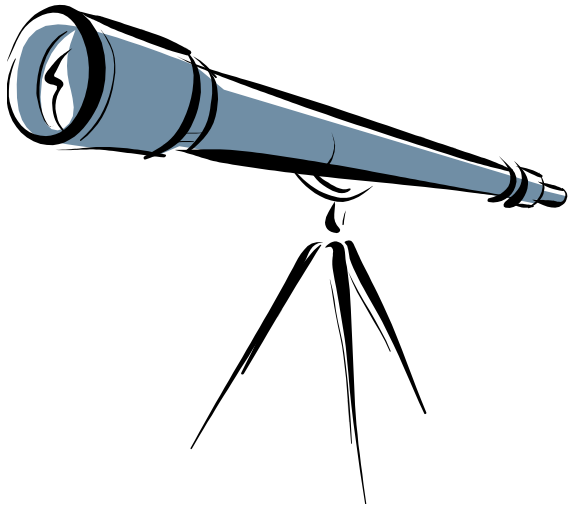
Introduction

A training about the basics of IEC 61131-3 programming tool CoDeSys

Source: The CoDeSys training material made from the text in the online help of CoDeSys programming tool version 3.5 and different help documents from CoDeSys (3S) as well as published text on the homepage http://www.codesys.com and various examples

Training Agenda (2 days)



- 3S-Smart Software Solutions GmbH & CODESYS
- Beijer Electronics offer
- Structured project, IEC 61131-3
  - Editors IL / LD / FBD / ST / SFC / CFC
- CoDeSys programming tool
  - User interface
- Task / POU / Variables
  - Declaration of Local and Global variables
- Exercises with editors and Elevator Simulator
  - Timers and Counters
  - Operands and Calculations
- Create user made blocks (FB / FUN)
- Library Management
- Diagnostics and other features
- Project Backup
- Device settings and Transfer to HW >> Appendix
  - Example with TxA or TxB SoftControl and Crevis I/O

CoDeSys V3

3S-Smart Software Solutions

3S-Smart Software Solutions Gmbh

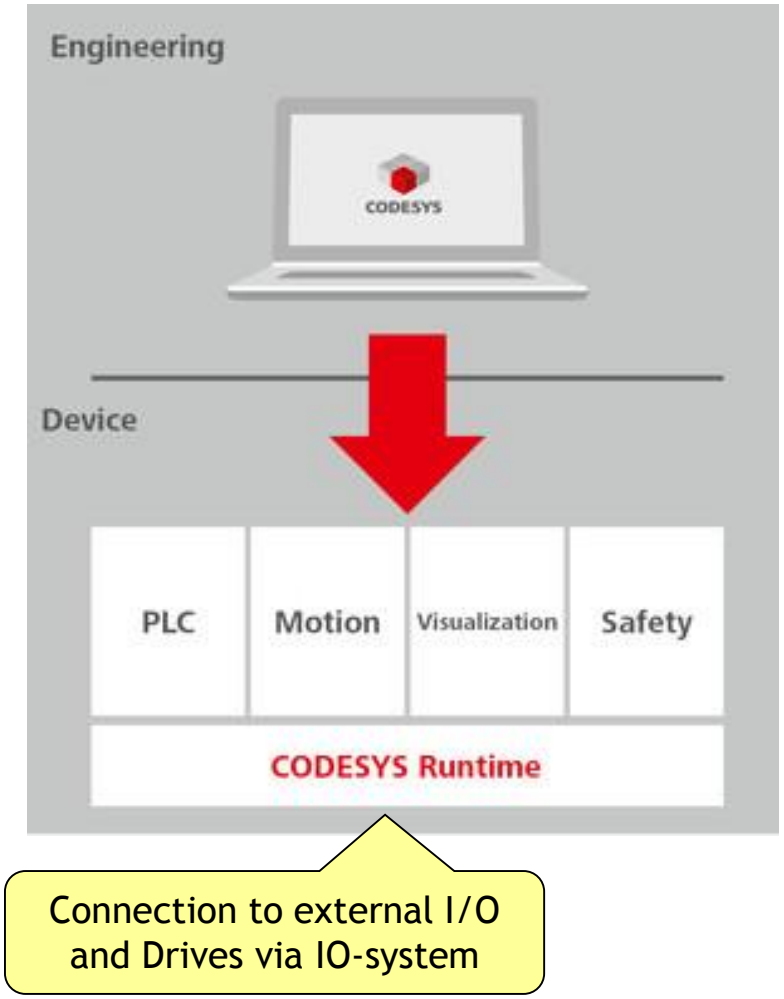
- **The company of CoDeSys**
  3S-Smart Software Solutions
  - Headquarters in Kempten, Germany
  - Founded in 1994 by Dieter Hess and Manfred Werner
  - More than 100 software enginers
  - The company is certified to ISO 9001
- **CoDeSys Products**
  - CODESYS Engineering, Runtime, Visualization, Fieldbus, Motion + CNC and Safety
  - CoDeSys is used in virtually all sectors of the automation industry
  - Different Devices programmable with CODESYS from >350 manufactures

http://www.codesys.com/

CoDeSys (Controller Development System)

- **CoDeSys is the product name of the complete software family of IEC 61131-3 programming tools**

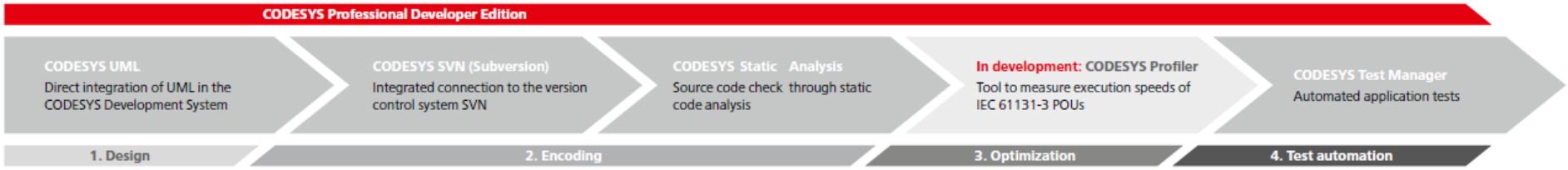
- **The runtime system CoDeSys Control provides the following main functions:**
  - Execution of the application(s), that are created with CoDeSys 3.x
  - Debugging of the IEC application
  - Connection to the IO-system and Drives
  - Communication with the programming tool CoDeSys 3.x or other clients (HMI)
  - Routing for communication to subordinate runtime systems
  - Runtime system to runtime system communication (“PLC-to-PLC”)



Overview CoDeSys - Key benefits

- **The IEC 61131-3 Development System**
  - Free programming tool, no fuzz. A large number of companies rely on CoDeSys!
- **CoDeSys Control - the "SoftPLC" Runtime System (OEM)**
  - Available for OS like e.g. Windows CE, VxWorks and Linux, further upon request
- **CoDeSys Control RTE – "Hard realtime" PLC control**
  - Turns any type of industrial PC with Windows XP/Vista/7 operating system into a powerful PLC
- **CoDeSys SoftMotion - Control and Motion become one**
  - Single or Multi axis movements with PLCopen motion POUs, CAM & gearing, CNC..
- **CoDeSys Safety – SIL 2/3 possibilities (IEC 61508)**
- **CoDeSys OPC-Server**
  - A part of the standard delivery package of CoDeSys Development System

- **CODESYS Professional Developer Edition – Efficient Application Development with integrated Add-Ons in the IEC 61131-3 Development System**
  - http://www.codesys.com/products/codesys-engineering/professional-developer-edition.html

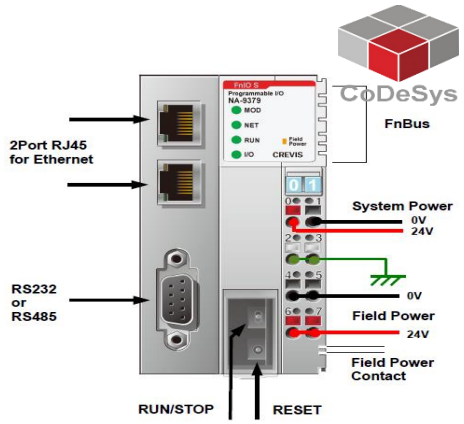


The offer from Beijer Electronics

- CoDeSys Embedded Controllers

Low-end PLC (HMI optional)

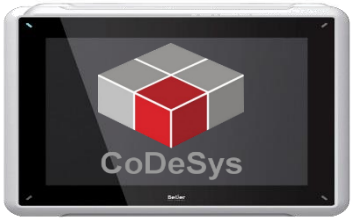
Crevis NA-9379



Cost-efficient HMI PLC + I/O

iX TxA SoftControl
+ Crevis I/O

Serial Modbus RTU
(Modbus TCP)



High performance HMI PLC + I/O

iX TxB SoftControl and SoftMotion
+ Crevis I/O

EtherCAT
Modbus TCP

Crevis NA-9379 - The Programmable I/O

- **NA-9379 ”the PIO”**
  - A smart and compact PLC expandable with various I/O-modules of FnIO-S series
- **General**
  - Modbus/TCP client for Remote I/O etc
  - Modbus/TCP server for HMI/SCADA communication
  - Modbus RTU slave on RS485 port
  - PLC<>PLC communication via standard CODESYS functionality
  - Application memory, 512 kB
  - Operating temperature -20 -> +50°C

The iX HMI SoftControl

- **A combination of two automation products:**
  - iX HMI solution from Beijer Electronics
    - » iX offer an open development platform through .NET components and to create customized functionality using C# scripting
  - CoDeSys, the SoftPLC runtime system

The iX TxA SoftControl range

Modbus RTU

The iX TxB SoftControl range

EtherCAT Modbus RTU/TCP

CoDeSys V3

Structured Project

CoDeSys, the standard in IEC 61131-3 Controller and PLC programming
Made by company 3S-Smart Software Solutions, located in south of Germany

Beijer ELECTRONICS

IEC 61131-3 standard

- Programmable Controller Program Languages

- There are 5 program languages defined in the IEC 61131-3 standard
  - IL (Instruction List)
  - LD (Ladder Diagram)
  - FBD (Function Block Diagram)
  - ST (Structured Text)
  - SFC (Sequential Function Chart)

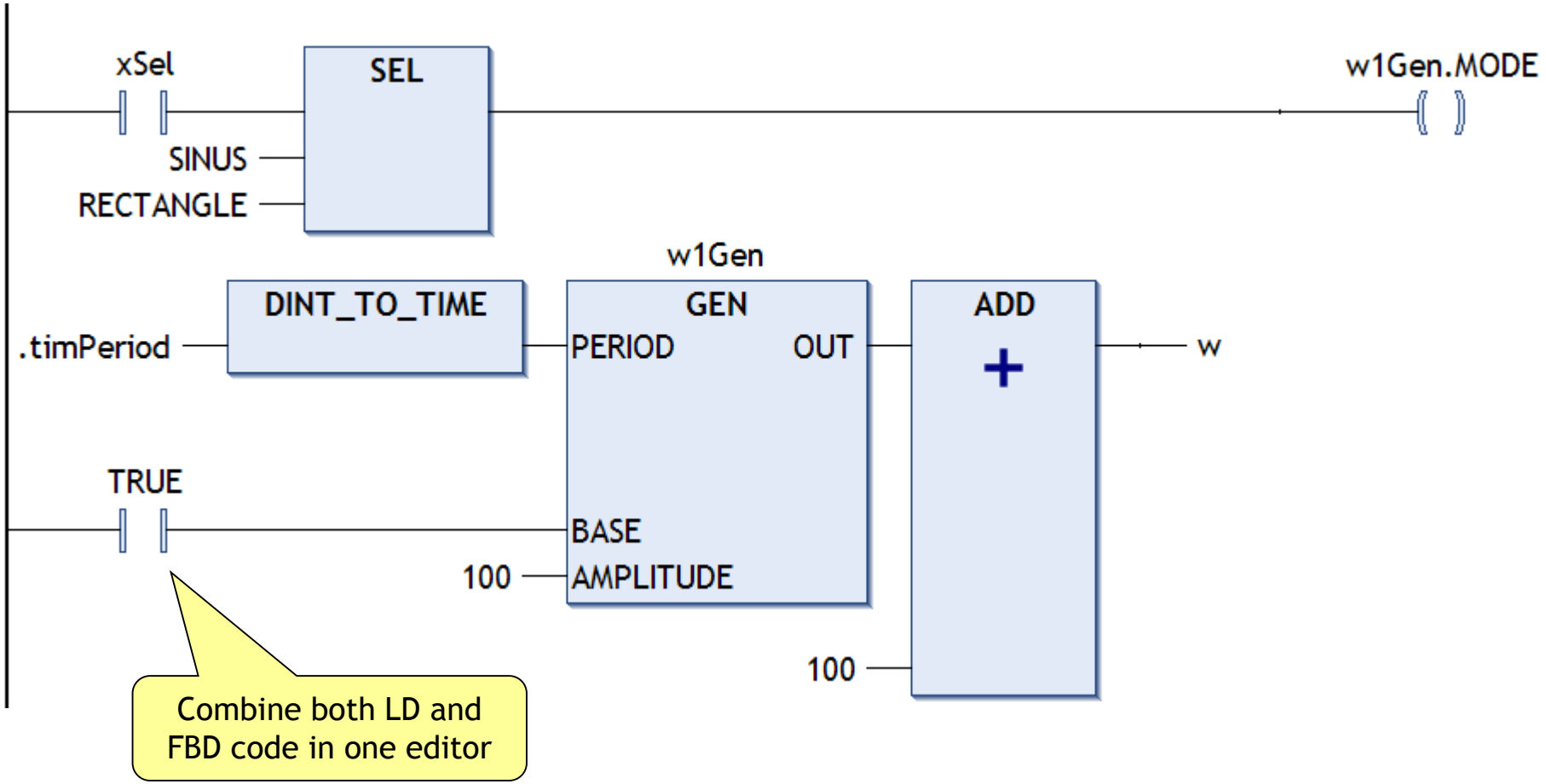
- CoDeSys provide one additional CFC-editor
  - CFC (Continuous Function Chart)
  - An extension to the IEC 61131-3 programming languages

- CoDeSys is certified by PLCopen, www.plcopen.org

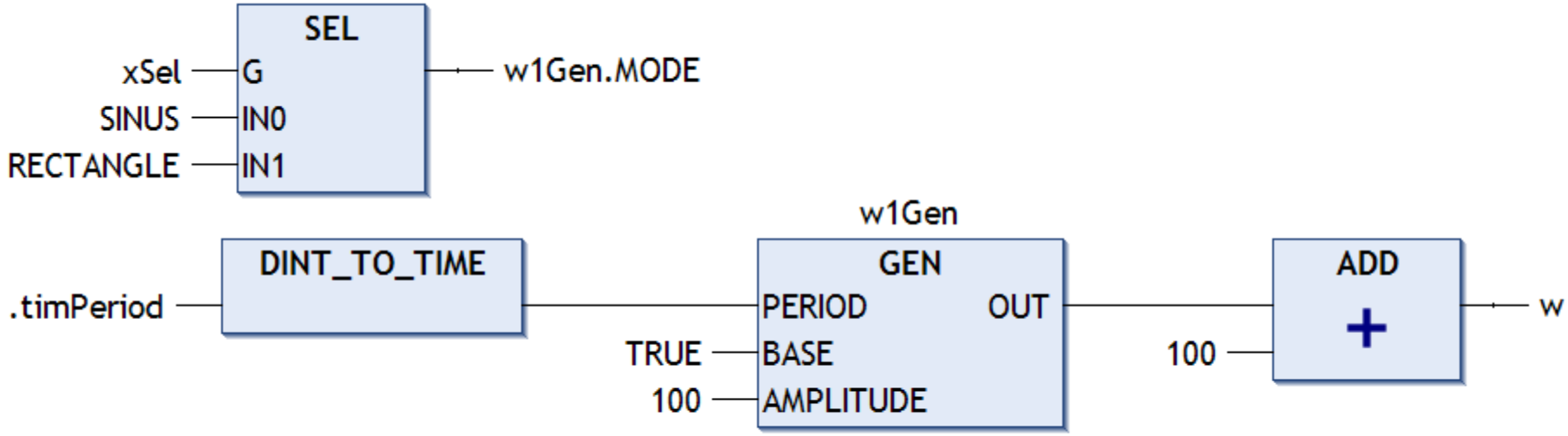
IEC 61131-3, Instruction List (IL)

```
LD                  xSel
SEL                 SINUS
                    RECTANGLE
ST                  w1Gen.MODE
LD                  .timPeriod
DINT_TO_TIME
ST                  w1Gen.PERIOD
CAL                 w1Gen(
          BASE:=    TRUE,
     AMPLITUDE:=    100)
LD                  w1Gen.OUT
ADD                 100
ST                  w
```

IEC 61131-3, Ladder logic (LD)



IEC 61131-3, Function Block Diagram (FBD)

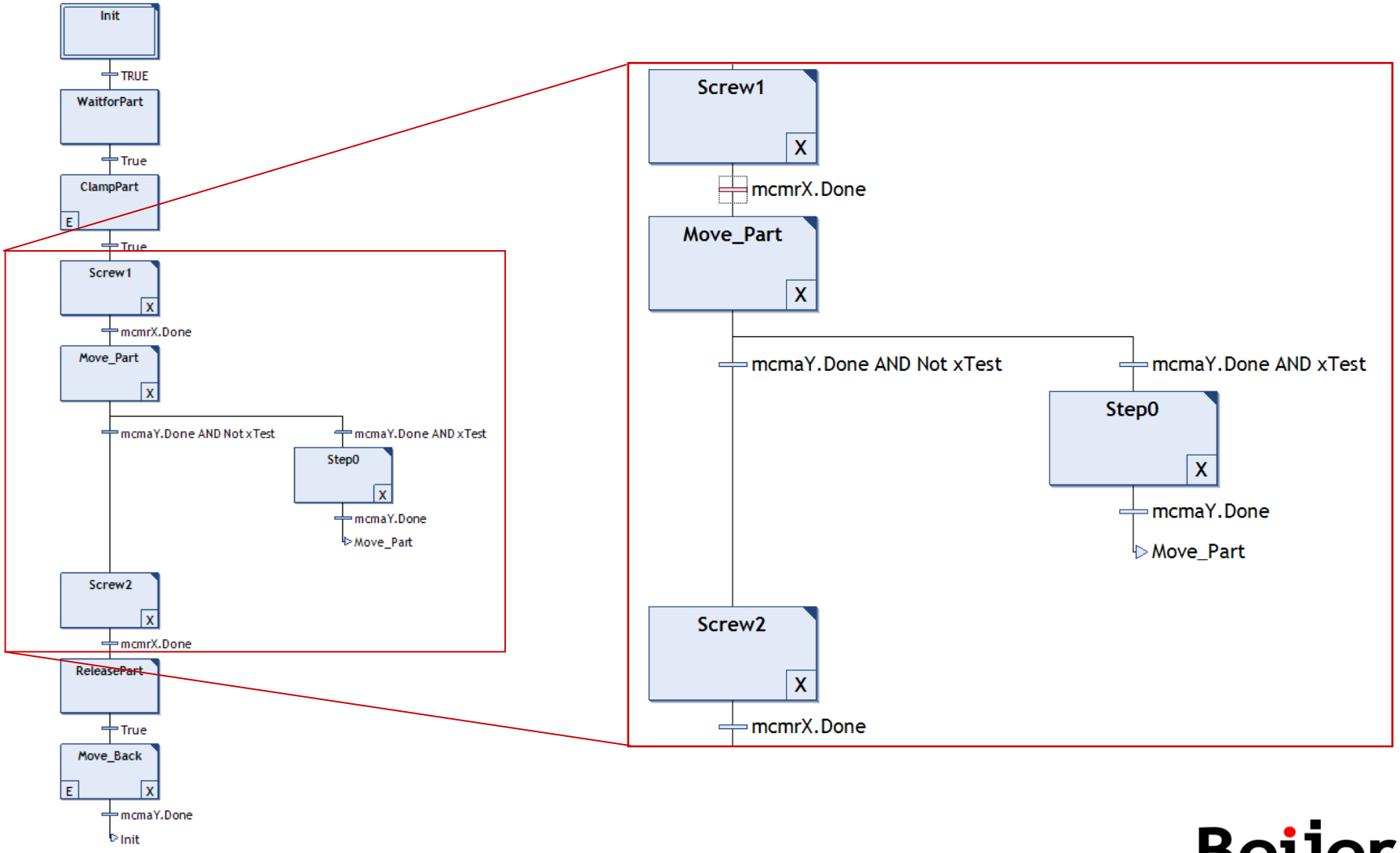


Beijer ELECTRONICS

IEC 61131-3, Structured Text (ST)

```
IF (uiCnt MOD uiSpeed) = 0 THEN
    shift := shift + 1;
    FOR i:= cOutMod TO 1 BY - 1 DO;
        abOut[i-1]:= abOut[i];
        IF shift > cZeichen THEN;
            abOut[i]:= 0;
        ELSE
            abOut[i] := abText[shift];
        END_IF
    END_FOR
    IF shift >  cZeichen + cOutMod THEN
        shift := 0;
        uiSpeed:= uiSpeed + 2;
        IF uiSpeed > 63 THEN
            uiSpeed := 0;
        END_IF
    END_IF
END_IF
```

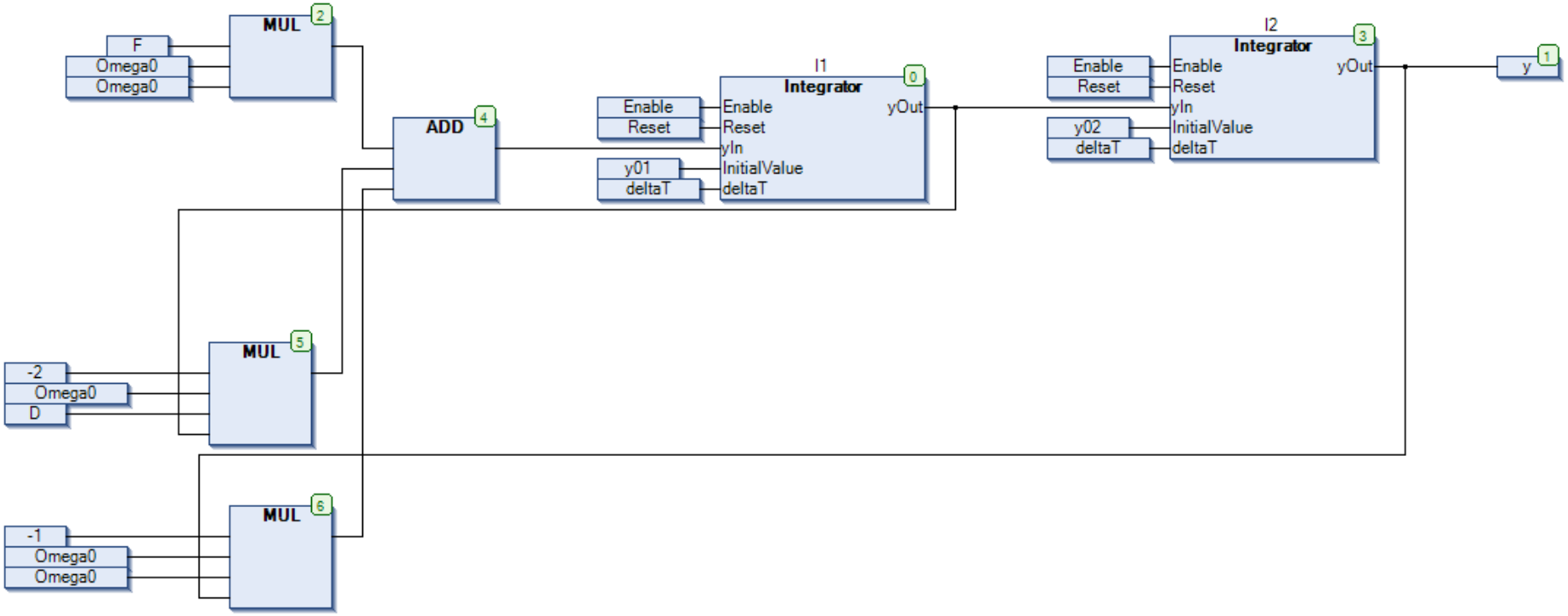
IEC 61131-3, Sequential Function Chart (SFC)



Beijer ELECTRONICS

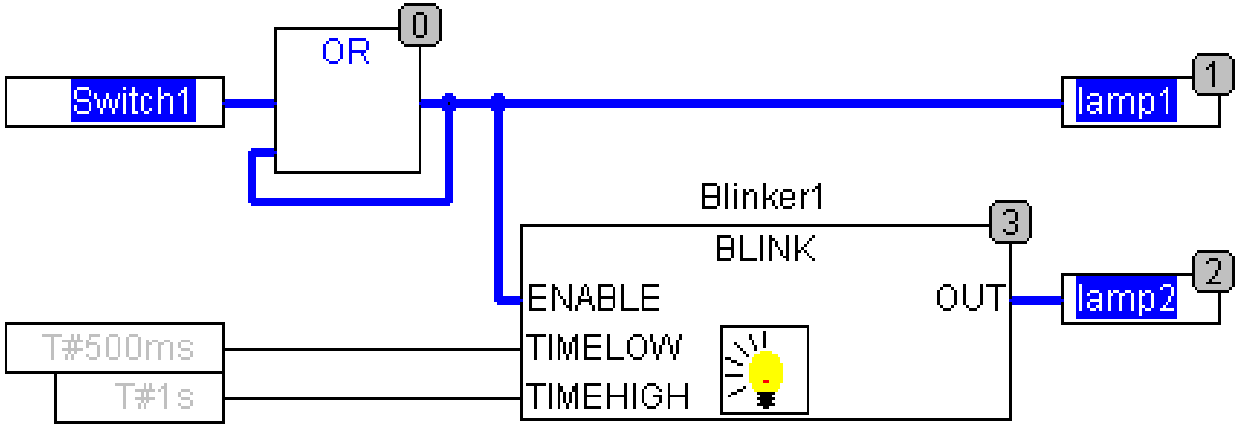
Continuous Function Chart (CFC)

- The extension to the IEC 61131-3 programming languages
- Another implementation of the Function Block Diagram (FBD-editor)
- The execution sequence can be controlled and visualized with the little boxes in the right top corner of each function box



Continuous Function Chart (CFC)

- Pros with CFC-editor:
  - Easy to understand the CFC graphical editor
  - The CFC editor allows continuous connections for example for programming feedback loops and to build macros of boxes and their connections
  - Make program with ready-made blocks (FUN / FB) link them together and set parameters, and allow “Auto routing” of connections
  - Makes it possible to explicitly control the execution order

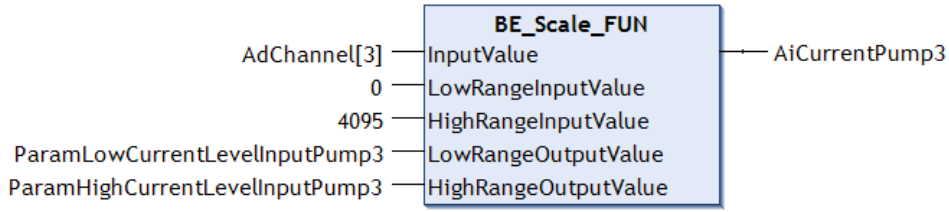


Program Organization Unit (POU)

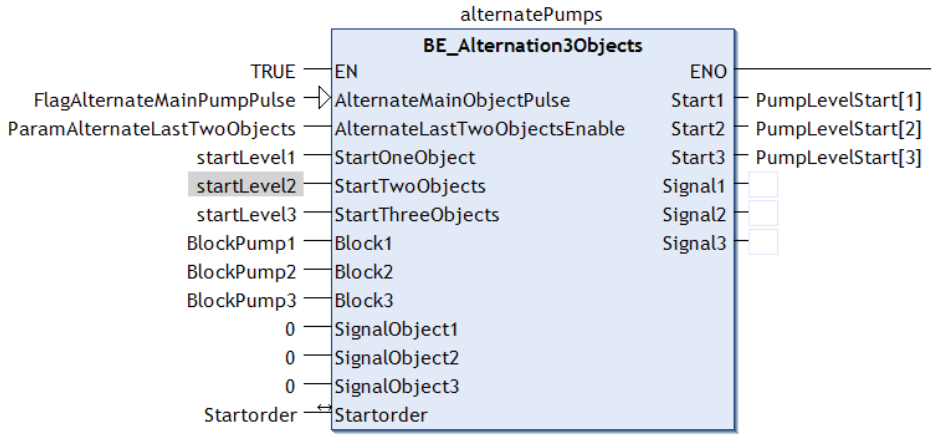
- IEC 61131-3 types of program parts available in CoDeSys

1) Program Block (PRG)
   - Editors of type IL, LD, FBD, ST, SFC and CFC
   - FUN and FB are called from the PRG

2) Function (FUN)
   - One output

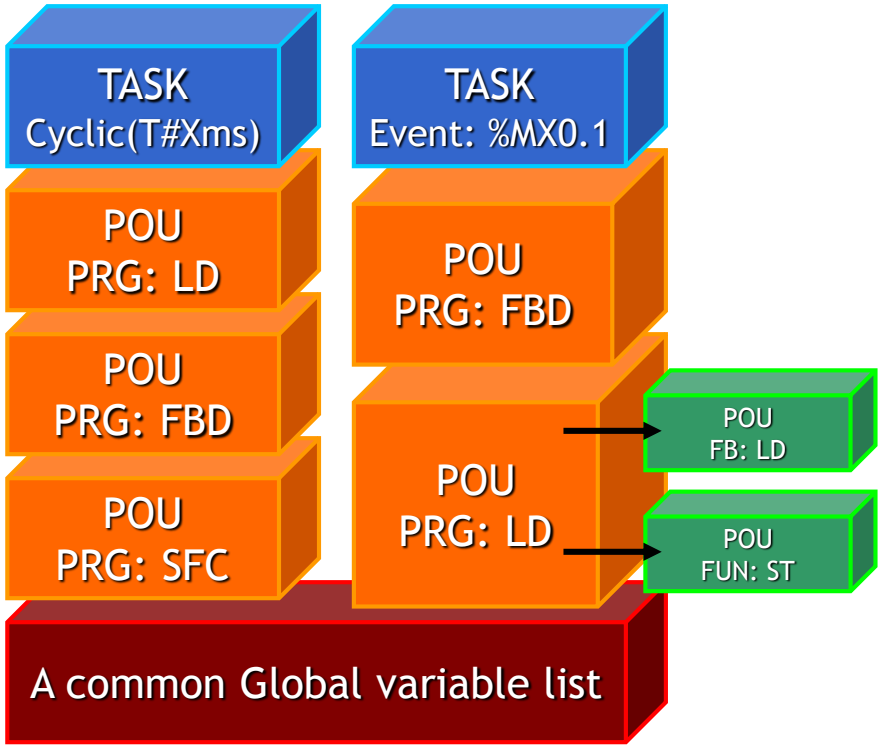


3) Function Block (FB)
   - Several outputs
   - Called by instance



Beijer ELECTRONICS

IEC 61131-3, Structured Project



- TASK – Execution control
  - An execution control element in the processing of IEC program
  - A Task is defined by a priority and by a type condition as Cyclic (Intervall), Event, Freewheeling or Status, that will trigger the start of the execution

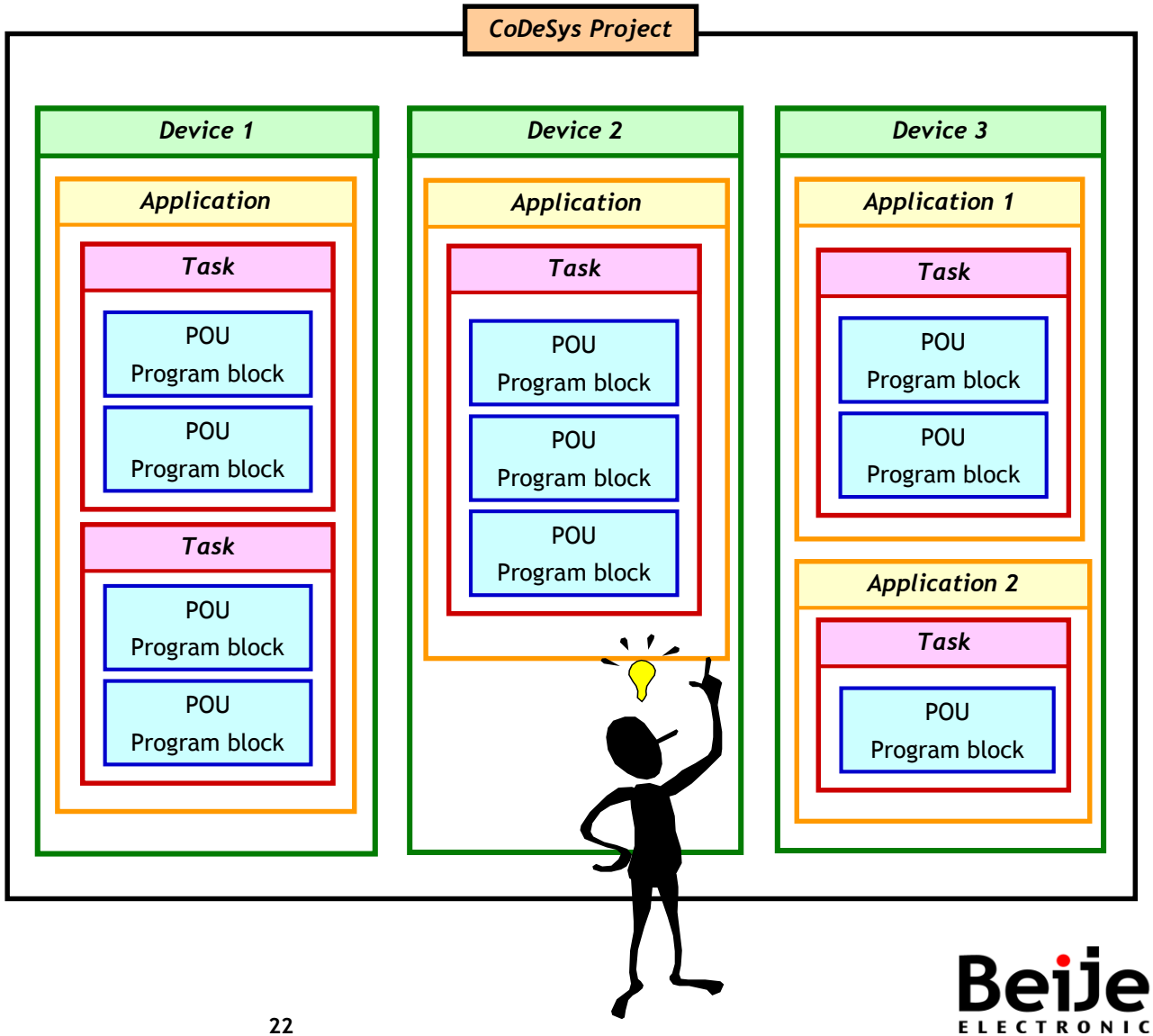
- POU - Program Organization Unit
  - PRG (Program)
  - FUN (Function)
  - FB (Function Block)

- GVL - Global Variable List
  - Multiple number of GVL per project

CoDeSys, Structured Project

- Each project contains at least - one Device (Soft PLC)
- Each device contains at least - one Application
- Each application contains at least - one Task
- Each task contains at least - one POU

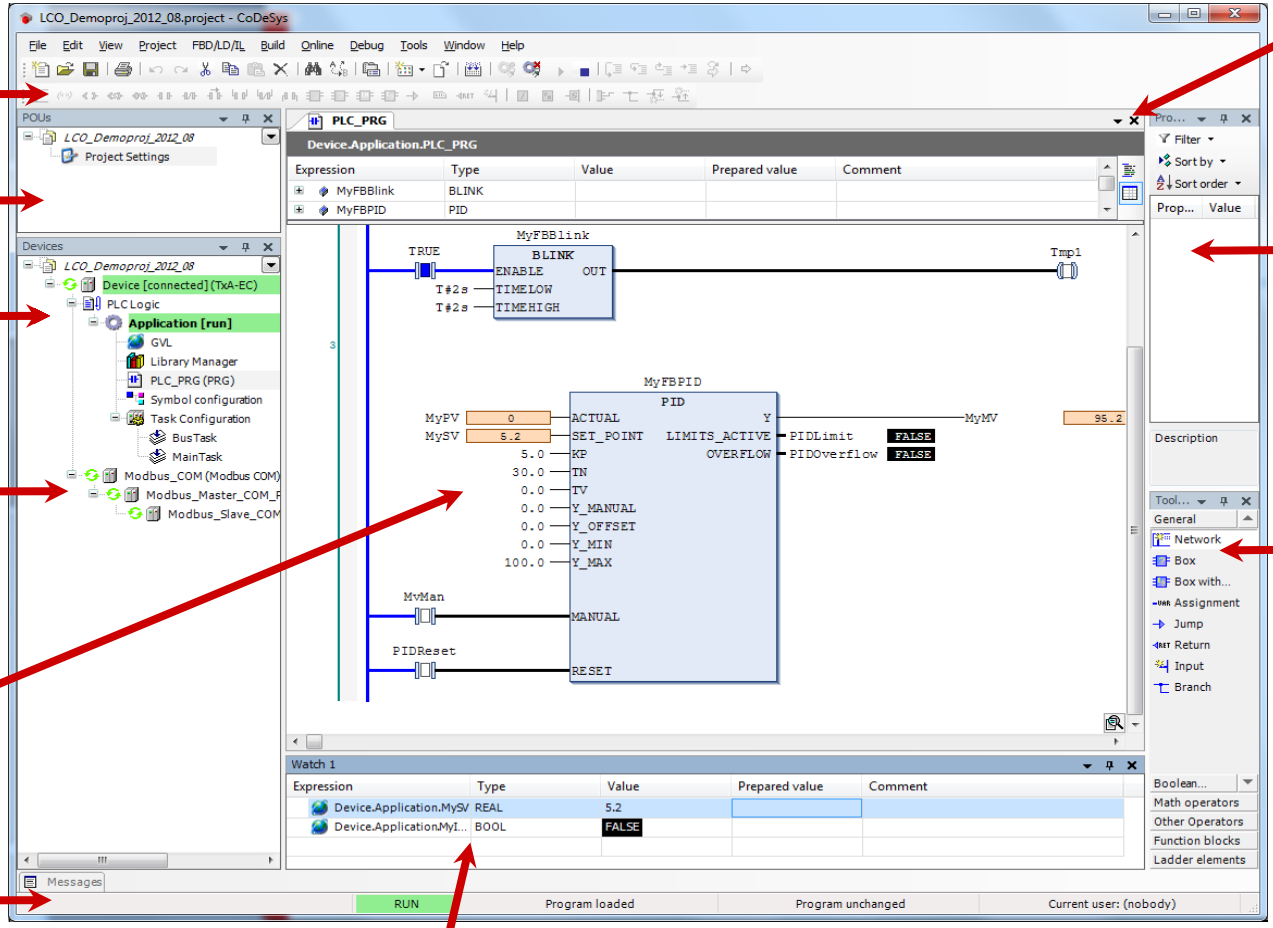
- Note, one Device may have more than one Application (compare multiple CPU solutions)



CoDeSys V3

Programming Tool

Launching CoDeSys



Toolbar

Project settings

Devices window, Navigation

I/O Driver/ Fieldbus status

Work window

Status bar

Watch window

Dockable windows

Properties window

ToolBox selection window

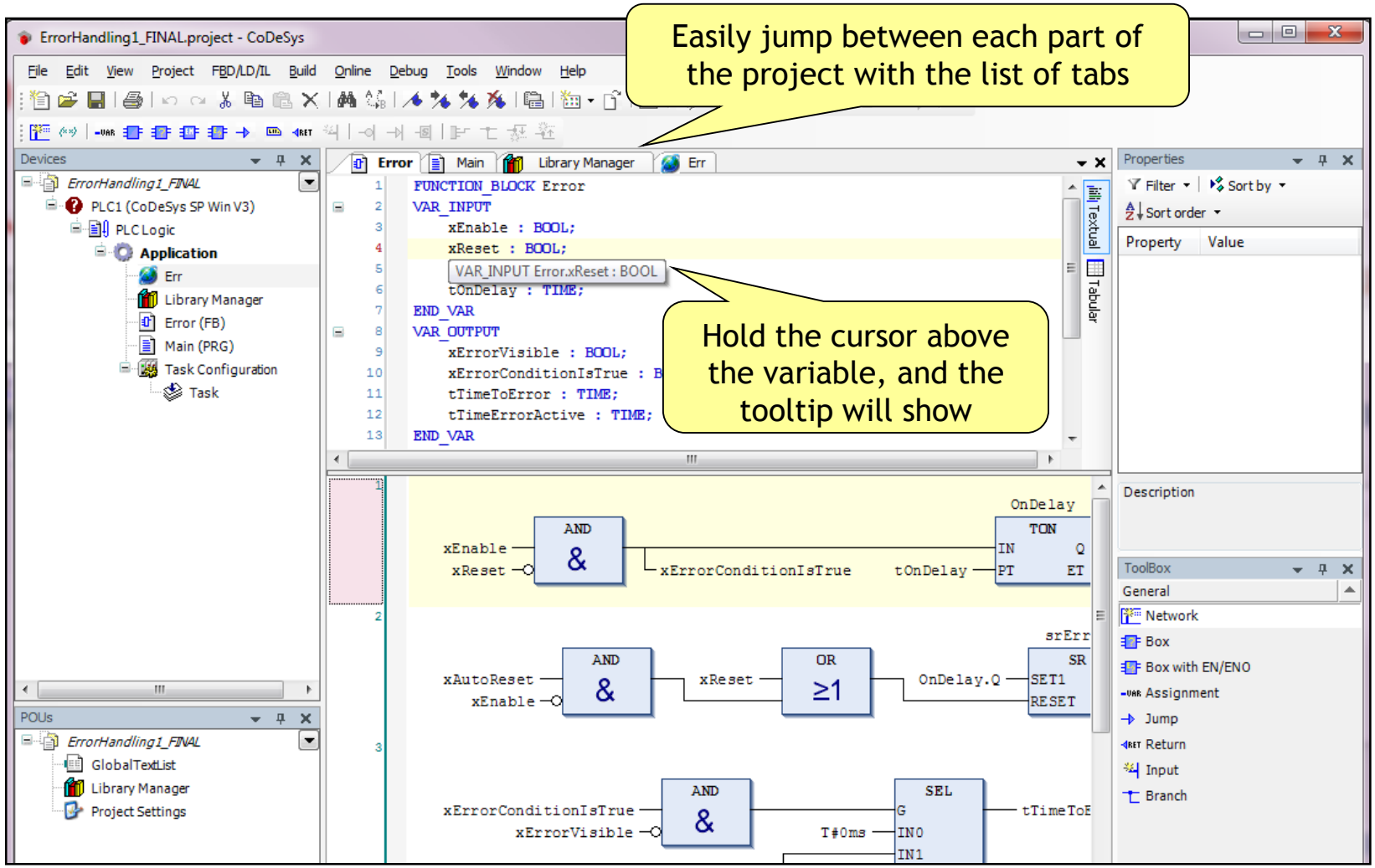
Beijer ELECTRONICS

Toolbars

- Docking/floating toolbars

Drag & Drop

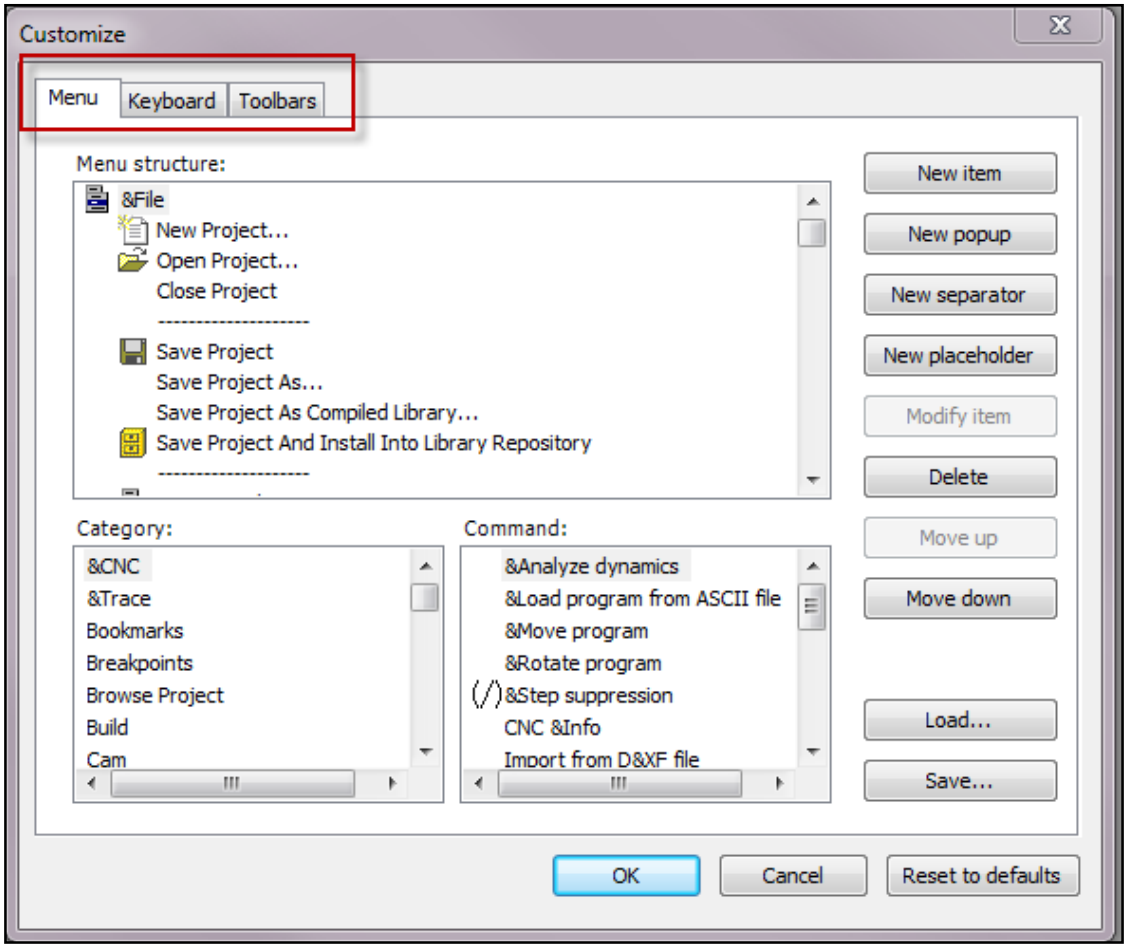
Work windows



Beijer ELECTRONICS

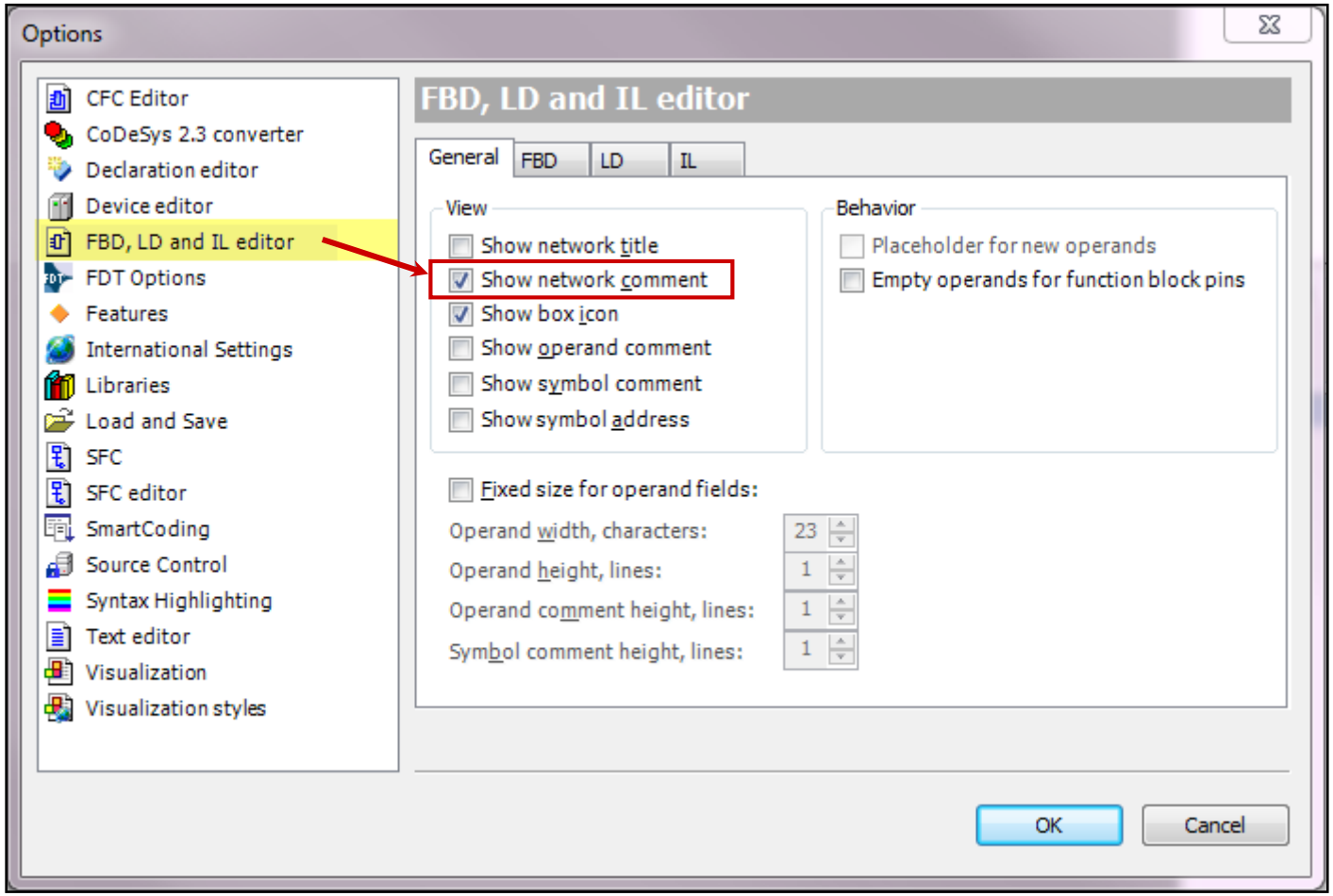
Customize the user interface

- Customize dialog, via menu selection Tools/Customize
- Sub-dialogs (tabs) for the configuration of Menu, Keyboard and Toolbars



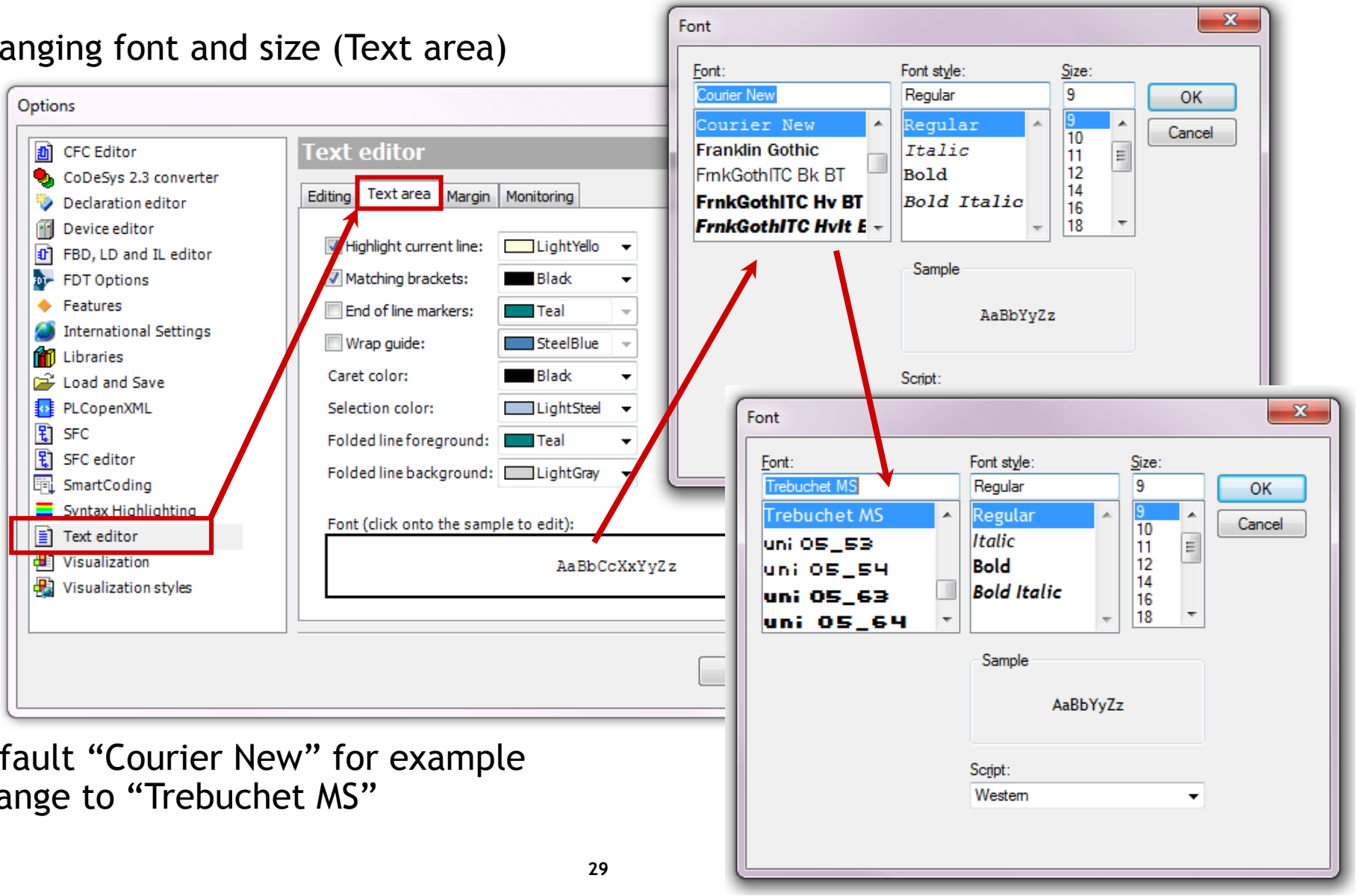
Options

- Menu selection Tools/Options, for user defined settings



Options example, change Text editor

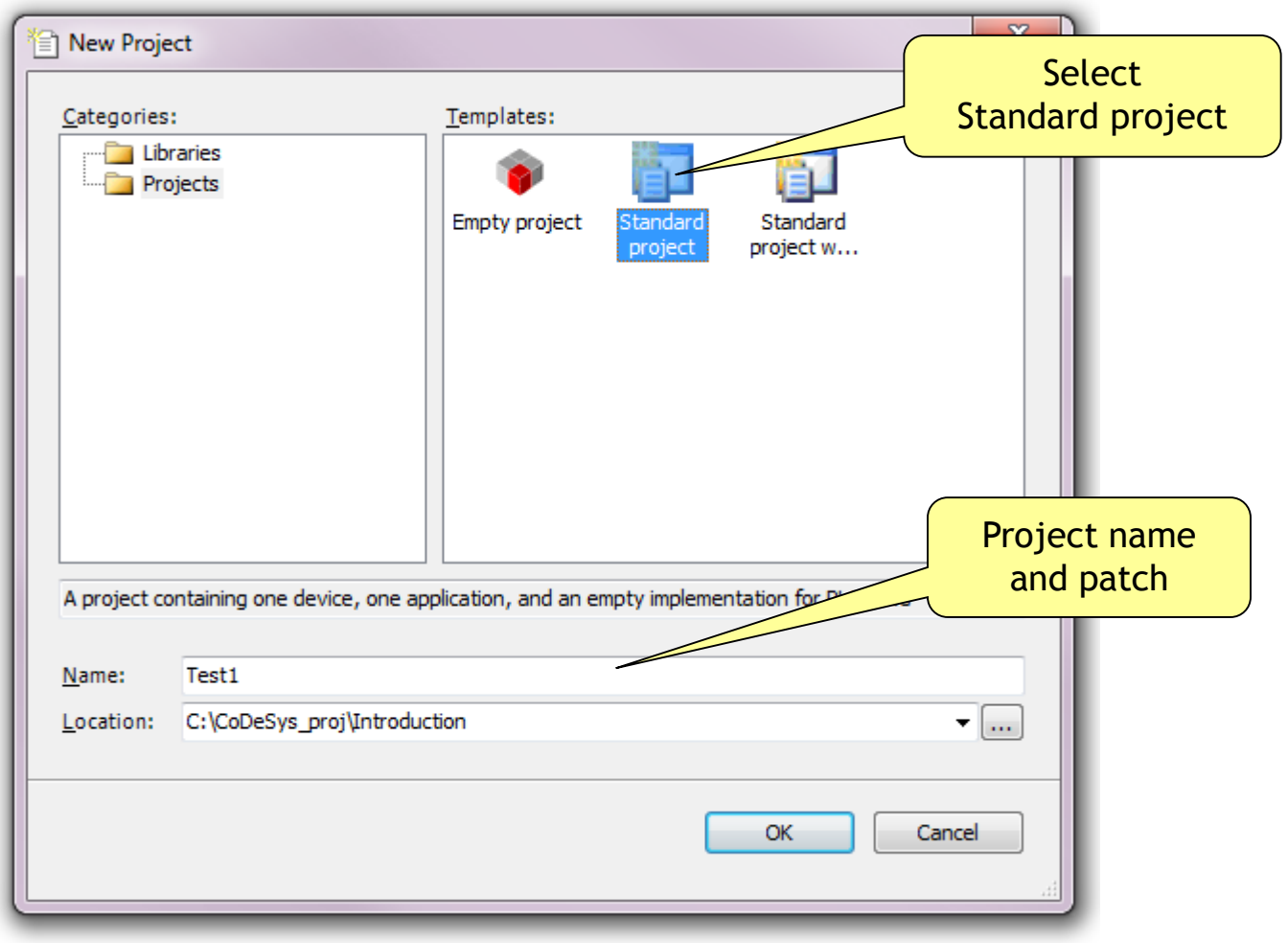
- Changing font and size (Text area)



- Default “Courier New” for example change to “Trebuchet MS”

Create new project

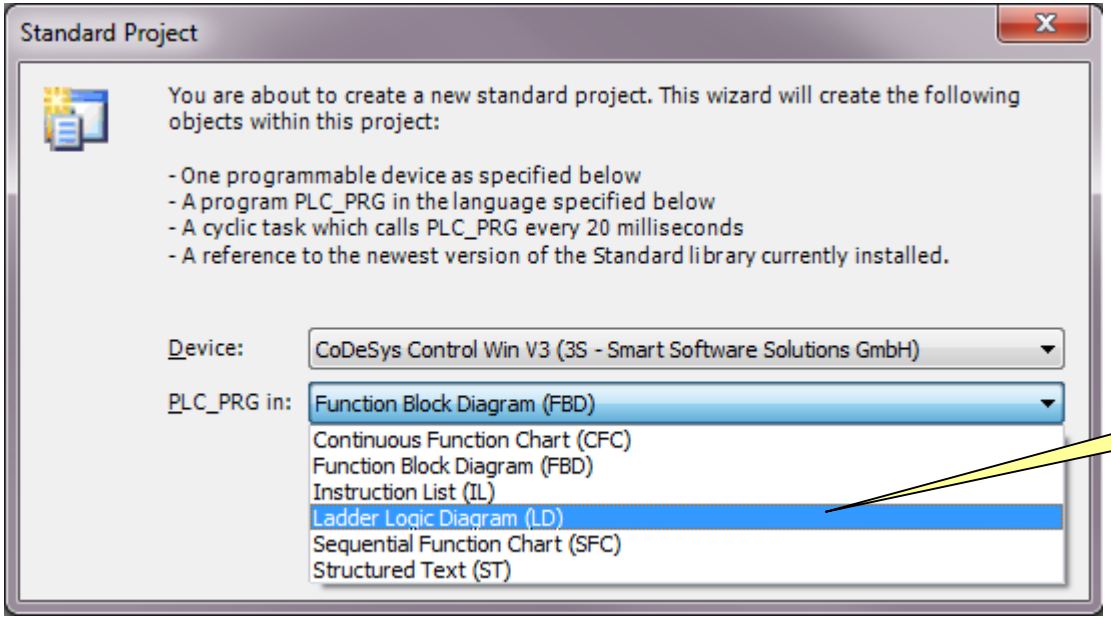
- File – New project… (CoDeSys V3.5)



Beijer ELECTRONICS

Create new project (wizard)

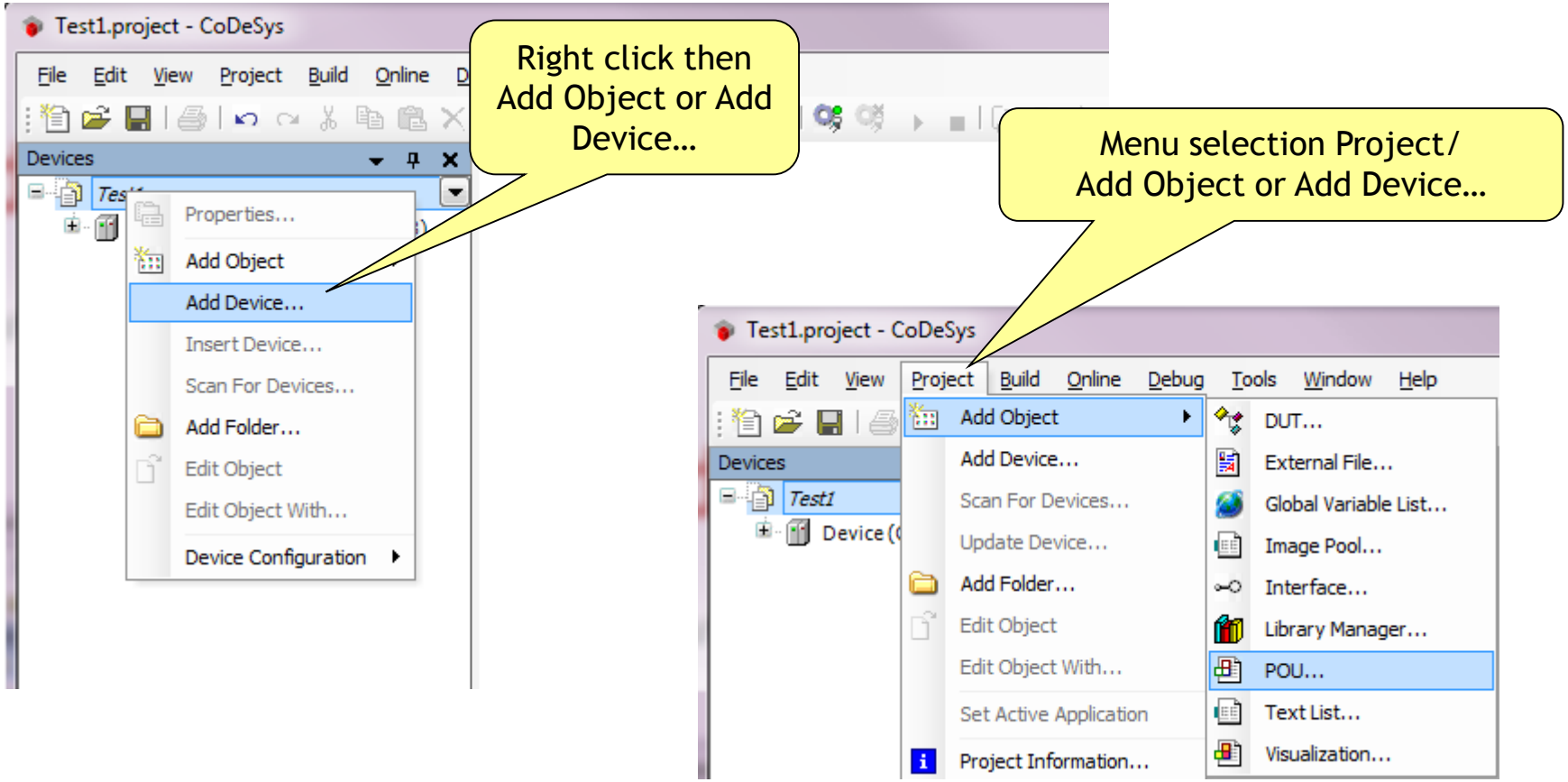
- When using the wizard a standard Device and Application (program) will be created automatically, select type of device and program editor
- Device: **CoDeSys Control Win V3** (Soft PLC)



Select LD-editor,
Ladder Logic Diagram

New Device and Object (without wizard)

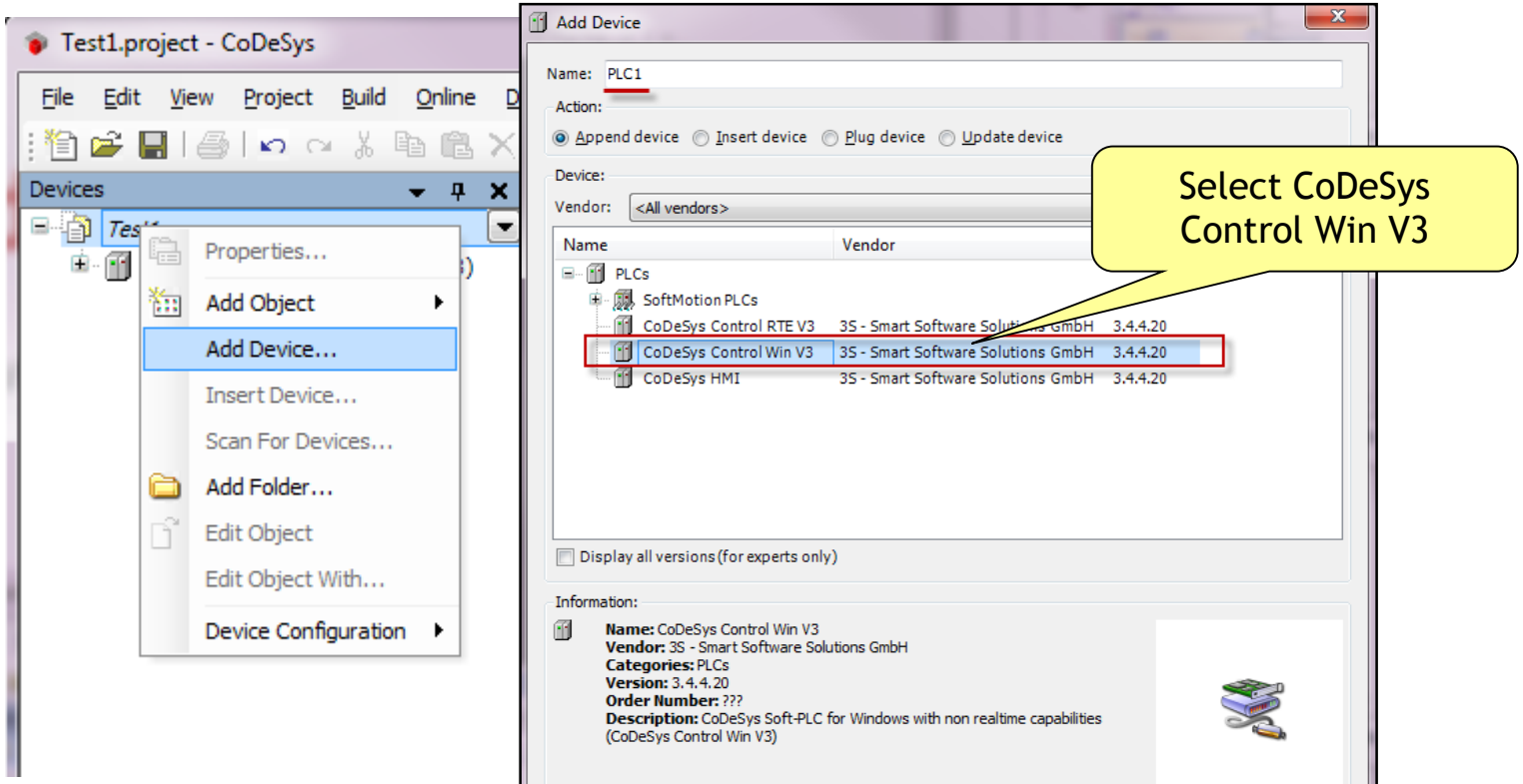
- Add Device or Object using context menu or menu selection



Beijer ELECTRONICS

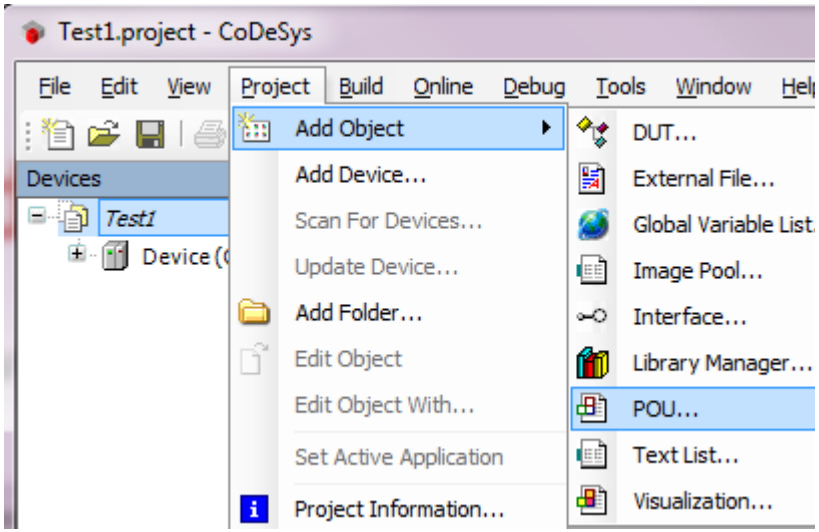
Adding device (without wizard)

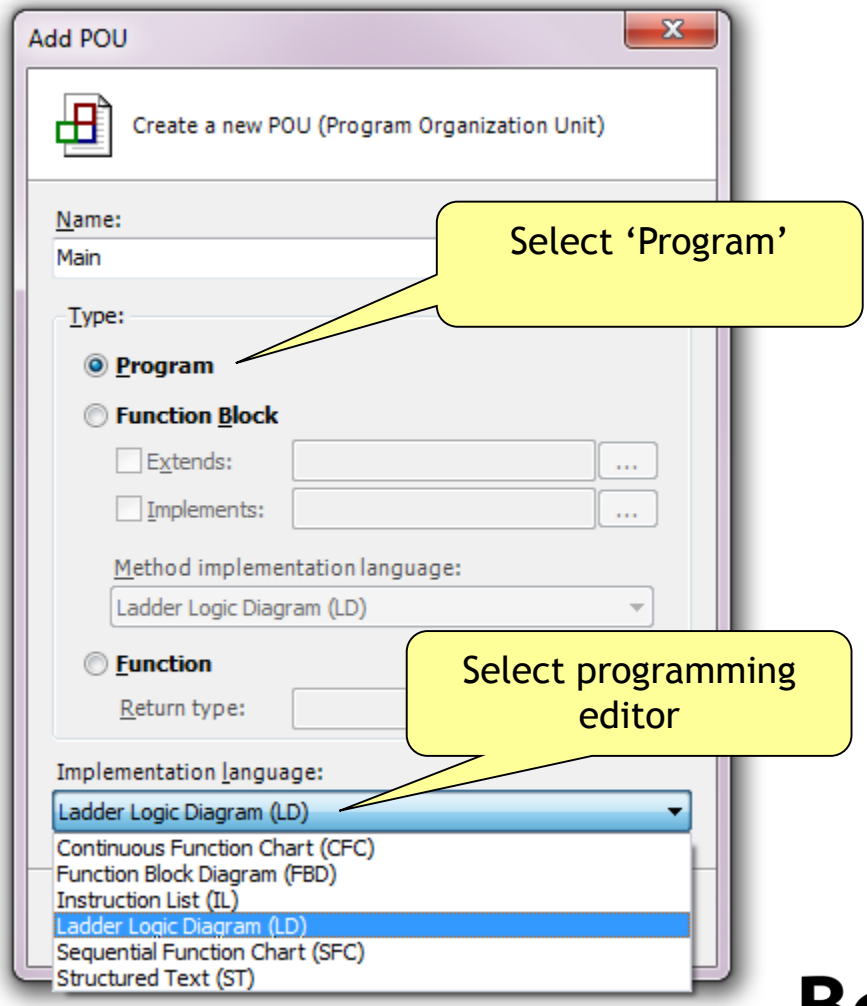
- Give a name to the device



Adding POU (without wizard)

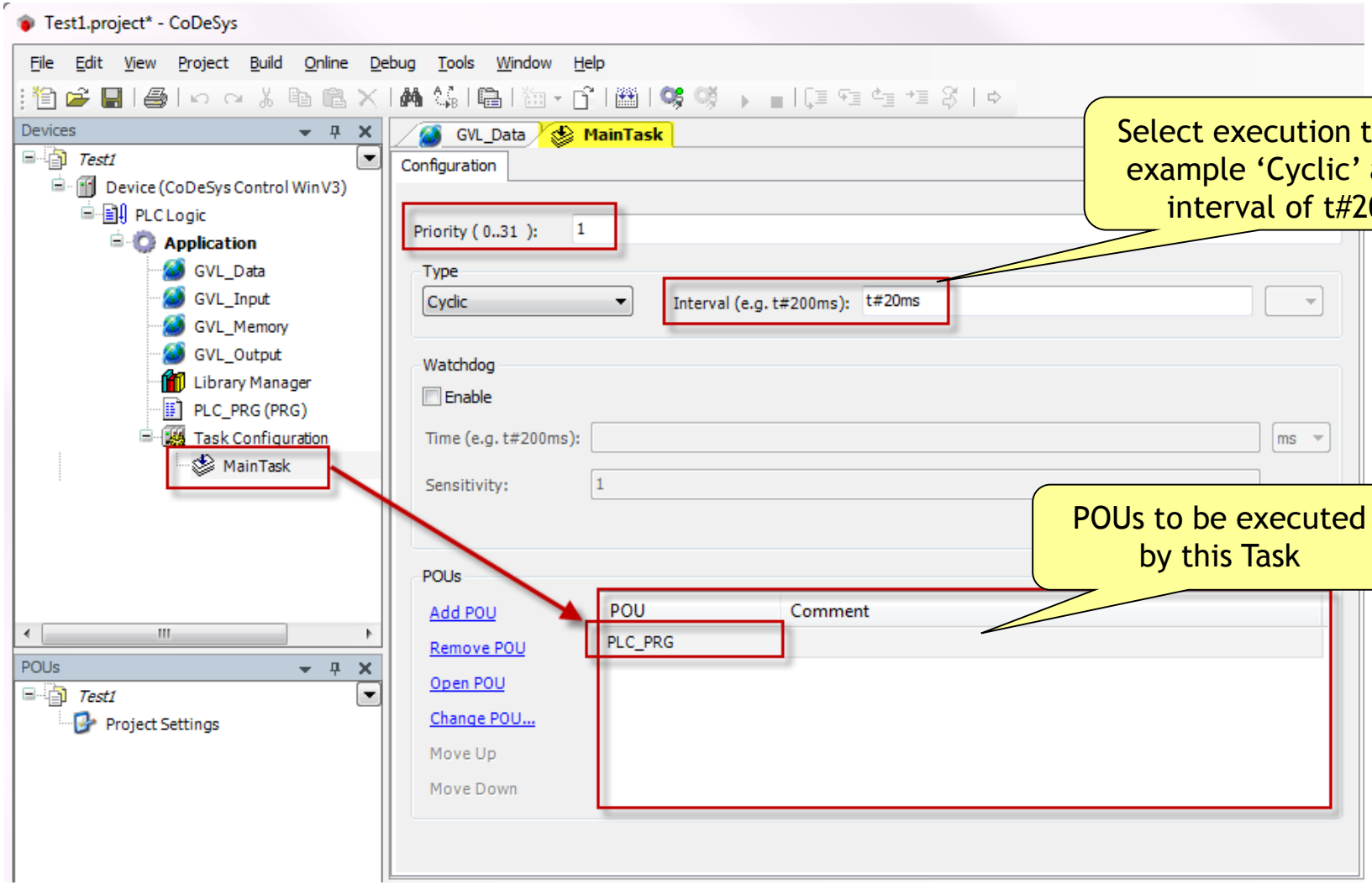
- Give a name to the POU and select programming language





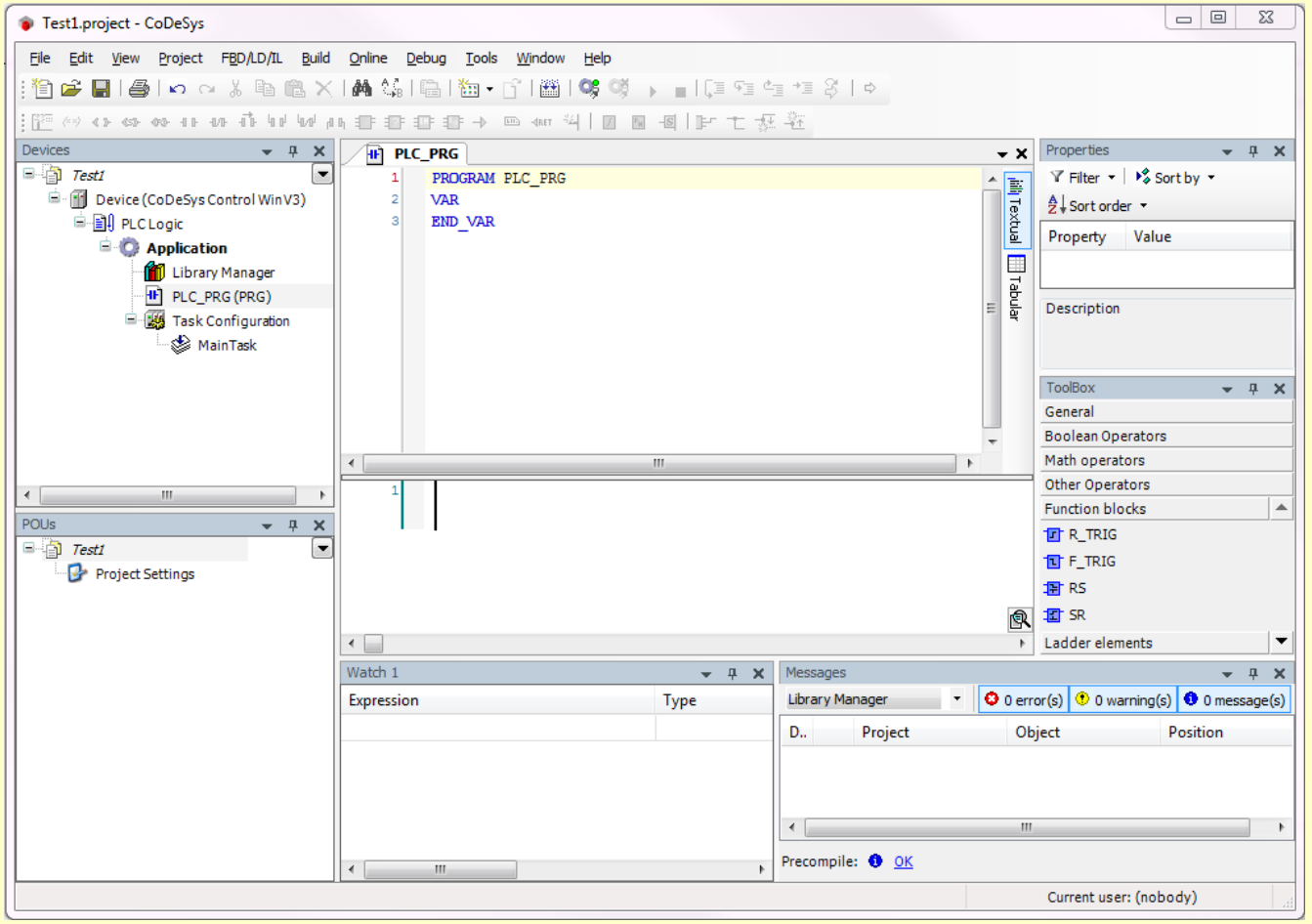
Append program to Task (without wizard)

- One or several programs (POUs) are connected to one Task

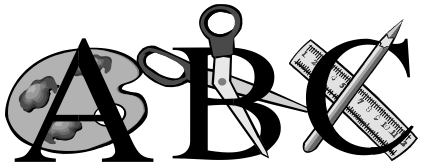


Exercise, Create a project

- Create new empty project with Device, POU and Task
- Try using toolbars and docking windows and check options menu



Beijer ELECTRONICS

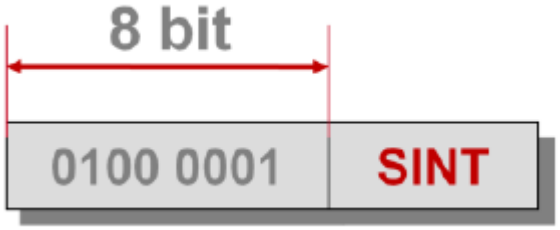


CoDeSys V3
Declaration

How to declare a variable?

- We need:
  - Variable name (Identifier) , Colon , Data type ,
    Initial value (optional) , Semicolon , **Comment (optional)**

```
siMyVariable : SINT := 65 ; (* init in dec *)
               (* := 16#41  init in hex *)
               (* := 2#0100_0001 init in binary *)
```



- The identifier in the example start with a prefix (si), that's the standard in samples from 3S (CoDeSys) showing that this is a Short Integer (si)
- Note, a list of "prefix" are given in the online help [F1] of CoDeSys, search for *Variable names* in chapter *Recommendations on the naming of identifiers*
- Each variable is assigned to a data type which defines how much memory space will be reserved and what type of values it stores
- The declaration can be done in the declaration part of a POU or via the Auto Declare dialog, as well as in a DUT or GVL editor

Variable naming restrictions

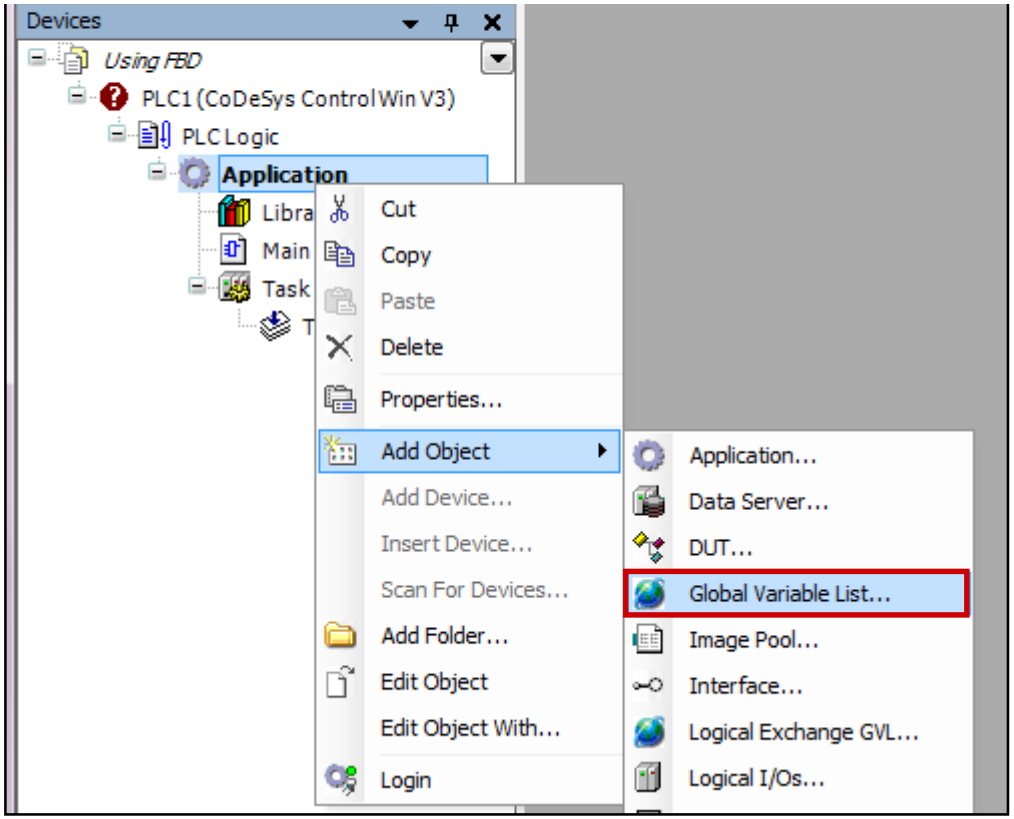
- Restrictions, the following identifier can be used:
  - No length limitation inside CoDeSys
  - Letters and numbers
  - Name must start with a letter
  - Only single underscores
  - Note that "A_BCD" and "AB_CD" are considered two different identifiers
    - Compare declaration of constants (and initial value)
  - Do not use spaces, or IEC keywords / operands, or special char: +, -, *,/,…
  - Not case-sensitive, which means that "VAR1", "Var1" and "var1" are all the same variable
  - An identifier must not be duplicated locally
  - An identifier can be declared with the same name in different GVL lists

- An instance path starting with “.” opens a global scope. So, if there is a local variable, for example “ivar” with the same name as a global variable “**.ivar**” the latter refers to the global variable (“.” is the global scope operator)

- Ignoring the restrictions above will result in a compile error!

Global or local variables

- When shall a global variable be used?
  - If it's used in more than one POU
  - If it's a physical in/output address
  - If it will be monitored by HMI or Scada



- Using variable names makes it more easy to understand and to maintain the project

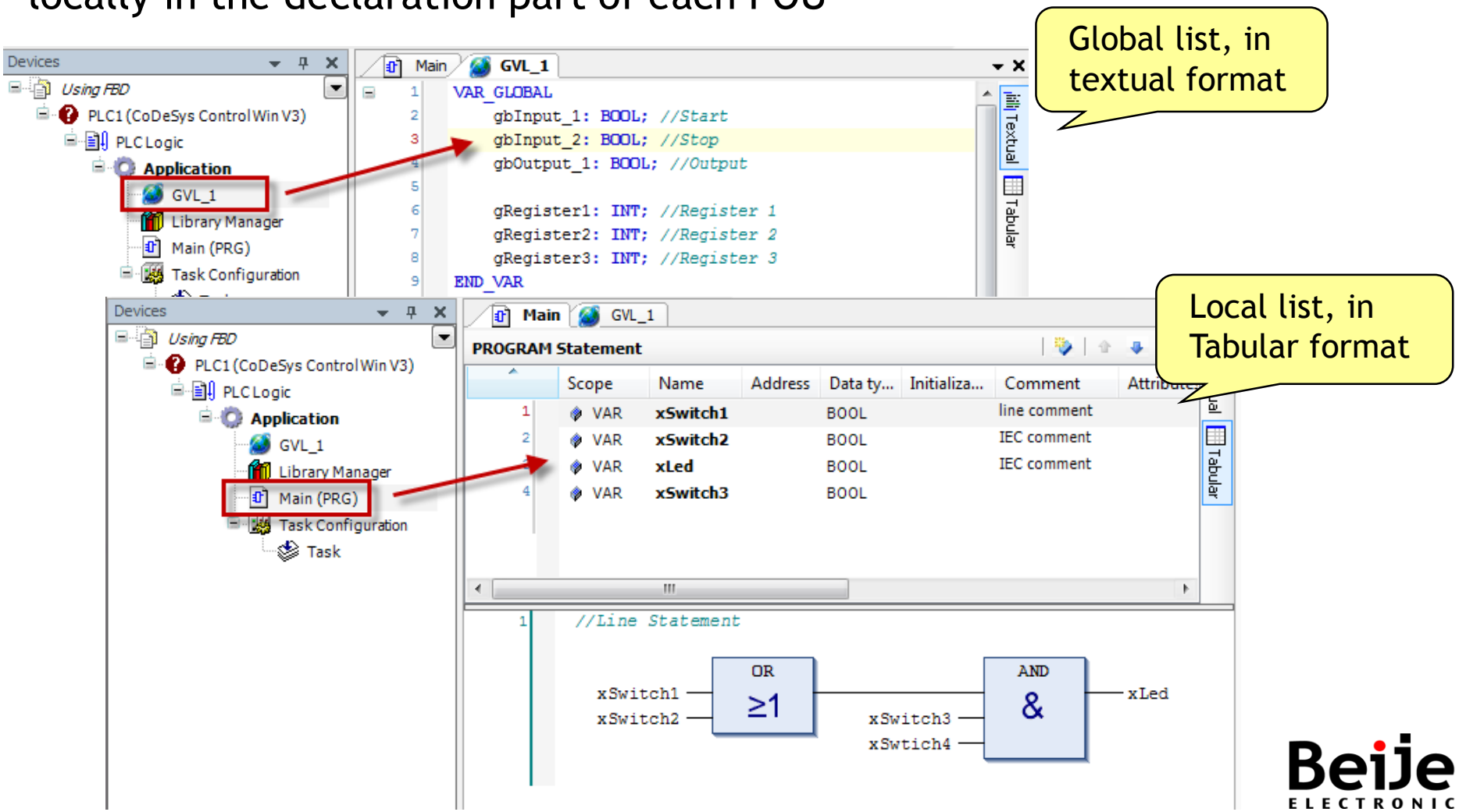
- **Global Variable names**, can be used in more than one POU
- **Local Variable names**, can only be used in one POU

- Note, a feature in CoDeSys is that multiple declaration of variable names are supported using the name of the GVL as a namespace for the included variables, example:

```
globlist1.ivar := globlist2.ivar;
(* ivar from GVL globlist2 is copied
to ivar in GVL globlist1 *)
```

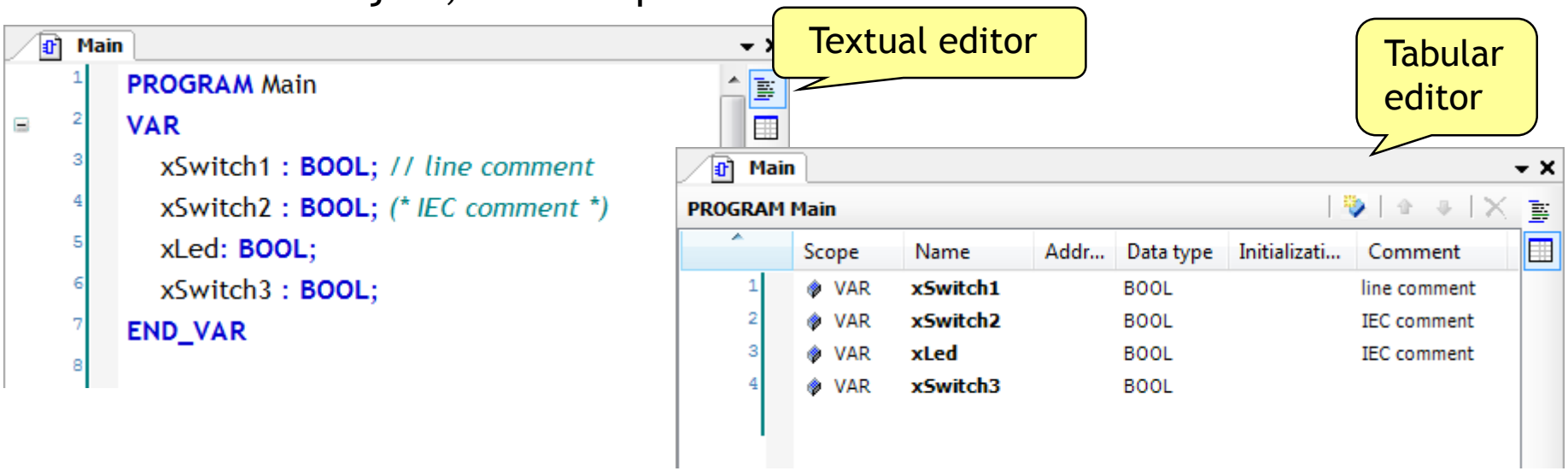
Declaration of variables

- Declare variables either globally in a Global Variable List or locally in the declaration part of each POU



Declaration of variables

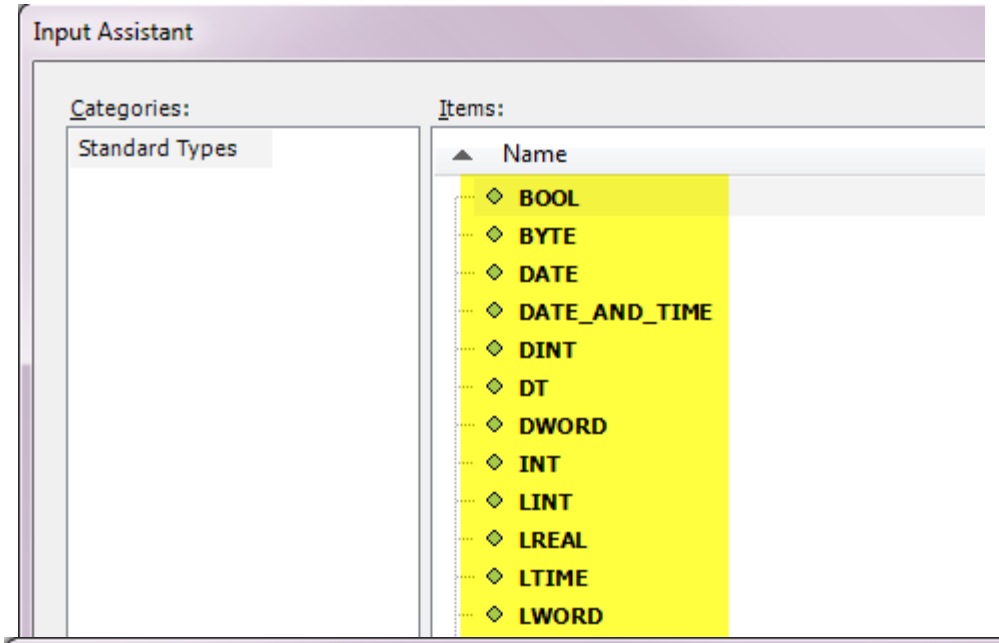
- Declaration can be made either in ‘Textual’ or ‘Tabular’ editor of a POU object, for example:

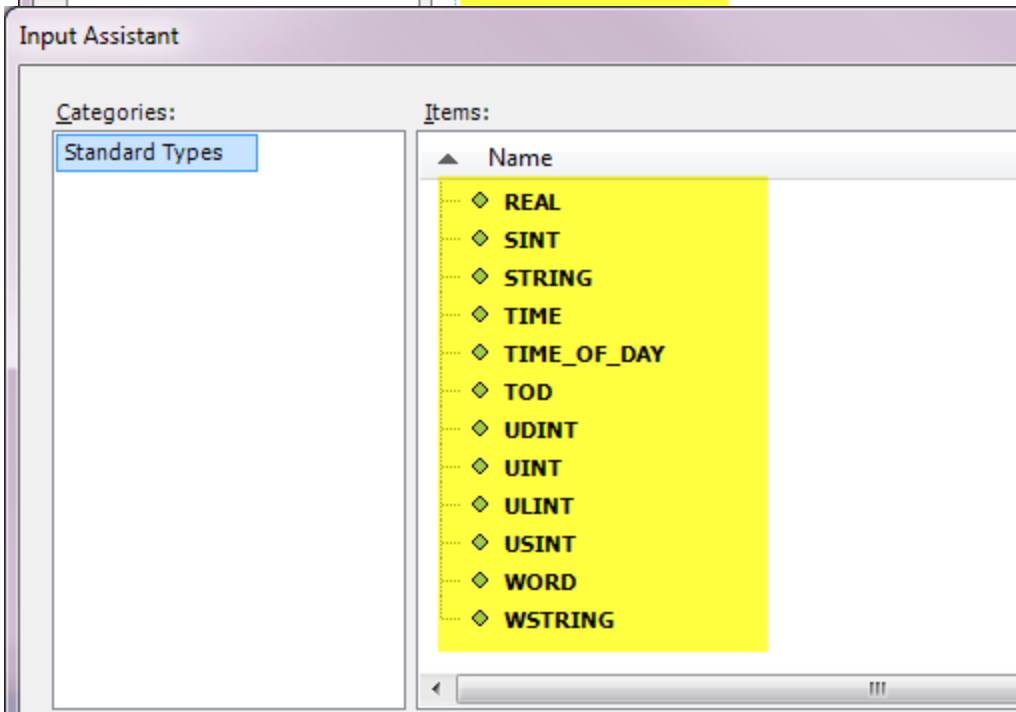


- Use standard data types, user defined data types (DUT = Structure, Enumeration, Alias and Union) and instances of function blocks
- Remanent Variables - RETAIN, PERSISTENT
- Attribute keywords RETAIN , PERSISTENT and CONSTANT can be added to the declaration of the variables "type" in order to specify the scope
- Each variable is assigned to a data type which defines how much memory space will be reserved and what type of values it stores

Standard Data Types

- **BOOL** (x, prefix)
  1 bit, Boolean; in-/outputs or Memory bits
- **INT** (i) - Integer 16-bit, with sign-bit
- **DINT** (di)- Double Integer 32-bit, with sign-bit
- **WORD** (w) - Word Unsigned 16-bit
- **DWORD** (dw) - Double Word Unsigned, 32-bit
- **TIME** (tim) - 16-bit, without sign-bit
- **ARRAY** (a) - Array with index up to 3 dimensions
- **REAL** (r) - 32-bit floating point
- **STRING** (s) - Character strings





More of data types in CoDeSys

- Data types in general

  BYTE – 8 bit

  LWORD - 64 bit Long Word

  SINT - Short Integer, 8 bit, with sign-bit

  LINT – Long Integer 64 bit, with sign-bit

  U – use the prefix U to make it unsigned byte or integer, for example USINT

  TIME – 16 bit, without sign-bit

  ARRAY – Array with index up to 3 dimensions

  STRING - character strings

Data type	Lower limit	Upper limit	Memory space
BYTE	0	255	8 Bit
WORD	0	65535	16 Bit
DWORD	0	4294967295	32 Bit
LWORD	0	2^{64}-1	64 Bit
SINT	-128	127	8 Bit
USINT	0	255	8 Bit
INT	-32768	32767	16 Bit
UINT	0	65535	16 Bit
DINT	-2147483648	2147483647	32 Bit
UDINT	0	4294967295	32 Bit
LINT	-2^{63}	2^{63}-1	64 Bit
ULINT	0	2^{64}-1	64 Bit

REAL- 32 bits Real (1.175494351e-38 to 3.402823466e+38)

LREAL – 64-bits Real (2.2250738585072014e-308 to 1.7976931348623158e+308)

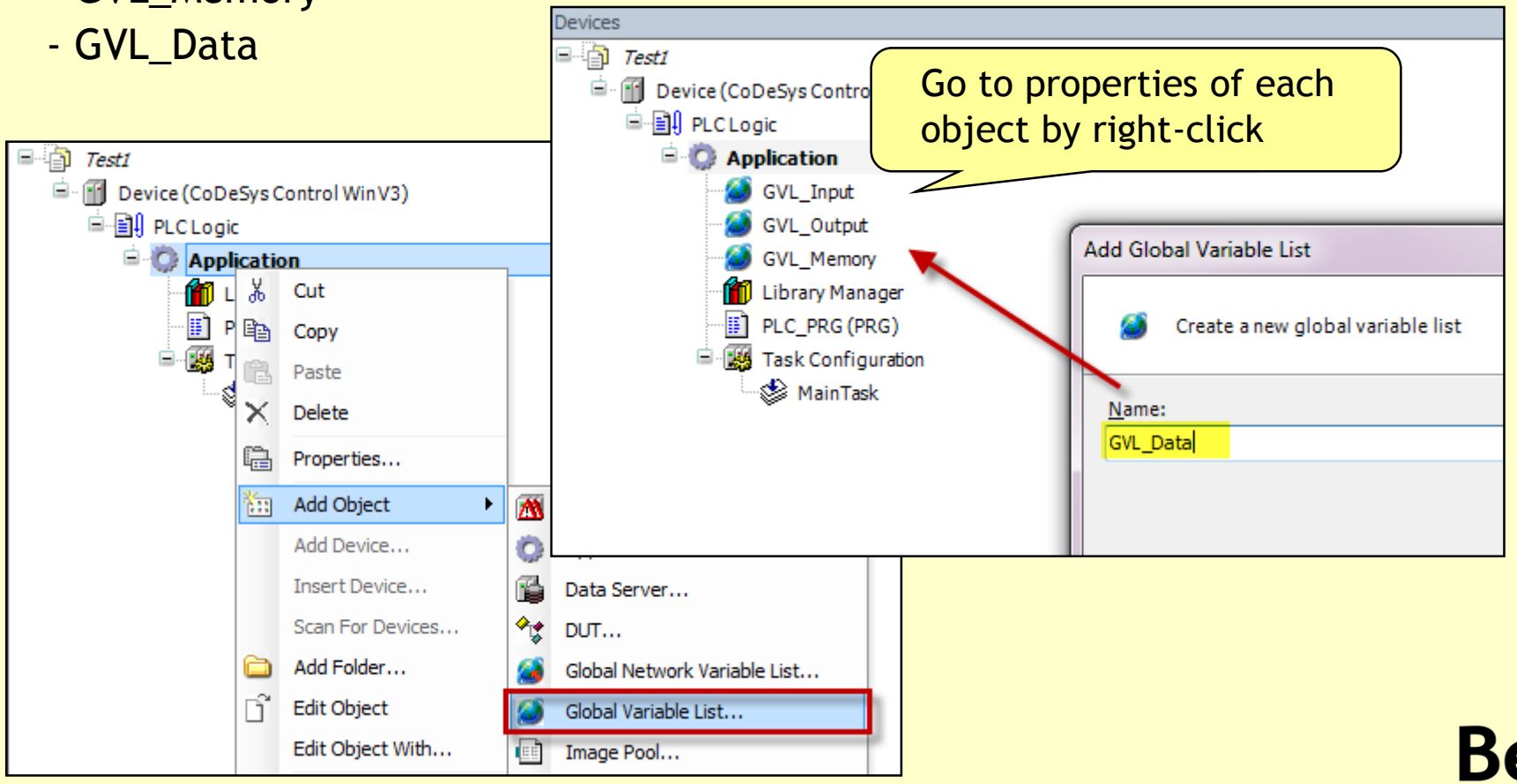
Classes IEC 61131-3

- Three classes
  - **Program**
    - **Program** *(PRG)*
    - **Function** *(FUN)*
    - **Function Block** *(FB)*
  - **Datatype**
    - **Simple type** *(Standard type)*
    - **Function Block**
    - **Data Unit Type** *(Struct)*
  - **Variable**
    - **Global** (VAR_GLOBAL; attribute CONSTANT, RETAIN, PERSISTENT)
    - **Local** (VAR; attribute CONSTANT, RETAIN, PERSISTENT)
    - **FB** (VAR, VAR_INPUT, VAR_OUTPUT,VAR_IN_OUT ; attribute CONSTANT, RETAIN, PERSISTENT)
- Data type classes
  - **Simple type**
    - **Bool**
    - **Int / Word**
    - **Dint**
    - **Array of...**
  - **Function block**
    - **MyBlock**
    - **CounterBlock**
  - **Data unit type**
    - **MyDataStructure**
    - **ProductionResult**
    - **Weather**

    Each of the above:
    - **Bool**
    - **Int / Word**
    - **Dint**
    - **Array of...**

Exercise, Global variable lists

- Creating variable lists
  - GVL_Input
  - GVL_Output
  - GVL_Memory
  - GVL_Data



Exercise, Declare inputs

- Declare some Input variables to the ’GVL_Input’ list

GVL_Input

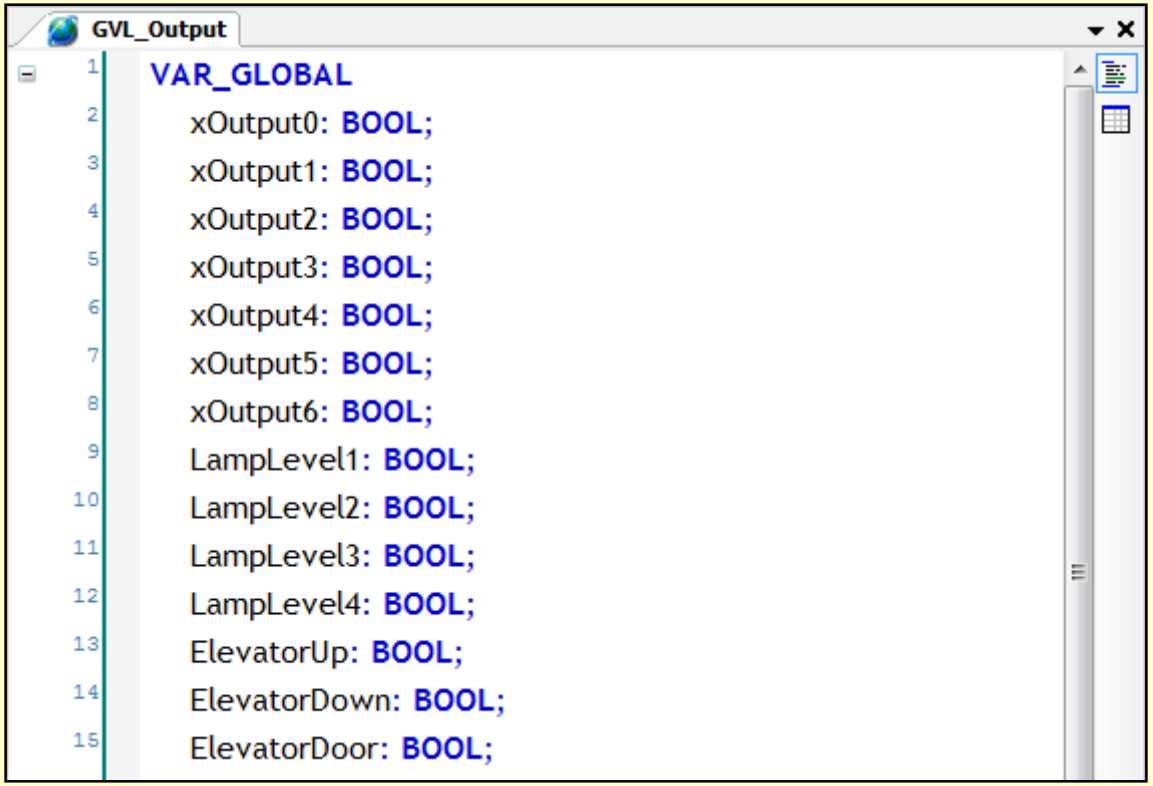
```
1  VAR_GLOBAL
2  END_VAR
3  VAR_GLOBAL
4     xInput0: BOOL; // Comment text 1
5     xInput1: BOOL; // Comment text 2
6     xInput2: BOOL; // etc...
7     xInput3: BOOL;
8     xInput4: BOOL;
9     xInput5: BOOL;
10    xManualAuto: BOOL;
11    ButtonLevel1: BOOL;
12    ButtonLevel2: BOOL;
13    ButtonLevel3: BOOL;
14    ButtonLevel4: BOOL;
15    SensorLevel1: BOOL;
16    SensorLevel2: BOOL;
17    SensorLevel3: BOOL;
18    SensorLevel4: BOOL;
19    ServiceReset: BOOL;
20 END_VAR
```

Some of the variables will be used for the Elevator Simulator in later exercises

Tip!
Import from text file, GlobalVariableList.txt

Exercise, Declare outputs

- Declare Output variables to the ’GVL_Output’ list



Tip!
Import from text file,
GlobalVariableList.txt

Beijer ELECTRONICS

Numeric Data

- The programming tool uses binary, octal, decimal and hexadecimal bases as shown in this table
  - Binary (base 2)
  - Octal (base 8)
  - Decimal (base 10)
  - Hexadecimal (base 16)

- Tip! Use the calculator on the computer to translate between different numerical bases
  - Run ‘Calc’

Binary	Octal	Decimal	Hexadecimal
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	**17**	**15**	**F**
10000	20	16	10
10001	21	17	11
			etc

Numbering Systems (Constants)

- 2#10011011 (bin) = 8#233 (oct) = 155 (dec) = 16#9B (hex)

	128	64	32	16	8	4	2	1	
2#	1	0	0	1	1	0	1	1	= 128+16+8+2+1=155
	2	1	4	2	1	4	2	1	
2#	1	0	0	1	1	0	1	1	= 2 | 2+1 | 2+1= 8#233
	8	4	2	1	8	4	2	1	
2#	1	0	0	1	1	0	1	1	= 8+1 | 8+2+1=16#9B

- These numeric values can be of data type BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL and LREAL

Constants in IEC 61131-3

- Decimal constants have no prefix
  82, -16000, 238, 1_234_667_778
- Hexadecimal constants have the prefix **16#**
  16#1A, 16#111, 16#3A0F, 16#3A_0F
- Octal constants have the prefix **8#**
  8#15, 8#707,
- Binary constants have the prefix **2#**
  2#1100, 2#1, 2#11011011, 2#1101_1011
- Floating constants
  3.141593, 1.43E-12, -1.75E-22, -12.0, -REAL#12
- Time constants
  T#1h20m, TIME#80m, T#500ms
- Time of day constant
  TOD#16:56:34, TIME_OF_DAY#16:56:34

238

16#FF0F

8#707

2#1011011

-1.75E-22

```
VAR_GLOBAL CONSTANT
   Constant1: INT := 2#1001_0110;
   Constant2: TIME := T#104S;
END_VAR
```

Display format

- Tip! Define display format upon declaration

```
PROGRAM ST_display_mode
VAR
    {attribute 'displaymode' := 'dec'}
    iDec: INT := 1333;
    {attribute 'displaymode' := 'hex'}
    iHex: INT;
    {attribute 'displaymode' := 'bin'}
    iBin: INT;
END_VAR
```

Device1_PLC.Application.ST_display_mode

Expression	Type	Value	Prepared value	Comment
iDec	INT	1333		
iHex	INT	16#0604		
iBin	INT	2#0000011000000100		

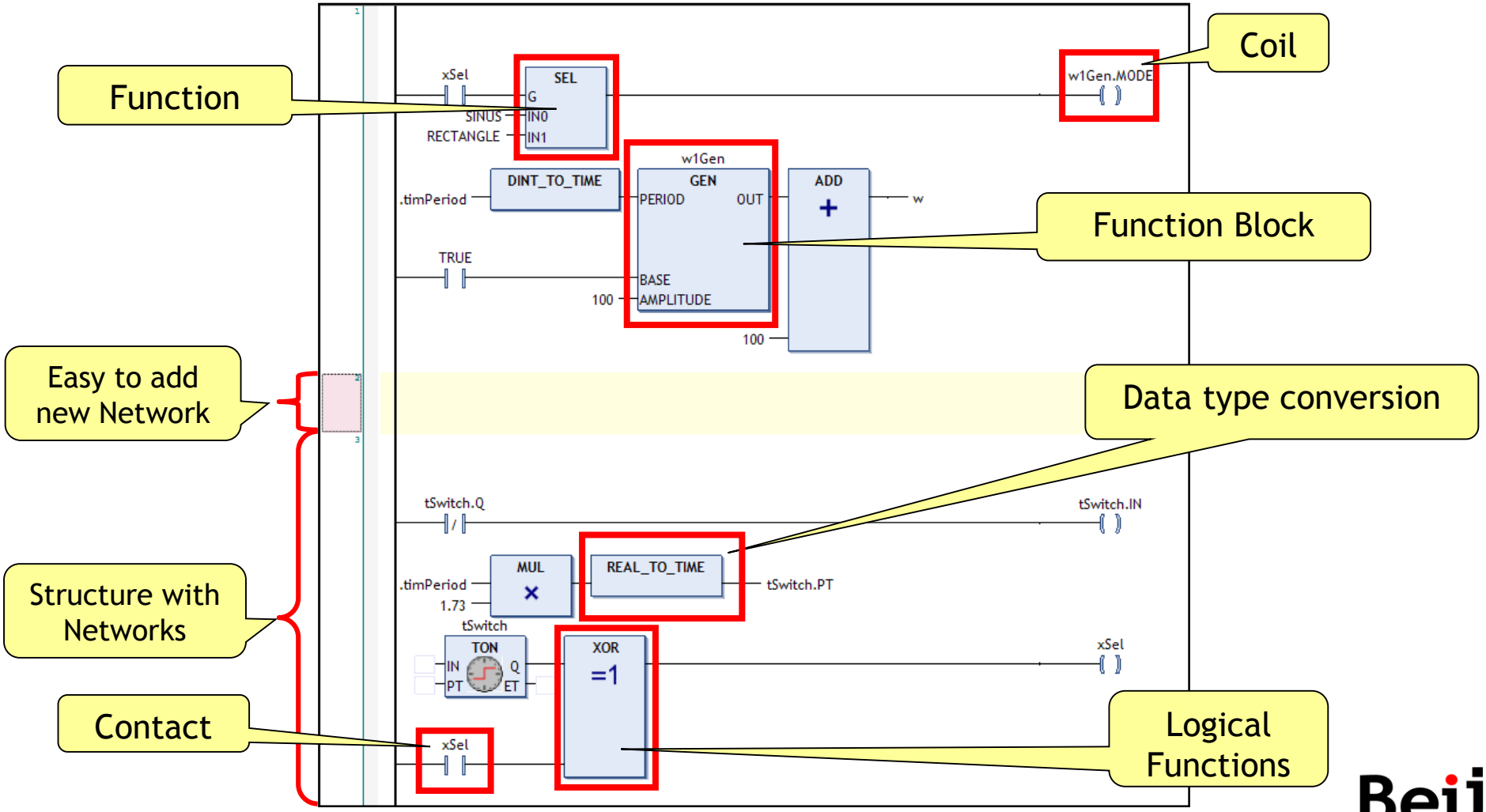
```
1  (* Select displaymode BY declaration *)
2  iDec 1333 := iHex 16#0604 := iBin 2#0000011000000100 := iDec 1333 +1;
3
```


Beijer ELECTRONICS

CoDeSys V3
Ladder logic

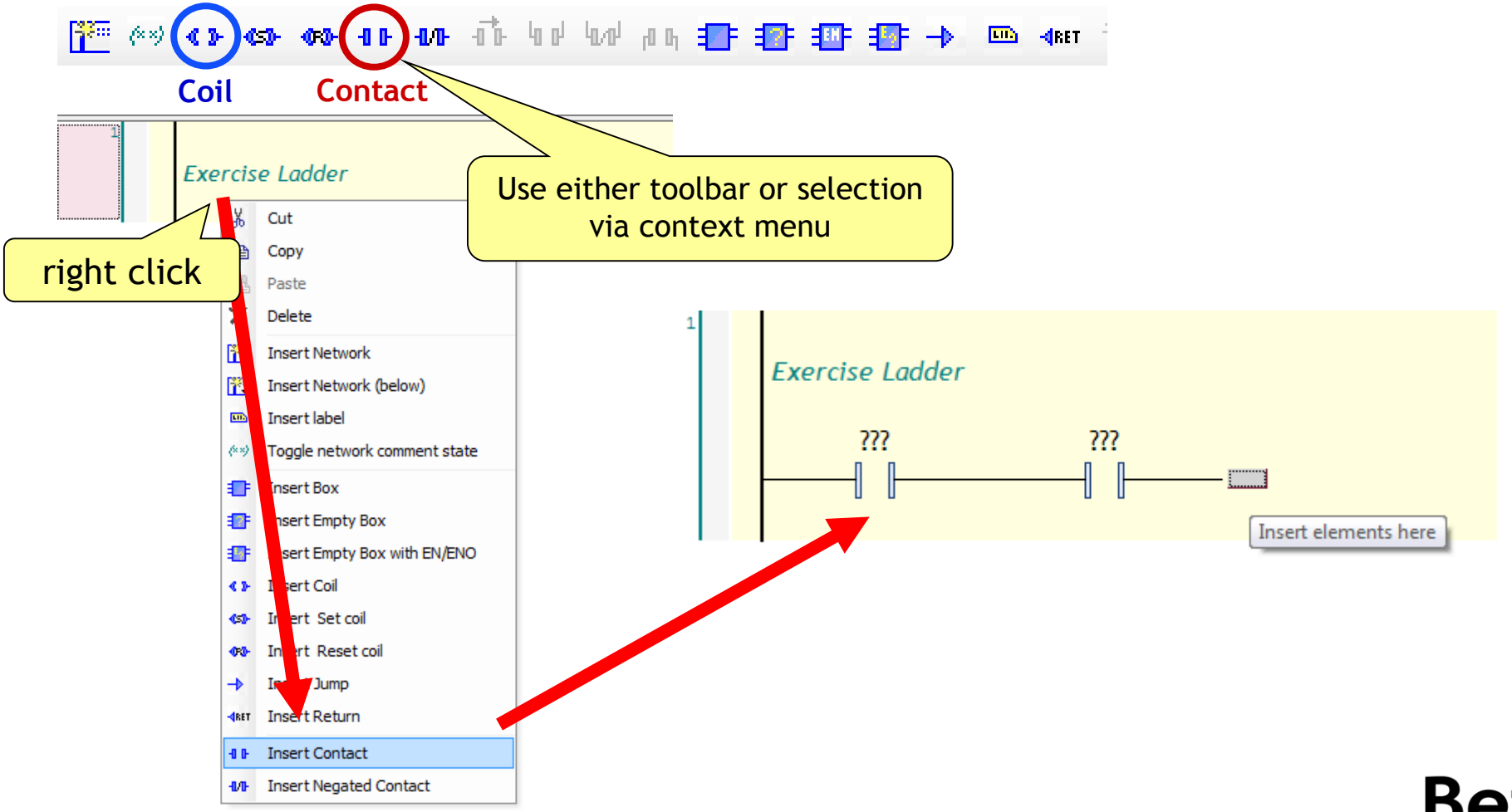
Creating Program Components

- The following Ladder example highlights the major features



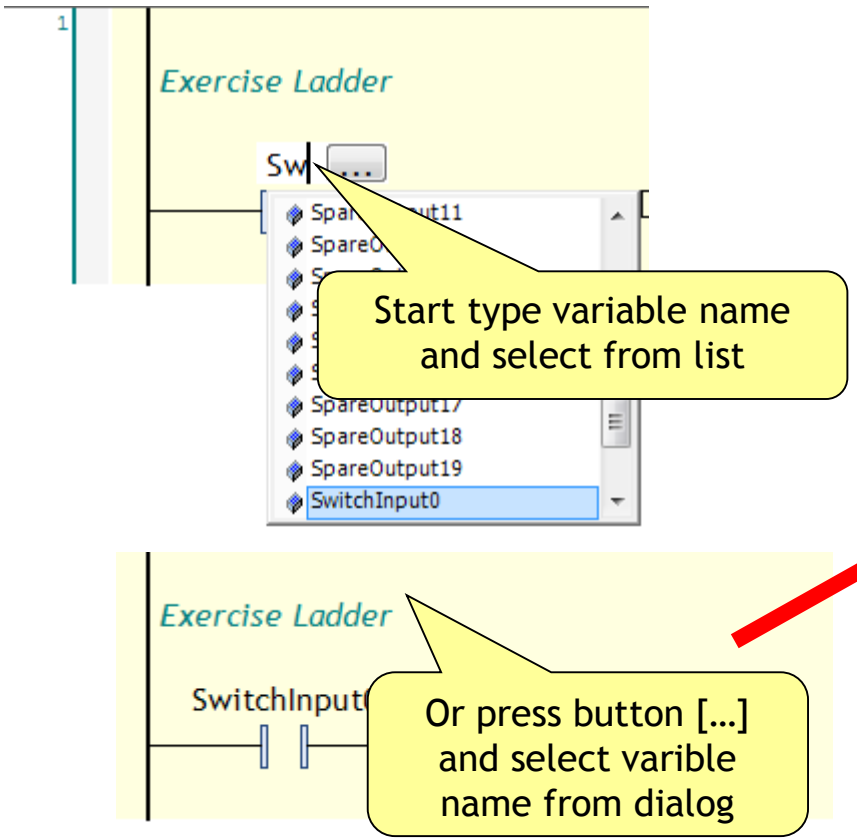
Ladder logic editor

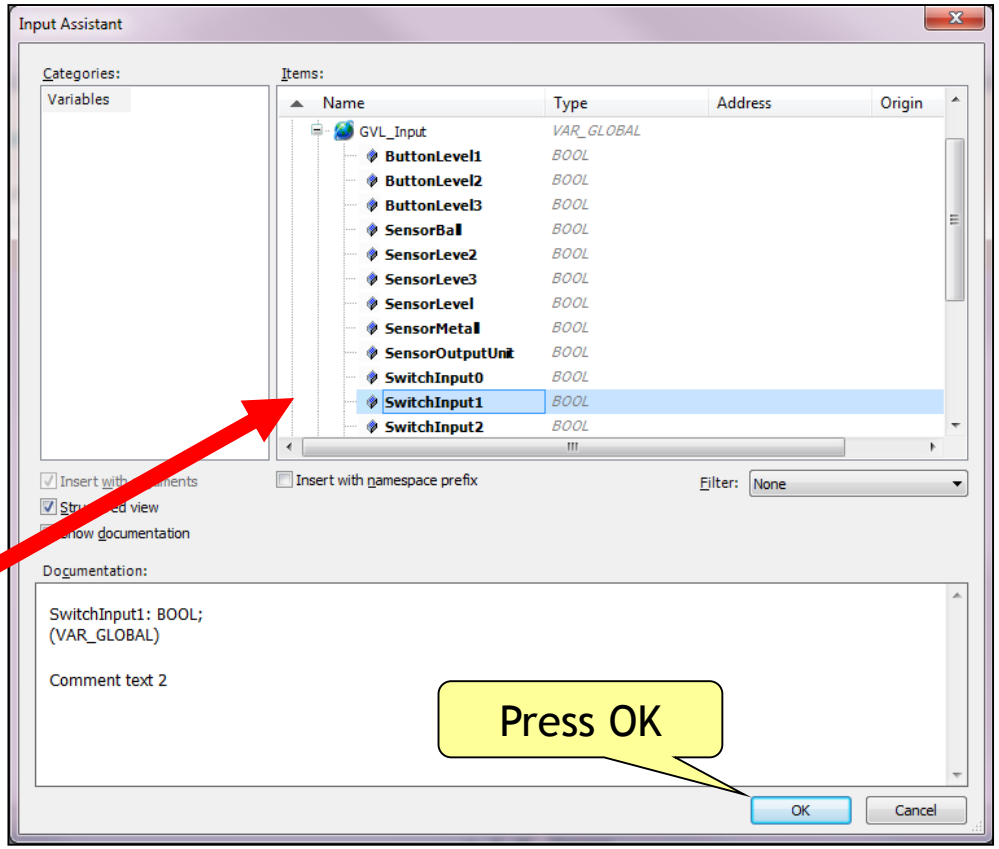
- Click (Contact) on the Ladder toolbar
  - and then click a desired position to position a Contact there



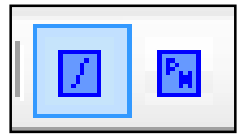
Ladder logic editor

- Add variable names to objects
  - This can be done via type ahead or via dialog

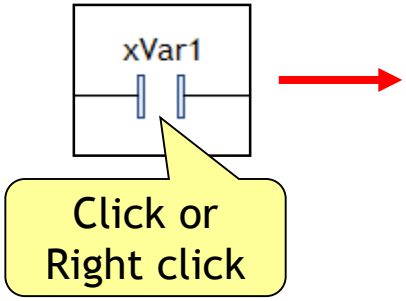


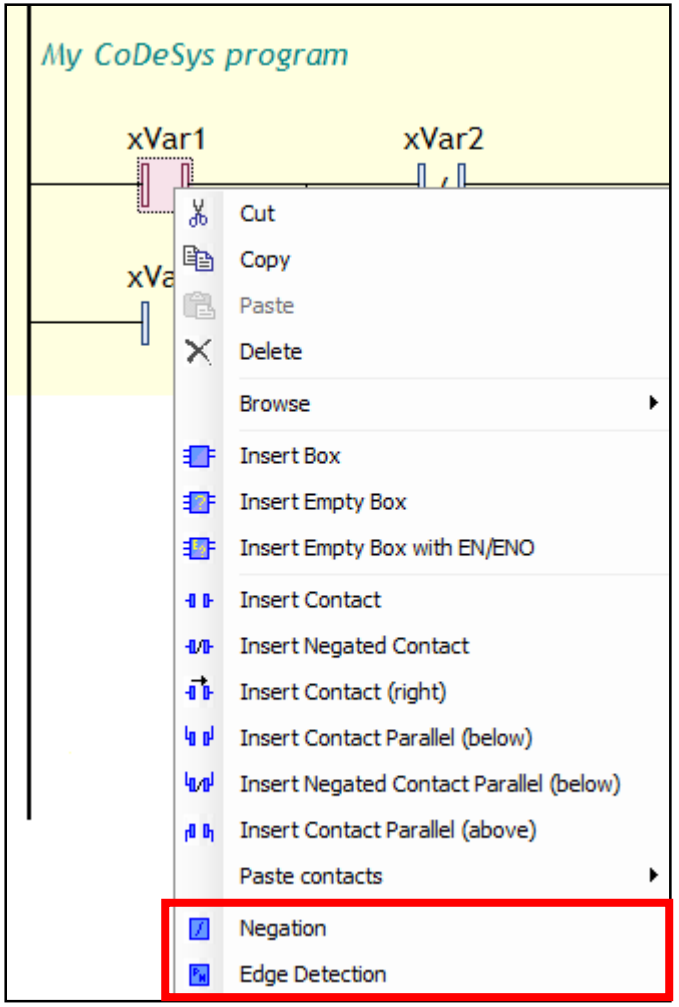


Ladder - Negation / Edge detection

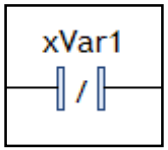


- Change Contact type, using toolbar or context menu

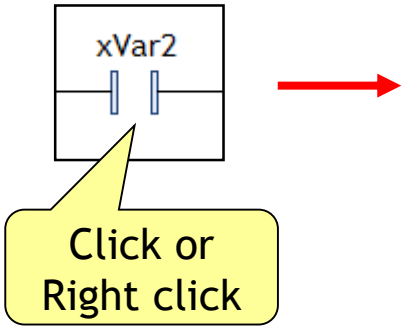




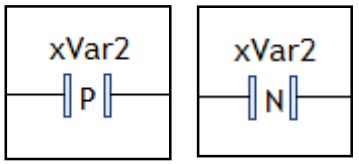
Negation



- Change Contact type



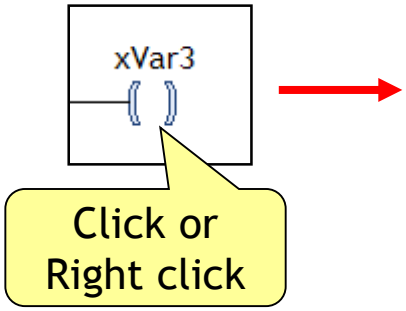
Edge detection

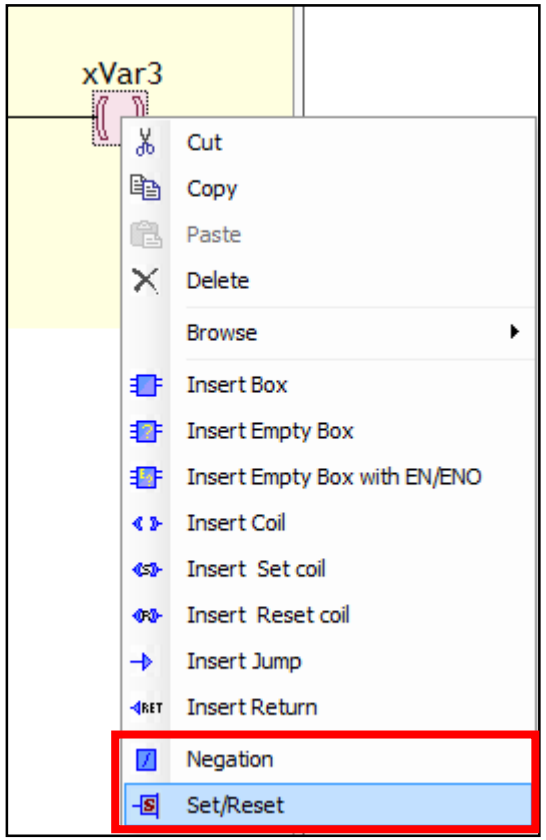


Beijer ELECTRONICS

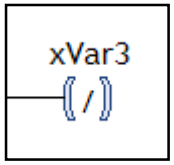
Ladder - Negation / Set or Reset

- Change Coil type, using toolbar or context menu

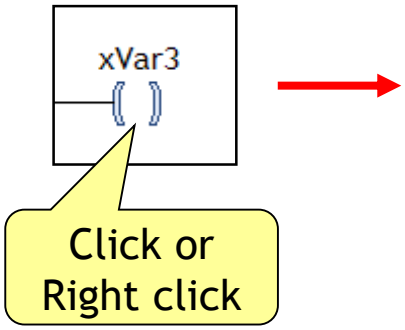




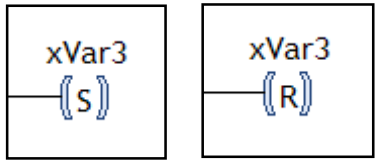
Negation



- Change Coil type

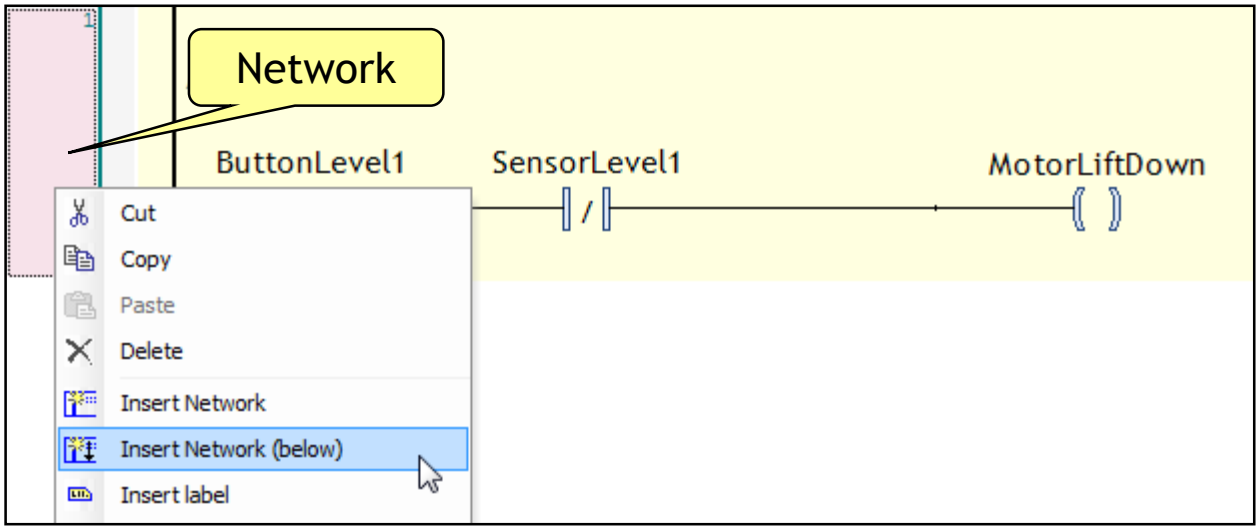


Set or Reset



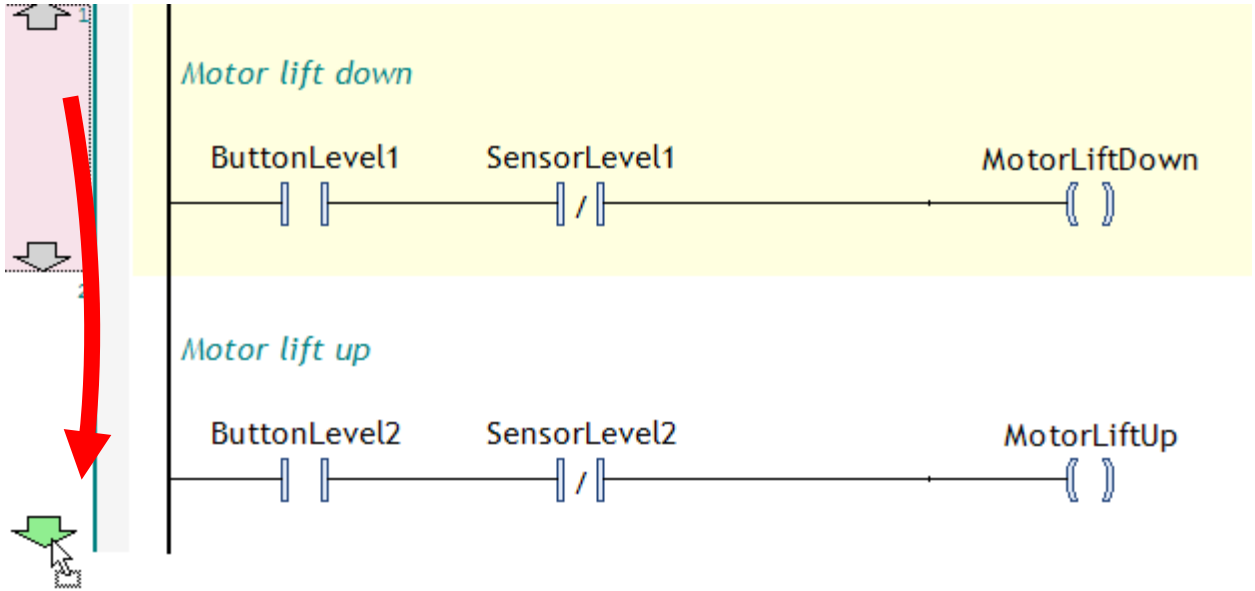
Adding Networks

- Select a network, and right-click
- Now select ’Insert Network (below)’



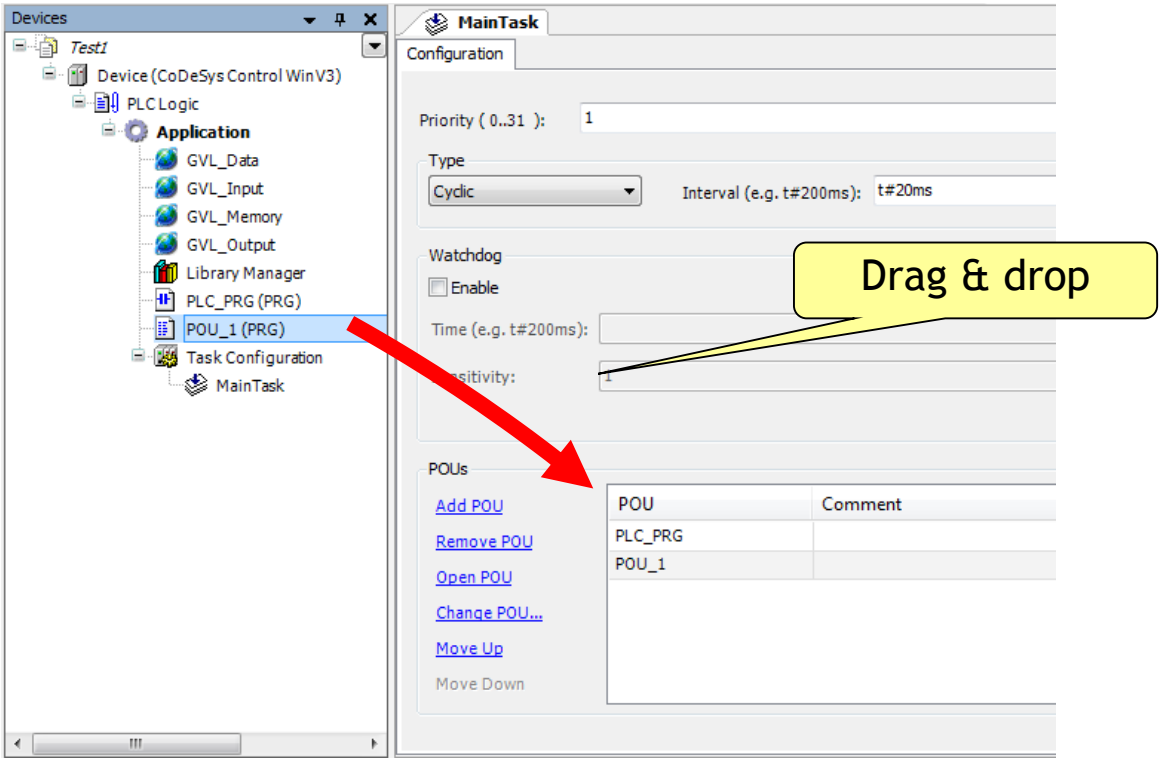
Drag and Drop / Cut-Copy-Paste

- Drag an existing network, and drop it to a new position
- While pressing the [Ctrl] key to copy the existing network
- Copying/moving ladder blocks using the clipboard
  - Code can be copied by the general menu options or shortcut keys using the clipboard

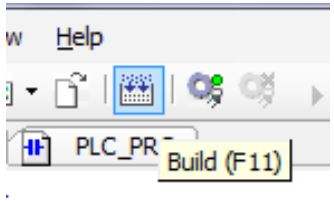


Drag and Drop objects

- To move programs (for example “POU_1”) into the desired Task configuration ‘Drag & drop’ can be used
- Once programs are assigned an execution type, they will get default parameters automatically



Build (Rebuild) project



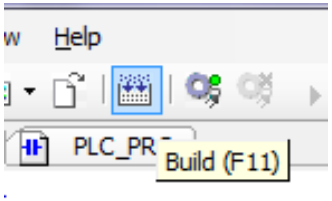
- ‘Build’ shortcut [F11], starts the build process of the active application
  - » All objects belonging to the application will be syntactically checked
  - » Notice that no compilation code will be generated, like it will be done when ‘login’ or ‘download‘ an application!
  - » The build process is done automatically before each ‘login’ with changed application program

- The syntactical check will give error messages or warnings
  - » These are displayed in the ‘Message’ view of category "Build“
  - » Max. no of displayed errors/warnings is 500

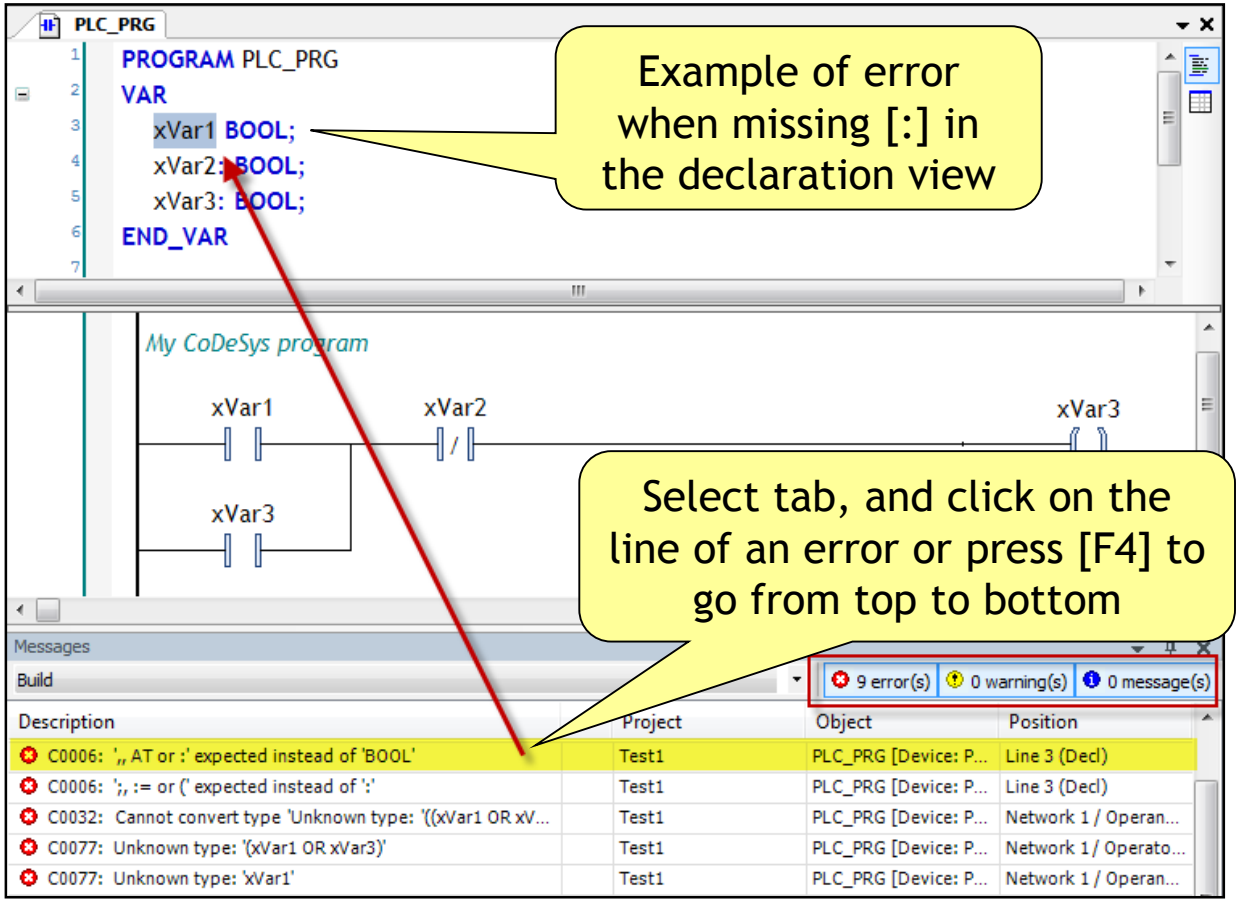
- If the program has not been changed since the last build-process, and no errors were detected, it will not be built again
  - » The message "The application is up to date" will be displayed

- To get the syntactical checks done again, do Rebuild

Build - Message view

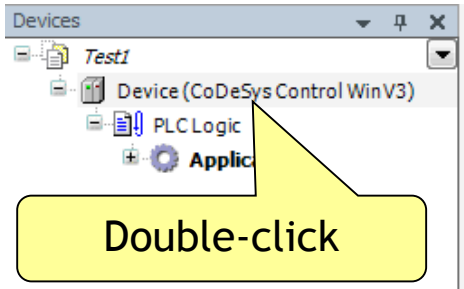


- If the build process will generate errors, warnings or messages please check the ‘Messages’ view
  - » Commands are available for navigating between messages and source code

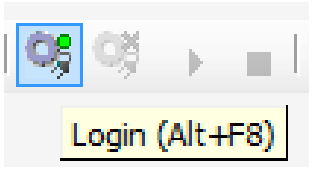


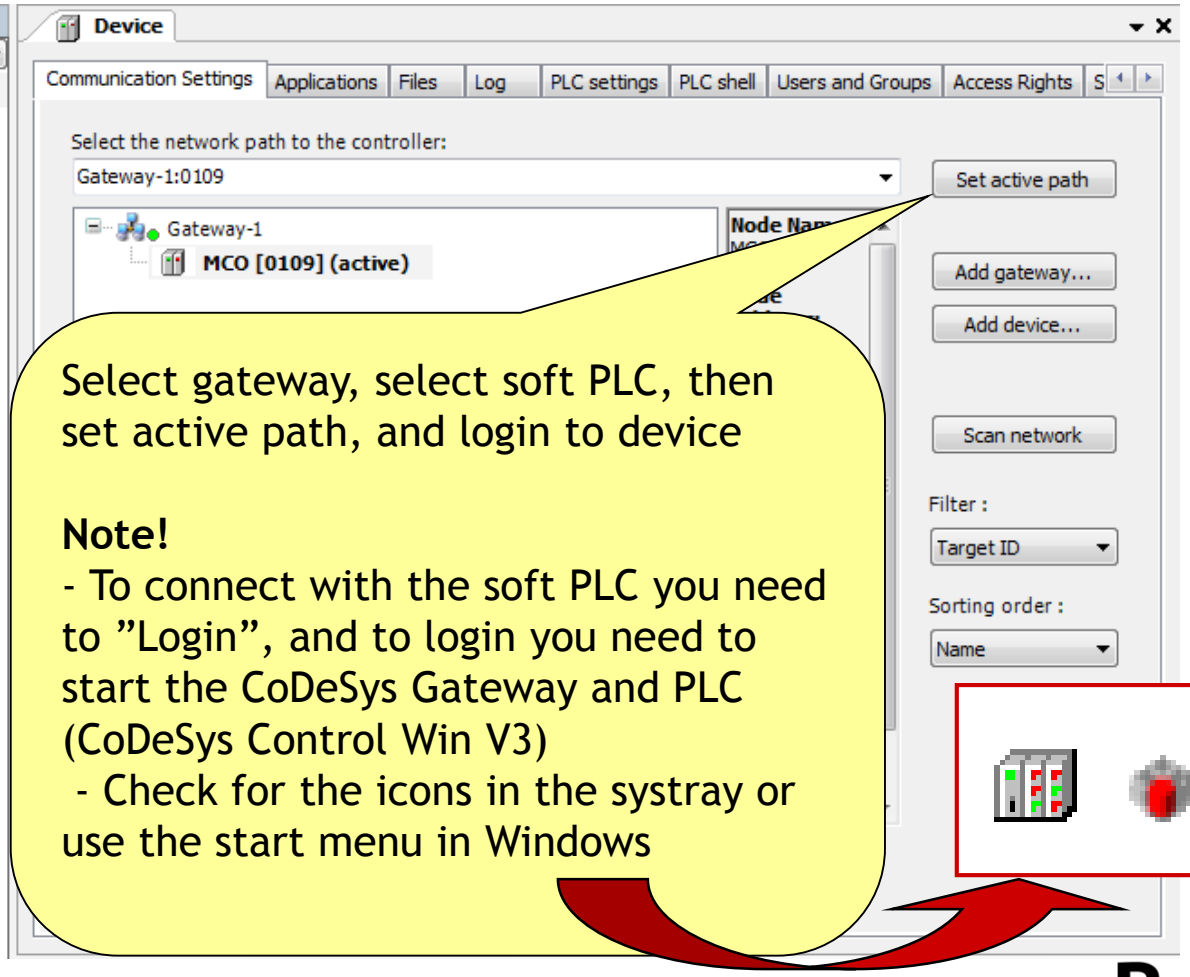
Set communication parameters

- Connect to device is done by ‘Set active path’ and ‘Login’ [Alt+F8]



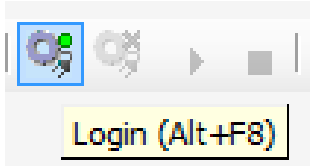
1. Select Gateway
2. Scan network
3. Select PLC
4. Device information
5. Set active path
6. Login to device:
   - » Menu selection Online – Login
   - » Toolbar or shortcut

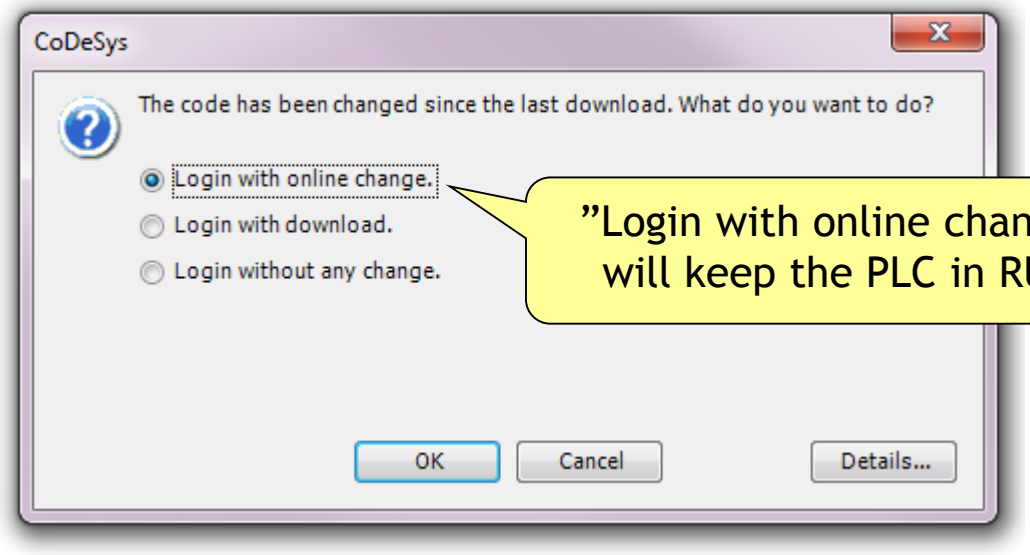




Go online by Login

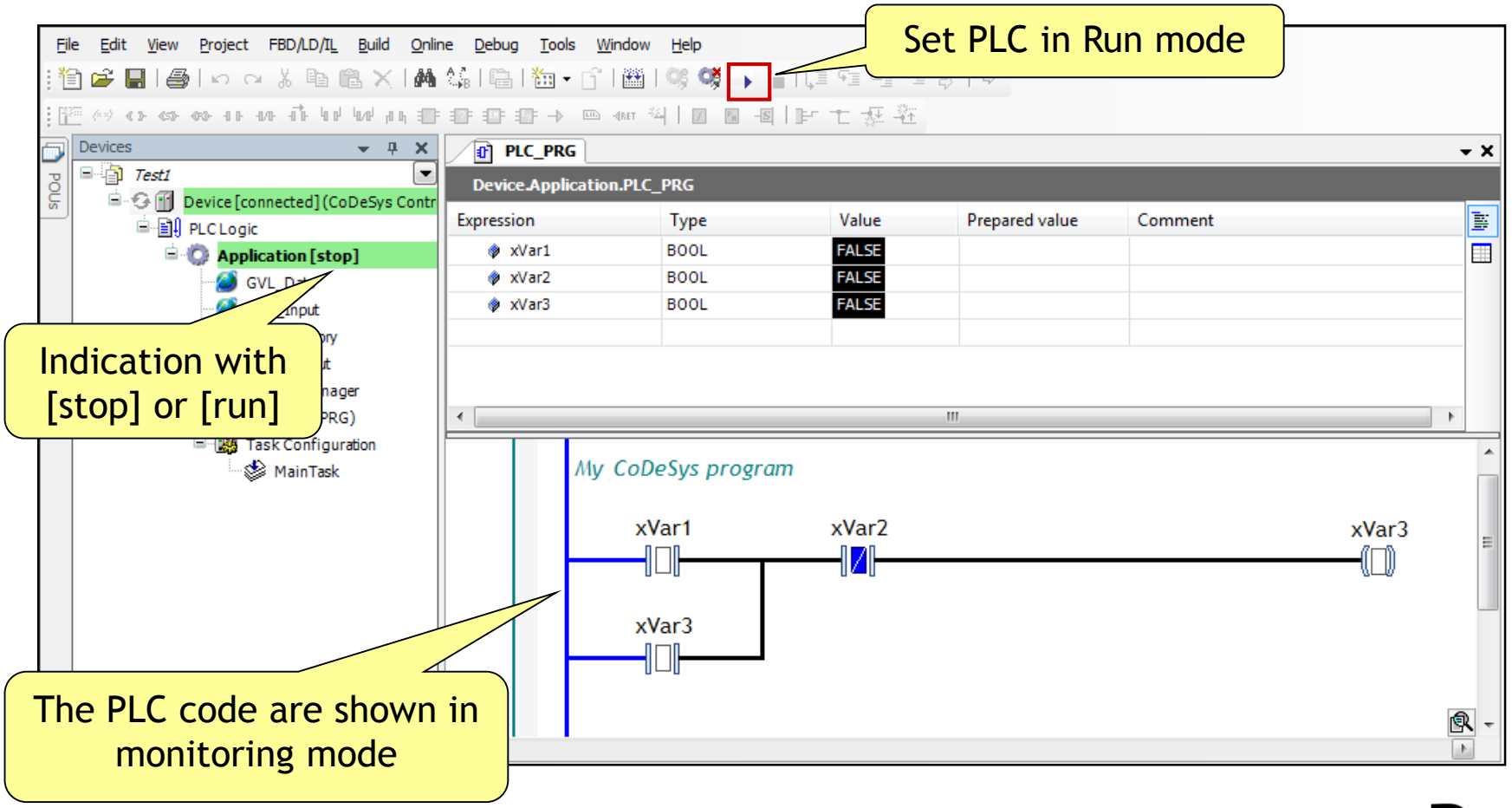
- At ‘Login’ [Alt+F8] you will get the option to download application to PLC
- Online change or download is the alternatives for the soft PLC
  - “Login with online change” will keep the CPU in run mode!
  - “Login with download” will set the PLC in stop!





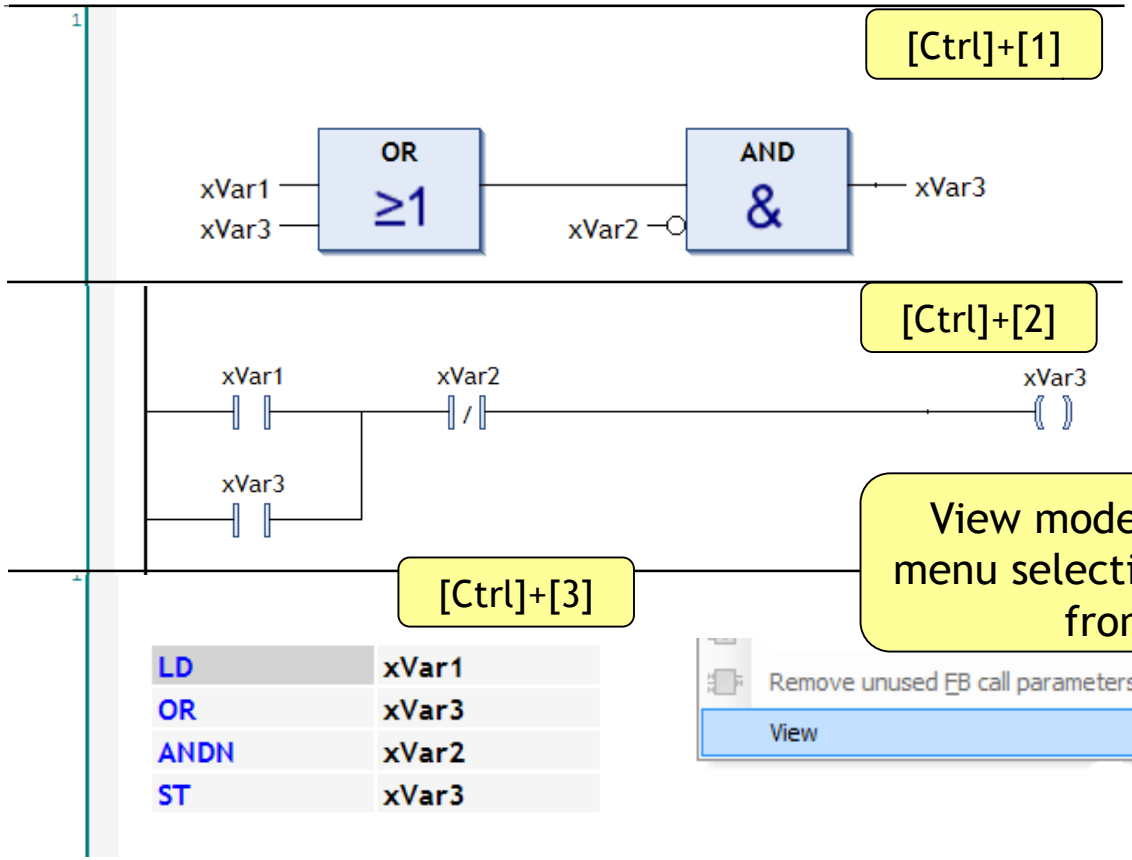
Start and stop the PLC

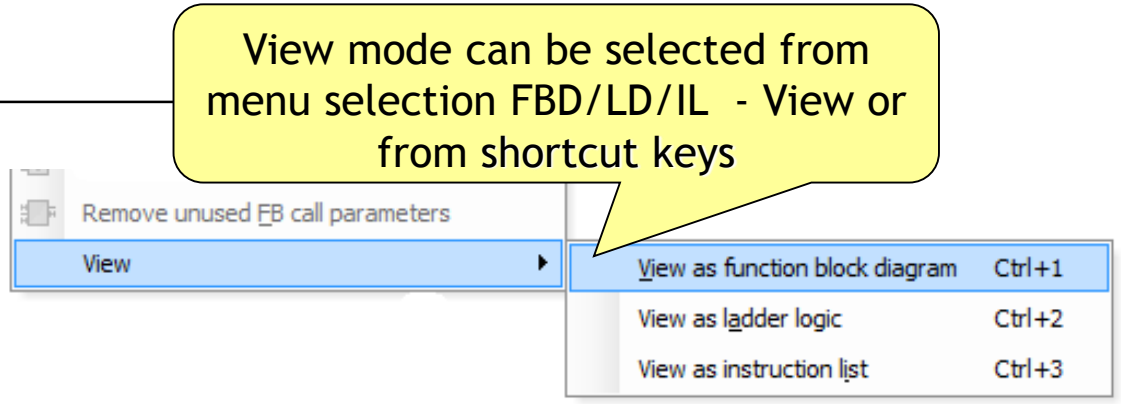
- After login you can Start the soft PLC by toolbar or menu selection
- The text in the navigator will change to [run]



View code in other language

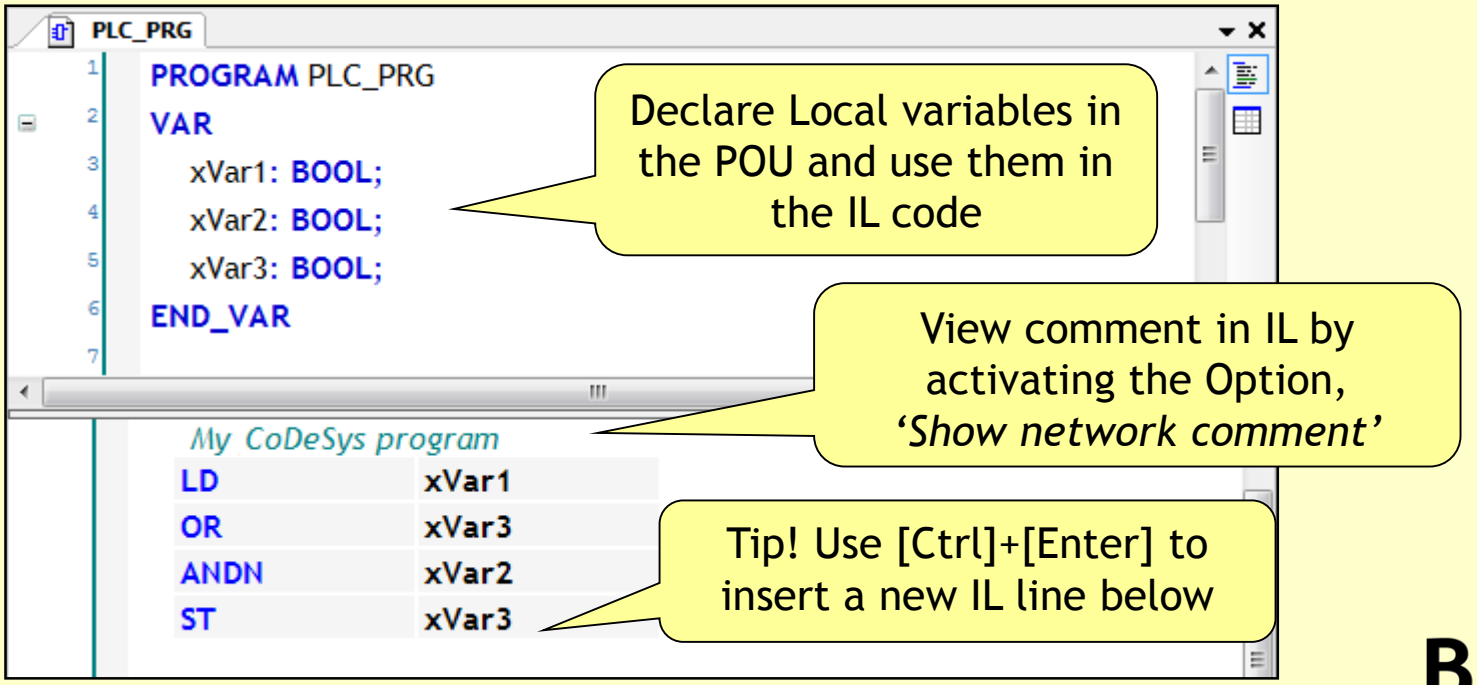
- Switch languages between FBD, LD and IL format





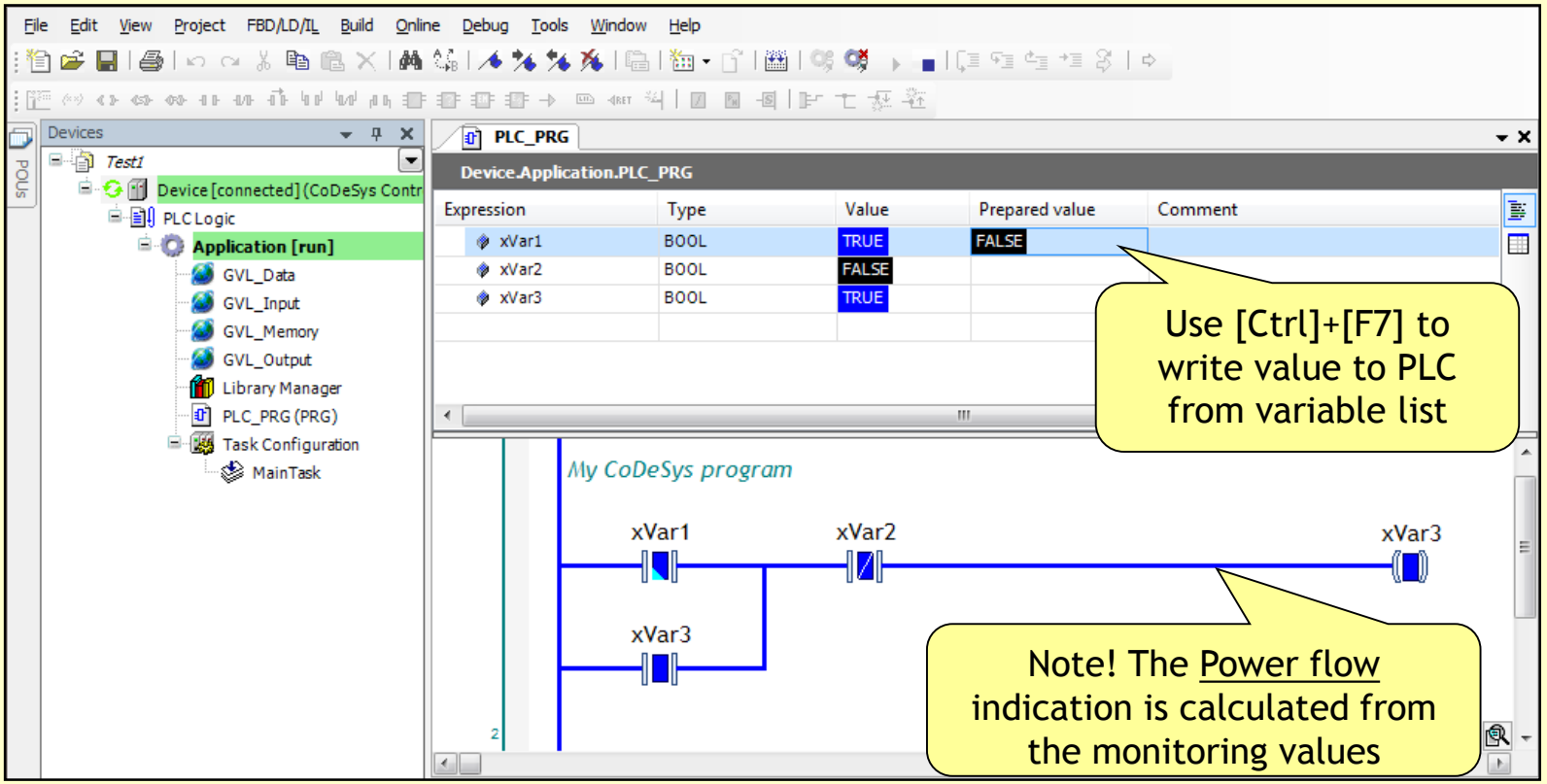
Exercise, Create first program code

1. Add PLC_PRG as a ’POU’ to the Application, use **Instruction List** (IL)
2. Declare local bit variables in ’Textual editor’ mode
3. Write a simple IL program code, make ’Build’ and check syntax
4. Create a minor error in the code, find the error from ’Message’ view
5. Try look at the code in FBD, LD or IL-editors by changing view mode:
   - Use menu selection FBD/LD/IL – View – View as…
   - Or use shortcut keys [Ctrl+1] , [Ctrl+2] or [Ctrl+3]



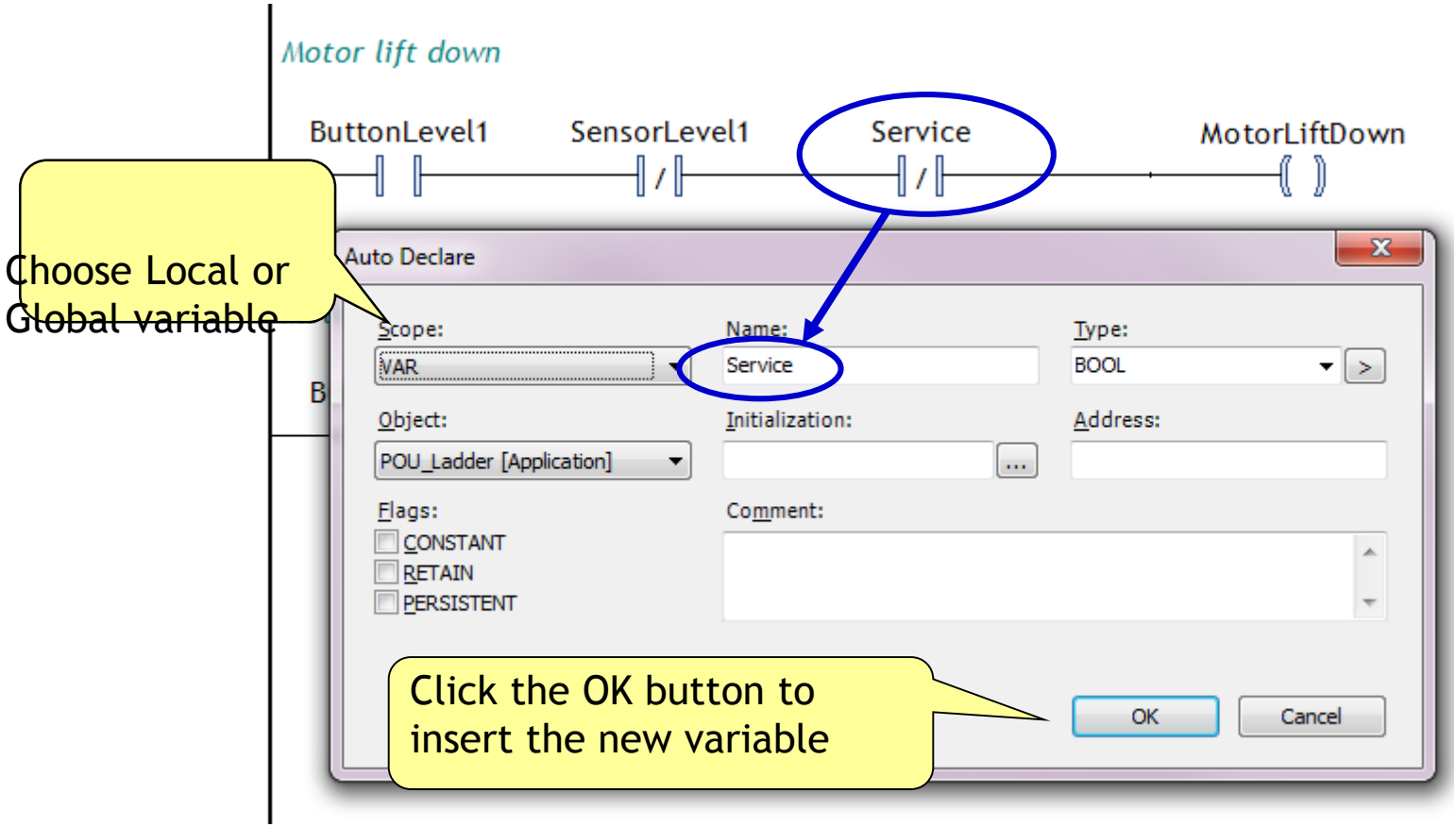
Exercise, Connect to device

- Set communication parameters (Gateway, PLC, Set path)
- Login [Alt]+[F8] and Start PLC [F5]
- View program code in monitor mode with ”power flow” indication
- Test program by writing values to the PLC
  >> Menu selection Debug – Write values [Ctrl]+[F7], or context menu



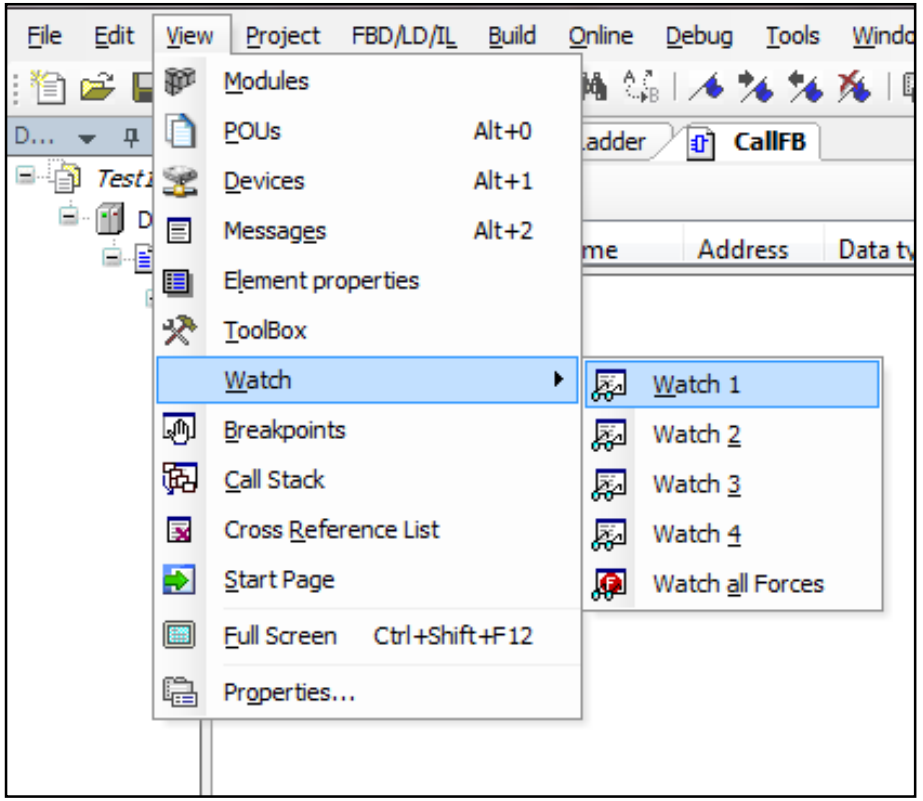
Declare new variable in editor

- Write a new label name and finish with [Enter]
- This will open the ‘Auto Declare’ window



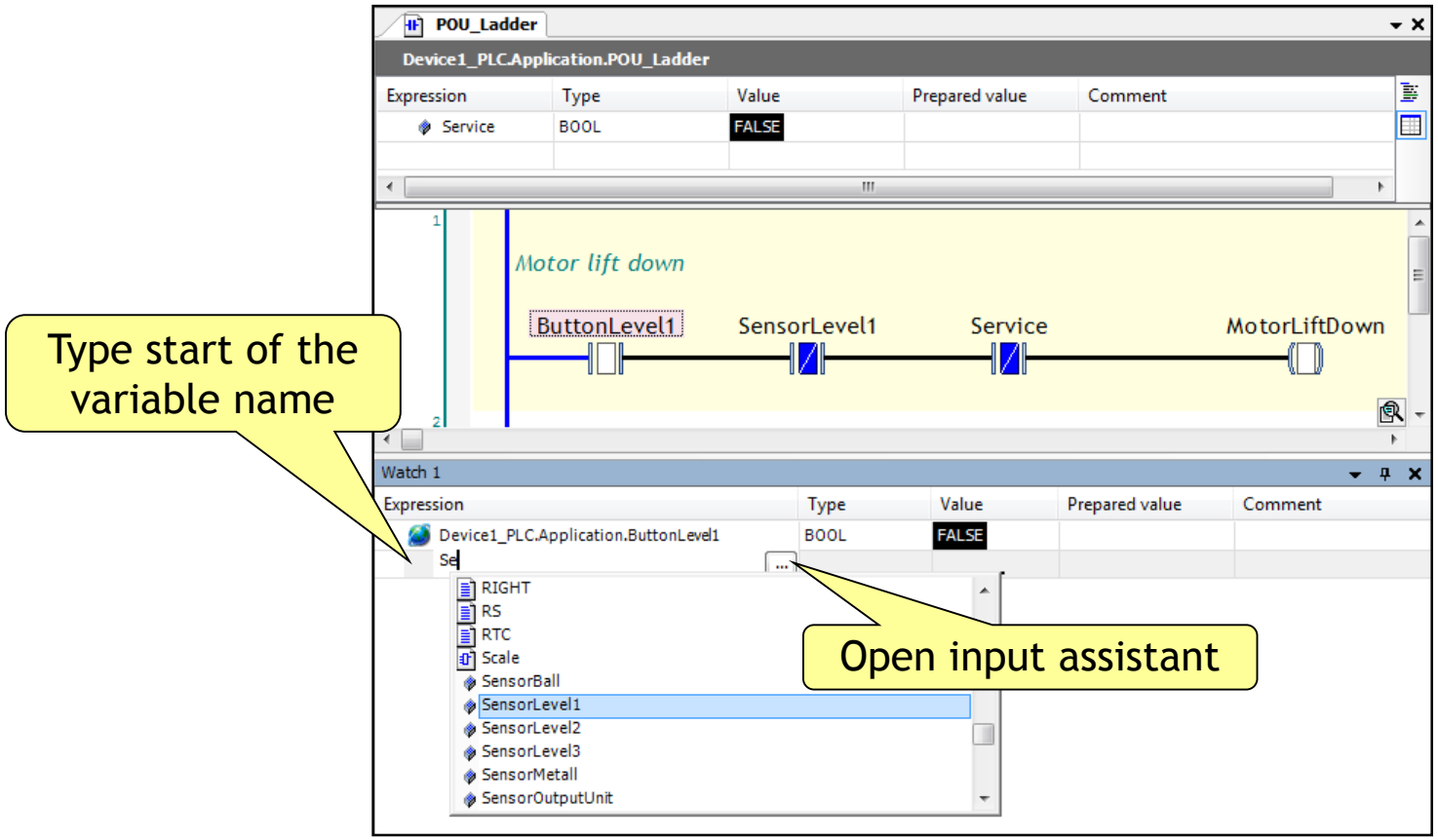
Watch Window - Open Watch view

- A watch list is a user-defined set of project variables for simultaneous monitoring of their values in a table
- By default four individual watch lists can be set up in the watch views Watch 1, Watch 2, Watch 3, Watch 4

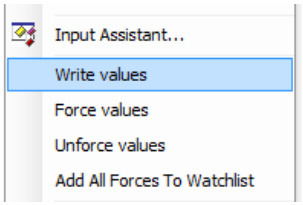


Watch Window - Register variables

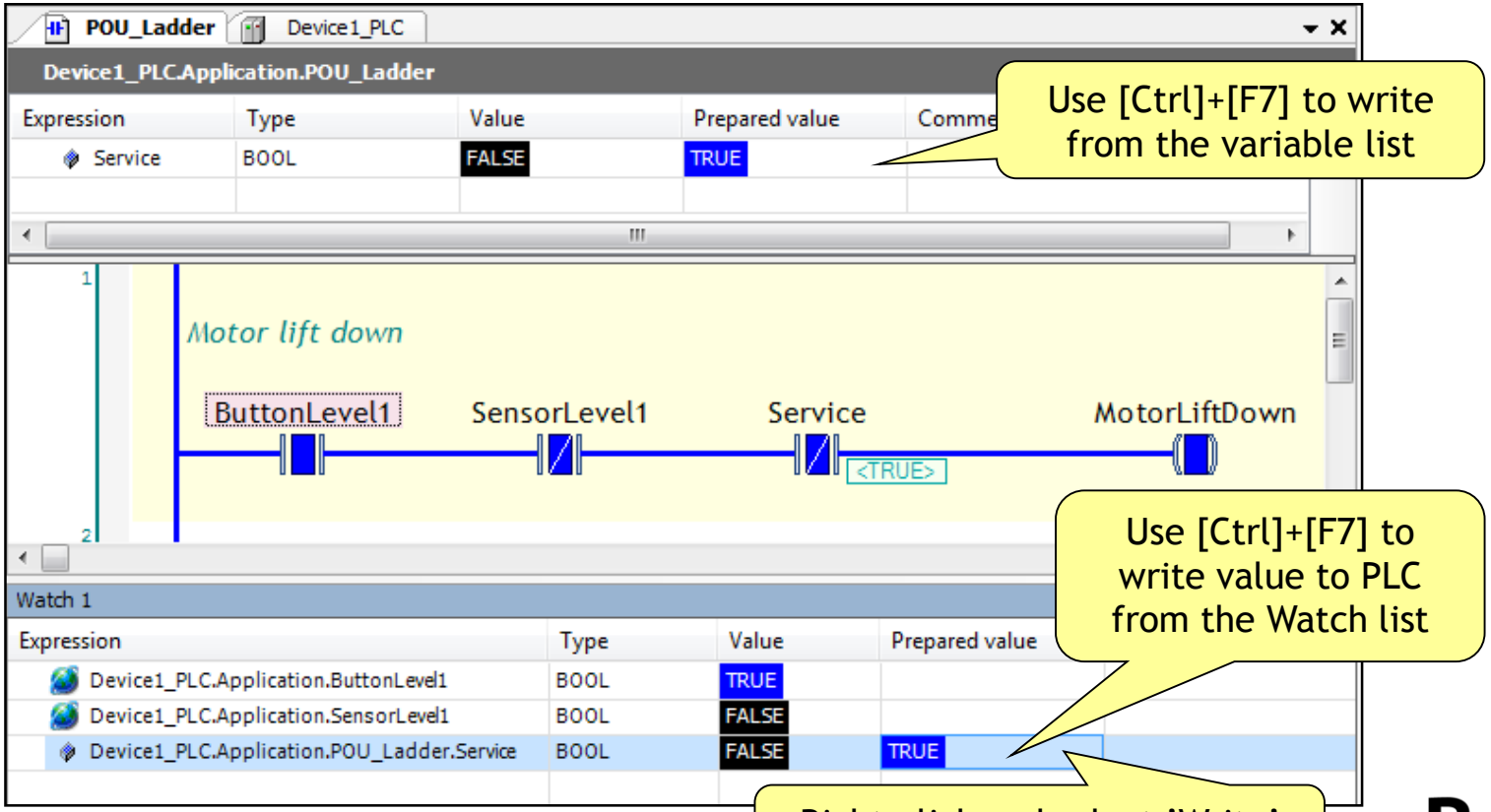
- Register variables in a watch list Watch1 open the edit frame of the column Expression by performing a mouse-click in a field of the expression column and pressing [space] and the complete path for the desired variable
- The input assistant is available via button [...]



Watch Window - Change value



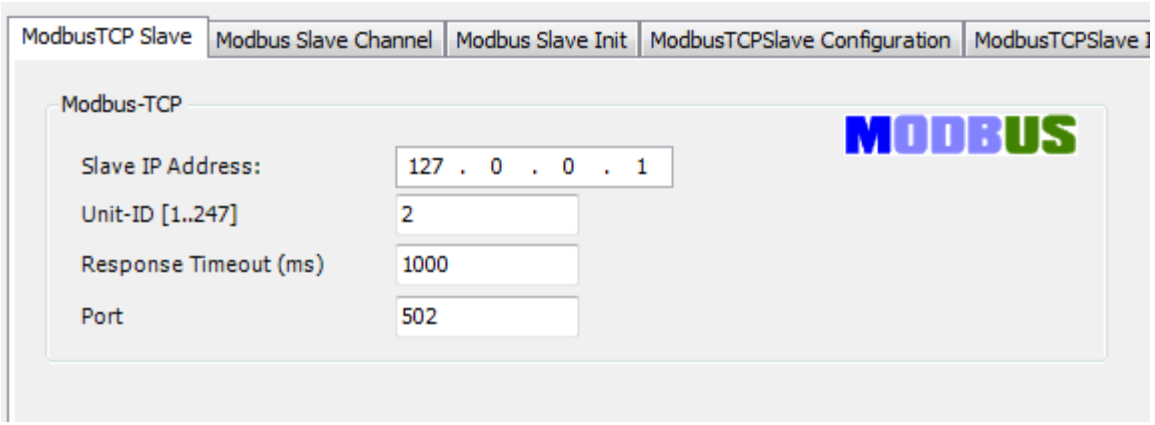
- Writing and forcing of the variables is possible within the watch view
- View Watch All Forces in online mode always gets filled automatically with all currently forced values of the active application



Settings for Elevator Simulator

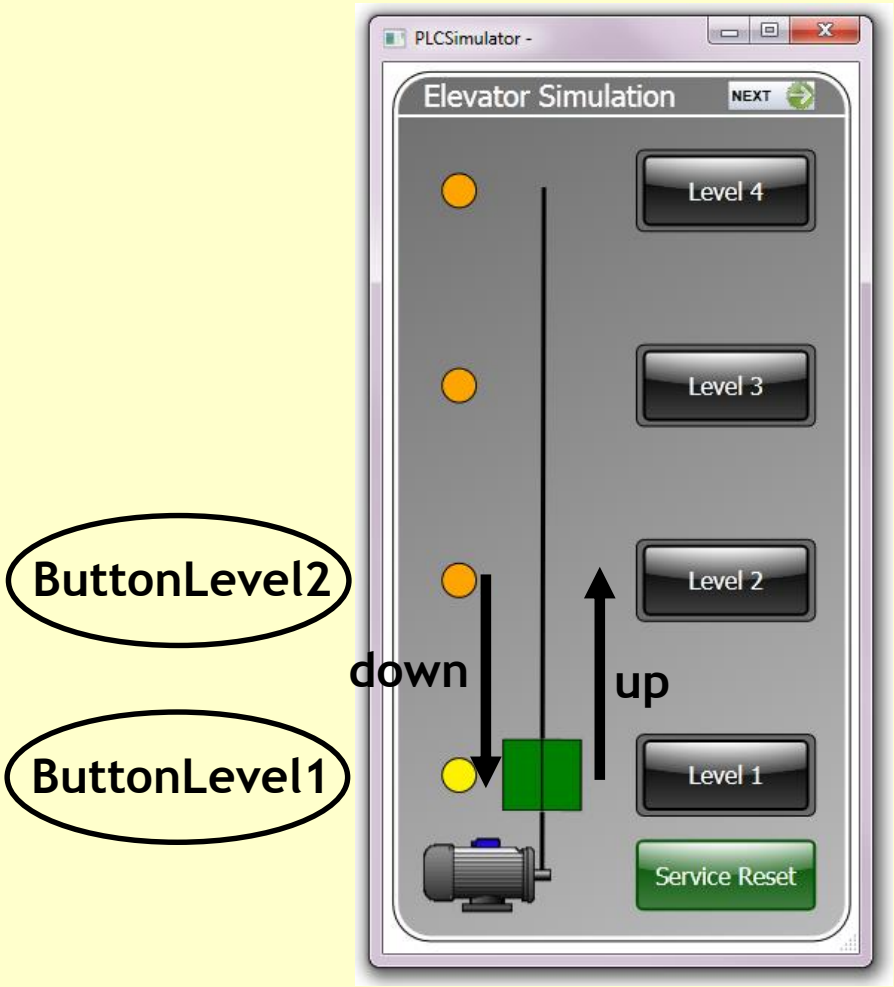
- In the exercises with CoDeSys we are using an 'Elevator Simulator'
- The Elevator application are written in iX Developer 2.0, and run as a standalone Modbus TCP slave on localhost (ip 127.0.0.1)

- Please, see additional settings in "Tab3" of the binder



Exercise, Simple lift

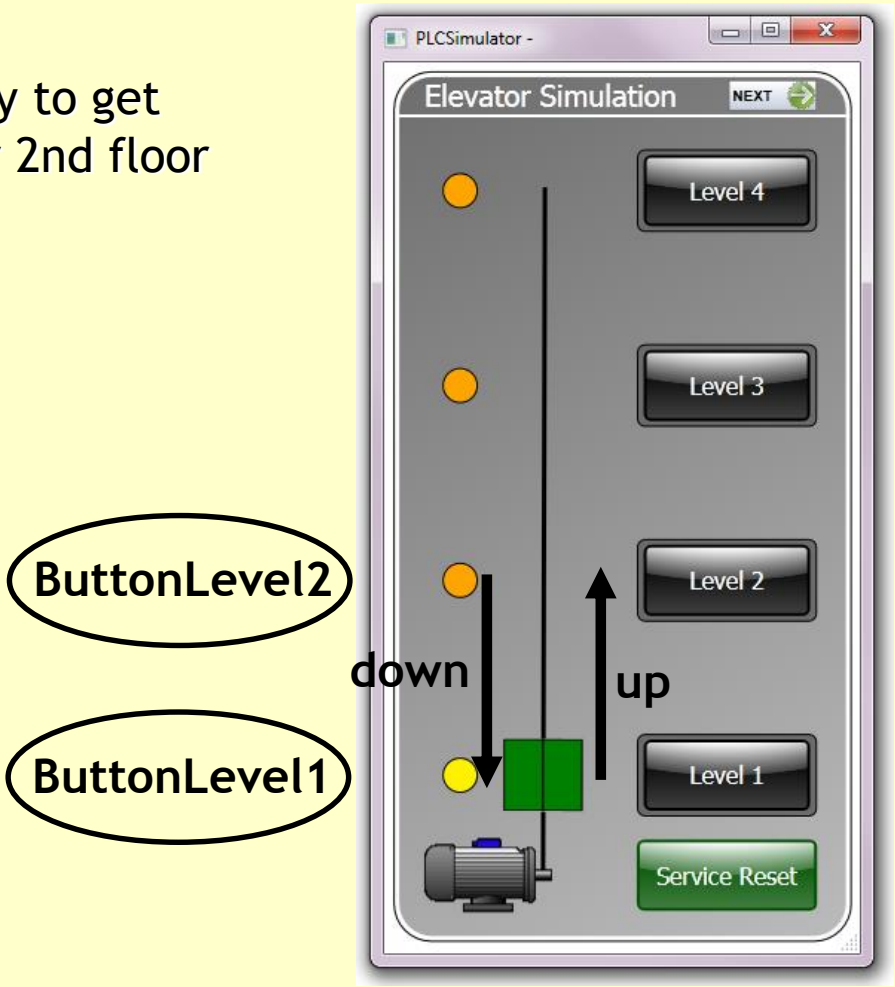
- Write a new Ladder program:
  - Use the global variable names
  - The elevator moves up as long as the push-button on 2nd floor is active, and stops when it reaches the sensor for 2nd floor
  - The elevator moves down as long as a push-button on 1st floor is active, and stops when it reaches the sensor for 1st floor



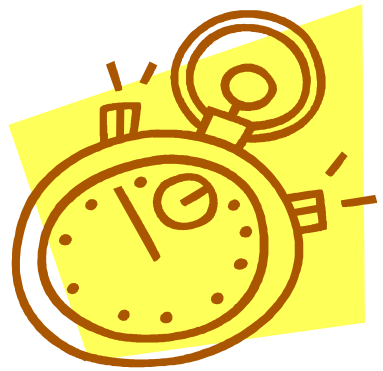
- Additional exercise: E1

Exercise, Latched function

- Use latched function
  - Just press the push-buttons shortly to get the elevator to move to the 1st or 2nd floor



- Additional exercise: E2

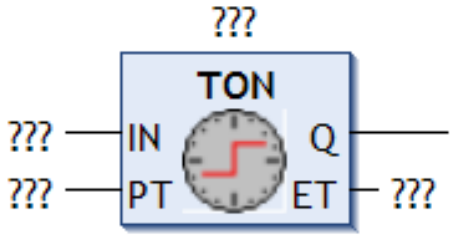


CoDeSys V3

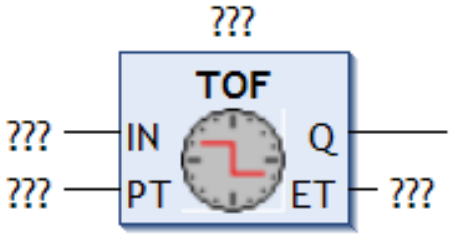
Timers and Counters

Timer block in CoDeSys

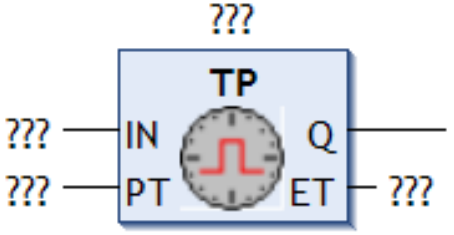
- TON , TOF and TP are the timers of IEC 61131-3 standard



Timer ON (TON)



Timer OFF (TOF)

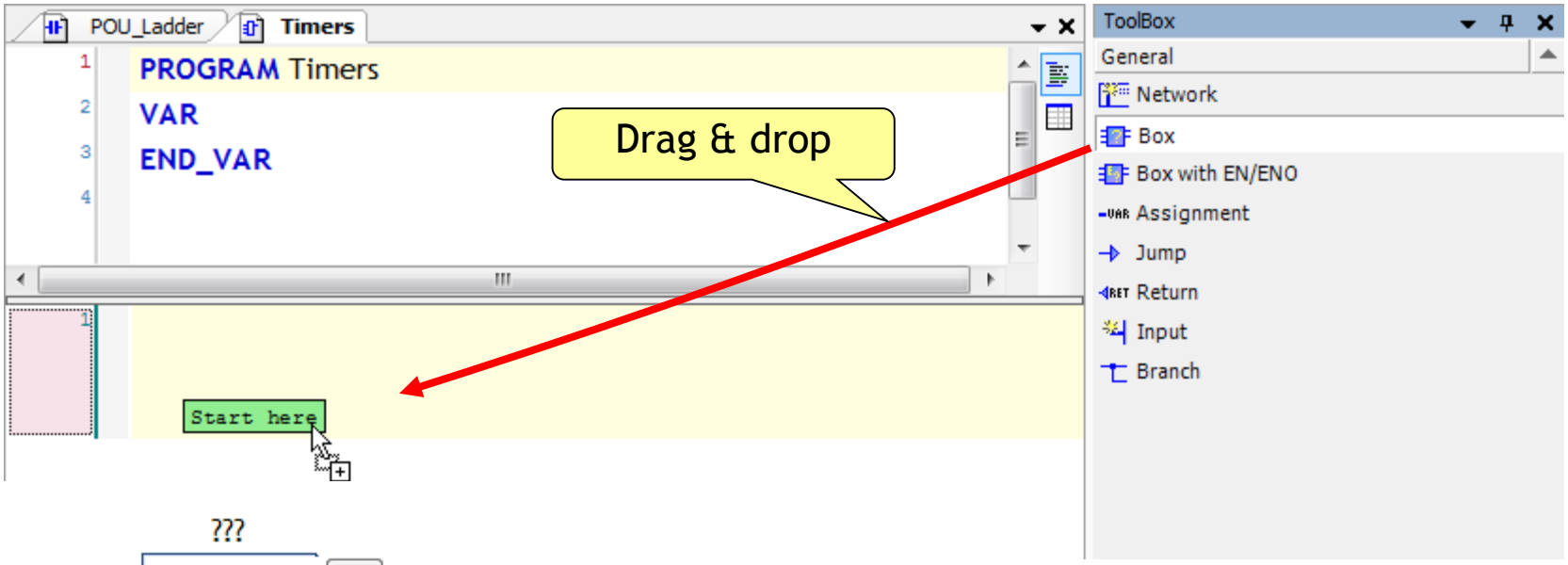


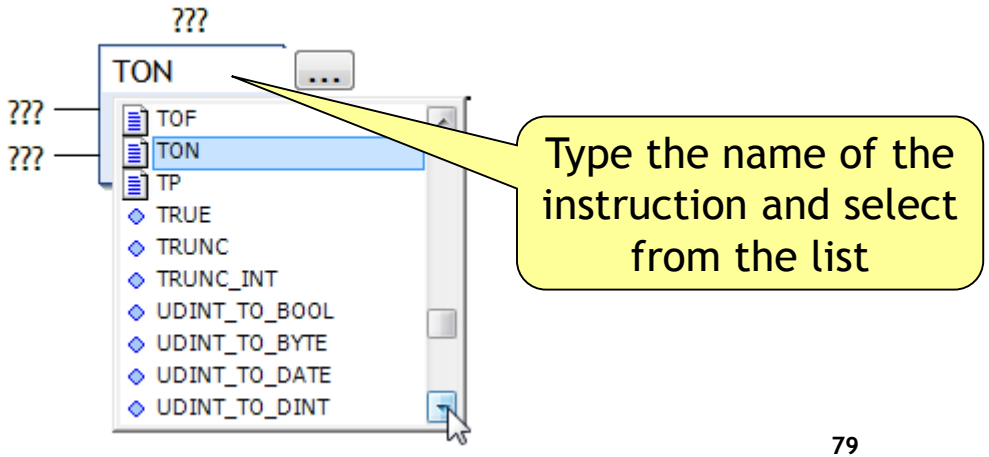
Timer PULS (TP)

- Note, the instructions are described in the online help of CoDeSys, press [F1]

Add timer / counter instructions

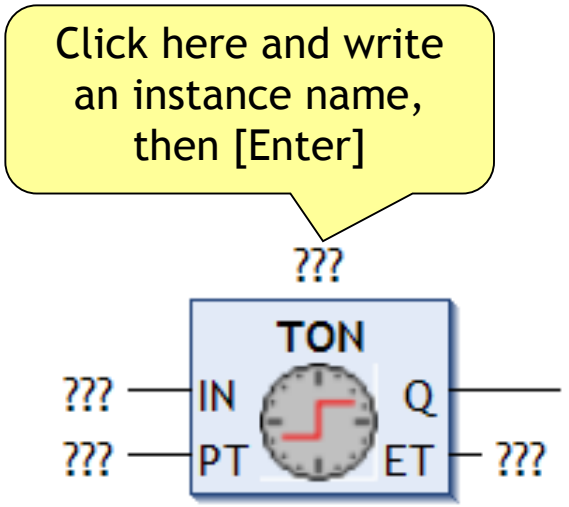
- Drag & drop items from the toolbox to a network in editor
- Drop the item at the green field “Start here”

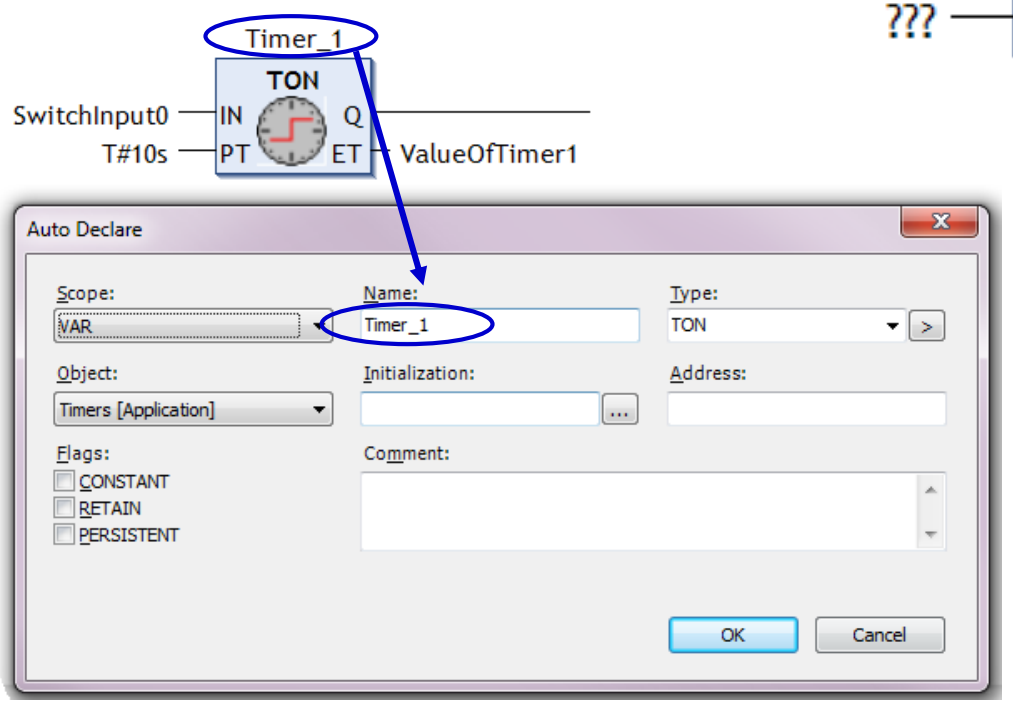




Timer declaration

- TON
  - **IN** = Executing condition
  - **PT**= Timer setting value, TIME constant (for example T#54m36s700ms)
  - **Q** = Output ValueOut = Preset
  - **ET**= Timer current value

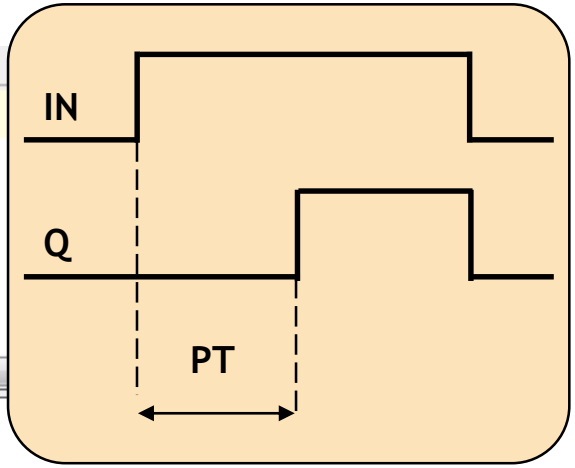


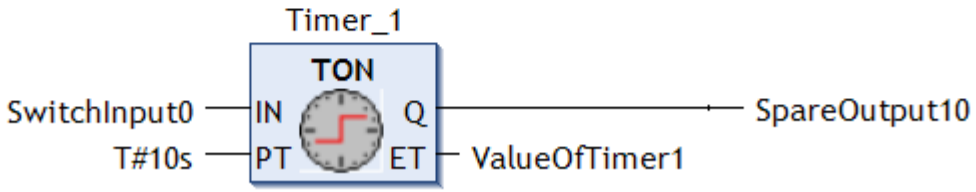


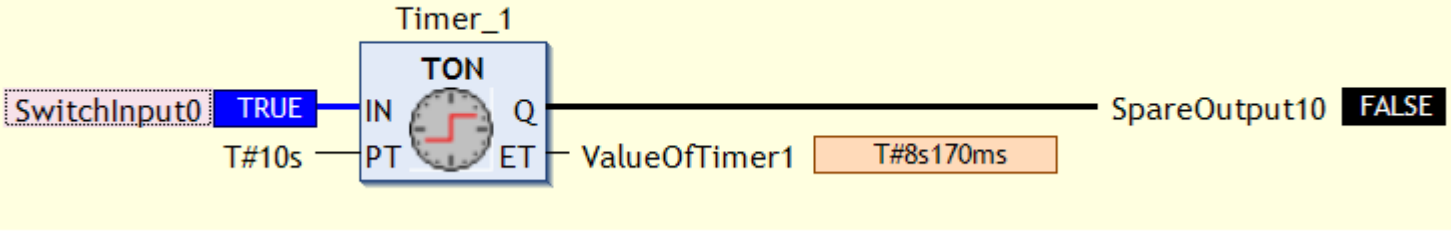
On Delay Timer

- TON

```
Timers
1  PROGRAM Timers
2  VAR
3      ValueOfTimer1: TIME;
4      Timer_1: TON;
5  END_VAR
6
```



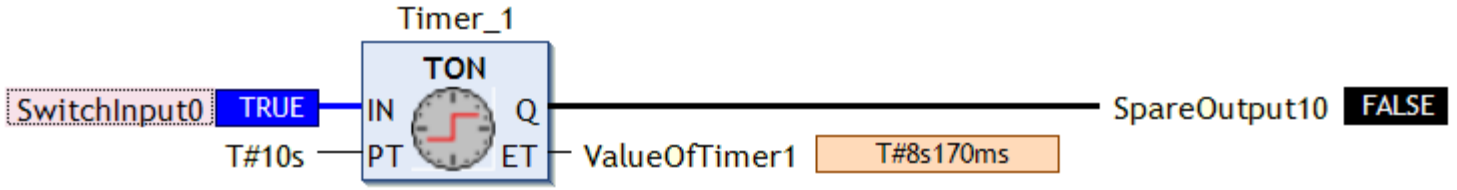




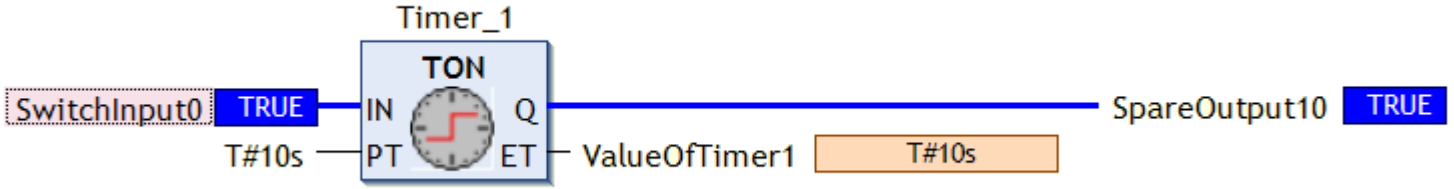
On Delay Timer (online)

- TON during execution and monitoring

- Not active

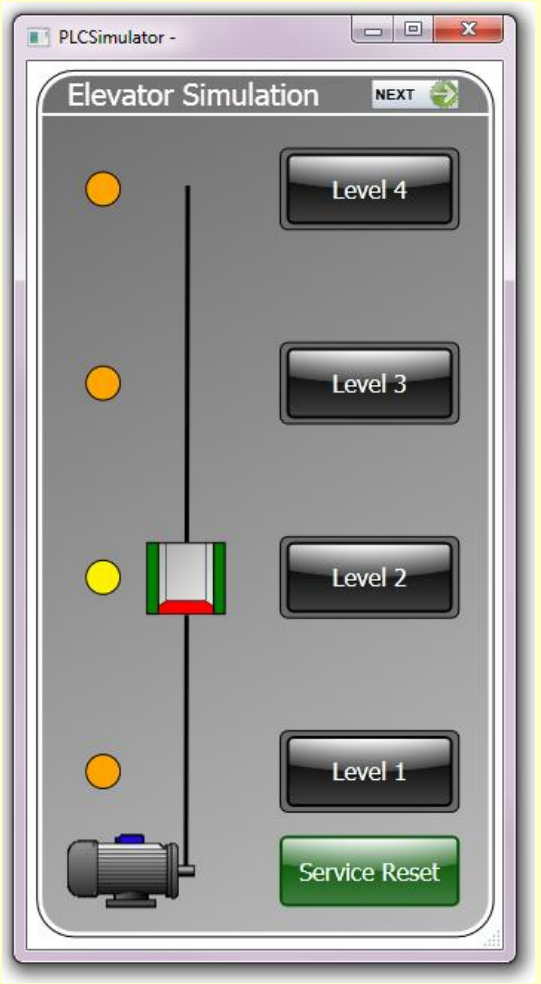


- Active



Exercise, Open door

- Open door with delay
  - When elevator arrives at a floor open the door after 2 seconds
  - The door is closed when button level is pressed before moving to next floor
  - Note, the elevator motor should not be allowed to start if the door is open!
  - Use outputsignal ElevatorDoor: BOOL;



- Additional exercise: E3

Exercise, Lamps

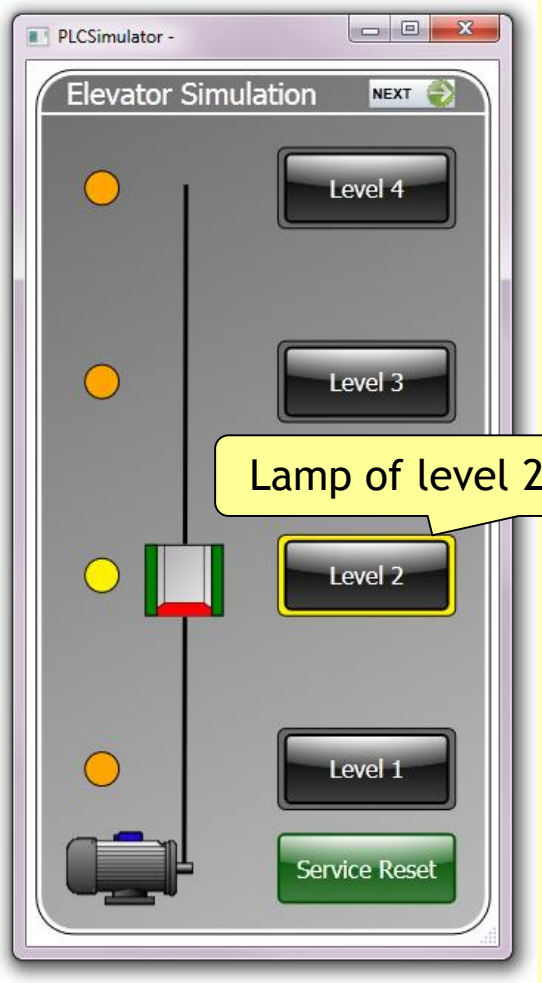
- Modify the program
  - Add instructions to make the lamps at each floor light up when the elevator has arrived
  - Output signal LampLevel1 to 4

LampLevel4

LampLevel3

LampLevel2

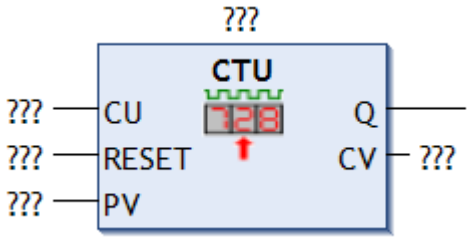
LampLevel1



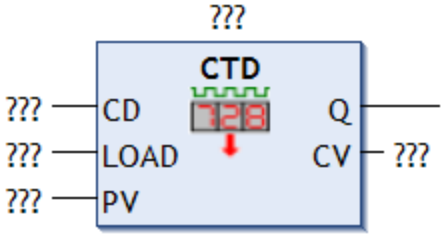
- Additional exercise: E4 and E5

Counters

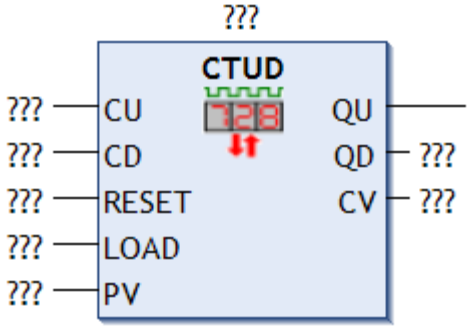
- CTU , CTD and CTUD are the countes of IEC 61131-3 standard



Count Up (CTU)



Count Down (CTD)



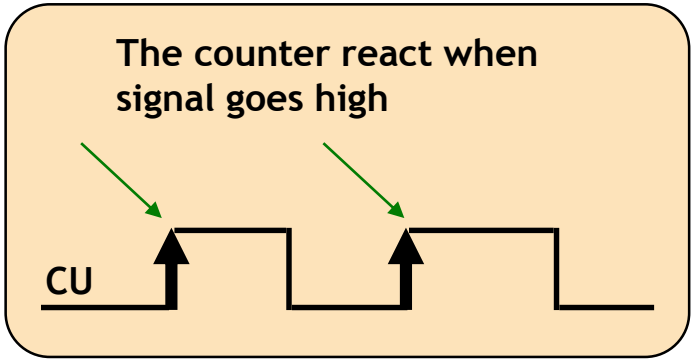
Count Up/Down (CTUD)

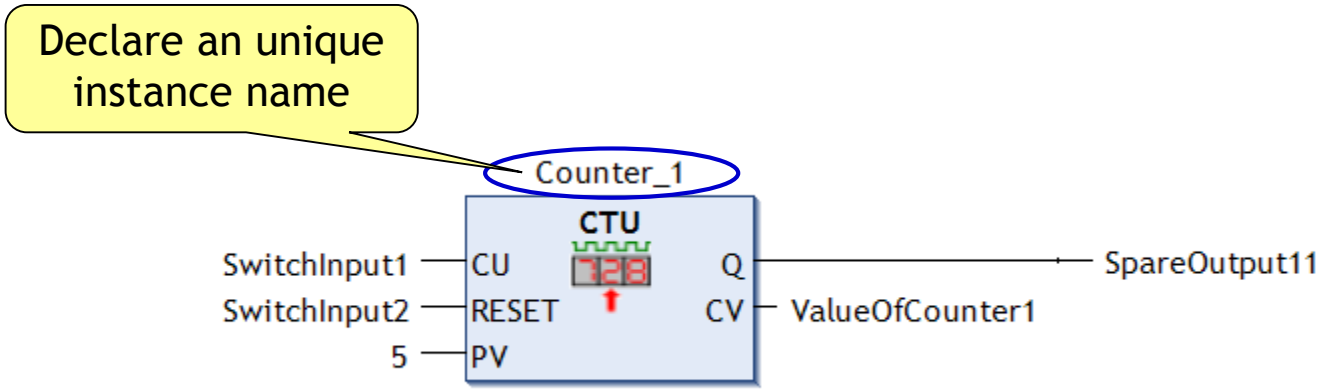
- Note, the counters are described in the online help of CoDeSys, press [F1]

Counter CTU (IEC)

- CTU

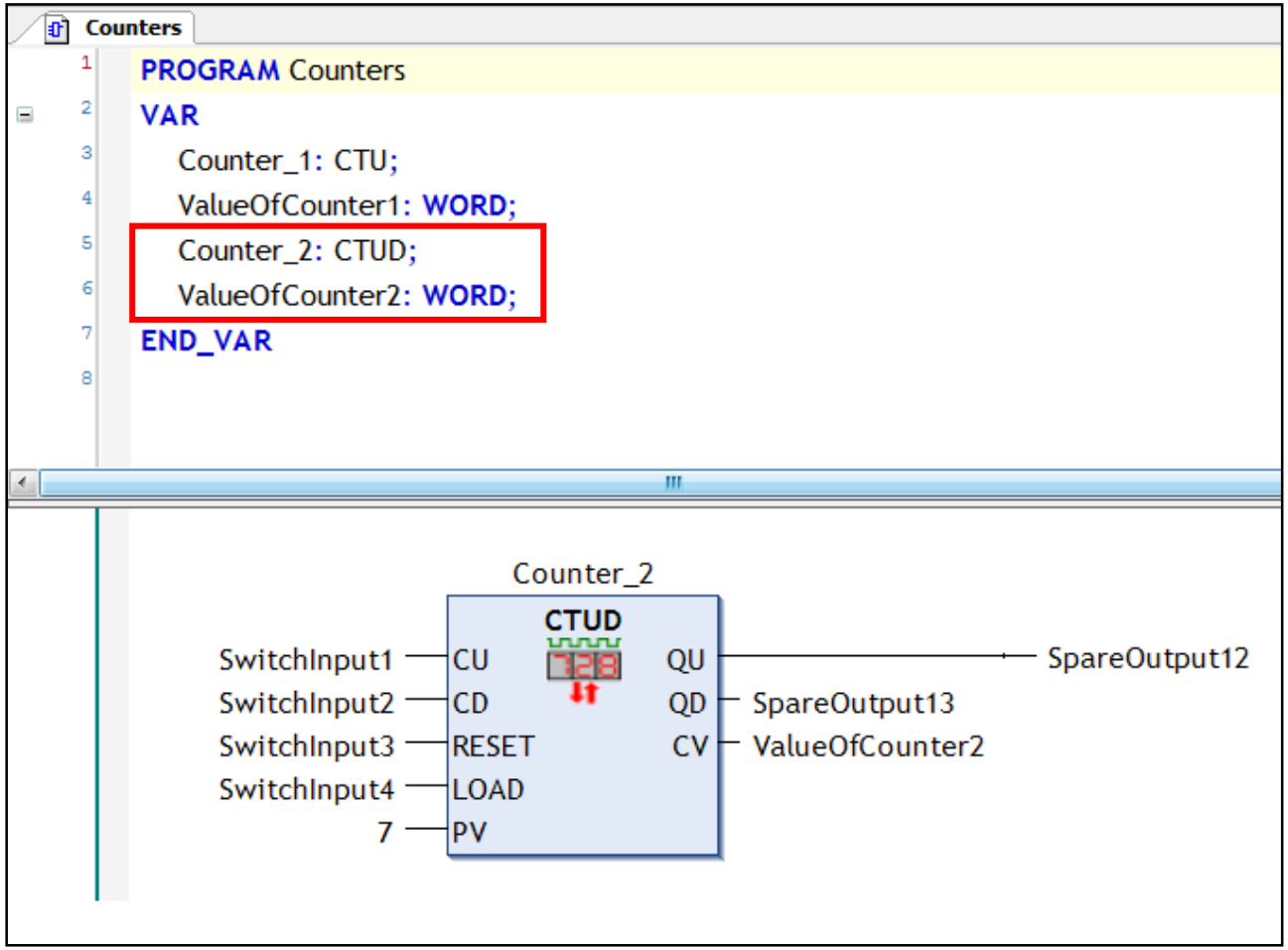
  CU = Executing condition
  RESET = Counter reset condition
  PV= Preset value, decimal constant
  Q = Output ValueOut = Preset
  CV= Counter current value





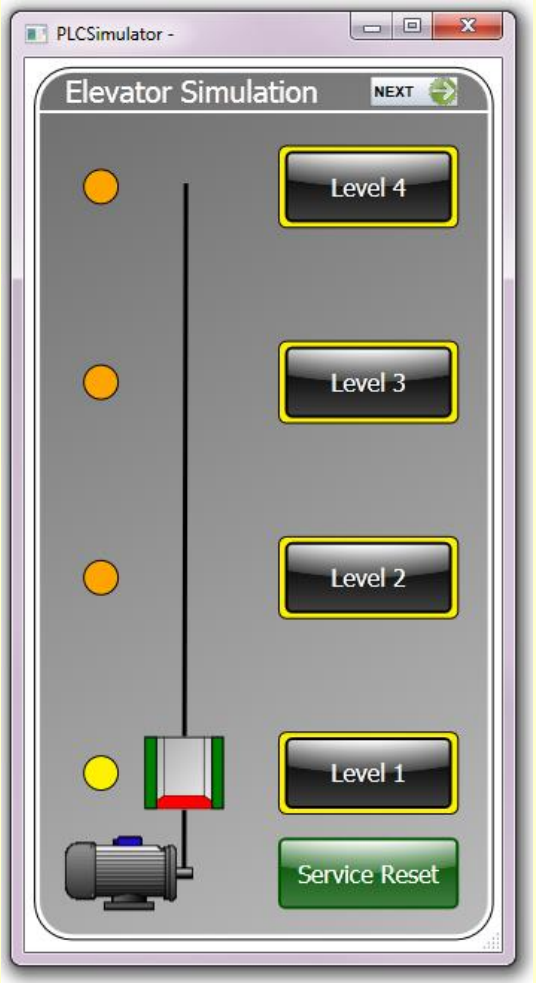
Example of counter CTUD

- Instance of CTUD in the local variable list

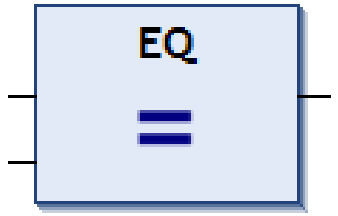


Exercise, Service Counter

- When the elevator has started 5 times, it should stop for service and maintenance.
- Let a counter keep track of how many times the motor has started.
- Activate all the lights in the elevator to shine steadily, and ensure that the elevator will not run until service is completed.
- After service make acknowledge via ServiceReset and the elevator should work normally until the next service occasion.



- Additional exercise: E7



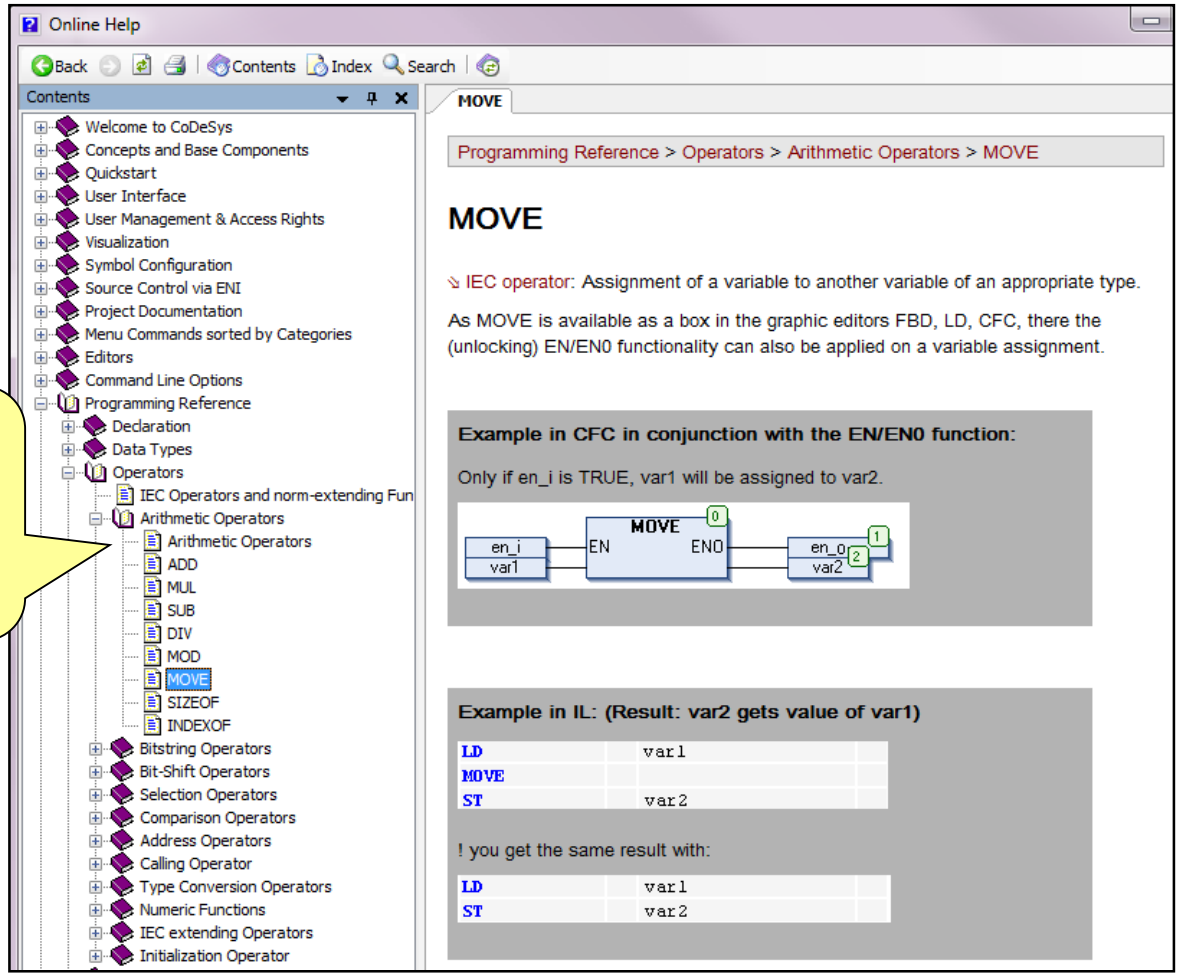
CoDeSys V3

Data Instructions

Find instructions and operators

- The online help of CoDeSys, include a summary of all standard instructions
- Menu selection Help/Content, and ‘Programming Reference’

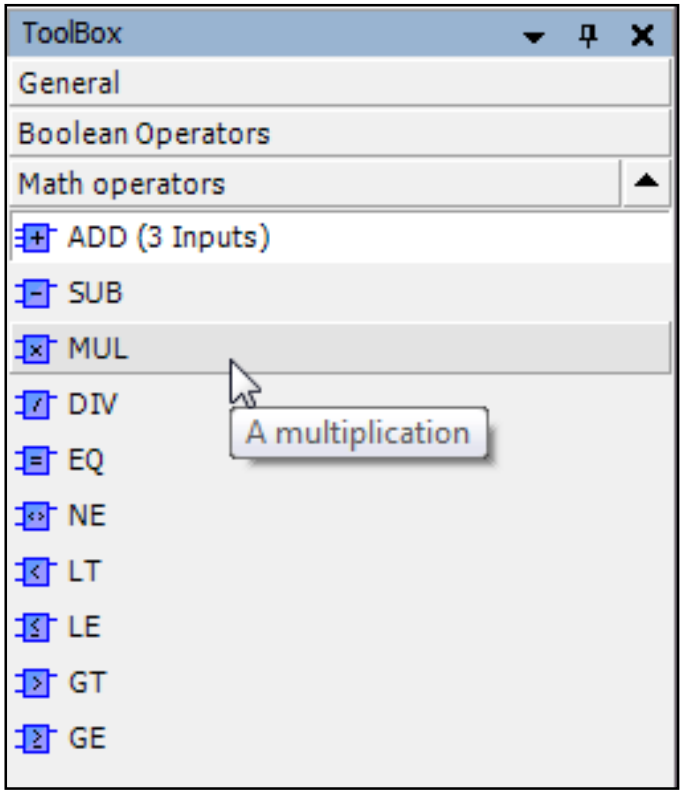
MOVE command found in section ‘Arithmetic Operators’



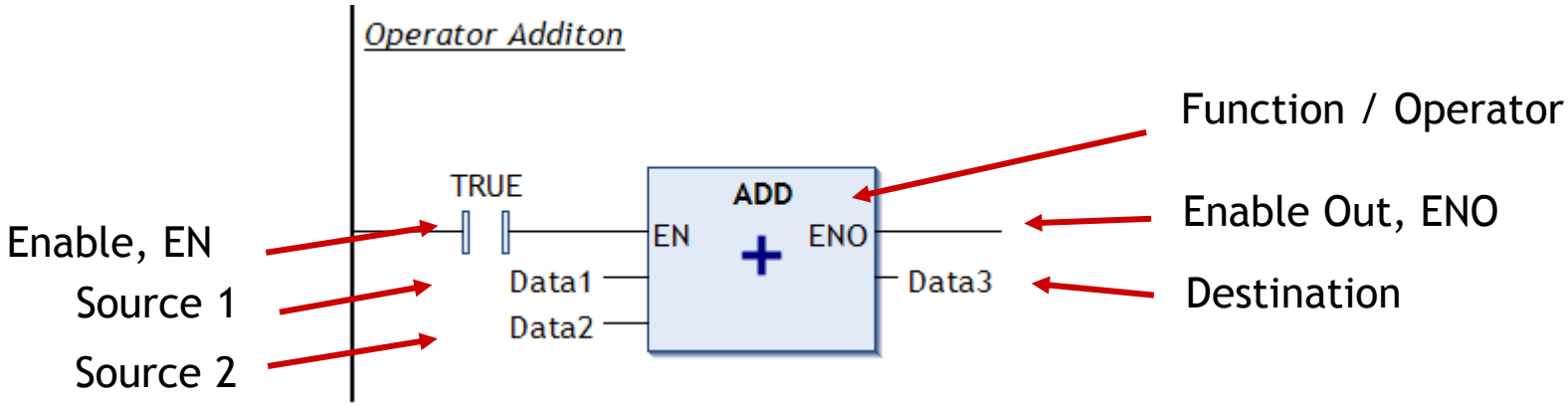
Data Instructions

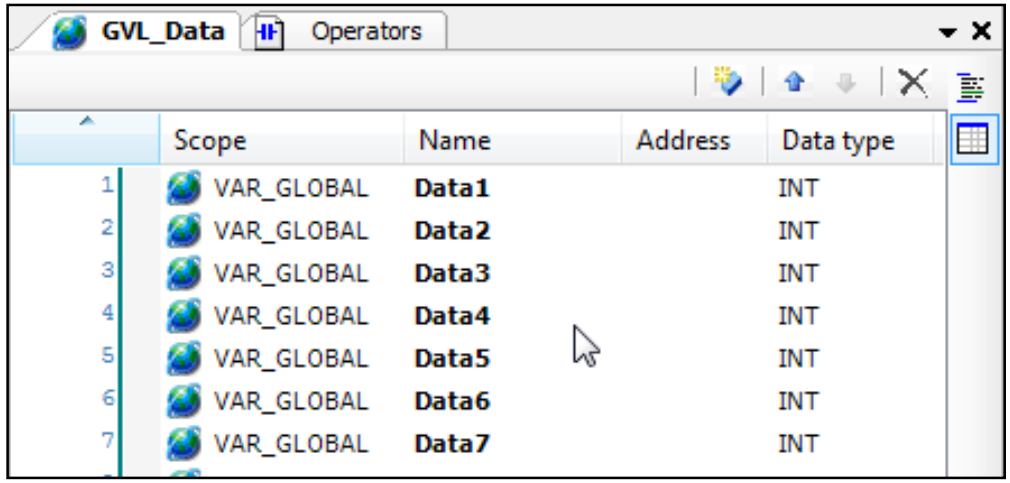
- Examples of data instructions from the toolbox in CoDeSys

ADD	Addition (2 or 3 inputs)
SUB	Subtraction
MUL	Multiplication
DIV	Division
MOVE	Data transfer
EQ, LT etc.	Comparison
SEL	Binary Selection
MUX	Mulitplexer
LIMIT	Limiting

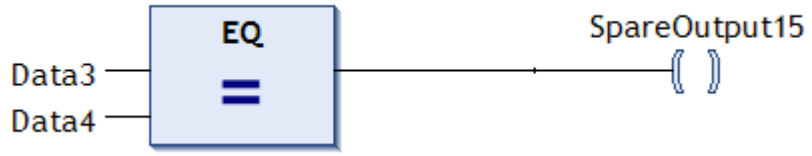


Data Instructions





Comparison



EQ Equal to (=)
Returns true when the operands are Equal

NE Not Equal to (<>)
Returns true when the operands are different (Not Equal)

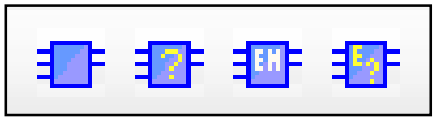
GE Greater than or Equal to (>=)
Returns true if the 1st operand is Greater than or Equal to the 2nd operand

GT Greater than (>)
Returns true if the 1st operand is Greater Than the 2nd operand

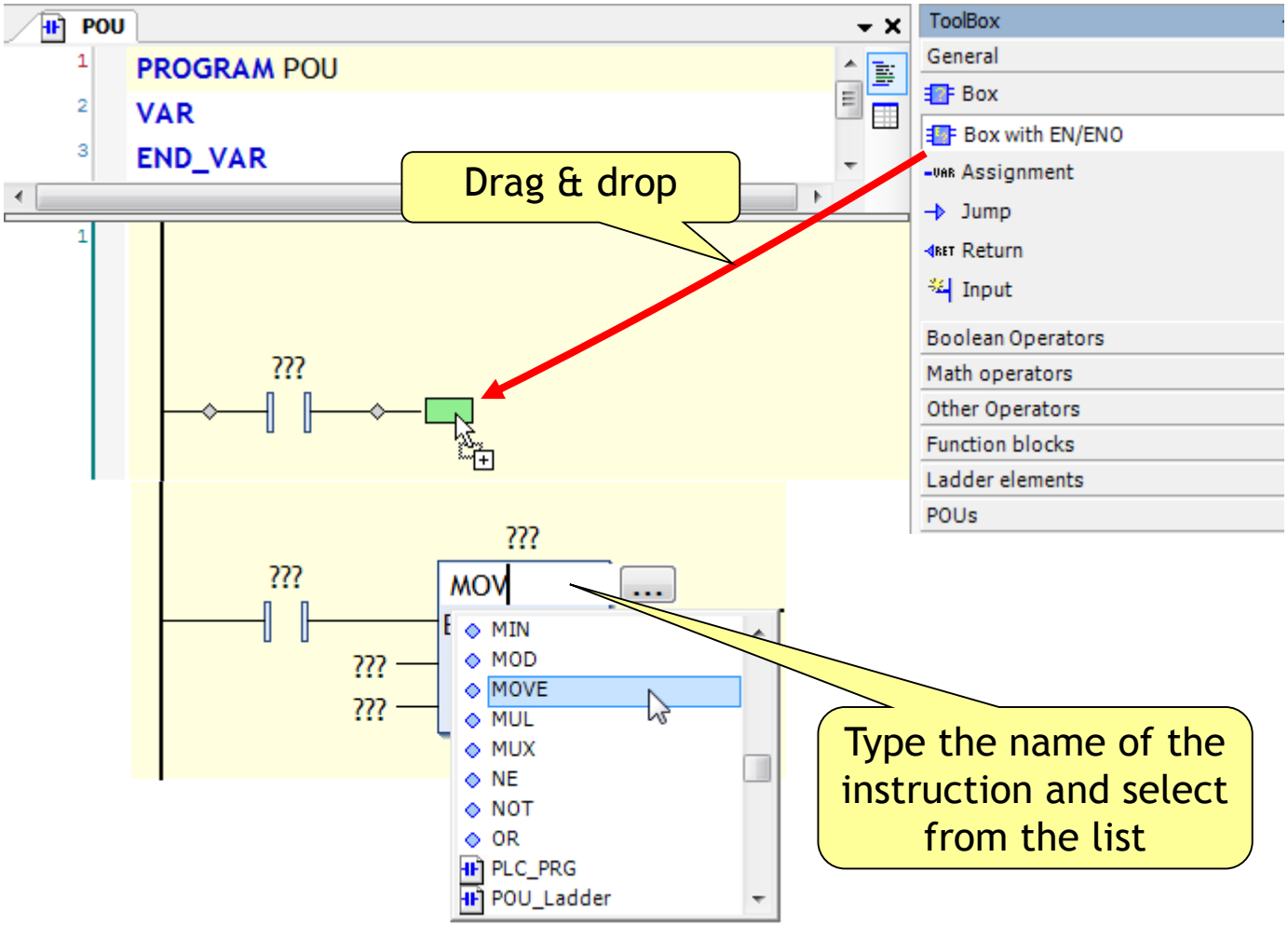
LE Less than or equal to (<=)
Returns true if the 1st operand is Less than or Equal to the 2nd operand

LT Less than (<)
Returns true if the 1st operand is Less Than the 2nd operand

Add data instructions / operators

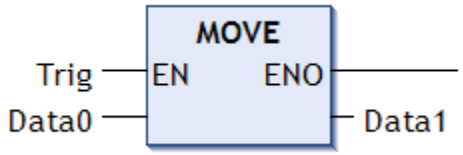


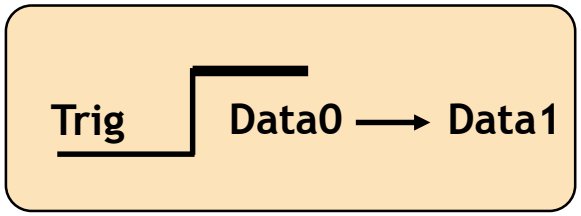
- Drag & drop items from the toolbox to a network in editor



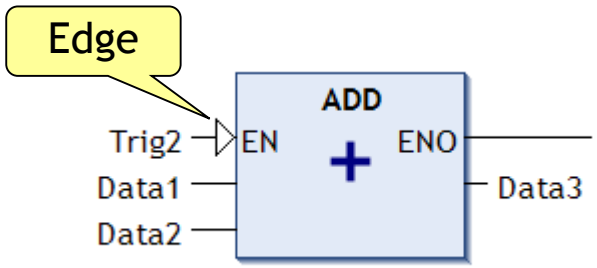
Example with MOVE, ADD, MUL (real)

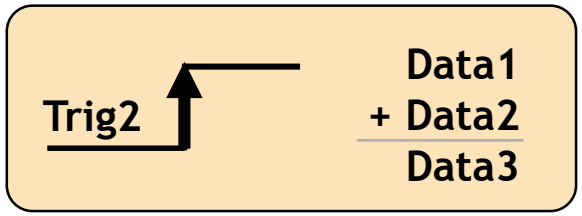
MOVE



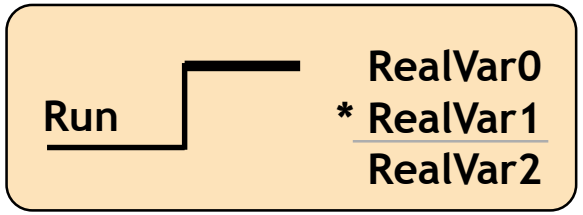


ADD





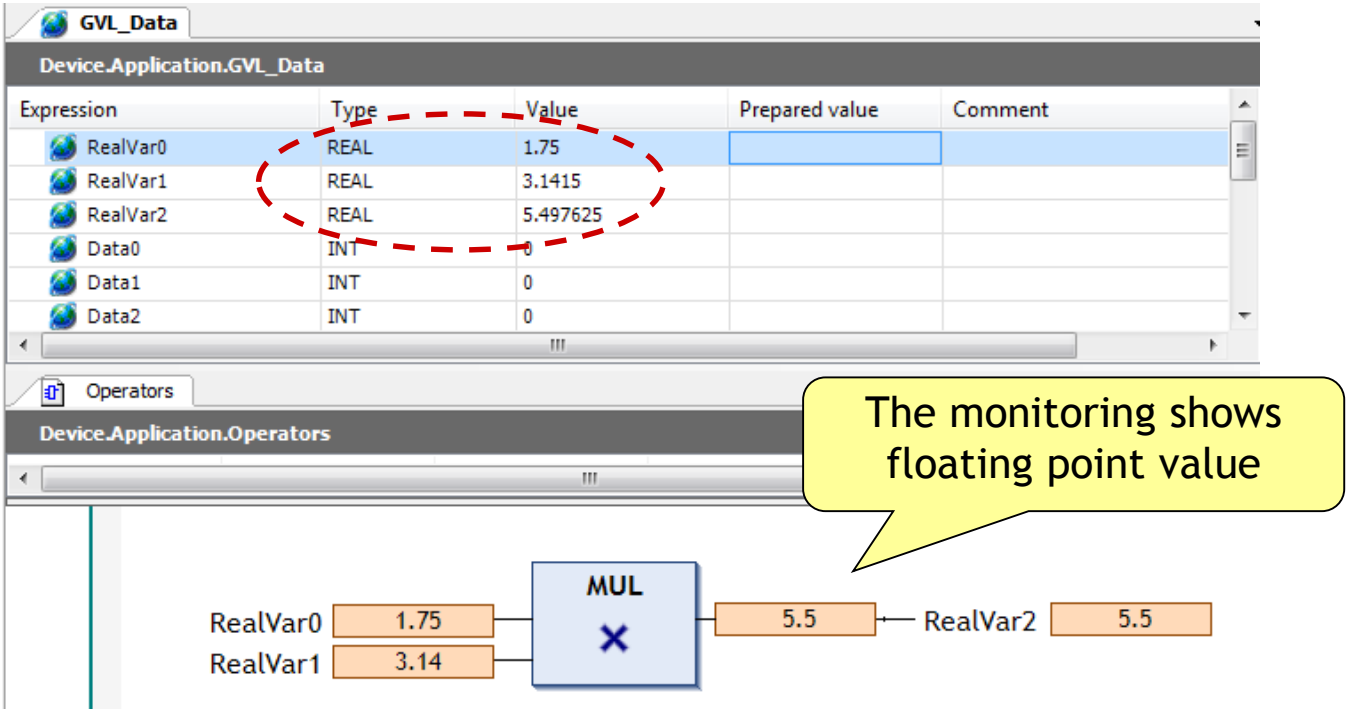
MUL



- Note, symbol for ‘Edge’ detection of the EN input of ADD operand
- Corresponds to inserting a R_TRIG function block for detecting a rising edge
- Compare the F_TRIG function block for detecting a falling edge

Floating Point Calculations

- Addition **ADD**
- Subtraction **SUB**
- Division **DIV**
- Multiplication **MUL**



Exercise, Declare INT and REAL

- Declare Data variables to the ’GVL_Data’ list

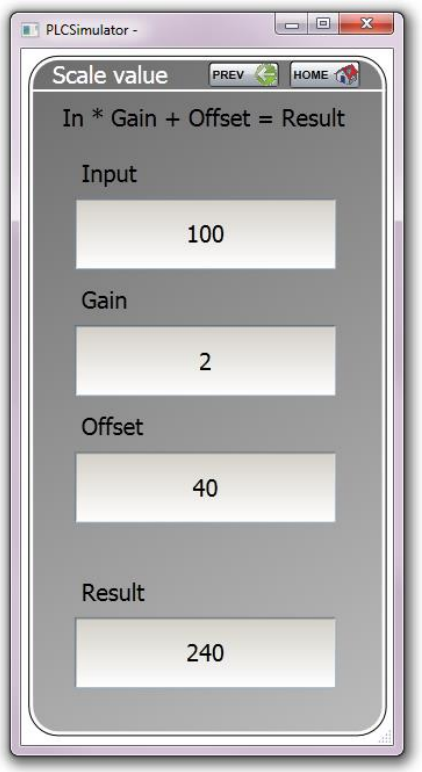
GVL_Data

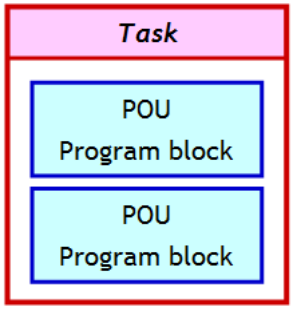
```
1  VAR_GLOBAL
2      iData0 AT %IW10: INT;
3      iData1 AT %QW11: INT;
4      iData2: INT;
5      iData3: INT;
6      iData4: INT;
7      iData5: INT;
8      iData6: INT;
9      iData7: INT;
10     Input: INT;
11     Gain: INT;
12     Offset: INT;
13     Result: INT;
14     RealVar0: REAL;
15     RealVar1: REAL;
16     RealVar2: REAL;
17     RealVar3: REAL;
18     RealVar4: REAL;
```

Tip!
Import from text file,
GlobalVariableList.txt

Exercise, Scale a value

- Make a new program for scaling a Integer value (INT)
- Use formula **In * Gain + Offset = Out**
  - Similar to the linear equation y=kx+m
- Declare the variables and write the code in FBD editor
- Test program in PLC and monitor values
  - Use the screen “Scale value” of the Elevator Simulator when available
- Note, if you make this exercise with local floating point variables and constant values these values must be written in decimal form with radix point
  - For example 123.45
- Additional exercise: E6



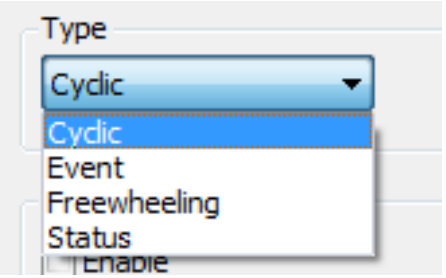


CoDeSys V3

Task conditions

Task Condition

- Task execution by Type and Event:



Cyclic: The task will be processed cyclic according to the time definition ("task cycle time") given in the field 'Interval'

Freewheeling: The task will be processed as soon as the program is started and at the end of one run will be automatically restarted in a continuous loop.
There is no cycle time defined

Status: The task will start if the the Event is true

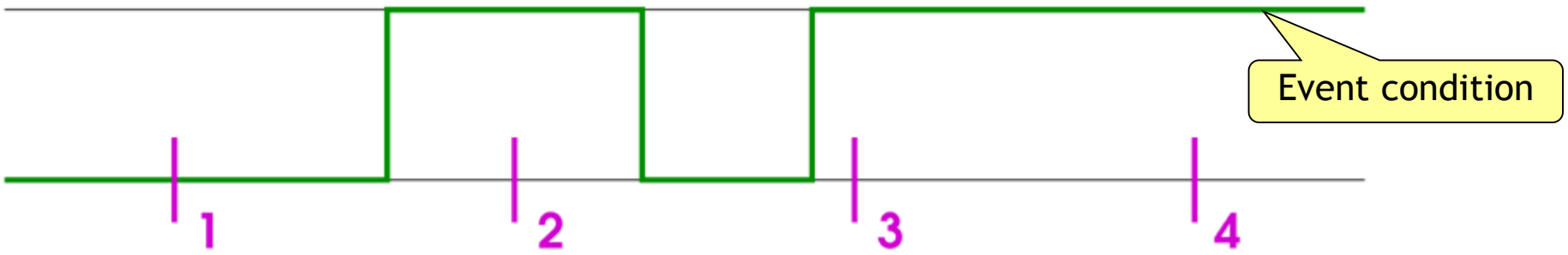
Event: The task will start as soon as the variable defined in the Event field gets a rising edge

External event: The task will be started as soon as the system event, which is defined in the Event field, occurs. It depends on the target, which events will be supported and offered in the selection list.
(Not to be mixed up with system events.)

Task Condition

- Difference between **Status** and **Event**:
  - The specified event being TRUE fulfills the start condition of a status driven task, whereas an event driven task requires the change of the event from FALSE to TRUE.
  - If the sampling rate of the task scheduler is too low, rising edges of the event may be left undetected.

- The following example illustrates the behaviour of the task in reaction to an event (green line):

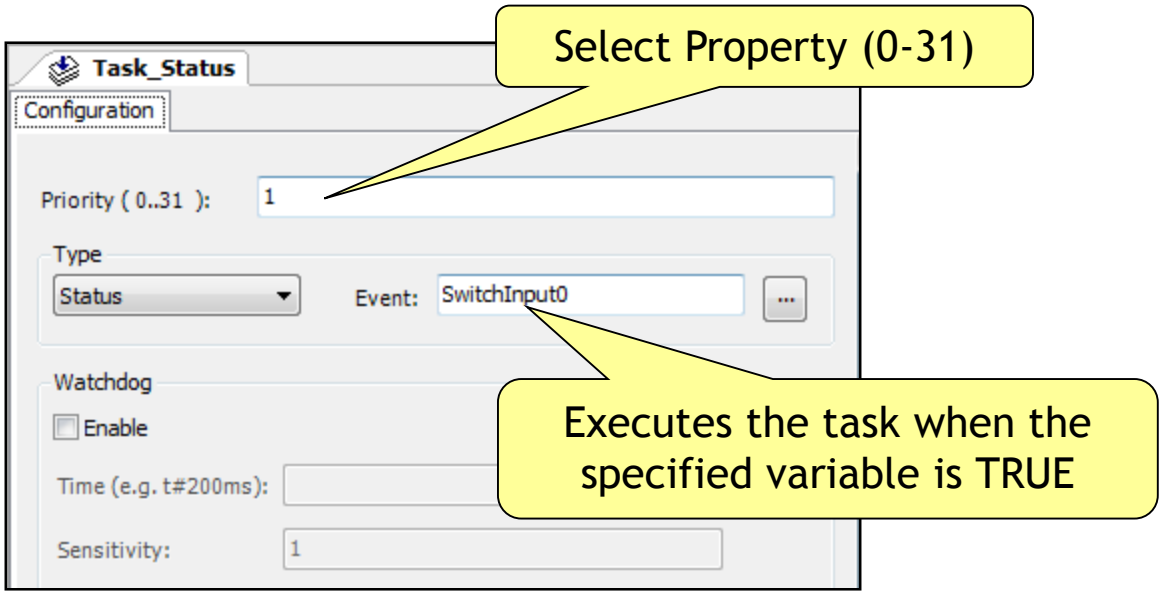


At sampling points 1-4 (magenta) tasks of different types show different reaction:

Behaviour at point:	1	2	3	4
State	no start	start	start	start
Event	no start	start	no start	no start

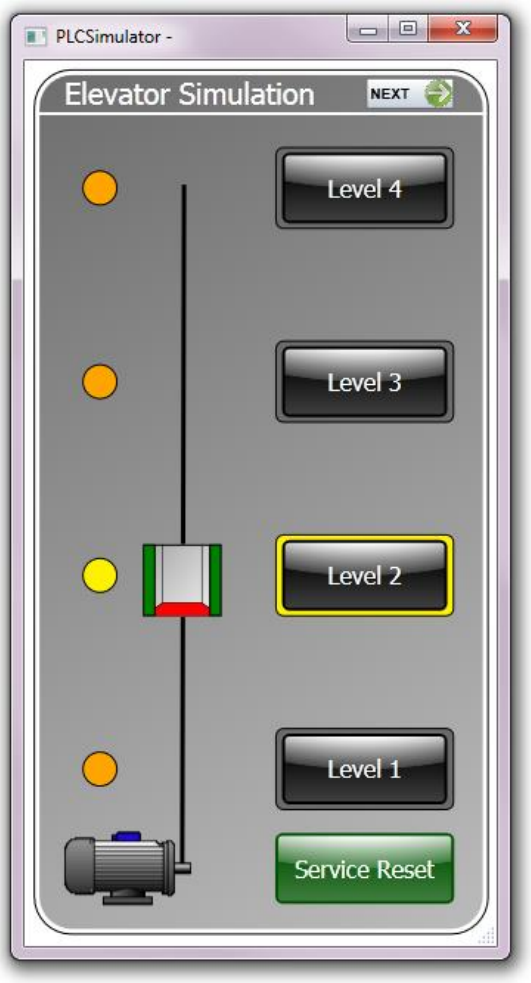
Event Condition

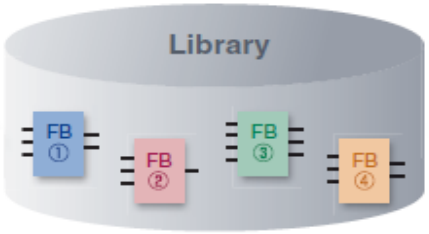
- Task with a “Status” driven execution by SwitchInput0



Exercise, Event Condition

- Manual operation
  - For some reason the elevator stops working and stops between two floors, then it's good if you manually can run the elevator to the next floor
  - Use the "Manual/Auto" switch in the program so that the service personnel should be able to manually control the elevator by additional inputs for manual up and manual down



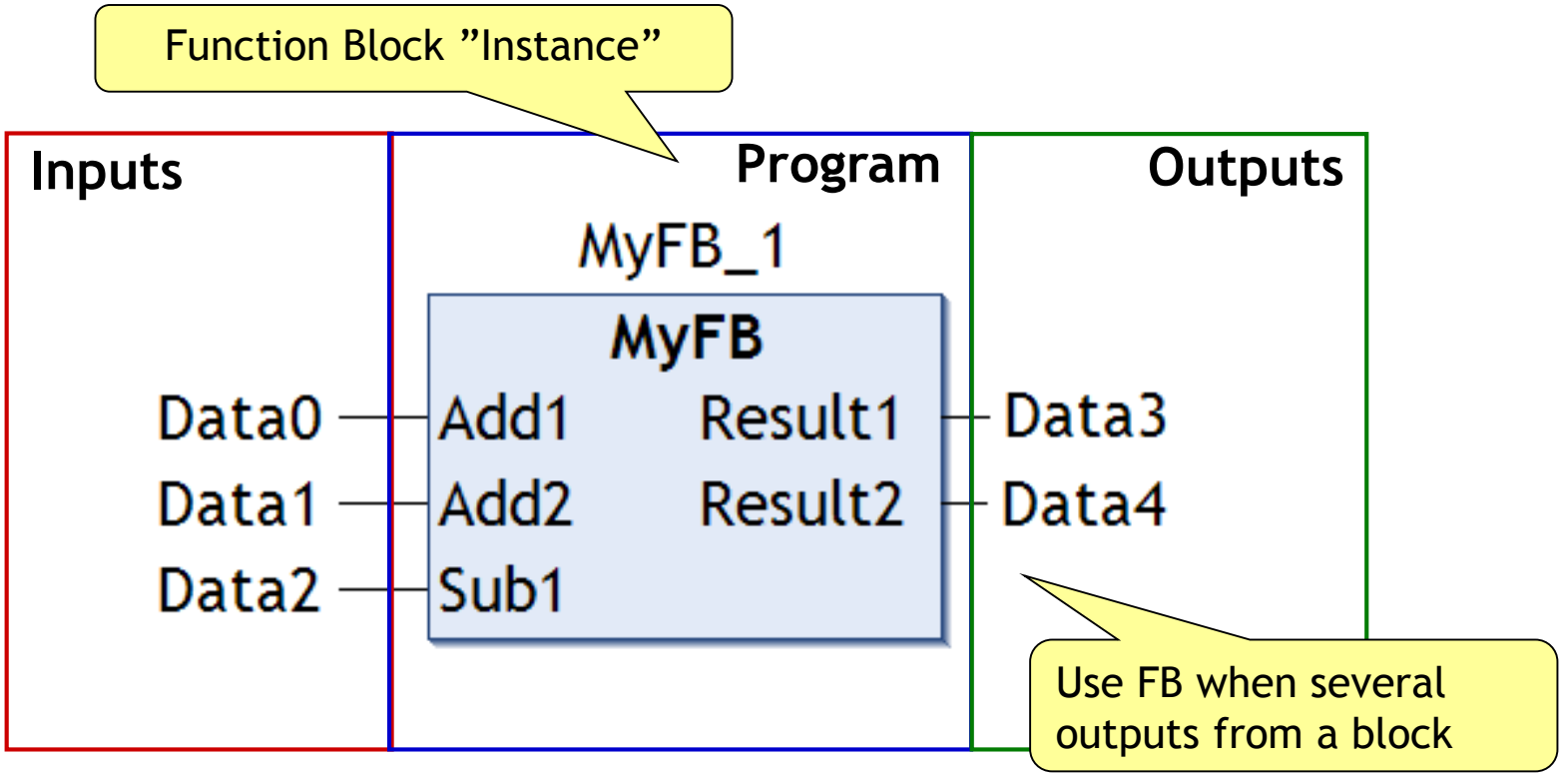


CoDeSys V3

Function Blocks

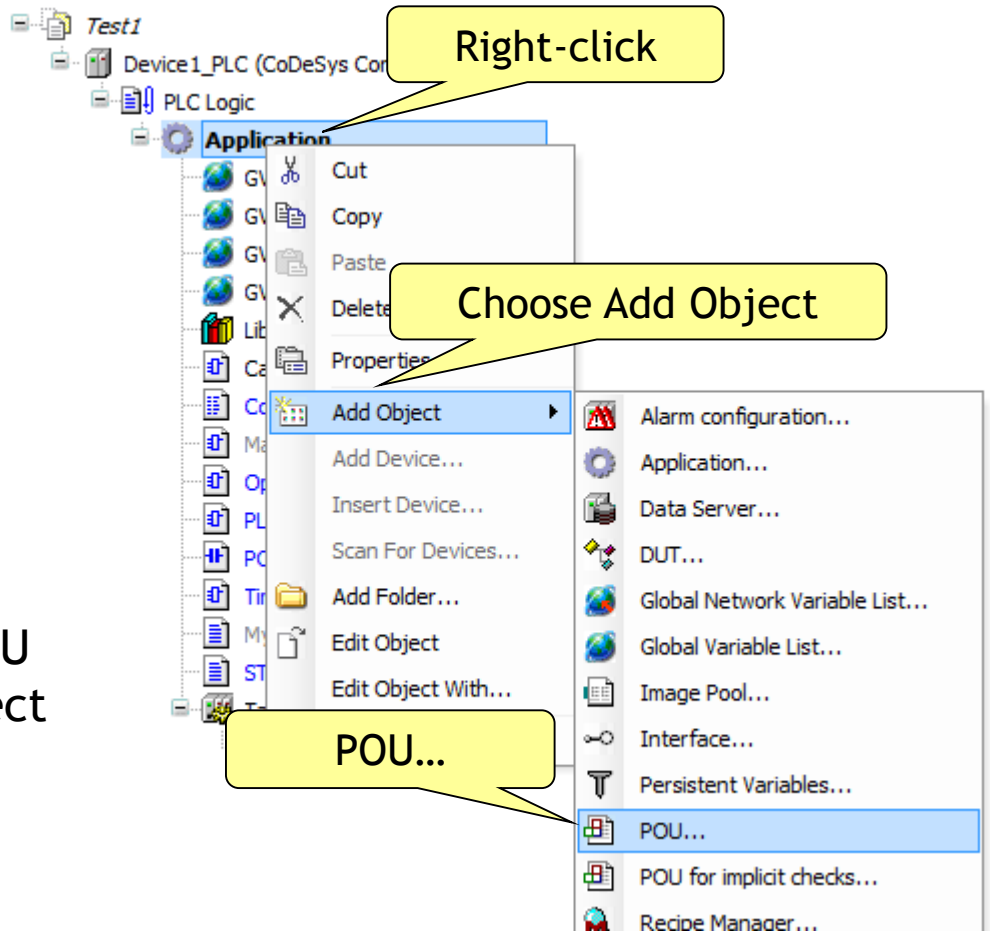
Function Block

- Function Block (FB) or a Function (FUN) are user made subroutines
  - Instead of writing the same program code several times, it can be written once and invoked as a block with new in-/out parameters



Create user defined blocks

- Function Blocks or Functions are created as in separate program components (POU)
  - **FB, Function block**
  - **FUN, Function**



Creating a new Function Block

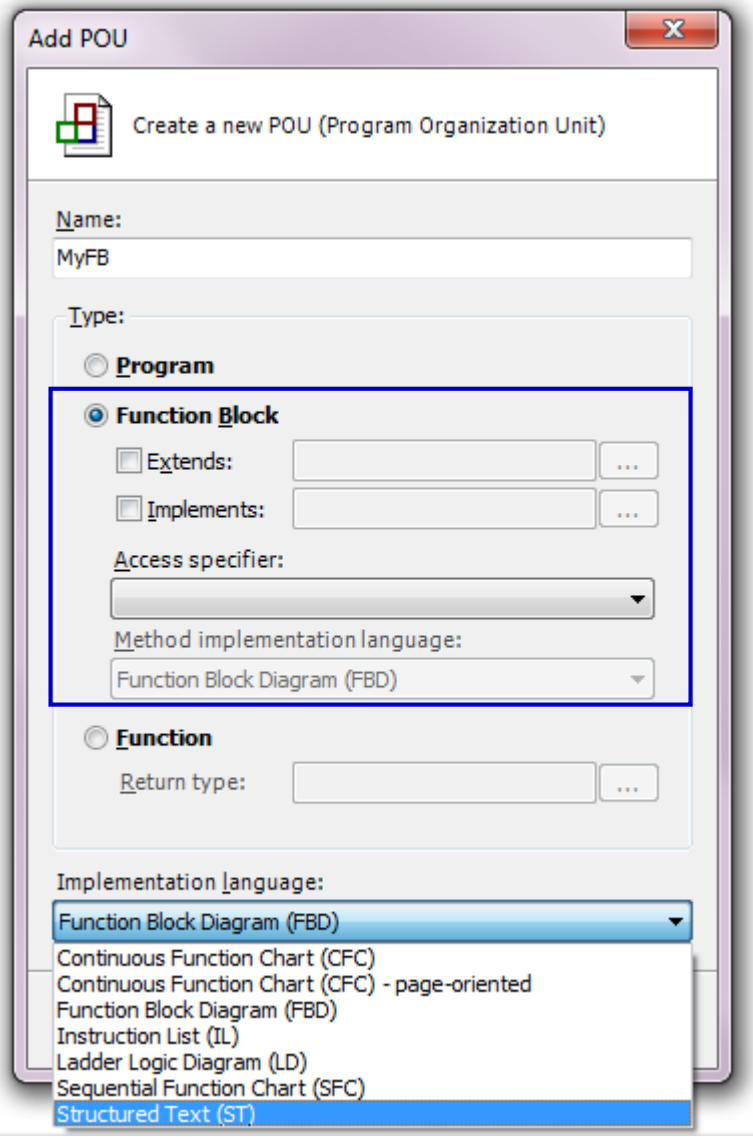
- **Step 1:**
  Menu selection Project/Add Object/POU or Right-click ‘Application’ in the project navigation tree select Add Object/POU

Create a new Function Block

- **Step 2**:
  From the dialog 'Add POU' select:
  Data Type = Function Block

- **Step 3**:
  Enter a Data Name = "MyFB"

- **Step 4**:
  Choose Language = Stuctured Text



- Write the blocks in any IEC-editor
- Blocks can be called from another POU (Program or Function Block)

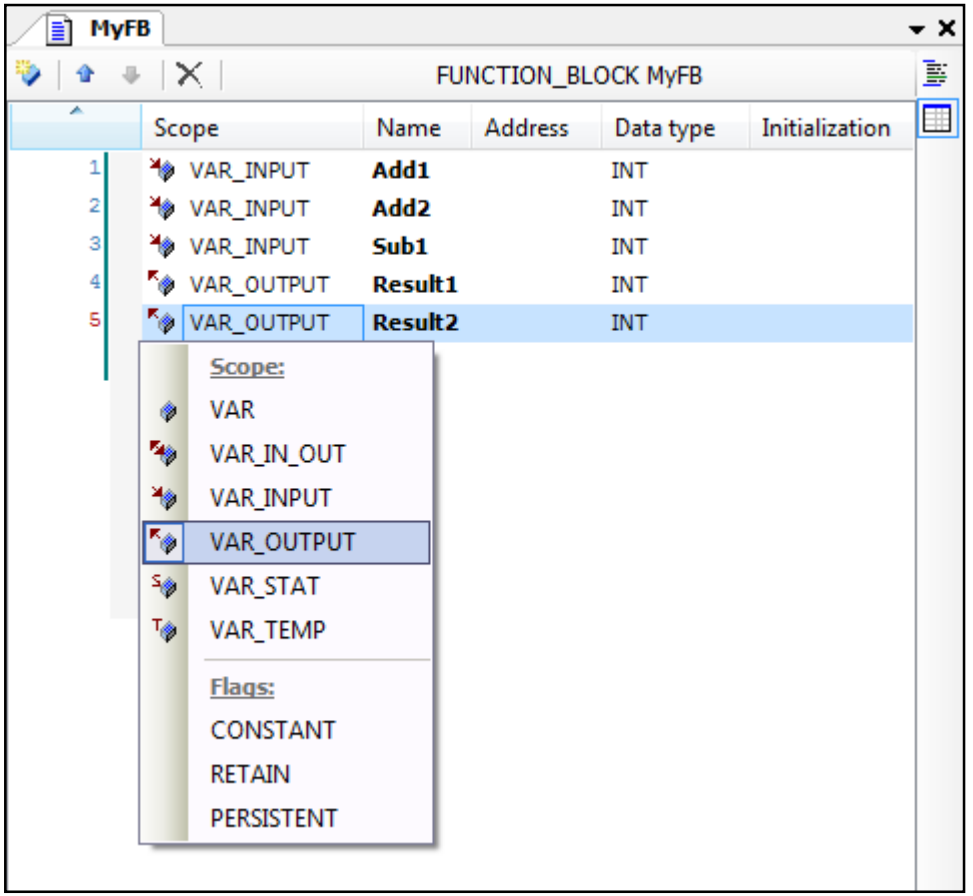
Function Block, Inputs and Outputs

- **Step 5**: Define the following variables for the function block

VAR_INPUT	=	input variable
VAR_OUTPUT	=	output variable
VAR	=	internal variable
VAR_IN_OUT	=	both input and output variable

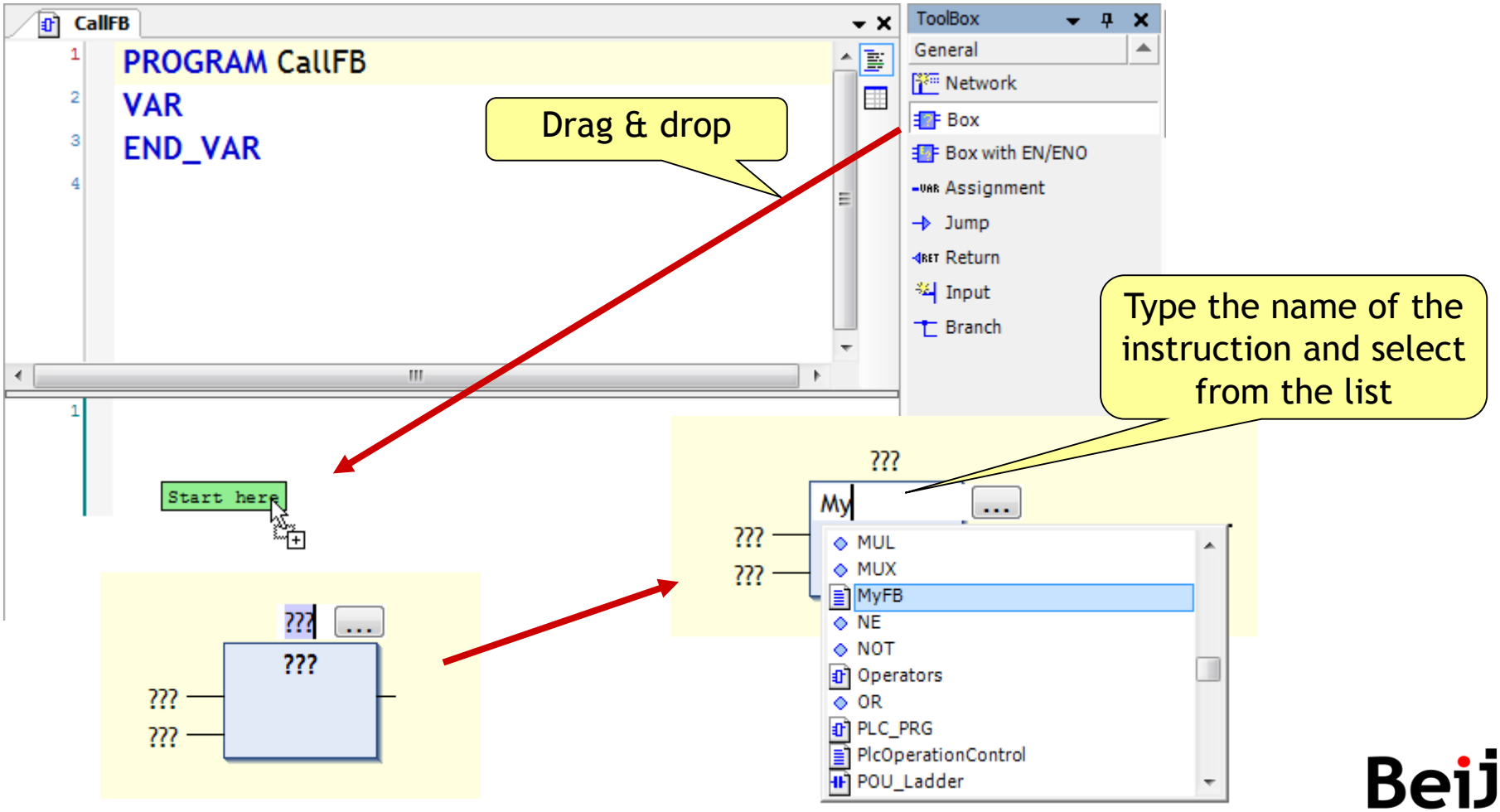
- **Step 6**: Write the code of the block using ST-editor and the variables just defined

```
(*My first Function Block *)
Result1:=Add1+Add2;
Result2:=Result1-Sub1;
```



Function Block, Selection

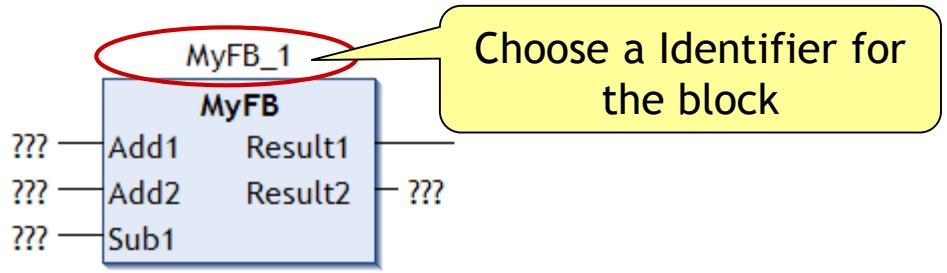
- Drag & drop items from the toolbox to a network in editor (FBD)
- Drop the item at the green field “Start here”

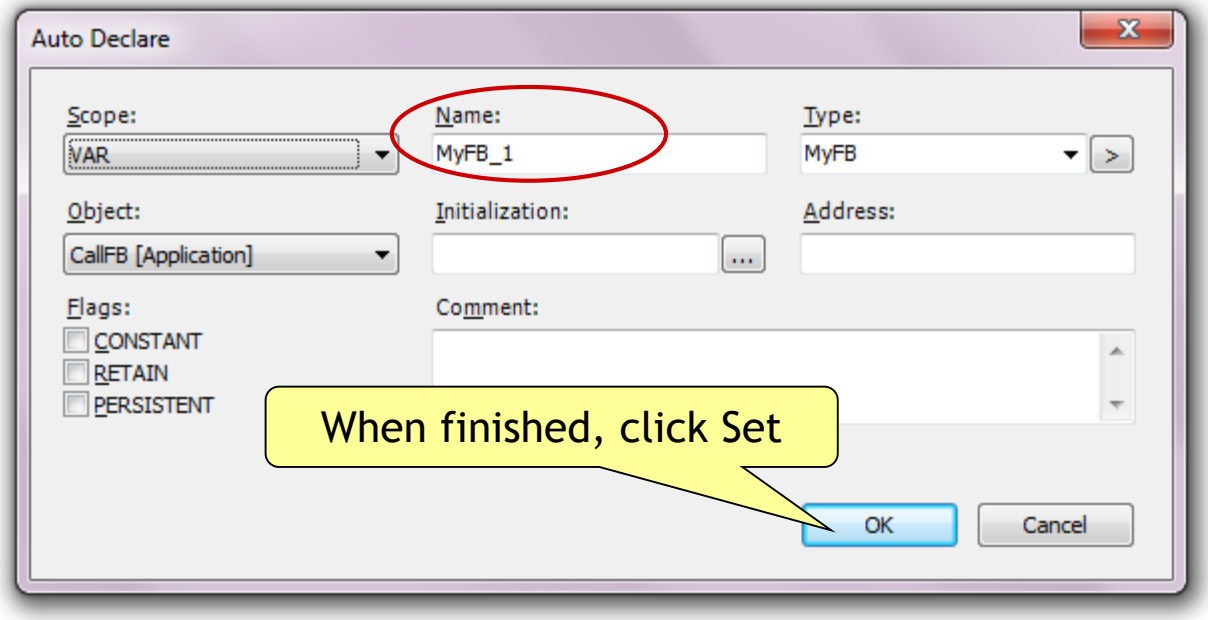


Beijer ELECTRONICS

Function Block, Instance name

- Name the instance of the block in the local or global list

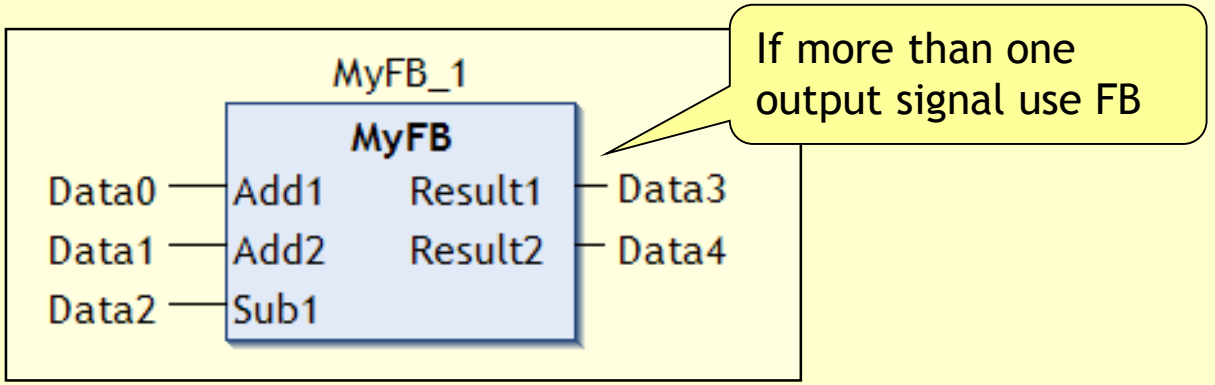




Exercise, Function Block (FB)

- Create the same function block as in the previous example, according to steps 1 to 6
- Try using FBD/Ladder/IL-editor instead of ST-editor if you like
- Download and test the program

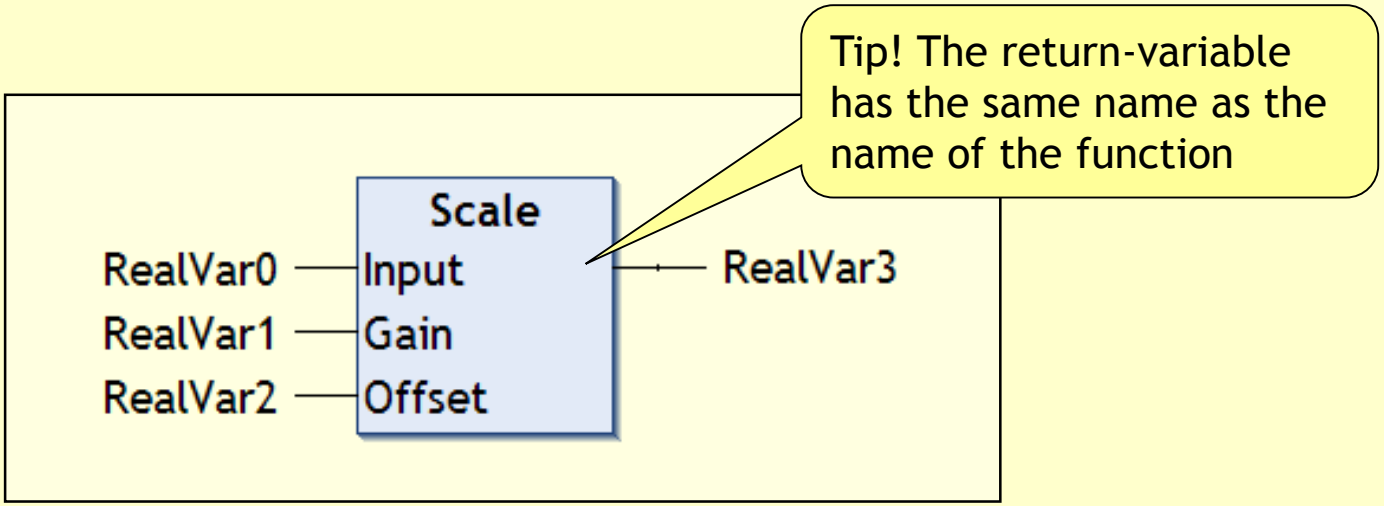
Tip!
Try to monitor the the internal instance of the block



- Additional exercise: E8

Exercise, Function (FUN)

- Create a Function, defined in ST-editor as follows:
  - Scale the input signal with specified gain and offset
  - Formula: **Output := Input*Gain + Offset**
  - Signal type: REAL
  - Result type: REAL



- Additional exercise: E9

CoDeSys V3

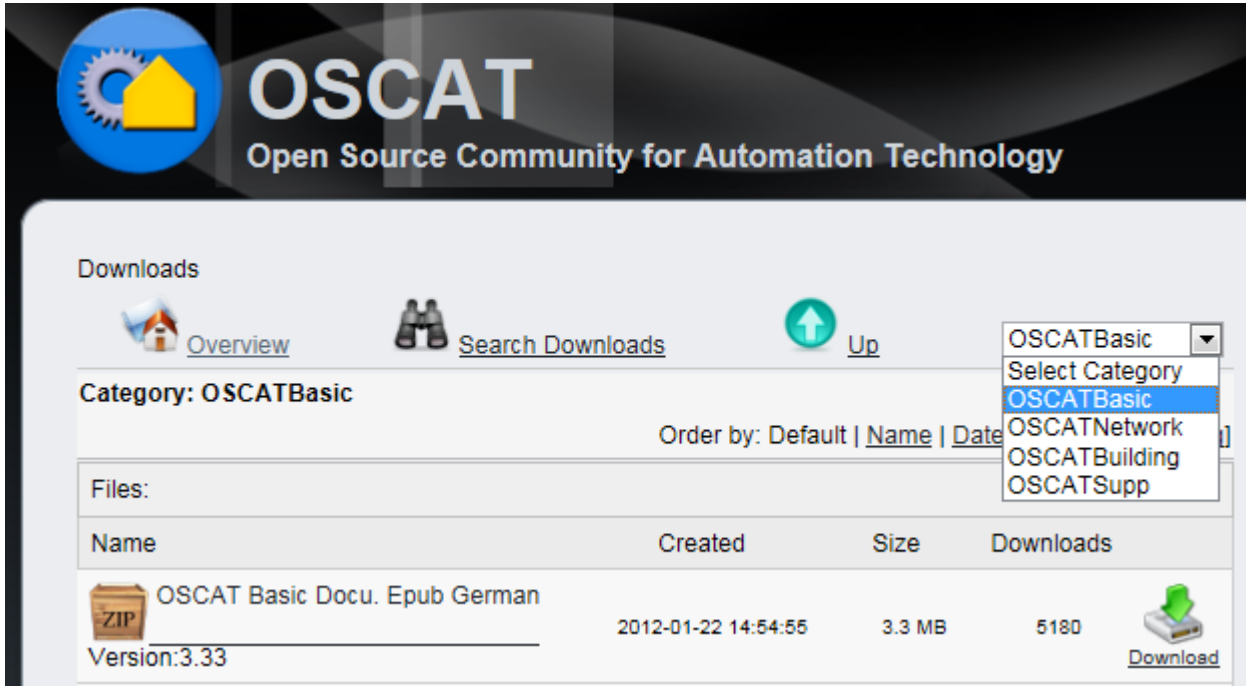
Library management

Libraries in CoDeSys

- Libraries can provide functions and function blocks as well as data types, global variables and even visualizations
- Can be used in the project just like the other POUs and variables which are defined directly within the project
- The default extension for a library file in CoDeSys V3 is ***.library**
- In contrast to ***.lib** used in CoDeSys V2.3 and earlier versions
- Encrypted libraries have the extension ***.compiled-library**
- Libraries might be protected by a license (dongle)

Open source libraries (OSCAT)

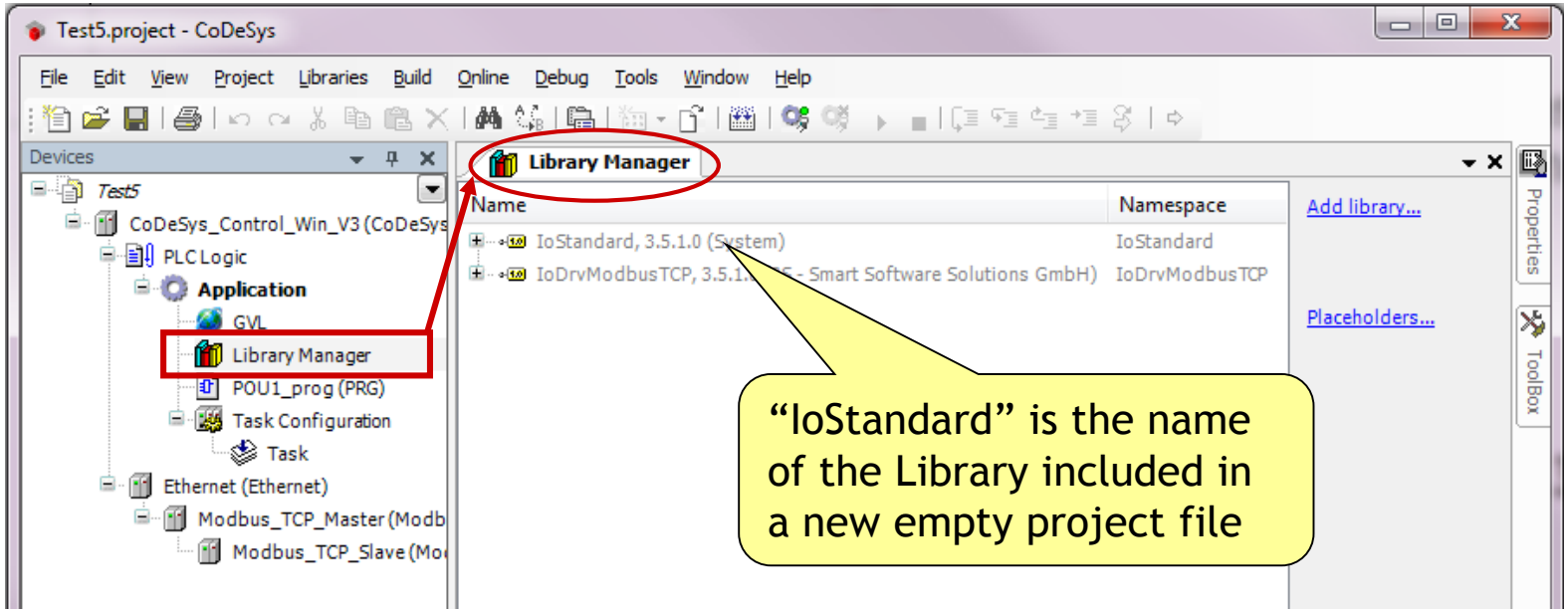
- Libraries can provide functions and function blocks as well as data types, global variables and even visualizations
- Open source CoDeSys libraries on the web, for example:
  http://www.oscat.de/
  http://www.oscat.de/downloadmanager.html



Library Manager

- The management of the libraries in a project is done in the “Library Manager” dialog, and the preceding installation on the system in the “Library Repository” dialog

- The project functions for local and global search and replace also work for included libraries



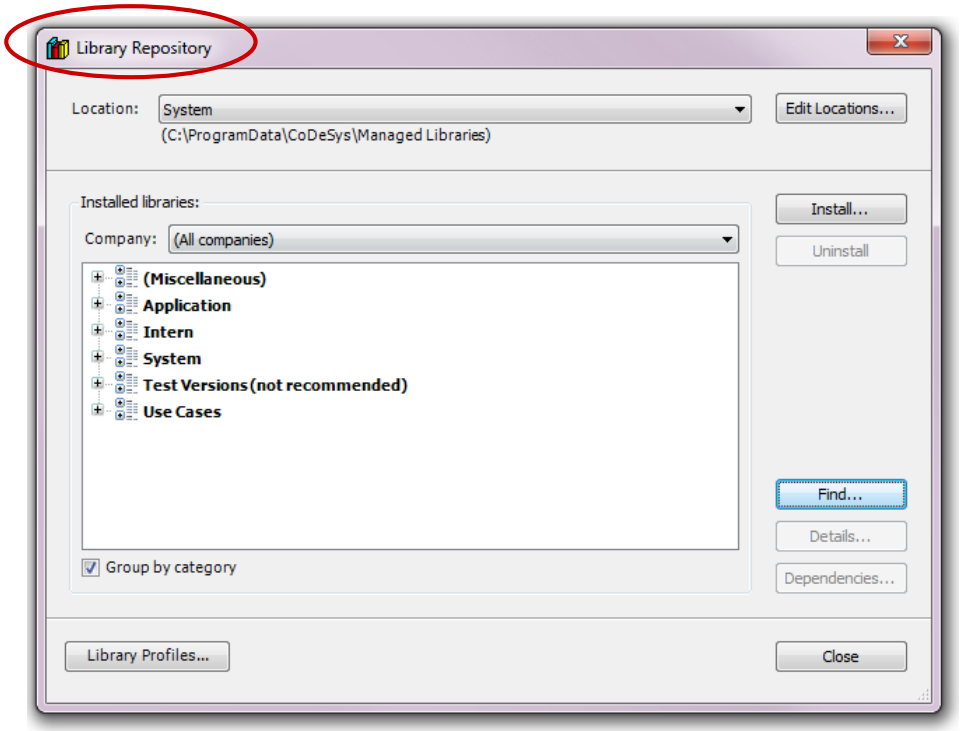
Library Repository

- The “library repository” is a database for libraries which have been installed on the local system in order to be available for getting included in CoDeSys projects

- The Library Manager Object provides the command “Library Repository” for handling library locations, installing and uninstalling of libraries

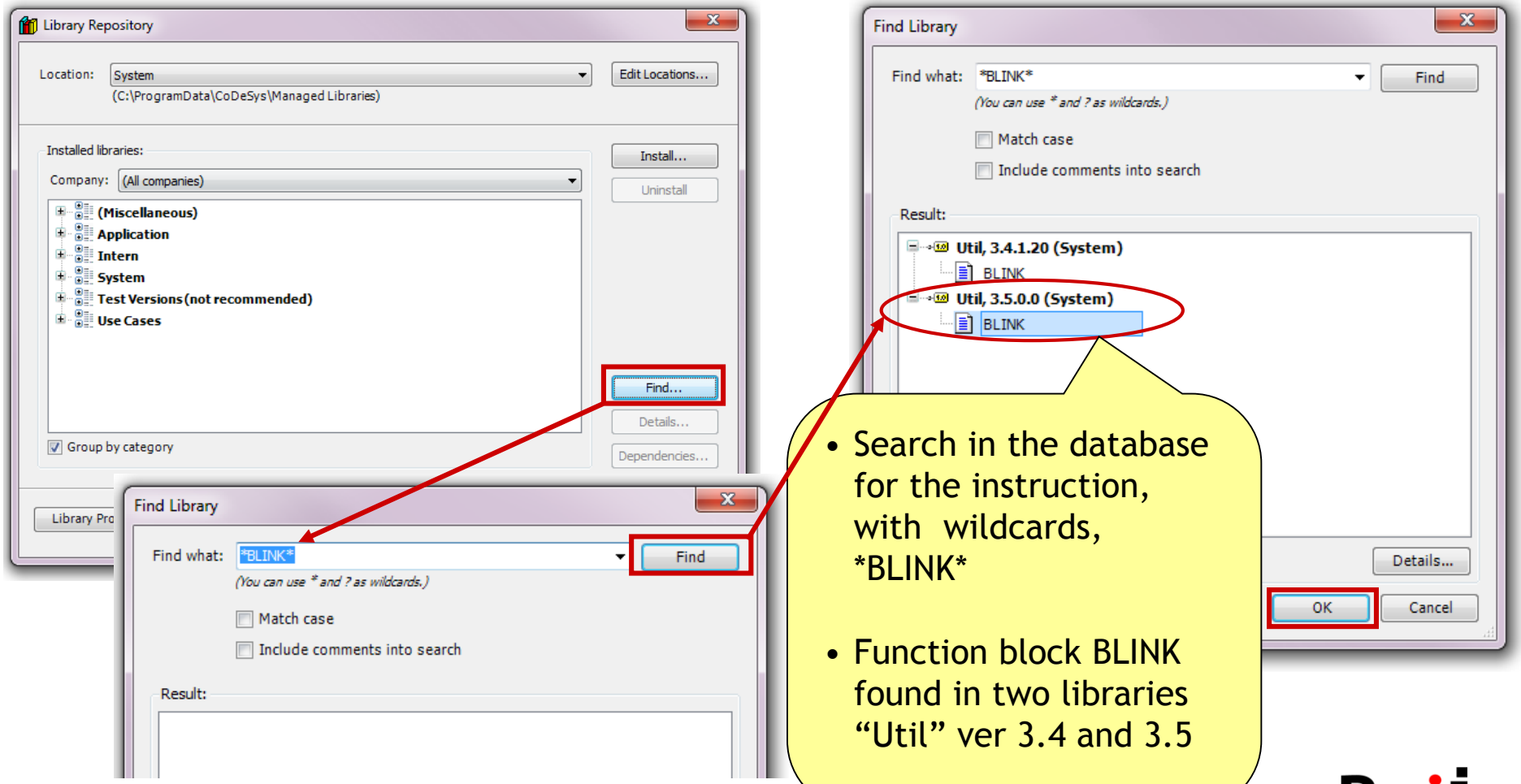
- By default this command is part of the Tools menu. If necessary, open the Customize dialog to view respective to modify the menu configuration

- For general information on the Library Management please see the online help



Find new functions in library

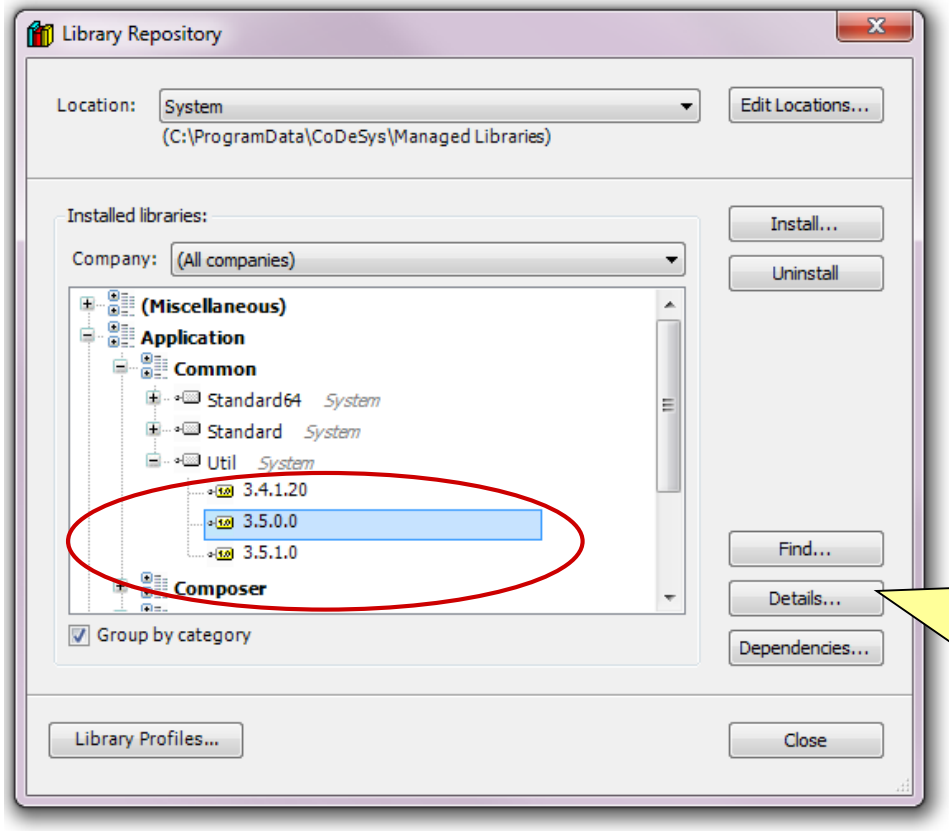
- Find a new instruction …



- Search in the database for the instruction, with wildcards, *BLINK*
- Function block BLINK found in two libraries “Util” ver 3.4 and 3.5
- Then end with OK

Add library to repository

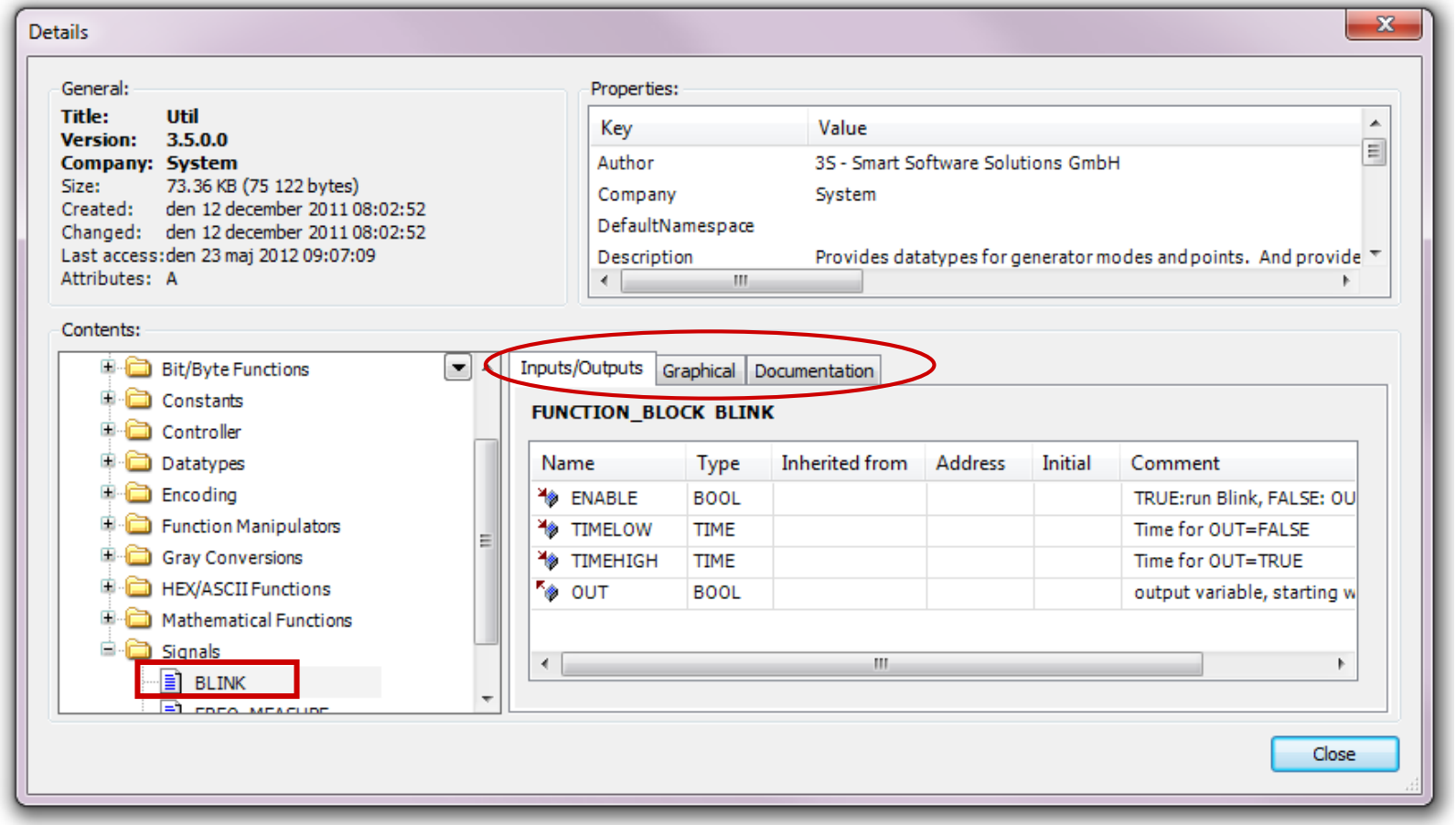
- The selected library “Util” was included in the repository



Details will show information about the instructions included in this library

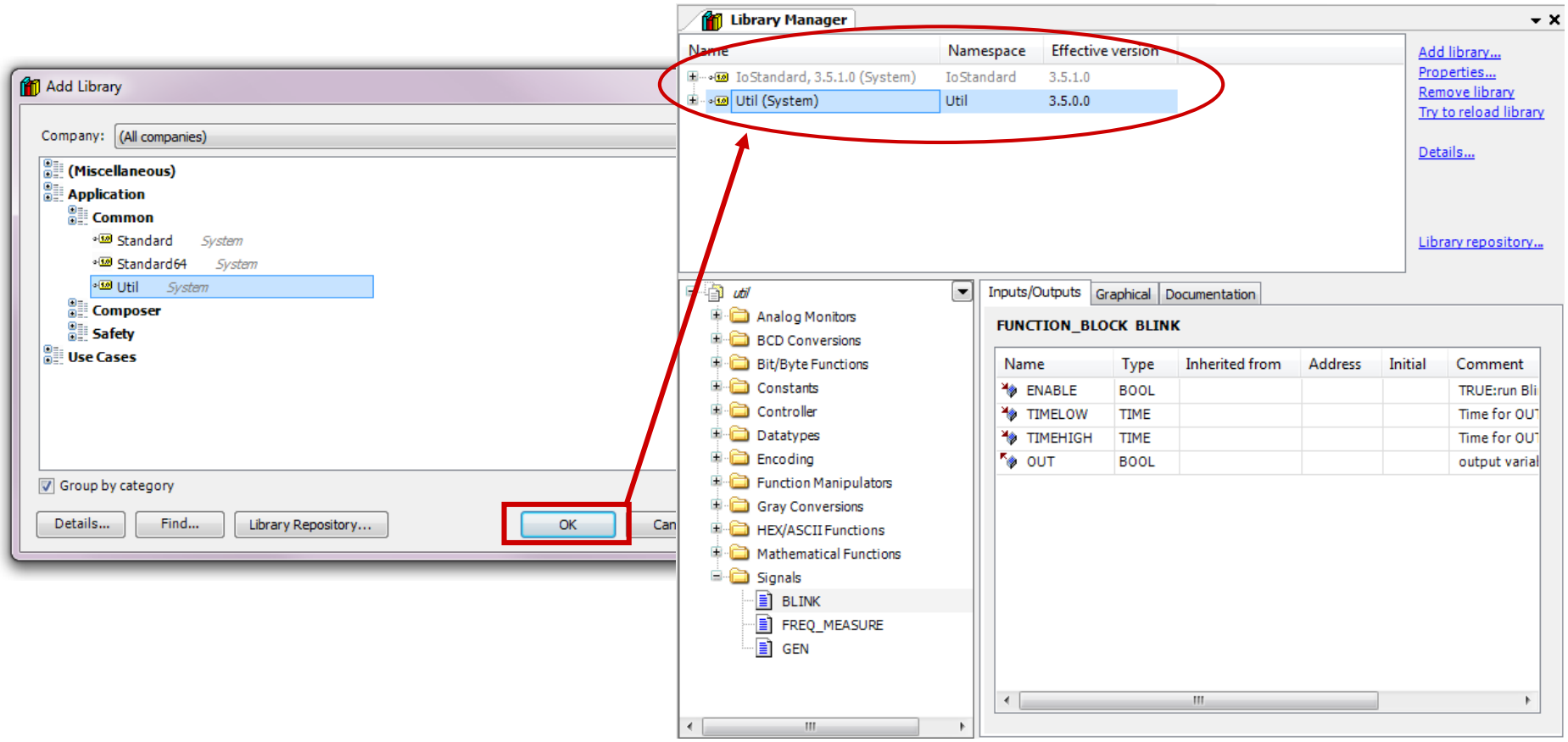
Show details of instruction

- Show the details of the instruction, for example BLINK
- Tabs for Inputs/Outputs, Graphical and Documentation are available

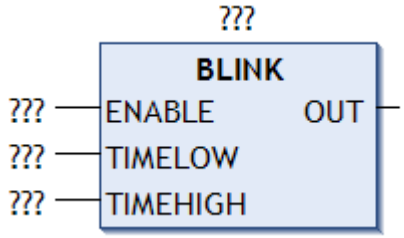


Add library to project

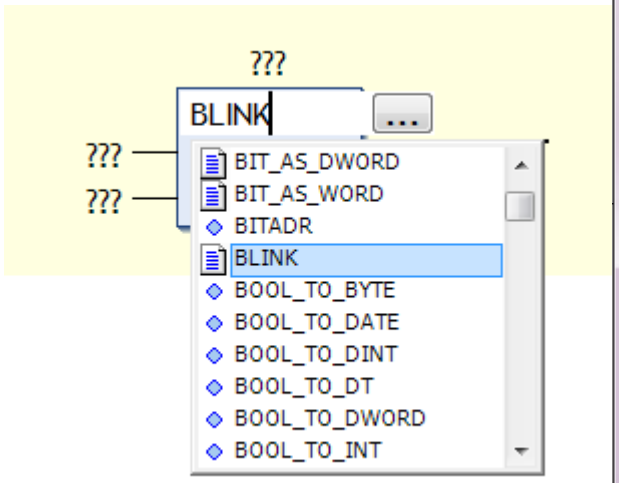
- Add the library “Util” to the project
- Now it will show in the list of libraries of the Library Manager

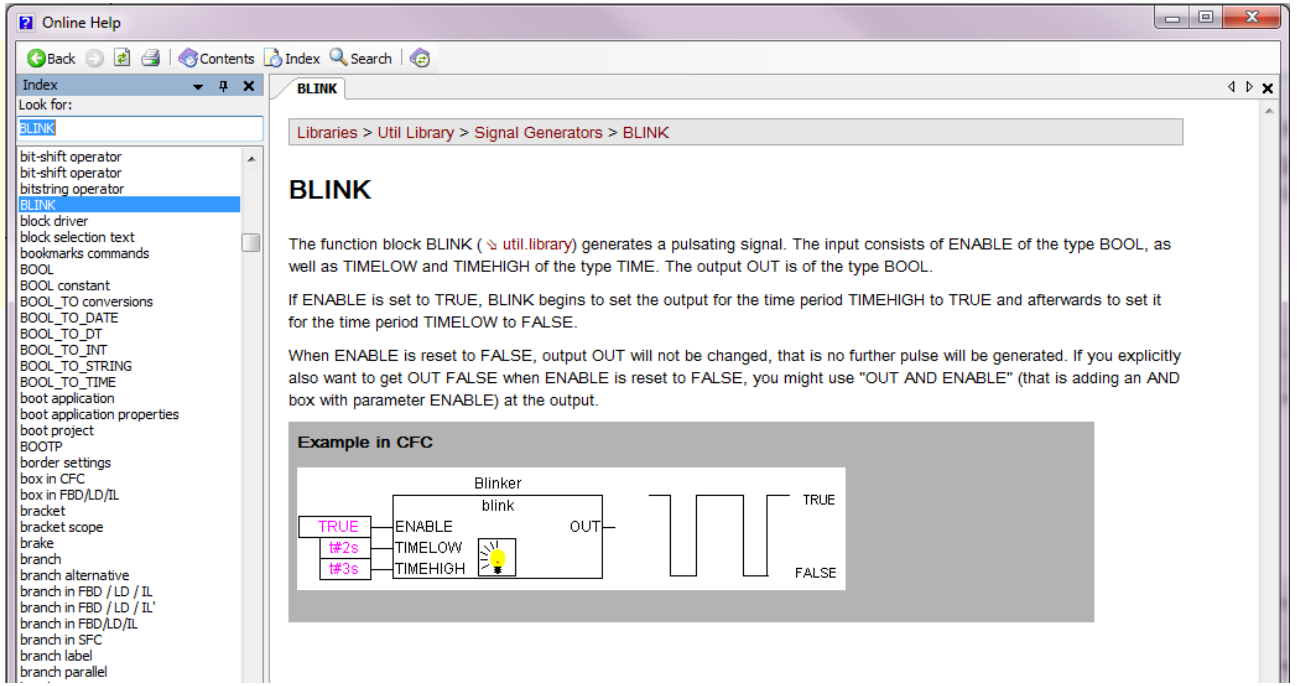


Use instruction in program



- Add the new function block to the program code
- Create an instance of the function block and attach variables
- Open the help of BLINK instruction with shortcut [F1]



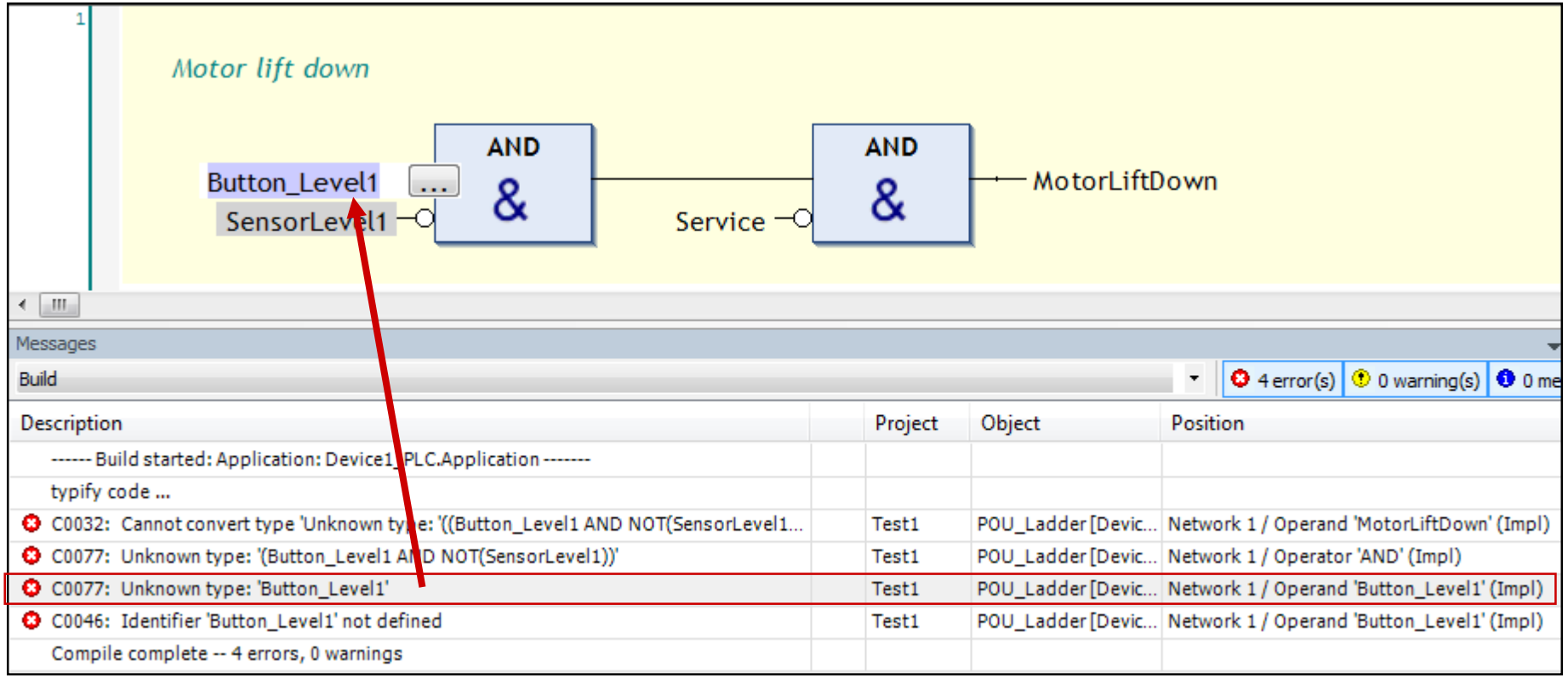


CoDeSys V3

Diagnostics and Other features

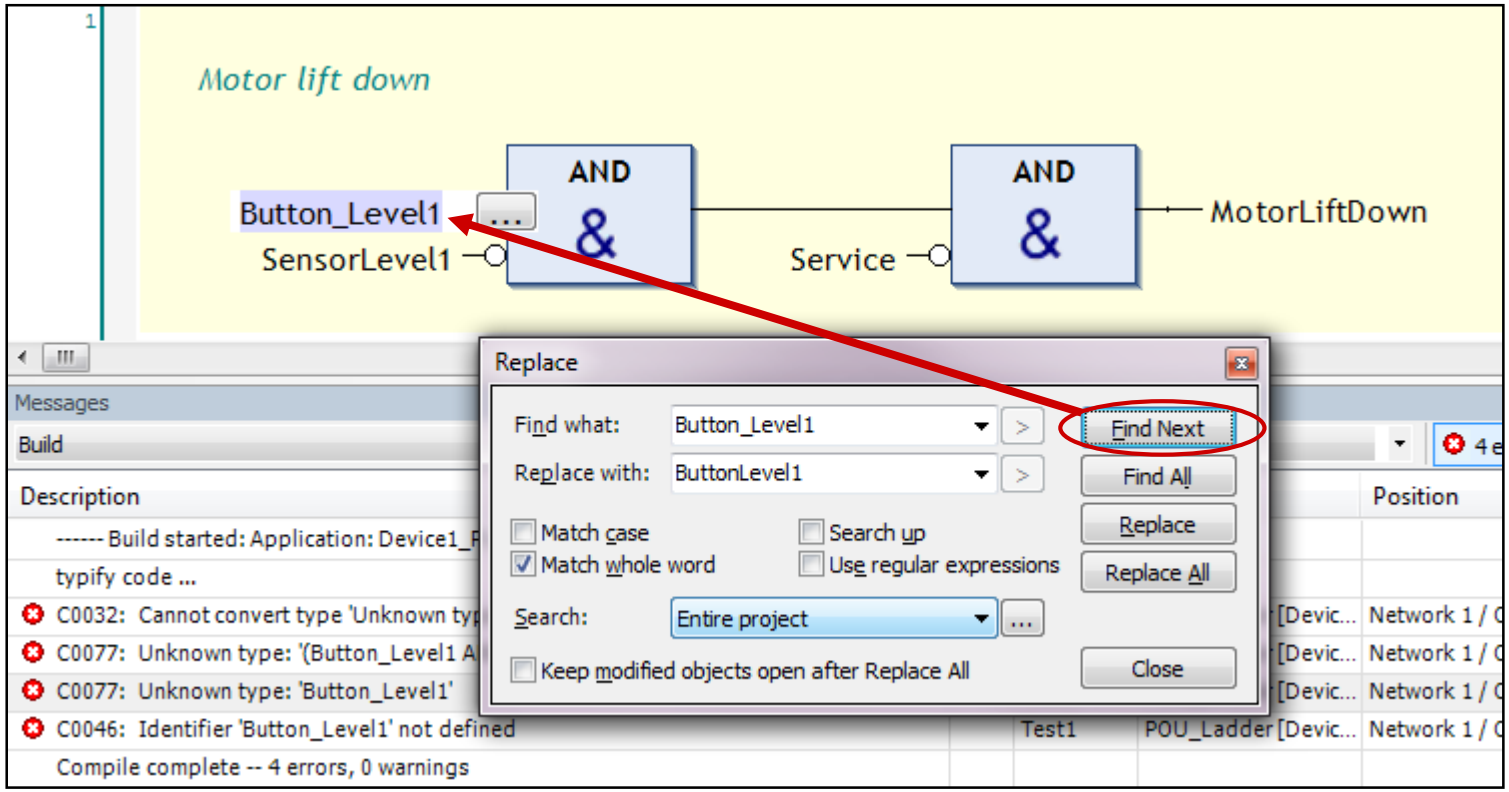
Correcting Errors and Warnings

- Menu selection View/Messages [Alt+2]
- Open the location of the error/warning by double-click of the message



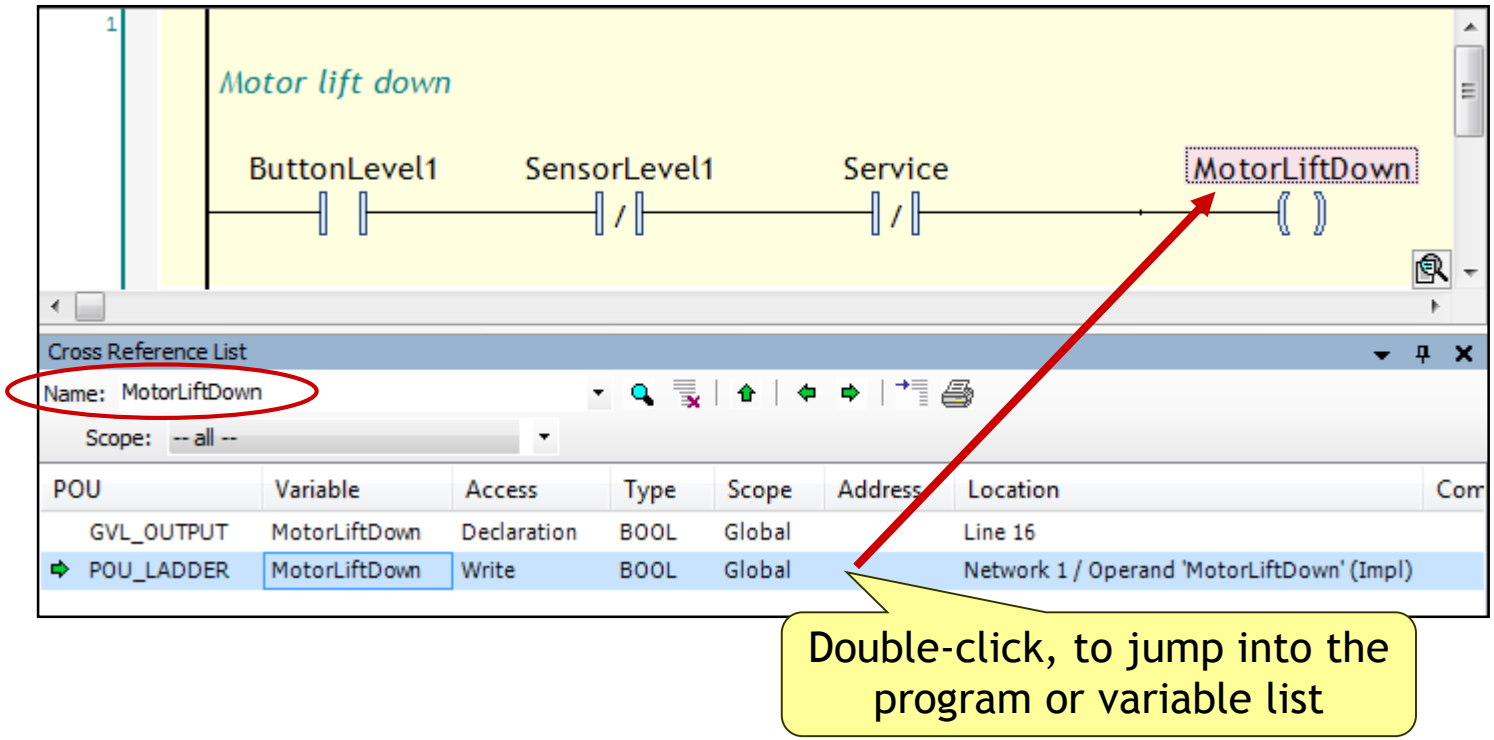
Find/Replace

- Menu selection Edit/Find Replace
- Searching for and replacing variables in the program



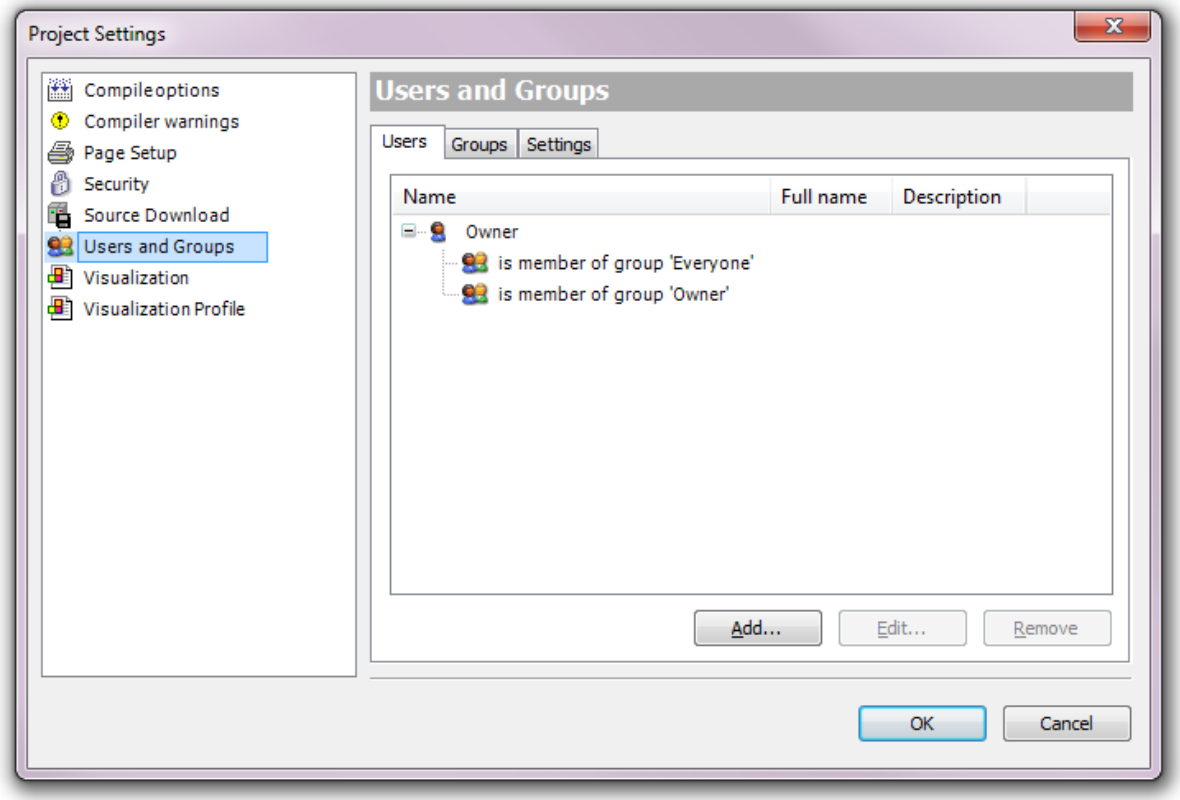
Cross Reference

- Menu selection View/Cross Reference List, opens a window with the cross references of a project variable
- It will show the locations where the variable is used within the project or just within the scope of the same POU, open location with double-click

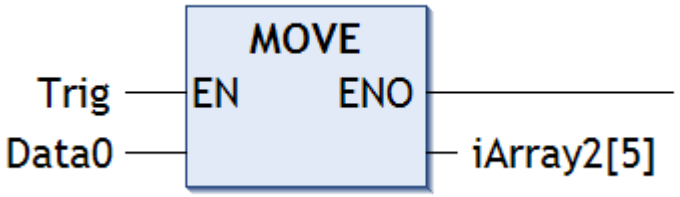


Security (Users and groups)

- Menu selection Project/Project settings/Users and Groups, provides three dialogs for the user management of the current project: Users, Groups, Settings…
- The access control for projects particular objects responsibilities, the right to perform certain actions in a project can be configured and managed via dialogs of the Project Settings, object Properties and User Management…



Array / Indexing

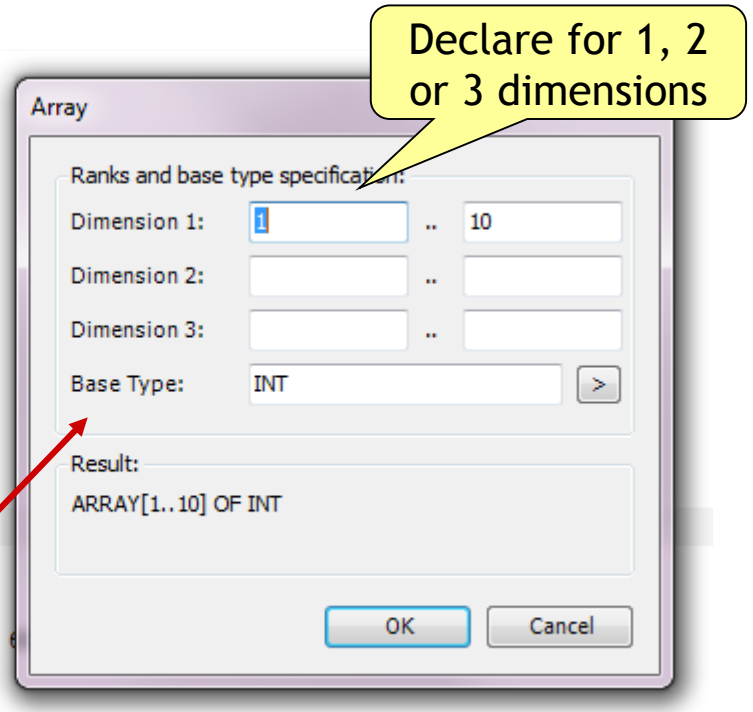


- Vector Management with IEC 61131-3

- An ARRAY is a collection of elements of same datatype

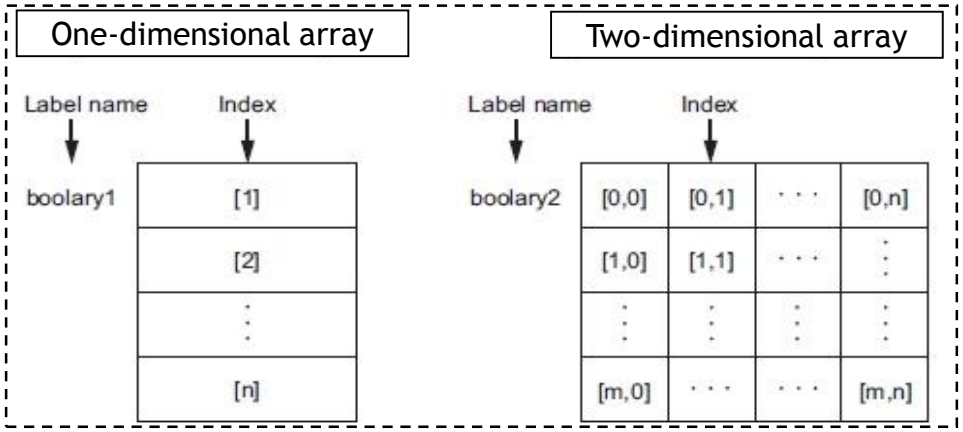
```
// Array of Word
wArray1: ARRAY[0..5] OF WORD;
// Array of Integer
iArray2: ARRAY[1..10] OF INT;
// Array of Bool
iArray3: ARRAY[1..16] OF BOOL;
```

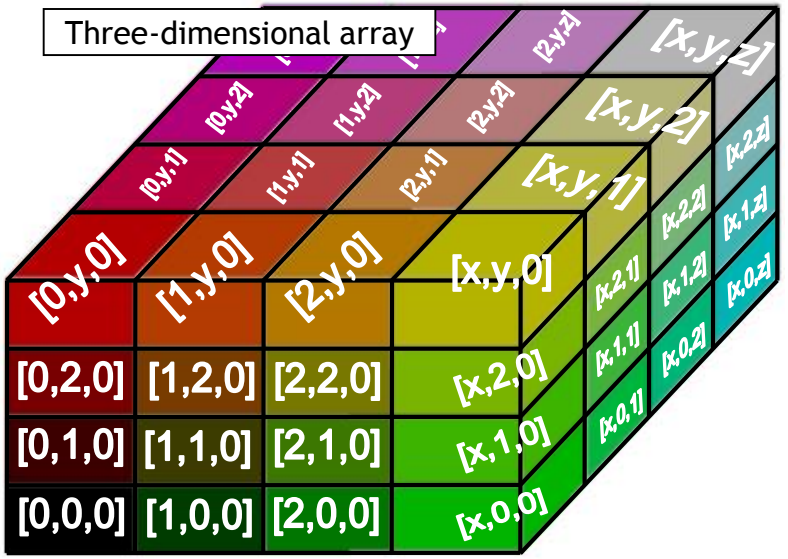
- Wizard for Array declaration available:

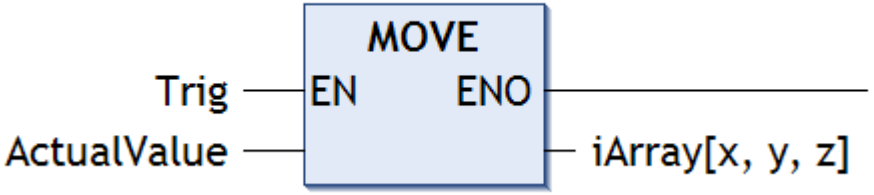


Arrays “LabelName[Index]”

- One-, two-, and three-dimensional Arrays are supported as elementary data types … <Array-Name>[Index1, Index2, Index3]
- Arrays can be defined both in the declaration part of a POU and in the global variable list
- Use constant or index for addressing







Indexing Example

- The operator can select an recipe number from the panel, that will make the Recipe Area load one of the 10 recipes to the Selected Recipe area in the PLC

Recipe#

Array???

Recipe Area

Recipe 1 — Reg1 ⋮ Reg20

Recipe 2 — Reg21 ⋮ Reg40

Recipe 3 — Reg41 ⋮ Reg60

Recipe 10 — Reg181 ⋮ Reg200

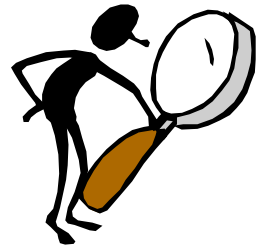
Array of 1..20

Selected Recipe

- A recipe may contain various parameters:
  - Number
  - Quantity
  - Color Code
  - Timer value
  - Temperature, etc…

Beijer ELECTRONICS

Indexing Example (ST-editor)

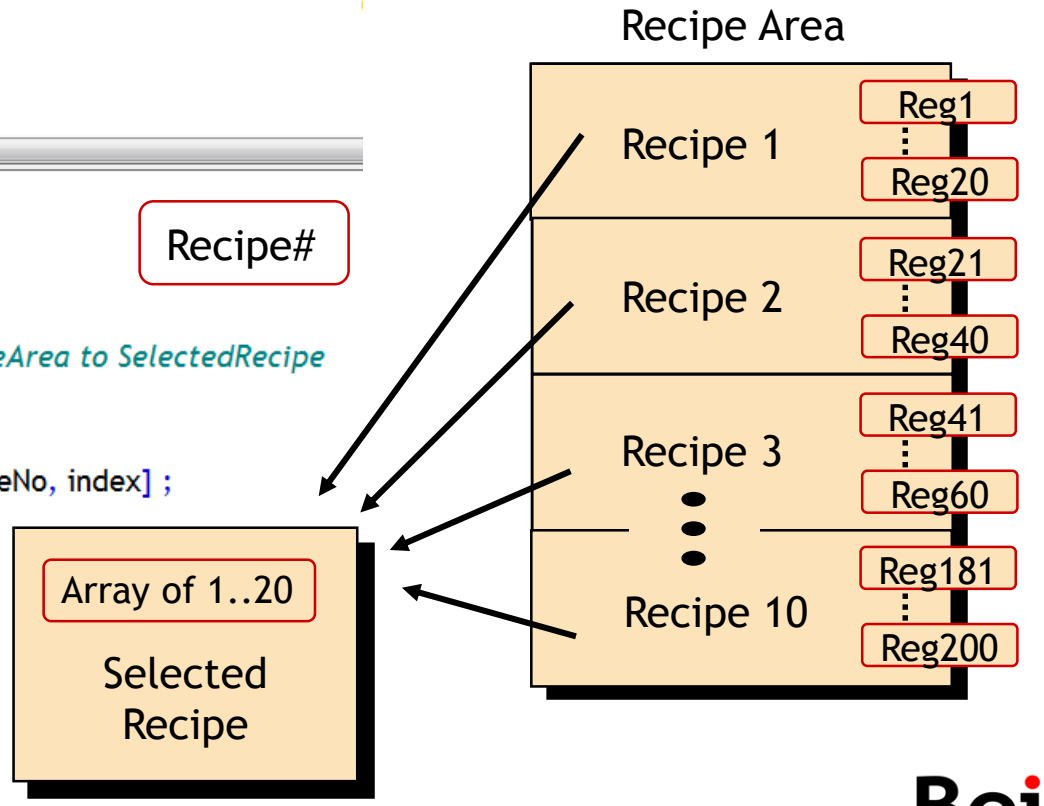


```
PROGRAM ST_PRG
VAR
    Trig: BOOL;
    RecipeNo: INT := 1; (*Default value*)
    SelectedRecipe: ARRAY[1..20] OF INT
    RecipeArea: ARRAY[1..10, 1..20] OF INT
    index : INT ;
END_VAR
```

```
//Check recipe number >0
IF RecipeNo=0 THEN
    RecipeNo:=1;
END_IF
//Loop 20 registers and block move from RecipeArea to SelectedRecipe
IF TRIG THEN
    FOR index:= 1 TO 20 DO
        SelectedRecipe[index] := RecipeArea[RecipeNo, index] ;
    END_FOR
END_IF
```

Using a Two-dimensional Array

Recipe#

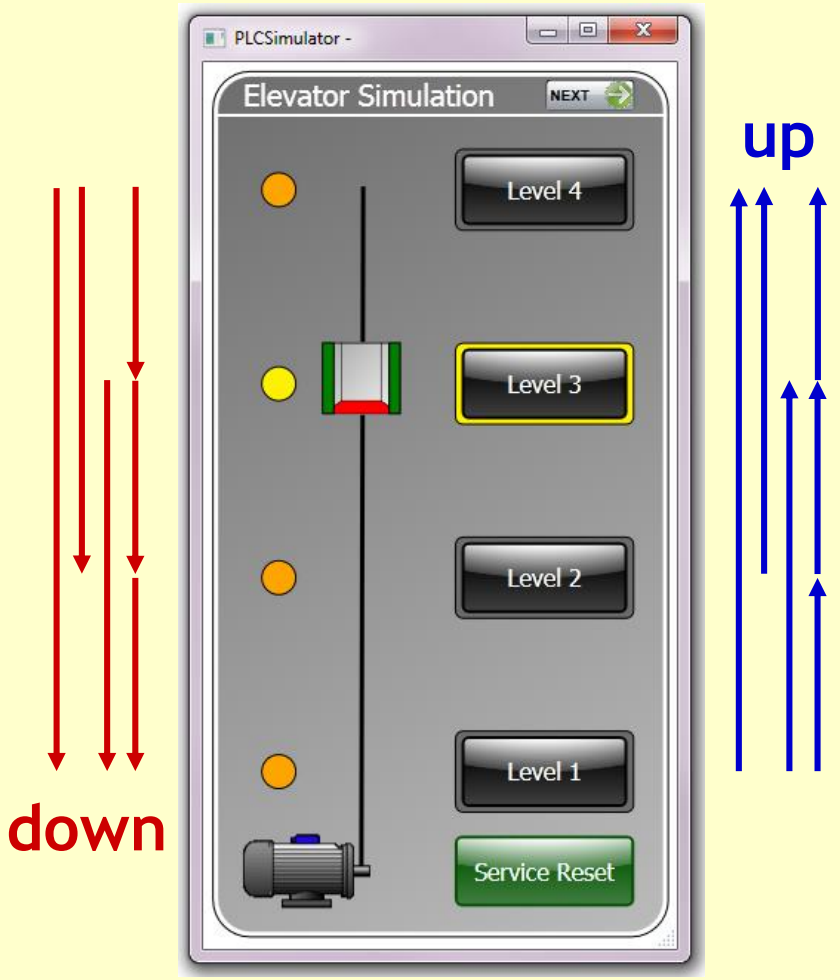


Beijer ELECTRONICS

Final Exercise, Elevator of four floors

- Elevator with memory
  - Improve the program so that the elevator can handle all 4 floors
  - Remember that you can get to the 2nd and 3rd floor from two directions
  - Tip, find out all possible routes to all floors and create a solution that uses memory for every possible route, declare in GVL_Memory list

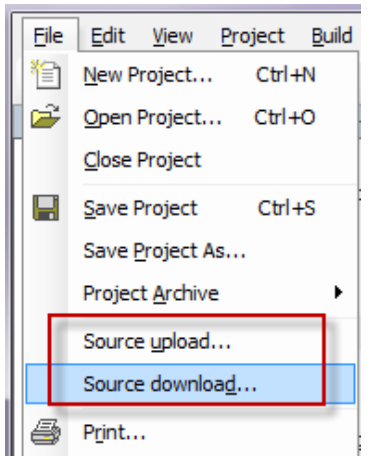
```
//Memory variables
VAR_GLOBAL
        MemoryDown: BOOL; // Down
        MemoryUp: BOOL; // Up
        Mem432to1: BOOL;
        Mem1to2: BOOL;
        Mem43to2: BOOL;
        Mem12to3: BOOL;
        Mem4to3: BOOL;
        Mem123to4: BOOL;
END_VAR
```



CoDeSys V3

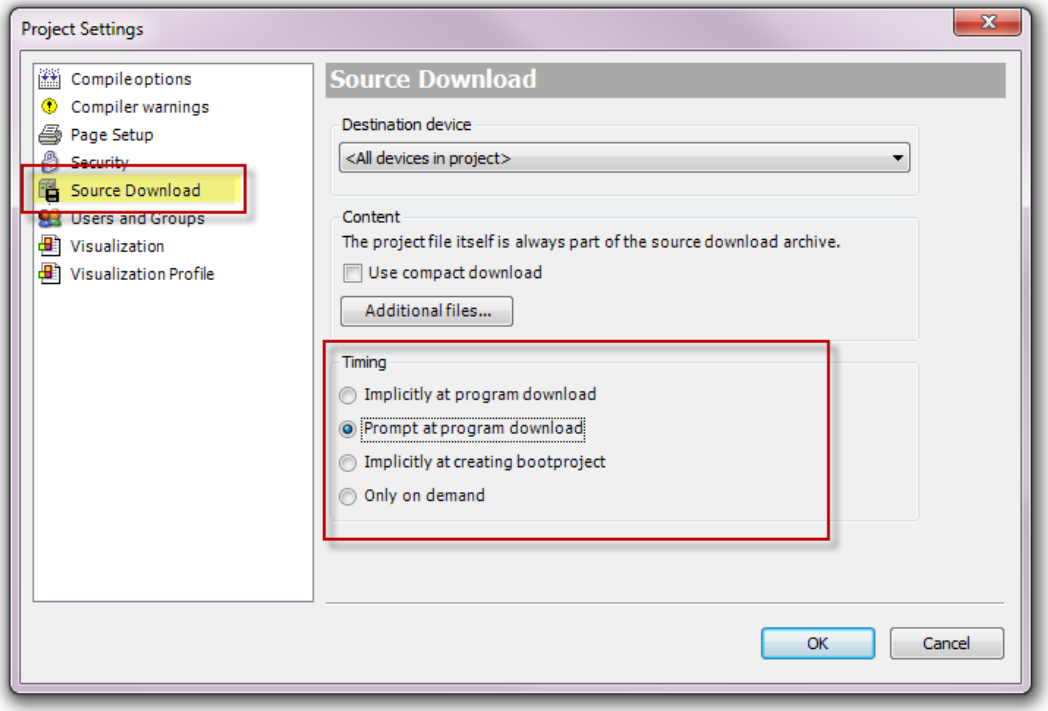
Project backup

Source upload / download



Source code download and upload

- Menu selection File/Source download…
- CoDeSys does not support the disassembling of downloaded projects! A much better option is the **source code download** where the whole project including all the graphical information is available on the controller device. All the security mechanism are available as well.
- Select Timing option in the menu selection Project/Project Settings to make it automatic.



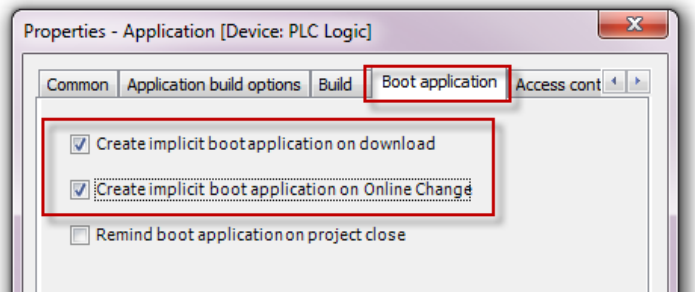
Boot application / Download file

Boot application download

- CoDeSys supports the generation of boot project, the “Boot application” will be loaded automatically when the PLC gets started.
- Note that Boot after Online Change and Restart of Device, must be done to make a safe restart after power off.
- Highlight the "Application" option in the "Device" window and right click, select "Properties" and "Boot application“.

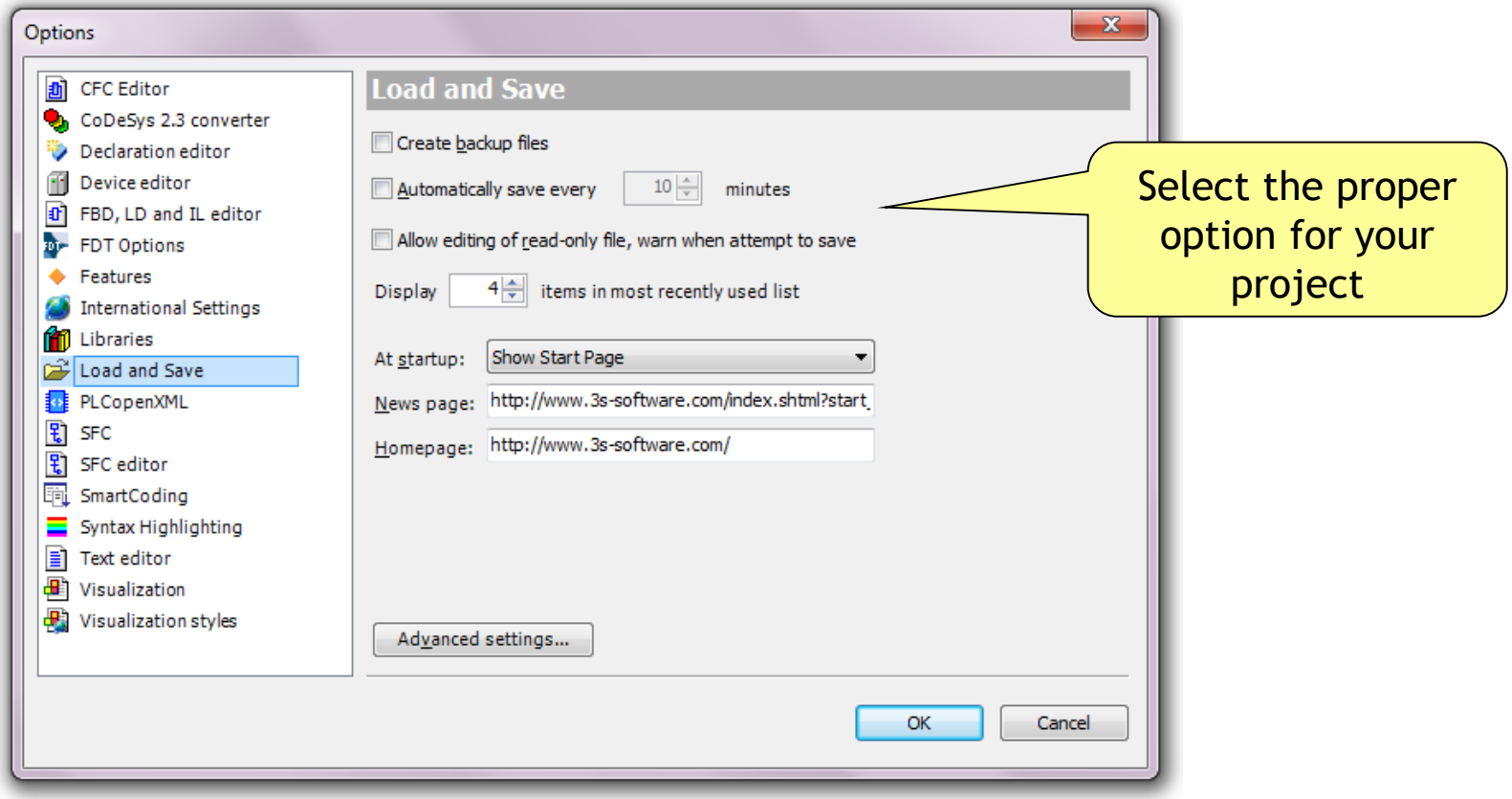
Download / Upload of a file

- CoDeSys supports the storage of any file on the controller. This can be very helpful in order to be able to use the target controller as a storage medium.



Load and Save options

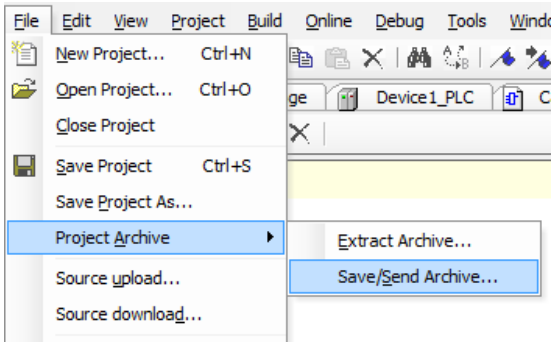
- Create backup files - If this option is activated, at each saving the project will not only be saved in <projectname>.project but also copied to a file <projectname>.backup.
- If needed you can rename this backup-file and re-open in CoDeSys.



Project backup

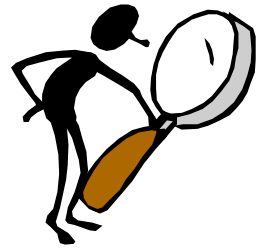
- Menu selection File/Project Archive
- The best way to get all components from a CoDeSys project is to make a '**Save/Send Archive**'
  - That will save and pack all files referenced by and used within the currently opened project in to one archive file!
  - The archive file can either be stored or sent as attachment of an email
- The archive file can easily be unpacked by use of '**Extract Archive**'
- Note, the archive function is not intended for restoring a project environment. It is designed for an easy packing of all files belonging to a project!

- All supported files are:
  - CoDeSys project archive (from V3) ***.projectarchive**
  - CoDeSys project files (from V3) ***.project**
  - CoDeSys library files (from V3) ***.library**
  - CoDeSys project files (before V3, i.e. V2.3) ***.pro**
  - CoDeSys library files (before V3, i.e. V2.3) ***.lib**



- CoDeSys library files from V3.0 has extension "**.library**" additionally there might be further file type options depending on the available project converters

CoDeSys, how to backup process data?



- How to make backup of process data from the Soft PLC (CoDeSys) to computer?
  - Use menu selection Project / Add object / Recipe Manager
  - Recipe Manager will create files of extension **".txtrecipe"**

- Procedure how to make backup of variable values from the PLC to a file in a computer using "Recipe Manager" in the CoDeSys application, can be found in below link.

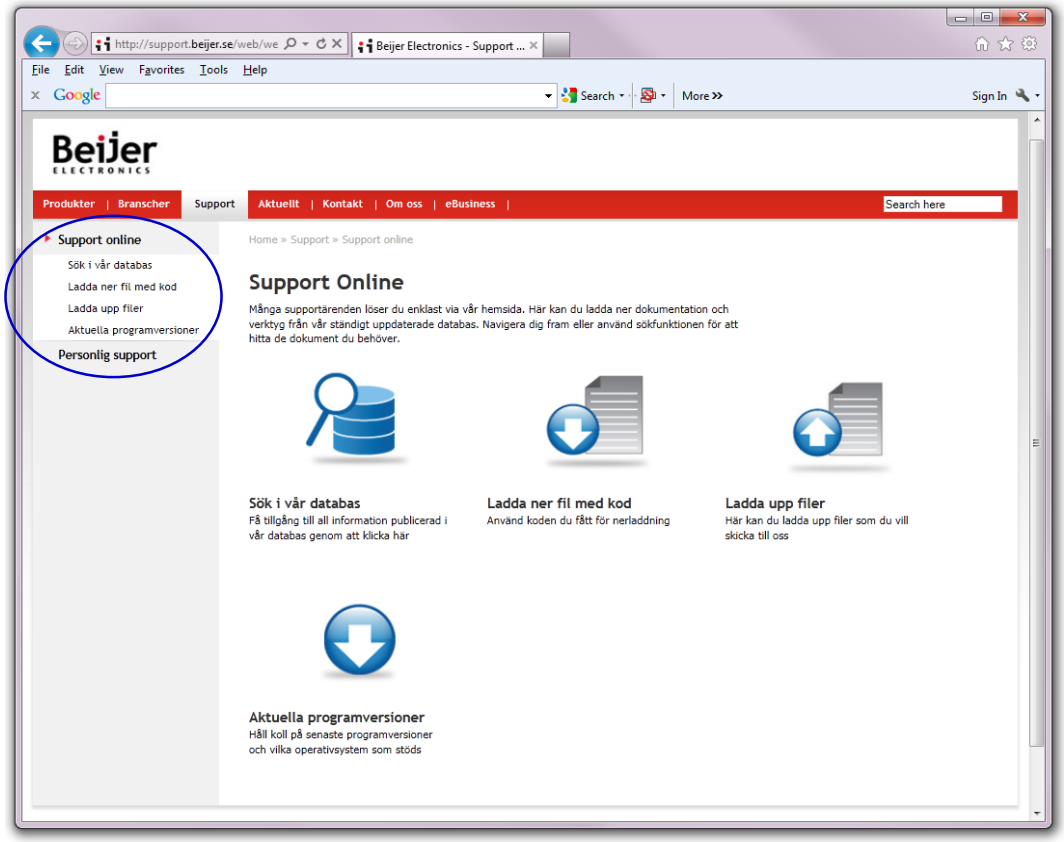
http://www.beijer.se/web/web_se_be_se.nsf/docsbycodename/filearchive?OpenDocument&mylink=/web/BexFilePileAUT.nsf/fm.be.searchframe?openform&Lang=SE&DocID=94B54BC3B26E94F5C1257AC4005C763C

- By using the function "Load and Write Recipe" the backup can be restored to the CoDeSys device by accessing the special text-file (for example ValueBackup1.txtrecipe), and it can be edited with a normal text editor too.

- Attached example project (Recipe_Backup.zip) including:
  - RecepieManagerExample.projectarchive (CoDeSys project of T4A SoftControl)
  - ValueBackup1.txtrecipe (Example of backup text file)
  - iX_T4A_SC_RecValues (iX Developer project of T4A SoftControl)

Web Site

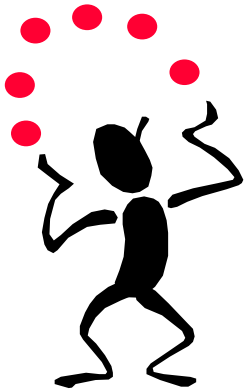
- http://support.beijer.se



- **Business Area Automation**
  www.beijer.se
  - Product
  - Branches
  - Support
  - Contact us
  - About us
  - eBusiness

- **Support Online**
  support.beijer.se
  - Download Knowledges
  - Program Examples (Function blocks)
  - Startup guidelines
  - User's Manuals, Configuration files
  - Cable guides and Drawings
  - Current software version
  - File transfer

- **Beijer Group**
  www.beijergroup.com

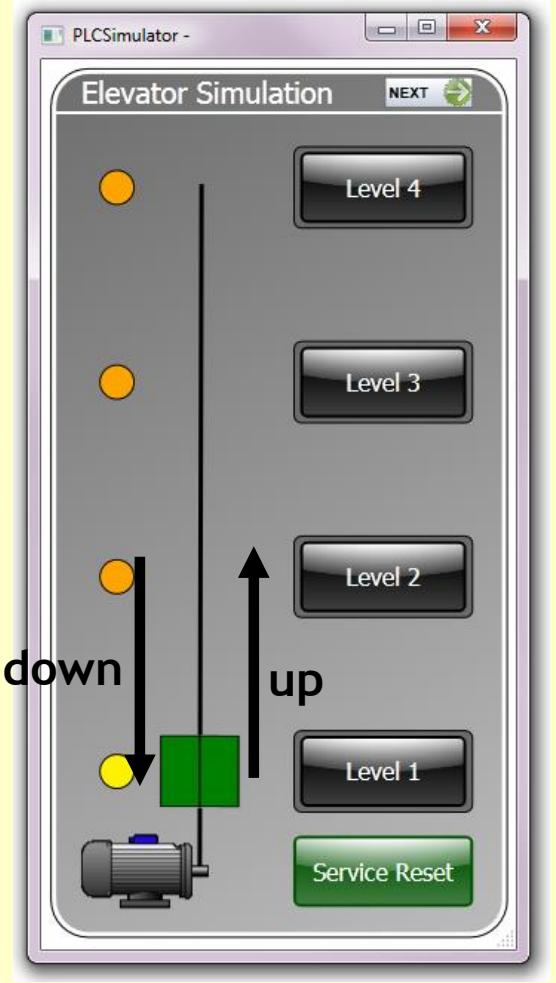


CoDeSys V3

Additional Exercises

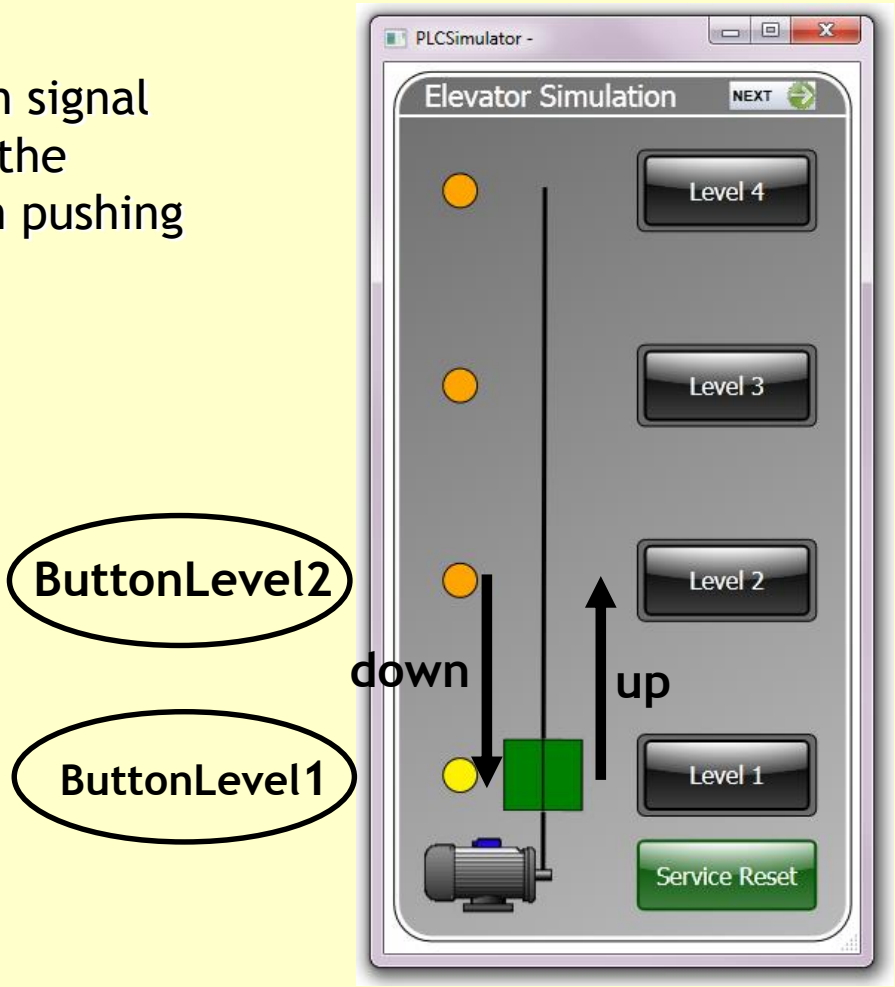
Additional Exercise, E1

- Modify the program
  - Make sure to control the motor of the elevator properly so it cannot run up and down at the same time!
  - For example ElevatorUp should not be started when ElevatorDown is active and vice-versa



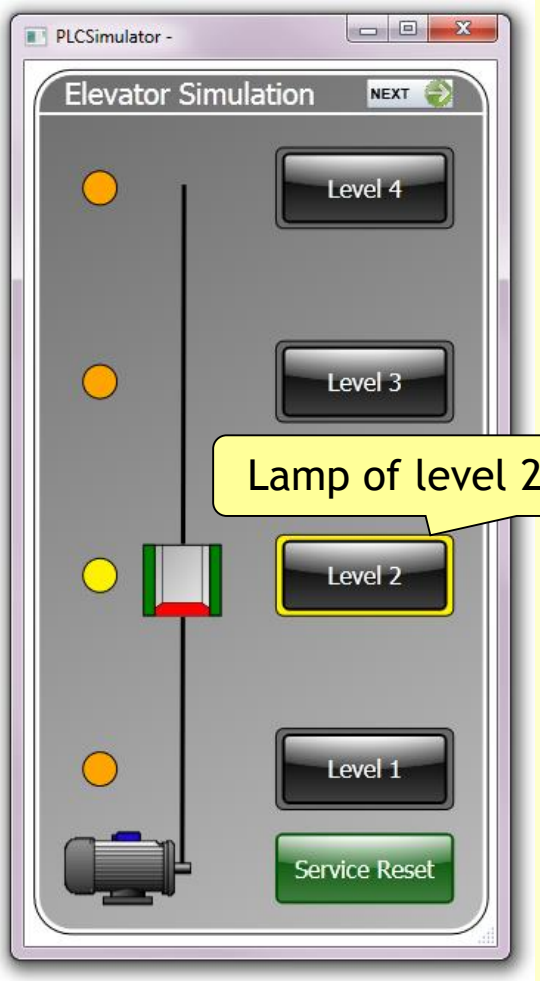
Additional Exercise, E2

- Interlocking
  - Add interlocks so that one boolean signal “Manual/Auto” must be TRUE for the elevator to run up and down when pushing the buttons on the elevator



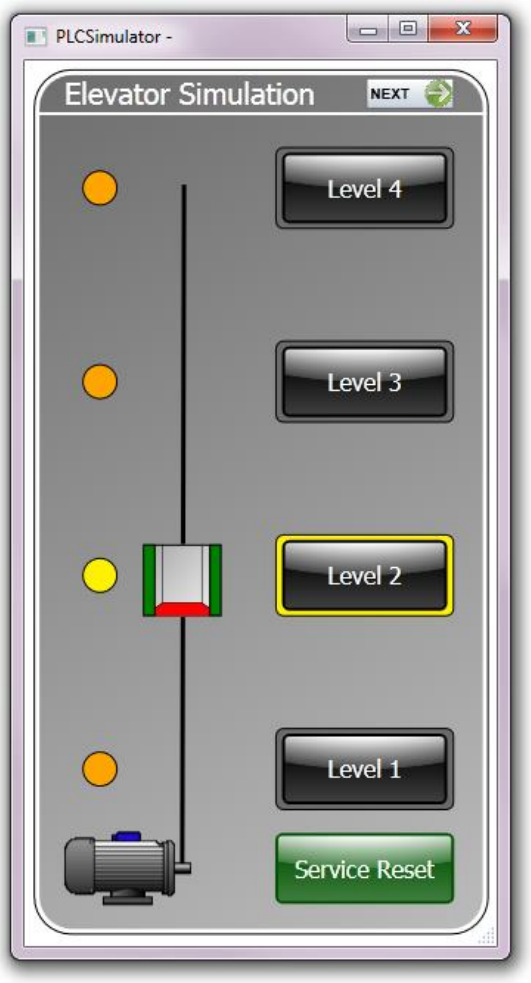
Additional Exercise, E3

- Lamps on each floor
  - Complete the program so that the light of the respective push-button is lit as long as the button is pressed
  - Use outputsignal LampLevel1 to 4



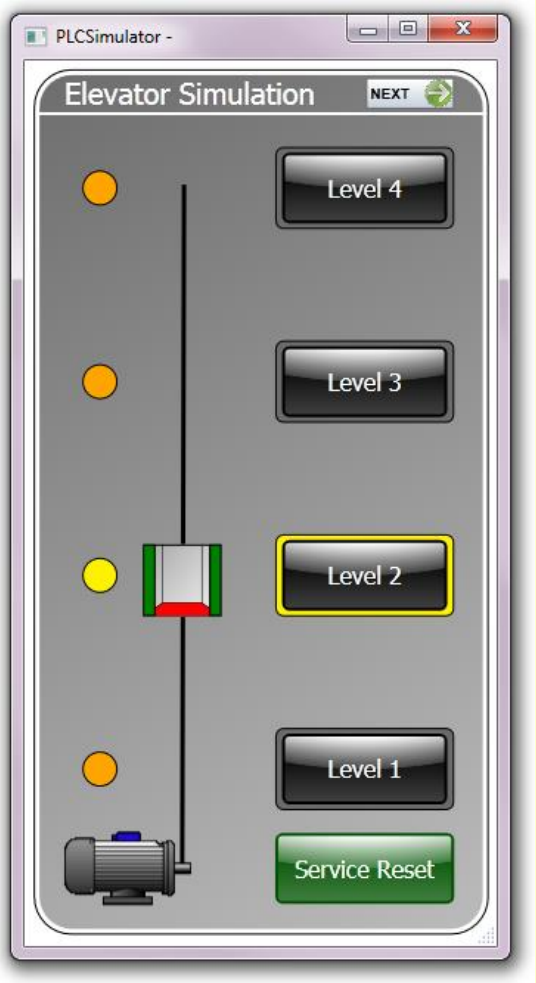
Additional Exercise, E4

- Start Delay
  - Complete with delay so that the level buttons must be pressed at least one second (T#1s) before the elevator goes up or down



Additional Exercise, E5

- Flashing function
  - Complete the program with a flashing function
  - Make the lamp blink at the floor to which the elevator is arriving
  - When the elevator arrives, make the lamp shine steadily
  - Outputsignal LampLevel1 to 4



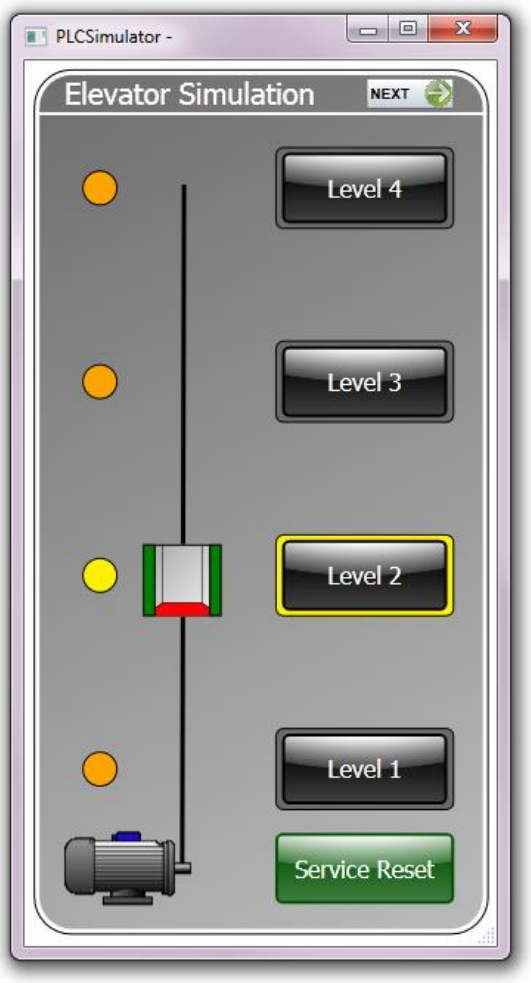
Additional Exercise, E6

- Remanent Variables

  - As a test, declare Remanent Variables (RETAIN and PERSISTENT) of some of the global variables of type INT

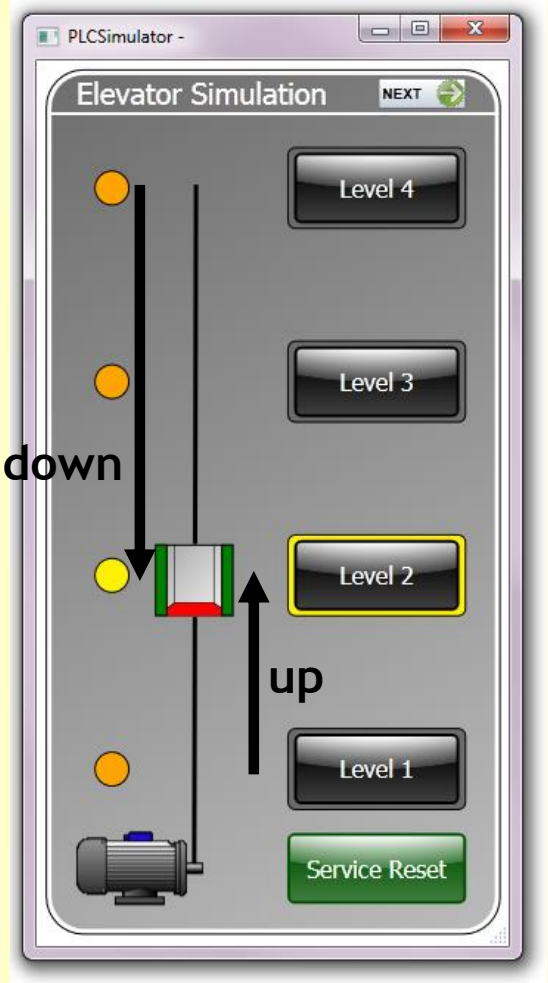
  - The difference is that Remanent Variables maintain their status even during power failure of the PLC

  - Login and define values to these Remanent Variables as a practical test, then use menu selection Online – Reset warm and check status



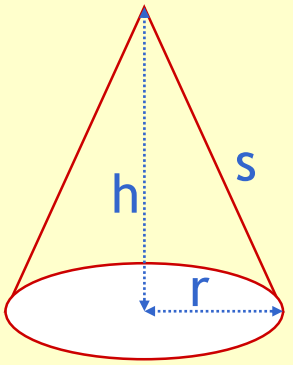
Additional Exercise, E7

- Automatic return to 2nd floor
  - Complete the program so that the elevator returns to the 2nd floor, from the 1st, 3rd or 4th floor after 10 seconds



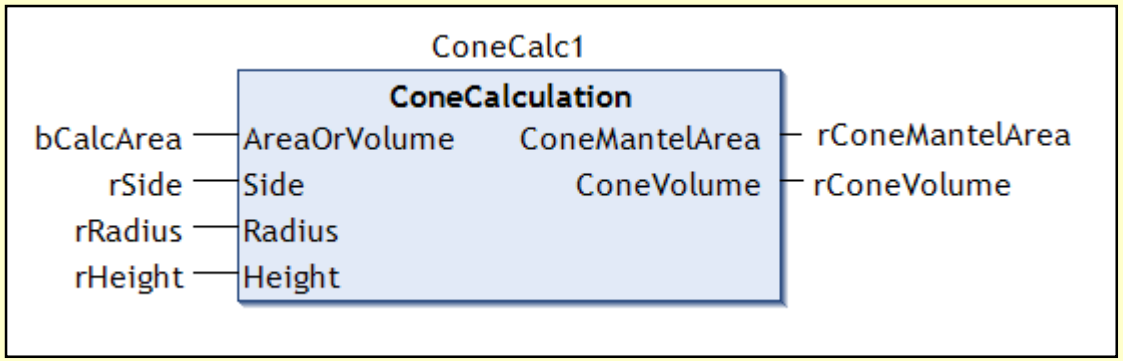
Exercise, Function Block, E8

- Create the following Function Block, 'ConeCalculation'
  - Input and output signals type: REAL
  - Use ST-editor, makes it more easy with the formulas
- If the boolean input 'AreaOrVolume' is true the mantle area 'ConeMantelArea' is calculated, otherwise the volume 'ConeVolume' is calculated.



- Formula: **Mantel Area = pi * radius * side**
- Formula: **Volume = 1/3 * pi * radius ² * height**

- Tip, declaring "pi" as a variable constant 3.1415



Exercise, Function, E9

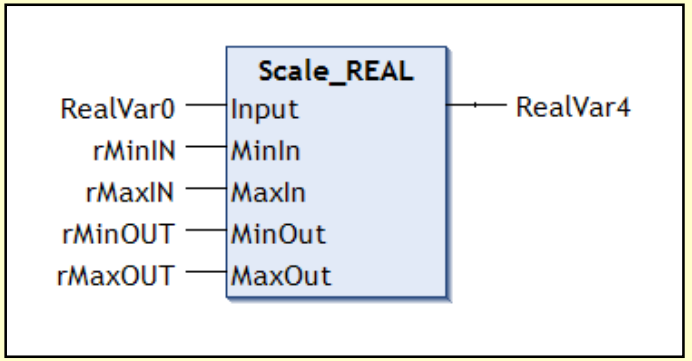
- Copy the more simple Scale block and make a function with the following features:

  - Result type: REAL

  - Editor: Structured Text (ST)

  - The block scales the input to a REAL value from MinOut to MaxOut

  - The input signal is expected to be between MinIn and MaxIn analog input resolution



- Use the block to scale the analog input signal to a value between 0.0 and 1000.0

- FORMULA: Output = Gain * Input + Offset
  Gain = (MaxOut-MinOut) / (MaxIn-MinIn)
  Offset = (MinOut – Gain * MinIn)

Beijer ELECTRONICS


Beijer
ELECTRONICS
easy when you know how