BECKHOFF New Automation Technology

Manual | EN

TS5070

TwinCAT 2 | PackAL

TwinCAT Supplement

2022-11-23 | Version: 1.1

Table of contents

1 Foreword

1.1 Notes on the documentation

This description is only intended for the use of trained specialists in control and automation engineering who are familiar with applicable national standards.
It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components.
It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

The responsible staff must ensure that the application or use of the products described satisfy all the requirements for safety, including all the relevant laws, regulations, guidelines and standards.

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.
We reserve the right to revise and change the documentation at any time and without prior announcement.
No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.
Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
with corresponding applications or registrations in various other countries.

EtherCAT® is a registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany

Copyright

1.2 Safety instructions

Safety regulations

Please note the following safety instructions and explanations!
Product-specific safety instructions can be found on following pages or in the areas mounting, wiring, commissioning etc.

Exclusion of liability

All the components are supplied in particular hardware and software configurations appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

Personnel qualification

This description is only intended for trained specialists in control, automation and drive engineering who are familiar with the applicable national standards.

Description of symbols

In this documentation the following symbols are used with an accompanying safety instruction or note. The safety instructions must be read carefully and followed without fail!

⚠ DANGER

Serious risk of injury!

Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.

⚠ WARNING

Risk of injury!

Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.

⚠ CAUTION

Personal injuries!

Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.

NOTE

Damage to the environment or devices

Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.

Tip or pointer

This symbol indicates information that contributes to better understanding.

1.3 Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at https://www.beckhoff.com/secinfo.

2 Introduction

2.1 Overview of the defined Packaging Application Function Blocks

OMAC Packaging workgroup (OPW), a subset of the Open Modular Architecture Controls Users Group, has defined a set of functions to make an easy way for the end users for the control and automation of packaging machinery and systems.

The PackSoft Subcommittee recommends the use of PLCopen Function Blocks for Motion Control all based on IEC 61131-3. The PackSoft Subcommittee develops Packaging Industry related Control Software guidelines that allow to commonly implement as technology, program machinery equipment and controls, maintain, train and learn the use of software on packaging industry devices.

Objectives of the committee

The objectives of the OMAC PWG PackSoft committee are to

- define software modules (based on appropriate standards, e.g. IEC61131-3) which describe basic packaging machinery control elements
- develop a programming convention and a set of functional software elements which lend themselves to be become common use throughout the Packaging Industry to simplify the structure, understanding and handling of generic machine elements and their representation in software
- define a set of functions which will serve the majority of packaging user's applications and needs
- promote the adoption of useful existing and emerging standards to own definitions

General Objectives of the lib

- Machine related as first priority, process related as second
- Providing an easy-to-use interface to the Packaging Functionality
- Related to existing packaging standards
- Re-usable parts, usable in a wide range of applications
- Application program should be implementable on any platforms
- Accepted / User-tested Functionality / Concepts providing the basis for FBs
- Providing a common basis, terminology, references
- Providing a 'style guide' for additional / future FBs
- Providing user guidelines / examples
- Standard Function Blocks Library for standard Packaging related Functionality
- Combining these FB's to an application program needs an environment that is suitable for Packaging related applications. Requirements and restrictions for such an environment are partly dealt with in this standard.

Packaging Process Functions

PS_Batch Counter [▶ 15]

PS_ [▶ 15] Dancer Control [▶ 16]

PS_ [▶ 15] DigitalPLS [▶ 18]

PS_ [▶ 15] FlyingSync [▶ 21]

PS_ [▶ 15] GearInDyn [▶ 25]

PS_ [▶ 15] Indexing [▶ 26]

PS_ [▶ 15] JogAxis [▶ 28]

PS_ [▶ 15] OrientedStop [▶ 30]

PS_ [▶ 15] Registration [▶ 36]

PS_ [▶ 15] RegCorrection [▶ 32]

PS_SetEnable [▶ 38]

PS_ [▶ 15] SetOverride [▶ 39]

PS_ [▶ 15] Wind_csv (constant surface velocity, csv mode) [▶ 43]

Packaging Machine Communication

PS_Send [▶ 50],

PS_Receive [▶ 49]

Packaging Machine Behavior Organization

PS_PackML_StateMachine_Auto [▶ 54]

PS_PackML_StateMachine_Maintenance [▶ 56]

PS_PackML_StateMachine_Manual [▶ 59]

PS_PackML_StateMachine_SemiAuto [▶ 62]

PS_UnitModeManager [▶ 65]

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3 Packaging Process

3.1 Data Types

3.1.1 INPUT_REF

```
TYPE INPUT_REF: MC_InputRef;
END_TYPE
```

INPUT_REF : Data structure of type MC_InputRef, that describes the trigger input for the recording of the axis position.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.1.2 OUTPUT_REF

```
TYPE OUTPUT_REF : ARRAY[1..32] OF BOOL;
END_TYPE
```

Array for signal ouputs.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.1.3 E_CamSwitchDirection

```
TYPE E_CamSwitchDirection :
(
    CAMSWITCHDIRECTION_BOTH         := 0,
    CAMSWITCHDIRECTION_POSITIVE     := 1,
    CAMSWITCHDIRECTION_NEGATIVE     := 2
);
END_TYPE
```

Defines the direction cam switch is active (Both (=0; Default); Positive (1); Negative (2)).

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.1.4 E_CamSwitchMode

```
TYPE E_CamSwitchMode :
(
    CAMSWITCHMODE_POSITION      := 0,
    CAMSWITCHMODE_TIME          := 1,
    CAMSWITCHMODE_BREAK         := 2
);
END_TYPE
```

CamSwitchMode can be Position, Time or other additional supplier specific types.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.1.5 AXIS_REF

```
TYPE AXIS_REF:
STRUCT
    NcToPlc     : POINTER TO NCTOPLC_AXLESTRUCT;
    PlcToNc     : POINTER TO PLCTONC_AXLESTRUCT;
END_STRUCT
END_TYPE
```

NcToPlc: Pointer to the axis structure provided by the NC.

PlcToNc: Pointer to the axis structure provided by the PLC.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.1.6 ST_CamSwitchInternal

```
TYPE ST_CamSwitchInternal :
STRUCT
    State   : BOOL;     (* current state of the digital cam switch *)
    Timer   : TP;       (* for internal use only *)
END_STRUCT
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.1.7 DANCER_REF

```
TYPE DANCER_REF:
STRUCT
    Tension_ctl     : DINT;
    GearRatio       : REAL;
    deltaGear       : REAL;
    GearOffset      : REAL;
    fKp             : REAL;
    fTn             : REAL;
    fTv             : REAL;
    fTd             : REAL;
    Accel_limit     : REAL;
END_STRUCT
END_TYPE
```

Tension_ctl: Target value for web tension, typically the target position for the dancer in balanced condition.

GearRatio: Ratio of gear factor g(t) between master (web) and Slave (spool) to balance the dancer. Default is 1.0.

deltaGear: Delta scaling multiplier of PID output (-1.0 … +1.0) to GearRatio – must be smaller than 1 but never 0. deltaGear would be e.g. 0.8 or 0.9 to limit the gear distortion.

GearOffset: scaling offset to fit GearRatio distortion by satisfying equation g(t) = deltaGear* PIDout + Gearoffset .

fKp: PID Control Proportional gain (P).

fTn: PID Control Integral gain Tn (I) [s].

fTv:PID Control Derivative gain Tv (D-T1) [s].

fTd: PID Control Derivative damping time Td (D-T1) [s] .

Accel_limit: corresponds indirectly, i.e. relative to a maximum master velocity, to a maximum permitted acceleration ($p_a = a_{SlaveMax} / v_{MasterMax}$). The limit parameter p_a corresponds to the reciprocal value of the run-up time $t_H = 1 / p_a$.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.1.8 DIGPLS_REF

```
TYPE DIGPLS_REF :
STRUCT
    NumberOfSwitches    : UDINT;  (* number of used array elements -
 can be less than the maximum number *)
    pSwitches           : POINTER TO PLS_CamSwitch;
    SizeOfSwitches      : UDINT;
END_STRUCT
END_TYPE
```

NumerOfSwitches : Defines the number of swtiches.

pSwitches : Typically a pointer to an array of switches passed with ADR(<array of digital cam switches>).

SizeOfSwitches : SIZEOF(<array of digital cam switches>).

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.1.9 PLS_CamSwitch

```
TYPE PLS_CamSwitch :
STRUCT
    TrackNumber         : INT;                  (* affected track number 1..32 *)
    FirstOnPosition     : LREAL;                (* trigger position for all cam types *)
    LastOnPosition      : LREAL;                (* for digital position cam only *)
    AxisDirection       : E_CamSwitchDirection;
    CamSwitchMode       : E_CamSwitchMode;      (* digital cam mode Position, Time etc. *)
    Duration            : TIME;                 (* [s] for digital time cam only *)
    MultiStrokeCycles   : LREAL;                (* multiplier for modulo cycle -
 just valid when TRACK_REF.Modulo is TRUE *)
    (* internal *)
    _internal           : ST_CamSwitchInternal; (* data for internal use only *)
END_STRUCT
END_TYPE
```

TrackNumber : TrackNumber is the reference to the track.

FirstOnPosition : Lower boundary where the switch is ON.

LastOnPosition : Upper boundary where the switch is ON.

AxisDirection : Direction [▶ 10] cam switch is active (Both (=0; Default); Positive (1); Negative (2)).

CamSwitchMode : Mode [▶ 10] of Cam switch (Position based (=0; Default); Time based (=1)).

Duration : Coupled to time based CamSwitchMode.

MultiStrokeCycles : Multiplier for modulo cycle - just valid when TRACK_REF.Modulo is TRUE .

_internal : This structure [▶ 11] defines the number of moves and hence the array size.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.1.10 PLS_TrackOptions

```
TYPE PLS_TrackOptions :
STRUCT
    OnCompensation  : LREAL;    (* compensation time [s] *)
    OffCompensation : LREAL;    (* compensation time [s] *)
    Hysteresis      : LREAL;    (* distance from last switch position (+ or -) *)
    BreakRelease    : BOOL;     (* allow break to be released when TRUE, break cams will be activate
d when FALSE *)
    Force           : BOOL;     (* override all digital cams and set track ON *)
    Disable         : BOOL;     (* override all digital cams and set track OFF -
 overrides Force as well *)
    _internal       : ST_TrackOptionsInternal;
END_STRUCT
END_TYPE
```

OnCompensation : Compensation time with which the switching on is advanced or delayed in time per track.

OffCompensation : Time compensation the switching off is delayed per track.

Hysteresis : Switching is shifted to a later reached position in order to avoid multiple switching around the switching point.

BreakRelease : allow break to be released when TRUE, break cams will be activated when FALSE.

Force : override all digital cams and set track ON.

Disable : override all digital cams and set track OFF - overrides Force as well.

_internal : ST_TrackOptionsInternal [▶ 14].

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.1.11 POS_REF

```
TYPE POS_REF :
STRUCT
    Position        : ARRAY [1..iMAX_POSITIONS] OF LREAL;
    Velocity        : ARRAY [1..iMAX_POSITIONS] OF LREAL;
    Acceleration    : ARRAY [1..iMAX_POSITIONS] OF LREAL;
    Deceleration    : ARRAY [1..iMAX_POSITIONS] OF LREAL;
    Jerk        : ARRAY [1..iMAX_POSITIONS] OF LREAL;
END_STRUCT
END_TYPE
```

Position : Array of position values for different number of moves.

Velocity : Array of velocity values for different number of moves.

Acceleration : Array of acceleration values for different number of moves.

Deceleration : Array of deceleration values for different number of moves.

Jerk : Array of jerk values for different number of moves.

iMAX_POSITIONS : This global variable defines the number of moves and hence the array size.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.1.12 TRACK_REF

```
TYPE TRACK_REF :
STRUCT
    Modulo      : BOOL;  (* all cam positions are interpreted as modulo positions when TRUE *)
    ModuloCycle : LREAL; (* e. g. 360 degrees *)
    DeadTime    : LREAL; (* [s] dead time compensation based on I/O delay and encoder velocity *)
    Track       : ARRAY[1..32] OF PLS_TrackOptions;
END_STRUCT
END_TYPE
```

Modulo : All cam positions are interpreted as modulo positions when TRUE.

ModuloCycle : Modulo cycle in degrees.

DeadTime : Dead time compensation based on I/O delay and encoder velocity

Track: Structure [▶ 13] with options for the track.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.1.13 ST_TrackOptionsInternal

```
TYPE ST_TrackOptionsInternal :
STRUCT
    SwitchPosition  : LREAL; (* latest rising or falling edge position of the digital cam *)
    BreakRelease    : BOOL;  (* internal storage - delayed to break position *)
END_STRUCT
END_TYPE
```

SwitchPosition : latest rising or falling edge position of the digital cam.

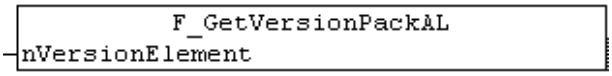
BreakRelease : internal storage - delayed to break position.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.2 Function Blocks

3.2.1 F_GetVersionPackAL



The function returns library version info.

FUNCTION F_GetVersionPackAL : UINT

```
VAR_INPUT
    nVersionElement : INT;
END_VAR
```

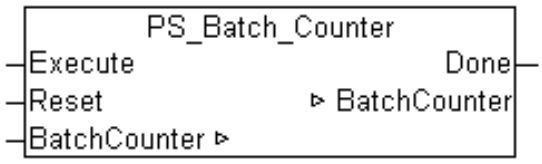
nVersionElement : Version parameter:

- 1 : major number;
- 2 : minor number;
- 3 : revision number;

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0	PC (i386)	TcPackALv3.0.Lib

3.2.2 PS_Batch_Counter



This function block provides a simple batch counter to handle the number of process cycles to accomplish a batch.

Note **This function block will – upon rising edge of Execute – decrement the BatchCounter by 1 to zero and set the “Done” flag. A reset input allows to reset the counter to 0 for an early termination.BatchCounter is set to the number of cycles to go and the FB counts down on every cycle. Once the Batch Counter reaches 0, the DONE Flag is set and the decrement stops on further Execute commands.**

VAR_INPUT

```
VAR_INPUT
    Execute         : BOOL;
    Reset           : BOOL;
END_VAR
```

Execute : Decrements the batch counter to zero on R_Trig.

Reset : Resets batch counter to zero

VAR_OUTPUT

```
VAR_OUTPUT
    Done            : BOOL;
END_VAR
```

Done: True if batch counter is zero

VAR_IN_OUT

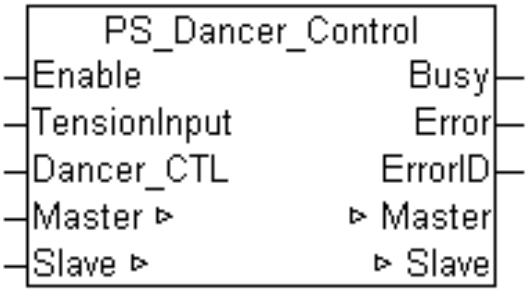
```
VAR_IN_OUT
    BatchCounter    : UDINT;     (* Counter *)
END_VAR
```

BatchCounter : Batch counter, decrementing.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.2.3 PS_Dancer_Control



This function block commands a controlled motion of an axis as slave of a dancer-coupled master axis. The Master axis is a physical or virtual axis. This FB provides the coupling between a slave axis (typically the infeed to the dancer) and a master axis (the outfeed of the dancer) via a dancer-PID controlled variable gearing factor.

Note The FB's purpose is to generate and re-adjust a constant surface velocity (peripheral speed), relative to the master axis, for the rotary calibrated and controlled slave axis, depending on a dancer position. Via a position signal (dancer position signal), the tension between master (web) and slave (e.g. spool) is represented. For general use, the raw dancer position is often aligned to a "balance position" by an offset (and optional multiplier) in a first dancer scaling algorithm. The PID calculates the control value depending on the difference to a scaled command dancer position. This PID control value output then tunes the gear ratio between the master (web) and the slave (e.g. spool). A multiplier limits the distortion of the gear, offset adjusts to gear setpoint. Scaling algorithm, dancer balance position, PID factors, gear factor, delta and offset to the gear are application specific.

Other possible implementations for Dancer Control, especially those with simpler two-switch control compared to PID control, may be represented by additional Function Blocks defined in the future. Tension_input is scanned in every cycle of the Function Block execution, other inputs in first cycle only.

Signal curve

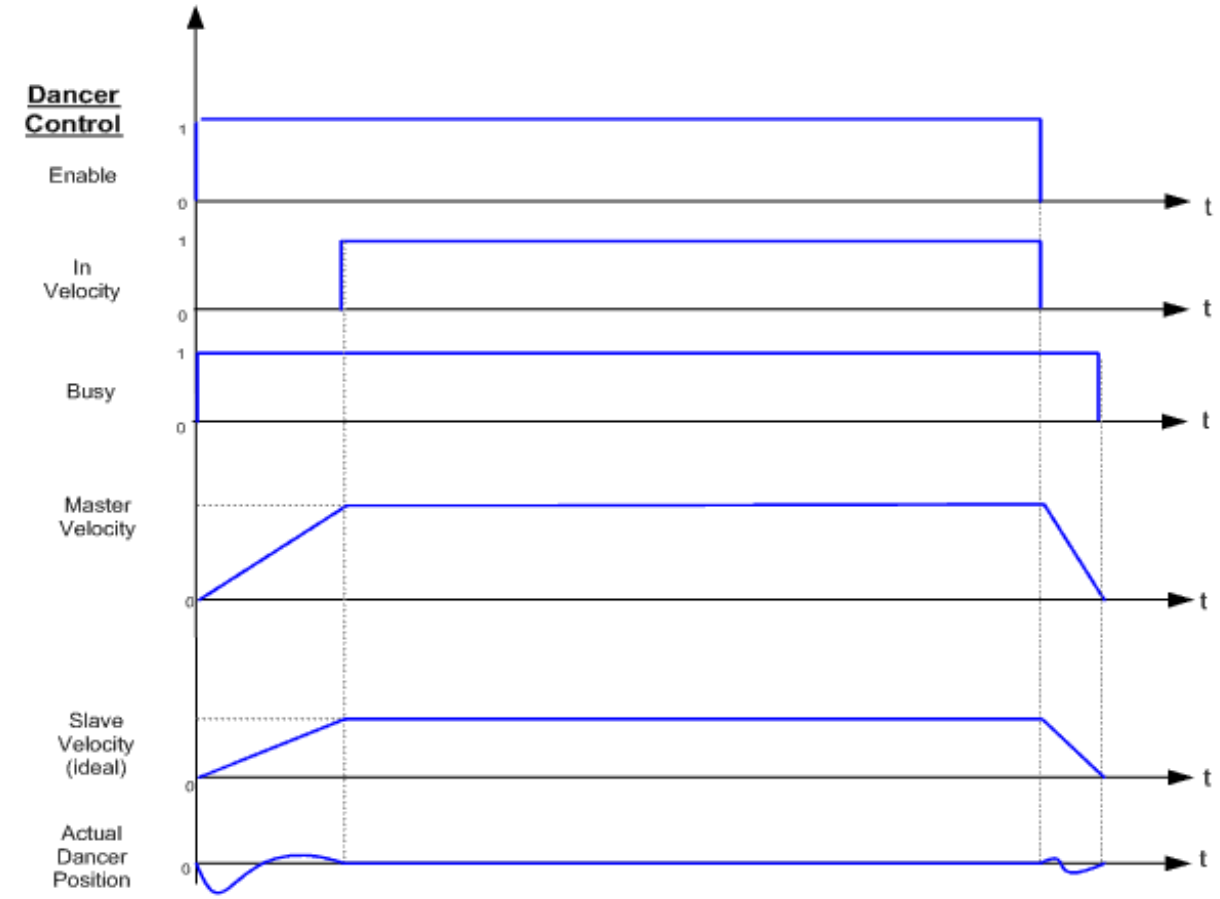


Figure: PS_Dancer_Control_Timing_Diagram

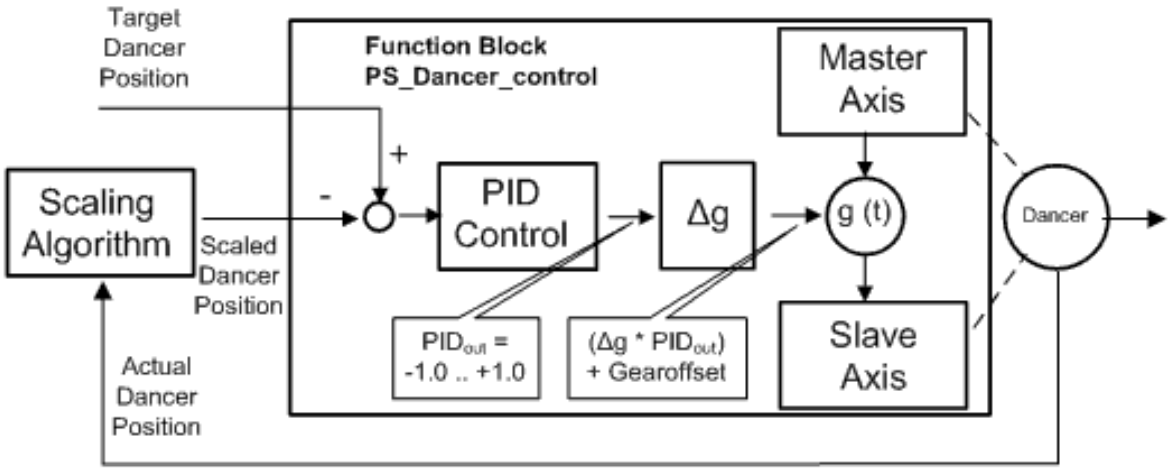


Figure: PS_Dancer_Control_Structure_Diagram

VAR_INPUT

```
VAR_INPUT
    Enable          : BOOL;     (* Start at high level *)
    TensionInput    : LREAL;
    Dancer_CTL      : DANCER_REF;
END_VAR
```

Enable : Start at high level, stop at low level.

TensionInput : Input signal for tension of the Web, usually represented by the relative actual dancer position sensor.

Dancer_CTL: Structure DANCER_REF [▶ 11] with Dancer Controller Parameters.

VAR_OUTPUT

```
VAR_OUTPUT
    Busy        : BOOL;   (* function block is currently busy *)
    Error       : BOOL;   (* Signals that an error has occurred within Function Block *)
    ErrorID     : UDINT;  (* Error identification *)
END_VAR
```

Busy: Executing status

Error : Becomes TRUE, as soon as an error occurs.

ErrorID : If the error output is set, this parameter supplies the error number

VAR_IN_OUT

```
VAR_IN_OUT
    Master      : AXIS_REF;      (* axis structure *)
    Slave       : AXIS_REF;      (* axis structure *)
END_VAR
```

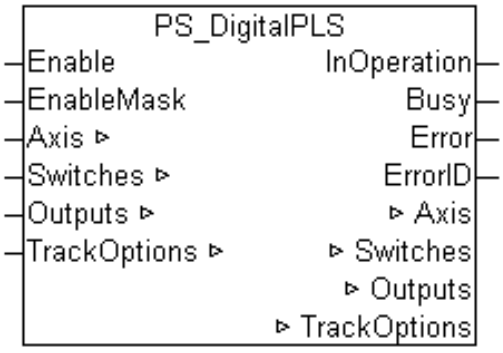
Master : Master axis structure [▶ 11].

Slave : Slave axis structure [▶ 11].

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.2.4 PS_DigitalPLS



This function block is the analogy to switches on a motor shaft: it commands a group of discrete output bits to switch in analogy to a set of mechanical cam controlled switches connected to an axis. Forward and backward movements are allowed.

Note that it is intended to keep this FB compatible with PLCopen MC FB Extensions (Part 2) MC_CamSwitch function Block.

Note **DIGPLS_REF is a supplier specific reference to the pattern data.**

Note **OUTPUT_REF is a supplier specific structure linked to the (physical) outputs.**

Note **TRACK_REF is supplier specific structure containing the track properties, e.g. the compensation per track (a track is a set of switches related to one output).**

This definition of cams has a start and an end position. So the user can define each single cam, that has a **FirstOnPosition** and a **LastOnPosition** (or time). In addition to a mechanical cam it has the possibilities to set it for a certain time and to give it a time compensation and a hysteresis. If (FirstOnPosition > LastOnPosition) it gives an inverse cam switch, which is off only within these positions.

Example of CAMSWITCHREF:

Parameter	Type	Switch01	Switch02	Swtich03	Switch04	...	SwitchN
TrackNumber	INTEGER	1	1	1	2		
FirstOnPosition [u]	REAL	2000	2500	4000	3000		
LastOnPosition [u]	REAL	3000	3000	1000	-		
AxisDirection	INTEGER	1=Pos	2=Neg	0=Both	0=Both		
CamSwitchMode	INTEGER	0=Position	0=Position	0=Position	1=Time		
Duration	TIME	-	-	-	1350		

Note **Values are Examples**

Example:

This example uses the values from the example for DIGPLS_REF above. It uses no On/OffCompensation, nor hysteresis.This is the behavior of the outputs, when the axis is moving continuously in positive direction. The axis is a modulo axis with a modulo length of 5000 u.

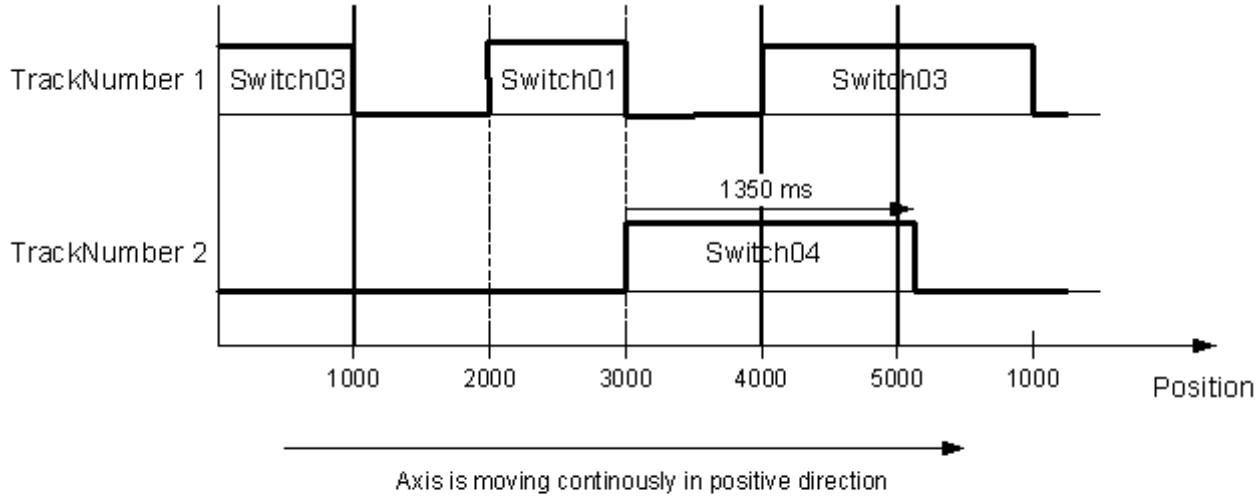


Figure : DigitalPLS timing diagram

Detailed description of Switch01.

This example additionally uses OnCompensation -125ms and OffCompensation +250ms.

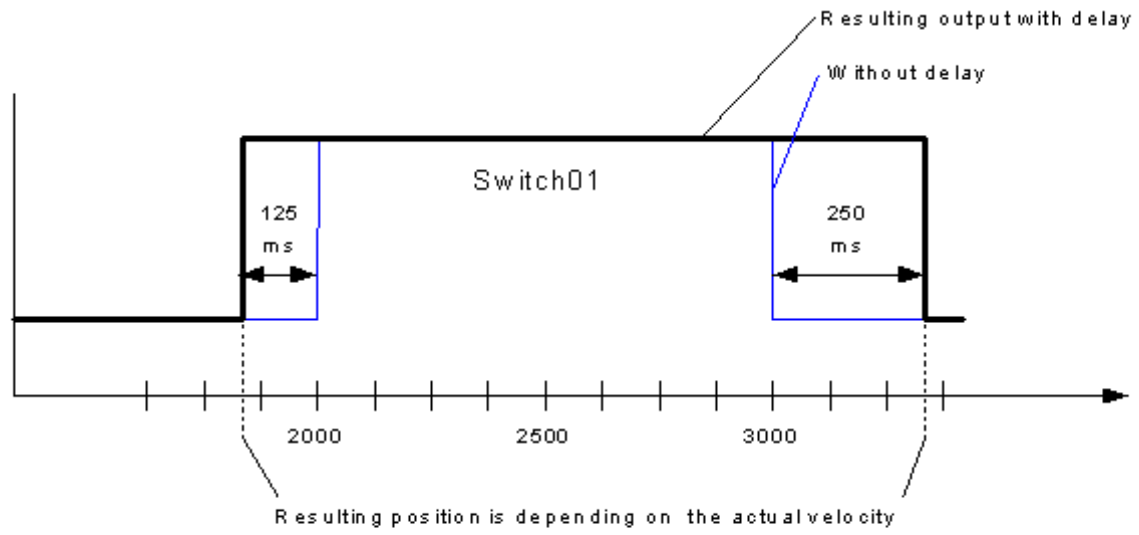


Figure : Digital PLS, detailed description of Switch01

This is the behavior of the outputs, when the axis is moving continuously in negative direction without On- or OffCompensation and without Hysteresis.

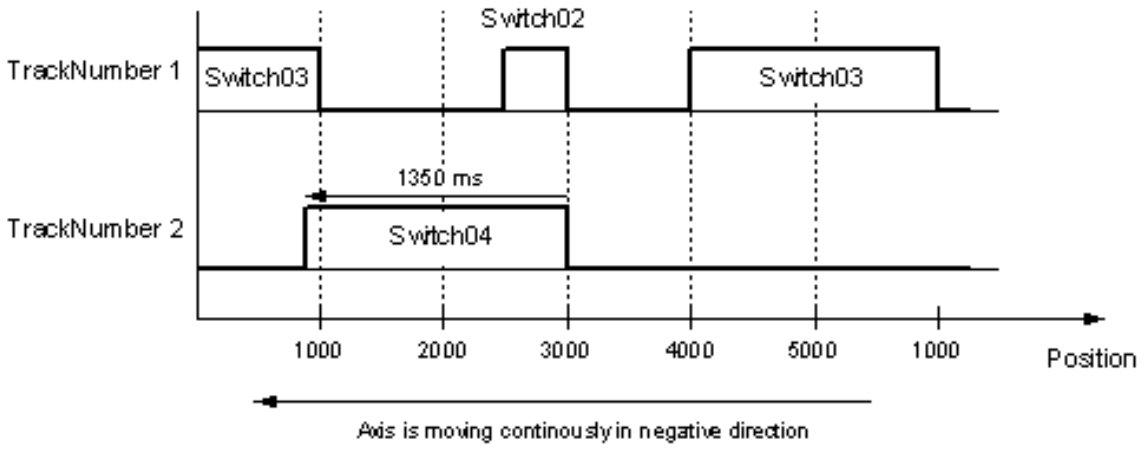


Figure : Ditigal PLS duration switch timing diagram

VAR_INPUT

```
VAR_INPUT
    Enable      : BOOL;
    EnableMask  : DWORD;
END_VAR
```

Enable : Decrements the batch counter to zero on R_Trig.

EnableMask : Resets batch counter to zero

VAR_OUTPUT

```
VAR_OUTPUT
    InOperation : BOOL;
    Busy        : BOOL;
    Error       : BOOL;
    ErrorID     : UDINT;
END_VAR
```

InOperation: The commanded tracks are enabled.

Busy: Shows that the function block is not finished.

Error: Signals that error has occurred within Function block.

ErrorID: Error Identification.

VAR_IN_OUT

```
VAR_IN_OUT
    Axis         : AXIS_REF;     (* Axis structure *)
    Switches     : DIGPLS_REF;   (* Counter *)
    Outputs      : OUTPUT_REF;   (* Counter *)
    TrackOptions: TRACK_REF;    (* Counter *)
END_VAR
```

Axis : Reference to the axis [▶ 11] to which the switches are connected to.

Switches : Reference to the switching actions [▶ 12].

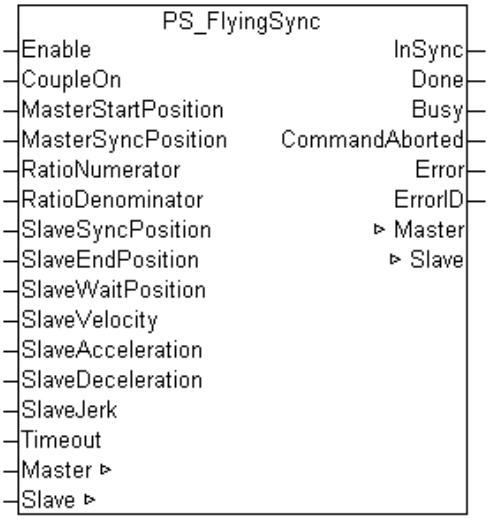
Outputs : Reference to the signal outputs [▶ 10], directly related to the referenced tracks. (max. 32 per function block) (First output = first TrackNumber).

TrackOptions : Reference to structure [▶ 14] containing track related properties, e.g. the ON and OFF compensations per output/track.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.2.5 PS_FlyingSync



The Flying Sync function provides a slave axis that can be synchronized to a moving master axis comparable to a “flying shears” or alternatively “flying cutter” functionality. The slave axis moves in synchronism with the master axis to perform machining processes. This kind of movement, synchronized to the master axis, allows to machine a work piece even while it is being transported.

The "Flying Sync" function is able to start synchronization of the slave even when the slave has already started, and is therefore no longer stationary. The "Flying Sync" function also calculates improved set value profiles, and these can be influenced by the user through a wide range of boundary conditions.

Note **Slave retracts after Slave reaches SyncEnd position to Wait Position.**

Note **Slave might either work in finite travel space (for flying shears function) by retracting to wait position.**

Note **Slave might work in infinite travel space (for cutter function) by traveling forward only.**

Note **If Slave Accel, Decel and Jerk are not supported, default values will apply.**

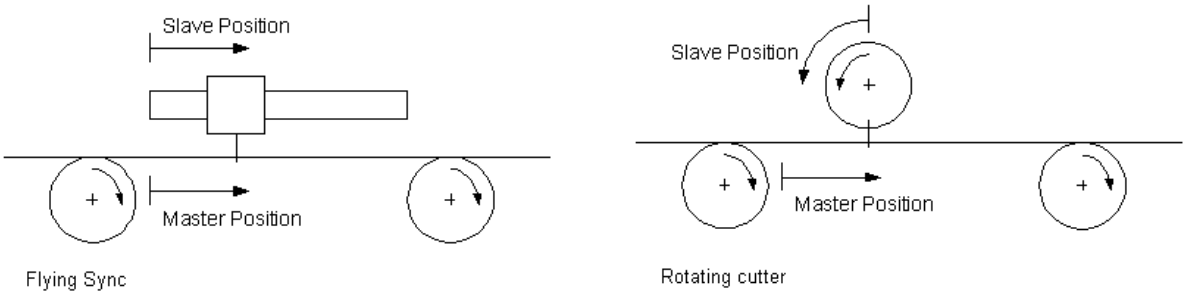


Figure : Flying Sync Geometries

The figures above give a sketch that shows the function to be solved with this function block. The primary function is to cut a passing continuous flow of material at specified positions into discrete pieces (i.e. products). Optionally, the FB allows to apply a Slave Wait Position larger than Slave End Position to create the function of a rotating cutter.

Both functions are quite similar, with the difference that for the Flying Shear the slave axis features a finite range of motion and for the rotating cutter the slave axis continues to move in one direction.

The motion can be split into two parts:

1. Synchronization between master and slave axis and synchronous motion of both axes (where the actual task is performed)
2. Slave axis moves to a defined waiting position and waits for next action request

For the first part, the master and slave axes have to operate at synchronous speeds while maintaining a specified phase relationship (e.g. to assure an accurate cutting point). With this function block, the sequence of actions can be easily generated. Figure 2 shows a timing diagram of this process.

Signal Curve

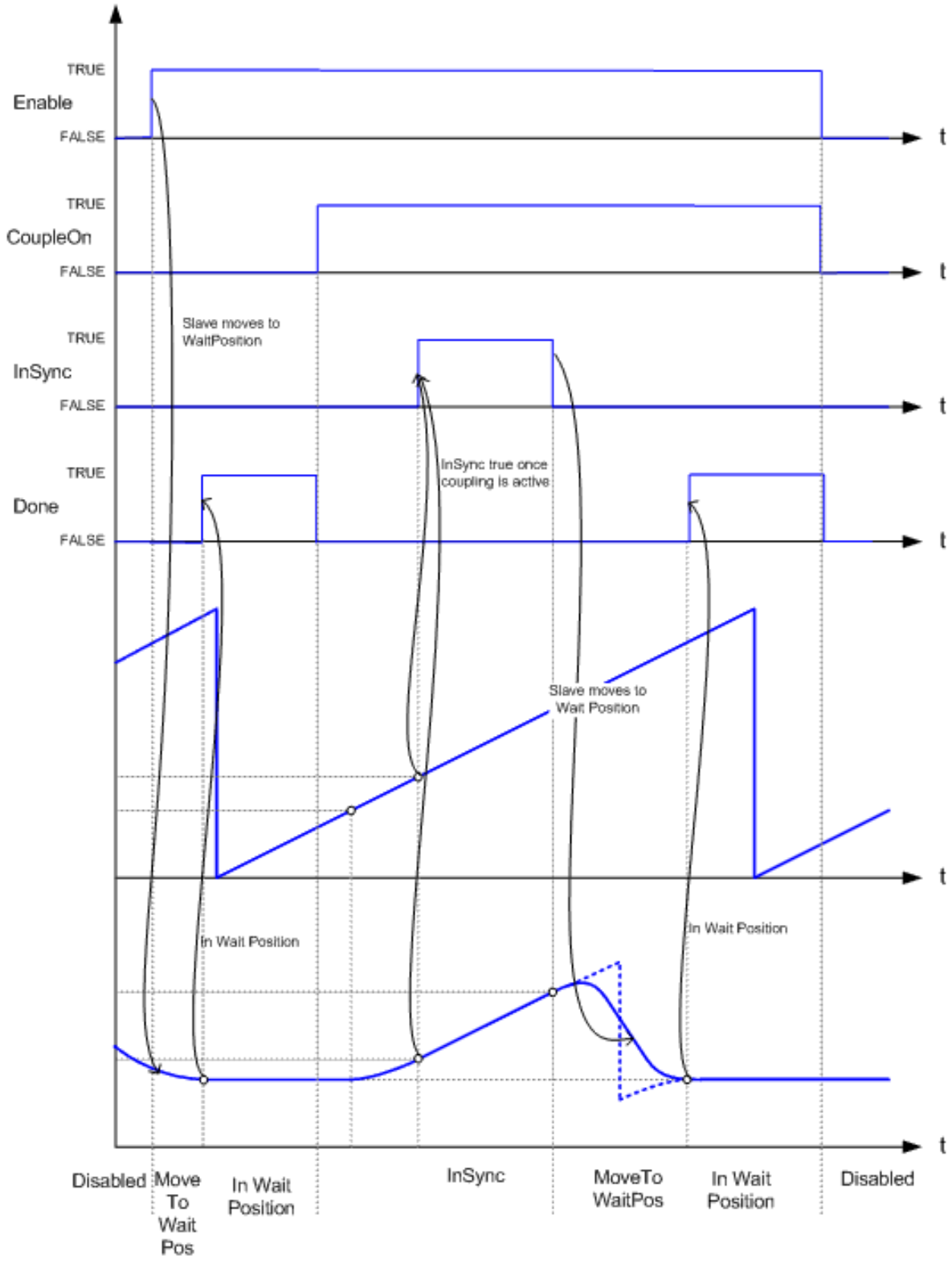


Figure : PS_FlyingSync timing diagram for a single synchronization

VAR_INPUT

```
VAR_INPUT
    Enable              : BOOL;
    CoupleOn            : BOOL;
    MasterStartPosition  : LREAL;
```

```
    MasterSyncPosition   : LREAL;
    RatioNumerator       : LREAL := 1;
    RatioDenominator     : UINT  := 1;
    SlaveSyncPosition    : LREAL;
    SlaveEndPosition     : LREAL;
    SlaveWaitPosition    : LREAL;
    SlaveVelocity        : LREAL;
    SlaveAcceleration    : LREAL;
    SlaveDeceleration    : LREAL;
    SlaveJerk            : LREAL;
    Timeout              : TIME;
END_VAR
```

Enable: Initiates the function block and moves slave initially to SlaveWaitPosition.

CoupleOn: Couple initiation, level sensitive -> loads MasterSyncPosition and SlaveSyncPosition for execution. On FALSE, coupling is disabled.

MasterStartPosition: Master position that determines the phase relation between master and slave axis [u].

MasterSyncPosition: Master position where synchronized motion starts [u].

RatioNumerator: Numerator for the gearing factor.

RatioDenominator: Denominator for the gearing factor.

SlaveSyncPosition: Corresponding slave position [u].

SlaveEndPosition: Slave position where synchronized part of motion ends [u].

SlaveWaitPosition: Slave position where slave axis waits [u].

SlaveVelocity: Value of the maximum slave velocity (always positive) (not necessarily reached) [u/s].

SlaveAcceleration: Value of the acceleration (always positive) (increasing energy of the motor) [u/s^2].

SlaveDeceleration: Value of the deceleration (always positive) (decreasing energy of the motor) [u/s^2].

SlaveJerk: Value of the Jerk [u/s^3].(always positive).

Timeout: supervisory timeout.

VAR_OUTPUT

```
VAR_OUTPUT
    InSync          : BOOL;
    Done            : BOOL;
    Busy            : BOOL;
    CommandAborted  : BOOL;
    Error           : BOOL;
    ErrorID         : UDINT;
END_VAR
```

InSync: True if slave is in sync with Master.

Done: True once Slave is in wait position.

Busy: True once Slave is synchronizing.

CommandAborted: True once Command is aborted.

Error: Signals that an error has occurred within Function Block, e.g. new command while still executing.

ErrorID: Error Identification.

VAR_IN_OUT

```
VAR_IN_OUT
    Master      : AXIS_REF;      (* Axis structure *)
    Slave       : AXIS_REF;      (* Axis structure *)
END_VAR
```

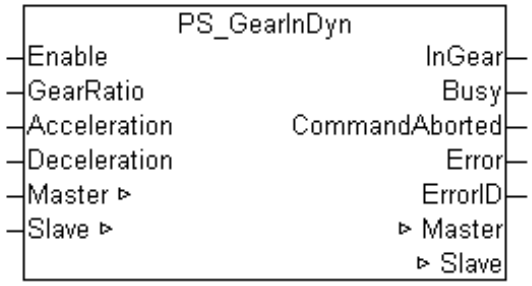
Master: Reference to axis to be coupled to as Master for a flying saw.

Slave: Reference to axis to be coupled as a flying saw.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.2.6 PS_GearInDyn



PS_GearInDyn activates a linear master-slave coupling (gear coupling). The gear ratio can be modified in every control cycle. The block is therefore suitable for velocity controlled master slave couplings. The acceleration parameter limits the acceleration of the slave in case of high gear ratio changes.

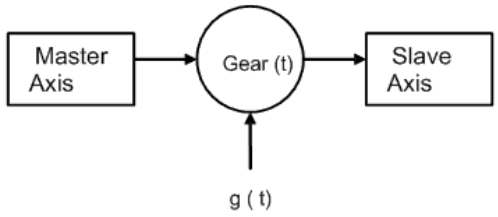


Figure : PS_GearInDyn structure of gear model

VAR_INPUT

```
VAR_INPUT
    Enable      : BOOL;
    GearRatio       : LREAL;
    Acceleration    : LREAL;
    Deceleration    : LREAL;
END_VAR
```

Enable : Coupling initiation, level sensitive. Initiates coupling and maintains during high, aborts on low signal.

GearRatio : Gearing factor. The gear ratio may be changed in every PLC cycle. 0.0 is invalid, default is 1.0.

Acceleration : Limit to acceleration of slave due to gear ration changes.

Deceleration : Limit to deceleration of slave due to gear ration changes.

VAR_OUTPUT

```
VAR_OUTPUT
    InGear      : BOOL;
    Busy        : BOOL;
    CommandAborted  : BOOL;
    Error       : BOOL;
    ErrorID         : UDINT;
END_VAR
```

InGear: True if coupling is being successfully performed.

Busy: Becomes TRUE, as soon as the block is active. Busy becomes TRUE with a rising edge at Enable and becomes FALSE again after the block finished or aborted its operation. A high level at Enable will then be accepted.

CommantedAborted: True if slave is decoupled during Enable is TRUE.

Error: Signals that an error has occurred within Function Block, e.g. new command while still executing.

ErrorID: Error Identification.

VAR_IN_OUT

```
VAR_IN_OUT
    Master      : AXIS_REF;      (* Axis structure *)
    Slave       : AXIS_REF;      (* Axis structure *)
END_VAR
```

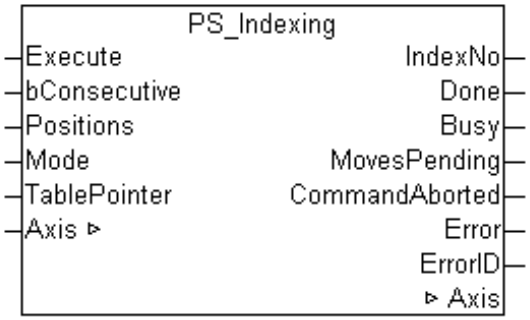
Master : Reference to axis to be coupled to as Master.

Slave : Reference to axis to be coupled.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.2.7 PS_Indexing



This function block will – upon R_Trig on Execute – do a number of relative moves, listed in a table (Struct or Array). The FB will position the axis to a complete stop at target position and continue with the next move from the table upon next R_Trig signal.

NOTE

This function block will – upon R_Trig on Execute – do a number of relative moves 1 - N, listed in a table (Struct or Array). The FB will position the axis to a complete stop at target position n and continue with the next move from the table upon next Execute signal.

Modes of operation are:

Mode: 0 pass through table elements once until finding zero distance entry,

Mode: 1 cycle through table, starting at first element on finding a zero distance entry,

Mode: 2 position relative for distance of table element N (Table_pointer)

The FB will complete a singular move if possible. In case a second Execute rising edge triggers the next move, the FB will not decelerate to stop but continue to new target position immediately. Inputs are updated on rising edge of Execute.

Table Positions:

Position 1	Velocity 1	Acceleration 1	Deceleration 1	Jerk 1
Position 2	Velocity 2	Acceleration 2	Deceleration 2	Jerk 2
...	...	...	...	...
Position N	Velocity N	Acceleration N	Deceleration N	Jerk N

: PS_Indexing Position table

Signal curve

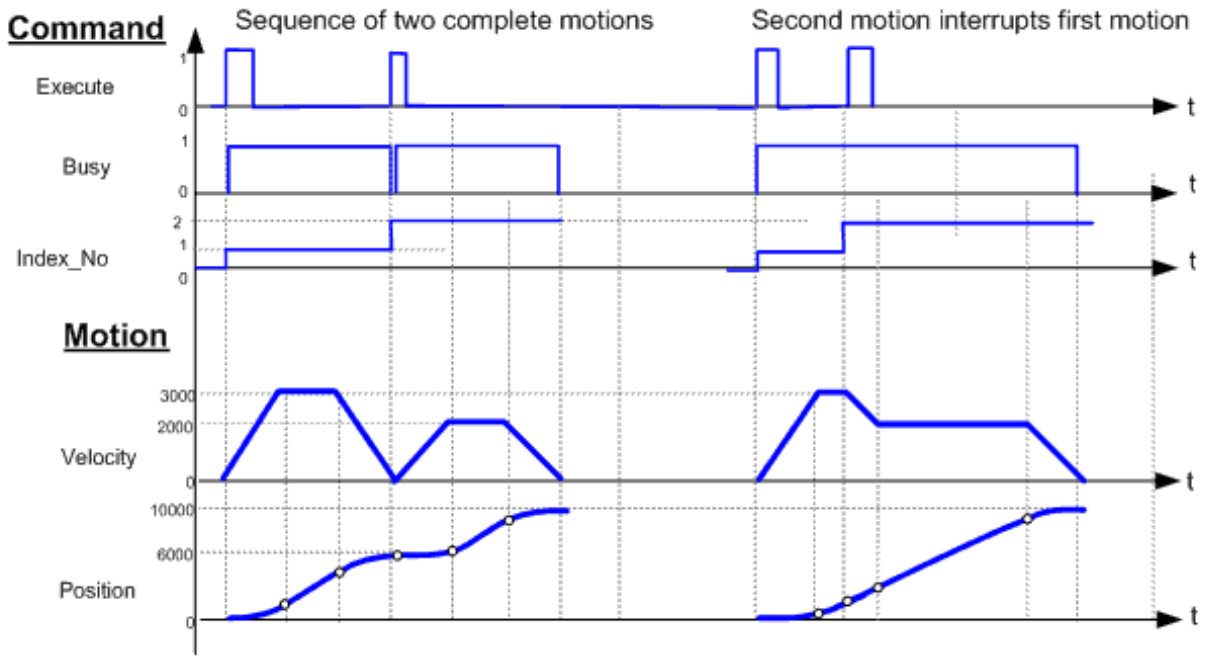


Figure: PS_Indexing timing diagram

VAR_INPUT

```
VAR_INPUT
    Execute          : BOOL;
    bConsecutive     : BOOL;
    Positions        : POS_REF;
    Mode             : INT;
    TablePointer     : INT;
END_VAR
```

POS_REF [▶ 13]

Execute : Start the next relative positioning by a rising edge.

bConsecutive : If True, commands in table are not interrupt. Commands are executed one after the other. If False, command is interrupted on arrival of next command.

Positions : Reference to Structure or Array with relative move distances listed. Typical Type of Positions: LREAL

Mode : Mode of Indexing, see notes

TablePointer : Pointer to element n of Table

VAR_OUTPUT

```
VAR_OUTPUT
    IndexNo          : INT;       Done        : BOOL; (* function block is done *)
    Busy             : BOOL; (* function block is currently busy *)
    MovesPending     : UDINT;
    CommandAborted   : BOOL;
```

```
    Error           : BOOL; (* Signals that an error has occurred within Function Block *)
    ErrorID         : UDINT;    (* Error identification *)
END_VAR
```

IndexNo: Index executing or last index completed.

Done: True once Indexing is done.

Busy: This output remains TRUE until the block has executed a command

MovesPending: Commands to be executed; If commands are executed one after another without interruption.

CommandAborted: True once command is aborted.

Error : Becomes TRUE, as soon as an error occurs.

ErrorID : If the error output is set, this parameter supplies the error number.

VAR_IN_OUT

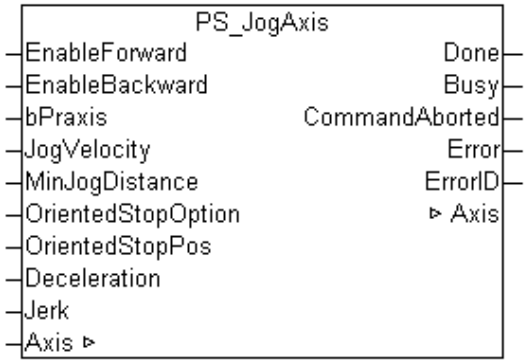
```
VAR_IN_OUT
    Axis        : AXIS_REF;      (* axis structure *)
END_VAR
```

Axis : Axis structure.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.2.8 PS_JogAxis



This function block jogs an axis for N increments (represented in t.u.) forward or backward with the selected (manual operation) jog velocity and acceleration. N must satisfy certain minimum count limitation, otherwise no movement is executed.

NOTE

This function block will – after rising edge on Enable_Forward or Enable_Backward - start a continuous move (at least for the minimum distance) and continue upon high-level on these inputs with a continuous motion – until they are FALSE – then, on their falling edge, the axis is regularly decelerated to stop. The movement is carried out on Jog velocity for the minimum distance or longer.

In case both Enable signals are high, the FB will assume the result to be low as in an EXOR conjunction.

Signal Curve

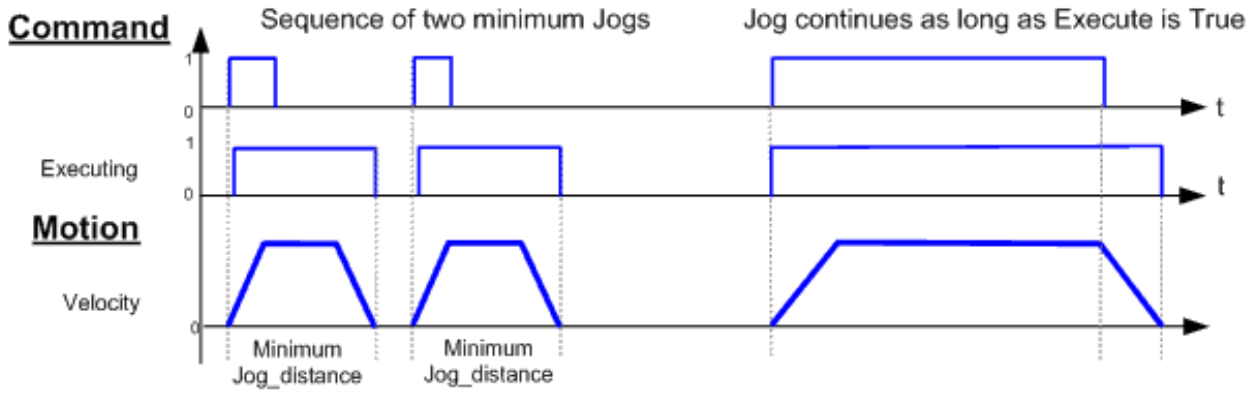


Figure : PS_Jog Axis timing diagram

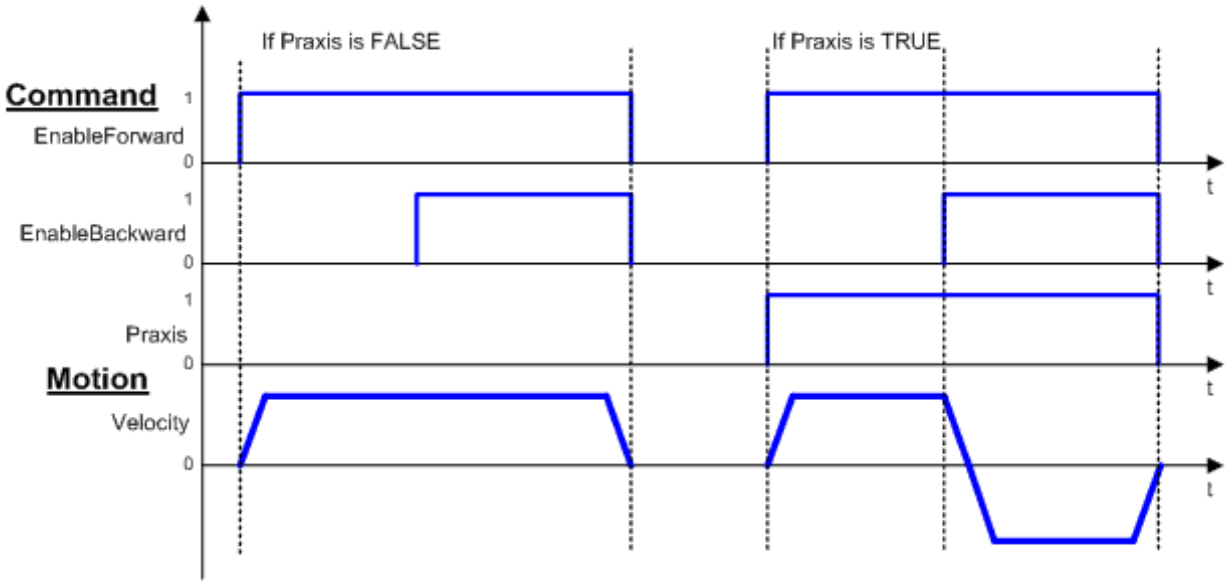


Figure : PS_Jog Axis, Praxis timing diagram

VAR_INPUT

```
VAR_INPUT
    EnableForward       : BOOL;
    EnableBackward      : BOOL;
    bPraxis             : BOOL;
    JogVelocity         : LREAL;
    MinJogDistance      : LREAL;
    OrientedStopOption  : BOOL;
    OrientedStopPos     : LREAL;
    Deceleration        : LREAL;
    Jerk                : LREAL;
END_VAR
```

EnableForward : Executes a sequence of a relative move Forward during high and a stop at signal = low.

EnableBackward : Executes a sequence of a relative move Backward during high and a stop at signal = low.

bPraxis : Two different modes available. Please see the signal curves.

JogVelocity : Velocity to jog (in t.u. [u]).

MinJogDistance : Minimum distance to jog (in t.u. [u]).

OrientedStopOption : If True, Oriented Stop performed at OrientedStopPos. If False, immediate stop performed.

OrientedStopPos : Oriented position for oriented stop.

Deceleration : Deceleration for stop.

Jerk : Jerk for stop.

VAR_OUTPUT

```
VAR_OUTPUT
    Done            : BOOL;
    Busy            : BOOL;
    CommandAborted  : BOOL;
    Error           : BOOL;
    ErrorID         : UDINT;
END_VAR
```

Done: True if jogging is done.

Busy: True if jogging is being performed .

CommantedAborted: True if Jog command is aborted.

Error: Signals that an error has occurred within Function Block, e.g. new jog command while still executing.

ErrorID: Error Identification.

VAR_IN_OUT

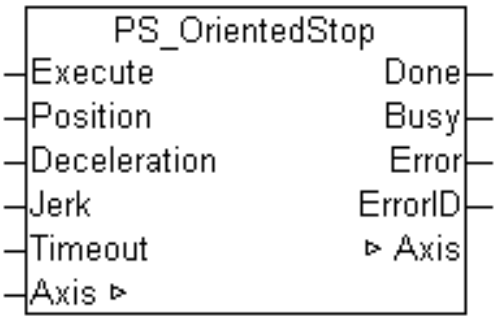
```
VAR_IN_OUT
    Axis        : AXIS_REF;      (* Axis structure *)
END_VAR
```

Axis : Reference to axis to be jogged

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.2.9 PS_OrientedStop



PS_OrientedStop is used to affect an "oriented" axis stop at the specified modulo position. Depending on the speed and the dynamic parameters of the axis, it will stop at the next possible oriented position.

For convenience in packaging, a stop with a resulting orientation of tool or product is often utilized. Therefore, this stop command calculates the deceleration ramp to final stop but will end always in the desired modulo position.

NOTE

The error output must be analyzed, because under certain conditions an oriented stop is not possible. For example, a standard stop may have been triggered just before, or an oriented stop would cause an active software limit switch to be exceeded. For all fault conditions, the axis is stopped safely, but it may subsequently not be at the required oriented position.

The position must be in the range [0 ≤ position < modulo period]. With a modulo period of 360 degrees, for example, 360 is therefore not a valid position.A value of 0 has to be specified instead.

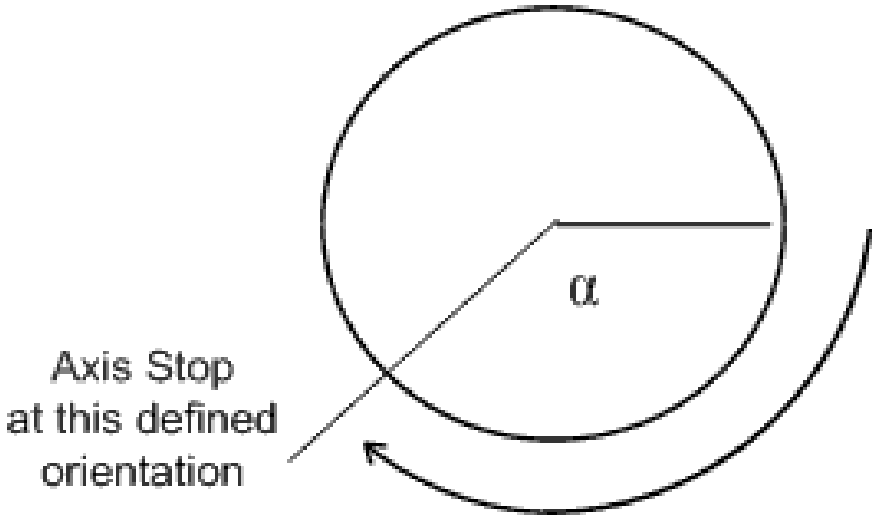


Figure : PS_OrientedStop graphical explanation of behavior

VAR_INPUT

```
VAR_INPUT
    Execute         : BOOL;
    Position        : LREAL;
    Deceleration    : LREAL;
    Jerk            : LREAL;
    Timeout         : TIME;
END_VAR
```

Execute : Stop initiation, rising edge trigger.

Position : Absolute modulo destination position at which the axis should stop.

Deceleration : Limit to deceleration of axis during decel ramp. This input is optional. The default deceleration specified at the start of the axis is used for stopping.

Jerk : Limit to jerk of of axis during decel ramp. This input is optional. The default jerk specified at the start of the axis is used for stopping.

Timeout : Supervisory timeout.

VAR_OUTPUT

```
VAR_OUTPUT
    Done        : BOOL;
    Busy        : BOOL;
```

```
    Error       : BOOL;
    ErrorID     : UDINT;
END_VAR
```

Done: True if stop was successfully performed and the axis has come to a complete stop.

Busy: True if coupling FB is being executed.

Error: Signals that an error has occurred within Function Block, e.g. new command while still executing.

ErrorID: Error Identification.

VAR_IN_OUT

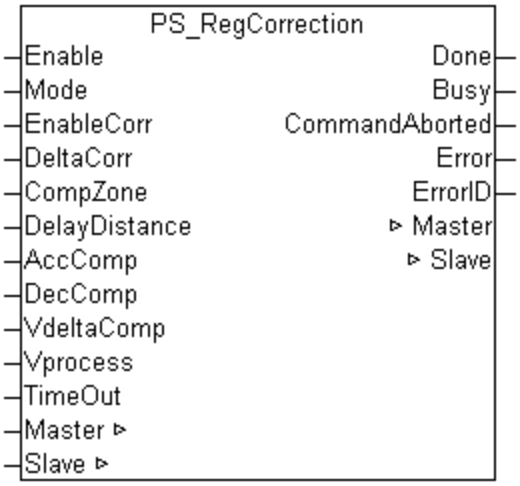
```
VAR_IN_OUT
    Axis        : AXIS_REF;      (* Axis structure *)
END_VAR
```

Axis : Reference to axis to be stopped.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.2.10 PS_RegCorrection



This function block corrects a distance offset, calculated after capturing an axis position, e.g. with the “Registration” FB, on receiving a rising edge execute command. The captured position is often referred to as “Latch Position”. The difference to the ideal position is the input “delta correction distance” to this FB.

Note This correction FB executes after e.g. a registration FB captures apposition and the delta to a desired position has been determined by this or other methods. The mode selector allows the user to choose different ways of correction. Partial implementation of the solutions is possible; in this case, selection of the other methods results in an error.

Mode: 0 (Offset correction) In this case, the DeltaCorr distance is directly applied to the Master Axis by means of a relative offset of the zero position, as used for CAM axes.

Mode: 1 (Distance correction) The DeltaCorr distance is used to calculate a single super-positioned relative move of the slave axis to correct the delta correction distance.

Other correction methods may apply, e.g. by dynamic gear factor, and could be implemented as supplier specific enhancements.

Vprocess, the actual master of process velocity, is scanned in every cycle of the execution of the Function Block. Other inputs are scanned once at start of FB.

Signal curve

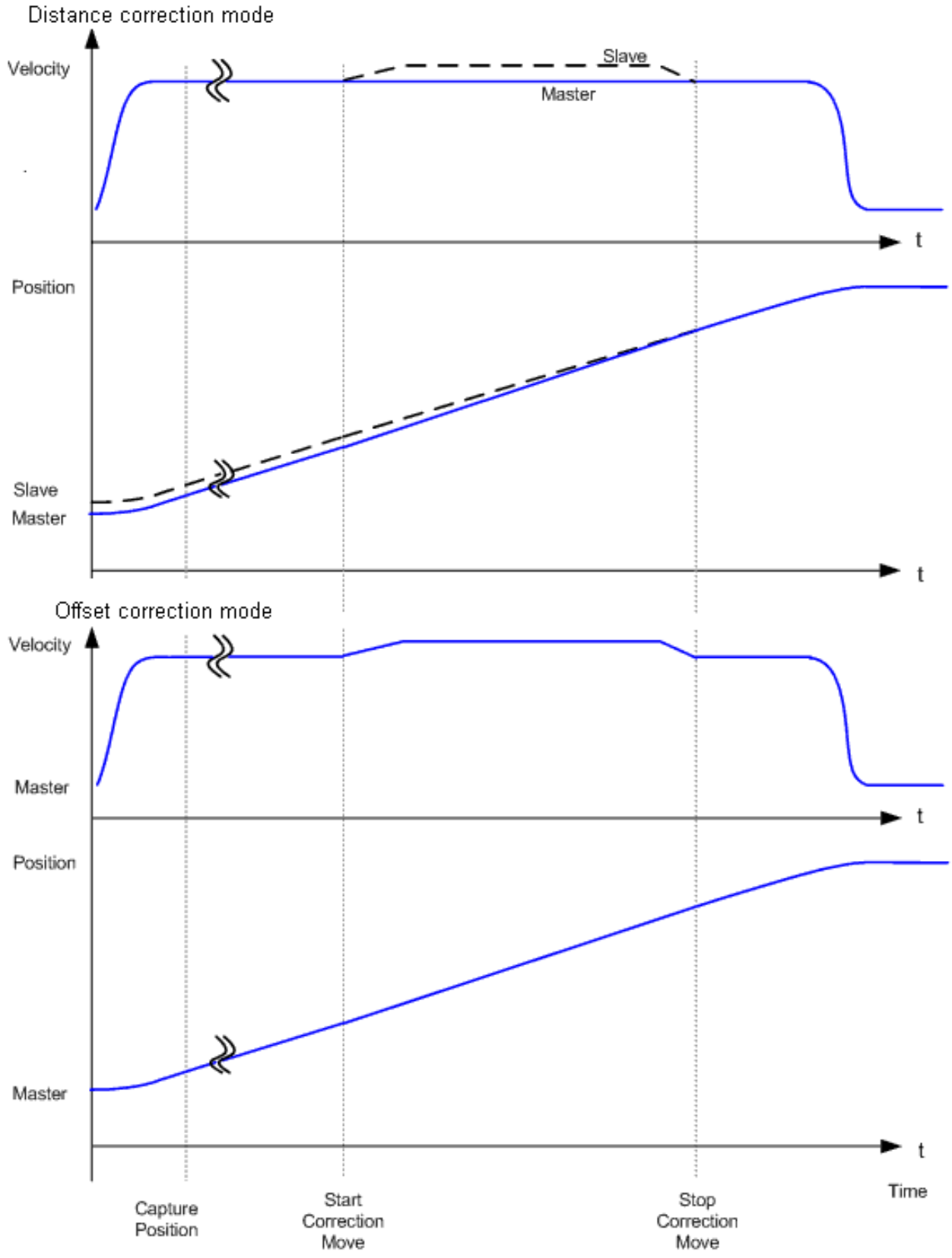


Figure: PS_RegCorrection timing diagram

blue: Master
black: Slave

Fig. 1: PS_RegCorrection_Timing Diagram

VAR_INPUT

```
VAR_INPUT
    Enable          : BOOL;     (* Start at high level *)
    Mode            : BYTE;     (* Mode 0: Offset correction, Mode 1: Distance correction *)
    EnableCorr      : BOOL;
    DeltaCorr       : LREAL;
    CompZone        : LREAL;
    DelayDistance   : LREAL;
    AccComp         : LREAL;
    DecComp         : LREAL;
    VdeltaComp      : LREAL;
    Vprocess        : LREAL;
    TimeOut         : TIME;
END_VAR
```

Enable : Start the Correction FB, level triggered

Mode : Correction method selector input. Two methods defined, other subject to supplier implementation

EnableCorr : Level sensitive, TRUE during correction motion

DeltaCorr : Correction distance (relative position) for correction move

CompZone : Distance range of master in which correction move is to complete

DelayDistance : Distance of master to move before correction starts

AccComp : Max. deceleration for compensation [u/s²]

DecComp : Max. deceleration for compensation [u/s²]

VdeltaComp : Max. velocity for compensation move [u/s].

Vprocess : Actual process (master) velocity [u/s].

TimeOut : Timeout supervision of the compensation move.

VAR_OUTPUT

```
VAR_OUTPUT
    Done        : BOOL;  (* function block is done *)
    Busy        : BOOL;  (* function block is currently busy *)
    Error       : BOOL;  (* Signals that an error has occurred within Function Block *)
    ErrorID     : UDINT; (* Error identification *)
END_VAR
```

Done: This output changes from FALSE to TRUE once the correction motion is correctly done. It resets with the rising edge of EnableCorr or with Enable = FALSE.

Busy: This output remains TRUE as long as Function Block executes

Error : Becomes TRUE, as soon as an error occurs.

ErrorID : If the error output is set, this parameter supplies the error number.

VAR_IN_OUT

```
VAR_IN_OUT
    Master      : AXIS_REF;      (* axis structure *)
    Slave       : AXIS_REF;      (* axis structure *)
END_VAR
```

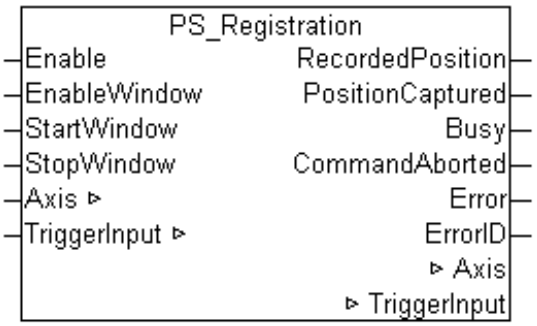
Master : Master axis structure.

Slave : Slave axis structure.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.2.11 PS_Registration



This function block captures an axis position on receiving a rising edge execute command. The captured position is often referred to as “Latch Position”.

Note This FB is a subset of the RegistrationCorrection FB. Registration is often referred to as “Probing”. Some Motion Devices have dedicated “Registration” or “Probe” Inputs to quickly latch the position on the fly.

Enable activates the Function Block, **RecordedPosition** is scanned in every cycle of the Function Block execution, other inputs in first cycle only.

For further details please see MC_TouchProbe.

Signal curve

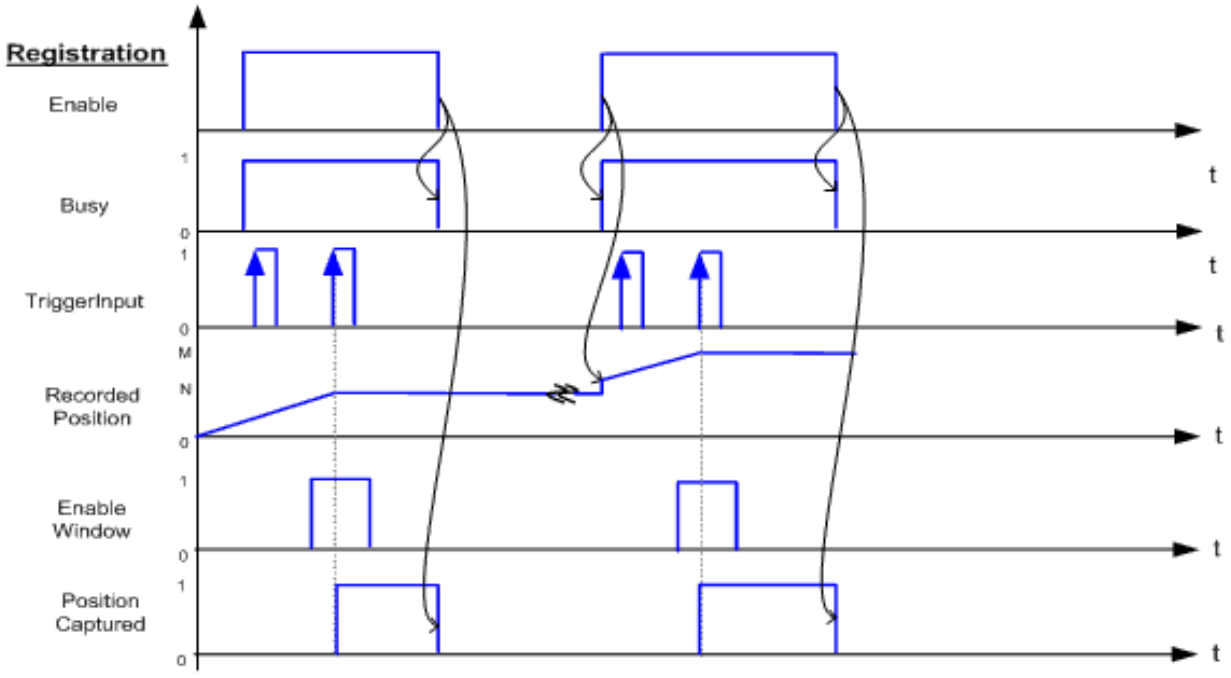


Figure: PS_Registration timing diagram

VAR_INPUT

```
VAR_INPUT
    Enable          : BOOL;     (* Start the Position Latch at high level *)
    ActualPosition  : POS_REF;  (* Actual Position of the Axis defined by Axis. *)
    EnableWindow    : BOOL;     (* Enable an “capture input” window *)
```

```
    StartWindow     : LREAL;    (* Start position of capture window zone *)
    StopWindow      : LREAL;    (* Stop position of capture window zone *)
END_VAR
```

Enable: Start the Position Latch at high level

ActualPosition: Actual Position of the Axis defined by Axis.

EnableWindow: Enable an “capture input” window, in which input trigger events are captured.

StartWindow: Start position of capture window zone in technical units [u].

StopWindow: Stop position of capture window zone in technical units [u].

VAR_OUTPUT

```
VAR_OUTPUT
    PositionCaptured  : BOOL; (* Trigger event recorded *)
    Busy              : BOOL; (* function block is currently busy *)
    CommandAborted    : BOOL;
    Error             : BOOL; (* Signals that an error has occured within Function Block *)
    ErrorID           : UDINT;    (* Error identification *)
    RecordedPosition  : LREAL;    (* Position where the trigger event occurred *)
END_VAR
```

PositionCaptured: Becomes TRUE, if an axis position has been recorded successfully. The position is sent to the output *RecordedPosition*.

Busy: Becomes TRUE as soon as the function block is active, and becomes FALSE when it has returned to its initial state.

CommandAborted: Becomes TRUE if the process is interrupted by an external event.

Error: Becomes TRUE, as soon as an error occurs.

ErrorID: If the error output is set, this parameter supplies the error number.

RecordedPosition: Latch Position, where the trigger event occurred (in technical units [u])

VAR_IN_OUT

```
VAR_IN_OUT
    Axis         : AXIS_REF;      (* Identifies the axis which position shall be latched at the trig
ger event *)
    TriggerInput : INPUT_REF;      (* Reference to the trigger signal source. *)
END_VAR
```

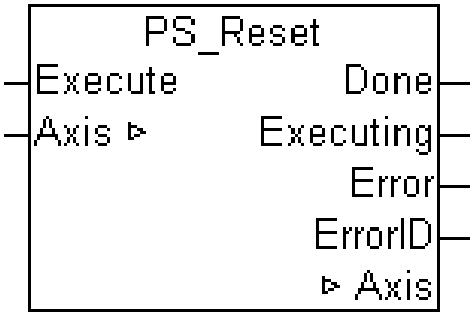
Axis: Axis structure.

TriggerInput: Reference to the trigger signal source. Trigger signal may be specified by the axis reference. Data structure of type INPUT_REF, that describes the trigger input for the recording of the axis position.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.2.12 PS_Reset



An axis reset is carried out with the function block.

VAR_INPUT

```
VAR_INPUT
    Execute     : BOOL;     (* Start at high level *)
END_VAR
```

Execute : Start the Correction FB, rising edge triggered

VAR_OUTPUT

```
VAR_OUTPUT
    Done        : BOOL; (* function block is done *)
    Executing   : BOOL; (* function block is currently busy *)
    Error       : BOOL; (* Signals that an error has occurred within Function Block *)
    ErrorID     : UDINT;    (* Error identification *)
END_VAR
```

Done: Becomes TRUE, if the states were determined successfully.

Executing: This output remains TRUE as long as Function Block executes

Error : Becomes TRUE, as soon as an error occurs.

ErrorID : If the error output is set, this parameter supplies the error number.

VAR_IN_OUT

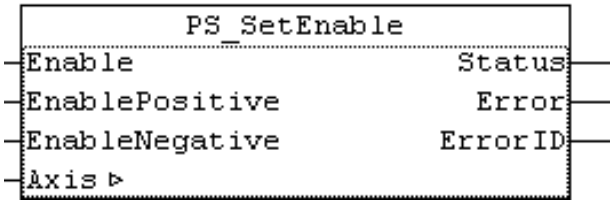
```
VAR_IN_OUT
    Axis        : AXIS_REF;     (* axis structure *)
END_VAR
```

Axis : Axis structure.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.2.13 PS_SetEnable



This function block sets the enable flags for the axis controller, the feed forward and feed backward of an NC axis.

VAR_INPUT

```
VAR_INPUT
    Enable          : BOOL;
    EnablePositive  : BOOL;
    EnableNegative  : BOOL;
END_VAR
```

Enable: Enables the axis controller.

EnablePositive: Enables forward motion (rising position).

EnableNegative: Enables forward motion (falling position).

VAR_OUTPUT

```
VAR_OUTPUT
    Status      : BOOL;
    Error       : BOOL;
    ErrorID     : UDINT;
END_VAR
```

Status: Is TRUE, if the axis controller is enabled.

Error: Signals that error has occurred within Function block.

ErrorID: Error Identification.

VAR_IN_OUT

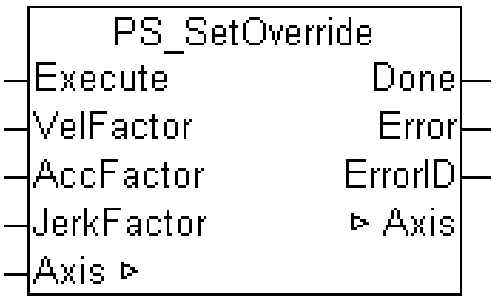
```
VAR_IN_OUT
    Axis        : AXIS_REF;      (* Axis structure *)
END_VAR
```

Axis: Identifies the axis to work upon.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10, build 1319 onwards	PC (i386)	TcPackALv3.0.Lib

3.2.14 PS_SetOverride



This function block sets the values of override. The override parameters act as a factor that is multiplied to the commanded velocity, acceleration, deceleration and jerk of the move function block.

Note that it is intended to keep this FB compatible with PLCopen MC FB Extensions (Part 2) MC_SetOverride function Block.

- The Input AccFactor acts on positive and negative acceleration (deceleration).
- This FB sets the factor. The override factor is valid until a new override is set.
- The default values of the override factor are 1.0.
- The value of the overrides can be between 0.0 and 1.0. The behavior of values > 1.0 is supplier specific. Values < 0.0 are not allowed. The value 0.0 is not allowed for AccFactor and JerkFactor.
- The value 0.0 set to the VelFactor stops the axis without bringing it to the state standstill.
- Override does not act on slave axes. (Axes in the state synchronized motion).
- The FB does not influence the state diagram of the axis.
- Override can be changed at any time and acts directly on the ongoing motion.
- Override overrides all other motion commands active in regards to velocity applied

Signal Curve

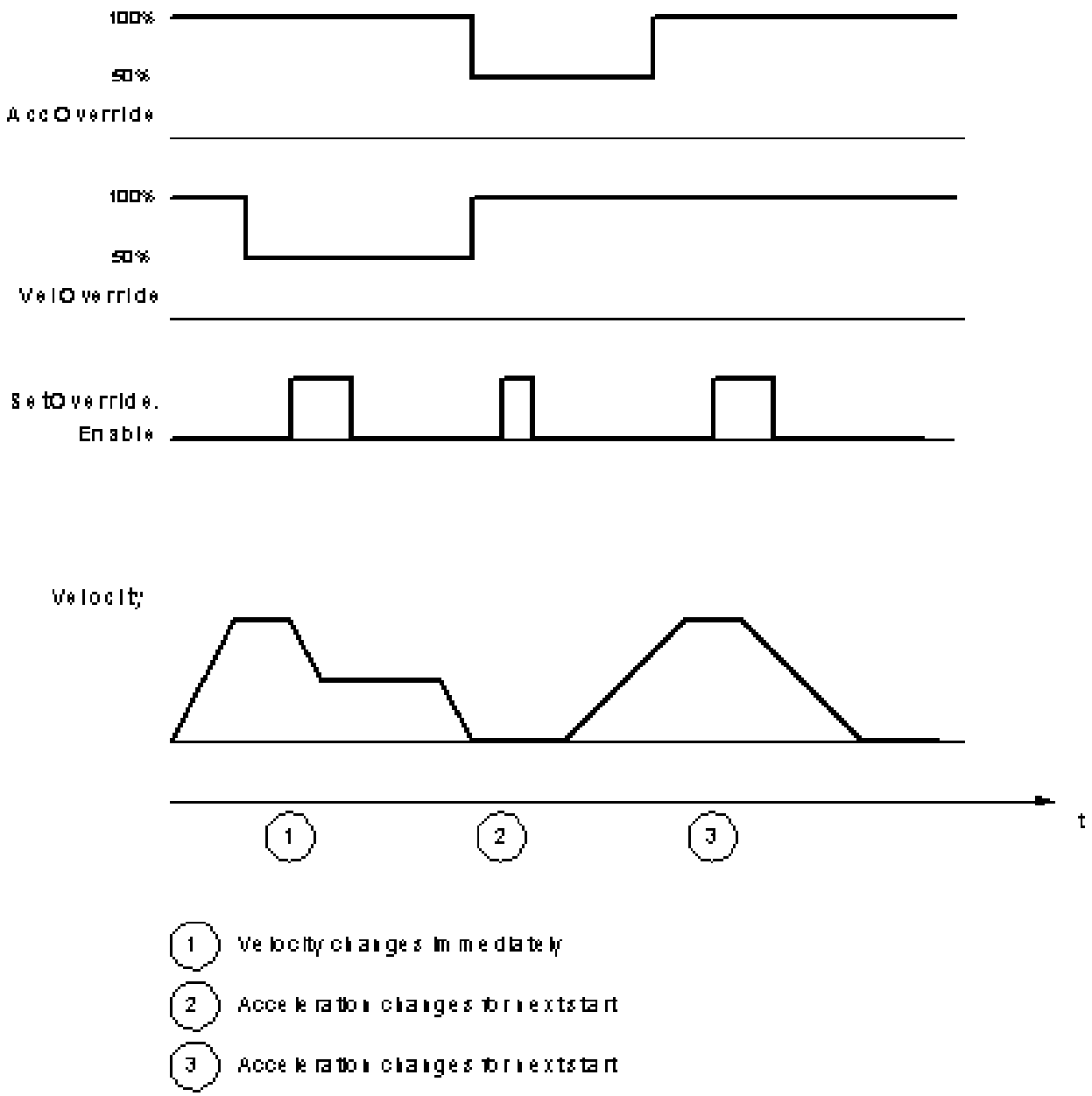


Figure : PS_SetOverride timing diagram

VAR_INPUT

```
VAR_INPUT
    Enable      : BOOL;
    VelFactor   : LREAL;
    AccFactor   : LREAL;
    JerkFactor  : LREAL;
    Timeout     : TIME;
END_VAR
```

Enable: Write the value of the override factor at rising edge.

VelFactor: New override factor for the velocity (e.g. 0…100 %).

AccFactor: New override factor for the acceleration/deceleration.

JerkFactor: New override factor for the jerk.

Timeout: timeout

VAR_OUTPUT

```
VAR_OUTPUT
    Done        : BOOL;
    Error       : BOOL;
    ErrorID     : UDINT;
END_VAR
```

Done: Signals that the override factor(s) is (are) set successfully.

Error: Signals that error has occurred within Function block.

ErrorID: Error Identification.

VAR_IN_OUT

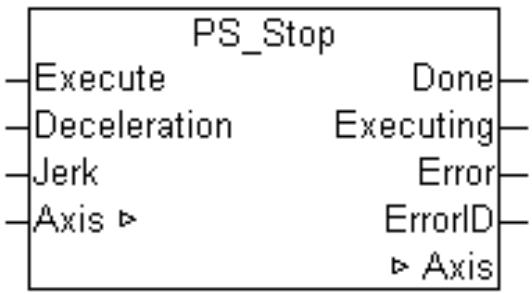
```
VAR_IN_OUT
    Axis        : AXIS_REF;      (* Axis structure *)
END_VAR
```

Axis : Identifies the axis to work upon.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.2.15 PS_Stop



An axis is stopped with the function block.

VAR_INPUT

```
VAR_INPUT
    Execute      : BOOL;
    Deceleration : LREAL;
    Jerk         : LREAL;
END_VAR
```

Execute: Start the Correction FB, rising edge triggered

Deceleration: Deceleration during brake travel. If the value is 0, the deceleration from the move command is used. Otherwise this value is supposed to be higher than the start value.

Jerk: Jerk during brake travel. If the value is 0, the jerk from the move command is used. Otherwise this value is supposed to be higher than the start value.

VAR_OUTPUT

```
VAR_OUTPUT
    Done        : BOOL; (* function block is done *)
    Executing   : BOOL; (* function block is currently busy *)
    Error       : BOOL; (* Signals that an error has occurred within Function Block *)
    ErrorID     : UDINT;    (* Error identification *)
END_VAR
```

Done: Becomes TRUE, once axis has finished its job and stands still.

Executing: This output remains TRUE as long as Function Block executes

Error: Becomes TRUE, as soon as an error occurs.

ErrorID: If the error output is set, this parameter supplies the error number.

VAR_IN_OUT

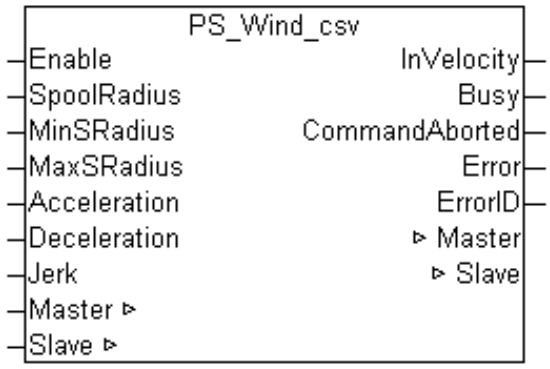
```
VAR_IN_OUT
    Axis        : AXIS_REF;      (* axis structure *)
END_VAR
```

Axis: Axis structure.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

3.2.16 PS_Wind_csv



This function block commands a controlled motion at a specified circumflex velocity for a wind / unwind axis. The circumflex is calculated from the radius of a wind/unwind spool, measured with a sensor.

Note This FB contains the mathematical calculation for coupling between a rotary slave axis and a translatory master axis. Its purpose is to generate and re-adjust a constant surface velocity (peripheral speed), relative to the master axis, for the rotary calibrated and controlled slave axis, depending on its spool diameter. Via a signal proportional to the spool radius, the spool radius of this slave axis is automatically evaluated and used for the calculation. This radius must never have the value 0.0 mm, since otherwise a calculation is no longer possible.

- The FB decelerates the axis to stop while winding above the MaxSRadius
- The FB decelerates the axis to stop while unwinding below the MinSRadius
- SpoolRadius needs to satisfy Min_S_Radius < SpoolRadius < MaxSRadius to execute the action, otherwise Error = True, ErrorID set
- Error ID is implementation specific

SpoolRadiusis scanned in every cycle of the Function Block execution, other inputs in first cycle only.

Signal curve

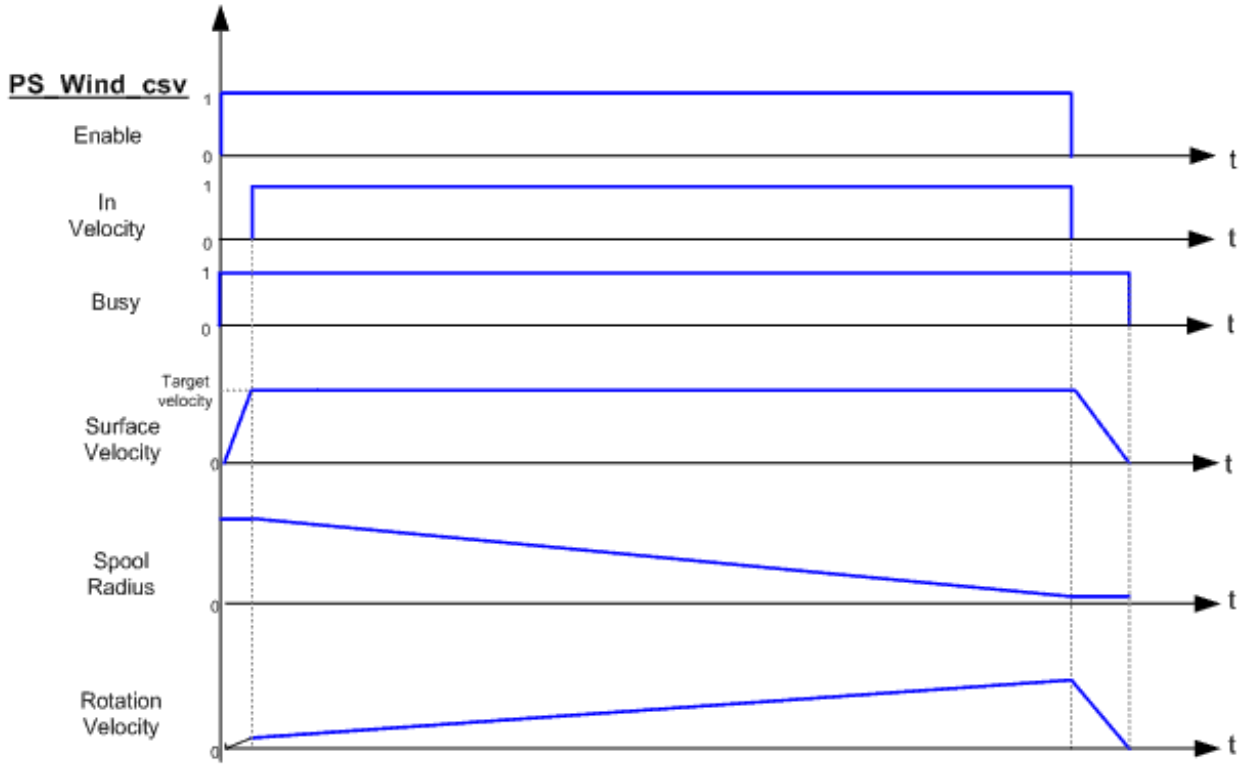


Figure: PS_Wind_csv timing diagram

VAR_INPUT

```
VAR_INPUT
    Enable          : BOOL;     (* Start the motion at high level *)
    SpoolRadius     : LREAL;
    MinSRadius      : LREAL;
    MaxSRadius      : LREAL;
    Acceleration    : LREAL;
    Deceleration    : LREAL;
    Jerk            : LREAL;
END_VAR
```

Enable: Start the motion at high level, stop at low level.

SpoolRadius: Actual value of the radius of the spool [u].

MinSRadius: Value of the Minimal Spool Radius, initial value [u].

MaxSRadius: Value of the Maximal Spool Radius, initial value [u]

Acceleration: Value of the acceleration (increasing energy of the motor) [u/s2]

Deceleration: Value of the deceleration (decreasing energy of the motor) [u/s2]

Jerk: Value of the Jerk [u/s3]

VAR_OUTPUT

```
VAR_OUTPUT
    InVelocity      : BOOL; (* is in velocity*)
    Busy            : BOOL; (* function block is currently busy *)
    CommandAborted  : BOOL;
    Error           : BOOL; (* Signals that an error has occurred within Function Block *)
    ErrorID         : UDINT;    (* Error identification *)
END_VAR
```

InVelocity: Command circumflex velocity phase active

Busy: Executing status

CommandAborted: Becomes TRUE if the process is interrupted by an external event.

Error: Becomes TRUE, as soon as an error occurs.

ErrorID: If the error output is set, this parameter supplies the error number.

VAR_IN_OUT

```
VAR_IN_OUT
    Master      : AXIS_REF;      (* axis structure *)
    Slave       : AXIS_REF;      (* axis structure *)
END_VAR
```

Master: Master axis structure.

Slave: Slave axis structure.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

4 Packaging Communication

Packaging communication Function blocks provide basic methods to communicate between cell controllers. Three types of functions are defined:

- Send/ Receive Data FB for basic (serial) Point-to-Point communications,
- A confirmed handshake for Telegram based communications, as known with many protocols, to cover multi-Point communication,
- Publish /Subscribe for variable publishing based communications as alternative multi-point communication.

Intention is to standardize these three basic communications to be used independently of network standards to abstract from certain specific network technologies to allow software to run regardless of underlying network offered.

Point to Point Communication by Function Block

Provides references to data elements e.g. variables or structures to communicate the content of these data elements. PackTags can be handed from control to control or to devices by means of these higher level function blocks.

PS_Send [▶ 50], PS_Receive [▶ 49] provide simple methods of communication over cell bus or other media. Sender and Receiver ID must be known, the communication is a point to point communication.

Message or Telegram based Communication

Establishes a confirmed connection between devices. The communication is message or telegram based, transferring data content in a transparent way. All information, e.g. sender and receiver header information, is encapsulated in the data field of implemented telegram in a transparent way. These FBs will not handle protocol specific content but manage raw telegram exchange, thus tunneling any specific information between sender and receiver.
The progress of this communication begins with an CommRequest, which arrives at the server where it invokes a CommIndication. The server replies with a CommResponse, which in turn is received by the client and creates a CommConfirmation flag.

Client	Telegram Transmission	Server
Request ->	-> Request telegram	-> Indication
Confirmation <-	<- Response telegram	<- Response

The sender creates a Communication Request, triggering the initiation of the communication by a first telegram. The Server receives the telegram, which creates an indication, and calculates a reply, transmitted by the response FB. The receipt of the answer telegram creates a confirmation at the client side. The communication is carried out by two telegrams in a confirmed way.

Messages which are sent autonomously by a server (e.g. error or other status messages) are registered by the client as a notification indication:

Client	Telegram Transmission	Server
Notification Indication <-	-> Notification telegram	-> Notification

The sender creates a Notification, which arrives at the Client without a previous Request handle. Therefore, it is noted as a Notification, forming an unconfirmed communication of data.
This includes of PS_CommRequest, PS_CommIndicate, PS_CommRespond, PS_CommConfirm, PS_CommNotify, PS_CommIndicate2

Communication by Publisher/Subscriber

The Publisher/subscriber model allows to exchange data on a network by means of publishing and subscribing to variables. It offers widest flexibility and asynchronous as well as synchronous connection methods independent from network technologies.

The sender publishes variables (data structures) at a certain cycle time without confirmed connections. Subscribers (one or many) may consume the published information at a different cycle time, synchronous or not. A quality variable informs about the age of information in sender cycle counts.

This includes of PS_DataPublish, PS_DataSubscribe

4.1 Data Types

4.1.1 DATA_REF

```
TYPE DATA_REF :
STRUCT
    DataAdr : DWORD;
    DataLen : UDINT;
END_STRUCT
END_TYPE
```

DataAdr: Address of Data.

DataLen: Size of Data.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

4.1.2 RECEIVER_REF

```
TYPE RECEIVER_REF :
STRUCT
    NetId   : T_AmsNetId;   (* AMS Net id *)
    Port    : T_AmsPort;    (* Port number *)
    IdxGrp  : UDINT;        (* Index group of service *)
    IdxOffs : UDINT;        (* Index offset of service *)
END_STRUCT
END_TYPE
```

NetID: AMS Net id.

Port: Port number.

IdxGrp: Index group of service.

IdxOffs: Index offset of service.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

4.1.3 SENDER_REF

```
TYPE SENDER_REF :
STRUCT
    NetId   : T_AmsNetId;   (* AMS Net id *)
    Port    : T_AmsPort;    (* Port number *)
```

```
    IdxGrp  : UDINT;        (* Index group of service *)
    IdxOffs : UDINT;        (* Index offset of service *)
END_STRUCT
END_TYPE
```

NetID: AMS Net id.

Port: Port number.

IdxGrp: Index group of service.

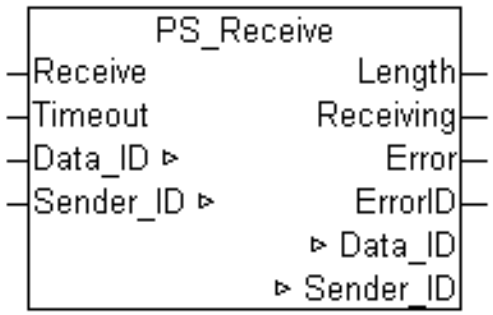
IdxOffs: Index offset of service.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

4.2 Function Blocks

4.2.1 PS_Receive



This function block receives data structures referenced to the receiver. This may take more than one cycle of control or network of communication.
The FB is executed in every control cycle.

VAR_INPUT

```
VAR_INPUT
    Receive : BOOL;
    Timeout : TIME;
END_VAR
```

Receive: Awaiting reception of data.

Timeout: Supervisory timeout.

VAR_OUTPUT

```
VAR_OUTPUT
    Length      : UDINT;
    Receiving   : BOOL;
    Error       : BOOL;
    ErrorID     : UDINT;
END_VAR
```

Length: Length of data set in bytes retrieved from Message.

Receiving: True if receiving data.

Error: Signals that an error has occurred within Function Block, e.g. new send command while still executing.

ErrorID: Error Identification.

VAR_IN_OUT

```
VAR_IN_OUT
    Data_ID         : Data_REF;
    Sender_ID       : Sender_REF;
END_VAR
```

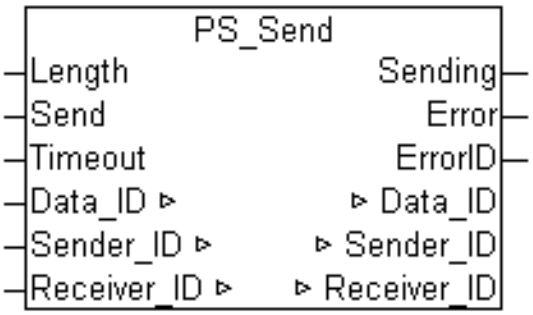
Data_ID: Reference to data to be stored after reception.

Sender_ID: Reference to Sender_ID retrieved from transmission.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

4.2.2 PS_Send



This function block sends the data structure referenced to the receiver. This may take more than one cycle of control or network of communication.

VAR_INPUT

```
VAR_INPUT
    Length  : UDINT;
    Send    : BOOL;
    Timeout : TIME;
END_VAR
```

Length: Length of data set in bytes.

Send: Executes a send sequence.

Timeout: Supervisory timeout.

VAR_OUTPUT

```
VAR_OUTPUT
    Sending     : BOOL;
    Error       : BOOL;
    ErrorID     : UDINT;
END_VAR
```

Sending: True if sending is being performed.

Error: Signals that an error has occurred within Function Block, e.g. new send command while still executing.

ErrorID: Error Identification.

VAR_IN_OUT

```
VAR_IN_OUT
    Data_ID         : Data_REF;
    Sender_ID       : Sender_REF;
    Receiver_ID     : Receiver_REF;
END_VAR
```

Data_ID: Reference to data to be sent.

Sender_ID: Reference to Sender_ID.

Receiver_ID: Reference to Receiver_ID.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

5 Packaging Machine State

Packaging Machine State Function Blocks provide a common interface to the existing PackML Machine State Model implementations available. It is expected that the application specific logic including the transitions between states is programmed in external function blocks, but the central logic of the state machine and the status representation should be handled by the Pack_ML_State_Machine Function block. Therefore, this FB comes with a recommendation how to combine with other logic.

The state transitions to a machine application are always application specific. Therefore, to facilitate standardization, it should be best practice to form state function blocks that are connected to the PS_PackML_State_Machine_V3. The state function blocks collect application specific signals anf form the transition logic to the adjacent states as displayed in the PackML state model. All state FB feed back into PS_PackML_State_Machine_V3, offering a standard state machine and state reporting. The state FBs will contain the machine execution code next to the application specific transition logic.

State Function Blocks are listed below and will be programmed by application programmer accordingly to maintain integrity and function of the PackML State Machine:

PS_Pack_ML_State_Machine_V3 Function Block Names:

- PS_Starting
- PS_Completing
- PS_Resetting
- PS_Holding
- PS_unHolding
- PS_Suspending
- PS_Clearing
- PS_Stopping
- PS_Aborting
- PS_Execute
- PS_Complete
- PS_Idle
- PS_Held
- PS_Suspended
- PS_Stopped
- PS_Aborted

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9, build 1318 onwards	PC (i386)	TcPackALv3.0.Lib

5.1 DataTypes

5.1.1 E_PMLState

E_PMLState

```
TYPE E_PMLState : (
    (* states according to PackTags v3.0 *)
    ePMLState_UNDEFINED     := 0,
    ePMLState_CLEARING      := 1,
    ePMLState_STOPPED       := 2,
    ePMLState_STARTING      := 3,
    ePMLState_IDLE          := 4,
    ePMLState_SUSPENDED     := 5,
```

```
    ePMLState_EXECUTE       := 6,
    ePMLState_STOPPING      := 7,
    ePMLState_ABORTING      := 8,
    ePMLState_ABORTED       := 9,
    ePMLState_HOLDING       := 10,
    ePMLState_HELD          := 11,
    ePMLState_UNHOLDING     := 12,
    ePMLState_SUSPENDING    := 13,
    ePMLState_UNSUSPENDING  := 14,
    ePMLState_RESETTING     := 15,
    ePMLState_COMPLETING    := 16,
    ePMLState_COMPLETE      := 17
);
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10, build 1319 onwards	PC (i386)	TcPackALv3.0.Lib

5.1.2 E_PMLUnitMode

E_PMLUnitMode

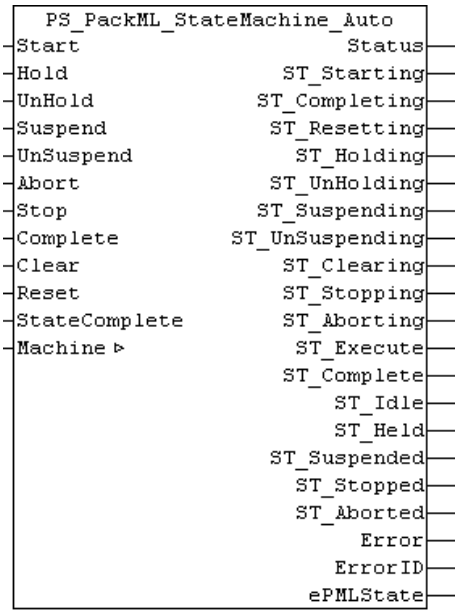
```
TYPE E_PMLUnitMode : (
    ePMLUnitMode_UNDEFINED      := 0,
    ePMLUnitMode_AUTOMATIC      := 1,
    ePMLUnitMode_MAINTENANCE    := 2,
    ePMLUnitMode_MANUAL         := 3,
    ePMLUnitMode_SEMIAUTOMATIC  := 4,
    ePMLUnitMode_DRYRUN         := 5,
    ePMLUnitMode_USERMODE1      := 6,
    ePMLUnitMode_USERMODE2      := 7,
    ePMLUnitMode_IDLE           := 8,
    ePMLUnitMode_ESTOP          := 9
);
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10, build 1319 onwards	PC (i386)	TcPackALv3.0.Lib

5.2 Function Blocks

5.2.1 PS_PackML_StateMachine_Auto



Packaging Machine State Function Blocks, in updated form, provide a common interface to the PackML Machine State Model V3. It is expected that the application specific logic including the transitions between states is programmed in external function blocks, but the central logic of the state machine and the status representation should be handled by the Pack_ML_State_Machine Function block. Therefore, this FB comes with a recommendation how to combine with other logic.

The logic for transitions depends on the application, especially for those between manual, semi-automatic and automatic mode.

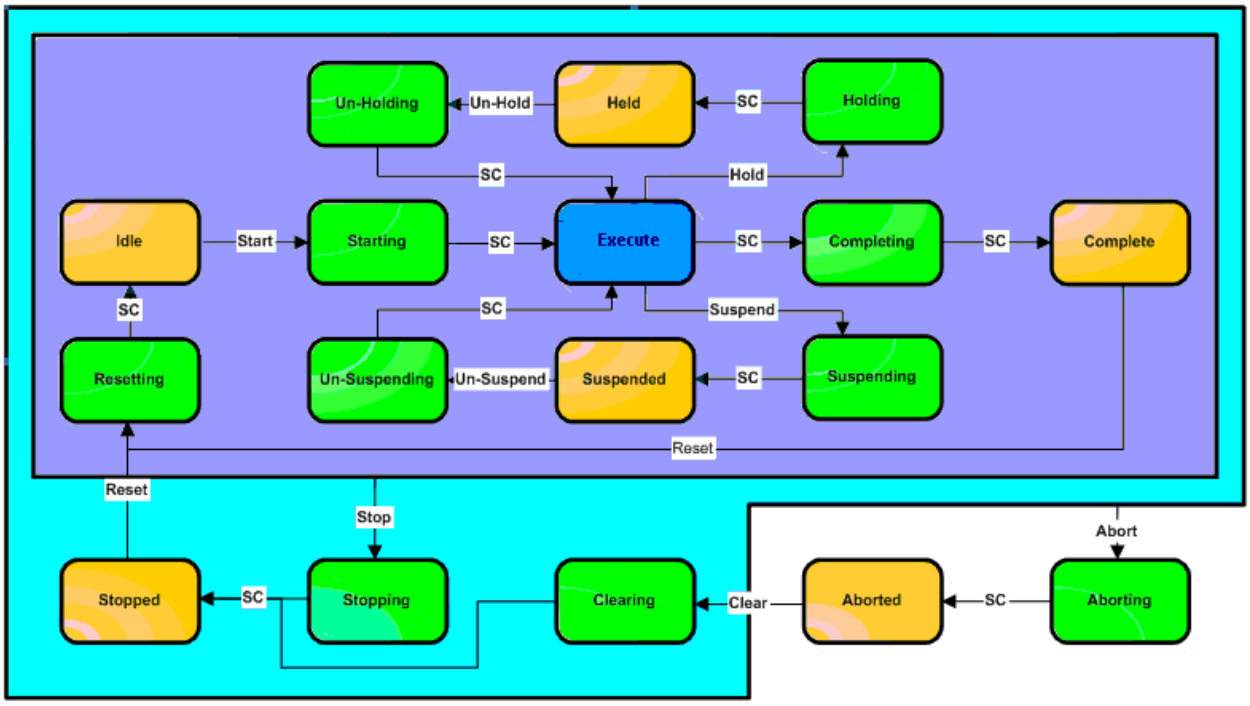


Figure : PS_Pack_ML_State_Model_Automatic Mode

Fig. 2: PS_Pack_ML_State_Model_Automatic Mode

VAR_INPUT

```
VAR_INPUT
    Start           : BOOL;
    Hold            : BOOL;
    unHold          : BOOL;
    Suspend         : BOOL;
    unSuspend       : BOOL;
    Abort           : BOOL;
    Stop            : BOOL;
    Complete        : BOOL;
    Clear           : BOOL;
    Reset           : BOOL;
    StateComplete   : BOOL;
END_VAR
```

Start: Execute State Machine from rising edge, to Starting

Hold: to Holding or Held

UnHold: to un-Holding

Suspend: to Suspending or Suspend

UnSuspend: to un-Suspending

Abort: to Aborting

Stop: to Stopping

Complete: to Resetting

Clear: to Clearing

Reset: to Resetting

StateComplete: Transition

VAR_OUTPUT

```
VAR_OUTPUT
    Status          : WORD;
    ST_Starting     : BOOL;
    ST_Completing   : BOOL;
    ST_Resetting    : BOOL;
    ST_Holding      : BOOL;
    ST_UnHolding    : BOOL;
    ST_Suspending   : BOOL;
    ST_UnSuspending : BOOL;
    ST_Clearing     : BOOL;
    ST_Stopping     : BOOL;
    ST_Aborting     : BOOL;
    ST_Execute      : BOOL;
    ST_Complete     : BOOL;
    ST_Idle         : BOOL;
    ST_Held         : BOOL;
    ST_Suspended    : BOOL;
    ST_Stopped      : BOOL;
    ST_Aborted      : BOOL;
    Error           : BOOL;
    ErrorID         : UDINT;
    ePMLState       : E_PMLState;
END_VAR
```

Status: Status Word representing the status of the State Machine.

ST_Starting: True if State Machine is in state Starting

ST_Completing: True if State Machine is in state Completing

ST_Resetting: True if State Machine is in state Resetting.

ST_Holding: True if State Machine is in state Holding.

ST_UnHolding: True if State Machine is in state UnHolding.

ST_Suspending: True if State Machine is in state Suspending.

ST_UnSuspending: True if State Machine is in state UnSuspending.

ST_Clearing: True if State Machine is in state Clearing

ST_Stopping: True if State Machine is in state Stopping

ST_Aborting: True if State Machine is in state Aborting

ST_Execute: True if State Machine is in state Execute

ST_Complete: True if State Machine is in state Complete

ST_Idle: True if State Machine is in state Idle.

ST_Held: True if State Machine is in state Held

ST_Suspended: True if State Machine is in state Suspended

ST_Stopped: True if State Machine is in state Stopped

ST_Aborted: True if State Machine is in state Aborted

Error: Becomes TRUE, as soon as an error occurs.

ErrorID: If the error output is set, this parameter supplies the error number.

ePMLState: Current PML state of the automatic state machine.

VAR_IN_OUT

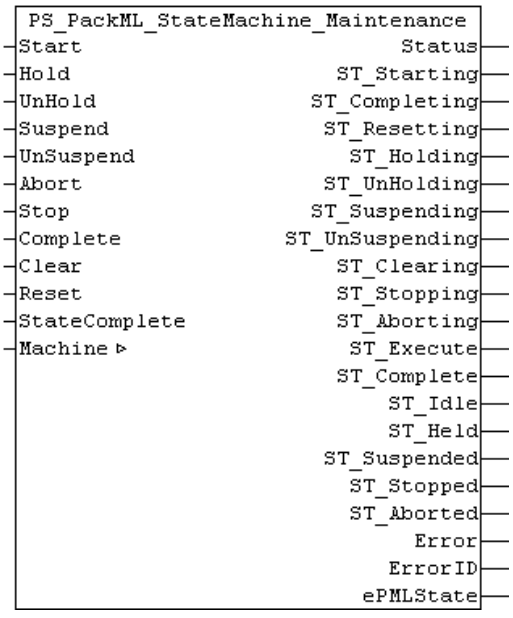
```
VAR_IN_OUT
    Machine         : MACHINE_REF;
END_VAR
```

Machine: To identify the machine executed by the state model.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10, build 1319 onwards	PC (i386)	TcPackALv3.0.Lib

5.2.2 PS_PackML_StateMachine_Maintenance



Packaging Machine State Function Blocks, in updated form, provide a common interface to the PackML Machine State Model V3. It is expected that the application specific logic including the transitions between states is programmed in external function blocks, but the central logic of the state machine and the status representation should be handled by the Pack_ML_State_Machine Function block. Therefore, this FB comes with a recommendation how to combine with other logic.

The logic for transitions depends on the application, especially for those between manual, semi-automatic and automatic mode.

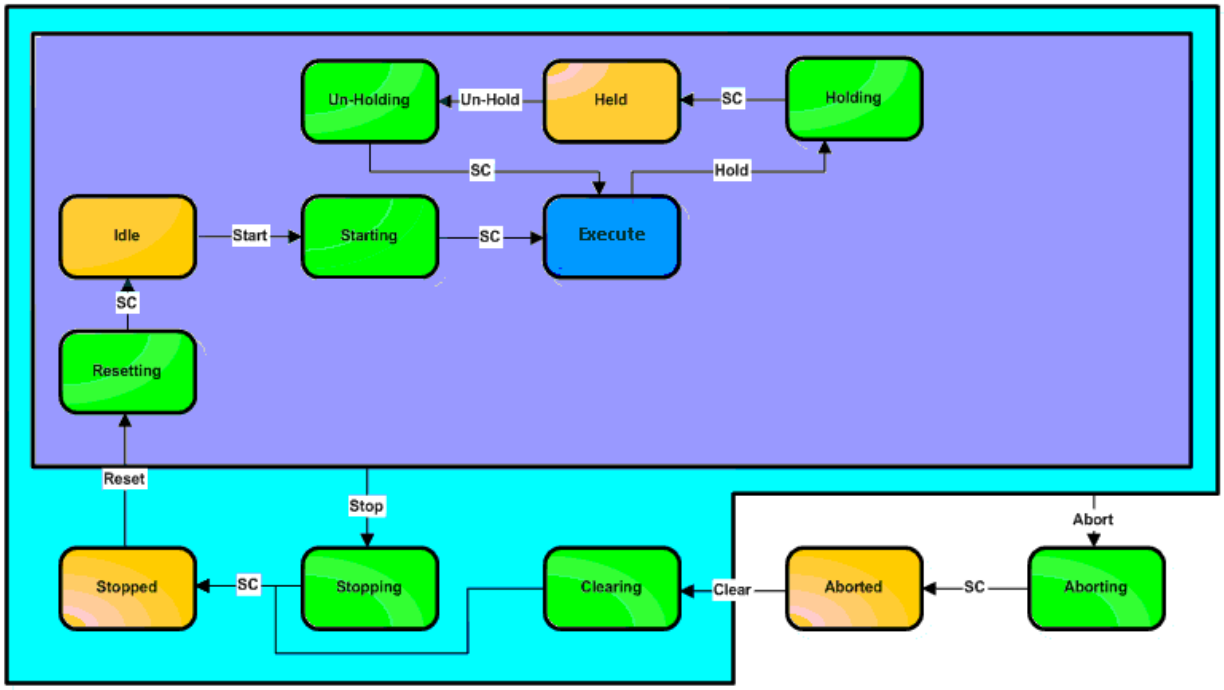


Figure : PS_Pack_ML_State_Model_Maintenance Mode

Fig. 3: PS_Pack_ML_State_Model_Maintenance Mode

VAR_INPUT

```
VAR_INPUT
    Start           : BOOL;
    Hold            : BOOL;
    unHold          : BOOL;
    Suspend         : BOOL;
    UnSuspend       : BOOL;
    Abort           : BOOL;
    Stop            : BOOL;
    Complete        : BOOL;
    Clear           : BOOL;
    Reset           : BOOL;
    StateComplete   : BOOL;
END_VAR
```

Start: Execute State Machine from rising edge, to Starting

Hold: to Holding or Held

UnHold: to un-Holding

Suspend: to Suspending or Suspend

UnSuspend: to un-Suspending

Abort: to Aborting

Stop: to Stopping

Complete: to Resetting

Clear: to Clearing

Reset: to Resetting

StateComplete: Transition

VAR_OUTPUT

```
VAR_OUTPUT
    Status          : WORD;
    ST_Starting     : BOOL;
    ST_Completing   : BOOL;
    ST_Resetting    : BOOL;
    ST_Holding      : BOOL;
    ST_UnHolding    : BOOL;
    ST_Suspending   : BOOL;
    ST_UnSuspending : BOOL;
    ST_Clearing     : BOOL;
    ST_Stopping     : BOOL;
    ST_Aborting     : BOOL;
    ST_Execute      : BOOL;
    ST_Complete     : BOOL;
    ST_Idle         : BOOL;
    ST_Held         : BOOL;
    ST_Suspended    : BOOL;
    ST_Stopped      : BOOL;
    ST_Aborted      : BOOL;
    Error           : BOOL;
    ErrorID         : UDINT;
    ePMLState       : E_PMLState;
END_VAR
```

Status: Status Word representing the status of the State Machine.

ST_Starting: True if State Machine is in state Starting.

ST_Completing: True if State Machine is in state Completing

ST_Resetting: True if State Machine is in state Resetting.

ST_Holding: True if State Machine is in state Holding.

ST_UnHolding: True if State Machine is in state UnHolding.

ST_Suspending: True if State Machine is in state Suspending.

ST_UnSuspending: True if State Machine is in state UnSuspending.

ST_Clearing: True if State Machine is in state Clearing

ST_Stopping: True if State Machine is in state Stopping

ST_Aborting: True if State Machine is in state Aborting

ST_Execute: True if State Machine is in state Execute

ST_Complete: True if State Machine is in state Complete

ST_Idle: True if State Machine is in state Idle.

ST_Held: True if State Machine is in state Held

ST_Suspended: True if State Machine is in state Suspended

ST_Stopped: True if State Machine is in state Stopped

ST_Aborted: True if State Machine is in state Aborted

Error: Becomes TRUE, as soon as an error occurs.

ErrorID: If the error output is set, this parameter supplies the error number.

ePMLState: Current PML state of the maintenance state machine.

VAR_IN_OUT

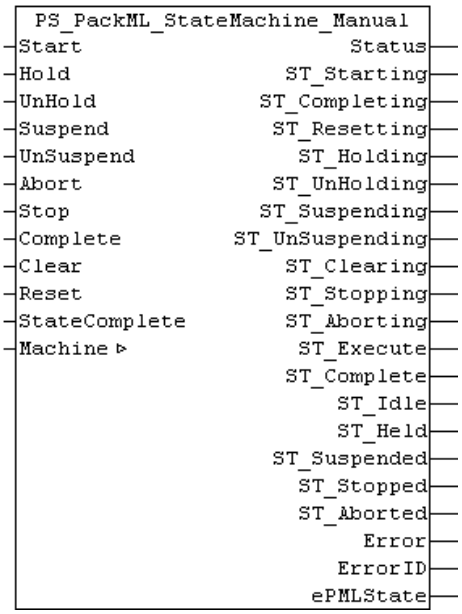
```
VAR_IN_OUT
    Machine         : MACHINE_REF;
END_VAR
```

Machine: To identify the machine executed by the state model.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10, build 1319 onwards	PC (i386)	TcPackALv3.0.Lib

5.2.3 PS_PackML_StateMachine_Manual



Packaging Machine State Function Blocks, in updated form, provide a common interface to the PackML Machine State Model V3. It is expected that the application specific logic including the transitions between states is programmed in external function blocks, but the central logic of the state machine and the status representation should be handled by the Pack_ML_State_Machine Function block. Therefore, this FB comes with a recommendation how to combine with other logic.

The logic for transitions depends on the application, especially for those between manual, semi-automatic and automatic mode.

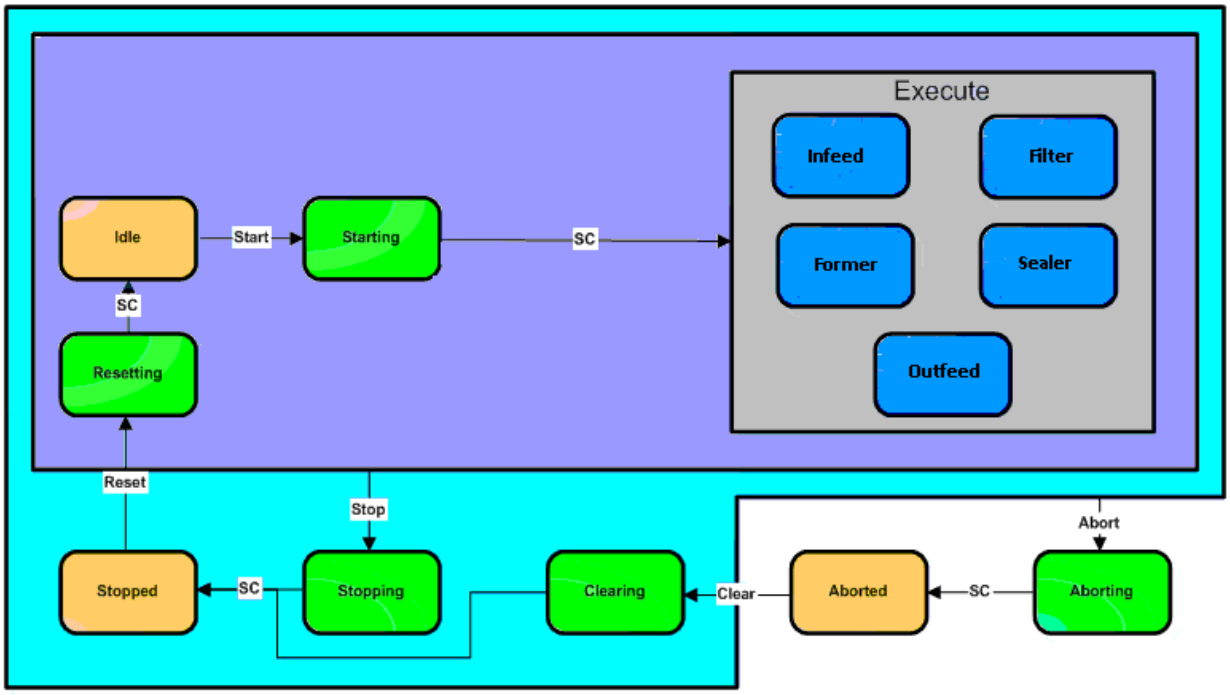


Figure : PS_Pack_ML_State_Model_Manual Mode

VAR_INPUT

```
VAR_INPUT
    Start           : BOOL;
    Hold            : BOOL;
    unHold          : BOOL;
    Suspend         : BOOL;
    UnSuspend       : BOOL;
    Abort           : BOOL;
    Stop            : BOOL;
    Complete        : BOOL;
    Clear           : BOOL;
    Reset           : BOOL;
    StateComplete   : BOOL;
END_VAR
```

Start: Execute State Machine from rising edge, to Starting

Hold: to Holding or Held

UnHold: to un-Holding

Suspend: to Suspending or Suspend

UnSuspend: to un-Suspending

Abort: to Aborting

Stop: to Stopping

Complete: to Resetting

Clear: to Clearing

Reset: to Resetting

StateComplete: Transition

VAR_OUTPUT

```
VAR_OUTPUT
    Status          : WORD;
    ST_Starting     : BOOL;
```

```
    ST_Completing   : BOOL;
    ST_Resetting    : BOOL;
    ST_Holding      : BOOL;
    ST_UnHolding    : BOOL;
    ST_Suspending   : BOOL;
    ST_UnSuspending : BOOL;
    ST_Clearing     : BOOL;
    ST_Stopping     : BOOL;
    ST_Aborting     : BOOL;
    ST_Execute      : BOOL;
    ST_Complete     : BOOL;
    ST_Idle         : BOOL;
    ST_Held         : BOOL;
    ST_Suspended    : BOOL;
    ST_Stopped      : BOOL;
    ST_Aborted      : BOOL;
    Error           : BOOL;
    ErrorID         : UDINT;
    ePMLState       : E_PMLState;
END_VAR
```

Status: Status Word representing the status of the State Machine.

ST_Starting: True if State Machine is in state Starting.

ST_Completing: True if State Machine is in state Completing

ST_Resetting: True if State Machine is in state Resetting.

ST_Holding: True if State Machine is in state Holding.

ST_UnHolding: True if State Machine is in state UnHolding.

ST_Suspending: True if State Machine is in state Suspending.

ST_UnSuspending: True if State Machine is in state UnSuspending.

ST_Clearing: True if State Machine is in state Clearing

ST_Stopping: True if State Machine is in state Stopping

ST_Aborting: True if State Machine is in state Aborting

ST_Execute: True if State Machine is in state Execute

ST_Complete: True if State Machine is in state Complete

ST_Idle: True if State Machine is in state Idle.

ST_Held: True if State Machine is in state Held

ST_Suspended: True if State Machine is in state Suspended

ST_Stopped: True if State Machine is in state Stopped

ST_Aborted: True if State Machine is in state Aborted

Error: Becomes TRUE, as soon as an error occurs.

ErrorID: If the error output is set, this parameter supplies the error number.

ePMLState: Current PML state of the manual state machine.

VAR_IN_OUT

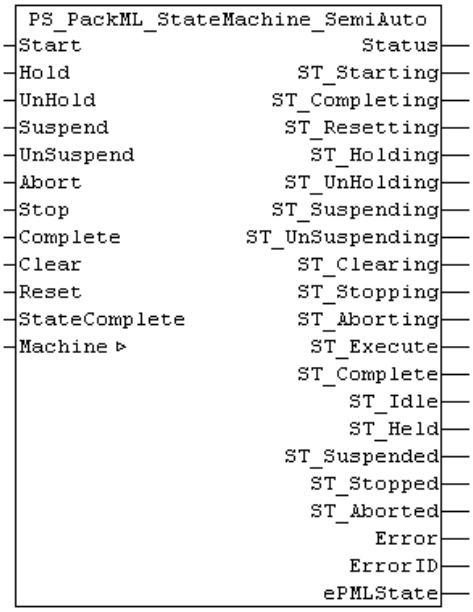
```
VAR_IN_OUT
    Machine         : MACHINE_REF;
END_VAR
```

Machine: To identify the machine executed by the state model.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10, build 1319 onwards	PC (i386)	TcPackALv3.0.Lib

5.2.4 PS_PackML_StateMachine_SemiAuto



Packaging Machine State Function Blocks, in updated form, provide a common interface to the PackML Machine State Model V3.. It is expected that the application specific logic including the transitions between states is programmed in external function blocks, but the central logic of the state machine and the status representation should be handled by the Pack_ML_State_Machine Function block. Therefore, this FB comes with a recommendation how to combine with other logic.

The logic for transitions depends on the application, especially for those between manual, semi-automatic and automatic mode.

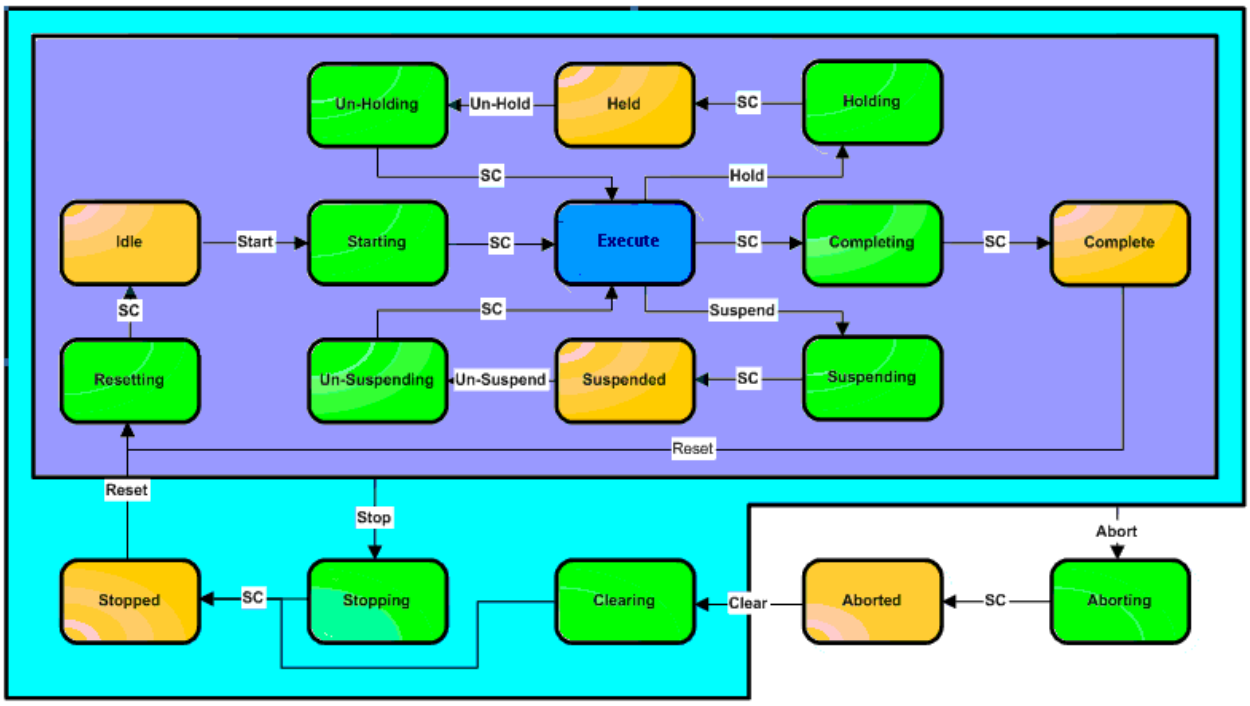


Figure : PS_Pack_ML_State_Model_Automatic Mode

VAR_INPUT

```
VAR_INPUT
    Start           : BOOL;
    Hold            : BOOL;
    unHold          : BOOL;
    Suspend         : BOOL;
    unSuspend       : BOOL;
    Abort           : BOOL;
    Stop            : BOOL;
    Complete        : BOOL;
    Clear           : BOOL;
    Reset           : BOOL;
    StateComplete   : BOOL;
END_VAR
```

Start: Execute State Machine from rising edge, to Starting

Hold: to Holding or Held

UnHold: to un-Holding

Suspend: to Suspending or Suspend

UnSuspend: to un-Suspending

Abort: to Aborting

Stop: to Stopping

Complete: to Resetting

Clear: to Clearing

Reset: to Resetting

StateComplete: Transition

VAR_OUTPUT

```
VAR_OUTPUT
    Status          : WORD;
    ST_Starting     : BOOL;
```

```
    ST_Completing   : BOOL;
    ST_Resetting    : BOOL;
    ST_Holding      : BOOL;
    ST_UnHolding    : BOOL;
    ST_Suspending   : BOOL;
    ST_UnSuspending : BOOL;
    ST_Clearing     : BOOL;
    ST_Stopping     : BOOL;
    ST_Aborting     : BOOL;
    ST_Execute      : BOOL;
    ST_Complete     : BOOL;
    ST_Idle         : BOOL;
    ST_Held         : BOOL;
    ST_Suspended    : BOOL;
    ST_Stopped      : BOOL;
    ST_Aborted      : BOOL;
    Error           : BOOL;
    ErrorID         : UDINT;
    ePMLState       : E_PMLState;
END_VAR
```

Status: Status Word representing the status of the State Machine.

ST_Starting: True if State Machine is in state Starting

ST_Completing: True if State Machine is in state Completing

ST_Resetting: True if State Machine is in state Resetting.

ST_Holding: True if State Machine is in state Holding.

ST_UnHolding: True if State Machine is in state UnHolding.

ST_Suspending: True if State Machine is in state Suspending.

ST_UnSuspending: True if State Machine is in state UnSuspending.

ST_Clearing: True if State Machine is in state Clearing

ST_Stopping: True if State Machine is in state Stopping

ST_Aborting: True if State Machine is in state Aborting

ST_Execute: True if State Machine is in state Execute

ST_Complete: True if State Machine is in state Complete

ST_Idle: True if State Machine is in state Idle.

ST_Held: True if State Machine is in state Held

ST_Suspended: True if State Machine is in state Suspended

ST_Stopped: True if State Machine is in state Stopped

ST_Aborted: True if State Machine is in state Aborted

Error: Becomes TRUE, as soon as an error occurs.

ErrorID: If the error output is set, this parameter supplies the error number.

ePMLState: Current PML state of the semi automatic state machine.

VAR_IN_OUT

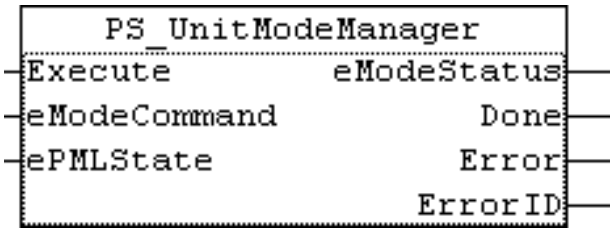
```
VAR_IN_OUT
    Machine         : MACHINE_REF;
END_VAR
```

Machine: To identify the machine executed by the state model.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10, build 1319 onwards	PC (i386)	TcPackALv3.0.Lib

5.2.5 PS_UnitModeManager



Packaging machinery has unit modes other than “automatic”, as noted above. Each unit mode is a defined by its own state model. In order to manage the change from one mode to the next a procedure known as a “mode manager” must be defined. The mode manager determines how, and in what state a machine may change unit modes; ie. the mode manager includes interlocks that prevent the machine changing at inappropriate states.

Note The logic for transitions between Modes depends on the application, especially for those between manual, semi-automatic and automatic mode. Additionally, these mode changes may require interlocks by means of hardware or safety related equipment. Responsibility for proper application of mode switching resides with whoever implements such means.

See the below figure for example.

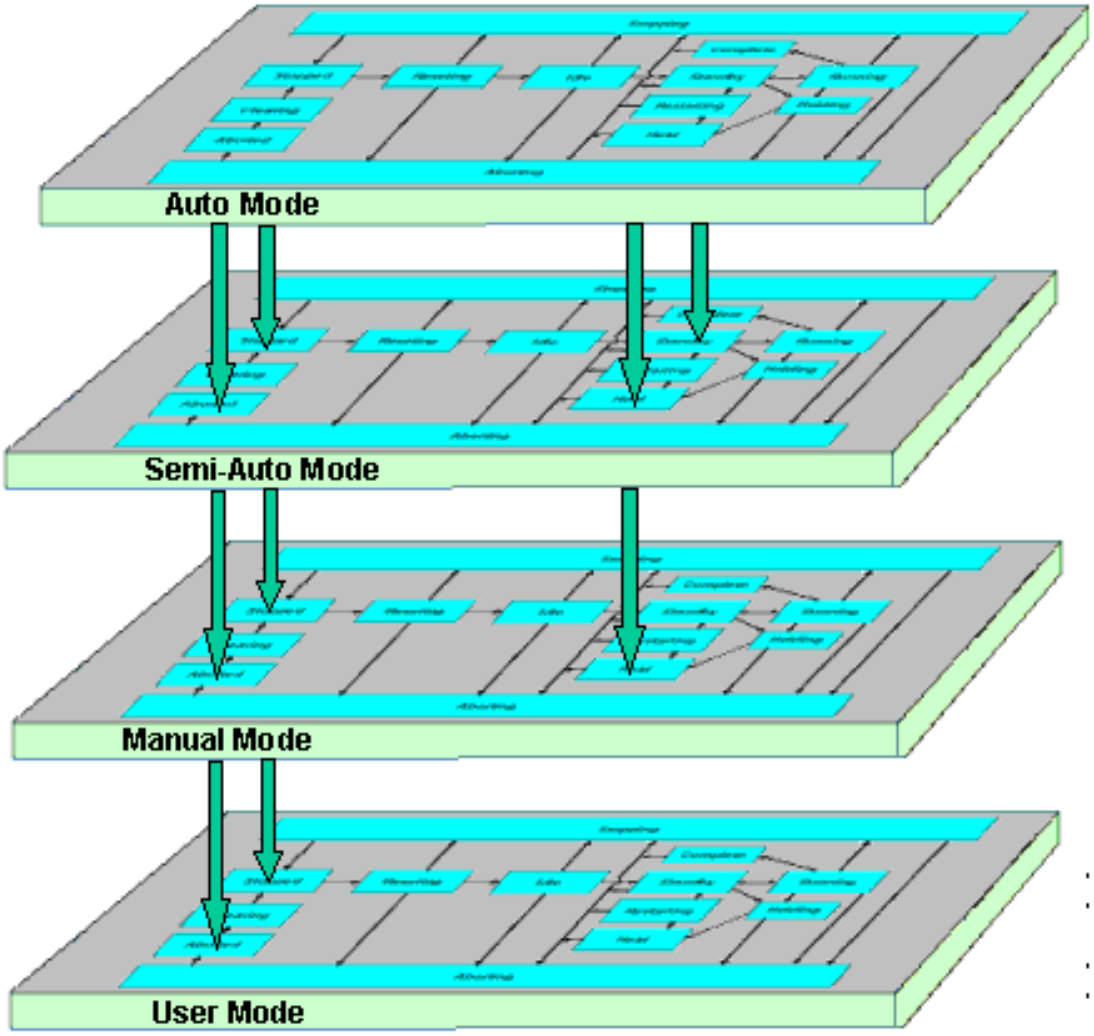


Figure : User Mode Operations State Model

VAR_INPUT

```
VAR_INPUT
    Execute          : BOOL;
    eModeCommand     : E_PMLUnitMode;
    ePMLState        : E_PMLState;
END_VAR
```

Execute: Mode change on rising edge.

eModeCommand: Requested Unit Mode. (E_PMLUnitMode [▶ 53])

ePMLState: Current PML state of the current mode. (E_PMLState [▶ 52])

VAR_OUTPUT

```
VAR_OUTPUT
    eModeStatus      : E_PMLUnitMode;
    Done             : BOOL;
    Error            : BOOL;
    ErrorID          : UDINT;
END_VAR
```

eModeStatus: Current Unit Mode. (E_PMLUnitMode [▶ 53])

Done: True if mode change was successfully performed

Error: Signals that an error has occurred within Function Block, e.g. mode change not permitted

ErrorID: If the error output is set, this parameter supplies the error number.
0 = No error;
1 = mode switch not permitted. The ePMLState is not in idle, stopped, aborted, held, suspended or completed or the corresponding state does not exist in the requested mode.

VAR_IN_OUT

```
VAR_IN_OUT
    Machine_ID      : MACHINE_REF;    (* Identifies the axis which position shall be latched at the
trigger event *)
END_VAR
```

Machine_ID: To identify the machine executed by the state model.

Implementation

The mode change is restricted in certain modes, see implementation below.

Note Not all modes are yet implemented.

```
rTrig(CLK:= Execute);
IF rTrig.Q THEN
    Done := FALSE;
    Error := FALSE;
    ErrorID := 0;
    CASE eModeStatus OF
    ePMLUnitMode_AUTOMATIC:
        IF (ePMLState = ePMLState_STOPPED) OR (ePMLState = ePMLState_ABORTED) OR (ePMLState = ePMLSt
ate_IDLE) THEN
            eModeStatus := eModeCommand;
            Done := TRUE;
        ELSIF ((ePMLState = ePMLState_SUSPENDED) OR (ePMLState = ePMLState_HELD) OR (ePMLState = ePM
LState_COMPLETE))
        AND (eModeCommand = ePMLUnitMode_SEMIAUTOMATIC) THEN
            eModeStatus := eModeCommand;
            Done := TRUE;
        ELSIF (ePMLState = ePMLState_HELD) AND (eModeCommand = ePMLUnitMode_MAINTENANCE) THEN
            eModeStatus := eModeCommand;
            Done := TRUE;
        ELSE
            Error := TRUE;
            ErrorID := 1;
        END_IF
    ePMLUnitMode_MAINTENANCE:
        IF (ePMLState = ePMLState_STOPPED) OR (ePMLState = ePMLState_ABORTED) OR (ePMLState = ePMLSt
ate_IDLE) THEN
            eModeStatus := eModeCommand;
            Done := TRUE;
        ELSIF (ePMLState = ePMLState_HELD) AND ((eModeCommand = ePMLUnitMode_AUTOMATIC) OR (eModeCom
mand = ePMLUnitMode_SEMIAUTOMATIC))          THEN
            eModeStatus := eModeCommand;
            Done := TRUE;
        ELSE
            Error := TRUE;
            ErrorID := 1;
        END_IF
    ePMLUnitMode_MANUAL:
        IF (ePMLState = ePMLState_STOPPED) OR (ePMLState = ePMLState_ABORTED) OR (ePMLState = ePMLSt
ate_IDLE) THEN
            eModeStatus := eModeCommand;
            Done := TRUE;
        ELSE
            Error := TRUE;
            ErrorID := 1;
        END_IF
    ePMLUnitMode_SEMIAUTOMATIC:
        IF (ePMLState = ePMLState_STOPPED) OR (ePMLState = ePMLState_ABORTED) OR (ePMLState = ePMLSt
ate_IDLE) THEN
            eModeStatus := eModeCommand;
            Done := TRUE;
        ELSIF ((ePMLState = ePMLState_SUSPENDED) OR (ePMLState = ePMLState_HELD) OR (ePMLState = ePM
LState_COMPLETE))
        AND (eModeCommand = ePMLUnitMode_AUTOMATIC) THEN
            eModeStatus := eModeCommand;
            Done := TRUE;
```

```
        ELSIF (ePMLState = ePMLState_HELD) AND (eModeCommand = ePMLUnitMode_MAINTENANCE) THEN
            eModeStatus := eModeCommand;
            Done := TRUE;
        ELSE
            Error := TRUE;
            ErrorID := 1;
        END_IF
    ePMLUnitMode_IDLE:
        IF (ePMLState = ePMLState_STOPPED) OR (ePMLState = ePMLState_ABORTED) OR (ePMLState = ePMLSt
ate_IDLE) THEN
            eModeStatus := eModeCommand;
            Done := TRUE;
        ELSE
            Error := TRUE;
            ErrorID := 1;
        END_IF
    ePMLUnitMode_ESTOP:
        IF (ePMLState = ePMLState_STOPPED) OR (ePMLState = ePMLState_ABORTED) OR (ePMLState = ePMLSt
ate_IDLE) THEN
            eModeStatus := eModeCommand;
            Done := TRUE;
        ELSE
            Error := TRUE;
            ErrorID := 1;
        END_IF
    ELSE
        eModeStatus := eModeCommand;
        Done := TRUE;
   END_CASE
END_IF
```

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10, build 1319 onwards	PC (i386)	TcPackALv3.0.Lib

6 Appendix

6.1 Sample of TcPackAL version 3.0

Samples programs available for download are classified as full and light version. Light version when compared to full version has no Flying Saw functionalities. Full version is meant for systems having a valid Flying Saw license.

Full Version:

Requirements: TcPackALv3.0.lib, TcPackTagsv3.0.lib and a valid TwinCAT-NC-Flying-Saw Supplement license.

tcpackalsample_v3.0.zip (Resources/zip/9007200074541451.zip)

Lite Version:

Requirements: TcPackALv3.0_Lite.lib and TcPackTagsv3.0.lib

tcpackalsample_v3.0_lite.zip (Resources/zip/9007200074543627.zip)

More Information:
www.beckhoff.com

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com