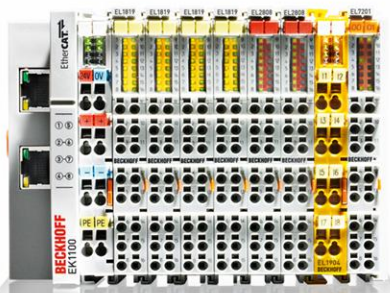
PLC_MOTION_LAYER
TwinCAT project

BECKHOFF



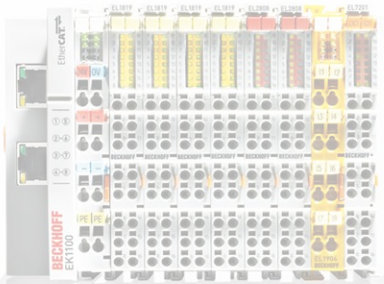
PLC_MOTION_LAYER

BECKHOFF

Daniel Hauer

Introduction to motion layer approach

- motivation for a motion layer
- use cases
- specific construction



Motivation:

- **Centralized control:**
  - Sync communication
  - Async communication
  - Agnostic to communication technology

- **Decentralized execution of motion tasks**
  - Data consistency in cyclic multi task environments
  - Local machine with modular design specifications
  - Unified procedure for modular software architecture
  - Use of top layer consistent through different architectures

Motivation:

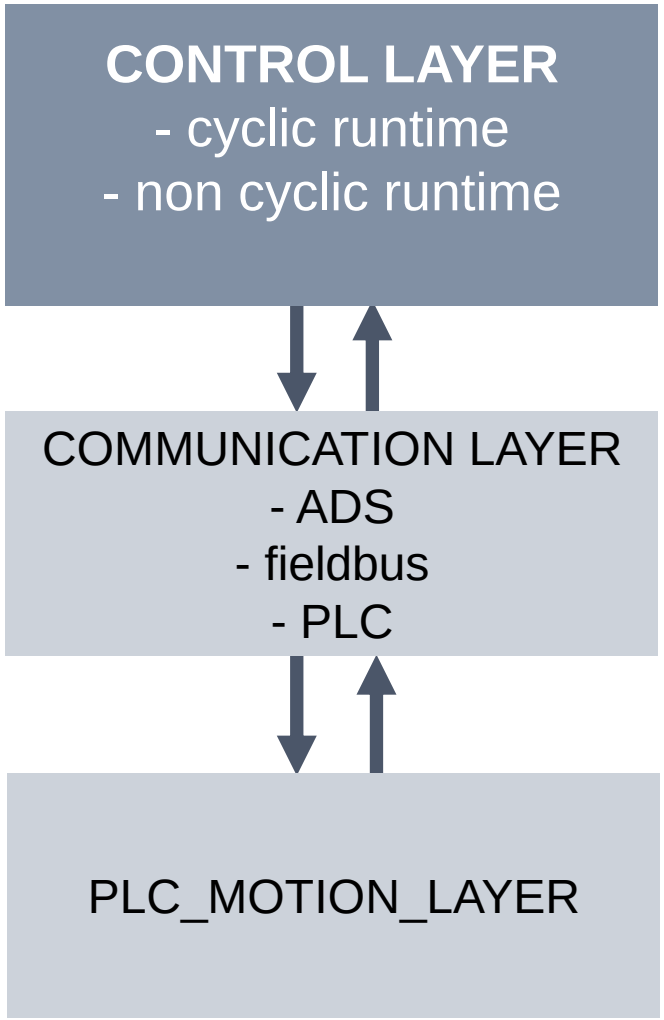
- **Use of TwinCAT supported/updated libraries**
  - Tc2_MC2
  - Tc2_MC2_Drive
  - Tc2_NC
  - Tc2_NCI
  - Tc2_PlcInterpolation
- **Open code base**
  - Migration to Tc3 MC in preparation
  - Customer/user specific changes possible
  - Access to code
  - Conversion to library possible by customer/user
- **Compiled PLC**
  - Source code is not on shipped machine

PLC_MOTION_LAYER

BECKHOFF

Motivation:

- Use of TwinCAT motion without detailed coding knowledge
- Transparency of communication layer
- Code base shall remain independent of control layer
- Configurable Options for specific libraries / TC functions
- Balanced load for configurable options in machine layout
- Stable cpu use for XFC applications

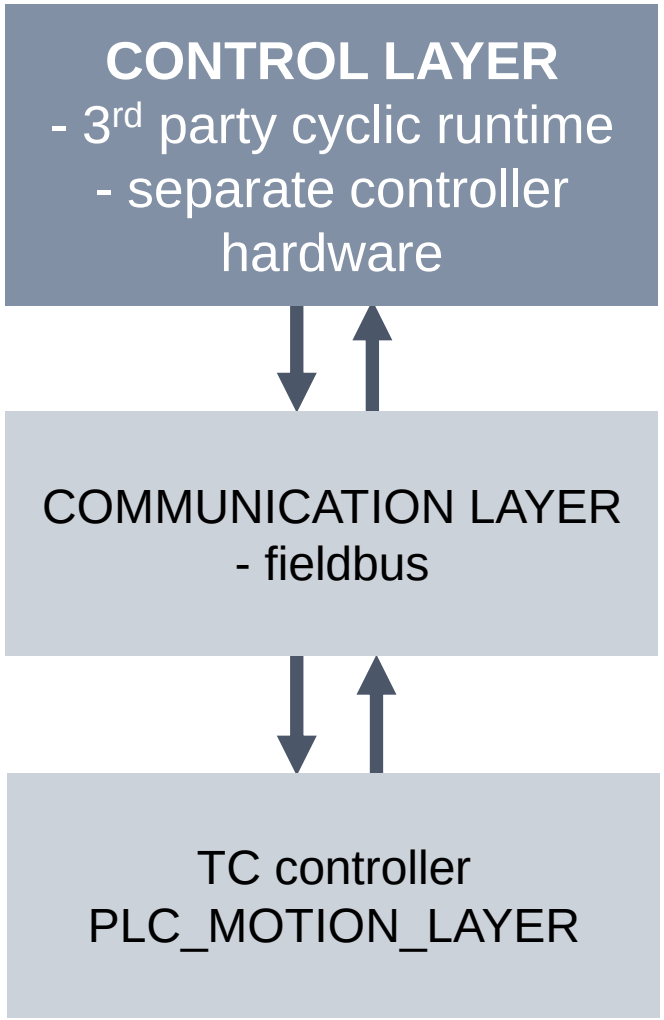


PLC_MOTION_LAYER

BECKHOFF

Use cases:

- Separate controller for machine logic
- Any fieldbus (EtherCAT, Profi…)
- Connected through TwinCAT mappings
- Execution of motion tasks in PLC_MOTION_LAYER TwinCAT controller

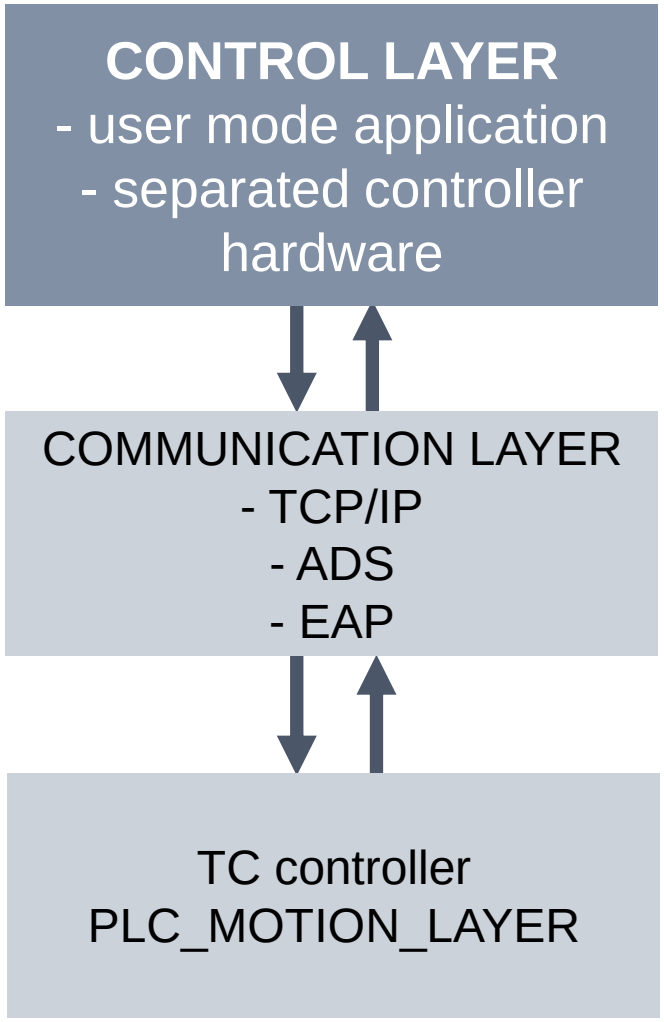


PLC_MOTION_LAYER

BECKHOFF

Use cases:

- Separate controller for machine logic
- Any network
- Connected through TwinCAT mappings
- Execution of motion tasks in PLC_MOTION_LAYER TwinCAT controller

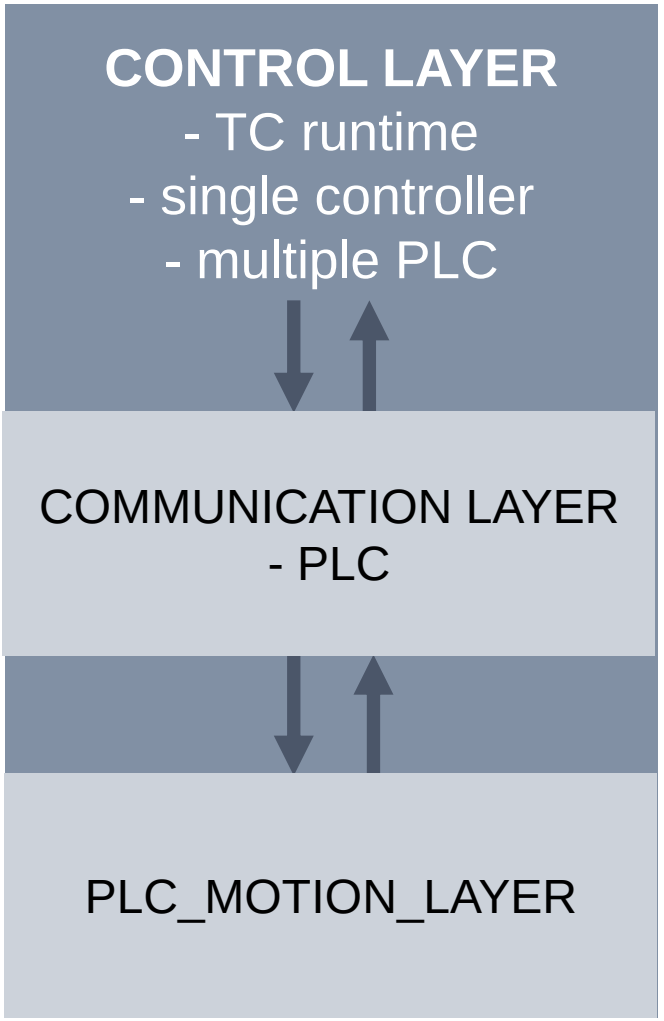


PLC_MOTION_LAYER

BECKHOFF

Use cases:

- One controller for machine logic
- Multiple PLC for machine logic
- Connected through TwinCAT mappings
- Execution of motion tasks

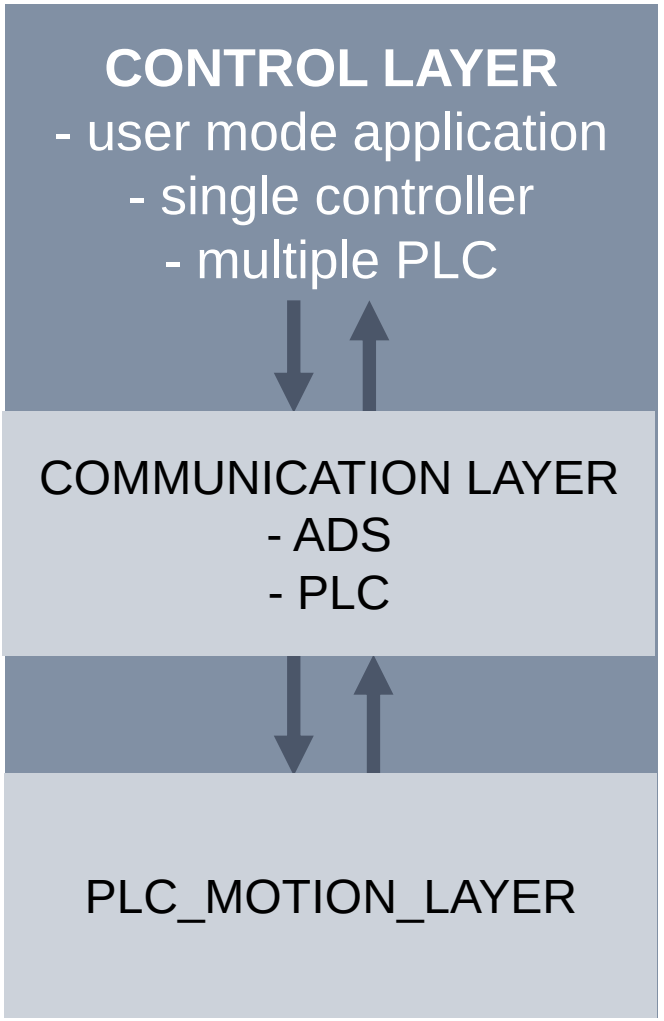


PLC_MOTION_LAYER

BECKHOFF

Use cases:

- One controller for machine logic
- User mode application AND/OR multiple PLC
- ADS for symbol access by user mode application
- TwinCAT mapping for connecting multiple PLCs for specific application purposes
- Execution of motion tasks



Specific construction:

- TwinCAT PLC project
- Use of specific syntactic code behaviour
- Software design
- Compiler defines / pragmas
- Logging system

TwinCAT project:

- Default TwinCAT project
  - Adjust core settings to target hardware
  - Add NC/PtP
    - Optional add NCI channel

- TwinCAT PLC
  - Add existing Item: PLC_MOTION_LAYER
  - Add task reference
  - Option: add compiler defines (NCI, CAM, BSD)

- Adjust constants in:
  - PLC_MOTION_LAYER/PLC_CONSTANT

- Compile
  - PLC_MOTION_LAYER Instance mapping is built

specific syntactic code behaviour:

- C like state machines
  - State changes **need not** consume one PLC cycle

  - Same cycle response to command on cyclic interface
    - → since we're not using C (unfortunately, but ST in TwinCAT3 is almost as beautiful)
      - → cases have to be 'broken up'
        - → take a close look, it probably looks weird, but it is fast AND safe to use in combination with E_PROGRESS

        - →look at FB_MaAxisBase.MovePosBuffer try to do it with IF conditionals, then come back and take a second look ;-)

specific syntactic code behaviour:

- OnChange detection for new commands
  - Cyclic check whether the command has changed

- State always carries offset about progress of command (E_PROGRESS) (example follows in a few pages)

- Library FBs are called within states
  - FBs are called when required and not a cycle longer (same is true for all Ctrl-Wrappers)

  - Working with empty cyclic calls is the best way to build voodoo software, just don’t do it!

```
_eCmd                           := _Ctrl^[_nAxisIndex].eCmd;
//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
//  OnChange react to command
//------------------------------------------------------------------------------
IF (_eCmd <> _eCmdOld)
THEN
  // get busy
  _eState                       := Cmd(_eCmd); // get execution state for command
  _eCmdOld                      := _eCmd;
  LogControl();
END_IF
```

Software Design:

- Every TwinCAT function has dedicated wrapper
  - Separate namespaces
  - Optional library binding

- Cross communication via interfaces
  - NCI, CAM, XFC use interfaces in order to enable optional binding
  - If compiler define is not set, empty interfaces are used instead of instances

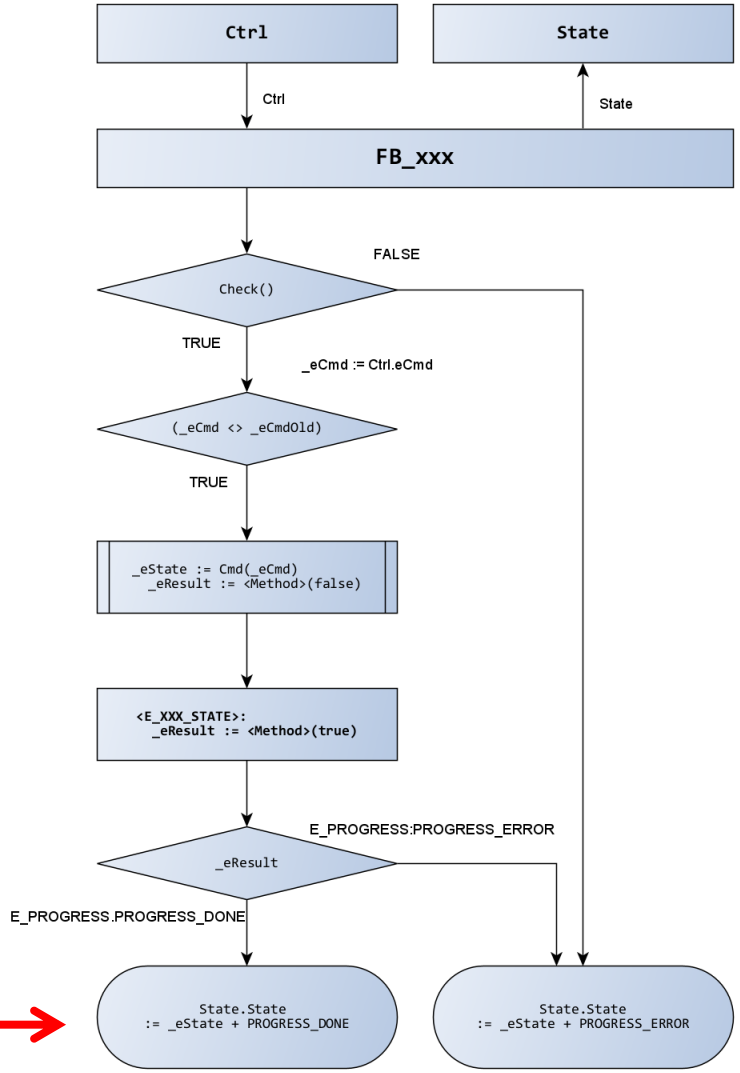
- Ctrl/State structures for commanding required function
  - PtP – ctrl/state
  - NCI – ctrl/state
  - CAMMING – ctrl/state

- Parameter structures carry required data for commanded function
  - PtP – (SetPos, SetVelo, SetAcc, MasterAxisIndex…)
  - NCI – (AxisGroupId, AxisIndex, MFunc, RParameter…)
  - CAMMING – (MasterAxisIndex, TableId…)

Software Design:

- State / Ctrl structures
  - Establishes unified access
  - Commands can simply be ‘dropped’ into Ctrl datafield
    - Enables asynchronuous communication with PLC_MOTION_LAYER (e.g. C# via ADS, C/C++ vis ADS, ADS over MQTT, …)
    - Enables cyclic communication with PLC_MOTION_LAYER since structures can easily be mapped onto any cyclic fieldbus TwinCAT supports (EtherCAT, Profinet, CanOpen, EAP, …)
  - State is updated by PLC_MOTION_LAYER, so you can move on doing other stuff, come back and check completion/error.

- State feedback for cyclic class wrappers
  - **Always** combined with E_PROGRESS
    - You can filter your response by a simple modulo division
      - The result may be your entry point for your reaction to State.



Software Design:

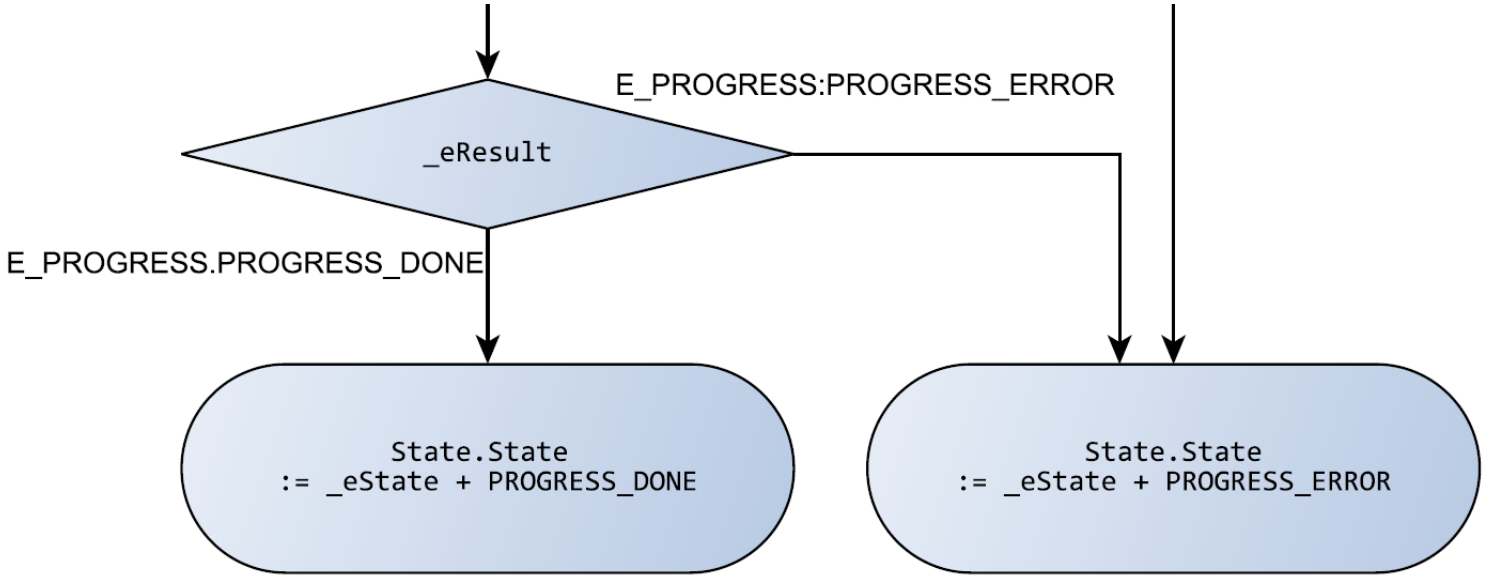
- E_PROGRESS
  - How far along is the command you just ‘dropped’?
    - This enum shall help you to build an answer state machine and not just a simple **IF** conditional
  - This enum shall be used everywhere and I recommend you use it in your extern control layer

E_PROGRESS

```
{attribute 'qualified_only'}
//{attribute 'strict'}
{attribute 'to_string'}
TYPE E_PROGRESS :
(
  // progress has 2 use cases
  // 1. as offset to cyclic interface's state
  //    for the requested command/function
  //    e.g. State := <Enum equivalent to eCmd> + E_PROGRESS
  //
  // 2. as state feedback (result) from (any) method
  PROGRESS_INVALID,
  PROGRESS_NOT_EXIST      :=   100,
  PROGRESS_INIT           :=  1000,
  PROGRESS_BUSY           :=  2000,
  PROGRESS_PREPARE        :=  3000,
  PROGRESS_STARTUP        :=  4000,
  PROGRESS_CHECK          :=  5000,
  PROGRESS_OCCUPIED       :=  6000,
  PROGRESS_WORKING        :=  7000,
  PROGRESS_STILL_WORKING  :=  8000,
  PROGRESS_ERROR          :=  9000,
  PROGRESS_DONE           := 10000
)UINT;
END_TYPE
```

Software Design:

- E_PROGRESS (example)
  - How far along is the command you just ‘dropped’?
    - This enum shall help you to build an answer state machine
  - This enum shall be used everywhere and I recommend you use it in your extern control layer too.



Software Design:

- E_PROGRESS (example)
  - How far along is the command you just ‘dropped’?
    - See example skeleton →
    - This enum shall help you to build an answer state machine
  - This enum shall be used everywhere and I recommend you use it in your extern control layer too.

```
//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
// react to operation modes (how to do it in an easy self documenting way)
//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
CASE (GVL_AXIS.State[i].eState MOD E_PROGRESS.PROGRESS_DONE)
OF
  E_AXIS_STATE.AXIS_INIT:
    ;// do something after success

  E_AXIS_STATE.AXIS_MOVE_POS::
    ;// do something after success
ELSE
  CASE (GVL_AXIS.State[i].eState MOD E_PROGRESS.PROGRESS_ERROR)
  OF
    E_AXIS_STATE.AXIS_INIT:
      ;// do something after success

    E_AXIS_STATE.AXIS_MOVE_POS::
      ;// do something after success
  ELSE
    ;// down the rabbit hole you go from here
  END_CASE
END_CASE
```

Software Design:

- E_PROGRESS (example log)
  - Different names
    - Same principle

```
eMessageInfo    2025-10-24-13:00:18.408 General_ID_0    General 30  ExampleEvalMachine :TRANSPORT_GROUP_CLEAR: PROGRESS_INIT
eMessageInfo    2025-10-24-13:00:18.418 General_ID_0    General 30  ExampleEvalMachine :TRANSPORT_GROUP_CLEAR: PROGRESS_BUSY
eMessageInfo    2025-10-24-13:00:18.458 General_ID_0    General 30  ExampleEvalMachine :TRANSPORT_GROUP_CLEAR: PROGRESS_PREPARE
eMessageInfo    2025-10-24-13:00:18.658 General_ID_0    General 30  ExampleEvalMachine :TRANSPORT_GROUP_CLEAR: PROGRESS_STARTUP
eMessageInfo    2025-10-24-13:00:18.858 General_ID_0    General 30  ExampleEvalMachine :TRANSPORT_GROUP_CLEAR: PROGRESS_CHECK
eMessageInfo    2025-10-24-13:00:18.888 General_ID_0    General 30  ExampleEvalMachine :TRANSPORT_GROUP_CLEAR: PROGRESS_OCCUPIED
eMessageInfo    2025-10-24-13:00:18.908 General_ID_0    General 30  ExampleEvalMachine :TRANSPORT_GROUP_CLEAR: PROGRESS_DONE
```

Software Design:

PLC_MOTION_LAYER / GVL_AXIS

```
//--------------------------------------
// command and state structure
//--------------------------------------
Ctrl  : ARRAY[1..MAX_AXIS] OF
        ST_AXIS_CTRL;

State : ARRAY[1..MAX_AXIS] OF
        ST_AXIS_STATE;

//--------------------------------------
// cyclic interface function block
//--------------------------------------
Control  : ARRAY[1..MAX_AXIS] OF
           FB_McAxisCtrl;
```

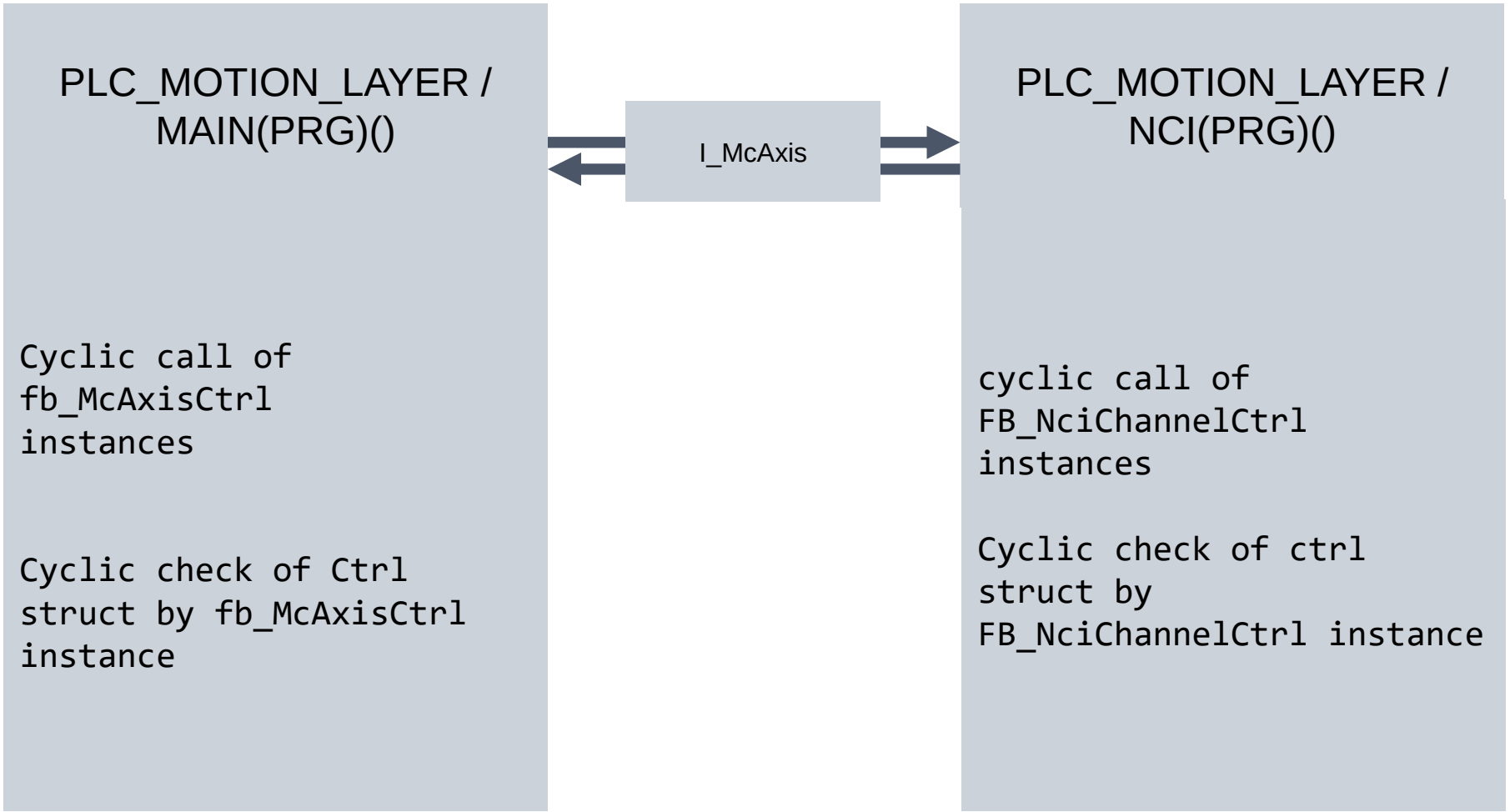
PLC_MOTION_LAYER / GVL_NCI

```
//--------------------------------------
// command and state structure
//--------------------------------------
stChannelCtrl  : ARRAY[1..MAX_NCI_CH] OF
                 ST_CTRL_NCI;

stChannelState : ARRAY[1..MAX_NCI_CH] OF
                 ST_STATE_NCI;

//--------------------------------------
// cyclic interface function block
//--------------------------------------
fbNciCtrl : ARRAY[1..MAX_NCI_CH] OF
            FB_NciChannelCtrl;
```

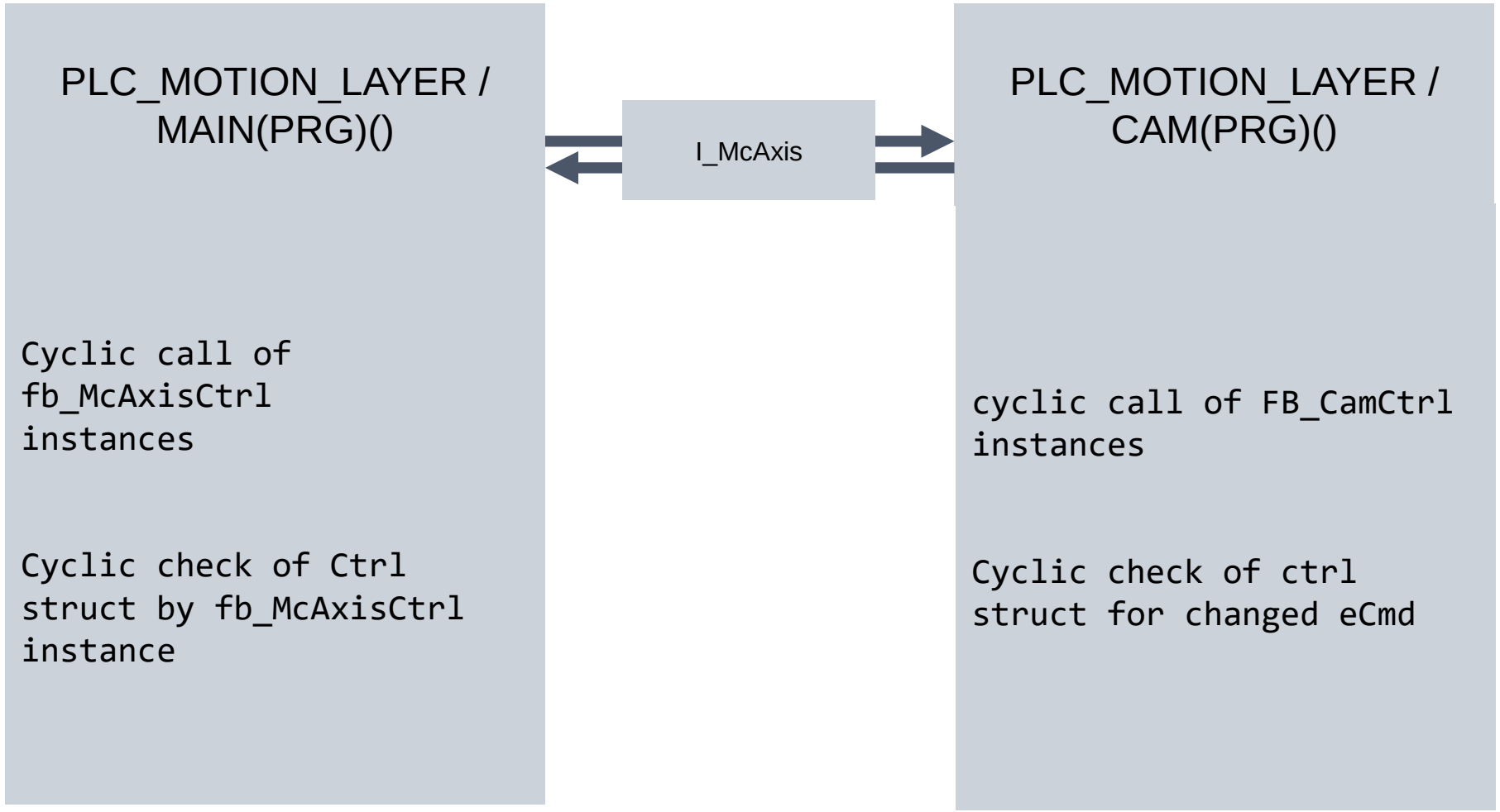
PLC_MOTION_LAYER / GVL_CAM

```
//--------------------------------------
// command and state structure
//--------------------------------------
Ctrl  : ARRAY[1..MAX_AXIS] OF
        ST_CAM_CTRL;

State : ARRAY[1..MAX_AXIS] OF
        ST_CAM_STATE;

//--------------------------------------
// cyclic interface function block
//--------------------------------------
Control  : ARRAY[1..MAX_AXIS] OF
           FB_CamCtrl;
```

Software Design:

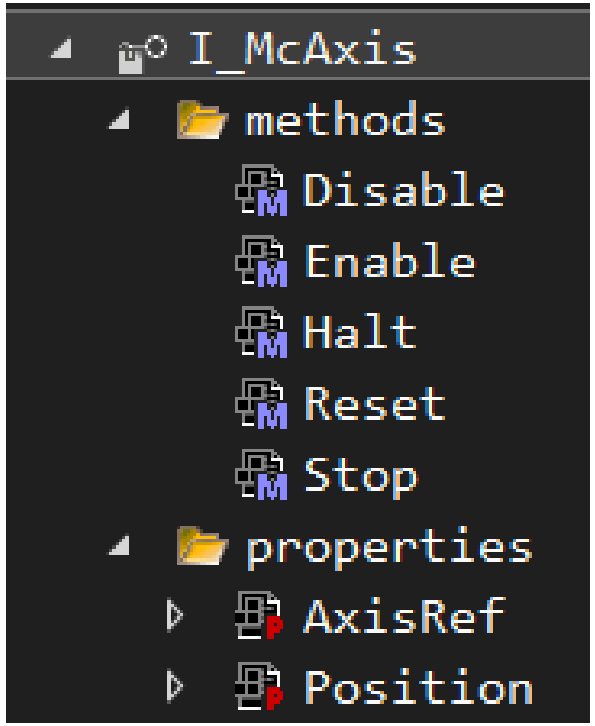


Software Design:



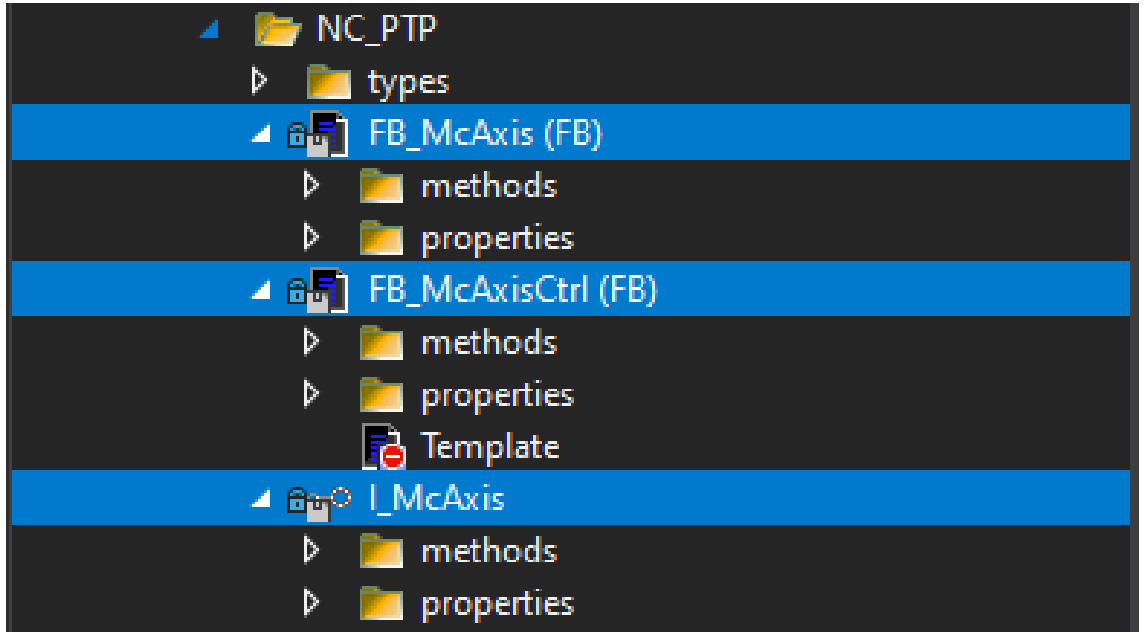
GVL_AXIS:

- I_McAxis: Interface for horizontal access between namespaces
  - Mandatory for using other namespaces
  - Provides essential methods
  - Provides reference to AXIS_REF
  - Direct access to AxisRef.NcToPlc.ActPos



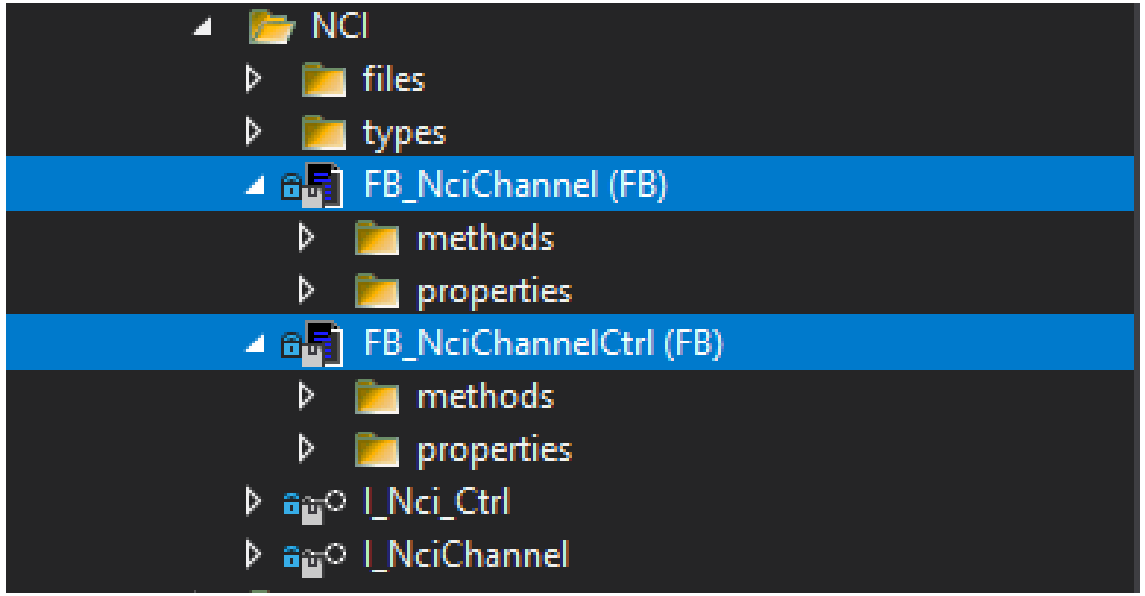
GVL_AXIS:

- McAxis: Point To Point axis
  - Base class **FB_McAxis** wraps Tc2_MC2 function blocks and implements interface
  - **FB_McAxisCtrl** extends base class with cyclic execution wrapper
  - **I_McAxis** is used in advanced motion features (NCI, CAM, XFC)



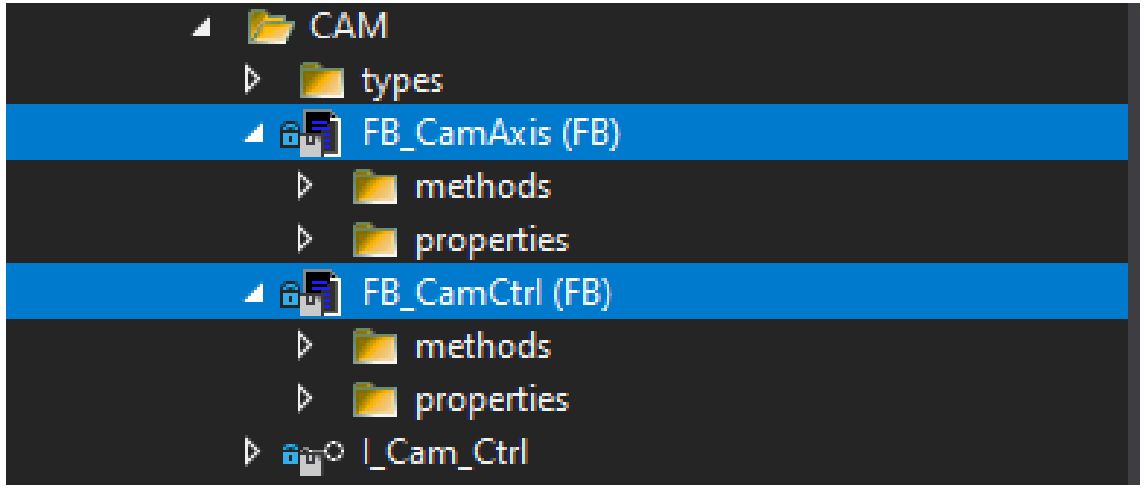
GVL_NCI:

- NCI Channel: XYZ interpolated
  - Base class **FB_NciChannel** wraps Tc2_NCI function blocks and implements interface
  - **FB_NciChannelCtrl** extends base class with cyclic execution wrapper and implements interface
  - **I_NciChannel** and **I_Nci_Ctrl** are only valid if compiler define is set before compiling project



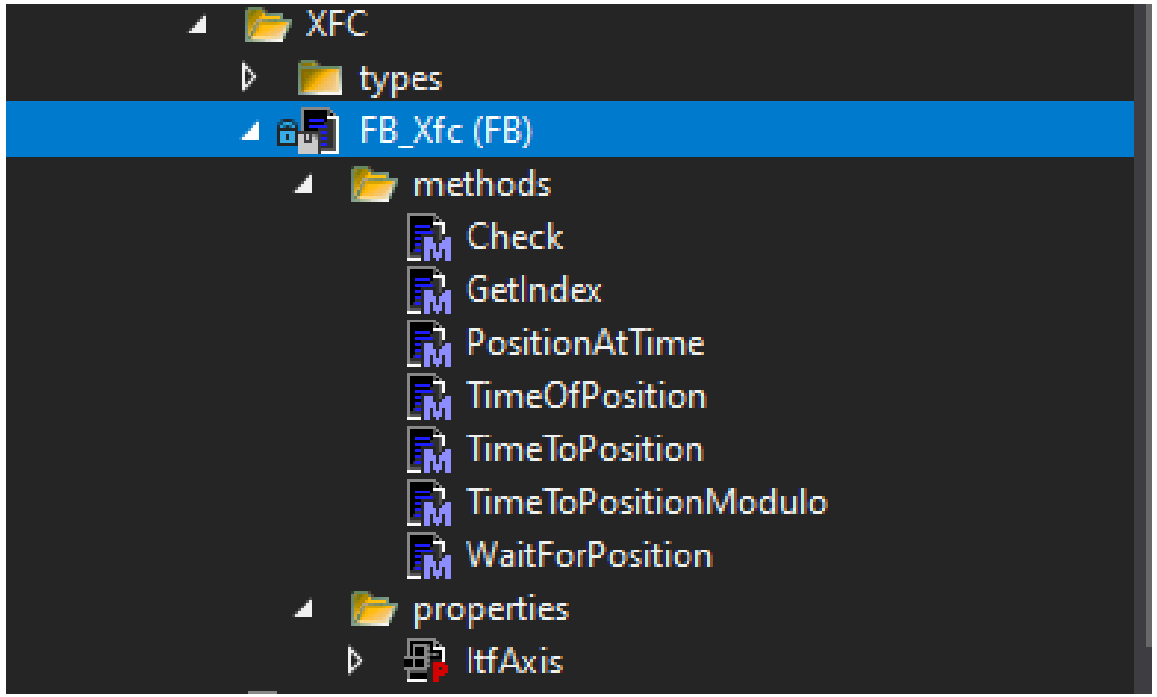
GVL_CAM:

- Camming:
  - Base class **FB_CamAxis** wraps Tc2_MC2_Camming function blocks and implements interface
  - **FB_CamCtrl** extends base class with cyclic execution wrapper and implements interface
  - **I_Cam_Ctrl** is only valid if compiler define is set before compiling project



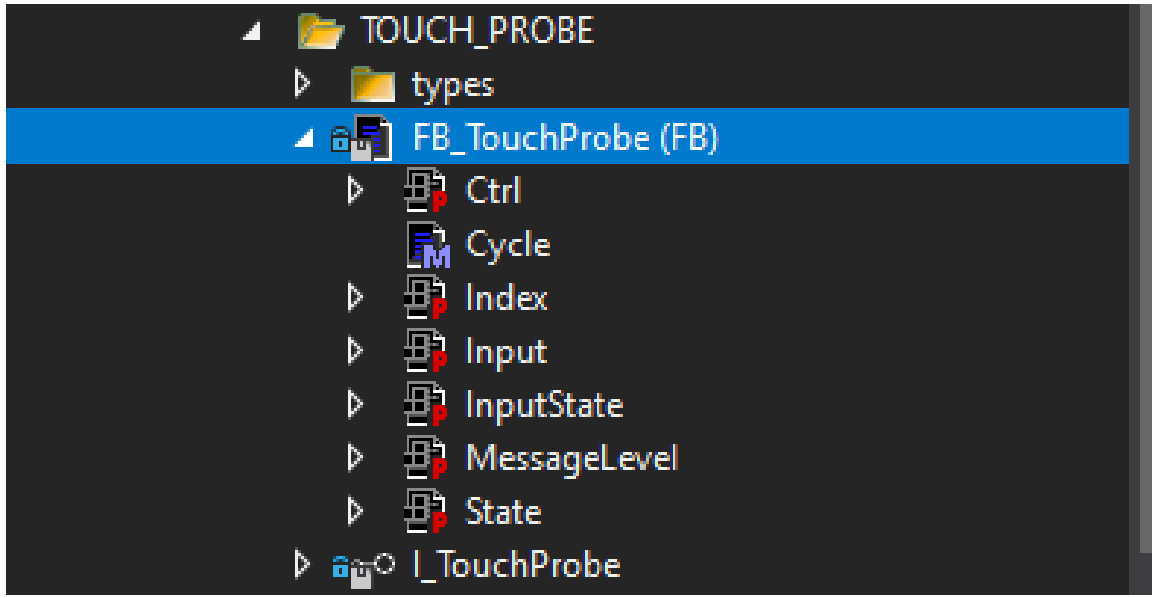
GVL_FUNCTIONS:

- XFC classes:
  - Base class **FB_Xfc** wraps Tc2_MC2_XFC function blocks for Distributed Clock position/time applications



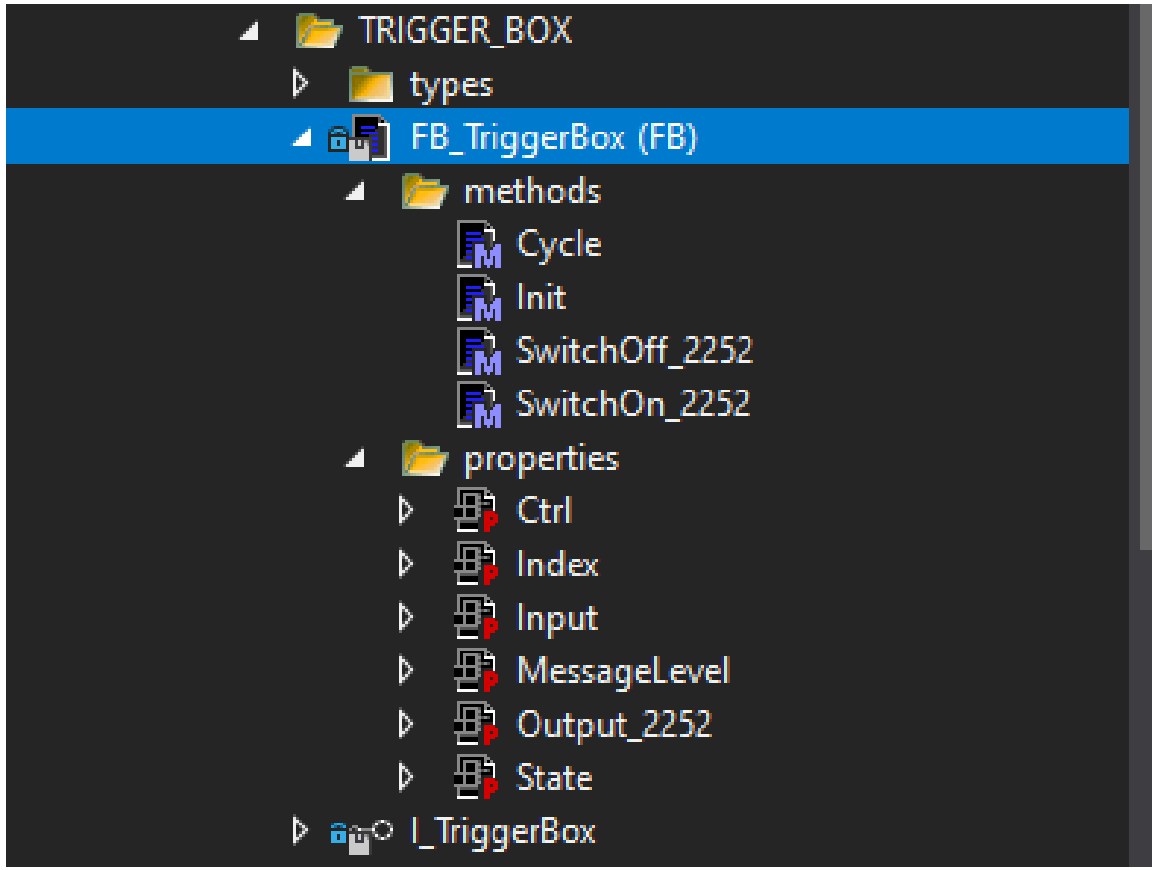
GVL_FUNCTIONS:

- Touch Probe:
  - **FB_TouchProbe** extends **FB_Xfc**
    - Cyclic execution with Ctrl/State pair
    - Must be connected to input device
  - **I_TouchProbe** is only valid if compiler define is set before compiling project



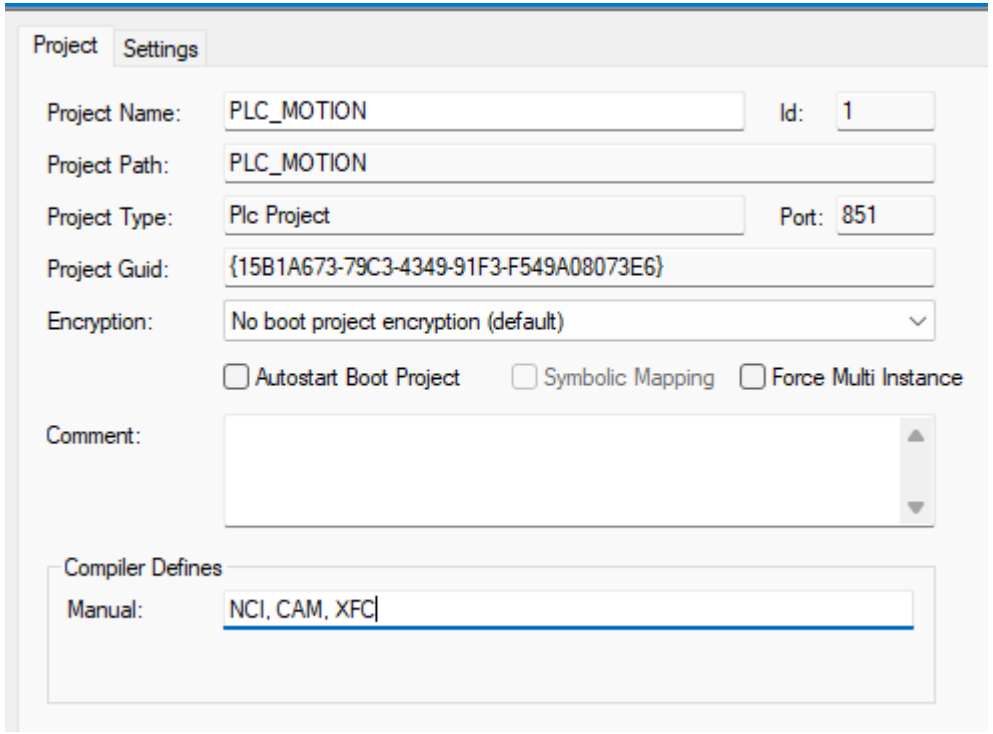
GVL_FUNCTIONS:

- Trigger Box:
  - **FB_TriggerBox** extends **FB_Xfc**
    - Cyclic execution with Ctrl/State pair
    - Must be connected to input/output device
  - **I_TriggerBox** is only valid if compiler define is set before compiling project



Compiler defines / pragmas:

- **BSD**
  - TC-BSD system specific variables are set
- **NCI**
  - NciChannel cyclic interfaces are used
- **CAM**
  - Camming cyclic interfaces are used
- **XFC**
  - TouchProbe and TriggerBox interfaces are used



Compiler defines / pragmas:

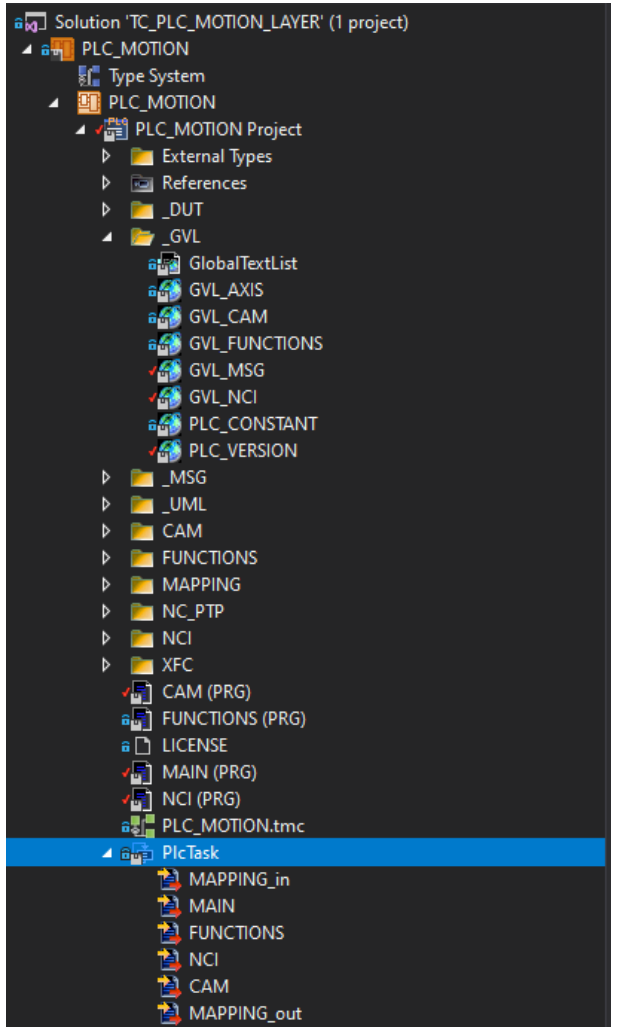
- **AXIS_MAP**
  - MAPPING_in.AxisCtrl is copied onto GVL_AXIS.Ctrl and data structures
  - GVL_AXIS.State is copied onto MAPPING_out.AxisState and info structures

- **CAM_MAP**
  - MAPPING_in.CamControl is copied onto GVL_CAM.Ctrl
  - GVL_CAM.State is copied onto MAPPING_out.CamState and info structures

- **TRIGGER_MAP**
  - TouchProbe and TriggerBox interfaces are used

Cyclic call tree:

- **MAPPING_in(PRG)**
  - Copies mapping data onto input structures
- **MAIN(PRG)**
  - Cyclic call to FB_McAxisCtrl instances
- **FUNCTIONS(PRG)**
  - Cyclic call to TouchProbe and TriggerBox instances
- **NCI(PRG)**
  - Cyclic call to NCI channel instances
- **CAM(PRG)**
  - Cyclic call to FB_CamCtrl instances
- **MAPPING_out**
  - Copies state data onto mapping structures



Logging System:

- Implementation from top down
  - → function based with global timestamp added automatically
- Enumeration based with 4 categories and timestamp
  - → ocurrance in strict timestamp order
- Error Id is mirrored directly from called instance
  - → Infosys error numbers can be searched in case of diagnosis
- Optional text for additional information
- Specific logging switch to get more detailed information aside error numbers
- Automated write procedure to ascii formatted file

LICENSE

BECKHOFF

PLC_MOTION_LAYER project