



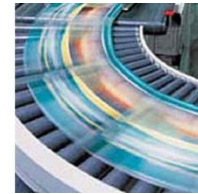
POU Types



CoDeSys



We software Automation.



POU Types

How can I structure my project?

Introduction

After this module you will

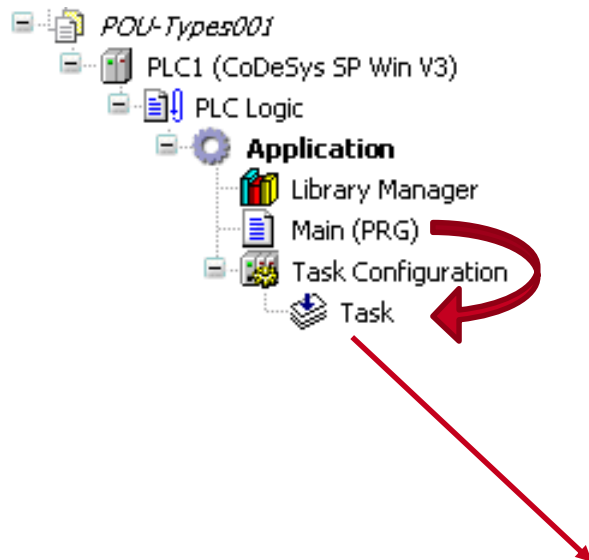
- know the difference between PRG, FB and FC
- be able to use existing library POU's and to create your own ones



- POU TYPES
 - PROGRAM
 - FUNCTION_BLOCK
 - FUNCTION
- Scope of variables
- Exercises with final goal “Alarm handling”



One PROGRAM must be appended to each task



Task

Configuration

Priority (0..31): 1

Type
Cyclic
Interval (e.g. t#200ms): t#10ms

Watchdog
☐ Enable
Time (e.g. t#200ms): ms
Sensitivity: 1

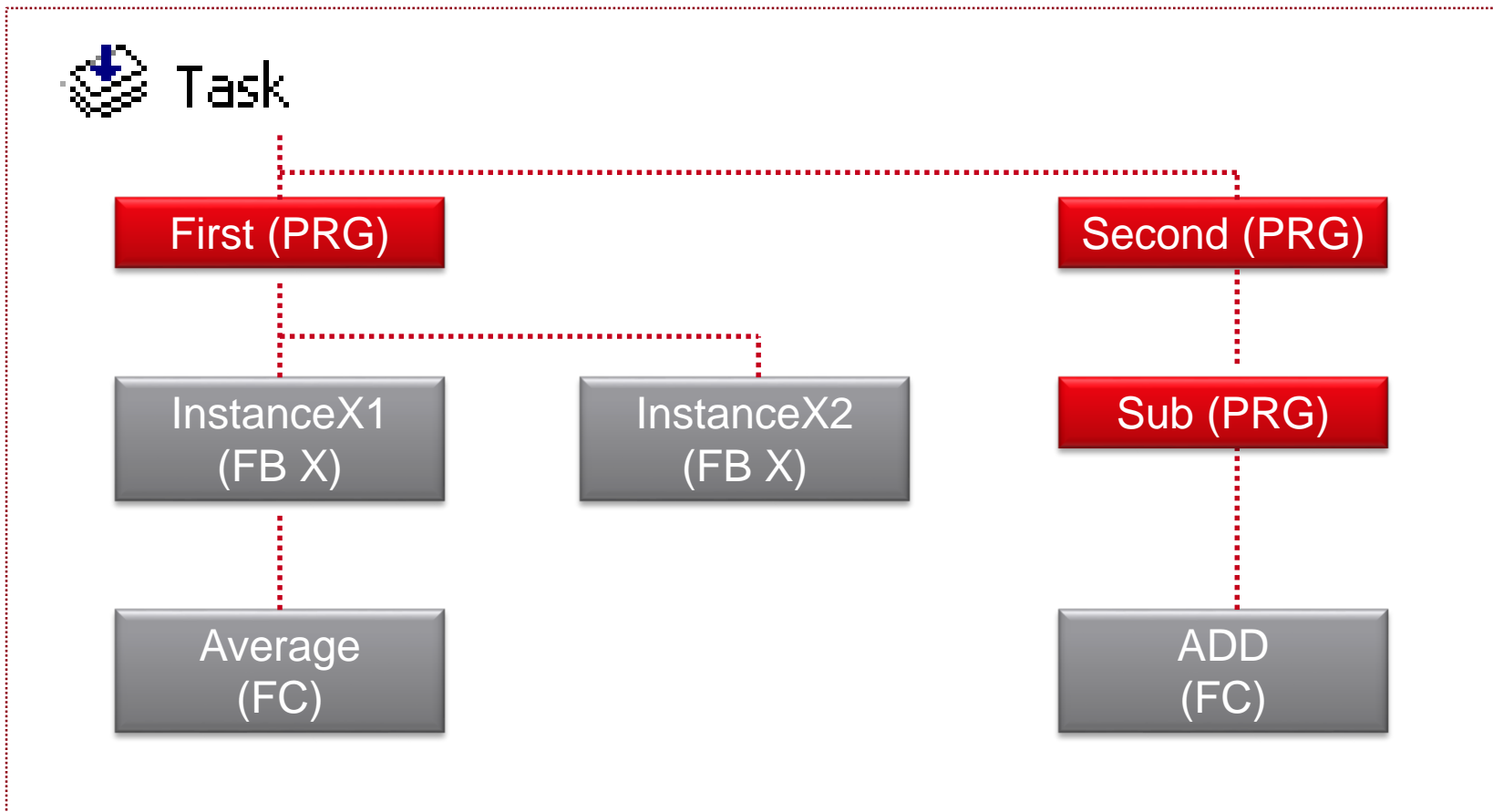
POUs

Add POU
Remove POU
Open POU
Change POU...
Move Up
Move Down

POU	Comment
Main	

One PROGRAM must be appended to each task

Task Configuration



There are three different kinds of POU

- PROGRAM
 - all variables keep their values
 - no copy of data necessary
 - useful for structuring your application



FUNCTION_BLOCK

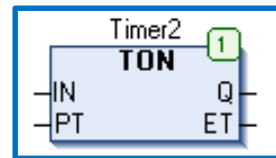
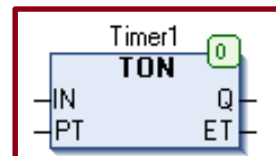
There are three different kinds of POU

FUNCTION_BLOCK

- all variables keep their values
- creating an instance will create a copy of the variables; the same goes for structures
- useful for programming reusable parts like counter, timer, trigger and other machine functions

```

PROGRAM PLC_PRG
VAR
    Timer1: TON;
    Timer2: TON;
END_VAR
    
```



Expression	Type	Value
Timer1	TON	
IN	BOOL	FALSE
PT	TIME	T#0ms
Q	BOOL	FALSE
ET	TIME	T#0ms
M	BOOL	FALSE
StartTime	TIME	T#0ms
Timer2	TON	
IN	BOOL	FALSE
PT	TIME	T#0ms
Q	BOOL	FALSE
ET	TIME	T#0ms
M	BOOL	FALSE
StartTime	TIME	T#0ms



There are three different kinds of POU

■ FUNCTION

- no chance to remember values from the last call
- Local variables will be initialized with each call.
- The name of the function is also a variable which is used to return a value => functions have a type!
- useful for calculating complex expressions
- using global variables is not a good style

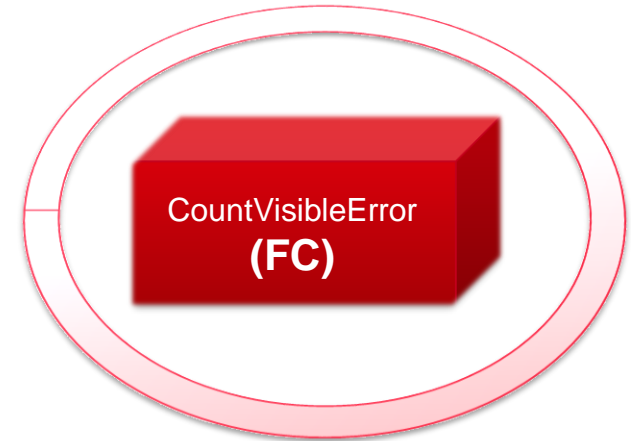
```

Average
1  FUNCTION Average : REAL
2  VAR_INPUT
3      iValue0 : INT;
4      iValue1 : INT;
5      iValue2 : INT;
6  END_VAR

1  Average := INT_TO_REAL(iValue0 + iValue1 + iValue2) / 3.0;

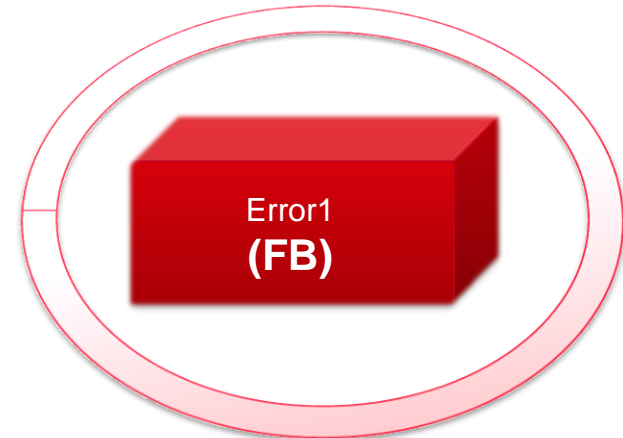
```

■ VAR

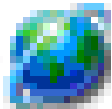


■ VAR_TEMP

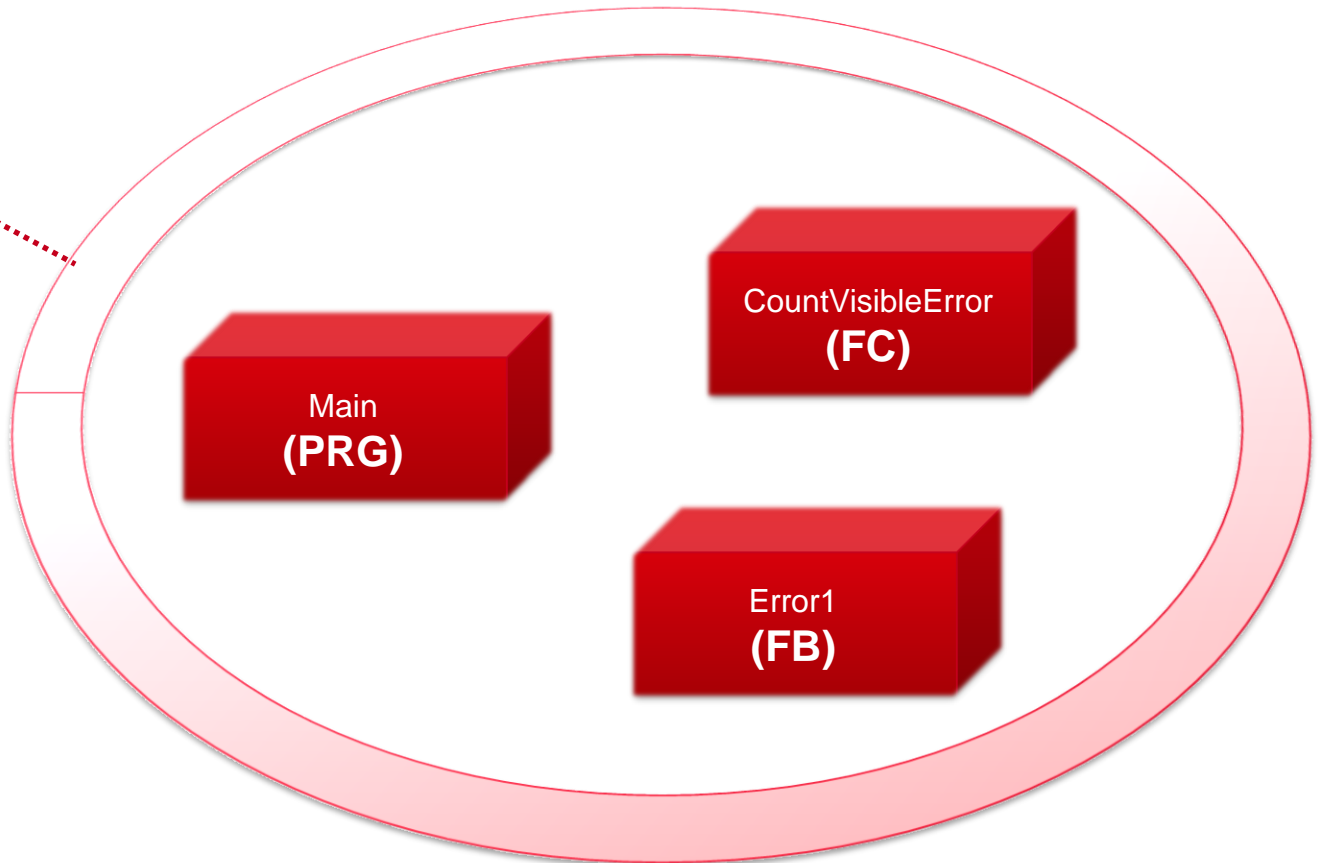
- won't increase size of instance
- is only possible in FBs



- VAR_GLOBAL

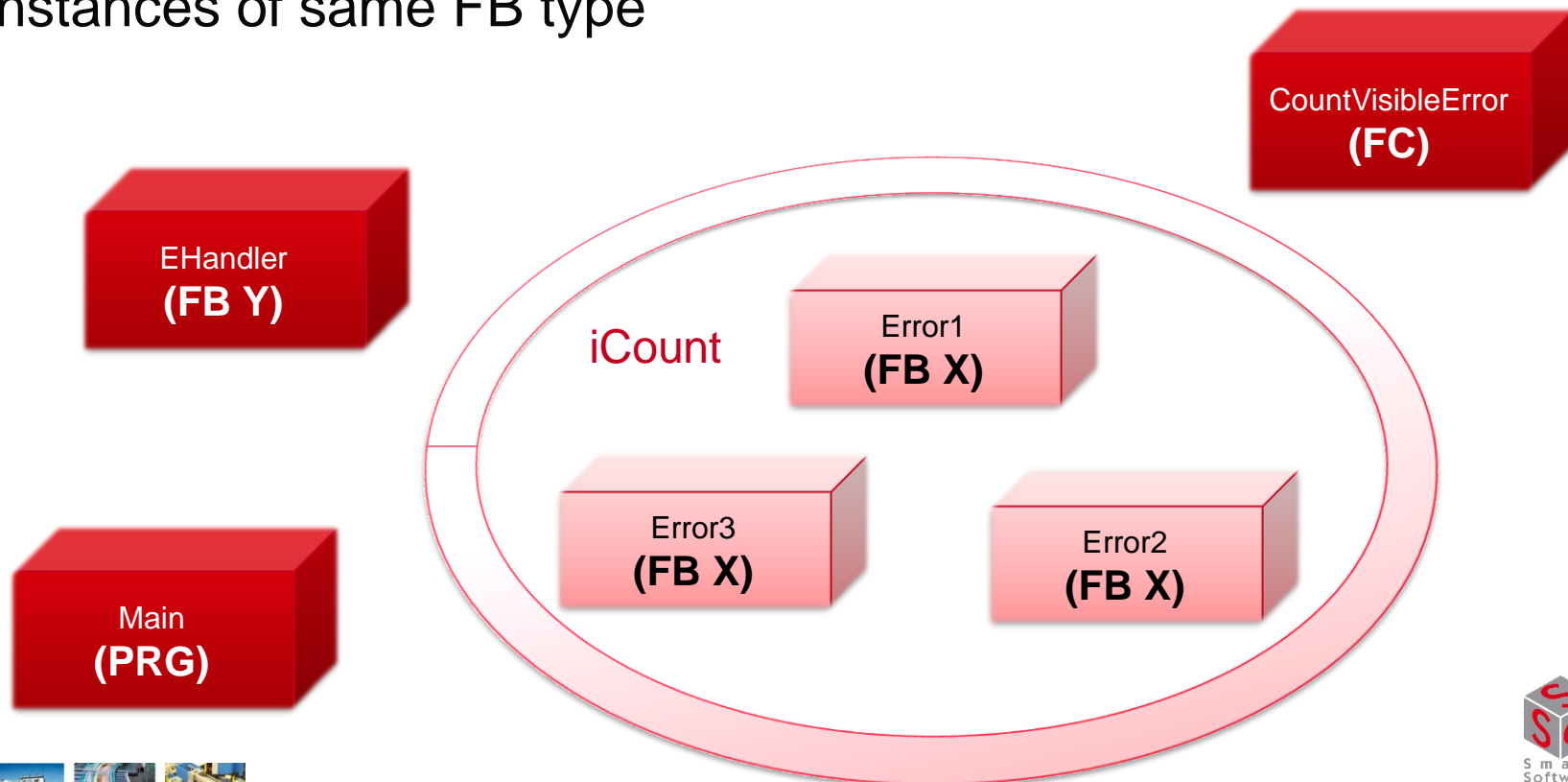


GVL



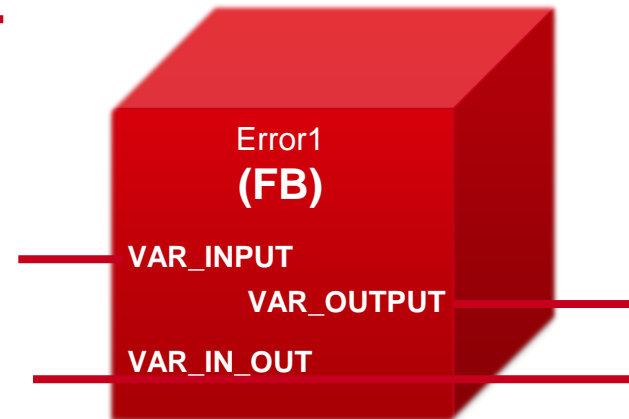
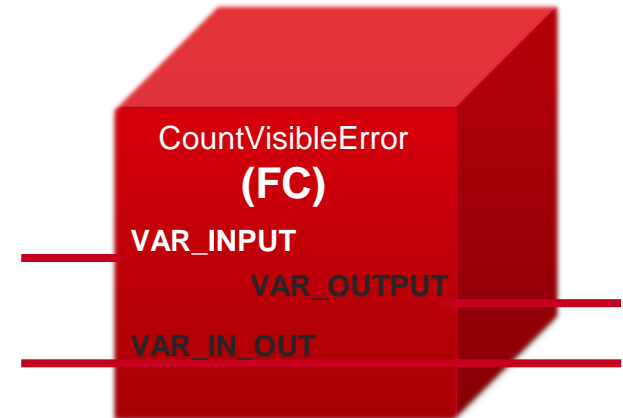
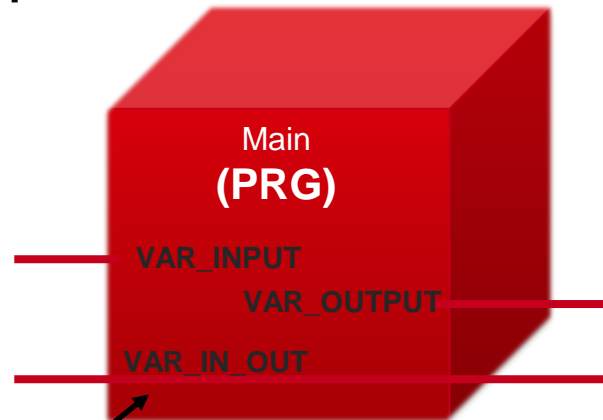
■ VAR_STAT

- behavior like global variables, but access only via FB instances of same FB type



Keywords

- VAR_INPUT
- VAR_OUTPUT
- VAR_IN_OUT



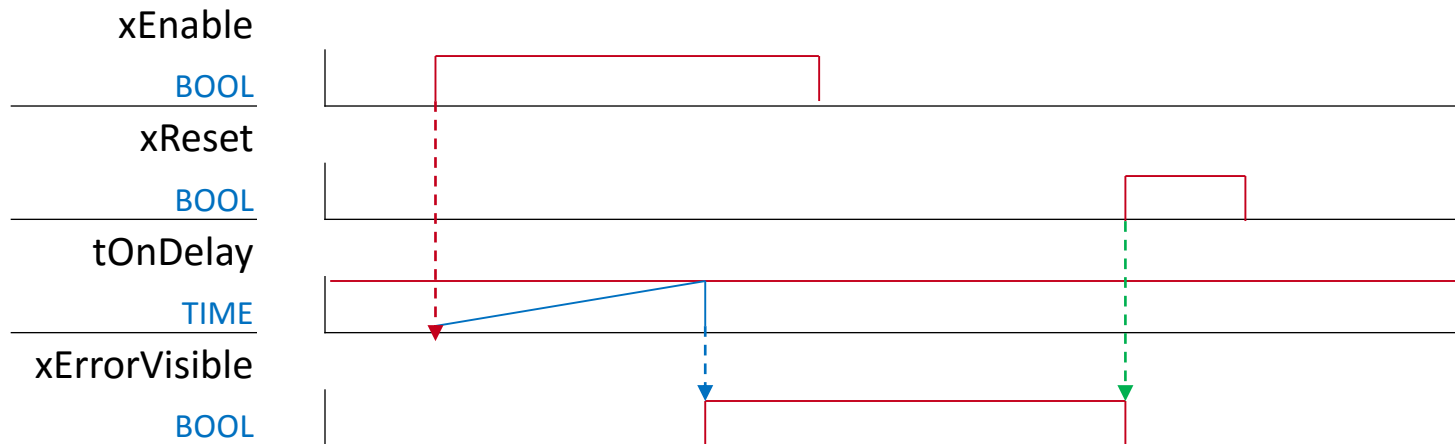
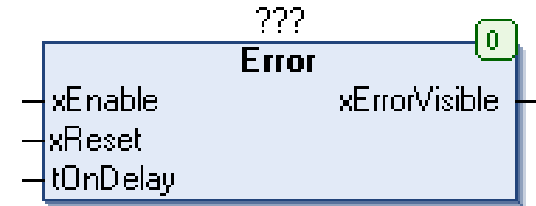
*only in
CoDeSys*



FUNCTION_BLOCK I

Create an FB “Error”

- functionality



FUNCTION_BLOCK I

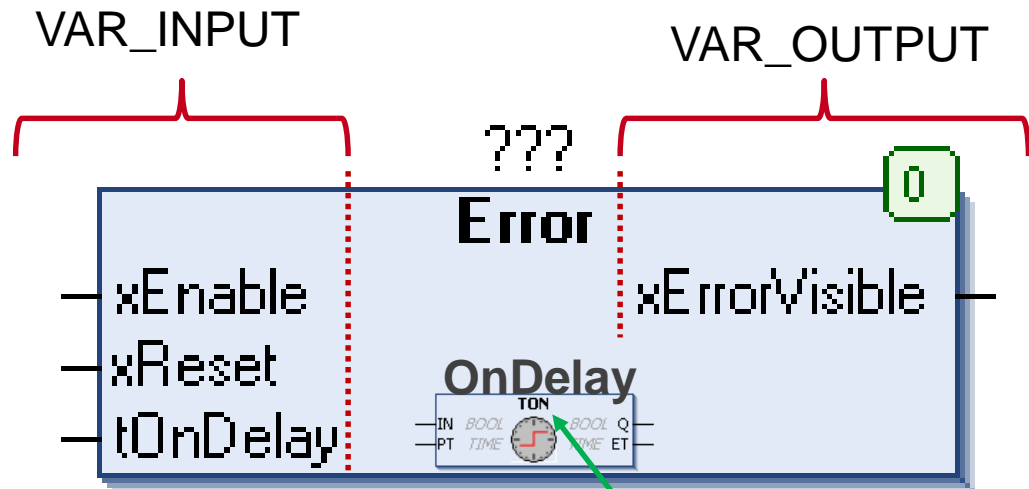
Remember the different scopes

- Declaration in FB Error

```

1 FUNCTION_BLOCK Error
2 VAR_INPUT
3     xEnable : BOOL;
4     xReset  : BOOL;
5     tOnDelay : TIME;
6 END_VAR
7 VAR_OUTPUT
8     xErrorVisible : BOOL;
9 END_VAR
10 VAR
11     OnDelay : TON;
12     srError : SR;
13 END_VAR

```



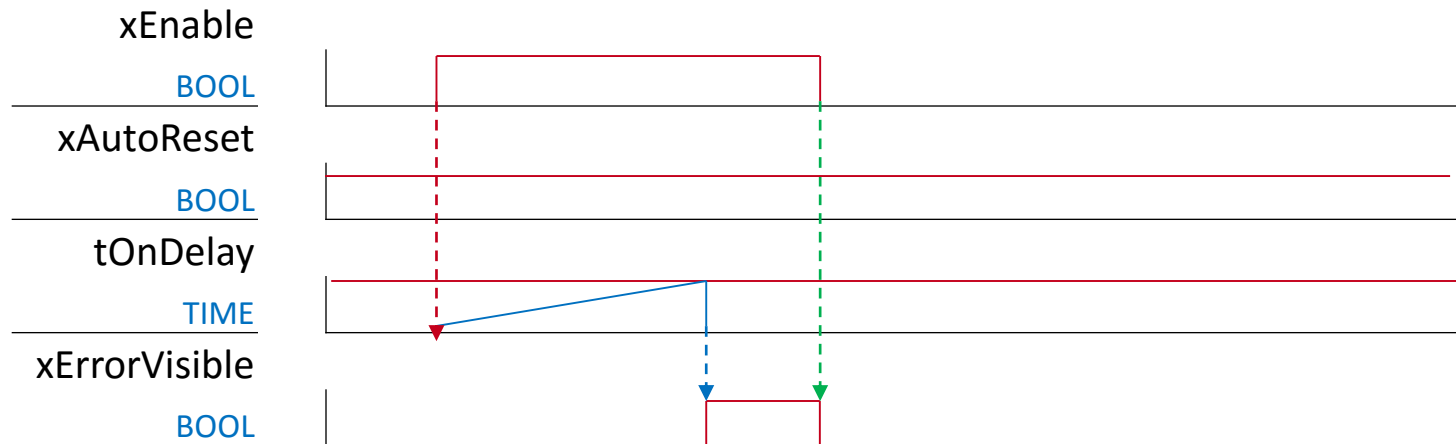
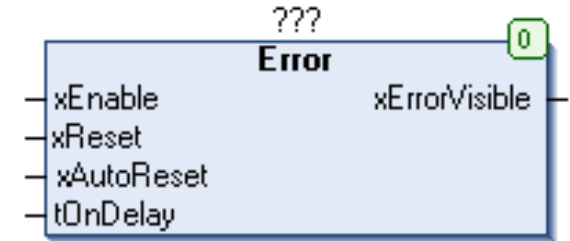
VAR
(Local)



FUNCTION_BLOCK II

Add “AutoReset”

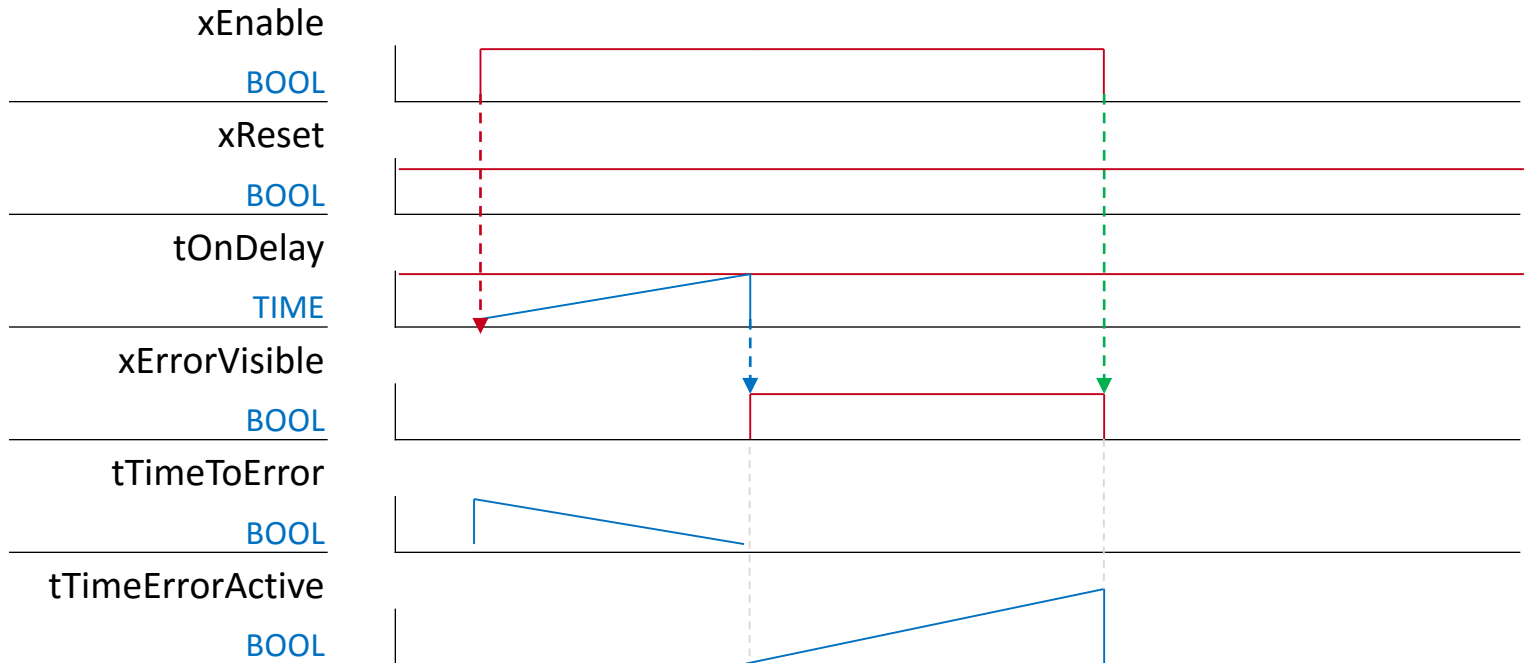
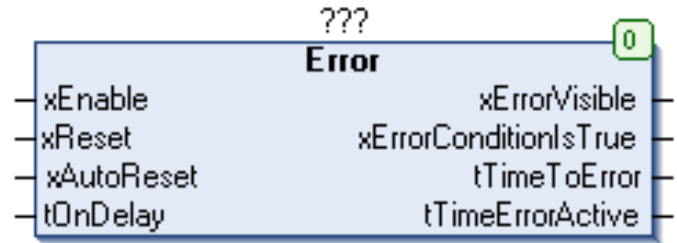
- functionality



FUNCTION_BLOCK III

Add error time information

- functionality



Error handling I

























Global error instances



Err

```

VAR_GLOBAL CONSTANT
    c_iNUM_OF_ERROR : INT := 10;
END_VAR
VAR_GLOBAL
    aError : ARRAY[0..c_iNUM_OF_ERROR - 1] OF Error
            := [
                c_iNUM_OF_ERROR/2(( xAutoReset:= FALSE, tOnDelay:= T#500MS )),
                c_iNUM_OF_ERROR/2(( xAutoReset:= TRUE,  tOnDelay:= T#1S      )) ];
END_VAR
    
```

	c_iNUM_OF_ERROR	INT		 10
		aError	ARRAY [0..(c_iNUM_OF_ERROR - 1)] OF Error	
		aError[0]	Error	
		aError[1]	Error	
		aError[2]	Error	
		aError[3]	Error	
		aError[4]	Error	
		aError[5]	Error	
		aError[6]	Error	
		aError[7]	Error	
		aError[8]	Error	
		aError[9]	Error	














FUNCTION

Create an FC CountErrorVisible



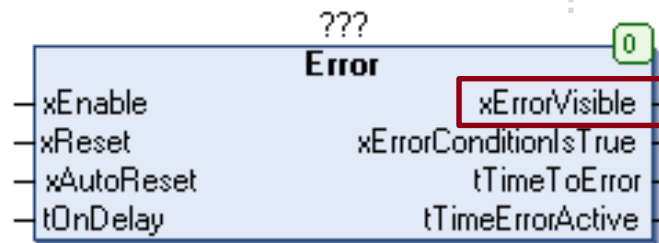
Err

	aError	ARRAY [0..(c_INUM_OF_ERROR - 1)] OF Error
	aError[0]	Error
	aError[1]	Error
	aError[2]	Error
	aError[3]	Error
	aError[4]	Error
	aError[5]	Error
	aError[6]	Error
	aError[7]	Error
	aError[8]	Error
	aError[9]	Error

CountErrorVisible
CountErrorVisible

0

Count "xErrorVisible"
is TRUE



0














Error handling II

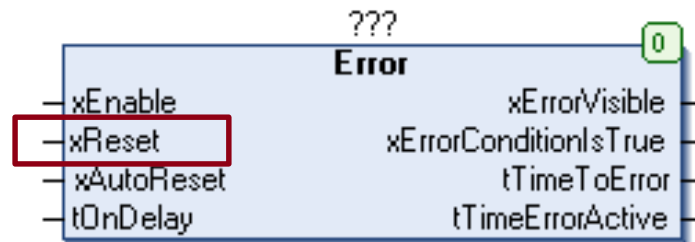
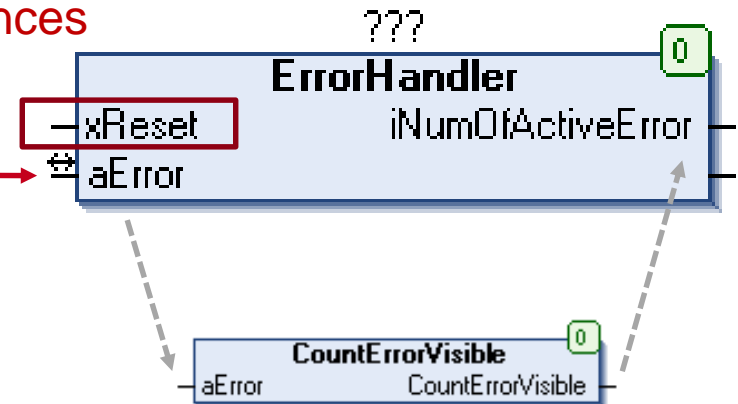
Create an FB "ErrorHandler"



Err

send Reset to all
error instances

	aError	ARRAY [0..(c_INUM_OF_ERROR - 1)] OF Error
	aError[0]	Error
	aError[1]	Error
	aError[2]	Error
	aError[3]	Error
	aError[4]	Error
	aError[5]	Error
	aError[6]	Error
	aError[7]	Error
	aError[8]	Error
	aError[9]	Error



Error handling III

Generate an effect definition for each error



Err

aError		ARRAY [0..(c_INUM_OF_ERROR - 1)] OF Error		
+ aError[0]	Error			
+ aError[1]	Error			
+ aError[2]	Error			
+ aError[3]	Error			
+ aError[4]	Error			
+ aError[5]	Error			
+ aError[6]	Error			
+ aError[7]	Error			
+ aError[8]	Error			
+ aError[9]	Error			
aErrorEffect		ARRAY [0..(c_INUM_OF_ERROR - 1)] OF ARRAY...		
aErrorEffect[0]		ARRAY [0..(c_INUM_OF_EFFECT - 1)] OF BOOL		
aErrorEffect[0][0]	BOOL		TRUE	
aErrorEffect[0][1]	BOOL		FALSE	
aErrorEffect[0][2]	BOOL		FALSE	
aErrorEffect[0][3]	BOOL		FALSE	
aErrorEffect[0][4]	BOOL		FALSE	
aErrorEffect[0][5]	BOOL		FALSE	
aErrorEffect[0][6]	BOOL		FALSE	
aErrorEffect[0][7]	BOOL		FALSE	
aErrorEffect[0][8]	BOOL		FALSE	
aErrorEffect[0][9]	BOOL		FALSE	
+ aErrorEffect[1]	ARRAY [0..(c_INUM_OF_EFFECT - 1)] OF BOOL			
+ aErrorEffect[2]	ARRAY [0..(c_INUM_OF_EFFECT - 1)] OF BOOL			
+ aErrorEffect[3]	ARRAY [0..(c_INUM_OF_EFFECT - 1)] OF BOOL			
+ aErrorEffect[4]	ARRAY [0..(c_INUM_OF_EFFECT - 1)] OF BOOL			
+ aErrorEffect[5]	ARRAY [0..(c_INUM_OF_EFFECT - 1)] OF BOOL			
+ aErrorEffect[6]	ARRAY [0..(c_INUM_OF_EFFECT - 1)] OF BOOL			
+ aErrorEffect[7]	ARRAY [0..(c_INUM_OF_EFFECT - 1)] OF BOOL			
+ aErrorEffect[8]	ARRAY [0..(c_INUM_OF_EFFECT - 1)] OF BOOL			
+ aErrorEffect[9]	ARRAY [0..(c_INUM_OF_EFFECT - 1)] OF BOOL			



Error handling III

Why to define effects?

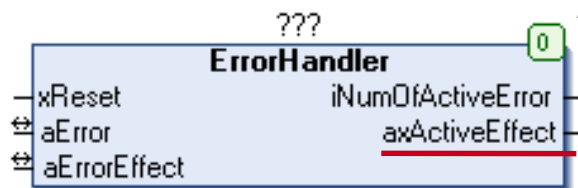
Error 0 is visible

aErrorEffect[0]	ARRA...	
aErrorEffect[0][0]	BOOL	TRUE
aErrorEffect[0][1]	BOOL	FALSE
aErrorEffect[0][2]	BOOL	FALSE
aErrorEffect[0][3]	BOOL	FALSE
aErrorEffect[0][4]	BOOL	FALSE
aErrorEffect[0][5]	BOOL	FALSE
aErrorEffect[0][6]	BOOL	FALSE
aErrorEffect[0][7]	BOOL	FALSE
aErrorEffect[0][8]	BOOL	FALSE
aErrorEffect[0][9]	BOOL	FALSE

OR

Error 9 is visible

aErrorEffect[9]	ARRA...	
aErrorEffect[9][0]	BOOL	FALSE
aErrorEffect[9][1]	BOOL	FALSE
aErrorEffect[9][2]	BOOL	FALSE
aErrorEffect[9][3]	BOOL	FALSE
aErrorEffect[9][4]	BOOL	FALSE
aErrorEffect[9][5]	BOOL	TRUE
aErrorEffect[9][6]	BOOL	TRUE
aErrorEffect[9][7]	BOOL	TRUE
aErrorEffect[9][8]	BOOL	TRUE
aErrorEffect[9][9]	BOOL	TRUE



axActiveEffect	ARRAY [0..(Err.c_iN...	
axActiveEffect[0]	BOOL	TRUE
axActiveEffect[1]	BOOL	FALSE
axActiveEffect[2]	BOOL	FALSE
axActiveEffect[3]	BOOL	FALSE
axActiveEffect[4]	BOOL	FALSE
axActiveEffect[5]	BOOL	TRUE
axActiveEffect[6]	BOOL	TRUE
axActiveEffect[7]	BOOL	TRUE
axActiveEffect[8]	BOOL	TRUE
axActiveEffect[9]	BOOL	TRUE



- What is the difference between a FUNCTION_BLOCK and a PROGRAM?
- Can you store local variables in a FUNCTION from one cycle to the next?
- What is the function of the keyword VAR_STAT?
- How can you use a FUNCTION_BLOCK?
- What is the right way to integrate big data structures into a FUNCTION_BLOCK?

