XTS TRANSPORT LAYER – a station based approach

BECKHOFF

XTS TRANSPORT LAYER

Daniel Hauer (HAUD)

BECKHOFF

1. Introduction
2. Requirements
   - XtsTransport (main control)
   - Xpu (XTS Processing Unit)
   - CaGroup (Collision Avoidance)
   - Mover (MC and CA)
   - Station (process handshake)
3. Design
   - use with any cyclic runtime
   - use with non cyclic software
4. Examples
5. License

1. Introduction

- This project collection is intended to convey the idea of a stand alone XTS transport layer to use in heterogeneous environments / applications.

- One main idea is that for every process you require, a corresponding position on the XTS exists.
  - The station

- Another fundamental principle is that before anybody does anything, they must know what to do and that everybody involved in transport gets a list with atomic RW access.

  - The **sending** of movers runs parallel to **sending** information
    - This rids us from stations needing to talk to each other

1. Introduction

- A transport layer shall a data driven way to manipulate a stations behaviour
- A transport layer shall work a combination of discrete processes and continuous processes
- A transport layer shall have an interface for guiding a mover through a process station
- A transport layer shall have an interface to manipulate a mover
- A transport layer shall have an interface for setting-up or clearing the CollisionAvoidance Group
- A transport layer shall provide higher level layers for grouping of stations and coding of transport logic
- A transport layer shall enable use of all, or some layers provided.

1. Introduction

- In combination with the Collision Avoidance library sending movers does not require extra monitoring of the movement. The Collision Avoidance Group is controlling the gap between movers. The gap is an input parameter of the CA motion function blocks.

- Can be used for a station based approach, in which a station class is available for interaction with your process control.

- Can be used for a mover based approach, your process control has a direct connection to every mover.

- Can be used as a combination of station based and mover based approach.

- The use of predefined datafields also enables control of XTS TRANSPORT LAYER via fieldbus or network.

1. Introduction

- The XTS transport system enables a flexible product transport for various use cases. This transport layer deliberately tackles a specific subset of the flexibility of the XTS system.

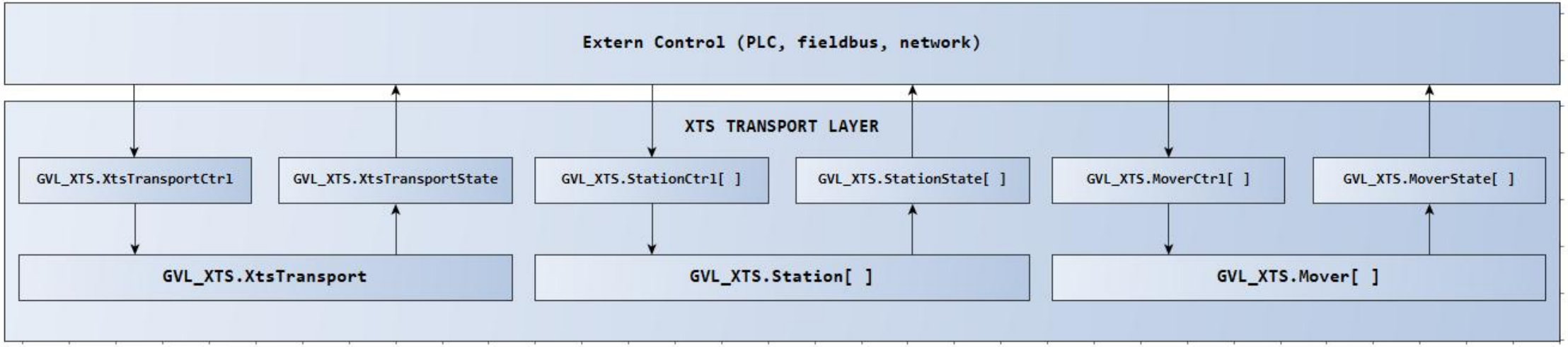
- This subset is working on:
  - One closed loop
  - One Track
  - Constant Mover count

- As all movers bound to the same rail:
  - No passing at all times
  - A station table seems like a good choice

- This transport layer will focus on enabling high throughput and high flexibility process flow
  - Layered approach to offer you granular BlackBox building.

1. Introduction

- designed for use with extern cyclic or non-cyclic flow control (PLC, EtherCAT, **any** network)
  - Ctrl / State datafields for extern to access
- station based approach and individual manipulation of mover
- handshake in station with extern process flow (ST_STATION_CTRL / ST_STATION_STATE)
- individual cyclic mover interface with given set of movement functionalities (ST_MOVER_CTRL / ST_MOVER_STATE)



2. Requirements

BECKHOFF

- TransportUnit
  - Access to CA group function blocks (interface pointer)
  - Access to Stations (interface pointer)
  - Access to Movers (interface pointer)
  - Commands for getting all members to defined state
  - Cyclic interface for access from extern control
    - Ctrl (write): command
    - State (read): response to command
      - information from Xpu
      - Information from CA Group

2. Requirements

BECKHOFF

- Xpu (XTS Processing Unit)
  - Check Init Parameter
  - Check Online Parameter
  - Get Module Info Data
  - Connect TcCOM Objects to instances from XTS_Utility.lib function blocks
  - Cyclic plausibility checks
    - Mover ID detection after init
  - Cyclic interface for access from main control
    - Ctrl (write): command
    - State (read): response to command
    - Info (read): details from cyclic checks

2. Requirements

BECKHOFF

- CaGroup
  - Access to group function blocks
  - Access to movers for group commands
  - Get Group Info Data
  - Implements interface for use in
    - TransportUnit

2. Requirements

BECKHOFF

- Mover
  - Access to MC function blocks
  - Access to CA function blocks
  - Cyclic interface for access from extern control
    - Ctrl (write): command
    - Data (write): command parameter
    - State (read): response to command
  - Implements Interface pointer for access from:
    - TransportUnit
    - Station
    - CaGroup

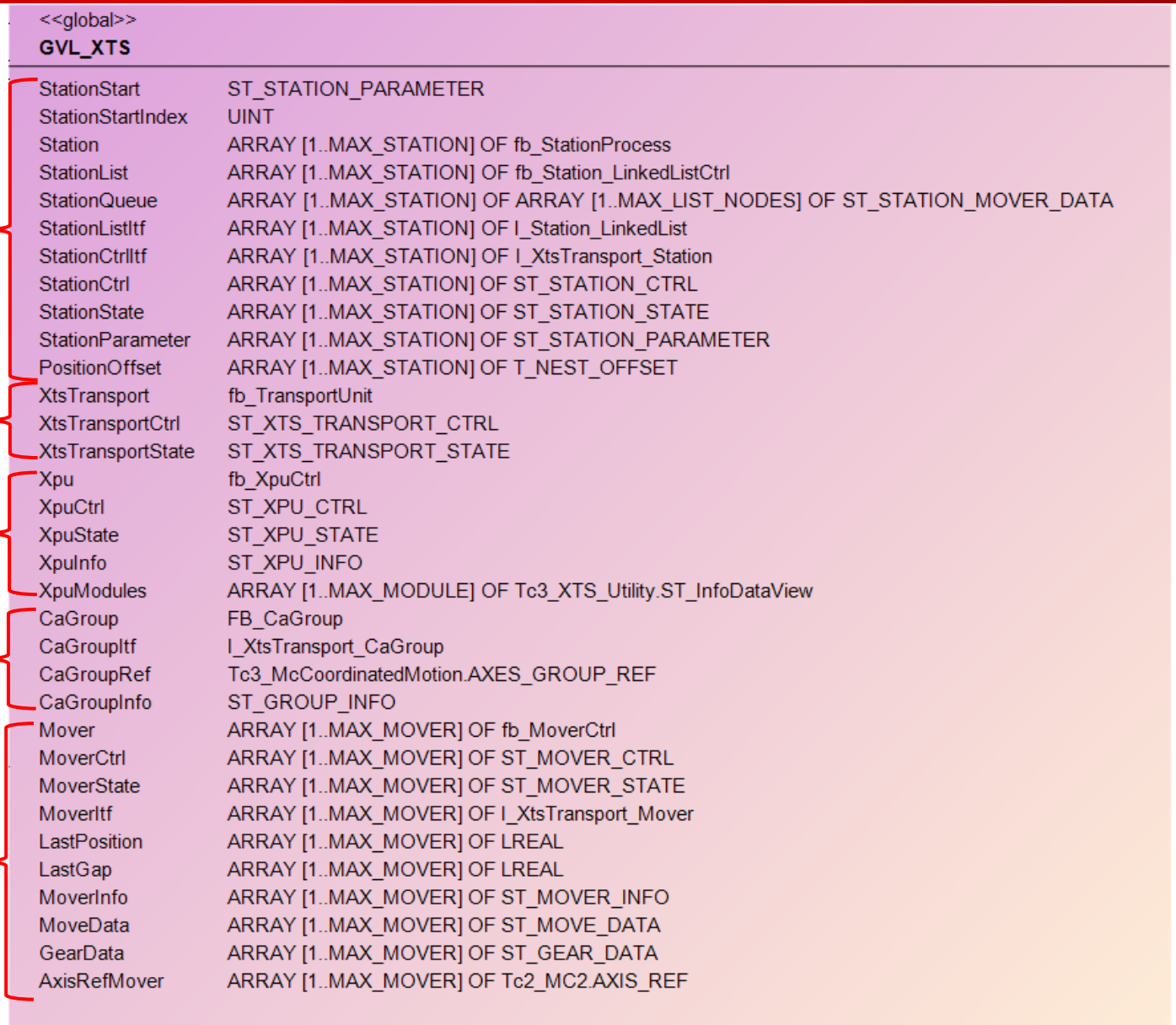
2. Requirements

BECKHOFF

- Station
  - Handshake mover transport with extern control
  - Close observation of movements in station with feedback to extern control
  - Linked List for movers in queue for infeed into station
  - Access to Linked List of target station for outfeed of mover
  - Cyclic interface for access from extern control
    - Ctrl (write): command and parameter
    - State (read): response to command and information about mover and queue
  - requires interface pointer to MC functionblocks

BECKHOFF

- Namespace GVL_XTS
  - **Station**
    - Handshake with Process for mover transport
  - **XtsTransport**
    - Main command interface to extern control
  - **Xpu**
    - Access to TcCOM Objects
    - Cyclic plausibility checks
  - **CaGroup**
    - Access to CA library
  - **Mover**
    - Access to MC and CA library



BECKHOFF

- **Yes, …but why?**
  - The ‚WHYs' for the station explains the rest of the design
    - I focus on stations first since they are the center of attention when production runs
  - The goal is to build maintainability into the transport layer
    (e.g. in case MC3 becomes available in the stable feed of the package manager)
  - This transport layer is intended to be flexible in complexity and application
    - This is why you'll discover ABSTRACT classes
  - OOP (… Yeah you know me!) **it makes things so much easier.**
    - Please try to overcome the reflex of it ‚being too much' for PLC programming
    - it is not 1993 anymore, OOP is around for 20 years now.
    - It is the reason this collection of projects is so easy.

3. Design

BECKHOFF

- Yes, …but why?

Fb / Class	Concern	Solution
Station	Manipulation of one mover	**I_Transport_Mover[nMoverDetected]**
	Handshake process-flow with external entity	Ctrl / State pair
	Where does the mover come from? Who cares, I'll just take the first	Use of Interface of LinkedList and struct datafields for tickets: **I_Station_LinkedList**
	Access to ST_STATION_PARAMETER of all stations	Pointer/Reference to table of stations
	Who am I?	MAIN injects loop counter: GVL_XTS.Station[nStation].**StationId** := nStation;
	Where am I?	ST_STATION_PARAMETER[StationId]
	Where am I precisely?	Pointer to static offset datafields[MAX_STATION, MAX_MOVER, MAX_NESTS]

3. Design

BECKHOFF

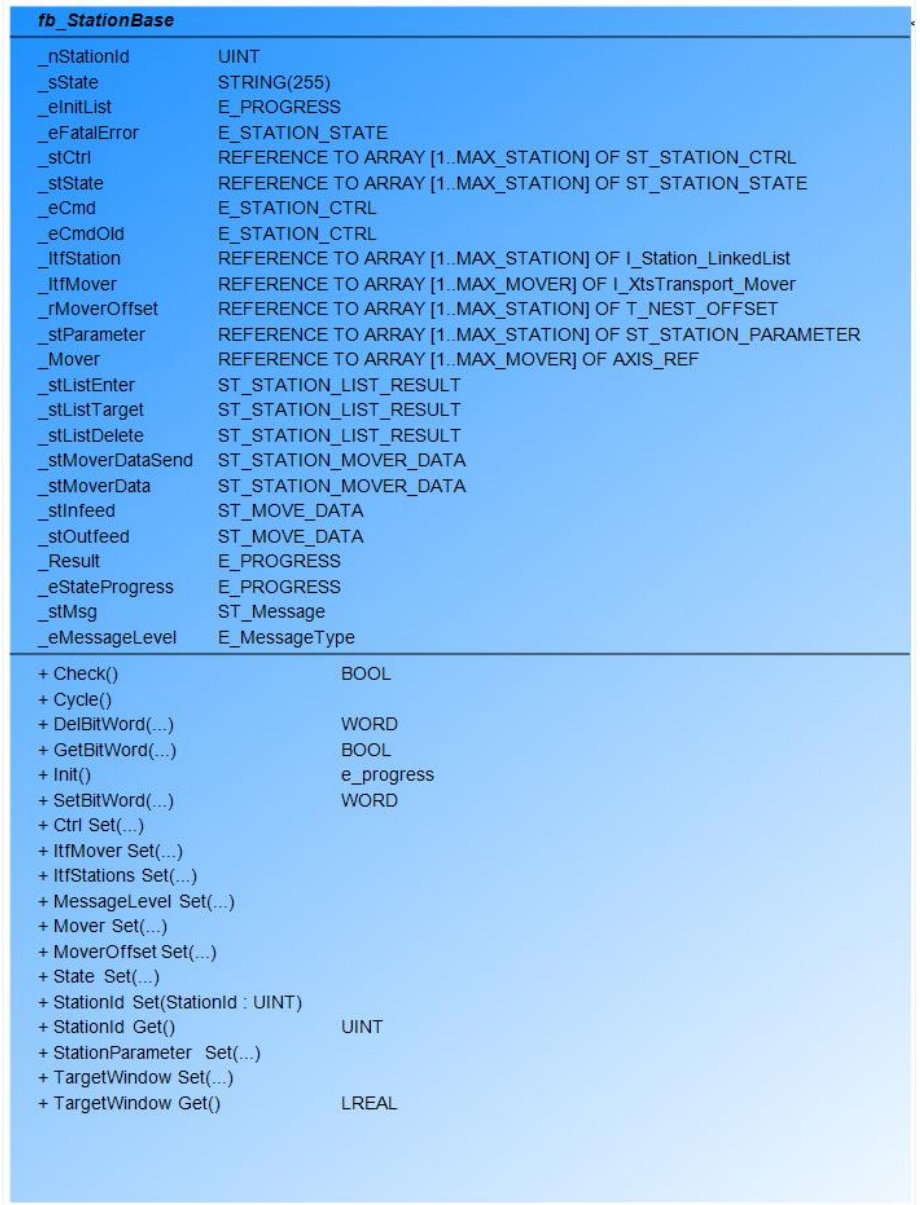
- Namespace XTS_Parameter
  - Project constants must match configuration:
  - Copied from ParameterList in: Tc3_XTS_Utility.TcIoXtsEnvironmentParameterList

```
MAX_MOVER           : UINT    := TO_UINT(Tc3_XTS_Utility.TcIoXtsEnvironmentParameterList.MaxXtsMoversPerXpu);
MAX_MODULE          : UINT    := TO_UINT(Tc3_XTS_Utility.TcIoXtsEnvironmentParameterList.MaxModulesPerPart);
MAX_PART            : UINT    := TO_UINT(Tc3_XTS_Utility.TcIoXtsEnvironmentParameterList.MaxXtsPartsPerXpu);
MAX_TRACK           : UINT    := TO_UINT(Tc3_XTS_Utility.TcIoXtsEnvironmentParameterList.MaxXtsTracksPerXpu);
MAX_GROUP           : UINT    := TO_UINT(Tc3_XTS_Utility.TcIoXtsEnvironmentParameterList.MaxXtsCaGroup);

MAX_INFO_SERVER     : UINT    := TO_UINT(Tc3_XTS_Utility.TcIoXtsEnvironmentParameterList.MaxXtsInfoServer);
MAX_INFO_STATION    : UINT    := TO_UINT(Tc3_XTS_Utility.TcIoXtsEnvironmentParameterList.MaxXtsInfoStation);
MAX_INFO_STOP_POS   : UINT    := TO_UINT(Tc3_XTS_Utility.TcIoXtsEnvironmentParameterList.MaxXtsStopPositionsPerStation);

MAX_PARAMETER_SET   : UINT    := TO_UINT(Tc3_XTS_Utility.TcIoXtsEnvironmentParameterList.MaxXtsParameterSet);

MAX_NCT_BASE_UNIT   : UINT    := TO_UINT(Tc3_XTS_Utility.TcIoXtsEnvironmentParameterList.MaxXtsNctBaseUnitInterfaces);

MAX_STATION         : UINT    := 2;

MAX_STATION_NEST    : UINT    := TO_UINT(SIZEOF(ST_STATION_STATE.nMask)*8);

MAX_LIST_NODES      : UINT    := MAX_MOVER;
MAX_MASTER          : UINT    := 1;
```

3. Design

BECKHOFF

- GVL_XTS.Station
  - **fb_StationBase**
    - Abstract base class
    - Allowing for user defined XtsStations
    - Provides datafields and properties
    - Cycle placeholder; override with your station logic when extending this base class



3. Design

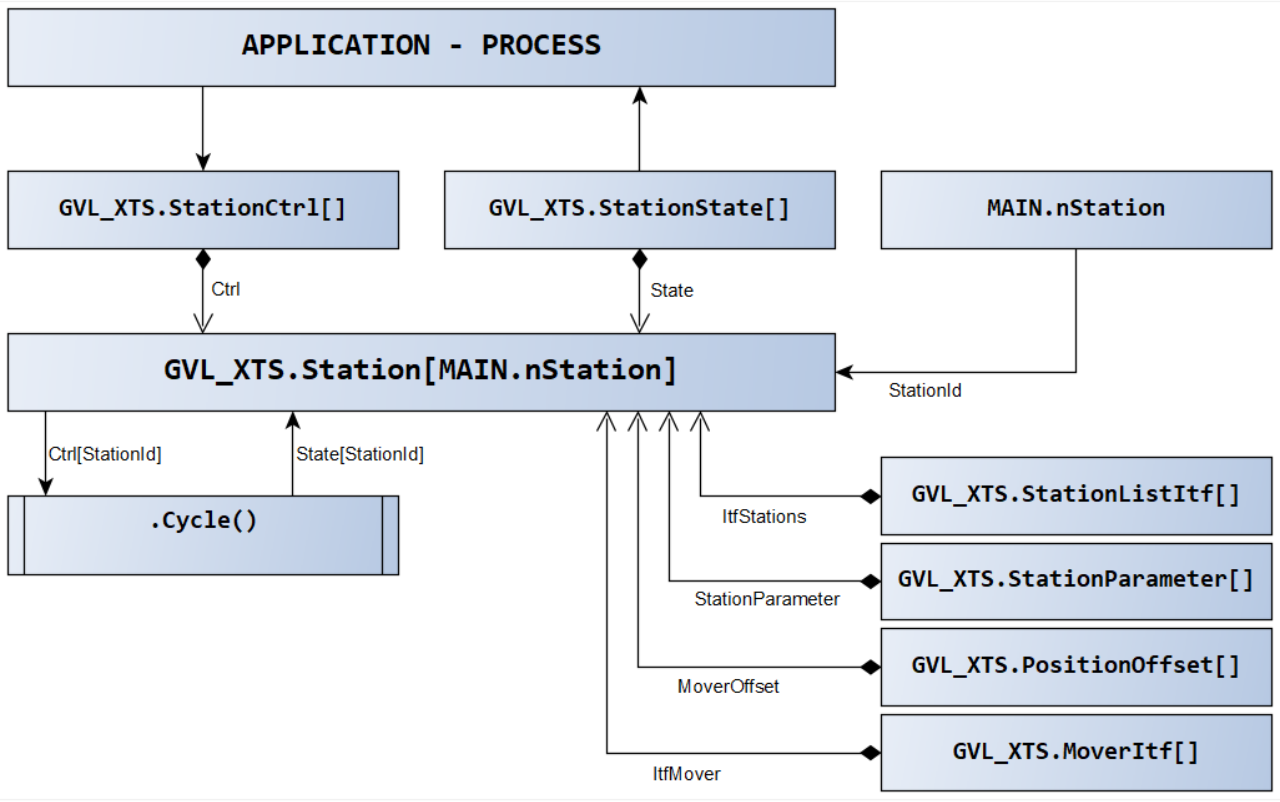
BECKHOFF

- GVL_XTS.Station
  - fb_StationProcess[].Cycle
    - State machine for handshaking with extern control (check example pdf in [doc] folder)
      - Init (clears everything in station)
      - Enable
      - Mover Enter
      - Stop Position(s)
      - Mover Out
      - Empty

- GVL_XTS.Station
  - fb_StationProcess[].Cycle
    - Control writes ticket for mover
      - MoverId
      - TargetStation
      - Mask
      - Offset

BECKHOFF

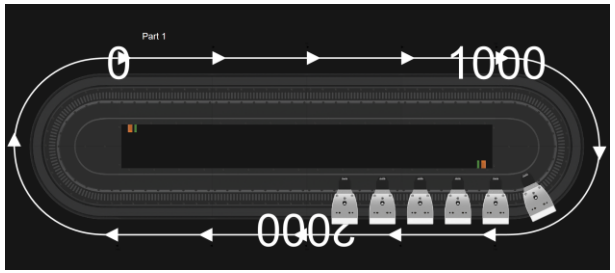
- GVL_XTS.Station
  - nStation index is passed as value from caller
  - Global datafields are passed as references (REF=) into fb_StationBase properties
    - Ctrl / State: handshakes
    - ItfStations: interface pointer to linked list methods for getting and setting of mover data
    - StationParameter: Coordinates and dynamic constraint of XtsStation
    - MoverOffset: correction values for every mover in every station with every nest (PosStop[])
    - ItfMover: interface pointer to CA movements



- **GVL_XTS.Station**
  - Planning requirements for use of fb_StationProcess

  - **One station: one mover**
    - If you require multiple movers for your process:
      - parallel → multiplication of the same task:
        - → fb_StationCollector and fb_ProcessCollector will help grouping the stations to a single Ctrl/State interface. (see example **XTS_DEMO_APPLICATION_108**)

  - **Station to Station: only forward**
    - We‘re on a closed loop (the defining part of the subset my focus lies on)



BECKHOFF

- **GVL_XTS.Station**
  - Planning requirements for use of fb_StationProcess

  - Put the Modulo turn anywhere, **BUT NOT** within PosWait, PosStop, ReleaseDistance of a station. The code does not support crossing the modulo turn within a station.

  - Since the project is designed for stations to send movers to a flexible target, with flexible nest positions, the control struct of a station you have to use, to forward those parameters. The MoverId is forwarded by the Station in possession of the mover.

    - **ST_STATION_CTRL.nMask:** commands the nest count and nest position of the mover in target station

    - **ST_STATION_CTRL.nTargetStation:** index of station in GVL_XTS.StationParameter[]

    - **ST_STATION_CTRL.rOffset:** optional dynamic inline offset (added to every nest)

BECKHOFF

- **GVL_XTS.Station**
  - Planning requirements for use of fb_StationProcess

  - The Use of **LinkedList** methods (AddTail, GetHead)
    requires thought about **when** the mover is entered into the target station.
    - → why? → The targets' list must stay sorted → see examples on next pages

    - fb_StationProcess.Cycle() is built in a way (see handshake flowcharts for this) that the movers' ticket is written via the list interface of the target station **after** having moved **rReleaseDistance** (a station parameter):

      - `The movement:→`

```
//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
// ckeck if mover has left the station
//  enter mover data into list of target station
//------------------------------------------------------------------------------
CASE _stState[_nStationId].eState
OF
  E_STATION_STATE.STATION_MOVER_RELEASE:
    _ItfMover[_stMoverData.nMoverId].MessageLevel := SEL(_bMatchMessageLevel, E_MessageType.eMessageError, _eMessageLevel);

    _Result                          := _ItfMover[_stMoverDataSend.nMoverId].SendToModuloPosCa(TRUE, _stOutfeed);

    IF (_Result = E_PROGRESS.PROGRESS_DONE)
    THEN
      // regular outfeed
      _stState[_nStationId].eState    := E_STATION_STATE.STATION_MOVER_WRITE_TARGET;

    ELSIF (_stCtrl[_nStationId].eCmd = E_STATION_CTRL.STATION_MOVER_GONE) // in case outfeed has to be terminated by ctrl
    THEN
      // in order to finish the outfeed, ticket data has to be written to target station
      _stState[_nStationId].eState    := E_STATION_STATE.STATION_MOVER_WRITE_TARGET;

    ELSIF (_Result = E_PROGRESS.PROGRESS_ERROR)
    THEN
      _stState[_nStationId].eState    := E_STATION_STATE.STATION_ERROR_OUTFEED_ERROR; // ctrl has to fix this, go tell ctrl
      _stMsg.eType                    := E_MessageType.eMessageError;
      LogState(_stMsg.eType);
    END_IF
END_CASE
```

BECKHOFF

- **GVL_XTS.Station**
  - Planning requirements for use of fb_StationProcess

  - The Use of **LinkedList** methods (AddTail, GetHead)
    requires thought about **when** the mover is entered into the target station.
    - → why? → The targets' list must stay sorted → see examples on next pages

    - fb_StationProcess.Cycle() is built in a way (see handshake flowcharts for this) that the movers' **ticket is written** via the list **interface** of the target station after having moved rReleaseDistance (a station parameter):

      - **`The ticket:→`**

```
//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
// write mover ticket in target station list
//
//------------------------------------------------------------------------------
CASE _stState[_nStationId].eState
OF
  E_STATION_STATE.STATION_MOVER_WRITE_TARGET:
    // add entry in target station list
    _stListTarget                   := _ItfStation[_stMoverDataSend.nTargetStation].AddTailValue(_stMoverDataSend);

    IF (_stListTarget.wState <> 0)
    THEN
    // adding mover in target list was NOT successful
      _stState[_nStationId].eState  := E_STATION_STATE.STATION_ERROR_LIST_ADD_TAIL_FAULT; // fatal error
      _stMsg.eType                  := E_MessageType.eMessageError;
      LogState(_stMsg.eType);

    ELSE
    // mover data successfully added to target station list
      IF (_eMessageLevel > E_MessageType.eMessageWarning)
      THEN
        _stMsg.eType                := E_MessageType.eMessageInfo;
        _stMsg.eSubdevice           := e_Subdevice.StationAddTail;
        _stMsg.iErrorNumber         := _stState[_nStationId].nMoverId;
        _stMsg.sText                := concat('TargetStation: ', TO_STRING(_stListTarget.stData.nTargetStation));
        f_MessageSet(_stMsg);
      END_IF
      _ItfMover[_stMoverDataSend.nMoverId].SendToModuloPosCa(FALSE, _stOutfeed);

      _stState[_nStationId].eState  := E_STATION_STATE.STATION_MOVER_GONE; // go tell ctrl that station is empty

      IF (_eMessageLevel = E_MessageType.eMessageVerbose)
      THEN
        _stMsg.eType                := _eMessageLevel;
        LogState(_stMsg.eType);
      END_IF
    END_IF
END_CASE
```

BECKHOFF

- **GVL_XTS.Station - Planning requirements for use of fb_StationProcess**
  - all coordinates are modulo values,
    - from station to station only forward
    - within station: movement by use of nest offset(PosStop[]) or use of **ST_MOVER_CTRL**.
    - Within station: all kind of MC functions
  - **IF** move backwards is required you have to make sure that there is room for it by setting ST_STATION_PARAMETER so the movement of the movers match your process requirements for each station
    - Check PosStop[]
    - Each PosStop[] is **relative** to PosWait

BECKHOFF

- GVL_XTS.Station (Example)

- **ST_STATION_PARAMETER**: parallel Xts stations for a process with common waiting position

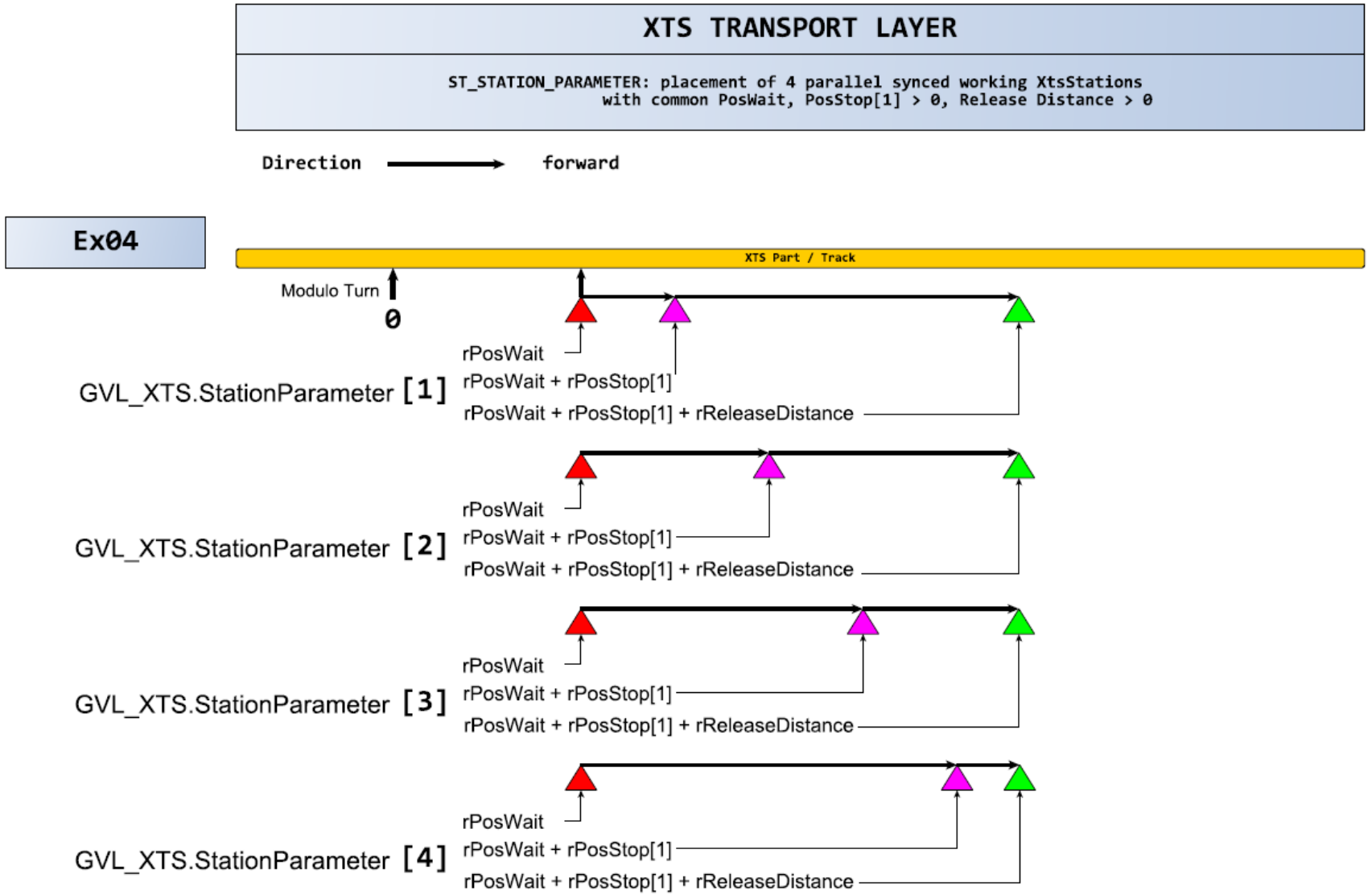
  - Process uses GVL_XTS.Station[1] to GVL_XTS.Station[4]
    - Define PosWait(Queue position)
      - **[1].rPosWait := 100**
      - **[2].rPosWait := 100**
      - **[3].rPosWait := 100**
      - **[4].rPosWait := 100**
    - Define how many rPosStop(nests) the stations may have (configured count)
      - **[1].nConfiguredStopCount := 1 (default)**
      - **[2].nConfiguredStopCount := 1**
      - **[3].nConfiguredStopCount := 1**
      - **[4].nConfiguredStopCount := 1**

  - Process uses GVL_XTS.Station[1] to GVL_XTS.Station[4]
    - Define the process position(s) relative to rPosWait
      - **[1].rPosStop[1] := 100**
      - **[2].rPosStop[1] := 200**
      - **[3].rPosStop[1] := 300**
      - **[4].rPosStop[1] := 400**
    - The ReleaseDistance of **the last station shall be shortest**, all other stations follow accordingly.
      - **[1].rReleaseDistance := 40**
      - **[2].rReleaseDistance := 30**
      - **[3].rReleaseDistance := 20**
      - **[4].rReleaseDistance := 10**

BECKHOFF

- GVL_XTS.Station (Example)

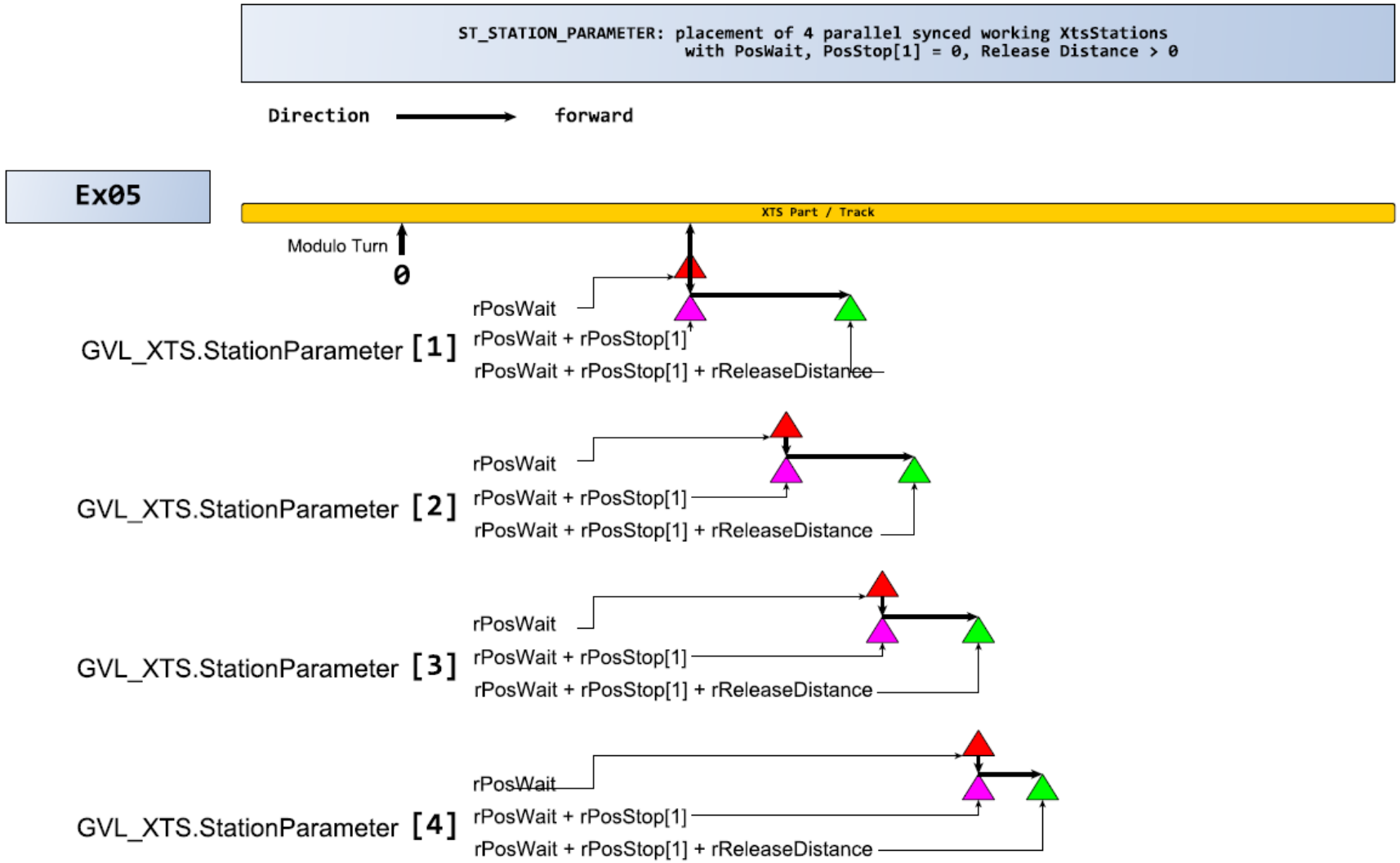
- **What it may look like**: parallel Xts stations for a process with common waiting position
  - drawings are not to scale, the possible configurations shall be illustrated
  - **Ex04**: This configuration is useful if you require the working area to be completely empty before pulling in movers. (move something other than your directly involved process mechanics out of the way of the mover etc.)



BECKHOFF

- GVL_XTS.Station (Example)

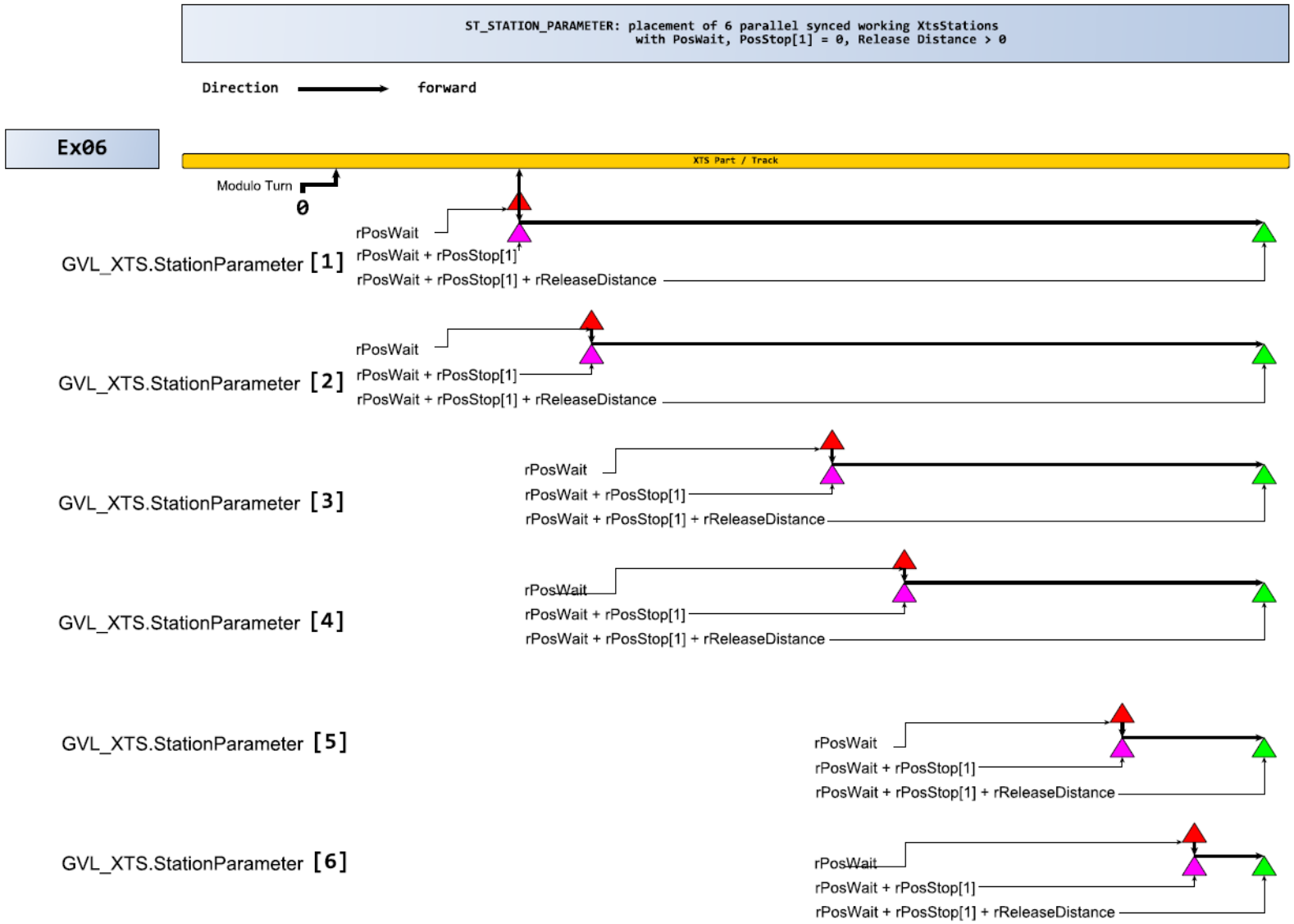
- **What it may look like**: parallel Xts stations for a process with common waiting position
  - drawings are not to scale, the possible configurations shall be illustrated
  - **Ex05**:
    This configuration is useful if you require high throughput with minimal stops of the mover. rPosWait[1-4] are already commanded by the sending station.



BECKHOFF

- GVL_XTS.Station (Example)

- **What it may look like**: parallel Xts stations for a process with common waiting position
  - drawings are not to scale, the possible configurations shall be illustrated
  - **Ex06**:
    This configuration is useful if you require high throughput and **irregular spacing** with minimal stops of the mover. rPosWait[1-4] are already commanded by the sending station.



BECKHOFF

- GVL_XTS.Station (Example)

- **using stations sparsely**:
  - In this case it is easiest to always handshake the stations and use the forwarding command if a station shall be skipped.
  - On Infeed state of mover, use: **E_STATION_CTRL. STATION_MOVER_SEND**.
    - There are different handshake ‘routes’ → see pdfs **GVL_XTS.Station_Example_*.pdf**

- **deactivating stations**:
  - Make sure the queue is empty before deactivating, since the waiting mover will hold up all other, in case of required deactivation while movers are in the queue:
    - handshake mover with **E_STATION_CTRL.STATION_MOVER_SEND** to new target station if mover in queue cannot be processed
    - Handshake regular infeed if mover in queue can still be processed.
    - Do not send any new mover to the station in question
      - If queue of station is empty: **E_STATION_CTRL.STATION_DISABLE**
      - preceding stations continue workflow with changed **ST_STATION_CTRL.nTargetStation**

BECKHOFF

- GVL_XTS.Station
  - Ctrl[nStation] : **ST_STATION_CTRL**
    - **eCmd (E_STATION_CTRL):**
      - enumeration for handshakes with State[nStation].eState **(E_STATION_STATE)**
    - **nMask (BYTE):**
      - bit mask to be used with multiple stop positions within a XtsStation.
        This mask tells the target station which PosStop[] (nest) has to be worked.
    - **nTargetStation (USINT):**
      - target to send mover to GVL_XTS.Station[nTargetStation].rPosWait
    - **rOffset (REAL):**
      - Optional offset for mover, used in target station in addition to static offset

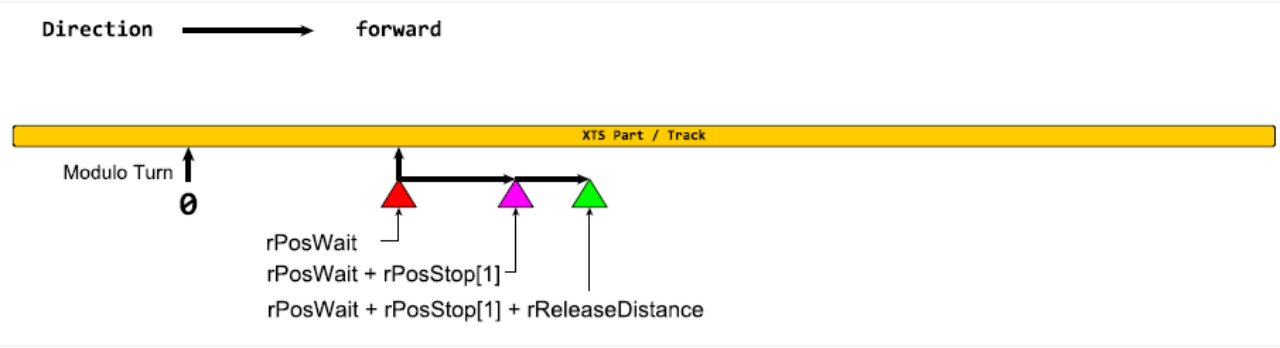
BECKHOFF

- GVL_XTS.Station
  - State[nStation] : **ST_STATION_STATE**
    - **eState (E_STATION_STATE):**
      - Enumeration for active station state, Ctrl has to react to
    - **nMask (BYTE):**
      - Bitmask for active PosStop[] (nest)
    - **nMoverId (USINT):**
      - Active mover index in station
    - **rMoverModPos (LREAL):**
      - Modulo position of active mover
    - **nQueue (USINT):**
      - Count of movers, which were sent to XtsStation

3. Design

BECKHOFF

- GVL_XTS.StationParameter
  - **eType**:
    - STATION_PROCESS, // default station with/without process
    - STATION_GEAR_IN_POS // GearInPos station for flying saw with MasterAxis
  - **sText :**
    - Description only
  - **rPosWait :**
    - start of station, a sending station is using this value to send mover to
  - **rReleaseDistance :**
    - distance mover has to travel (from ActPos) in order for station to go back to mover detection
  - **rGap :**
    - Active gap on infeed and outfeed of station
  - **rVelo :**
    - Active velocity on infeed and outfeed of station
  - **rAccDec :**
    - Active dyn constraint
  - **rJerk :**
    - Active dyn constraint
  - **nConfiguredStopCount :**
    - Count of PosStop (nests) a mover may has to stop at in XtsStation
  - **rPosStop[] :**
    - Relative to rPosWait

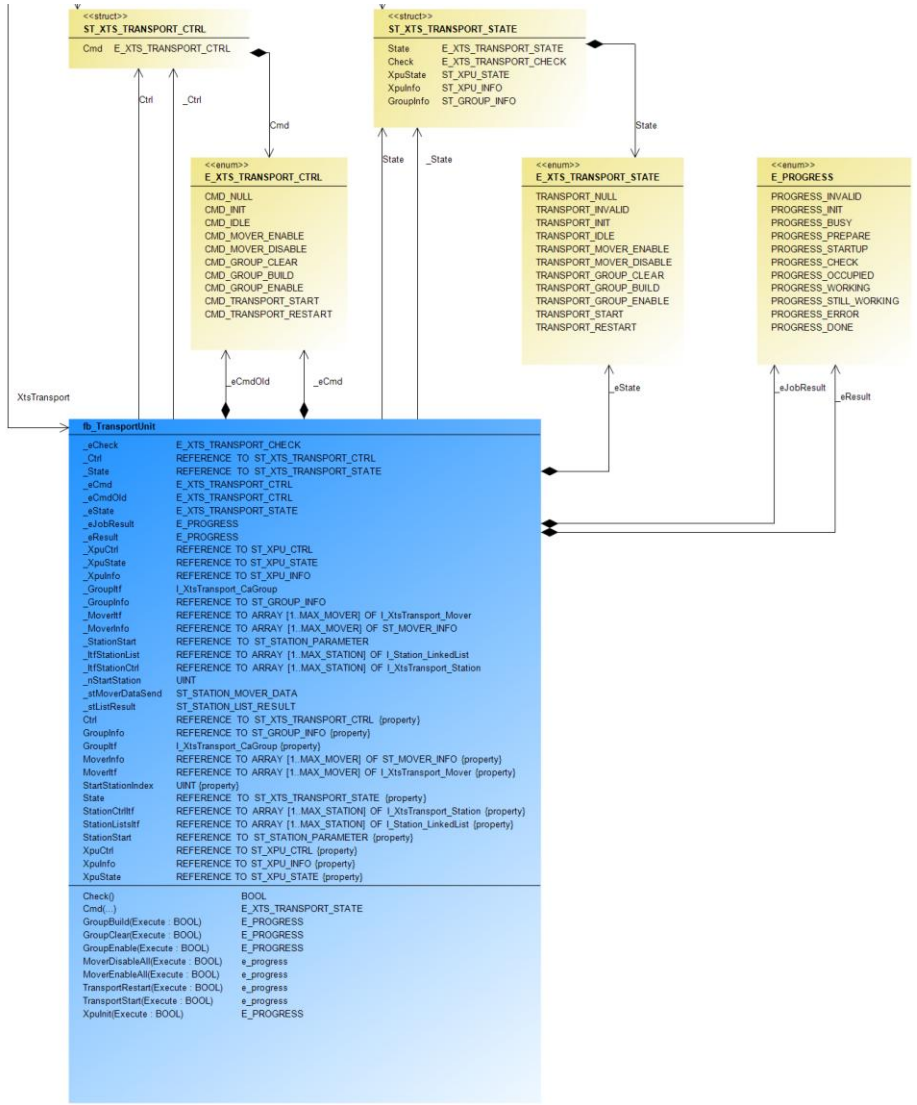


3. Design

BECKHOFF

- TransportUnit
  - Fb_TransportUnit():
    - Top level control of XtsTransport
    - Cycle check for change of command:
      - **E_XTS_TRANSPORT_CTRL:**

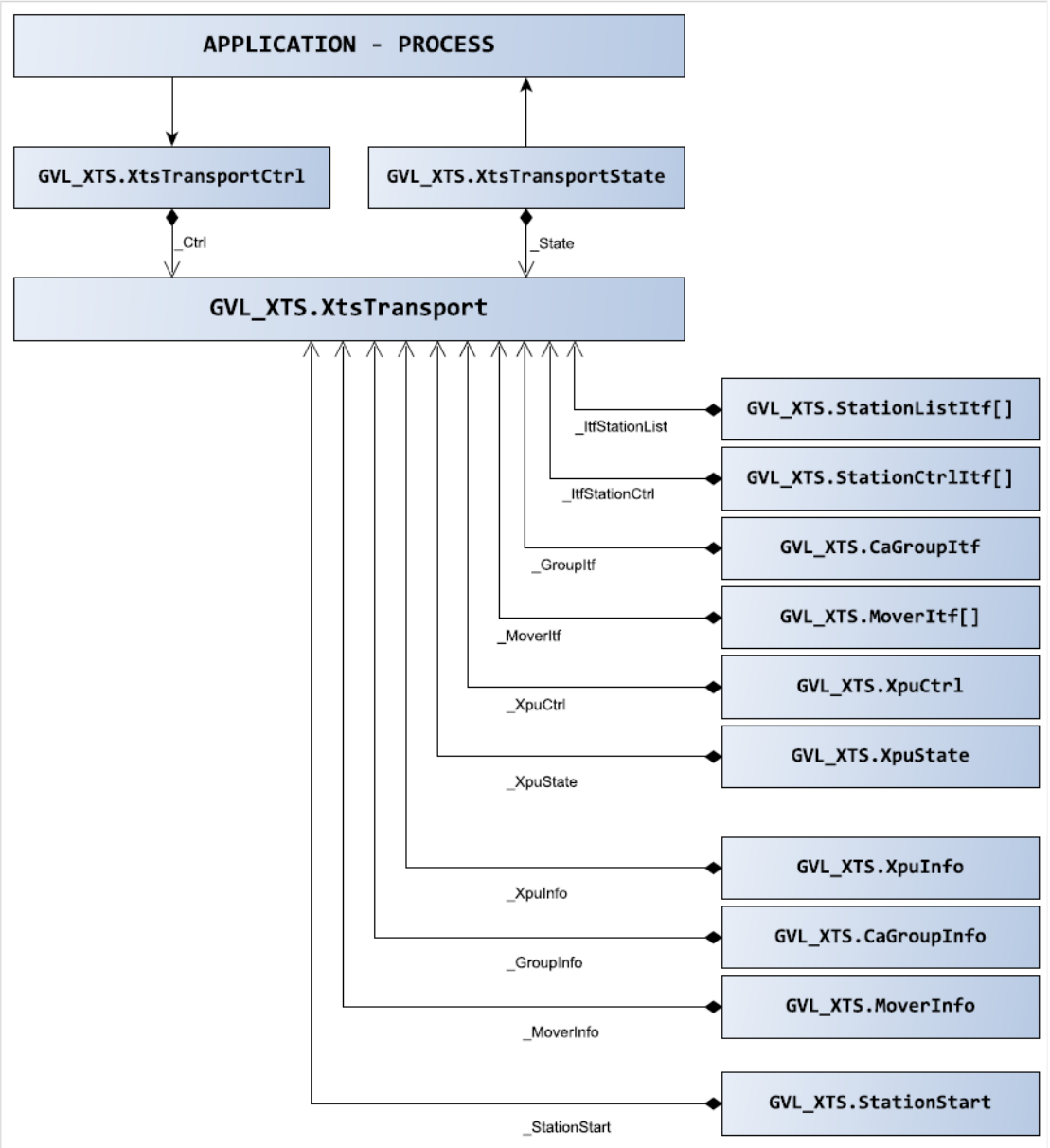
```
{attribute 'strict'}
{attribute 'to_string'}
TYPE E_XTS_TRANSPORT_CTRL :
(
  CMD_NULL,
  CMD_INIT                  := 10,
  CMD_IDLE,

  CMD_MOVER_ENABLE          := 20,
  CMD_MOVER_DISABLE,
  CMD_MOVER_HALT_CA,
  CMD_MOVER_STOP,

  CMD_GROUP_CLEAR           := 30,
  CMD_GROUP_BUILD,
  CMD_GROUP_ENABLE,
  CMD_GROUP_STOP,

  CMD_TRANSPORT_START       := 40,
  CMD_TRANSPORT_RESTART

)UINT;
END_TYPE
```



3. Design

BECKHOFF

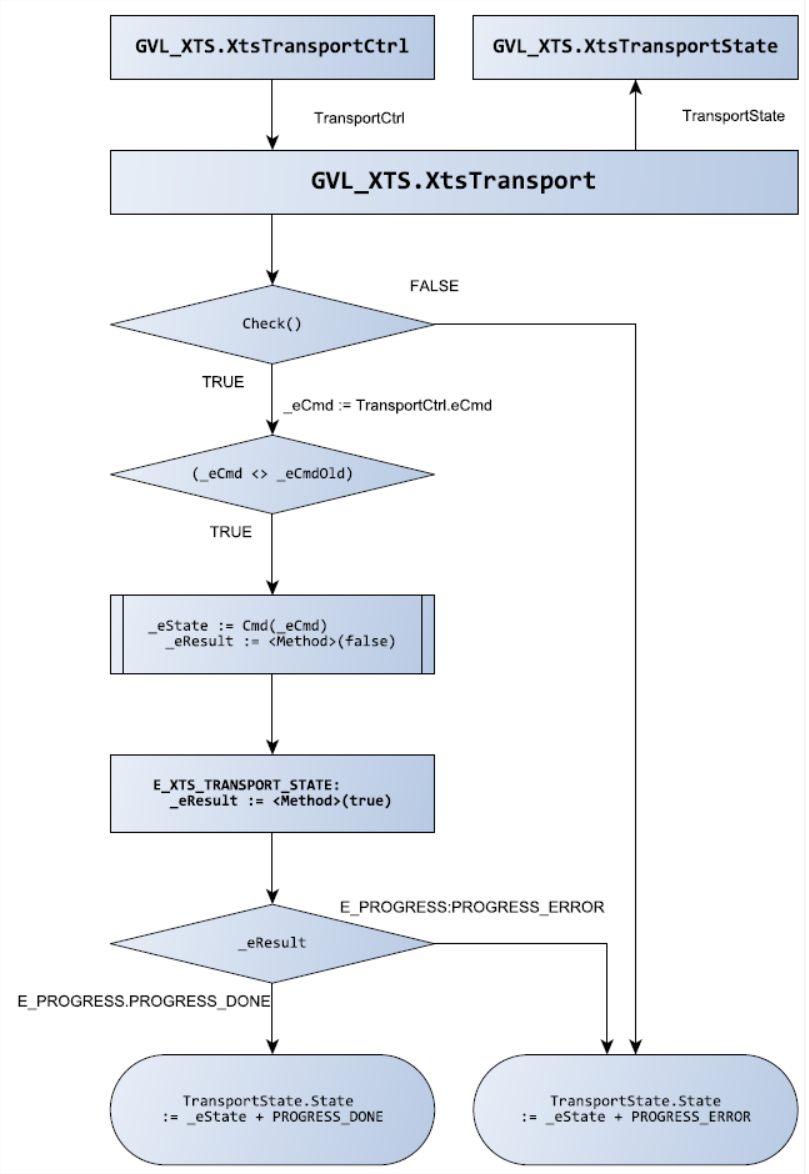
- TransportUnit
  - Fb_TransportUnit():
    - Members:



3. Design

BECKHOFF

- TransportUnit
  - Fb_TransportUnit():
    - Change of command triggers execution
    - Execution result is added to state
    - Extern control needs to react to BUSY, DONE or ERROR

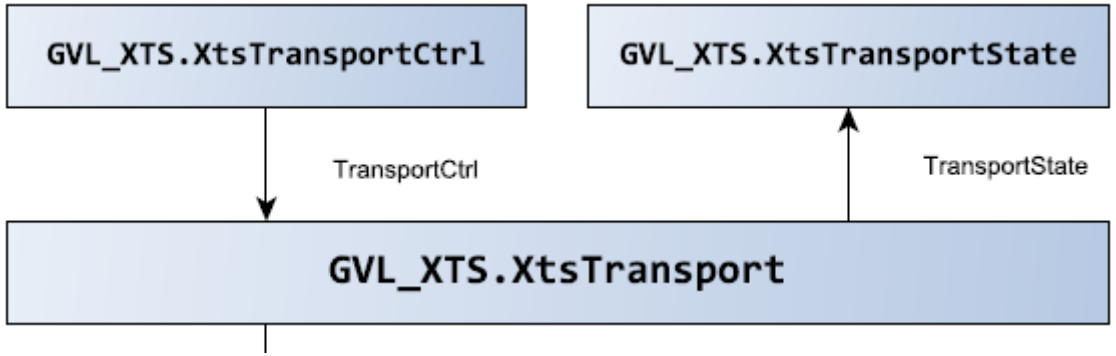


BECKHOFF

- TransportUnit
  - GVL_XTS.XtsTransportCtrl: **ST_TRANSPORT_UNIT_CTRL**
    - Struct for commanding FB_TransportUnit
    - eCmd : E_XTS_TRANSPORT_CTRL

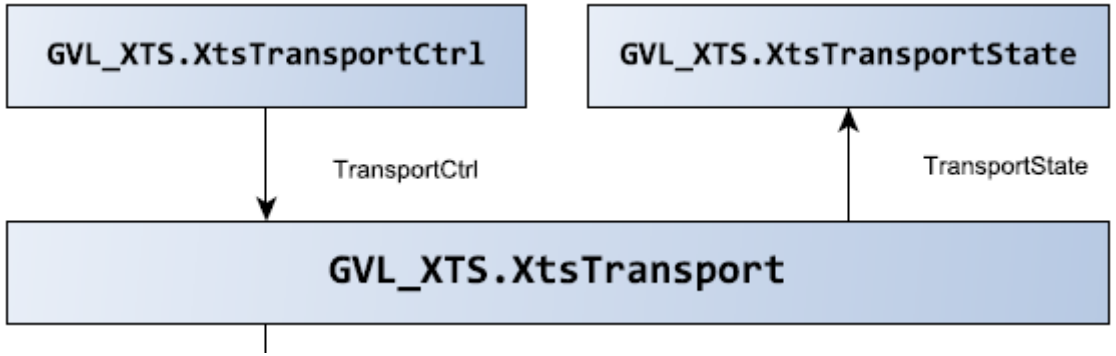
ST_XTS_TRANSPORT_CTRL

```
1  TYPE ST_XTS_TRANSPORT_CTRL :
2  STRUCT
3    Cmd        : E_XTS_TRANSPORT_CTRL;
4  END_STRUCT
5  END_TYPE
6
```



BECKHOFF

- TransportUnit
  - GVL_XTS.XtsTransportState: **ST_TRANSPORT_UNIT_STATE**
  - State: combines active command and result
  - Check: cyclic pointer checks
  - XpuState: state from fb_Xpu
  - XpuInfo: cyclic plausibility checks to TcCOM Objects
  - GroupInfo: cyclic information from FB_CaGroup



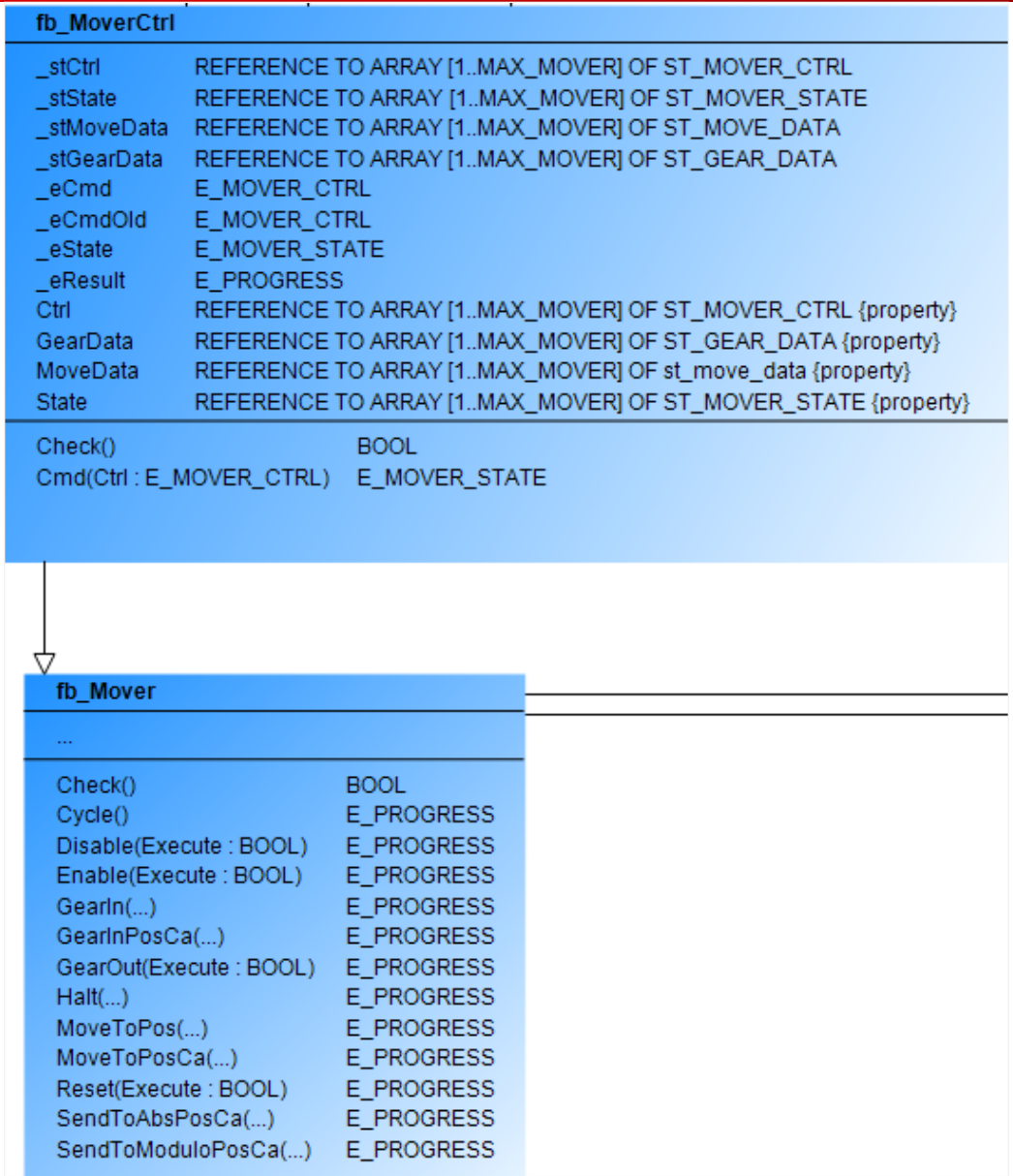
ST_XTS_TRANSPORT_STATE

```
1  {attribute 'pack_mode' := '2'}
2  TYPE ST_XTS_TRANSPORT_STATE :
3  STRUCT
4    State          : E_XTS_TRANSPORT_STATE;
5    Check          : E_XTS_TRANSPORT_CHECK;
6
7    XpuState       : ST_XPU_STATE;
8    XpuInfo        : ST_XPU_INFO;
9    GroupInfo      : ST_GROUP_INFO;
10 END_STRUCT
11 END_TYPE
12
```

3. Design

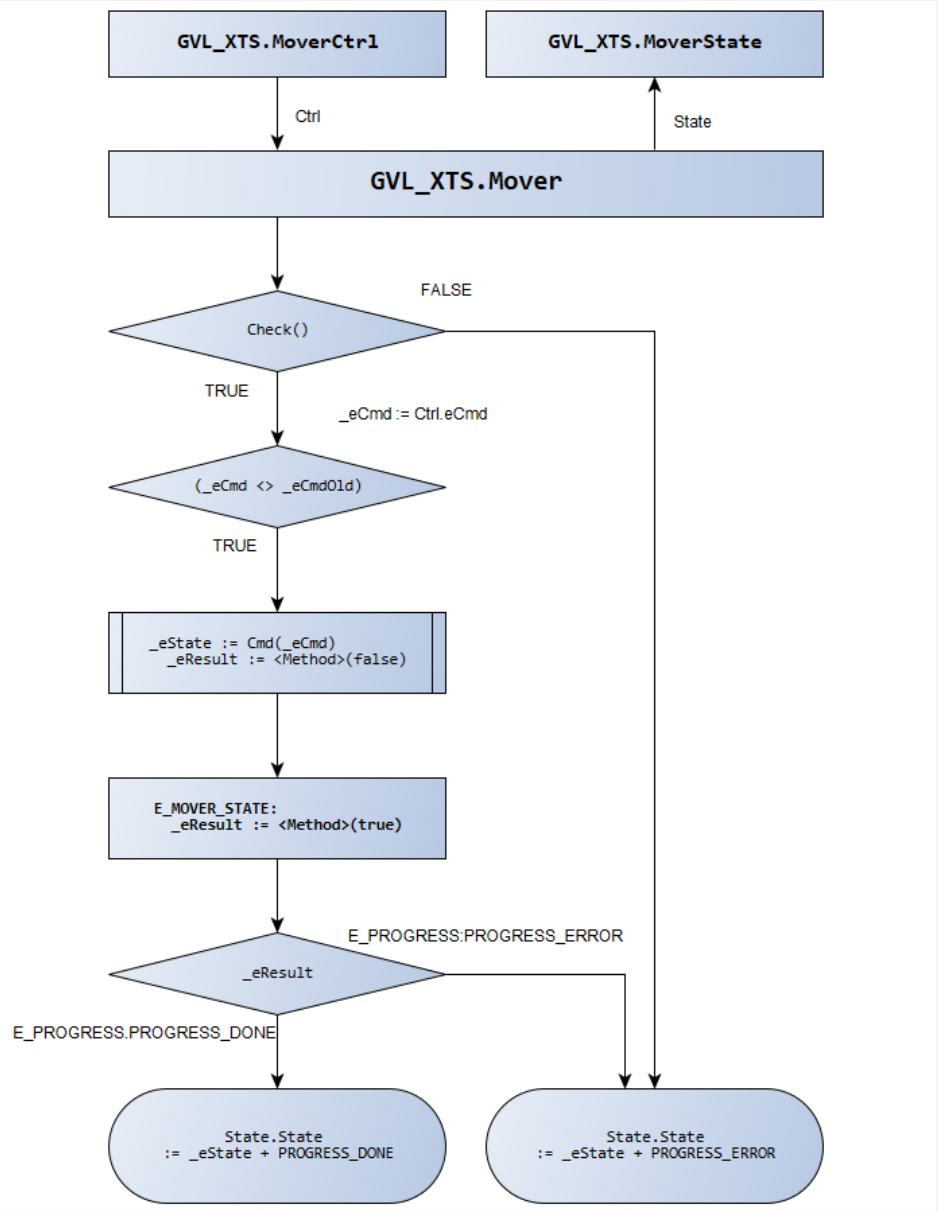
BECKHOFF

- GVL_XTS.Mover[] (**fb_MoverCtrl**)
  - Inherits **fb_Mover**
    - Access to MC function blocks in library
    - Implements Interface for use in other classes
  - Contains cyclic interface
    - OnChange check of command
    - Ctrl datafield for setting commands
    - State data field for checking responses
    - Parameter datafields for using motion functions



BECKHOFF

- fb_MoverCtrl:
  - Mover index is passed as value from caller
  - Global datafields are passed as references (**REF=**) into fb_MoverCtrl properties
    - OnChange Ctrl / State: handshakes
      - standard return value for method **(E_PROGRESS)**
    - OnExec log LastPosition of CA/MC function
    - OnExec log LastGap on CA function



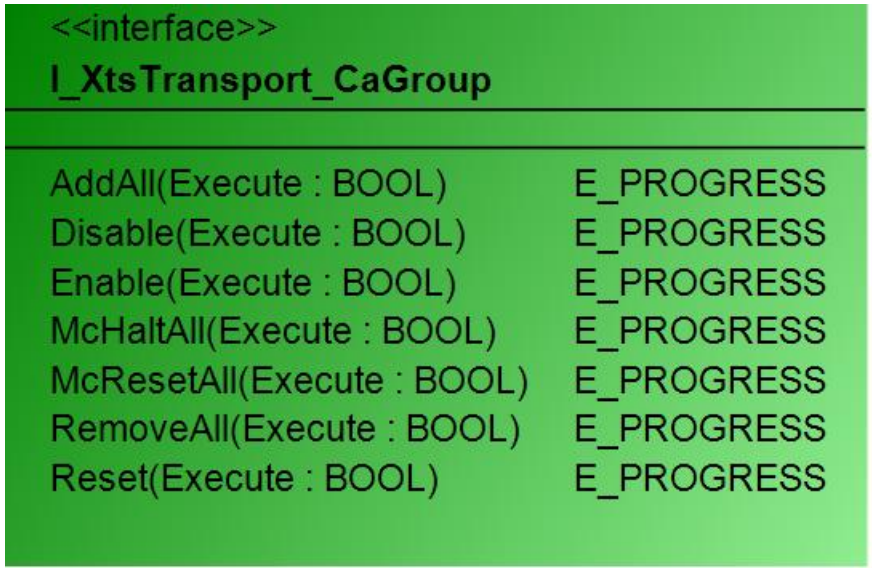
BECKHOFF

- fb_CaGroup:
  - Collision Avoidance class wrapper
  - Implements **I_Transport_CaGroup**
  - Cyclic information from **AXES_GROUP_REF**
  - Mover commands via interface **I_XtsTransport_Mover**

FB_CaGroup	
GROUP_HALT_JERK	LREAL
GROUP_HALT_DEC	LREAL
_eCheck	E_GROUP_CHECK
_bError	BOOL
_GroupRef	REFERENCE TO Tc3_McCoordinatedMotion.AXES_GROUP_REF
_GroupCommon	MCTOPLC_GROUP_COMMON_PART
_AxisRefMover	REFERENCE TO ARRAY [1..MAX_MOVER] OF Tc2_MC2.AXIS_REF
_MoverItf	REFERENCE TO ARRAY [1..MAX_MOVER] OF I_XtsTransport_Mover
_stMoveData	ST_MOVE_DATA
_fbAddAxisGroup	ARRAY [1..MAX_MOVER] OF Tc3_McCoordinatedMotion.MC_AddAxisToGroup
_fbRemoveAxisGroup	ARRAY [1..MAX_MOVER] OF Tc3_McCoordinatedMotion.MC_RemoveAxisFromGroup
_fbGroupDisable	Tc3_McCoordinatedMotion.MC_GroupDisable
_fbGroupEnable	Tc3_McCoordinatedMotion.MC_GroupEnable
_fbGroupErrorRead	Tc3_McCoordinatedMotion.MC_GroupReadError
_fbGroupStatusRead	Tc3_McCoordinatedMotion.MC_GroupReadStatus
_fbGroupReset	Tc3_McCoordinatedMotion.MC_GroupReset
_stGroupInfo	ST_GROUP_INFO
_rtrigGroupStatusRead	Tc2_Standard.R_TRIG
_rtrigGroupErrorRead	Tc2_Standard.R_TRIG
_stMsg	ST_Message
_eMessageLevel	E_MessageType
AxisRef	REFERENCE TO ARRAY [1..MAX_MOVER] OF Tc2_MC2.AXIS_REF {property}
GroupInfo	REFERENCE TO ST_GROUP_INFO {property}
GroupRef	REFERENCE TO Tc3_McCoordinatedMotion.AXES_GROUP_REF {property}
MessageLevel	e_messagetype {property}
MoverItf	REFERENCE TO ARRAY [1..MAX_MOVER] OF I_XtsTransport_Mover {property}
...	

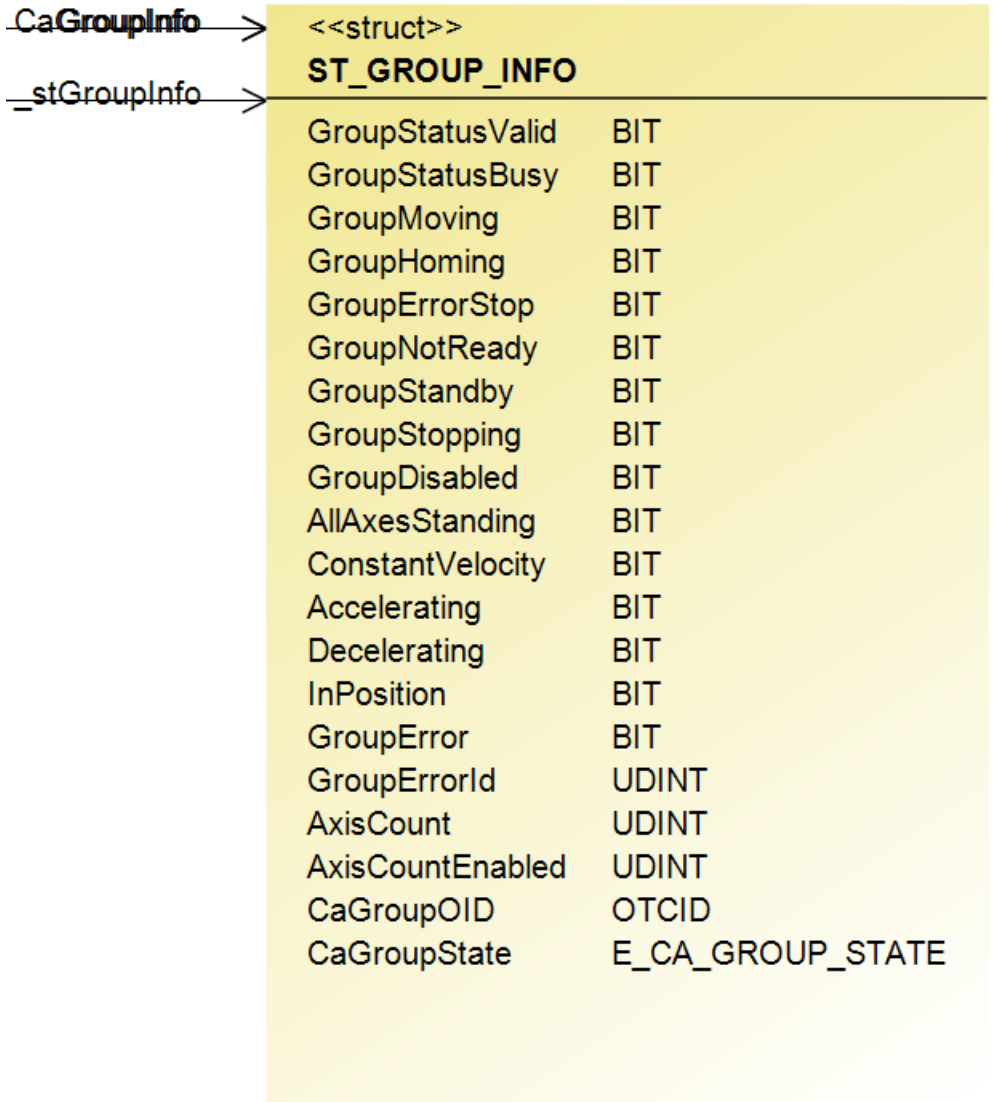
BECKHOFF

- fb_CaGroup:
  - Implements **I_Transport_CaGroup**
    - Used in fb_TransportUnit



BECKHOFF

- fb_CaGroup:
  - Cyclic information to **ST_GROUP_INFO**



BECKHOFF

- GVL_XTS.Xpu (**fb_XpuCtrl**)
  - Inherits **fb_Xpu**:
    - Class for interacting with XTS ProcessingUnit
    - XpuInit()
      - Connects to OTCIDs of XTS TcCOM Objects
    - Cycle
      - Plausibitlity checks, get module info data
    - ModuleInfoData, used in Cycle

fb_XpuCtrl

_Ctrl	REFERENCE TO ST_XPU_CTRL
_State	REFERENCE TO ST_XPU_STATE
_eCmd	E_XPU_CTRL
_eCmdOld	E_XPU_CTRL
_eResult	E_PROGRESS
_eState	E_XPU_STATE
Ctrl	REFERENCE TO ST_XPU_CTRL {property}
State	REFERENCE TO ST_XPU_STATE {property}

Check()	BOOL
Cmd(Ctrl : E_XPU_CTRL)	E_XPU_STATE
DetectMoverId(Enable : BOOL)	E_XPU_CHECK

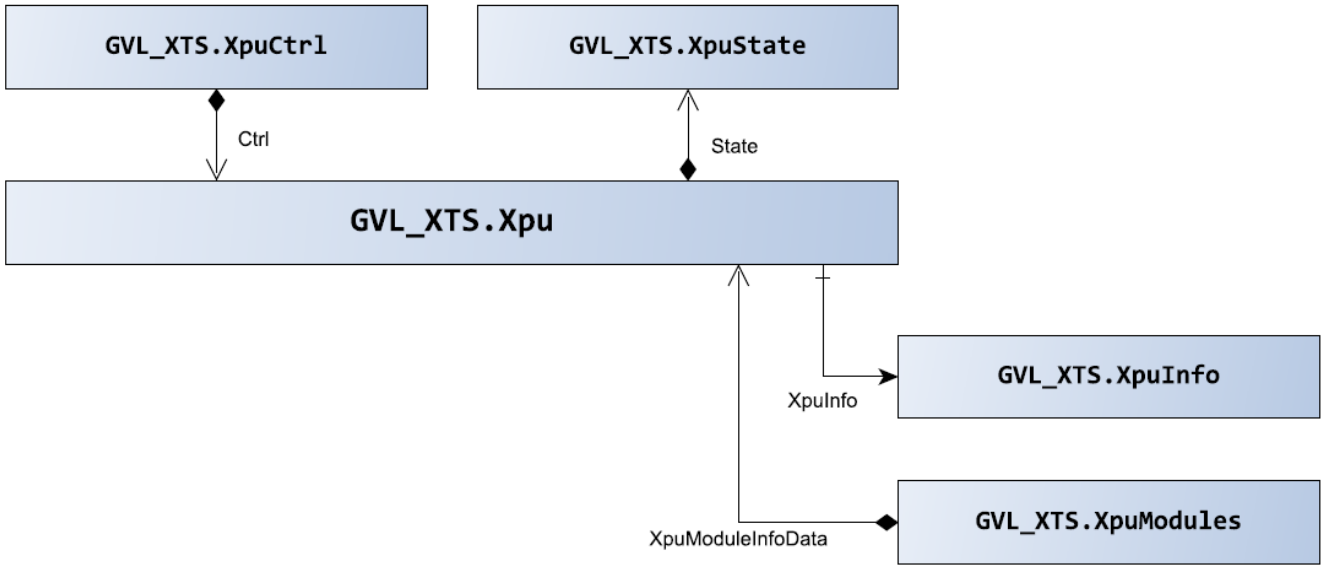
fb_Xpu

...

Cycle(...)	E_PROGRESS
GetEnvironment()	I_TcIoXtsEnvironment
IdDetectionModeToString()	STRING(20)
ModuleInfoData(Enable : BOOL)	E_PROGRESS
MoverPositionAssignementToString()	STRING(20)
OpModeToString()	STRING(20)
XpuInit(...)	E_XPU_INIT

BECKHOFF

- GVL_XTS.Xpu (**fb_XpuCtrl**)
  - Wraps cyclic execution of fb_Xpu
  - Cyclic check for command change **(ST_XPU_CTRL.Cmd)**



E_XPU_CTRL

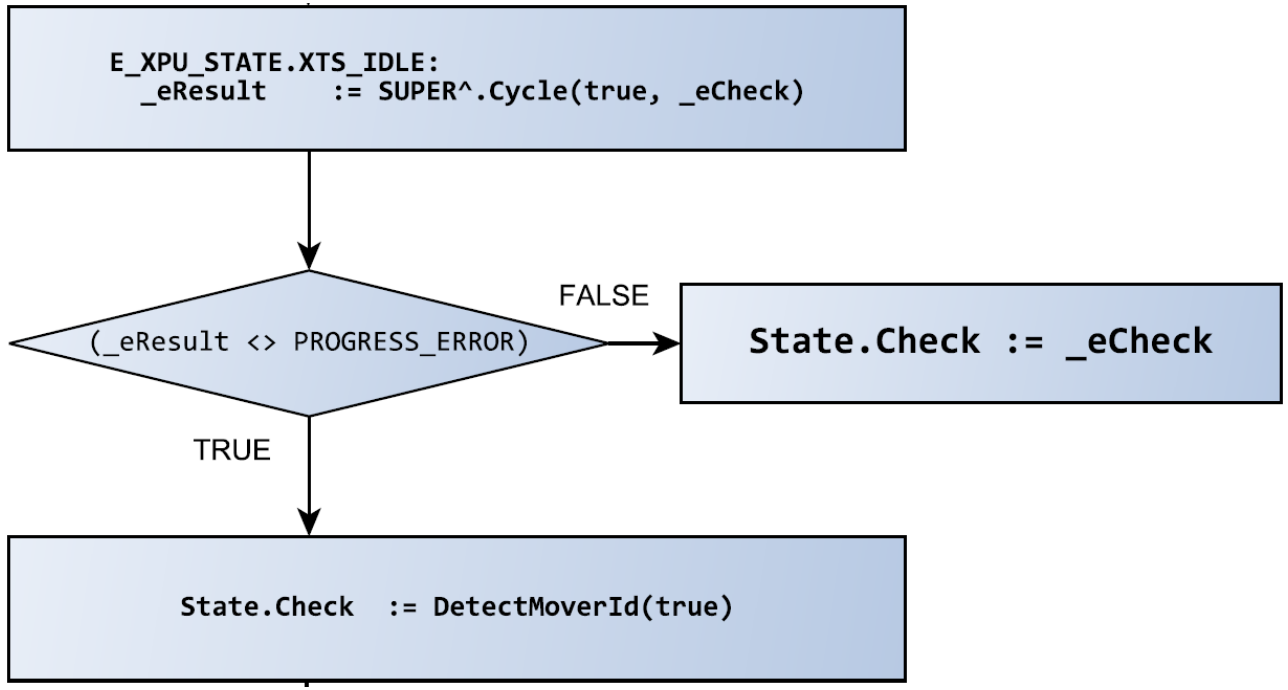
```
{attribute 'strict'}
TYPE E_XPU_CTRL :
(
  XTS_CMD_NULL,
  XTS_CMD_INIT                := 10,
  //XTS_CMD_READ_STN_CONFIG     := 20,
  XTS_IDLE                    := 30
)UINT;
END_TYPE
```

BECKHOFF

- fb_XpuCtrl cyclic data:
  - **SUPER^.Cycle():**
    - Cyclic plausibility checks
    - Cyclic update motor modules data
  - Cyclic data **ST_XPU_INFO:**

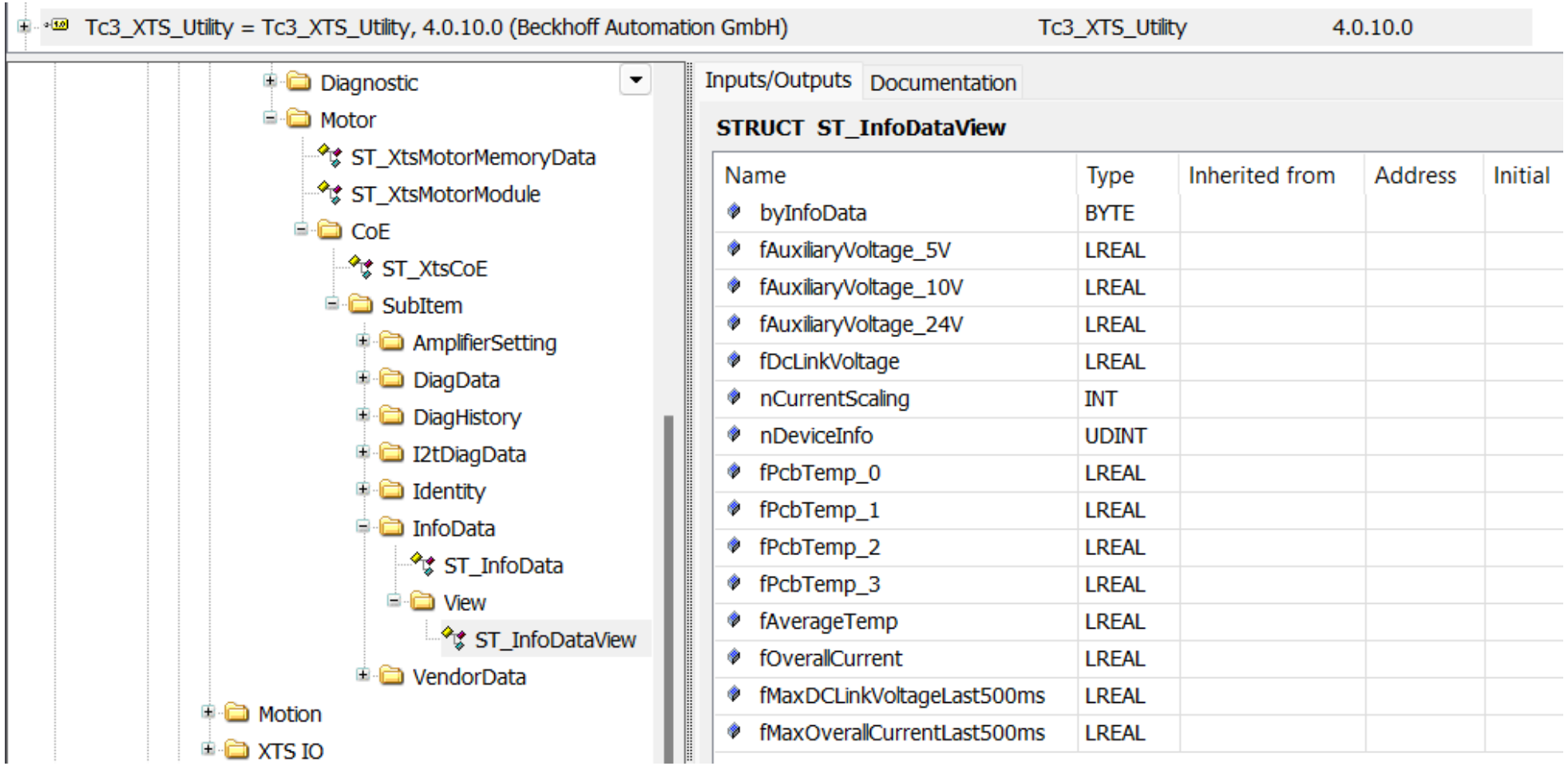
ST_XPU_INFO

```
{attribute 'pack_mode' := '2'}
TYPE ST_XPU_INFO :
STRUCT
  AllPositionsValid    : BIT;
  IdDetectionError     : BIT;
  IdDetectionValid     : BIT;
  IdDetectionActive    : BIT;

  OperationMode        : UINT;

  IdDetectionMode           : UINT;
  MoverPositionAssignement  : UINT;

  nDetectedAxisCount   : UINT;
  nExpectedAxisCount   : UINT;


END_STRUCT
END_TYPE
```



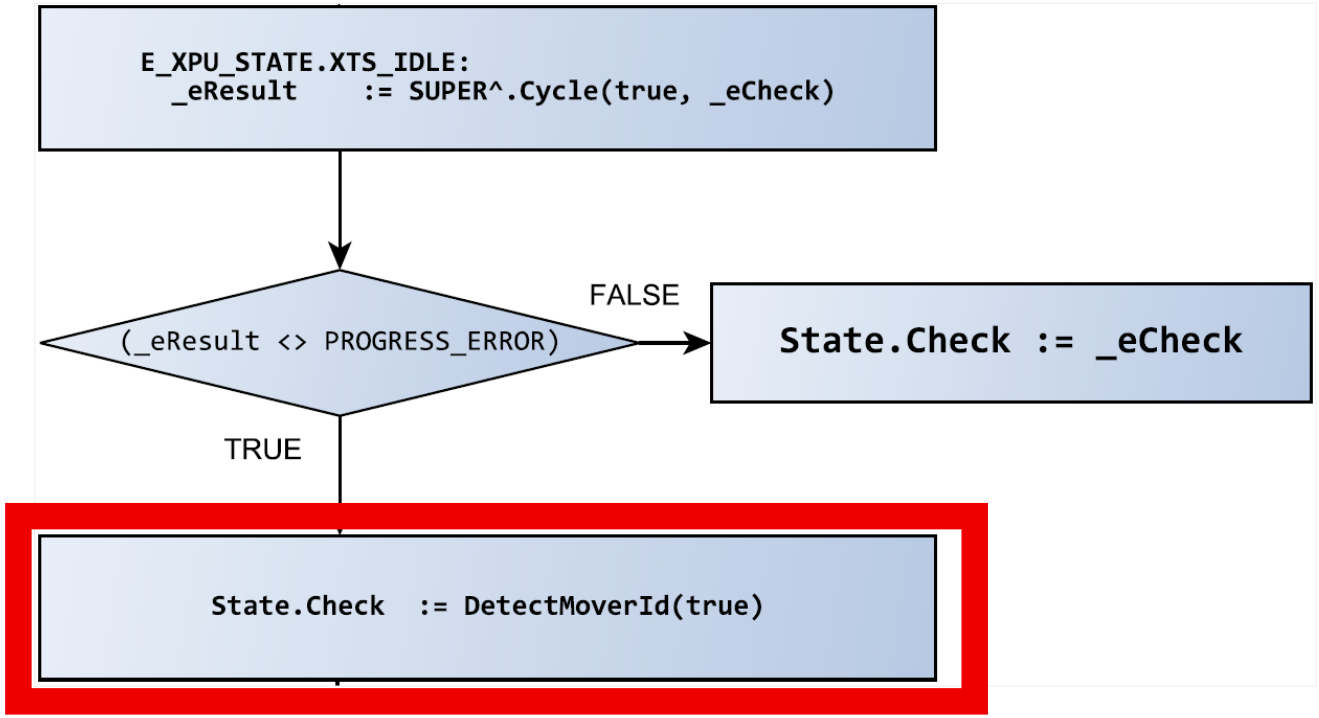
BECKHOFF

- fb_XpuCtrl:
  - Cyclic motor module data:



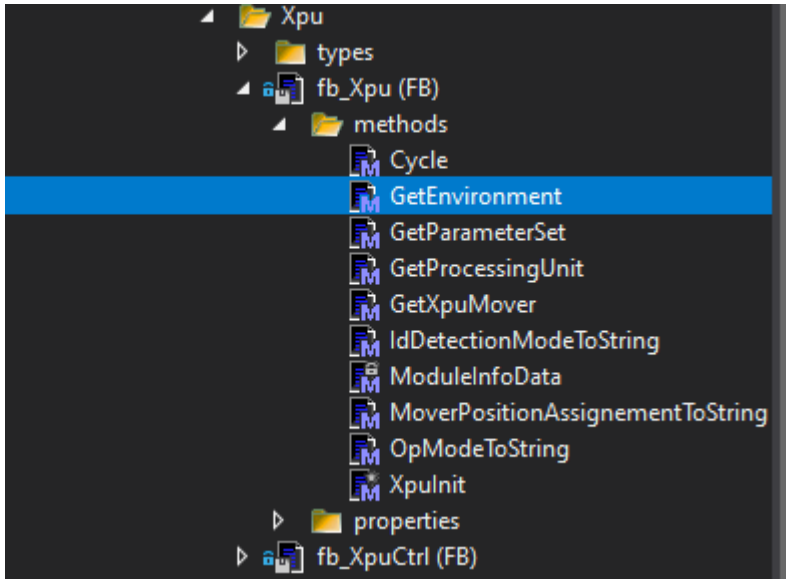
BECKHOFF

- fb_XpuCtrl:
  - **DetectMoverId:**
    - Cyclic checks for valid Mover ID Detection
    - Check pdf flowchart in [doc] folder of project



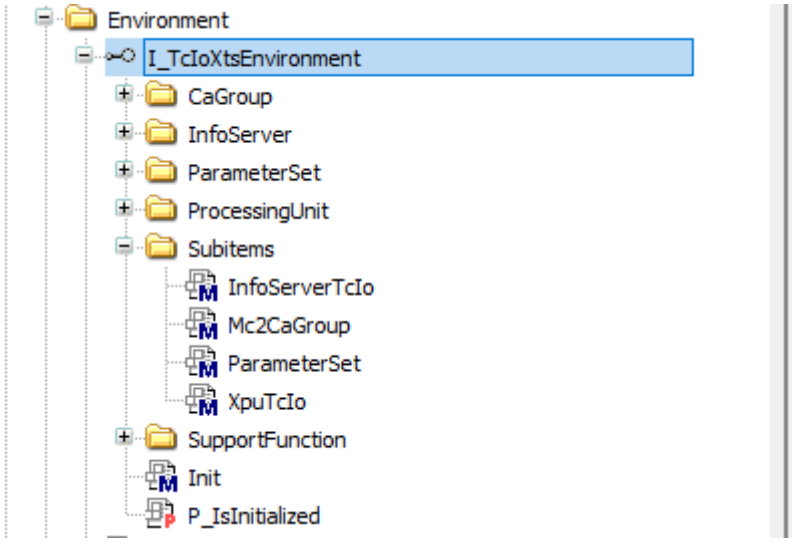
BECKHOFF

- fb_Xpu.GetEnvironment():
  - **Interface methods to Tc3_XTS_Utility.lib:**
    - GetEnvironment() : I_TcIoXtsEnvironment
      - Startup initialization of _fbEnvironment is done by fb_Xpu.Init()
      - Top level interface
        - See interface structure (Library Manager) in order to reach lower level interfaces



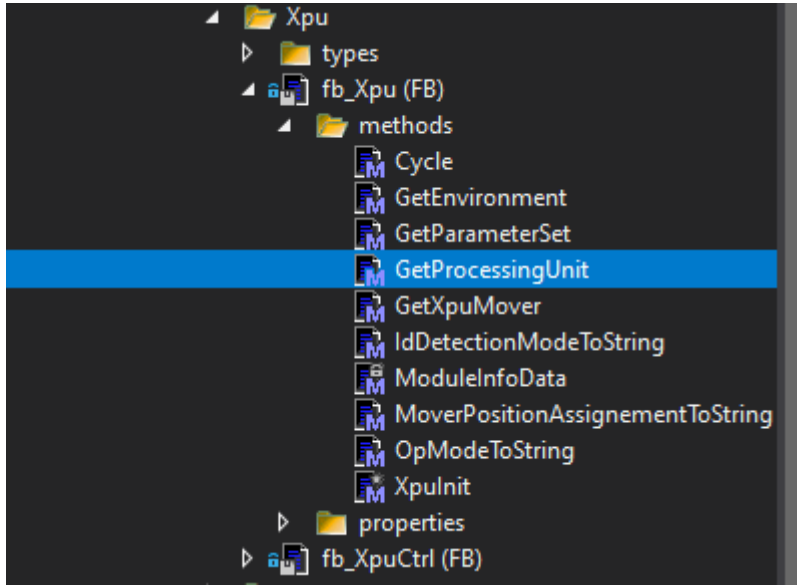
BECKHOFF

- fb_Xpu.GetEnvironment():
  - **I_TcIoXtsEnvironment:**
    - See LibraryManager
    - Provides access to members



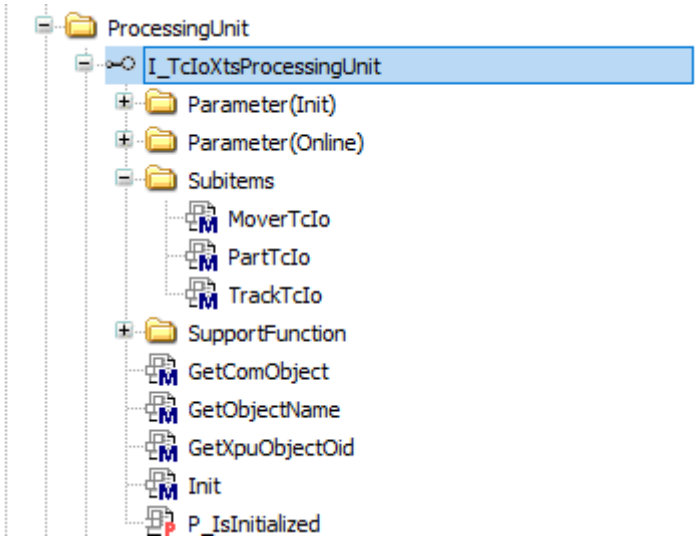
BECKHOFF

- fb_Xpu.GetProcessingUnit():
  - **Interface methods to Tc3_XTS_Utility.lib:**
    - GetProcessingUnit : I_TcIoXtsProcessingUnit
      - Startup initialization of _fbProcessingUnit is done by fb_Xpu.Init()
      - Interface for Processing Unit
        - See interface structure (Library Manager) in order to reach lower level interfaces



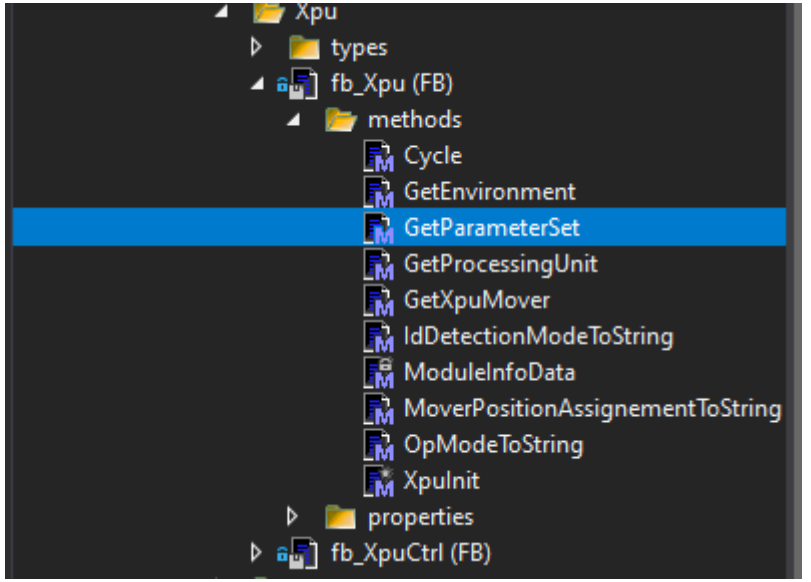
BECKHOFF

- fb_Xpu.GetProcessingUnit():
  - **I_TcIoXtsProcessingUnit:**
    - See LibraryManager
    - Provides access to members



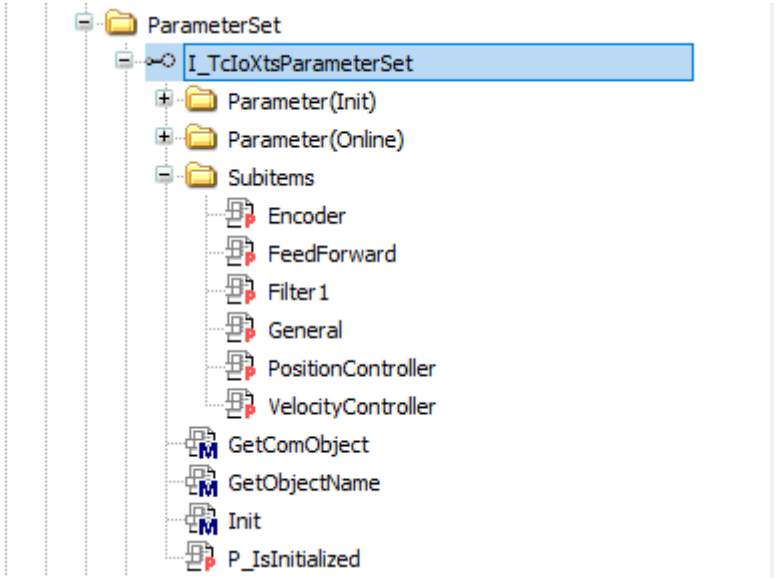
BECKHOFF

- fb_Xpu.GetParameterSet(Index):
  - **Interface methods to Tc3_XTS_Utility.lib:**
    - GetParameterSet(Index) : I_TcIoXtsParameterSet
      - Startup initialization of _fbParameterSet[ ] is done by fb_Xpu.Init()
      - Interface for ParameterSets
        - See interface structure (Library Manager) in order to reach lower level interfaces



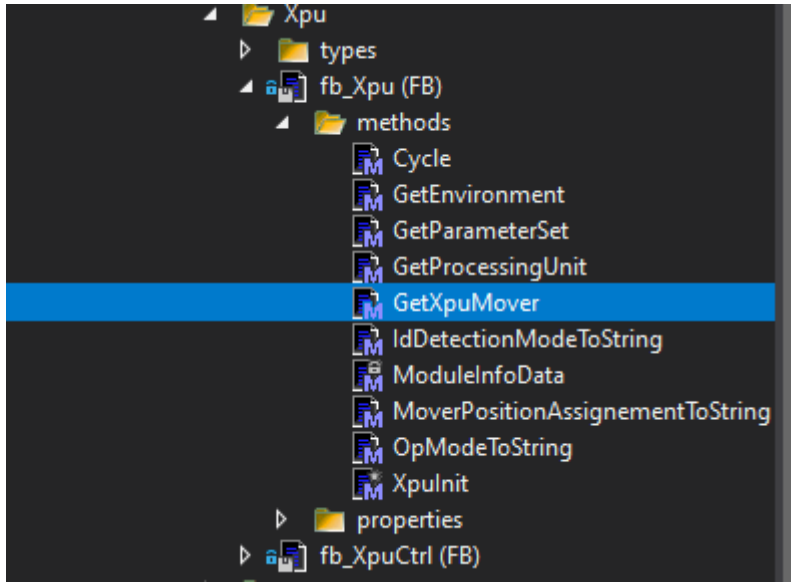
BECKHOFF

- fb_Xpu.GetParameterSet(Index):
  - **I_TcIoXtsParameterSet:**
    - See LibraryManager
    - Provides access to members



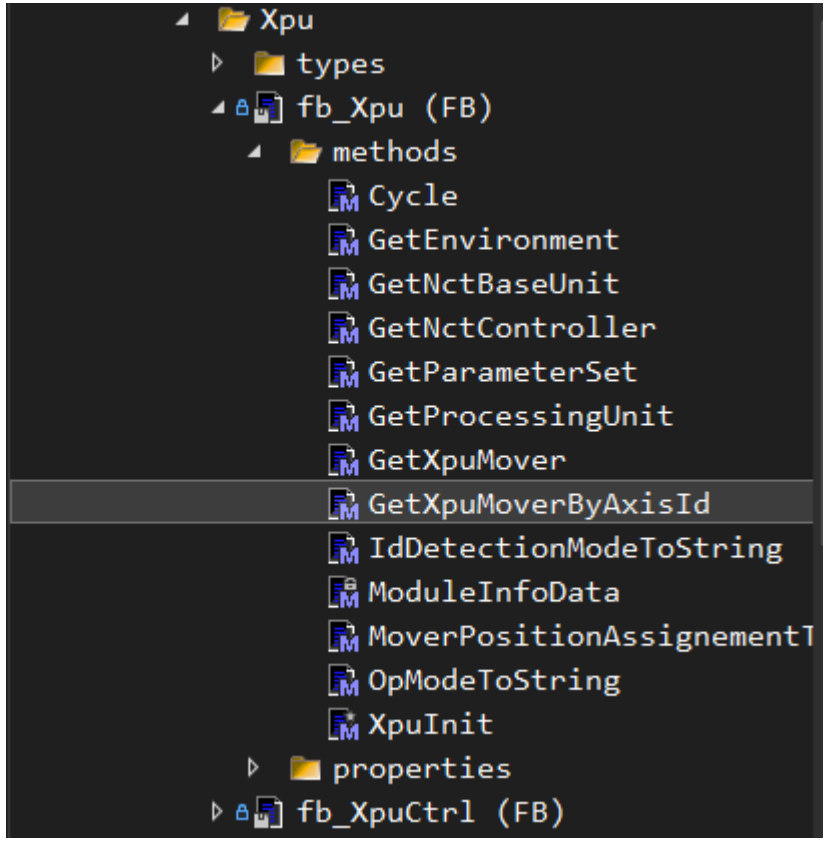
BECKHOFF

- fb_Xpu.GetXpuMover(Index):
  - **Interface methods to Tc3_XTS_Utility.lib:**
    - GetXpuMover(Index) : I_TcIoXtsXpuMover
      - Startup initialization of _fbMoverXpu[ ] is done by fb_Xpu.Init()
      - Interface for Mover
        - See interface structure (Library Manager) in order to reach lower level interfaces



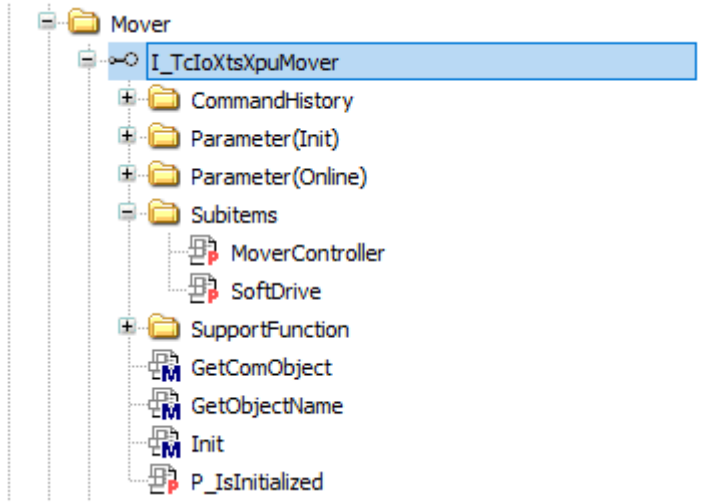
BECKHOFF

- fb_Xpu. GetXpuMoverByAxisId(AxisId):
  - **Interface methods to Tc3_XTS_Utility.lib:**
    - GetXpuMoverByAxisId(AxisId): I_TcIoXtsXpuMover
      - Startup initialization of _fbMoverXpu[ ] is done by fb_Xpu.Init()
      - Interface for Mover
        - See interface structure (Library Manager) in order to reach lower level interfaces



BECKHOFF

- fb_Xpu.GetXpuMover(Index):
- fb_Xpu.GetXpuMoverByAxisId(AxisId):
  - **I_TcIoXtsXpuMover:**
    - See LibraryManager
    - Provides access to members



4. Examples

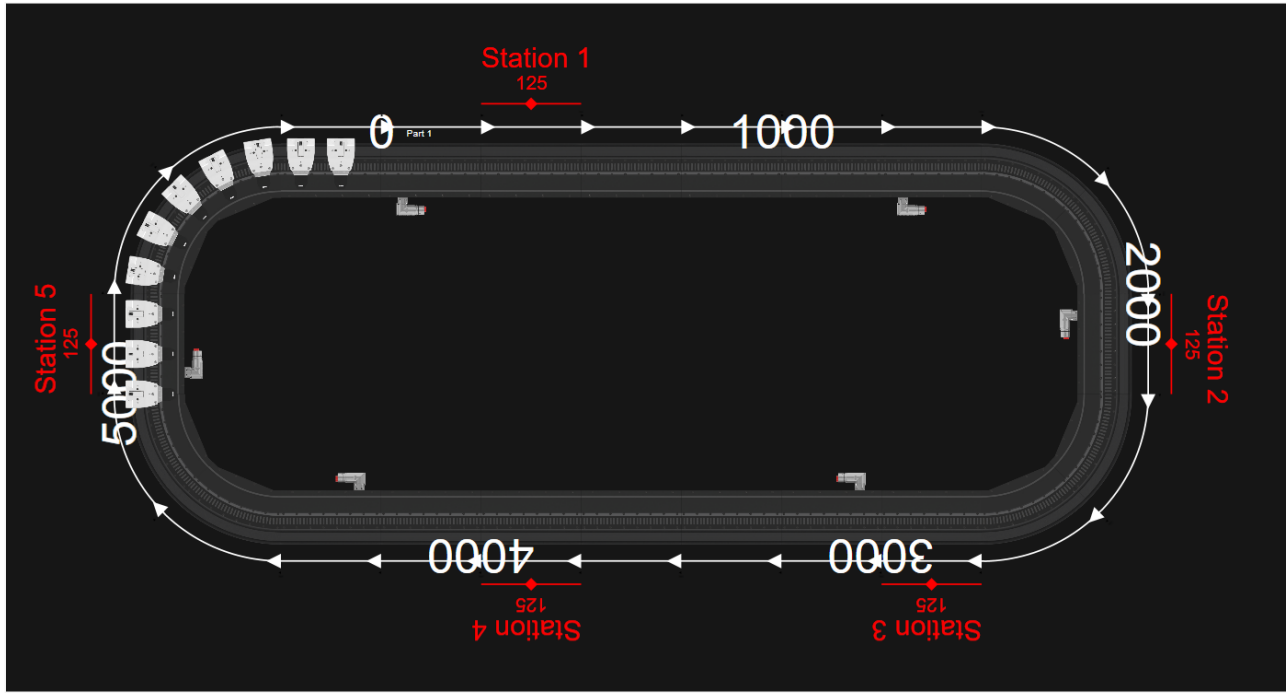
BECKHOFF

- **Transport Logic can be easy or may be complex**
  - **Examples shall help you to easily build any product transport.**
    - Please contact me via github; the place where you found this repo ;-)
  - All examples consider one specific aspect of the XTS transport system
    - Work as simple IndexTable
    - Work as Index Table where processes may host a number of XtsStations, working in parallel.
    - Complex, lot based decisions in order to achieve maximum production flexibility in one machine
    - Fast and precise for high speed packaging machines
    - Fast, synchronuous and precise for high end sorting and/or assembly machines

BECKHOFF

- XTS_DEMO_11
  - Single Station, Single Nest
    - ST_STATION_CTRL/ST_STATION_STATE is used for mover handshakes
  - Easy example for a XTS transport which requires only stations in which a mover stops once.
  - Easy transport logic;

    Target[n] := n+1; n+1 > MAX; n+1 := 1

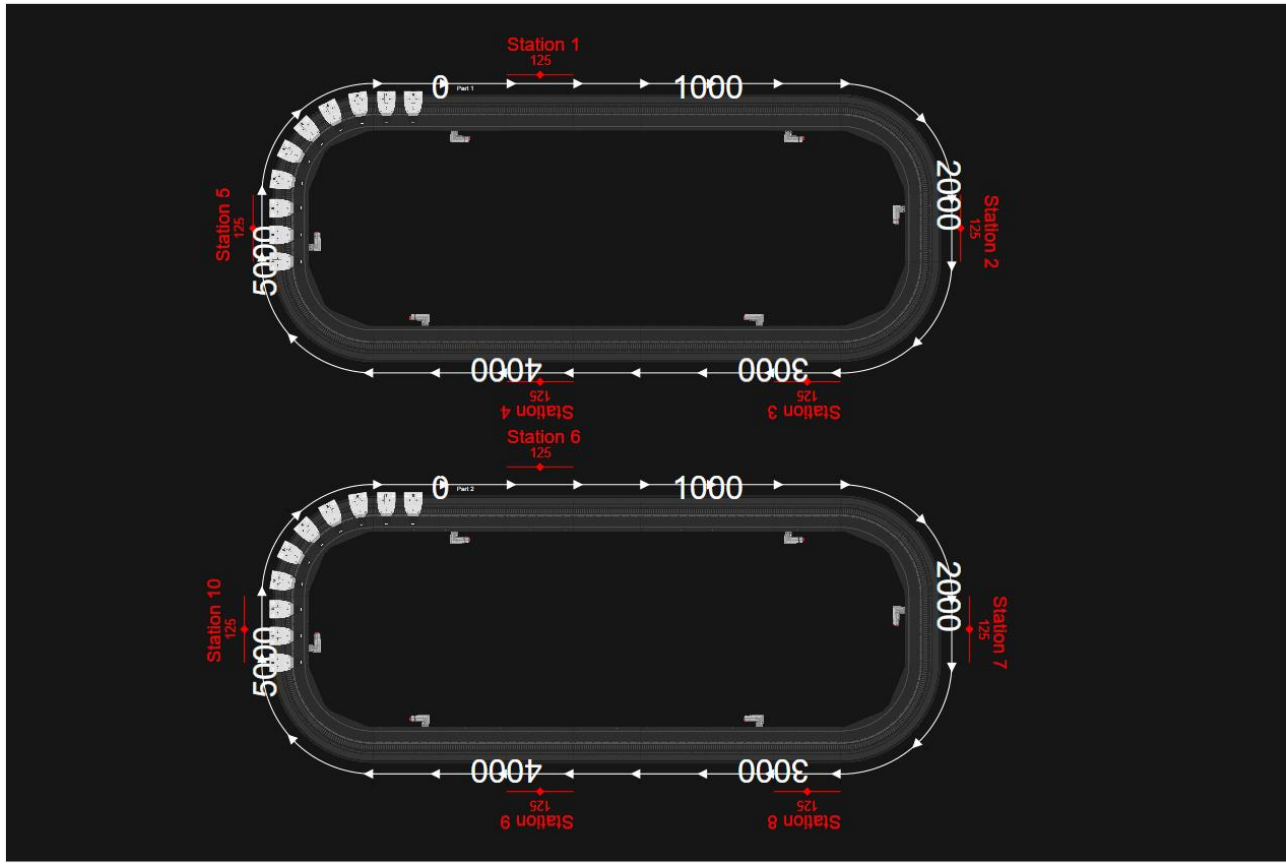


4. Examples

BECKHOFF

- XTS_DEMO_22
  - Multiple XPU
    - 2 ProcessingUnits in one PLC
  - Single Station, Single Nest
    - ST_STATION_CTRL/ST_STATION_STATE is used for mover handshakes
  - Easy example for an XTS transport requiring only stations in which a mover stops once.
  - Easy transport logic:

    Target[n] := n+1; n+1 > MAX; n+1 := 1

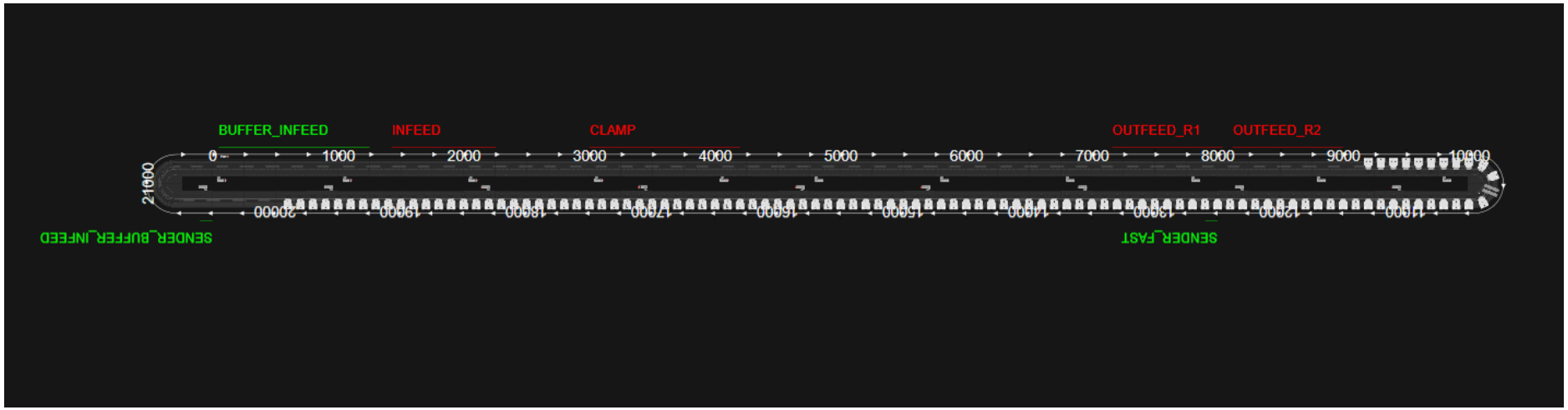


4. Examples

BECKHOFF

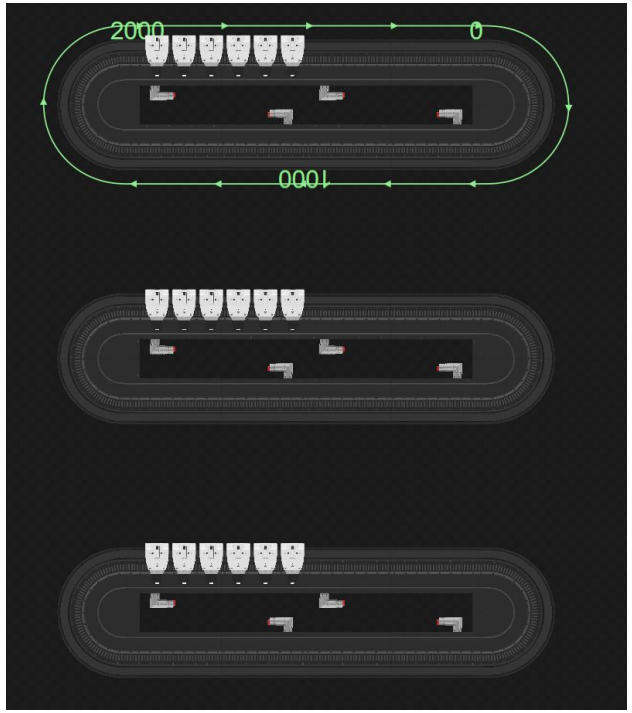
- XTS_DEMO_APPLICATION_108
  - High volume packaging application
  - 450 – 500 Mover/Minute
  - INFEED with gaps [1..12] possible
  - Any OUTFEED without gaps
    - 12 Movers per outfeed
  - 4 OUTFEED scenarios
    - `software switch (PLC)`
    - `R1 or  R2`
    - `R1 and R2`
    - `R1 xor R2`
    - `CLAMP`

-
  - Complex transport logic:
    - fb_ProcessCollector: grouping of XtsStations
    - fb_Instance: base class for logic implementation
    - Process_Instances: extending fb_Instance with transport details



4. Examples

- XTS_DEMO_LASER_CUT
  - Flying Saw application with 3 XPU
  - ConveyerBelt MasterAxis
  - CutMark detection (simulated in example)
  - GearInPosCA for 3 XPU with 6 Movers each
  - Use of multiple PLC
    - XtsTransport
    - ExternControl



-
  - Complex transport logic:
    - fb_ProcessCollector: grouping of XtsStations
    - fb_Instance: base class for logic implementation
    - Process_Instances: extending fb_Instance with transport details
    - fb_GearInPos extends fb_StationBase
    - Control and Feedback datafields for ADS or fieldbus access

4. Examples

BECKHOFF

- XTS_DEMO_DISPENSING_MOVER
  - Station based example
  - Dispensing Station
    - GearIn of Mover in Station (standstill)
    - Select *.nc file
    - Build NCI Config
    - Activate NCI config
    - Start *.nc program
    - Use of Mfunc to start dispenser axis
  - Use of PLC_MOTION_LAYER
    - https://github.com/haud-ba/PLC_MOTION_LAYER
-
  - Complex transport logic:
    - fb_ProcessCollector: grouping of XtsStations
    - fb_Instance: base class for logic implementation
    - Process_Instances: extending fb_Instance with transport details

XTS_TRANSPORT_LAYER project