

Codesys

Paolo Burgio

paolo.burgio@unimore.it



UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

High Performance
Real Time **Lab**



Load the main program interface

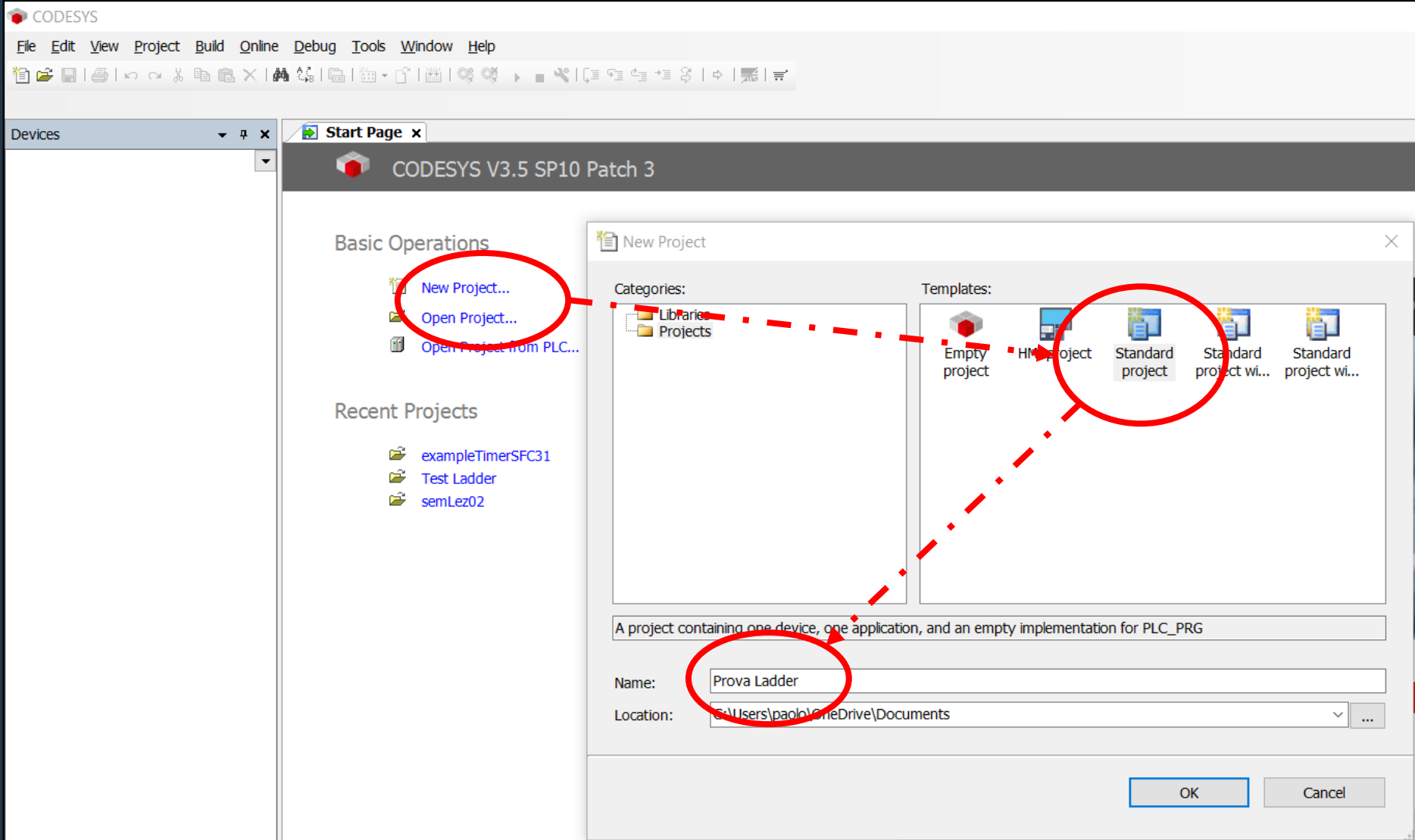
What is it?

- › An IDE to create PLC programs, and **simulate them**
- › In any of the five main languages
- › I use V3.5 SP1 patch 3, recommended version (for compatibility with the examples I'll give you)



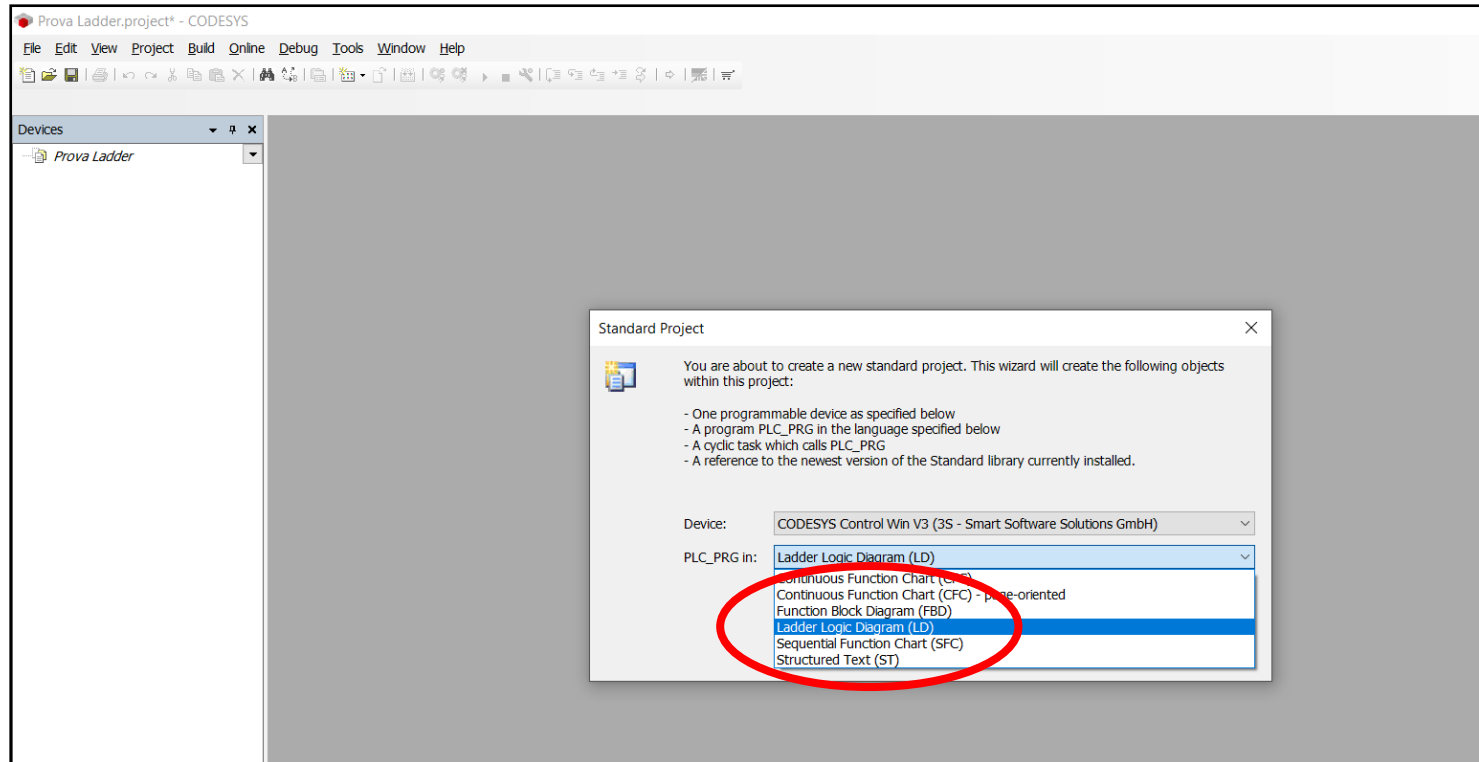


Create a project





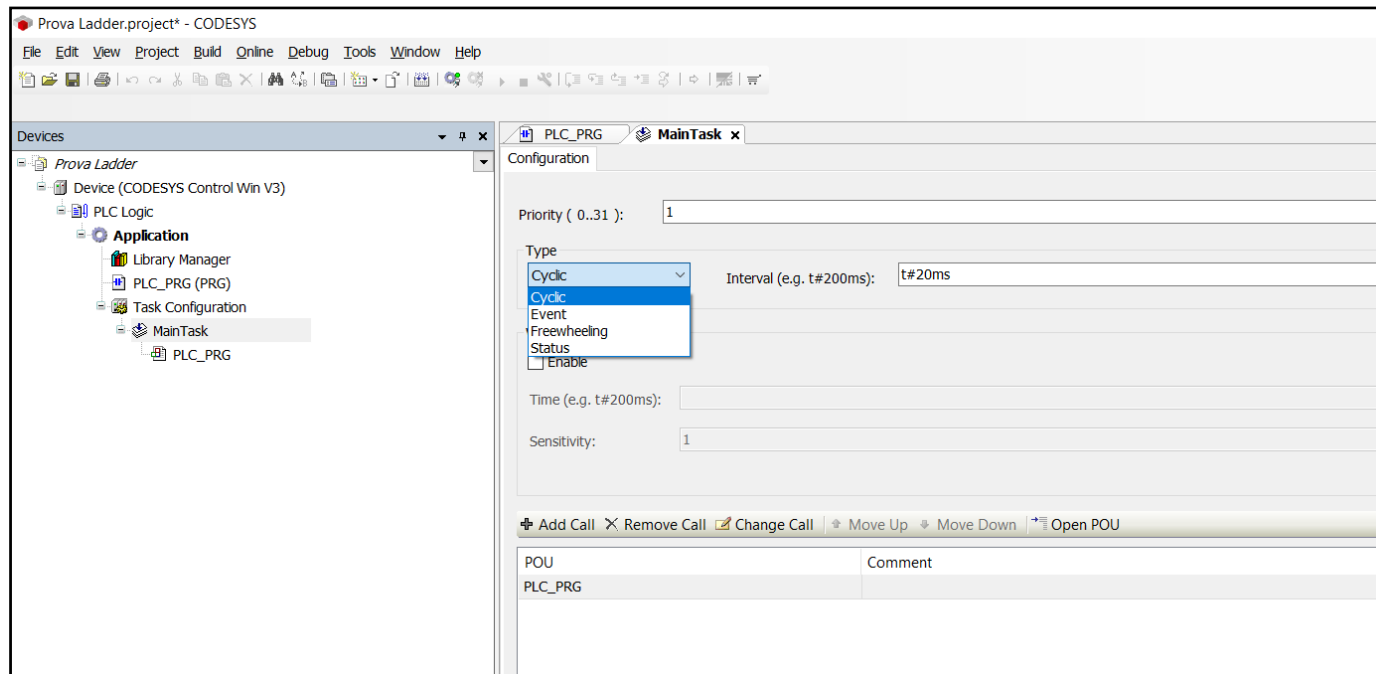
Select the language





Project workbench

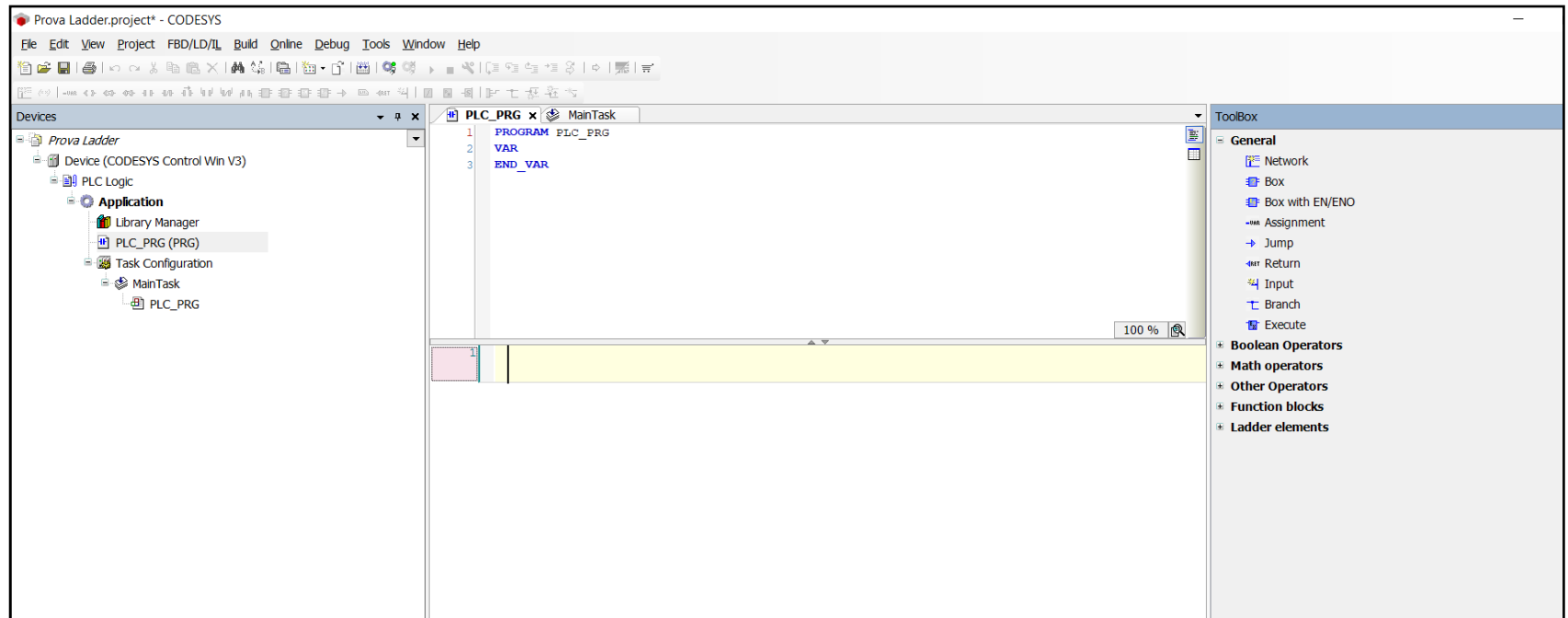
- › Your application has a Main task, that (here) runs cyclically





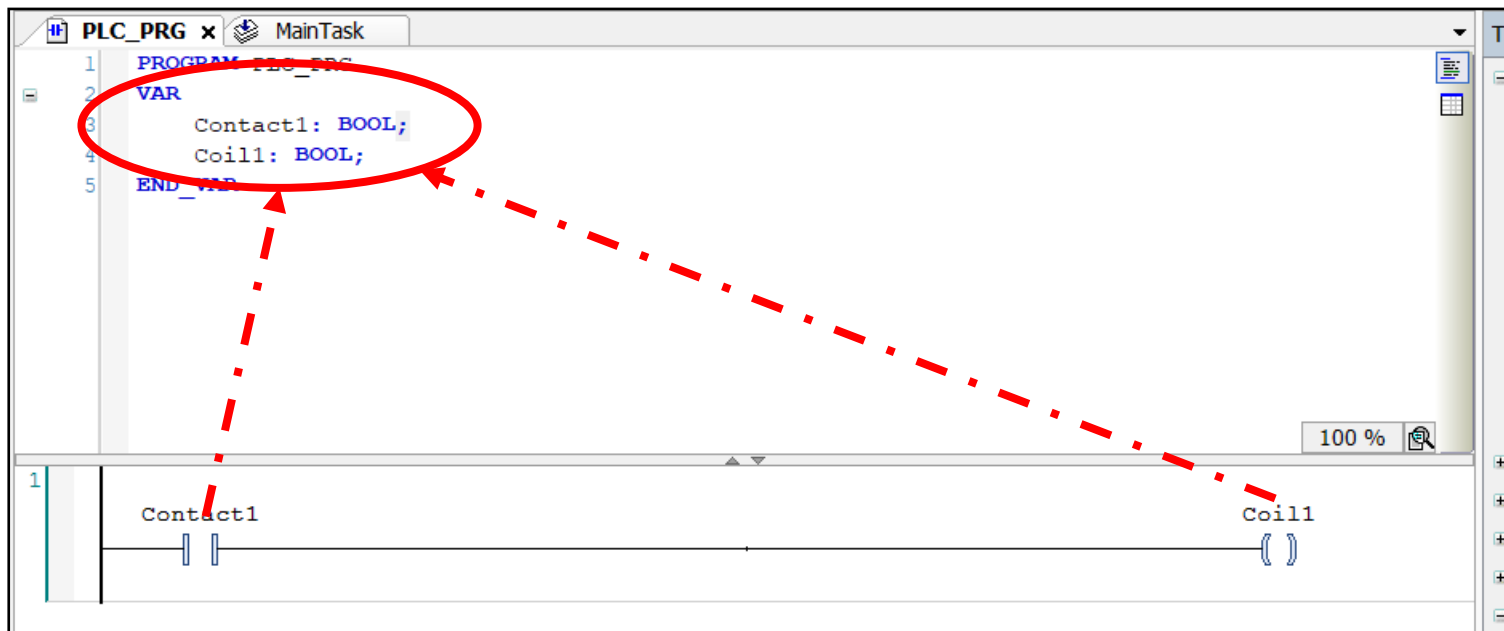
Project workbench - Ladder

- › You can create Ladder diagrams using drag/drop from the toolbox



Adding a contact + coil

- › Two global variables are automatically created in the variable definition window always in ST lang), both of `bool` type, as specified by us
- › Here, we want a switch that turns on a lamp, hence we need a NO contact and a coil
- › PS here you don't see the right power rail as it's implicit

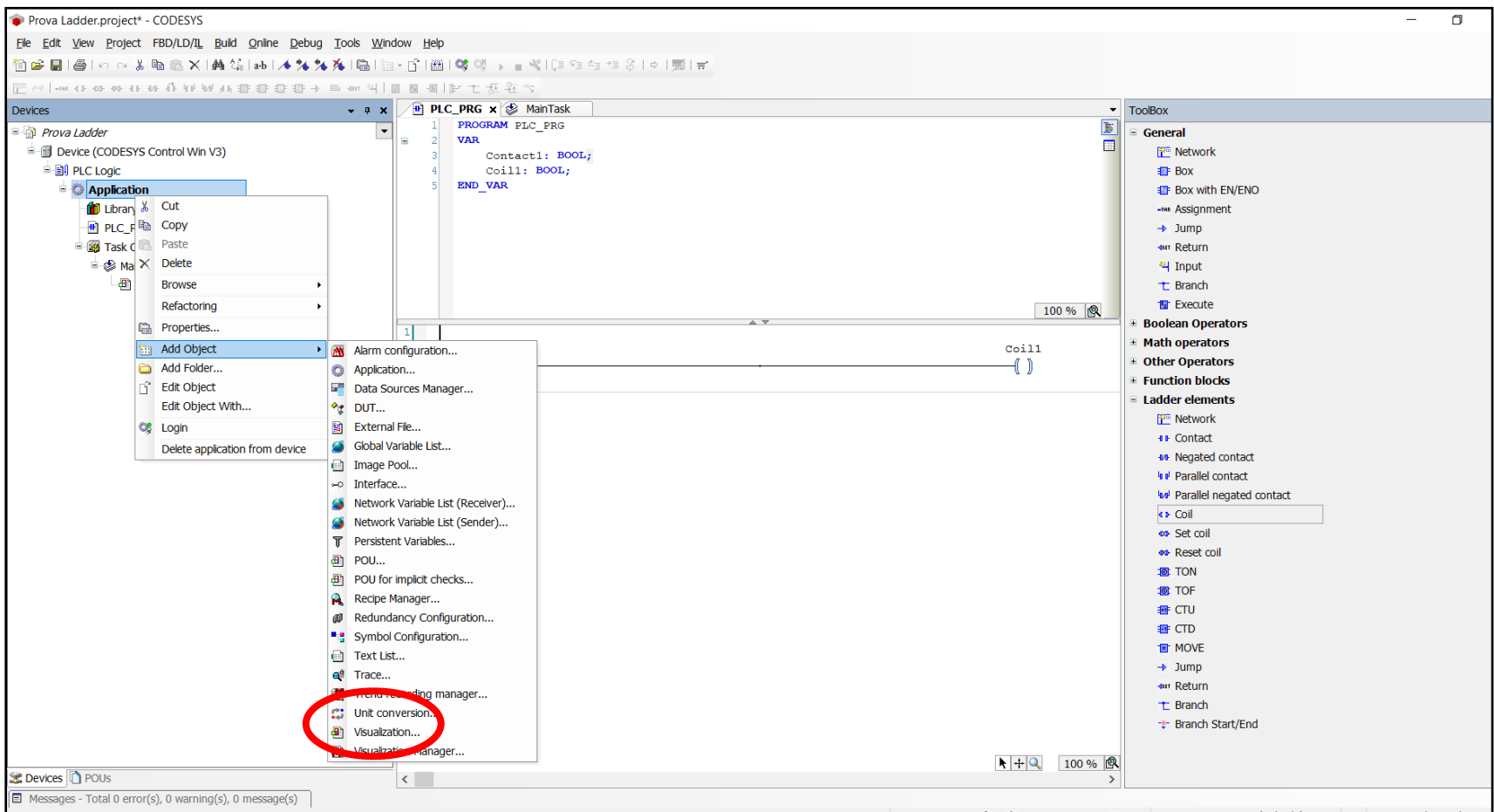




View the simulated system

Add a Visualization object

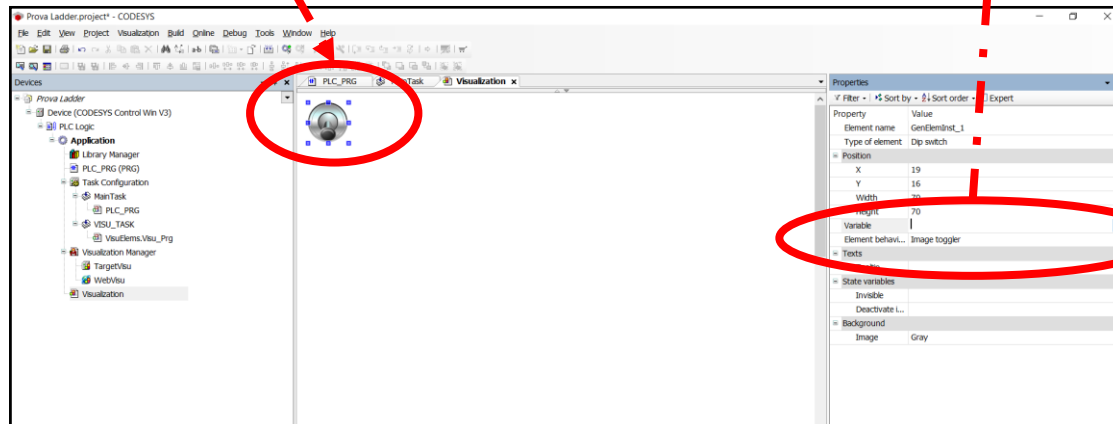
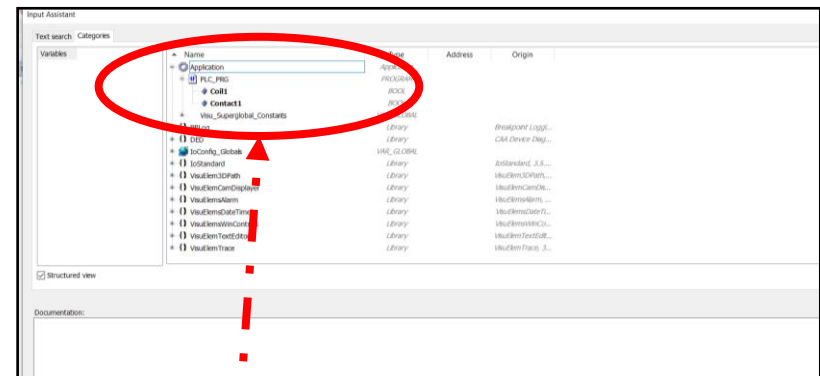
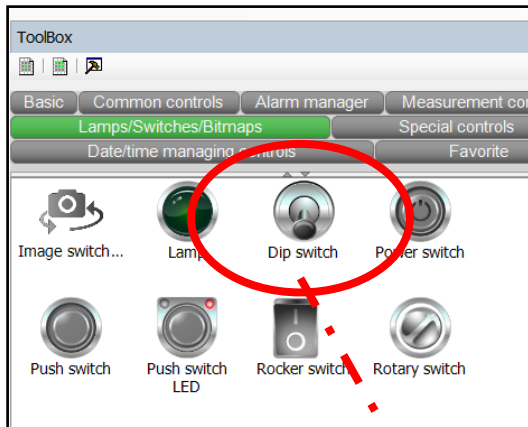
› Application -> Add Object -> Visualization





Add elements, and link to variables

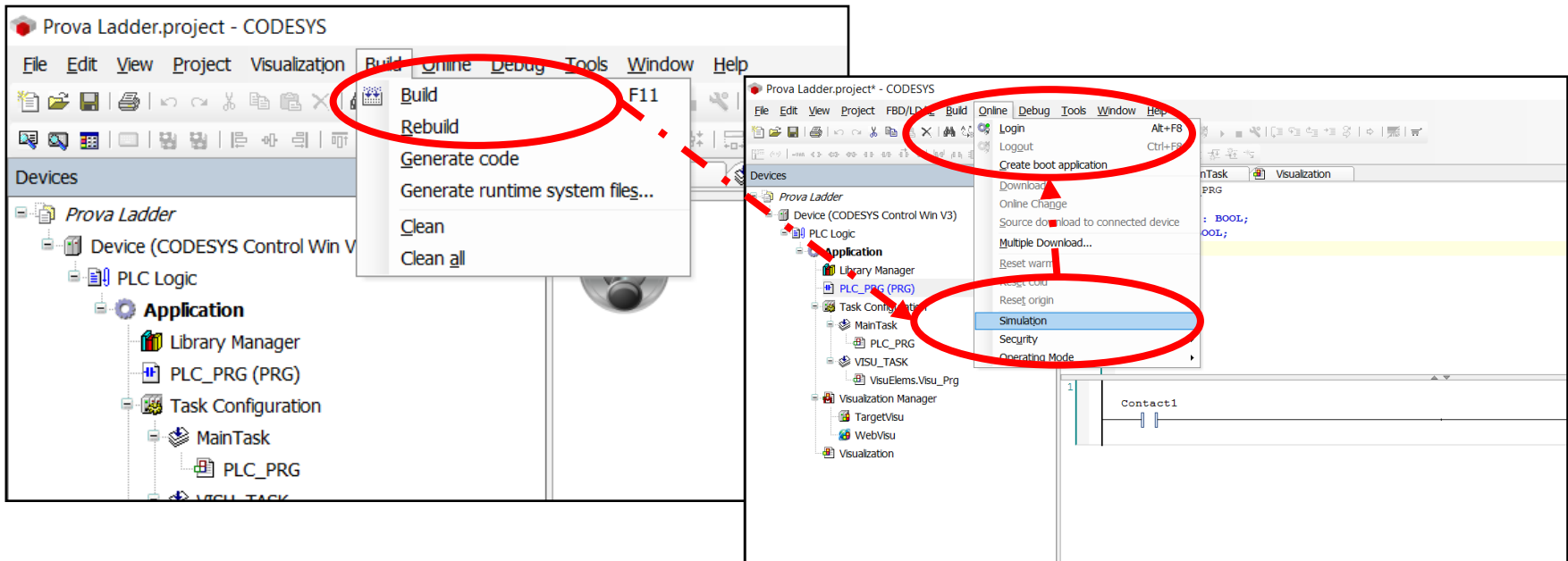
- › Here, we added a dip switch from the toolbox, and we select the `Contact1` var from the Properties window
- › Now, add a lamp and bind it to `Coil1`





Compile and set up simulator

- › Build the system, from the menu or with F11
- › Login from the Online menu to download the required run libs
 - Before..make sure you ticked “Simulation”!
- › Now, we’re ready to go





Run workbench

- › After a while, simulator/simulation is set up
- › Click on Debug -> Start to go
- › Nothing happens

The screenshot shows the CODESYS Run Workbench interface for a project named "Prova Ladder.project* - CODESYS". The interface includes a menu bar (File, Edit, View, Project, Visualization, Build, Online, Debug, Tools, Window, Help) and a toolbar with various icons for file operations, simulation, and debugging.

The left sidebar displays the project tree under "Prova Ladder". The "Device [connected] (CODESYS Control Win V3)" is expanded, showing "PLC Logic" and "Application [stop]". The "Application [stop]" is further expanded, showing "Library Manager", "PLC_PRG (PRG)", "Task Configuration", "MainTask", "PLC_PRG", "VISU_TASK", "VisuElems.Visu_Prg", "Visualization Manager", "TargetVisu", "WebVisu", and "Visualization".

The main workspace shows the "PLC_PRG" ladder logic program. The "Device.Application.PLC_PRG" is selected, and the "Expression" table is visible:

Expression	Type	Value	Pre
Contact1	BOOL	FALSE	
Coil1	BOOL	FALSE	

The ladder logic diagram shows a single rungs with "Contact1" in series with "Coil1". The "RET" instruction is at the end of the rung.

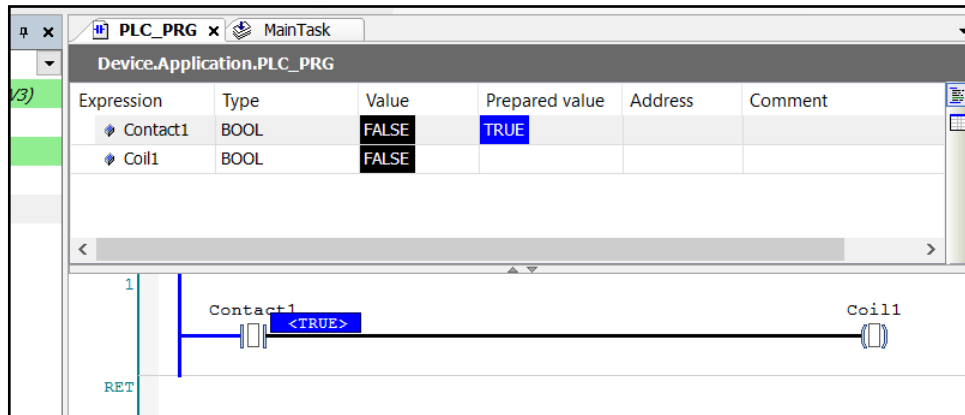
The right sidebar shows the "Visualization" window, which displays the message: "The online visualization is waiting for a connection. Please start the application."





Modify values

- › Via the “watch expression” window, use the “Prepared value”
- › Then, apply the value with the Debug -> Write value menu item (or CTRL+F7)



- › In this case, in our example, we can also manually acting on the switch

Remember to log out after you're done! 😊



Sequential contacts vs. parallel contacts

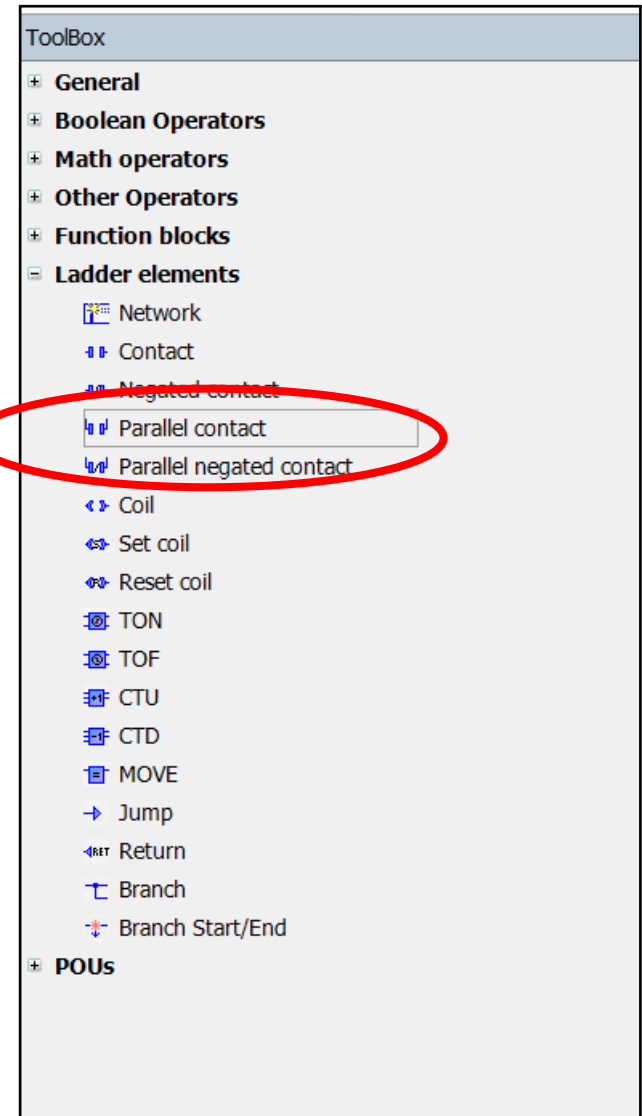
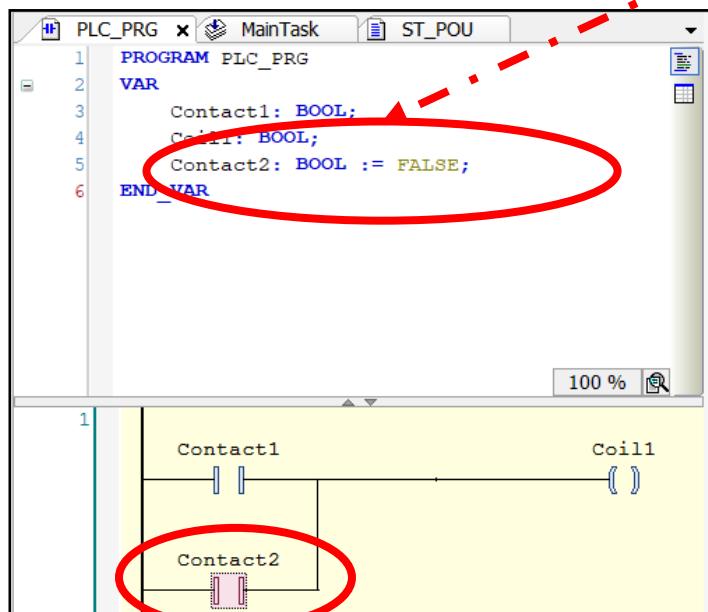
Logical “AND”


- › ..easy, simply drag&drop

Logical “OR”


- › “Parallel contact” components from toolbox
- › IDE helps us to insert it...

PS good programmers remember to initialize vars ;)





Structured Text



Add new ST POU

- › Program Organization Unit let you add logics in the same application, using different languages
- › We now add a **Program POU**

IEC 61131 does not allow spaces in names



Write the ST code

Prova Ladder.project* - CODESYS

File Edit View Project Build Online Debug Tools Window Help

Devices

- Prova Ladder
 - Device (CODESYS Control Win V3)
 - PLC Logic
 - Application
 - Library Manager
 - PLC_PRG (PRG)
 - ST_POU (PRG)
 - Task Configuration
 - MainTask
 - PLC_PRG
 - ST_POU
 - VISU_TASK
 - VisuElems.Visu_Prg
 - Visualization Manager
 - TargetVisu
 - WebVisu
 - Visualization

PLC_PRG MainTask ST_POU x

```
1 PROGRAM ST_POU
2 VAR
3     Contact1: BOOL;
4     Coil1: BOOL;
5 END_VAR
6
```

100 %

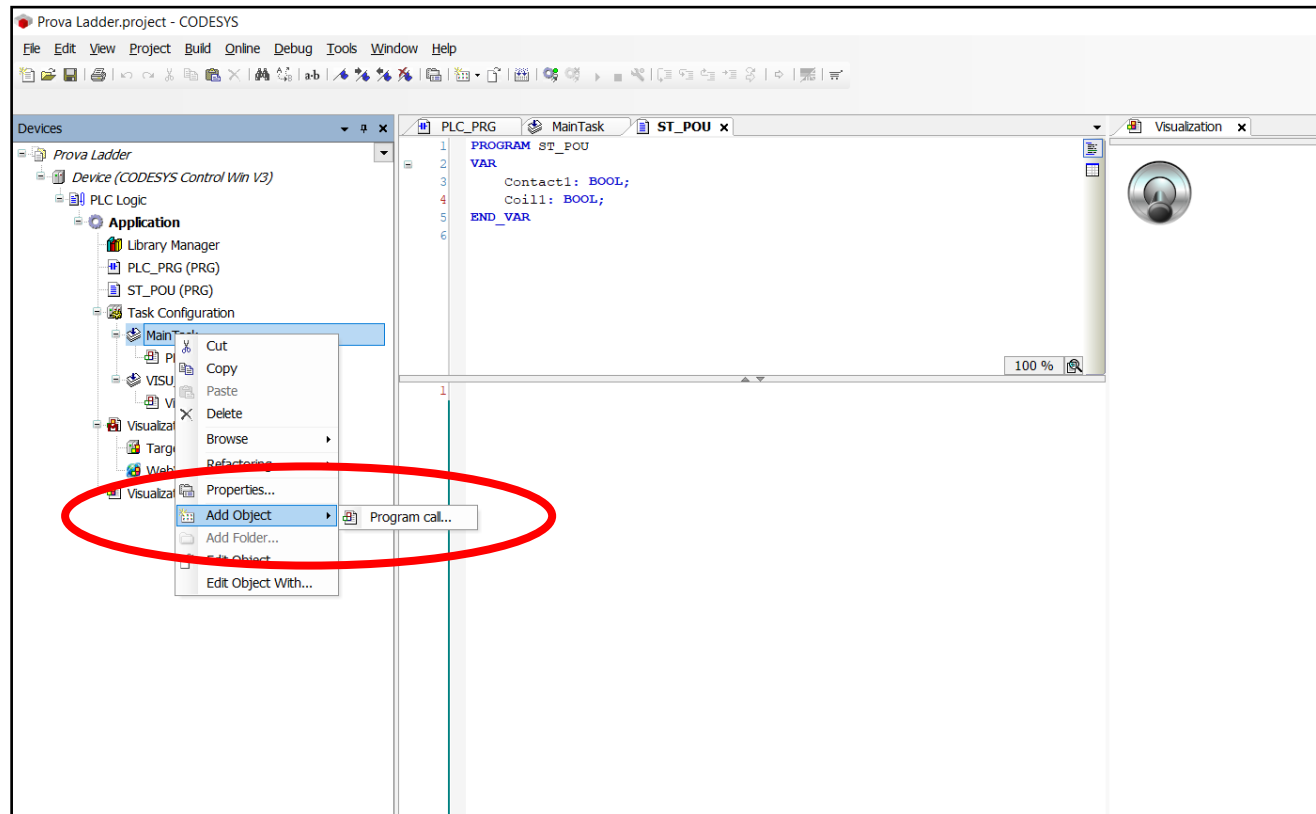
```
1 IF contact1 = TRUE THEN;
2     Coil1:=TRUE;
3 ELSE;
4     Coil1:=FALSE;
5 END_IF;
```

Visualization x



Are we done? Not yet...

- › We created a POU Program, but we haven't called it yet from within the MainTask...





Run and set values

- › If you set `Contact1` to `TRUE`, then `Coil1` goes to `TRUE`
- › ..but the simulated Light & Switch don't turn on!

Why?

- › Because they are **not** attached to **those** `Contact1` and `Coil1` vars you think...
- › Look out when you write names...

Should we attach those vars to the two simulated objects?

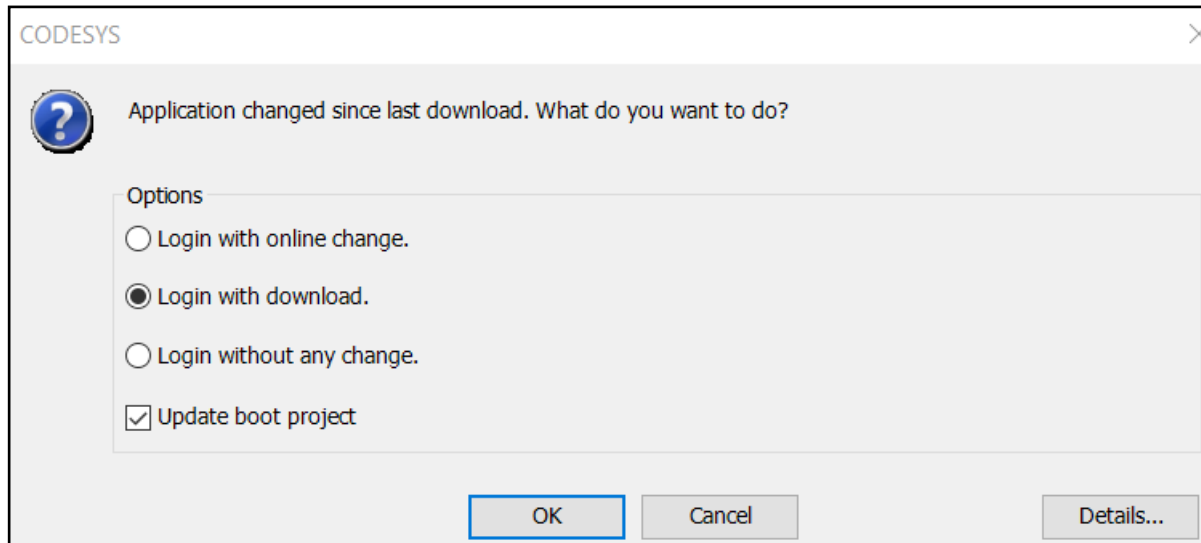
- › (recommendation) Only if requested by the application specs
- › In this case, I use them for debugging/teaching purposes, so my specs say "no" 😊



Compile & Login again

We added a ST block, so the simulation engine might require some components

› Codesys will prompt us

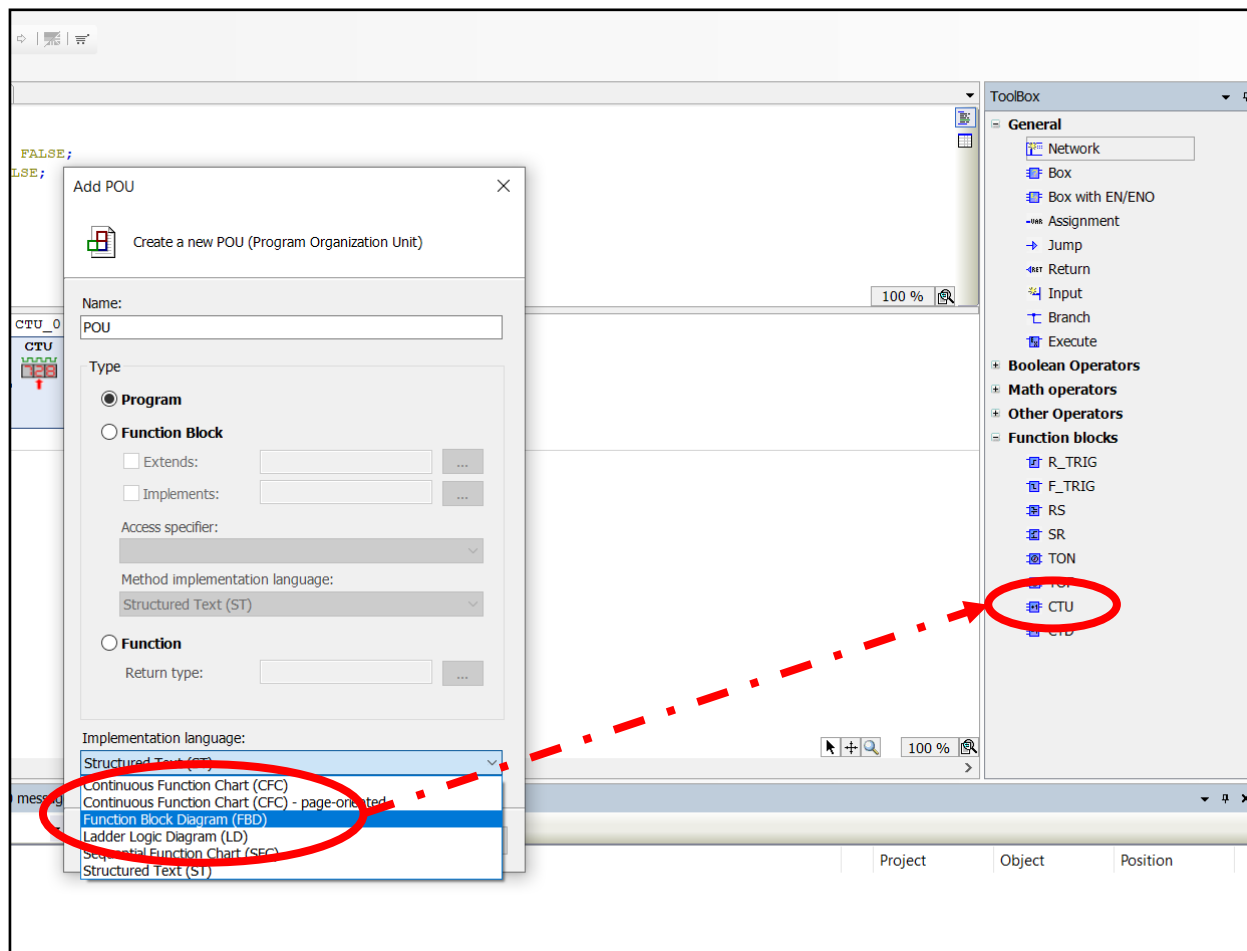




Programming with Function Blocks

Counters - CTU

- › Create a program, or add a POU of type “FBD”
- › Then, drag a Counter (CTU) in the workbench

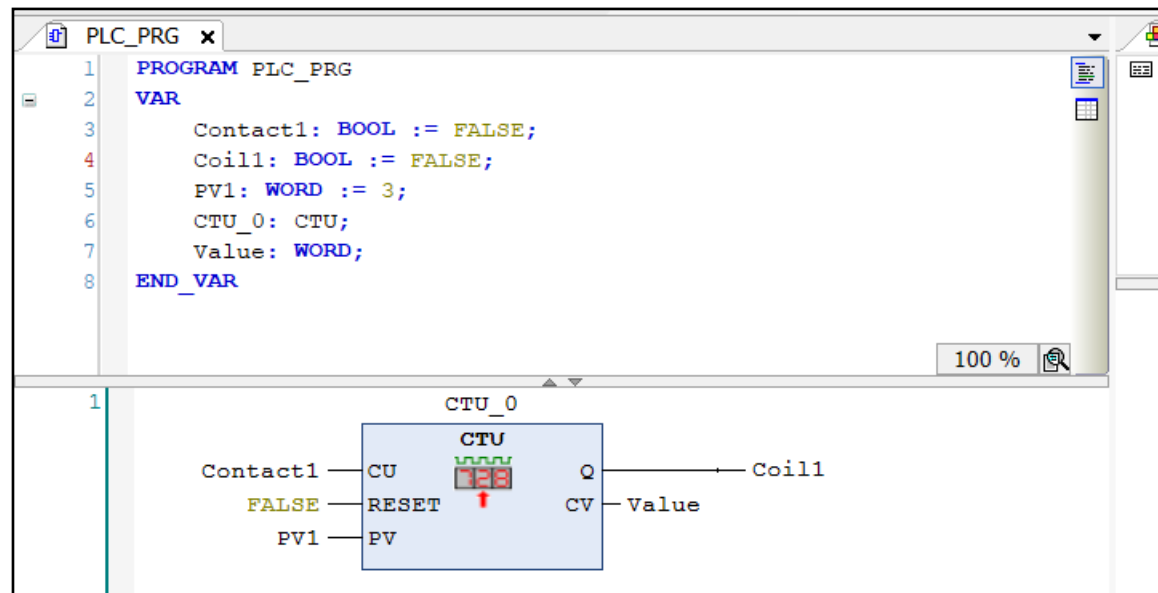




Bind the CTU

Connect CTU in&outs to vars

- › CU to a contact (press&hold button)
- › Here, RESET is false (just for this example)..might use a button/contact?
- › PV1 is a WORD
- › Q to a coil (we want to turn on a lamp)
- › CV to a WORD variable, to monitor the status





Add visualization

- › Attach a “simple” press&hold button to `Contact1`
- › Add a lamp and attach to `Coil1`
- › ..or, can also attach it directly to CTU out `PLC_PRG.CTU_0.Q`

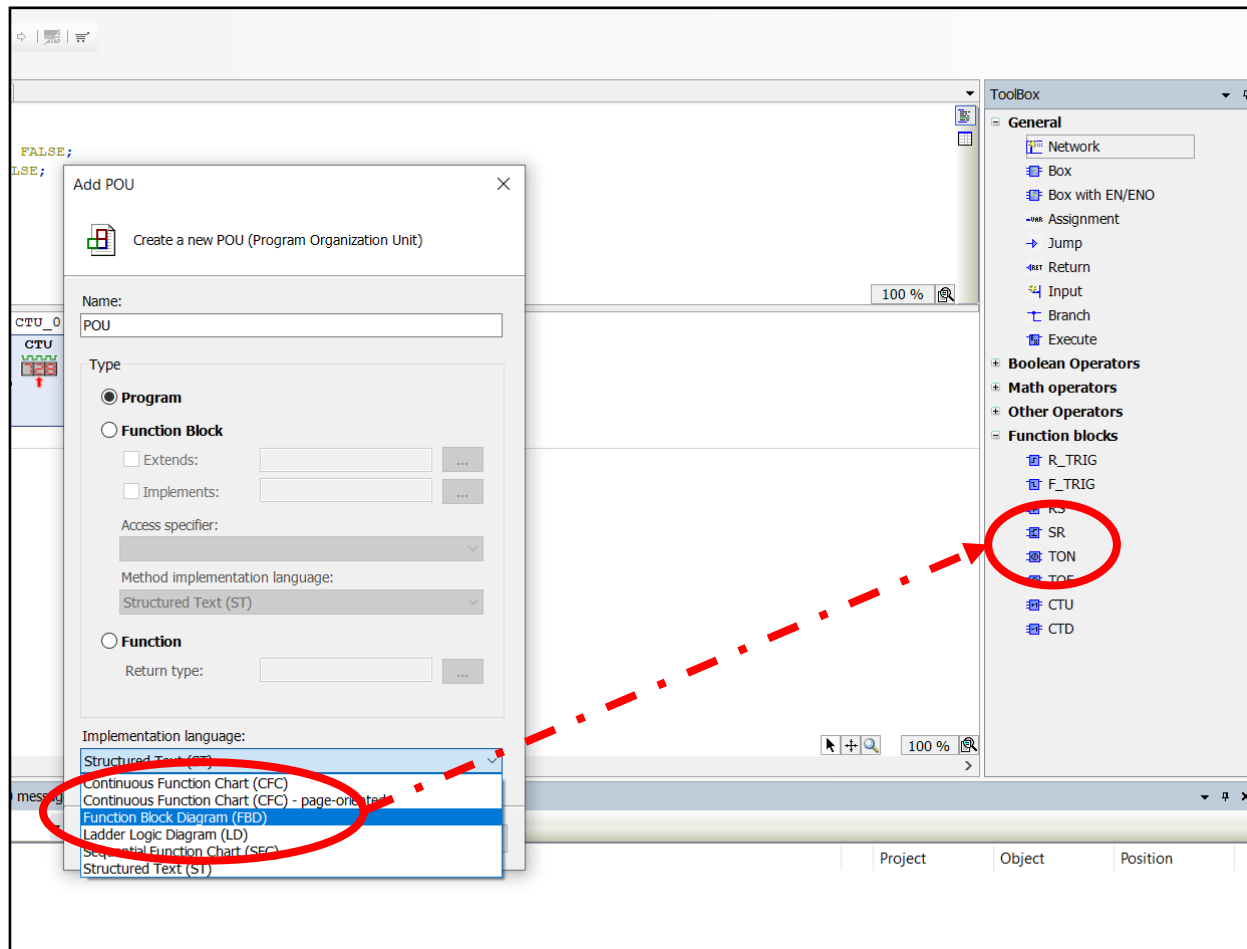
The screenshot displays the Siemens SIMATIC Manager interface with three main windows:

- PLC_PRG (Ladder Logic):** Shows a program with variables `Contact1`, `Coil1`, `PV1`, `CTU_0`, and `Value`. The ladder logic diagram shows `Contact1` connected to the `CU` (Current Underflow) input of `CTU_0`, `PV1` to the `PV` (Present Value) input, and the `Q` (Output) of `CTU_0` connected to `Coil1`. The `RESET` input of `CTU_0` is set to `FALSE`.
- Visualization:** Shows the `Interface Editor` with a `Press me` button and a lamp icon. The lamp icon is circled in red, and a red arrow points from it to the `Variable` property in the Properties window.
- Properties:** Shows the properties of the selected element (Lamp). The `Variable` property is set to `PLC_PRG.CTU_0.Q`, which is also circled in red.

Property	Value
Element name	GenElemInst_1
Type of element	Lamp
Position	
X	295
Y	18
Width	70
Height	70
Variable	PLC_PRG.CTU_0.Q
Texts	
Tooltip	
State variables	
Visible	

Timers – TON

- › Create a program, or add a POU of type “FBD”
- › Then, drag a Timer On (TON) in the workbench





Bind the TON

- › IN to a contact (press&hold/standard button)
- › Remember, IN starts timer @its rising edge, and resets @its falling edge
- › PT1 is a TIME
- › Q to a coil (we want to turn on a lamp) using the **assignment** operator
- › ET to a TIME variable, to monitor the status

The screenshot displays a PLC programming environment with the following components:

- PLC_PRG x**: A text editor window showing the following code:

```
1 PROGRAM PLC_PRG
2 VAR
3   Contact1: BOOL := FALSE;
4   Coill1: BOOL := FALSE;
5   PT1: TIME;
6   Value: TIME;
7   TON_0: TON;
8 END_VAR
```
- Visualization x**: A window containing an **Interface Editor** and an **Elementlist**. The **Interface Editor** shows a variable declaration:

```
1 VAR_IN_OUT
2
3 END_VAR
```
- ToolBox**: A sidebar on the right with categories like **General**, **Boolean Operators**, **Math operators**, **Other Operators**, and **Function blocks**. The **General** category is expanded, and the **Box with EN/ENO** and **Assignment** items are circled in red.
- Ladder Logic Diagram**: A diagram at the bottom showing a **TON** (Timer On Delay) block. The **IN** input is connected to **Contact1**, the **PT** (Preset Time) input is connected to **PT1**, and the **Q** (Output) is connected to a coil. The **VALUE** output is also shown.
- Visualization**: A graphical representation of the ladder logic diagram, showing a button labeled "Press me" and a lamp. A red dashed arrow points from the "Press me" button to the **IN** input of the **TON** block in the ladder logic diagram.



Defining Function Blocks

Create a new POU of type Function Blocks

› (of course, in ST)

Add POU

Create a new POU (Program Organization Unit)

Name:
MyFB

Type

☐ Program

☒ **Function Block**

☐ Extends: ...

☐ Implements: ...

Access specifier:
Public

Method implementation language:
Structured Text (ST)

☐ Function

Return type: ...

Implementation language:
Structured Text (ST)

Add Cancel



Implement the Function Block

Add in/out vars

- › num1, num2 as ins, SumResult and SubResult as outs, all REAL numbers

Add FB logics

- › In ST workbench

```
1 FUNCTION_BLOCK MyFB
2 VAR_INPUT
3     num1: REAL;
4     num2: REAL;
5 END_VAR
6 VAR_OUTPUT
7     AddResult: REAL;
8     SubResult: REAL;
9 END_VAR
10 VAR
11 END_VAR
12
```



```
1 AddResult := num1 + num2;
2 SubResult := num1 - num2;
```



Use it in the Application

In the main POU, create vars

- › A and B are assigned, respectively, 11 and 5
- › Also, instantiate the FB

The screenshot shows a software interface with a project tree on the left and a code editor on the right. The project tree includes 'SYS Control Win', 'ion', 'y Manager', '(FB)', 'PRG (PRG)', 'Configuration', 'ainTask', and 'PLC_PRG'. The code editor has two tabs: 'MyFB' and 'PLC_PRG x'. The 'PLC_PRG x' tab is active, displaying the following code:

```
1 PROGRAM PLC_PRG
2 VAR
3     A: REAL := 11;
4     B: REAL := 5;
5     Sum: REAL;
6     Subtr: REAL;
7     SumAndSubtract: MyFB;
8 END_VAR
```

Below the code editor, there is a ladder logic editor showing a single rungs with a coil and a normally open contact, both labeled '1'.



Call the FB from application

Need to explicitly bind FB input vars to main POU the vars

- › E.g., A to num1
 - › Can use dot notation to fetch output values, after calling
- Slightly different than in “traditional” programming languages
- › Why?

RG)	8	END_VAR
ration		
RG	1	SumAndSubtract(num1 := A, num2 := B);
	2	
	3	Sum := SumAndSubtract.AddResult;
	4	Subtr := SumAndSubtract.SubResult;





Simulate..and enjoy! 😊

MyFB PLC_PRG x

Device.Application.PLC_PRG

Expression	Type	Value
A	REAL	11
B	REAL	5
Sum	REAL	16
Subtr	REAL	6
SumAndSubtract	MyFB	

```
1 SumAndSubtract (num1 11 := A 11, num2 5 := B 5);  
2  
3 Sum 16 := SumAndSubtract.AddResult 16;  
4 Subtr 6 := SumAndSubtract.SubResult 6; RETURN
```



Call from another POU

Now, your amazing FB can be used in another POU!

- › Instantiate a FBD POU
- › Find MyFB in the toolbox
- › Instantiate and try it!

The screenshot displays a PLC programming environment. The main window shows a POU (Program Organization Unit) named 'POU x' being edited. The code in the POU is as follows:

```
1 PROGRAM POU
2 VAR
3     SumAndSub: MyFB;
4 END_VAR
5
```

Below the code, a ladder logic network is shown. A red oval highlights a call to the function block 'MyFB' named 'SumAndSub'. The block has two inputs labeled 'num1' and 'num2', both with red wavy lines indicating they are not yet defined. It has two outputs labeled 'AddResult' and 'SubResult', both with red wavy lines.

The 'ToolBox' on the right side of the interface shows various PLC symbols. Under the 'POUs' section, 'PLC_PRG' and 'MyFB' are listed, with 'MyFB' highlighted by a red oval. A red dashed arrow points from the 'MyFB' entry in the toolbox to the 'MyFB' block in the network.



Exercise

Let's
code!

Implement any of the automatas that we saw so far using an FSM written using `ST CASE-SWITCH`

- › Base automata

*“Identify even sequences of a (even empty),
followed by one, or more, or no, b, ended by c”*

- › The traffic light

- › Whatever you want!

You might want to use Function blocks to separate and test different functionalities using different POU's



How to run the examples

Let's
code!

- › Find them in `Code/` folder from the course website

To download Codesys, ask the teacher, or open an issue in our GitHub page



References



Course website

- › http://hipert.unimore.it/people/paolob/pub/Industrial_Informatics/index.html

My contacts

- › paolo.burgio@unimore.it
- › <http://hipert.mat.unimore.it/people/paolob/>

Resources

- › Brian Hobby, Codesys tutorials (a must to learn the tool in 5 mins)
- › A small blog
 - www.google.com