
UiA
University of Agder
Norway

Lecture #2.3

Object-oriented PLC programming

MAS418

Programming for Intelligent Robotics and Industrial systems

Part II: PLC Software Development

Spring 2024

Daniel Hagen, PhD

UiA

Previous Lecture

Procedural oriented PLC programming

I. Data types cont. & Std. library

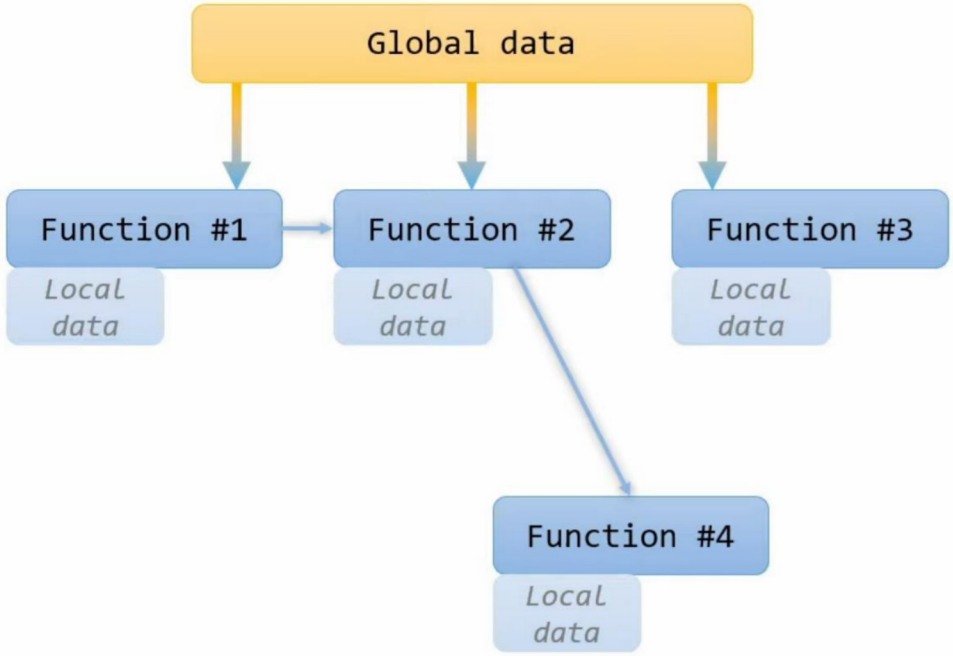
- Time
- Pointers
- Reference
- Arrays
- Enumeration
- Standard library

II. Structures & functions

- Introduction
- Structures
- Functions
- Pass by value & pass by reference

III. Instructions

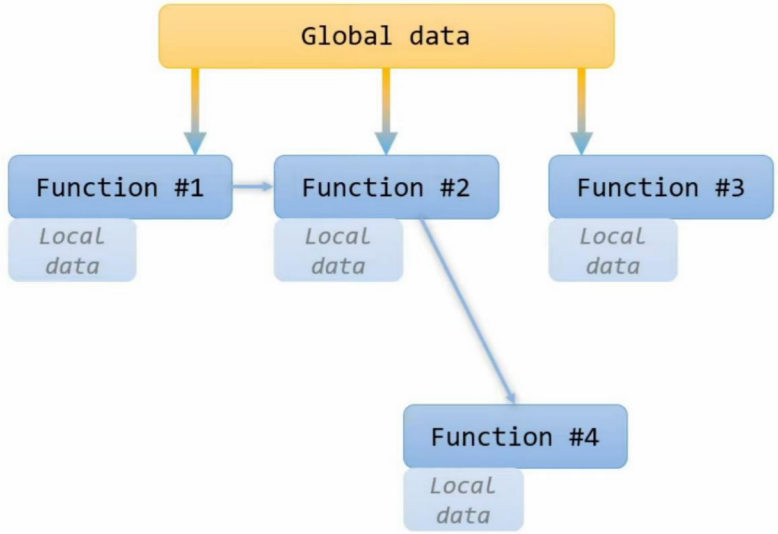
- Introduction
- Conditional statements
- CASE instruction
- FOR loops
- WHILE loops

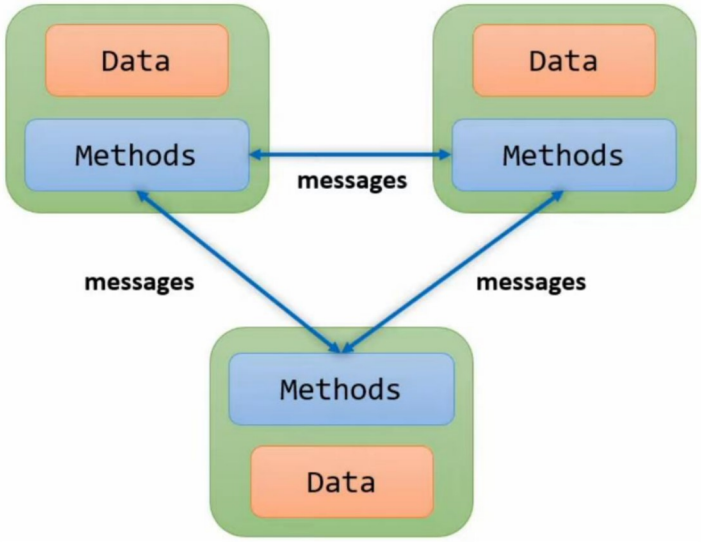


UiA

Object Oriented Programming (OOP)

- In **OOP** you have objects carrying their own set of data and their own attached methods
- The objects can communicate with each other by so called messages
- In **POP** importance is given to functions as well as sequence of actions to be done, while in **OOP** importance is given to data
- With **POP** the program is divided into functions, while it for **OOP** the program is divided into objects

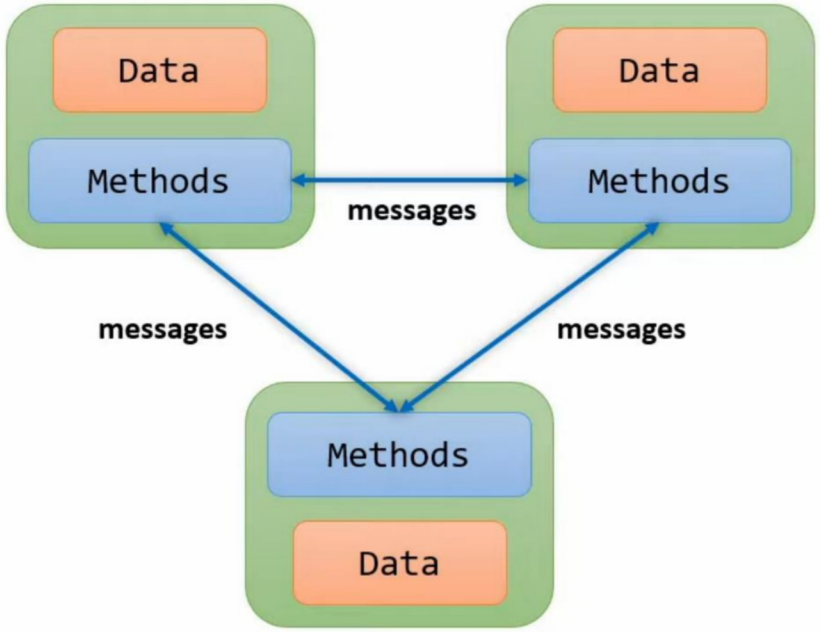




https://www.youtube.com/watch?v=JSFvSu8uB9U&list=PLimaF0nZKYHz3I3kFP4myaAYjmYk1SowO&index=4

Key takeaways

- **Procedural-oriented PLC programming**
  - **Function blocks**
  - **Methods**
  - **Inheritance**
  - **Interfaces**



Overview

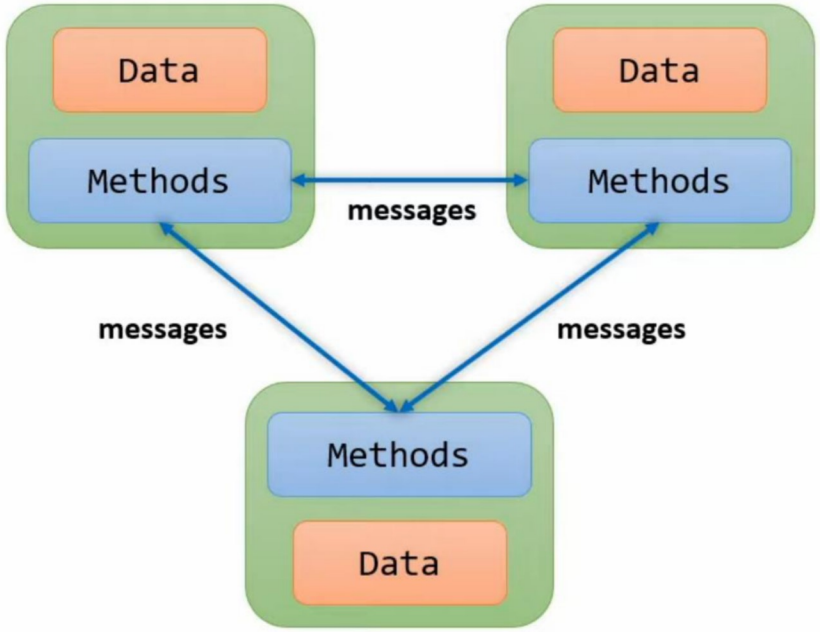
Introduction

Part I: Presentation of application

Part II: Function Blocks

Part III: Interfaces

Summary



UiA

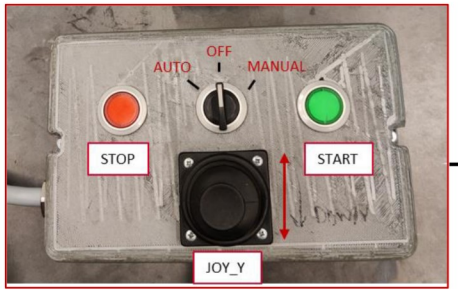
Part I: Presentation of application

1. System overview
2. Relevant IO
3. Motion control
4. Safety functions
5. Control input
6. Visualization/PLC HMI
7. Programming task

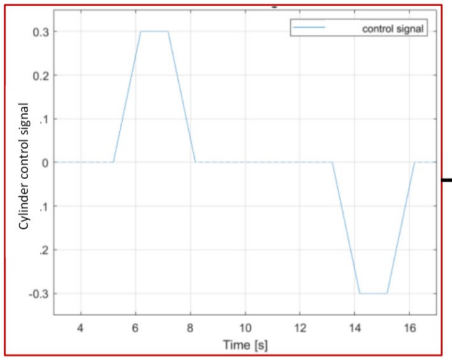
UiA

System overview

Manual Mode



Auto Mode



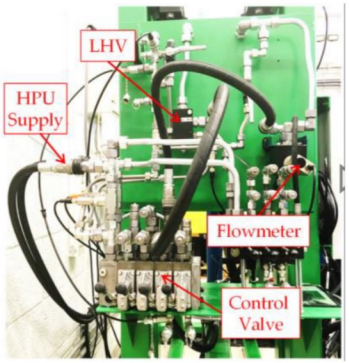
Real-Time Controller (PLC)

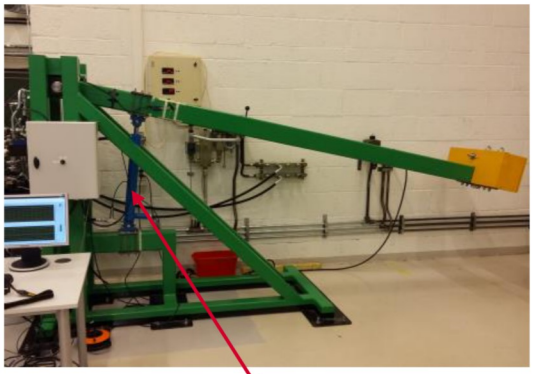
Mechatronics Application

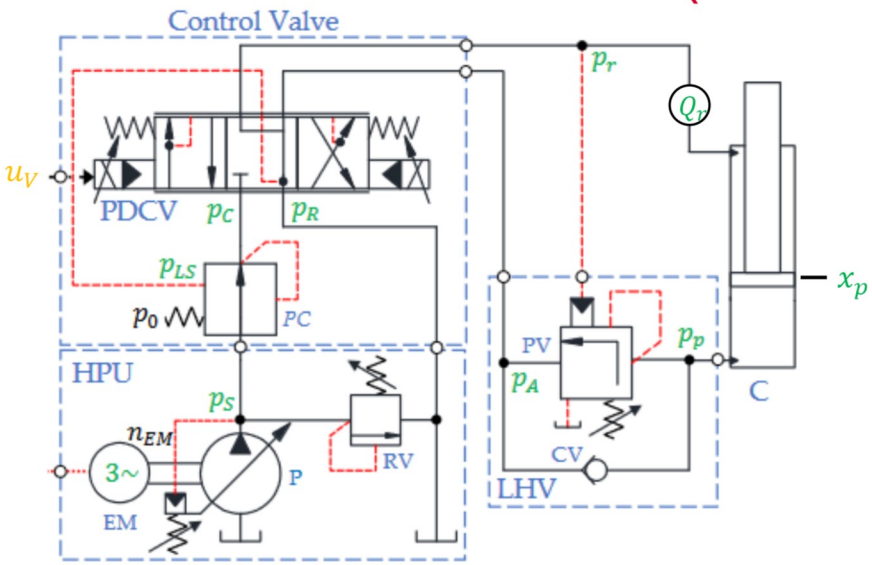
System overview

Relevant IO:

- Cylinder piston position sensor (input: 0…0.5 [m]) - x_p
- Pressure sensors (Input: 0…400 [bar])
  - Supply pressure - p_S
  - Return pressure - p_R
  - Cylinder piston-side pressure - p_p
  - Cylinder rod-side pressure - p_r
  - Pressure between **Control Valve** and **L**oad-**H**olding **V**alve (**LHV**) - p_A
- Flow sensors (Input: -150…150 [l/min])
  - Rod-side cylinder outlet flow - Q_r
- Control Valve (output: -1…1 [-]) - u_V

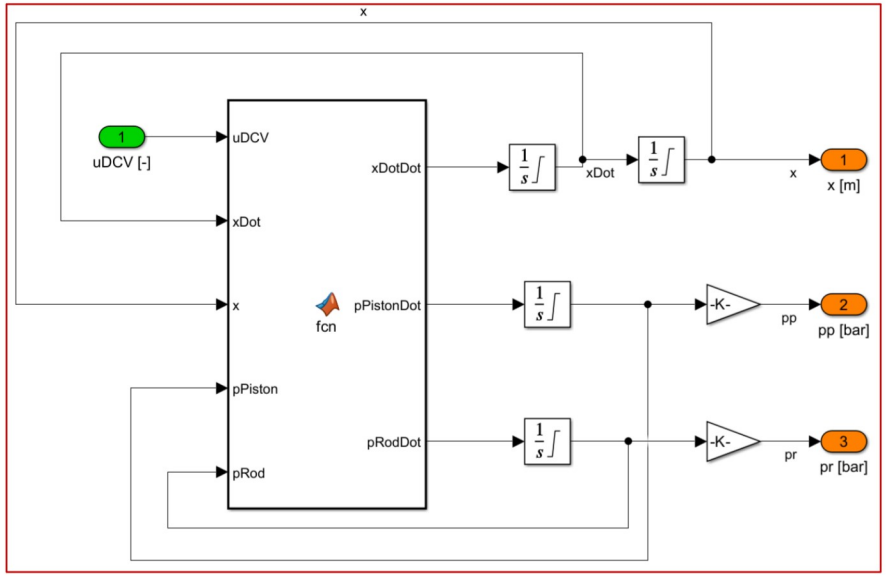


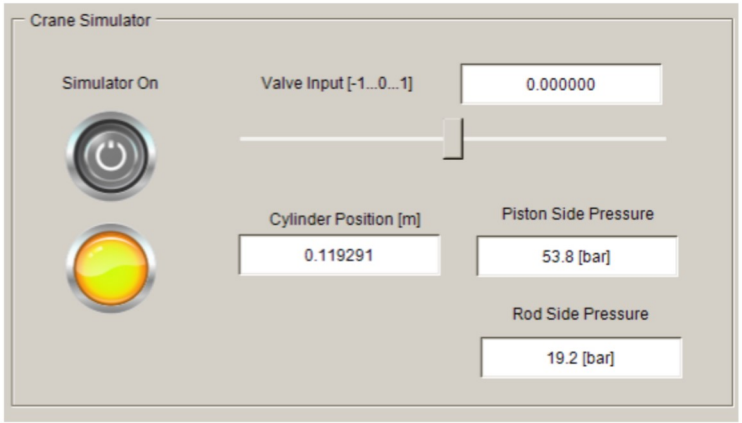




System overview

Simulator – simplified system:





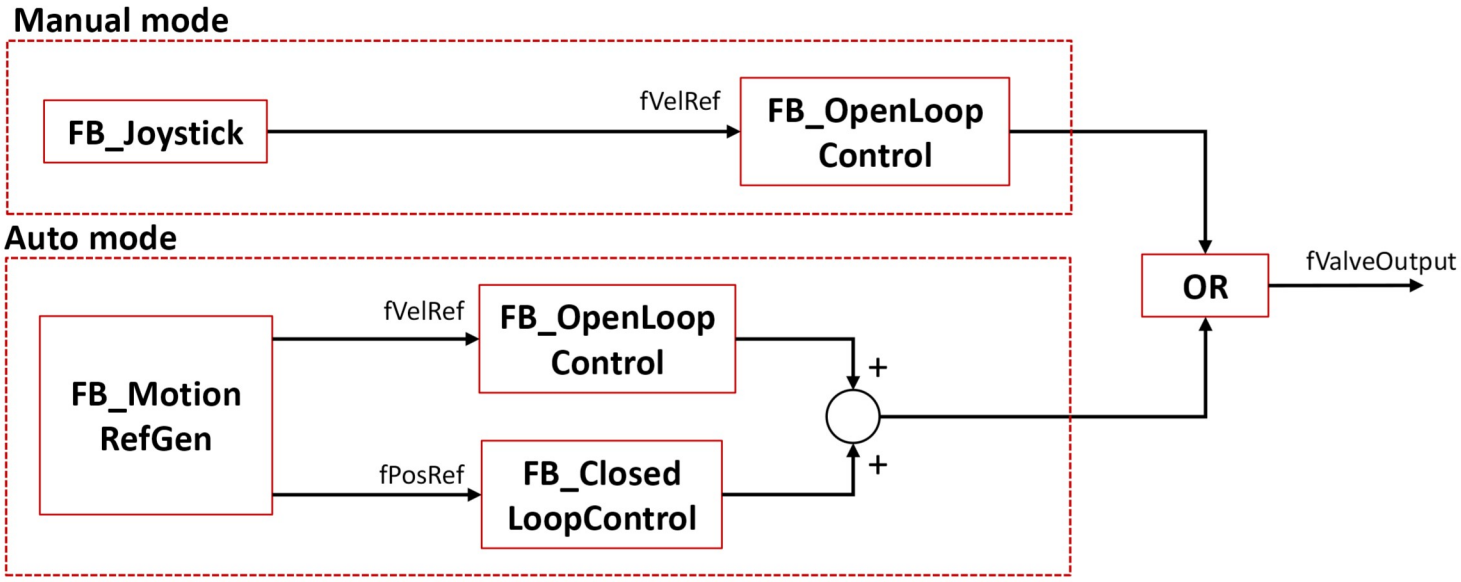
See Simulink model for details:
Simulink_PLC_coder_example_LAB_11_simulator.slx
https://github.com/DrDanielh/MAS418_SimulinkModel

UiA

Motion control

Overview

- **Manual mode**: Operator-in-the-loop control with joystick velocity feed forward control
  - Open-loop **velocity** control
- **Auto mode**: Automatic motion reference generation and position feedback control
  - Open-loop **velocity** control + Closed-loop **position** control

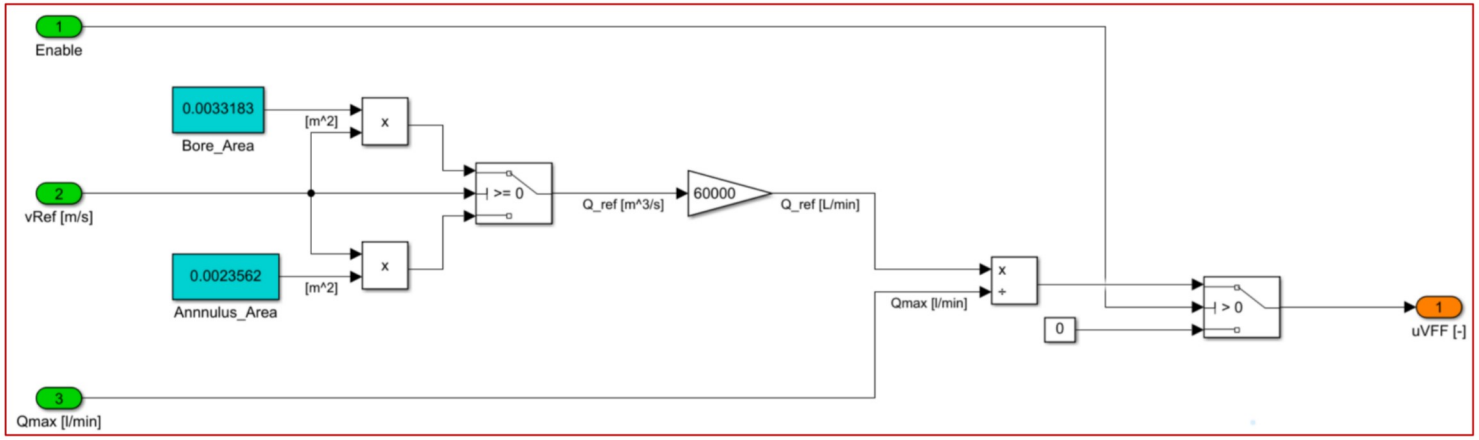


Motion control

FB_OpenLoop Control

Open-loop velocity control

- **Manual mode:** Operator-in-the-loop control with joystick input



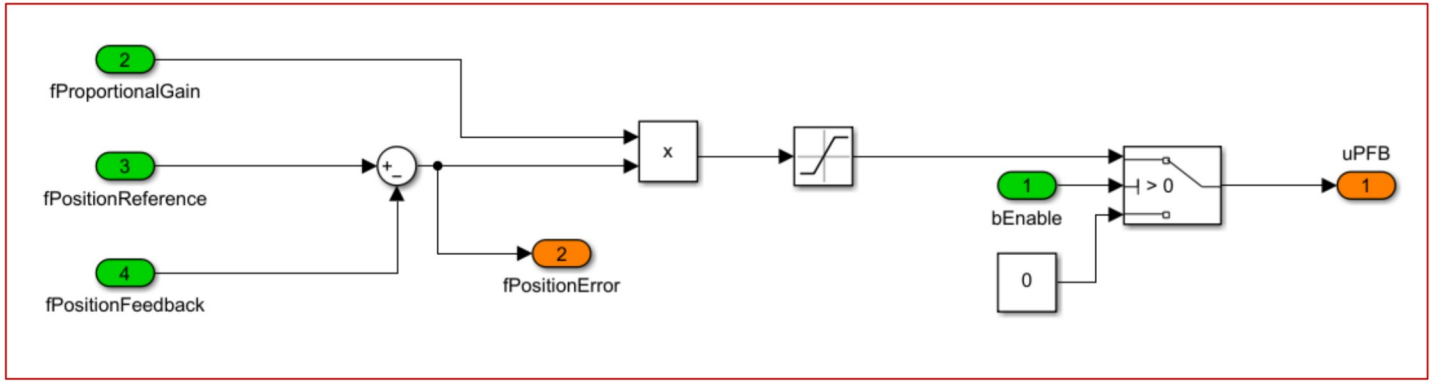
See Simulink model for details:
Simulink_PLC_coder_example_LAB_10.slx
https://github.com/DrDanielh/MAS418_SimulinkModel

Motion control

FB_Closed LoopControl

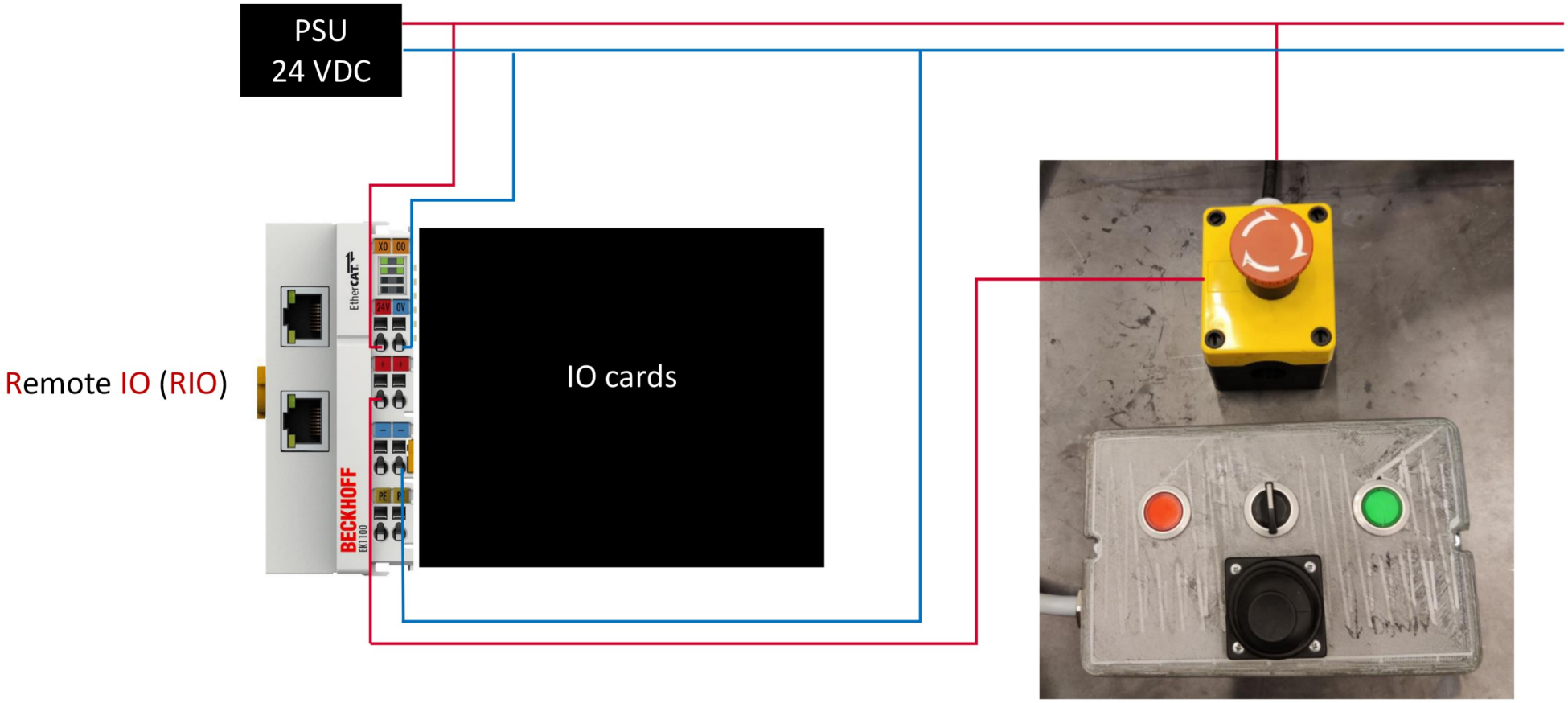
Closed-loop position control

- **Auto mode:** Automatic motion reference generation and position control



See Simulink model for details:
Simulink_PLC_coder_example_LAB_10.slx
https://github.com/DrDanielh/MAS418_SimulinkModel

Safety functions



Safety functions

- **OFF:** bEnable = FALSE

- **MANUAL:** Operator-in-the-loop control with joystick input

IF bEnable
AND bManualMode
AND bStart
AND NOT(bStop)

→ **FB_Joystick** → **FB_OpenLoop Control** → fValveOutput

- **AUTO:** Automatic motion reference generation and position control

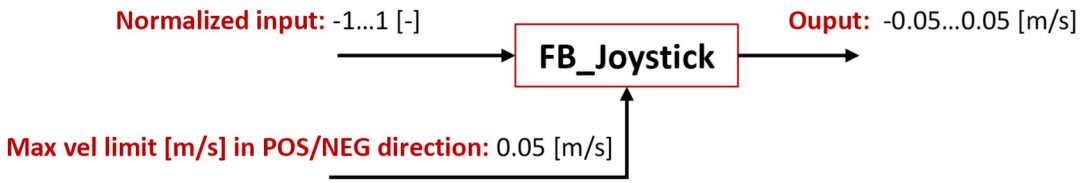
IF bEnable
AND bAutoMode
AND bStart (hold)
AND NOT(bStop)

→ **FB_Motion RefGen** → **FB_Position Control** → fValveOutput

UiA

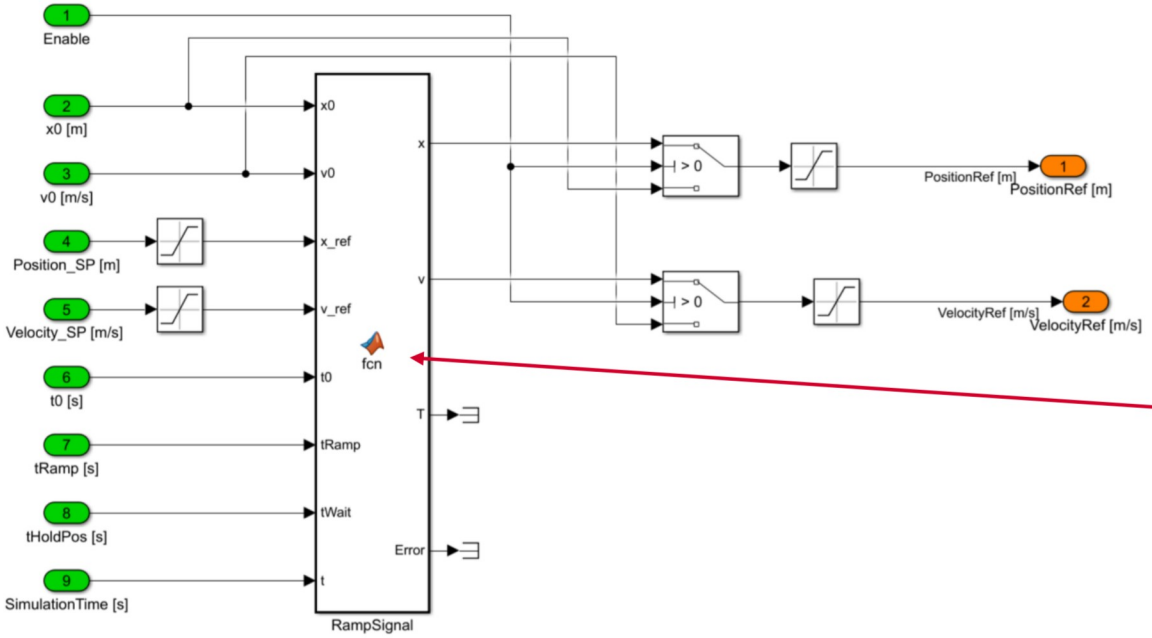
Control input

Joystick



Control input

Motion Reference Generator (Ramp function)




```
function [x,v,T, Error] = fcn(x0,v0, x_ref,v_ref,t0,tRamp,tWait,t)
x_SP = x_ref - x0;
vs=v_ref;
slopeExt=v0-vs;
slopeRetr=-vs-v0;

as = vs/tRamp;
s_acc=(vs^2-v0^2)/as;

tHold=(x_SP-s_acc)/vs;

if tHold < 0
Error = 1;
else
Error = 0;
end

t1=tRamp;
t2=tHold;
t3=tRamp;
t4=tWait;
t5=t1;
t6=t2;
t7=t3;

x1 = x0 + v0*((t0+t1)-t0)-(slopeExt/t1)*((t0+t1)-t0)^2/2;
x2 = x1 + vs*((t0+t1+t2)-(t0+t1));
x4 = x_ref - v0*((t0+t1+t2+t3+t4+t5)-(t0+t1+t2+t3+t4))+(slopeRetr/t5)*((t0+t1+t2+t3+t4+t5)-(t0+t1+t2+t3+t4))^2/2;
x5 = x4-vs*((t0+t1+t2+t3+t4+t5+t6)-(t0+t1+t2+t3+t4+t5));

if Error == 1
x = x0;
v = v0;
elseif t>=0 && t<t0
x = x0;
v = v0;
elseif t>=t0 && t<(t0+t1)
x = x0 + v0*(t-t0)-(slopeExt/t1)*(t-t0)^2/2;
v = v0-(slopeExt/t1)*(t-t0);
elseif t>=(t0+t1) && t<(t0+t1+t2)
x = x1 + vs*(t-(t0+t1));
v = vs;
elseif t>=(t0+t1+t2) && t<(t0+t1+t2+t3)
x = x2+vs*(t-(t0+t1+t2))+(slopeExt/t3)*(t-(t0+t1+t2))^2/2;
v = vs+(slopeExt/t3)*(t-(t0+t1+t2));
elseif t>=(t0+t1+t2+t3) && t<(t0+t1+t2+t3+t4)
x = x_ref;
v = v0;
elseif t>=(t0+t1+t2+t3+t4) && t<(t0+t1+t2+t3+t4+t5)
x = x_ref - v0*(t-(t0+t1+t2+t3+t4))+(slopeRetr/t5)*(t-(t0+t1+t2+t3+t4))^2/2;
v = v0+(slopeRetr/t5)*(t-(t0+t1+t2+t3+t4));
elseif t>=(t0+t1+t2+t3+t4+t5) && t<(t0+t1+t2+t3+t4+t5+t6)
x = x4-vs*(t-(t0+t1+t2+t3+t4+t5));
v = -vs;
elseif t>=(t0+t1+t2+t3+t4+t5+t6) && t<(t0+t1+t2+t3+t4+t5+t6+t7)
x = x5-vs*(t-(t0+t1+t2+t3+t4+t5+t6))-(slopeRetr/t3)*(t-(t0+t1+t2+t3+t4+t5+t6))^2/2;
v = -vs-(slopeRetr/t3)*(t-(t0+t1+t2+t3+t4+t5+t6));
else
x = x0;
v = v0;
end

T = t0+t1+t2+t3+t4+t5+t6+t7;
```

FB_Motion RefGen

See Simulink model for details:
Simulink_PLC_coder_example.slx
https://github.com/DrDanielh/MAS418_SimulinkModel

Visualization/PLC HMI

- Create a visualization representing the physical control box and the functions labeled in the figure
  - The Joystick can be programmed as a slider gain with input **-1**...**1** with **0** in zero position
  - Rotary knob gives feedback (**DI**) only when in **AUTO** or **MANUAL**
    - Since the rotary knob in the visualization is either **ON** or **OFF** use two rotary knobs, one for **ON**/**OFF** signal, and one for **AUTO**/**MANUAL**
  - The push buttons (**DI**) for START and STOP have light (**DO**). **RUNNING** status → GEEN light and **FAULT** status → RED light.
    - Since the push buttons in the visualizer don't have variable for light use separate lamps
    - In Auto mode, the start button must be programmed to be pressed in all the time to generate motion ref (i.e. clock input). If released clock (motion ref) stops.

Programming task – Exercise #2.3-2.4

Functionality that must be programmed:

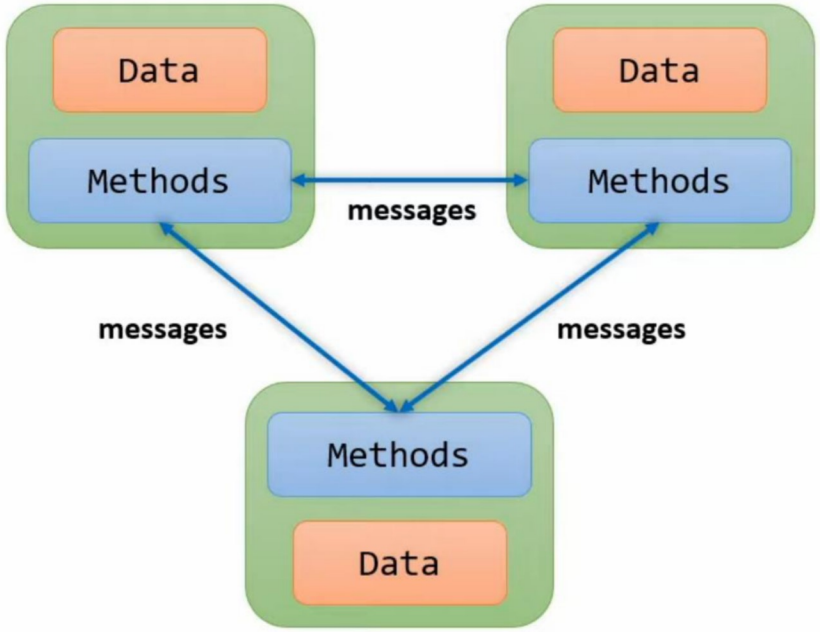
- **Virtual control box** with feedback on selected mode (Off, Manual, Auto) and machine status (NotReady, Ready, Starting, Running, Stopping, Fault)
- **Manual mode** with open-loop velocity feed forward control of both actuators from control box
- **Auto mode** with closed-loop position control of selected actuator with start/stop from control box
- IO interface (next lecture)

Part II: Function Blocks

1. Introduction
2. Function blocks
3. Methods
4. Inheritance

Introduction

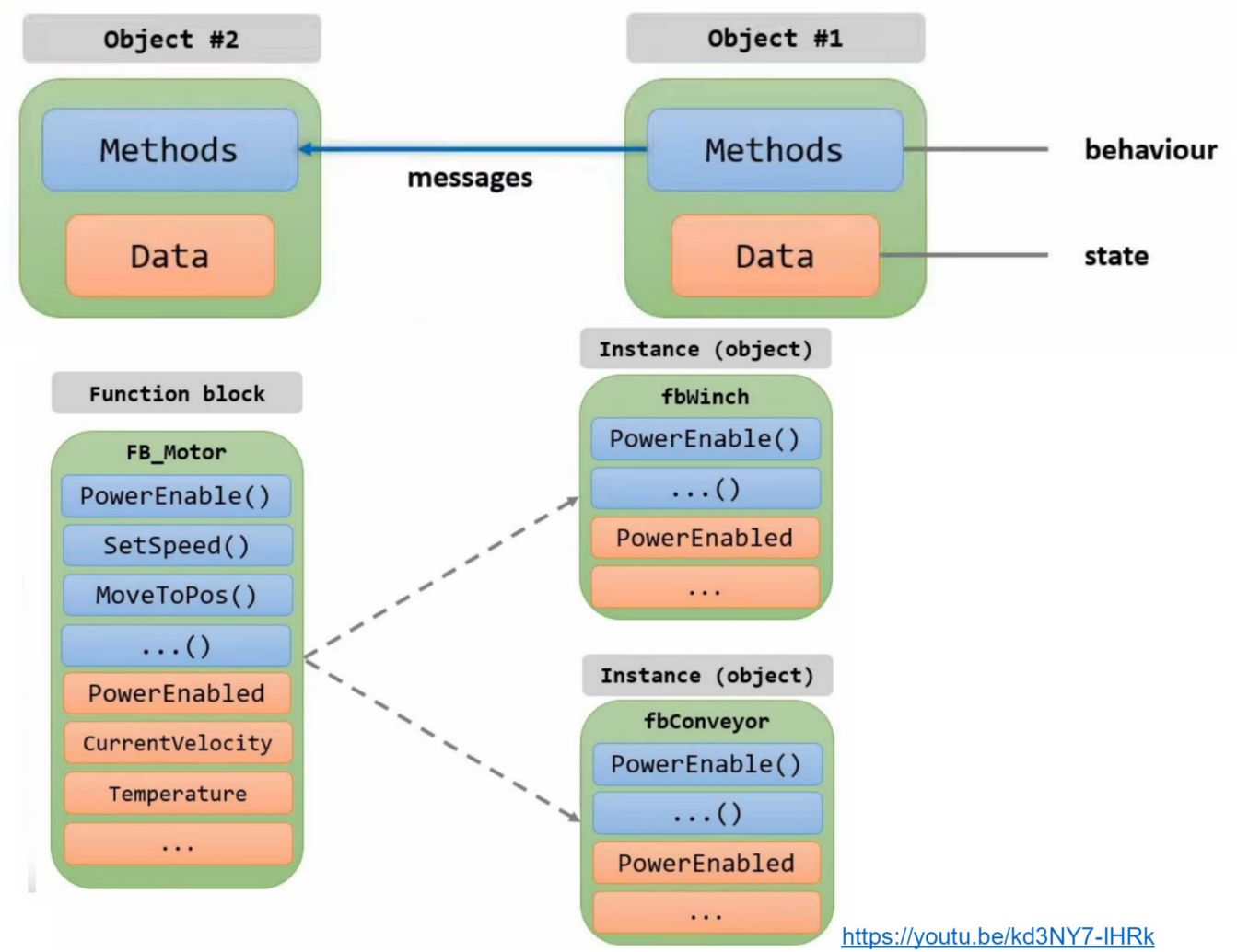
- **Function blocks** are our first steps into the world of object oriented programming, and thanks to them we can combine state with behavior
- Starting using **function blocks** in structured text is comparable to when going from C-style structures and functions and into classes in C++
- With **function blocks** we can go from working in a procedural style programming into object-oriented style programming



https://youtu.be/kd3NY7-IHRk

Introduction

- **What is an object?**




```
VAR
    fbWinch : FB_Motor;
    fbConveyor : FB_Motor;
END_VAR
```

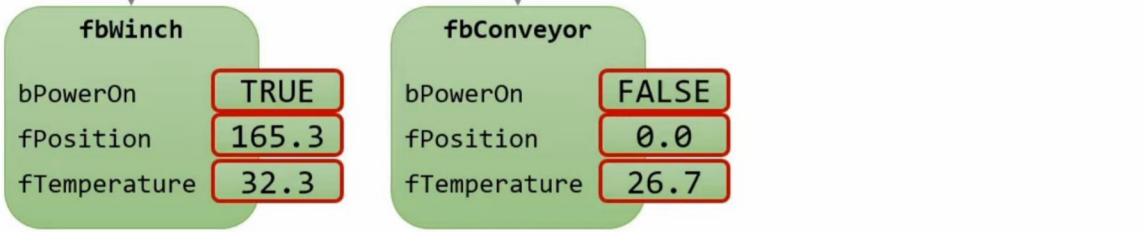
https://youtu.be/kd3NY7-IHRk

Function blocks

```
FUNCTION_BLOCK FB_Motor
VAR
    bPowerOn : BOOL; // [true = on, false = off]
    fPosition : REAL; // [mm]
    fTemperature : REAL; // [°C]
END_VAR
```

internal (instance) variables

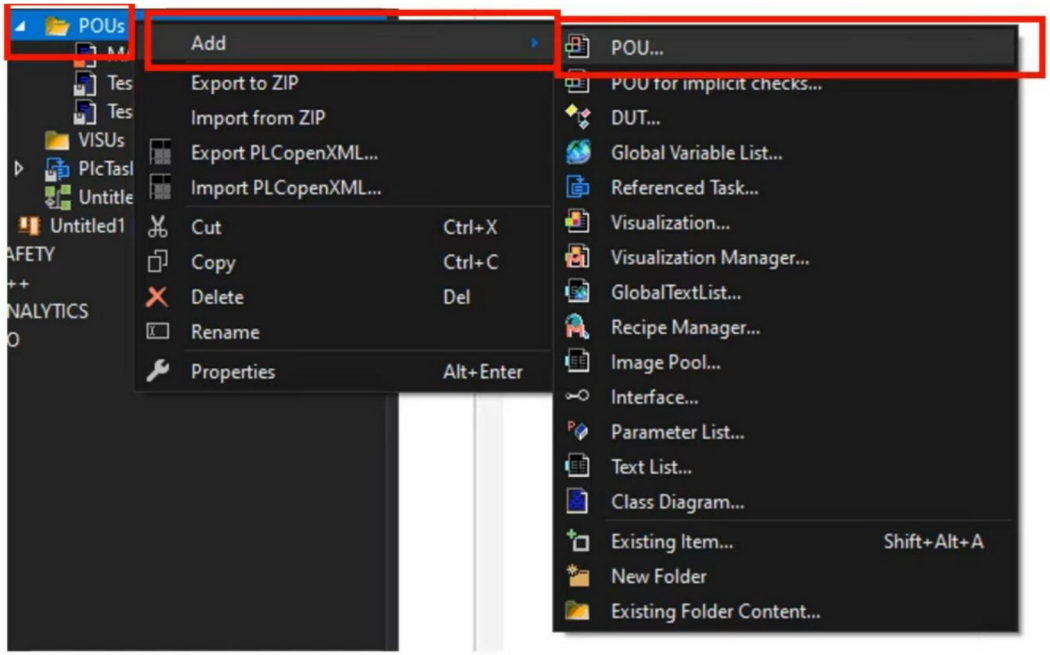
```
PROGRAM MAIN
VAR
    fbWinch : FB_Motor;
    fbConveyor : FB_Motor;
END_VAR
```

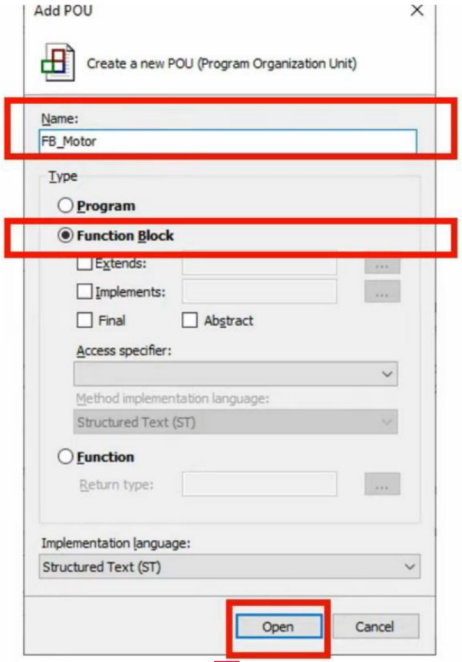


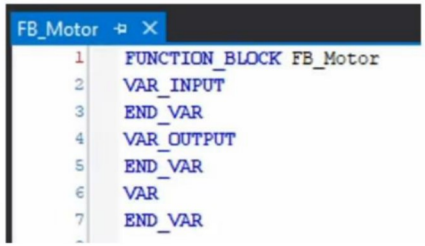
UiA

https://youtu.be/kd3NY7-IHRk

Function blocks







https://youtu.be/kd3NY7-IHRk

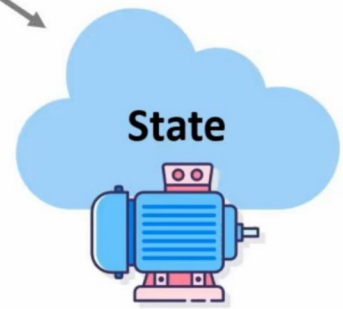
Methods

How do we access the internal state of the object, and how do we change that state?

```
FUNCTION_BLOCK FB_Motor
```

```
METHOD PUBLIC MoveToPosition
VAR_INPUT
    fPosition : REAL; // [mm]
END_VAR
VAR
    // local (scratch) variables
END_VAR
// code
```

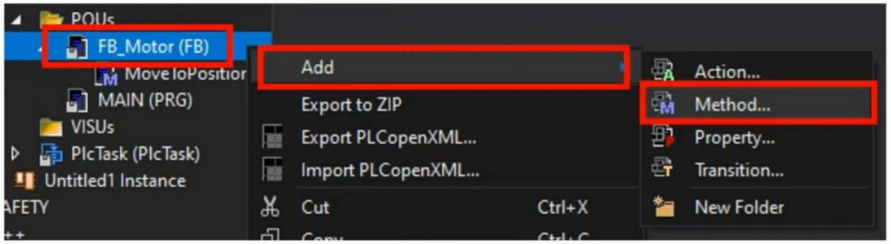
```
METHOD PUBLIC PowerEnable
VAR_INPUT
    bPowerOn : BOOL;
END_VAR
// code
```

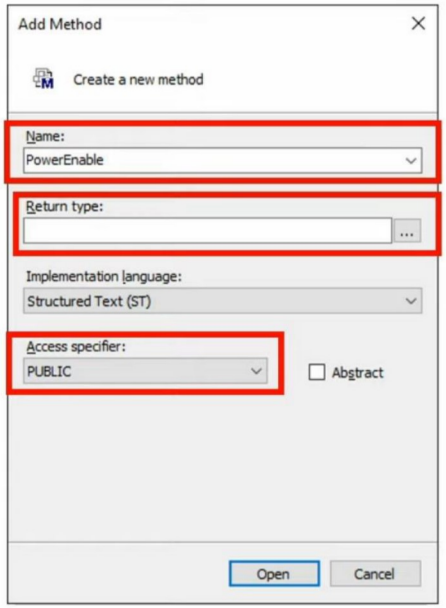


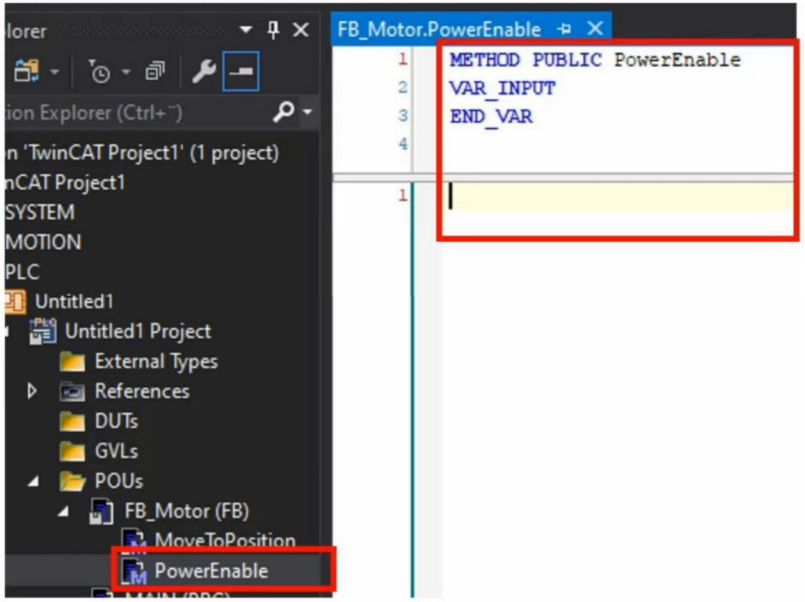

```
PROGRAM MAIN
VAR
    fbWinch : FB_Motor;
END_VAR
fbWinch.PowerEnable(true);
fbWinch.MoveToPosition(42.0);
```

https://youtu.be/kd3NY7-IHRk

Methods





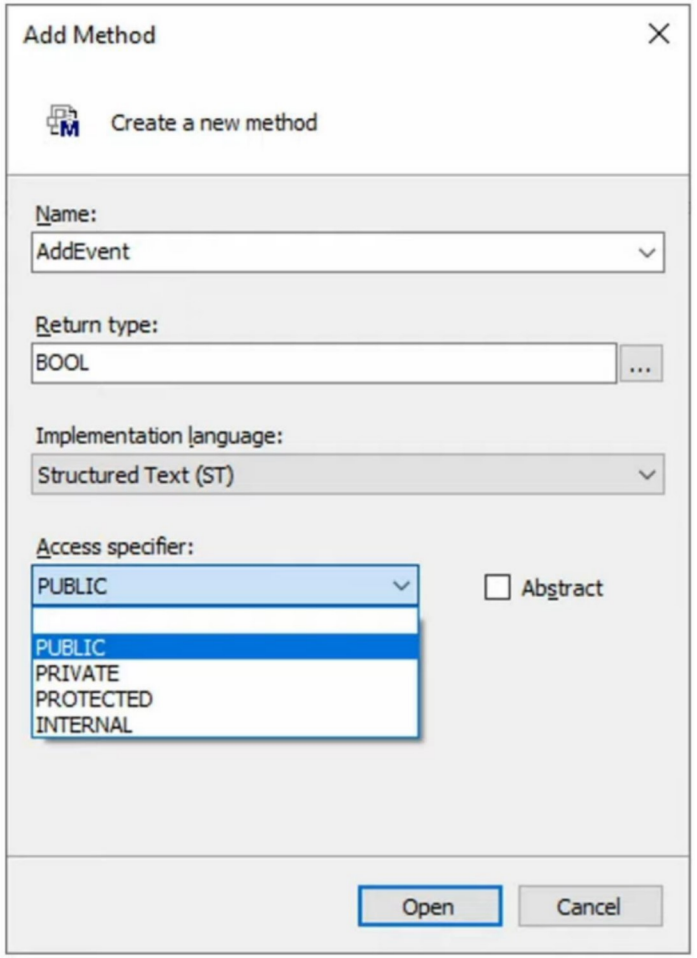


https://youtu.be/kd3NY7-IHRk

Methods

Access specifiers

- **PUBLIC**
  - No restriction on accessing methods
- **PRIVATE**
  - Access is limited to within function block definition
- **PROTECTED**
  - Access is limited to within the function block definition and any function block that inherits from the function block
- **INTERNAL**
  - Access is limited to the library



https://youtu.be/kd3NY7-IHRk

Methods

```
FUNCTION_BLOCK FB_Motor
VAR_INPUT
    fPosition : REAL; // [mm]
END_VAR
```

```
PROGRAM MAIN
VAR
    fbWinch : FB_Motor;
END_VAR
```

```
fbWinch(fPosition := 42.0);
```

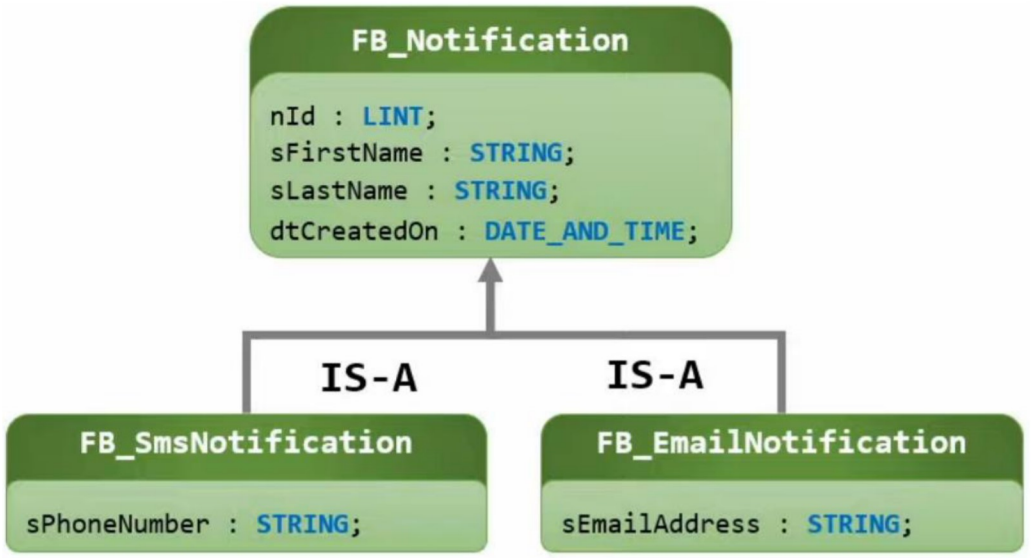
https://youtu.be/kd3NY7-IHRk

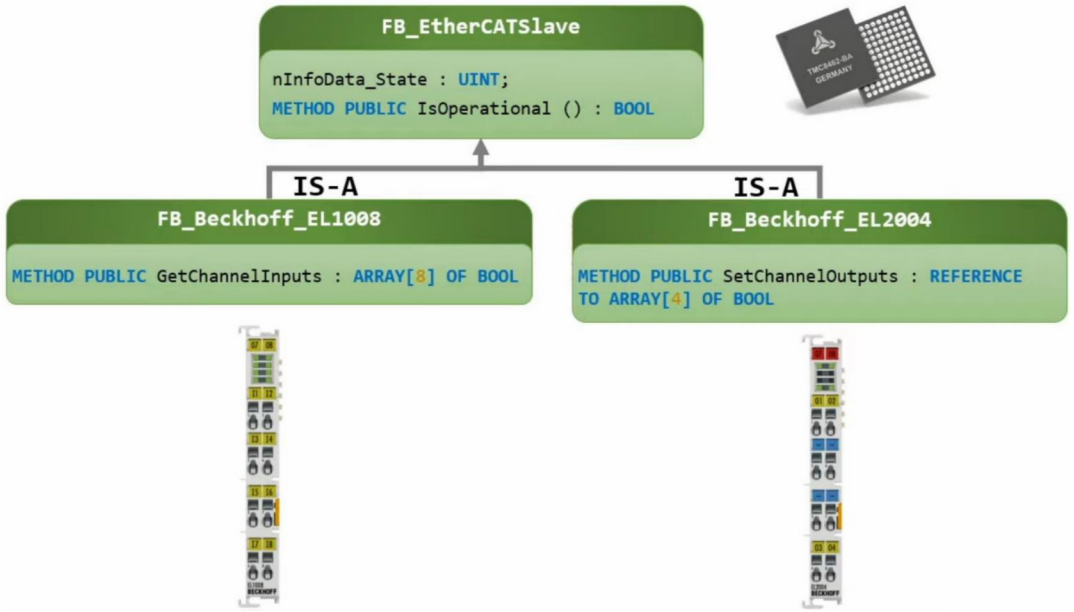
Methods

- When should you use the body of the **function block** and when should you use methods for changing the state of an instance of the **function block**?
  - **In general:**
    - Use the body of the **function block** for cyclically calls (i.e. calls that need to be made in every PLC-cycle)
    - and methods when they are a one-shot call for a request or message to change state
- You generally want to have very small **function blocks** that are doing one thing only, but that are doing that one thing very good

https://youtu.be/kd3NY7-IHRk

Inheritance






```
FUNCTION_BLOCK FB_Beckhoff_EL1008 EXTENDS FB_EtherCATSlave
```

```
PROGRAM MAIN
VAR
    fbEl1008_term1 : FB_Beckhoff_EL1008;
    bIsTerm1_Op : BOOL;
END_VAR
```

```
bIsTerm1_Op := fbEL1008_term1.IsOperational();
```

UiA

https://youtu.be/kd3NY7-IHRk

Inheritance

- Used in the right place's **inheritance** can be very useful
  - But it can cause a lot of problems

https://youtu.be/kd3NY7-IHRk

Part III: Interfaces

Interfaces

- Just as with **inheritance**, **interfaces** in IEC 61131-3 is a mechanism to achieve abstraction
- **Interfaces** are one way of achieving one of the pillars of object-oriented programming called **polymorphism**
  - **Polymorphism** is the abstract concept of dealing with multiple types in a uniform manner, and interfaces are one way to implement that concept

Interfaces:

1. **What to do, not how**
2. **Blueprint of function block**
3. **Programming by contract**

UiA

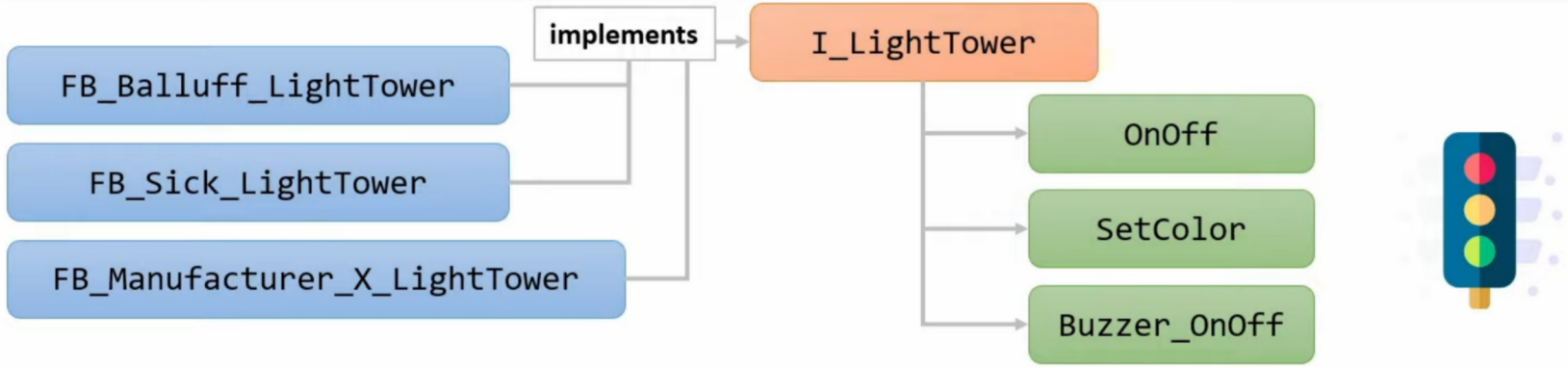
https://youtu.be/phpFd2UIFwo

Interfaces

```
INTERFACE I_LightTower
    METHOD OnOff
    VAR_INPUT
        On : BOOL; // [True=Light On]
    END_VAR
    METHOD SetColor
    VAR_INPUT
        Color : E_Color;
    END_VAR
    METHOD Buzzer_OnOff
    VAR_INPUT
        On : BOOL; // [True=Buzzer on]
    END_VAR
```

Interfaces:

1. **What to do, not how**
2. **Blueprint of function block**
3. **Programming by contract**



https://youtu.be/phpFd2UIFwo

Interfaces

```
INTERFACE I_LightTower
    METHOD OnOff
    VAR_INPUT
        On : BOOL; // [True=Light On]
    END_VAR
```

```
FUNCTION_BLOCK Balluff_LightTower IMPLEMENTS I_LightTower
    METHOD OnOff
    VAR_INPUT
        On : BOOL; // [True=Light On]
    END_VAR
```

```
// Implementing code for actually turning
// on/off the Balluff light tower, for
// example by using digital outputs or by a
// fieldbus such as EtherCAT
```

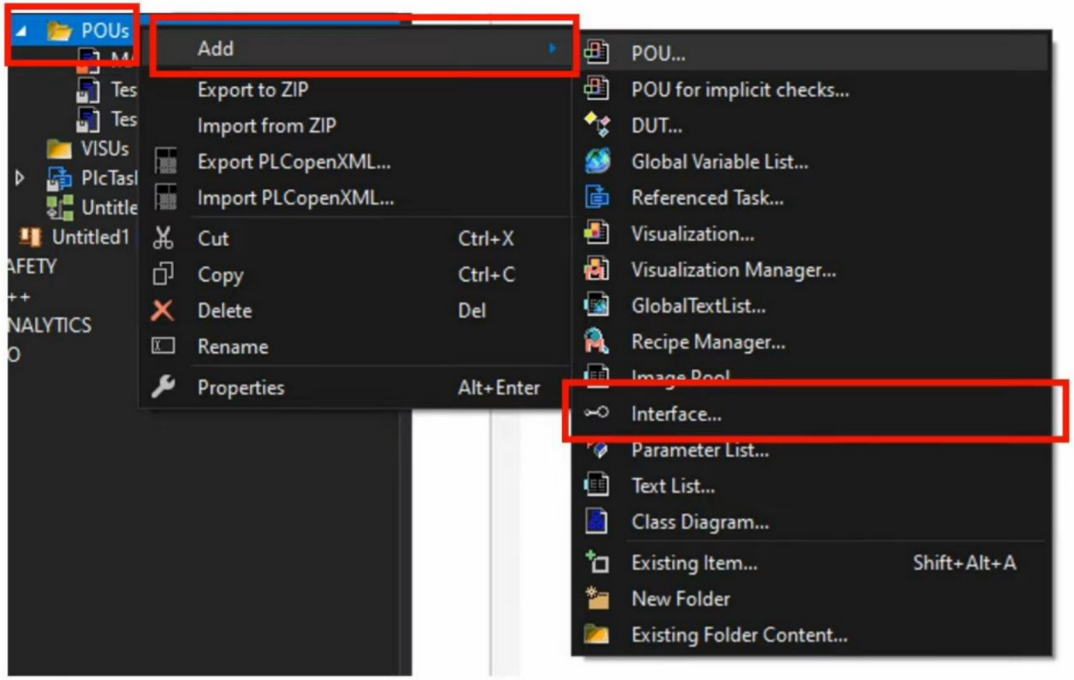
```
PROGRAM MAIN
VAR
  fbBalluffLightTower : FB_Balluff_LightTower;
  fbSickLightTower : FB_Sick_LightTower;
  fbLightTower : I_LightTower := fbBalluffLightTower;
END_VAR
```

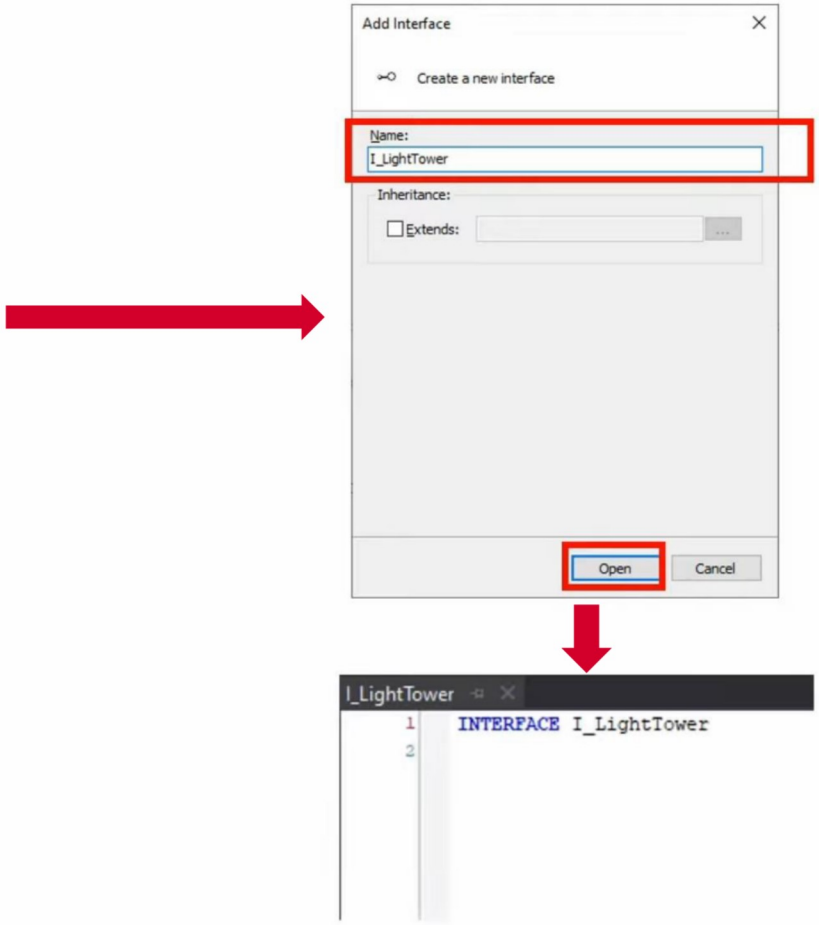
```
fbLightTower := fbSickLightTower;
fbLightTower.On(TRUE);
```

UiA

https://youtu.be/phpFd2UIFwo

Interfaces





https://youtu.be/phpFd2UIFwo

Interfaces

- Now you might be wondering why you need **interfaces**?

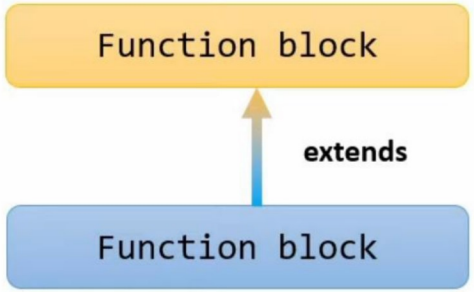
1. Abstraction
2. Team collaboration
3. Testing

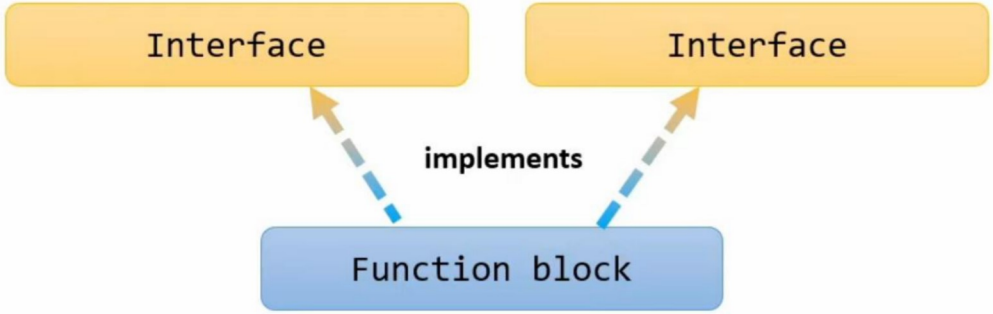
- The key points of **interfaces** is that they provide a layer of abstraction so that you can write code that is ignorant of unnecessary details

https://youtu.be/phpFd2UIFwo

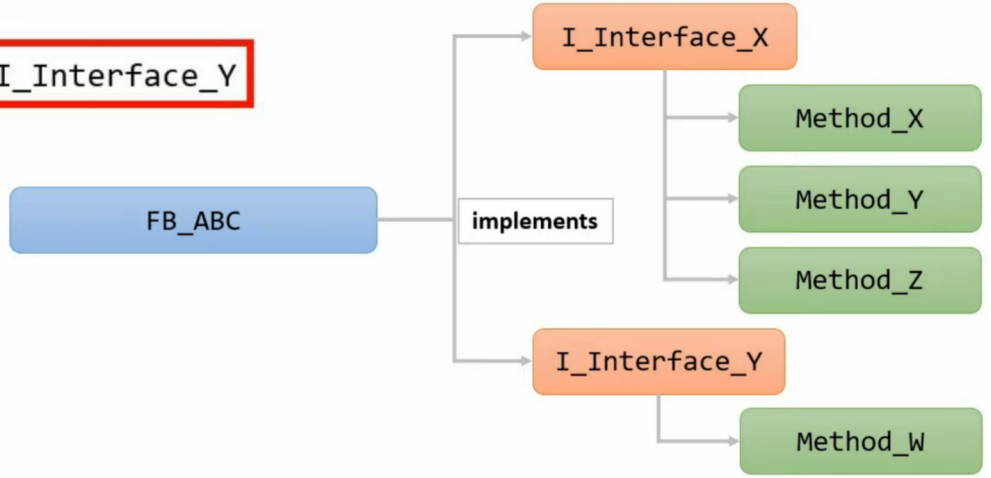
Interfaces

```
FUNCTION_BLOCK FB_A EXTENDS FB_B
```






```
FUNCTION_BLOCK FB_ABC IMPLEMENTS I_Interface_X, I_Interface_Y
```



https://youtu.be/phpFd2UIFwo

Interfaces

- Just as with the part about **function blocks**, we only have scratched the surface of the possibilities of **interfaces**
- **Interfaces** are so much more than just writing IMPLEMENTS and implementing an interface
- By using **interfaces**, we can design our software in a much more modularized way and by utilizing certain design patterns we can make our software more robust

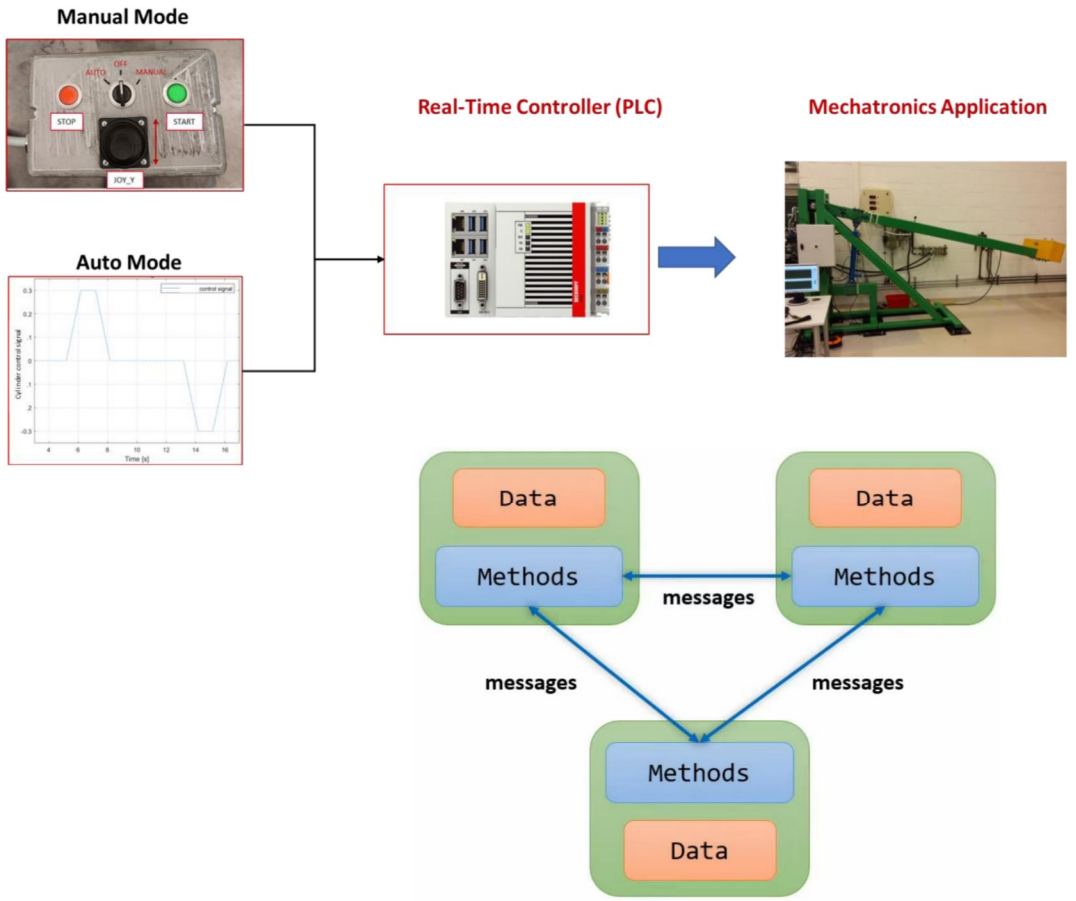
https://youtu.be/phpFd2UIFwo

Summary

UiA

Summary

- **Object-oriented PLC programming**
  - **I. Presentation of application**
    - System overview
    - Relevant IO
    - Motion control
    - Safety system
    - Control input
    - Programming task
  - **II. Function Blocks**
    - Introduction
    - Function blocks
    - Methods
    - Inheritance
  - **III. Interfaces**



Next Lecture

Advance topics and Machine interface:

I. TwinCAT advance
II. Test driven development
III. Machine Interface

- **Homework:**
  - Watch video: [Stop using global variables!]
  - Work with the exercise.
  - Look at earlier exams (will be shared in Canvas) with respect to the grading method presented in [Lecture #2.1] slide 11.

Januar 2024

Uke	Ma	Ti	On	To	Fr	Lø	Sø
1	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14
3	15	16	17	18	19	20	21
4	22	23	24	25	26	27	28
5	29	30	31				

1.1: 1. nyttårsdag

Februar 2024

Uke	Ma	Ti	On	To	Fr	Lø	Sø
5				1	2	3	4
6	5	6	7	8	9	10	11
7	12	13	14	15	16	17	18
8	19	20	21	22	23	24	25
9	26	27	28	29			

Mars 2024

Uke	Ma	Ti	On	To	Fr	Lø	Sø
9					1	2	3
10	4	5	6	7	8	9	10
11	11	12	13	14	15	16	17
12	18	19	20	21	22	23	24
13	25	26	27	28	29	30	31

24.3: Palmesøndag, 28.3: Skjærtorsdag, 29.3: Langfredag, 31.3: 1. påskedag

April 2024

Uke	Ma	Ti	On	To	Fr	Lø	Sø
14	1	2	3	4	5	6	7
15	8	9	10	11	12	13	14
16	15	16	17	18	19	20	21
17	22	23	24	25	26	27	28
18	29	30					

1.4: 2. påskedag

Mai 2024

Uke	Ma	Ti	On	To	Fr	Lø	Sø
18			1	2	3	4	5
19	6	7	8	9	10	11	12
20	13	14	15	16	17	18	19
21	20	21	22	23	24	25	26
22	27	28	29	30	31		

1.5: Offentlig høytidsdag, 9.5: Kristi Himmelfartsdag, 17.5: Grunnlovsdag, 19.5: 1. pinsedag, 20.5: 2. pinsedag

Juni 2024

Uke	Ma	Ti	On	To	Fr	Lø	Sø
22						1	2
23	3	4	5	6	7	8	9
24	10	11	12	13	14	15	16
25	17	18	19	20	21	22	23
26	24	25	26	27	28	29	30

Self-study | Part #1 | Part #2 | Part #3 | Exam

Lab exercise

#2.3 - Object-oriented PLC programming