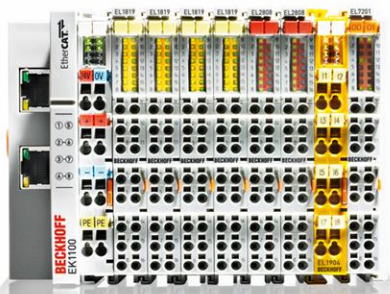
PLC_MOTION_LAYER
TwinCAT project

BECKHOFF



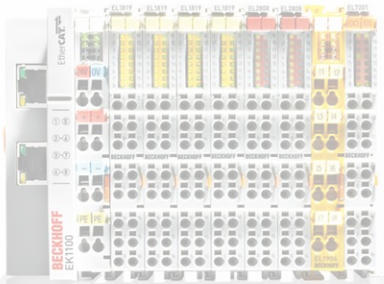
PLC_MOTION_LAYER

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:

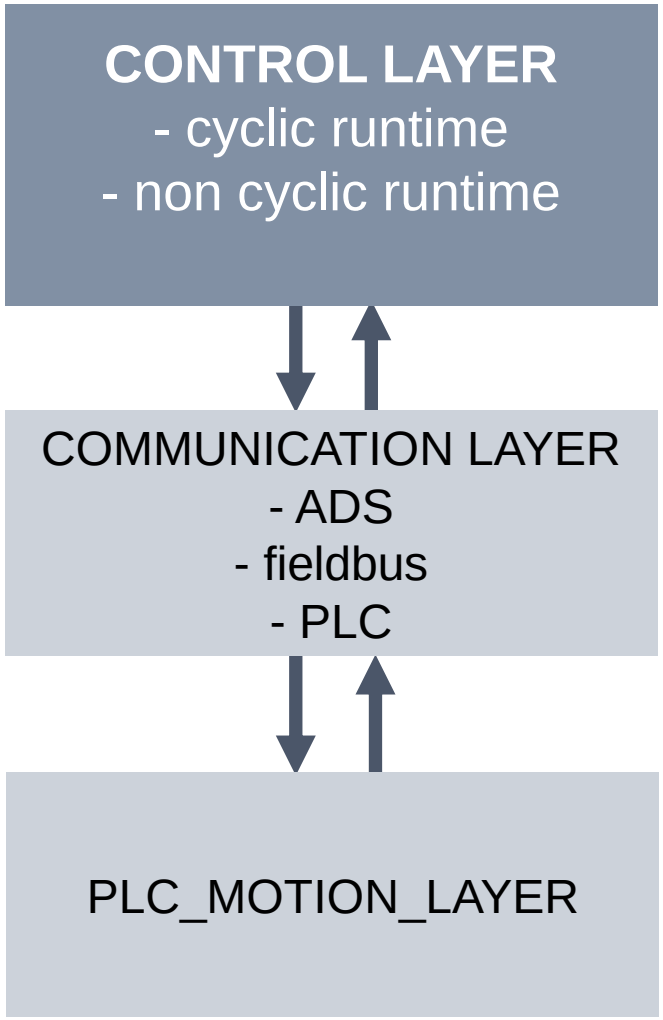
- **Use of TwinCAT supported/updated libraries**
  - Tc2_MC2
  - Tc2_MC2_Drive
  - Tc2_NC
  - Tc2_NCI
  - Tc2_PlcInterpolation
- **Open code base**
  - Migration to Tc3 MC in preparation
  - Customer/user specific changes possible
  - Access to code
  - Conversion to library possible by customer/user
- **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

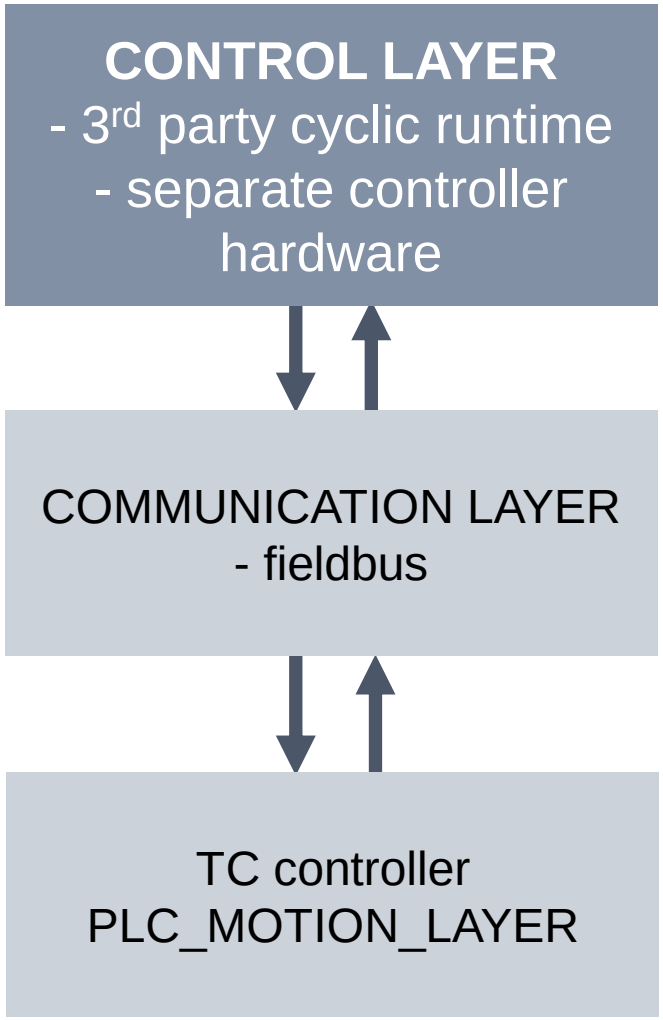


PLC_MOTION_LAYER

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

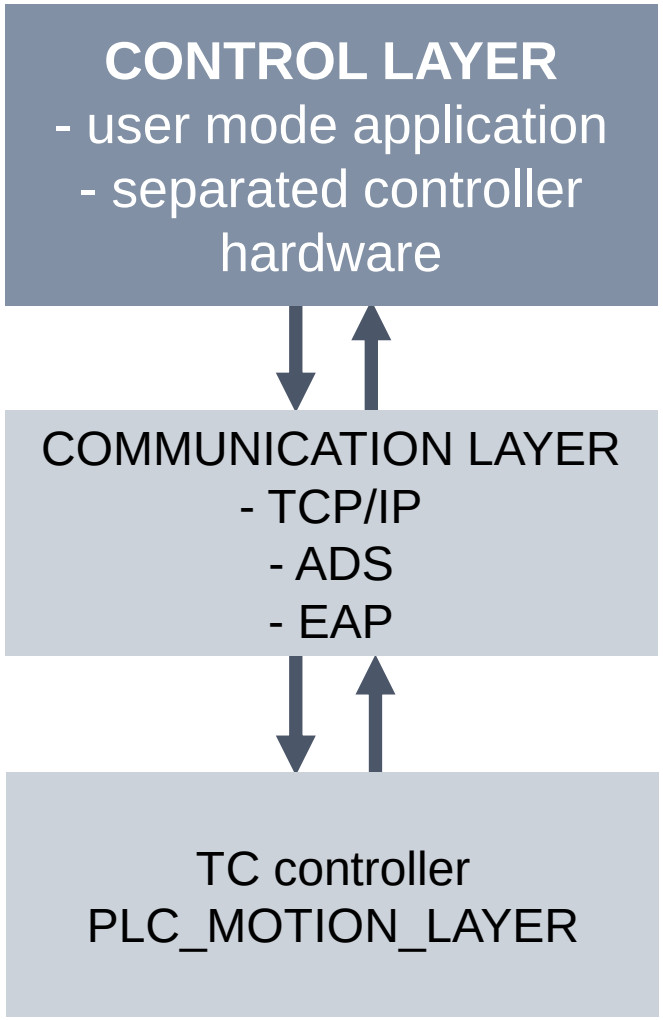


PLC_MOTION_LAYER

BECKHOFF

Use cases:

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

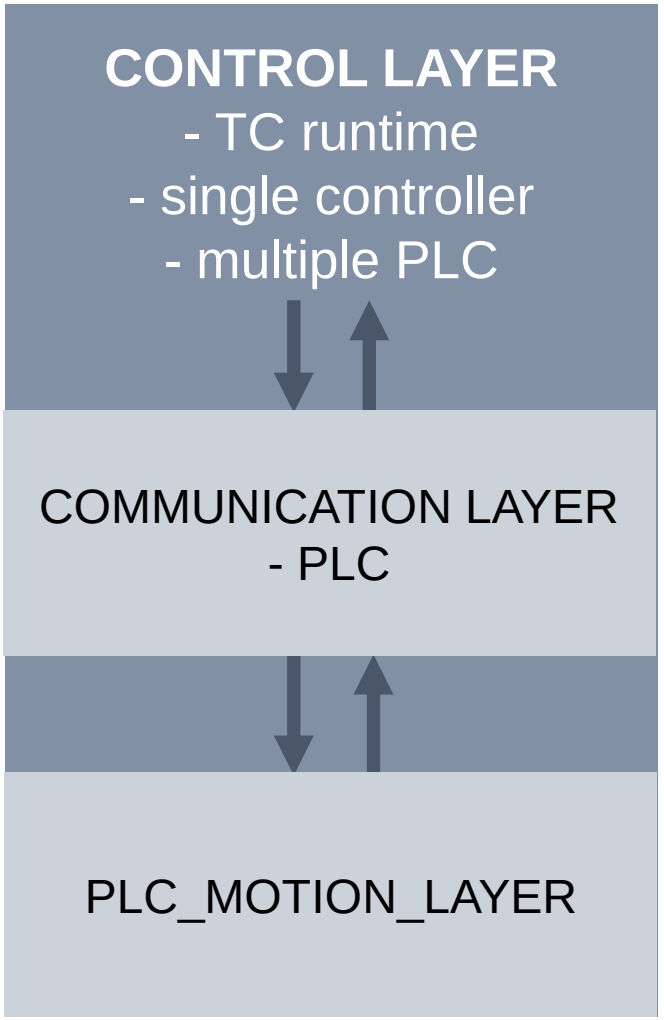


PLC_MOTION_LAYER

BECKHOFF

Use cases:

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

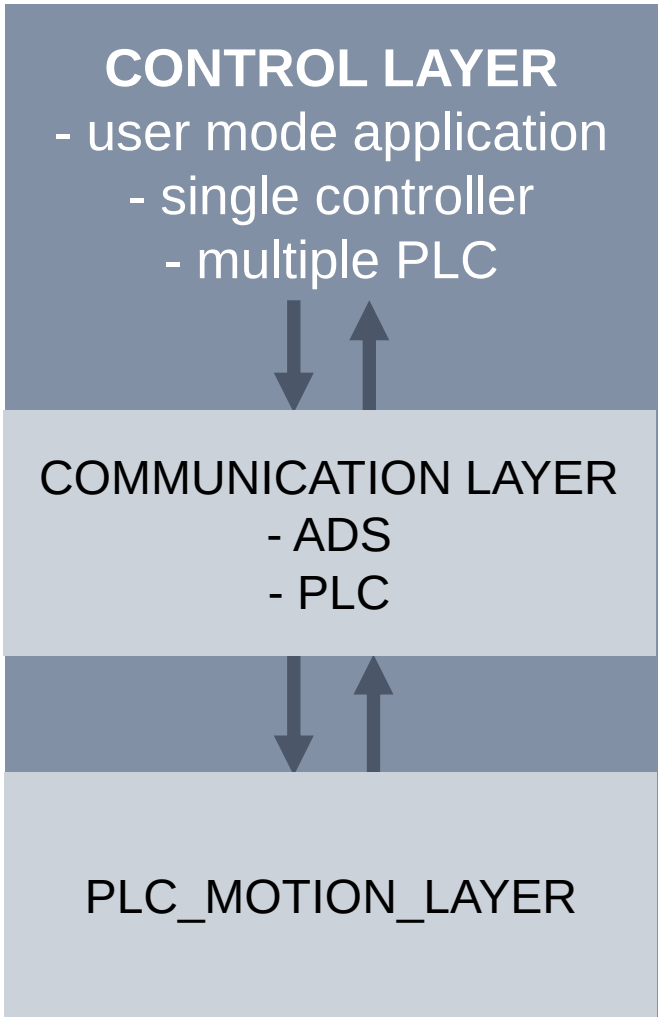


PLC_MOTION_LAYER

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
- OnChange detection for new commands
  - Cyclic check whether the command has changed
- State always carries offset about progress of command (busy, error, done)
- Instance FBs are called within states

Software Design:

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

- Cross communication via interfaces

- 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:

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;
```

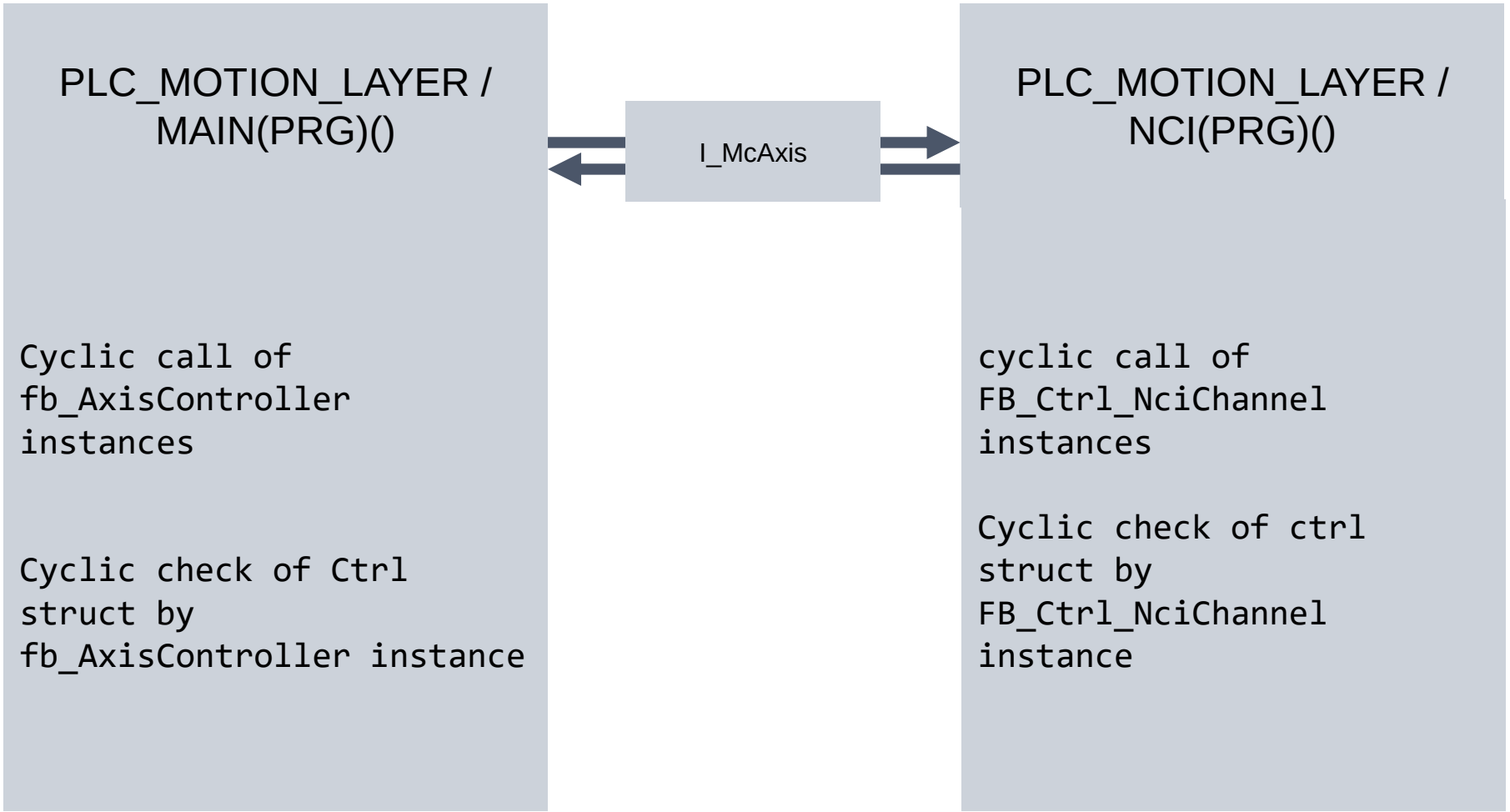
PLC_MOTION_LAYER / GVL_NCI

```
//--------------------------------------
// 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;
```

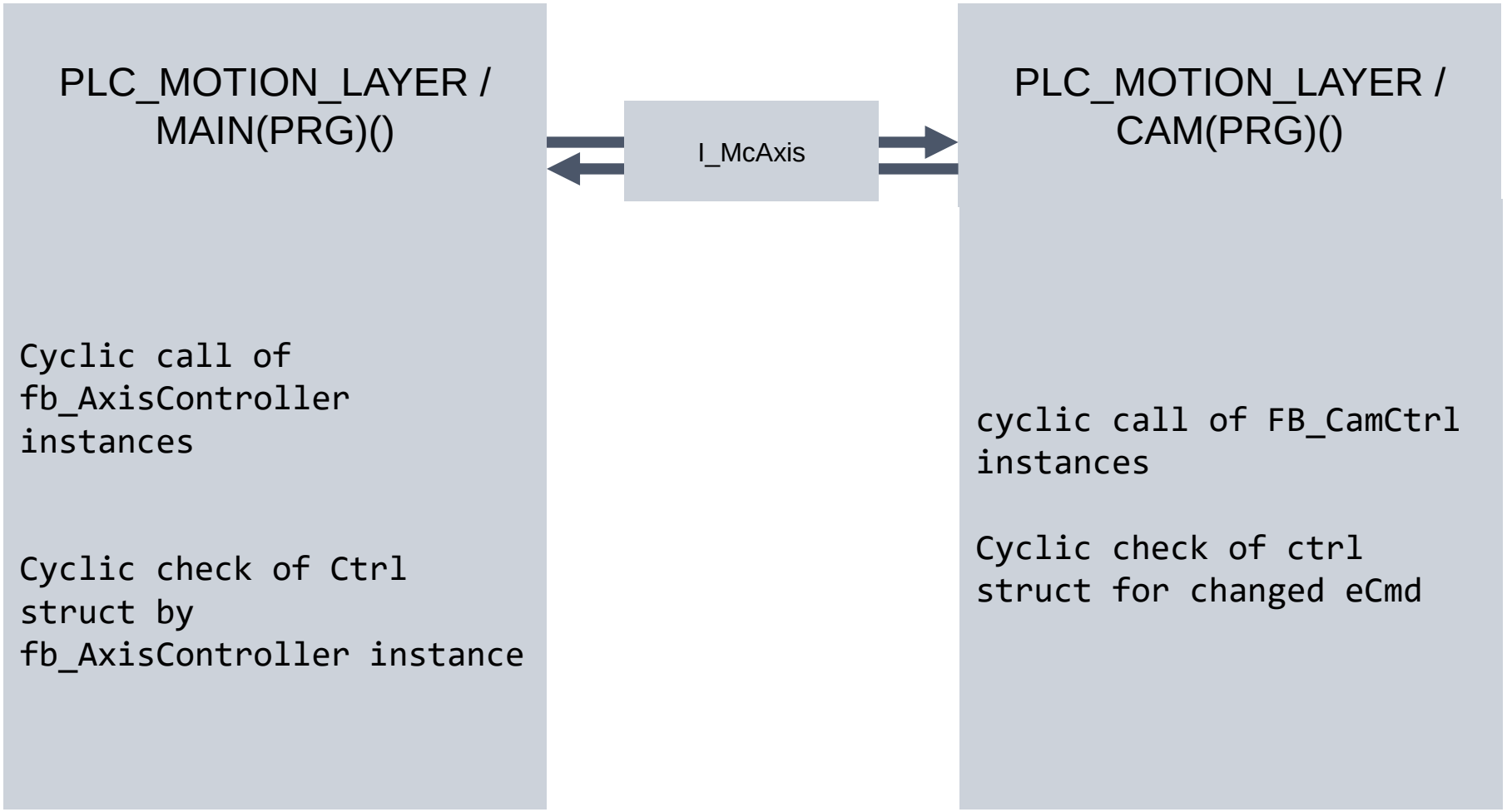
PLC_MOTION_LAYER / GVL_CAM

```
//--------------------------------------
// 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:

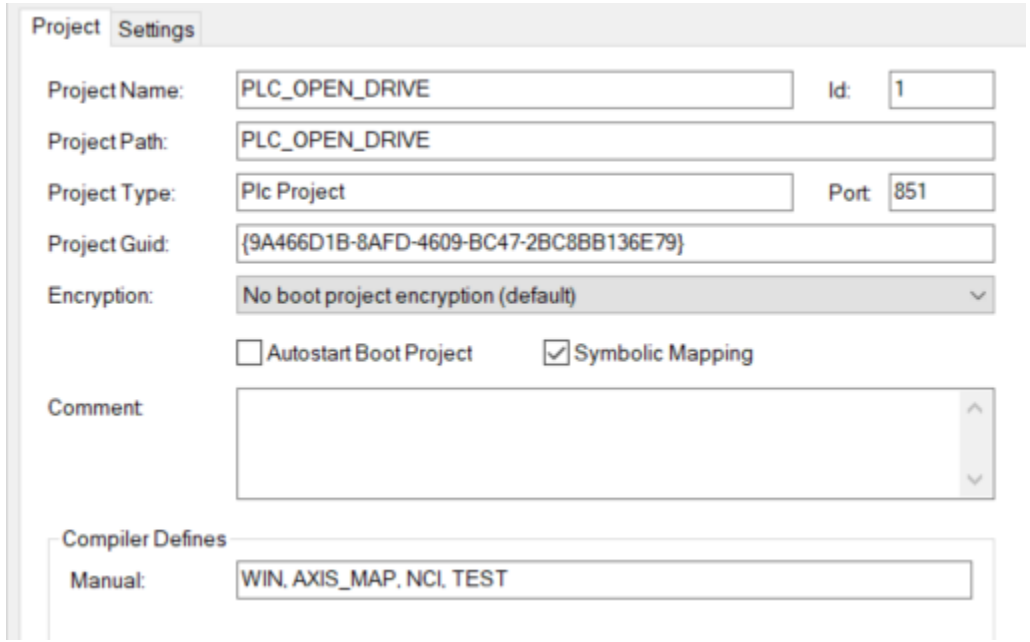


Software Design:



Compiler defines / pragmas:

- use of NCI and/or CAMMING
- operating specific values (Windows / TC-BSD)
- automated link to mapping



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