptersInfo，获取网卡信息；调用 FB_GetEtherCatAlarm 获取 EtherCAT 从站的名字，地址和状态信息以及报错复位。**EtherCAT** 总线诊断硬件链接，一共有 3 处：

1）WcState

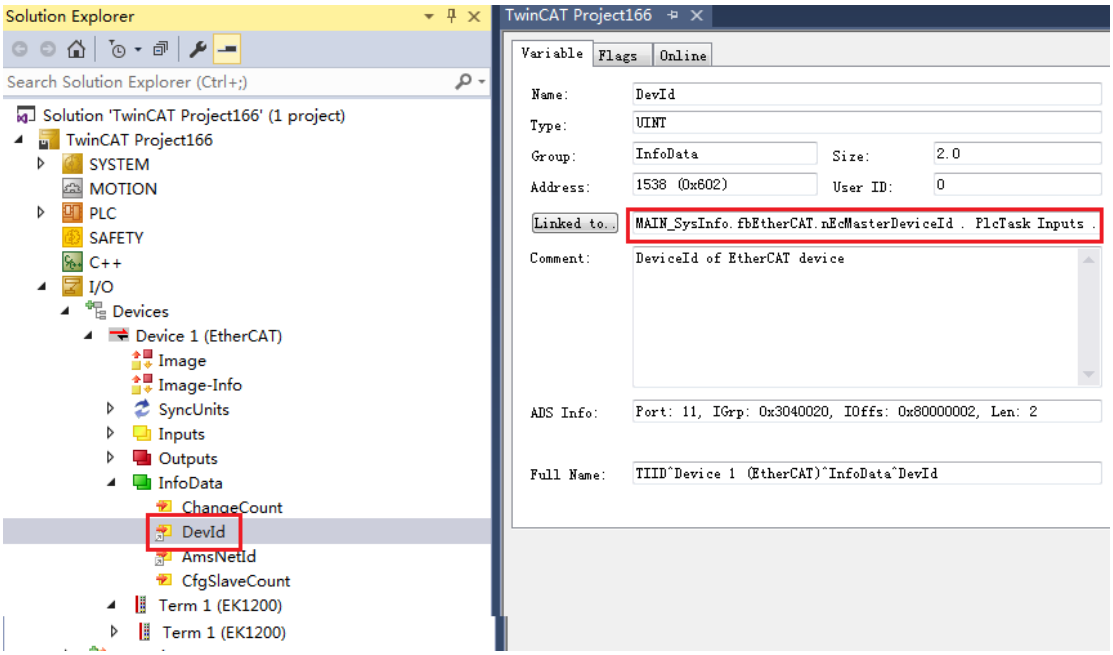



其中，有几个”FrmXXwcState”，就需要链接几个变量。如下图，最多有 4 个。

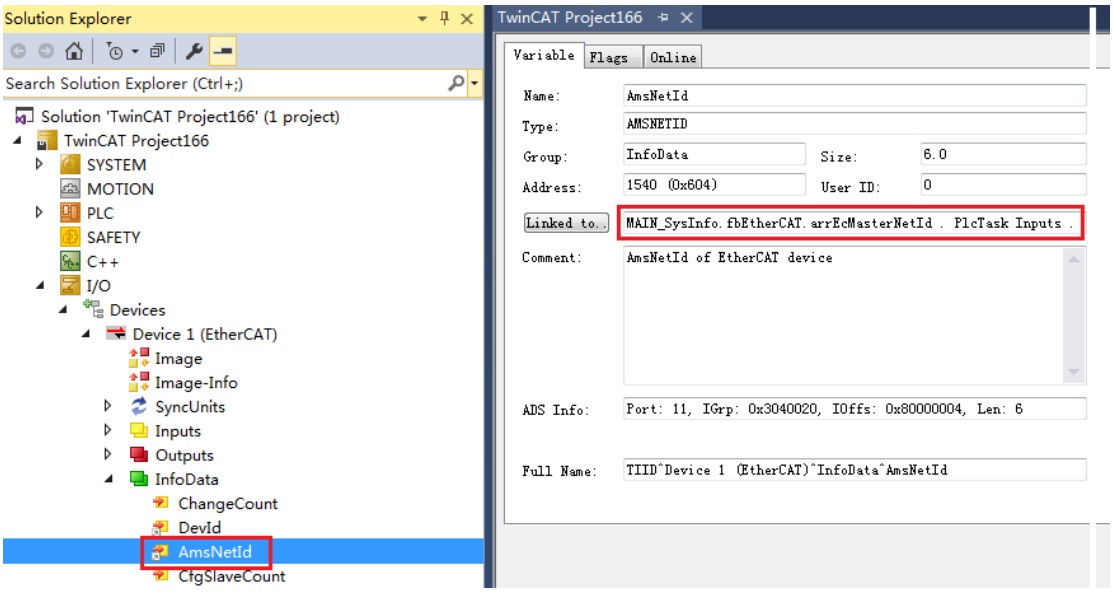

WcState 的数量根据 PLC 程序的不同，会有所不同，最多 4 个。例如下面的 EtherCAT 总线，就有 3 个 WcState，需要链接 3 个。




2) DevId


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'TwinCAT Project166' (1 project)
TwinCAT Project166
SYSTEM
MOTION
PLC
SAFETY
C++
I/O
Devices
Device 1 (EtherCAT)
Image
Image-Info
SyncUnits
Inputs
Outputs
InfoData
ChangeCount
DevId
AmsNetId
CfgSlaveCount
Term 1 (EK1200)
Term 1 (EK1200)
TwinCAT Project166
Variable
Flags
Online
Name: DevId
Type: UINT
Group: InfoData
Size: 2.0
Address: 1538 (0x602)
User ID: 0
Linked to...
MAIN_SysInfo.fbEtherCAT.nEcMasterDeviceId . PlcTask Inputs .
Comment: DeviceId of EtherCAT device
ADS Info: Port: 11, IGrp: 0x3040020, IOffs: 0x80000002, Len: 2
Full Name: TIID^Device 1 (EtherCAT)^InfoData^DevId

3) NetId


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'TwinCAT Project166' (1 project)
TwinCAT Project166
SYSTEM
MOTION
PLC
SAFETY
C++
I/O
Devices
Device 1 (EtherCAT)
Image
Image-Info
SyncUnits
Inputs
Outputs
InfoData
ChangeCount
DevId
AmsNetId
CfgSlaveCount
TwinCAT Project166
Variable
Flags
Online
Name: AmsNetId
Type: AMSNETID
Group: InfoData
Size: 6.0
Address: 1540 (0x604)
User ID: 0
Linked to...
MAIN_SysInfo.fbEtherCAT.arrEcMasterNetId . PlcTask Inputs .
Comment: AmsNetId of EtherCAT device
ADS Info: Port: 11, IGrp: 0x3040020, IOffs: 0x80000004, Len: 6
Full Name: TIID^Device 1 (EtherCAT)^InfoData^AmsNetId

# 终章:结束语

衡量一台设备好坏的指标，编者认为有四个方面：

功能性、强壮性、便利性、智能化。这些指标是逐渐递增的。

功能性：

完成指定的工艺，达到产能要求。这是设备的最基本要求。

强壮性

强壮性，也就是鲁棒性、稳定性。鲁棒是Robust的音译，也就是健壮和强壮的意思。它是在异常和危险情况下系统生存的关键。比如说，计算机软件在输入错误、磁盘故障、网络过载或有意攻击情况下，能否不死机、不崩溃，就是该软件的鲁棒性。所谓“鲁棒性”，是指控制系统在一定（结构，大小）的参数摄动下，维持其它某些性能的特性。

在设备控制领域，强壮性就是对外部、内部扰动等不确定因素的对应处理程序的完善。比如：空压气切断、急停按钮按下、真空失败、断膜、缺料、漏料、伺服故障、安全门打开、传感器故障等等，这些各种影响工艺流程正常执行的干扰因素。

编者认为，强壮性是设备控制最关键的地方。我们编写程序，应该花最多的时间去保证系统强壮性。只有设备足够强壮、稳定，才能减少设备停机的时间。不仅保证了生产的正常执行，也大大减少了维护成本。也节省了开发工程师排查故障的精力。

同时，详细周全的报警提示、监控，有助于设备故障的排查。也是后期维护的重要工具和手段。也是可以投入大量精力去完善的项目。

便利性：

也就是方便性、舒适性、人性化。

智能化：

智能制造、中国制造2025、工业4.0

简单地，我们拿一台汽车做比喻：

功能性就是汽车能开会跑。

强壮性就是基本不用保养或保养少。

便利性就是导航、空调等享受设施。

智能化就是自动驾驶、无人驾驶。

对比看出，虽然智能化是最高目标，但强壮型才是设备制造之生存根本。

**上海（ 中国区总部）**
中国上海市静安区汶水路 299 弄 9号（市北智汇园）
电话：021-66312666　　传真：021-66315696　　邮编：200072

**北京分公司**
北京市西城区新街口北大街 3 号新街高和大厦 407 室
电话：010-82200036　　传真：010-82200039　　邮编：100035

**广州分公司**
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**
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

请用微信扫描二维码
通过公众号与技术支持交流

倍福中文官网：
http://www.beckhoff.com.cn/
倍福虚拟学院：
http://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