BECKHOFF New Automation Technology

Manual | EN

TS4100

TwinCAT 2 | PLC Controller Toolbox

Supplement | Control

2022-10-28 | 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.

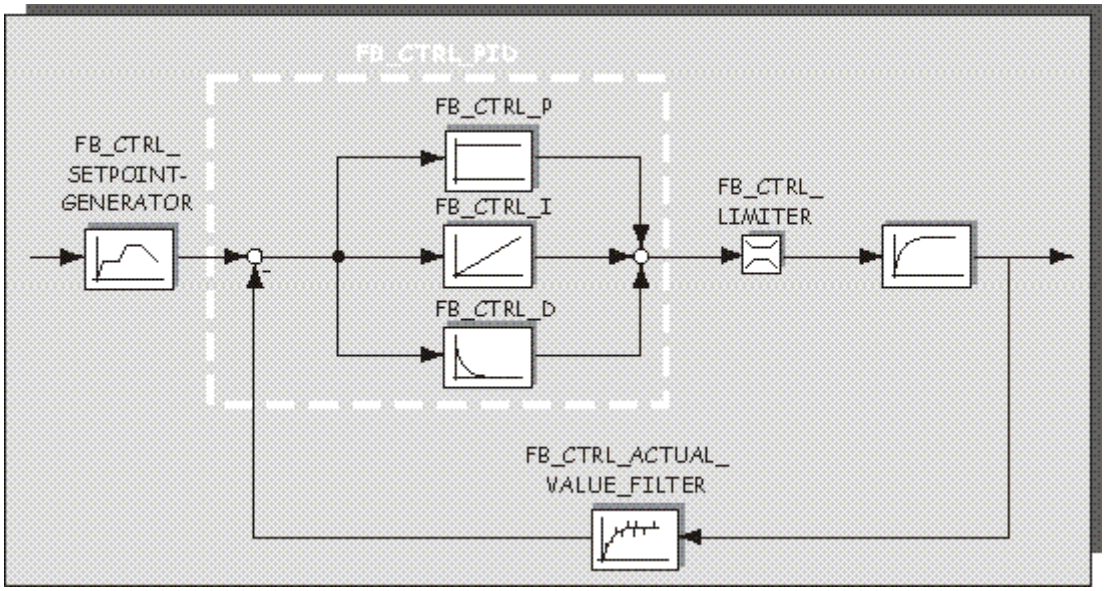
In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

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 Overview

This library contains function blocks that represent various control engineering transfer elements in a functional diagram. Complex controllers that can be used for a large number of applications are included, as well as basic function blocks with which unique controller structures can be implemented for special applications.



Function blocks

Name	Description
FB_CTRL_2POINT [▶ 39]	2-position controller
FB_CTRL_2POINT_PWM_ADAPTIVE [▶ 40]	Adaptive 2-position controller with PWM output
FB_CTRL_3PHASE_SETPOINT_GENERATOR [▶ 140]	3 phase setpoint generator
FB_CTRL_3POINT [▶ 43]	3-position controller
FB_CTRL_3POINT_EXT [▶ 45]	Extended 3-position controller
FB_CTRL_ACTUAL_VALUE_FILTER [▶ 77]	Actual value filter
FB_CTRL_ARITHMETIC_MEAN [▶ 78]	Arithmetic mean value filter
FB_CTRL_CHECK_IF_IN_BAND [▶ 112]	Area monitoring
FB_CTRL_D [▶ 24]	D element
FB_CTRL_DEADBAND [▶ 120]	Dead band
FB_CTRL_DIGITAL_FILTER [▶ 79]	Digital filter
FB_CTRL_FLOW_TEMP_SETPOINT_GEN [▶ 147]	Specification of the flow temperature depending on the outdoor temperature
FB_CTRL_GET_SYSTEM_TIME [▶ 19]	Output of the Windows system time
FB_CTRL_GET_TASK_CYCLETIME [▶ 20]	Determination of the task cycle time
FB_CTRL_HYSTERESIS [▶ 26]	Hysteresis element
FB_CTRL_I [▶ 27]	I element
FB_CTRL_I_WITH_DRIFTCOMPENSATION [▶ 29]	I element with drift compensation
FB_CTRL_LEAD_LAG [▶ 84]	Lead/lag element
FB_CTRL_LIMITER [▶ 121]	Control value limiter
FB_CTRL_LIN_INTERPOLATION [▶ 107]	Linear interpolation element
FB_CTRL_LOG_DATA [▶ 113]	Data logger in *.csv ASCII format

Name	Description
FB_CTRL_LOG_MAT_FILE [▶ 116]	Data logger in Matlab 5 format
FB_CTRL_LOOP_SCHEDULER [▶ 22]	Distribution of computing power in situations with several control loops
FB_CTRL_MOVING_AVERAGE [▶ 82]	Moving average filter
FB_CTRL_MULTIPLE_PWM_OUT [▶ 123]	PWM element with multiple outputs
FB_CTRL_NORMALIZE [▶ 109]	Characteristic curve linearization
FB_CTRL_NOISE_GENERATOR [▶ 87]	Noise generator
FB_CTRL_NOTCH_FILTER [▶ 88]	Notch filter
FB_CTRL_nPOINT [▶ 47]	n-position controller
FB_CTRL_P [▶ 31]	P element
FB_CTRL_PARAMETER_SWITCH [▶ 49]	Parameter switching algorithm for a split range controller
FB_CTRL_PI [▶ 51]	PI controller
FB_CTRL_PI_PID [▶ 54]	Cascaded PI-PID controller
FB_CTRL_PID [▶ 57]	PID controller
FB_CTRL_PID_EXT [▶ 66]	Extended PID controller
FB_CTRL_PID_EXT_SPLITRANGE [▶ 61]	Extended PID controller with parameter set switchover
FB_CTRL_PID_SPLITRANGE [▶ 71]	PID controller with parameter set switchover
FB_CTRL_PT1 [▶ 90]	PT_1 element
FB_CTRL_PT2 [▶ 92]	PT_2 element
FB_CTRL_PT2oscillation [▶ 94]	Oscillating PT_2 element
FB_CTRL_PT3 [▶ 96]	PT_3 element
FB_CTRL_PTn [▶ 98]	PT_n element
FB_CTRL_PTt [▶ 100]	PT_t element
FB_CTRL_PWM_OUT [▶ 127]	PWM element
FB_CTRL_PWM_OUT_EXT [▶ 128]	Extended PWM element
FB_CTRL_RAMP_GENERATOR [▶ 149]	Ramp generator
FB_CTRL_RAMP_GENERATOR_EXT [▶ 151]	Extended ramp generator
FB_CTRL_SCALE [▶ 131]	Range adjustment
FB_CTRL_SERVO_MOTOR_OUT [▶ 132]	Actuator control
FB_CTRL_SERVO_MOTOR_SIMULATION [▶ 102]	Actuator simulation
FB_CTRL_SETPOINT_GENERATOR [▶ 153]	Setpoint generator
FB_CTRL_SIGNAL_GENERATOR [▶ 156]	Signal generator
FB_CTRL_SPLITRANGE [▶ 135]	Signal decomposition into a positive and negative part.
FB_CTRL_STEPPING_MOTOR_OUT [▶ 137]	Stepper motor control
FB_CTRL_TRANSFERFUNCTION_1 [▶ 32]	Transfer function according to the first standard form
FB_CTRL_TRANSFERFUNCTION_2 [▶ 35]	Transfer function according to the second standard form
FB_CTRL_TuTg [▶ 103]	TuTg element
FB_CTRL_ZERO_ZONE_DAMPING [▶ 105]	Zero damping

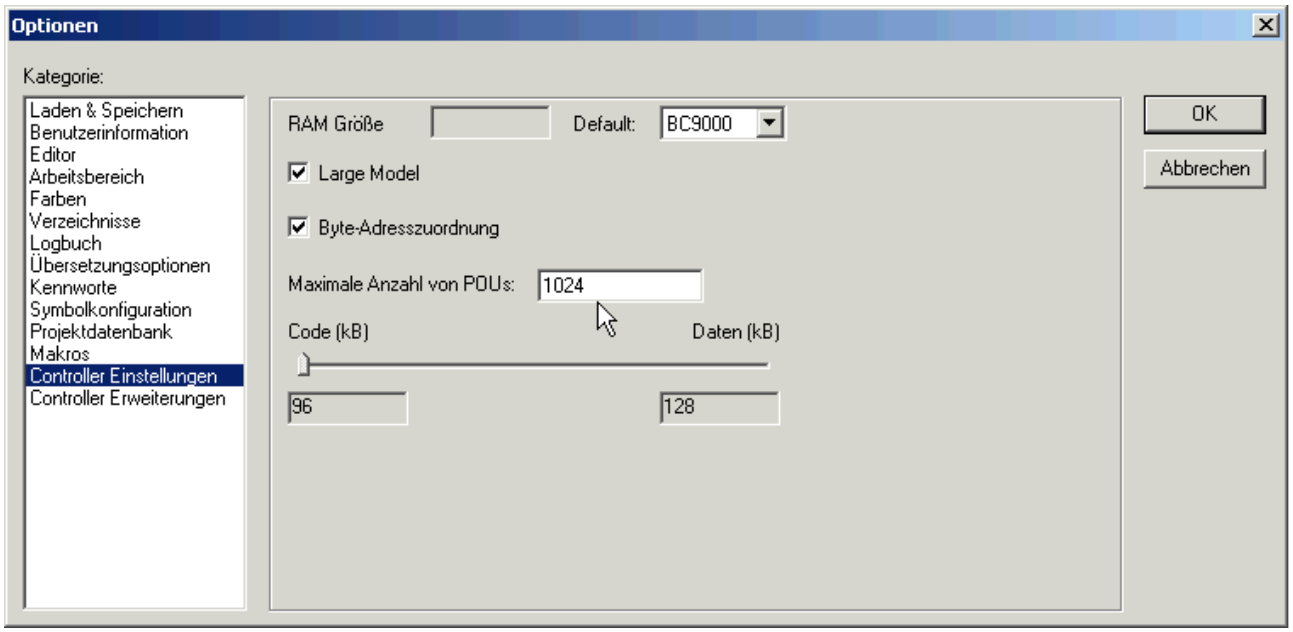
3 General functioning of the FB_CTRL_... Function blocks

The general functioning of the function blocks in the Controller Toolbox are described in the following paragraphs.

General Notes

General Notes

In general, the TcControllerToolbox can be used on PC, BX or BC systems. If the toolbox is used on a BC system, the maximum number of POUs has to be adjusted in the PLC control under menu item Project, Options...



Discretization

The continuous transfer functions of the transfer elements assembled in this library are transformed to discrete values using the trapezoidal rule (Tustin formula).

The Tustin formula:

$$\frac{1}{s} = \frac{T}{2}\frac{z+1}{z-1}$$

Function block inputs

eMode:

The operating mode of the majority of blocks can be selected with this input. This makes it possible to select one of the following operating modes:

eCTRL_MODE_PASSIVE	The output or outputs of the block are set to zero, but the internal states are retained.
eCTRL_MODE_ACTIVE	The block is executed in accordance with its description, and appropriate output values are calculated (normal operation).
eCTRL_MODE_RESET	All internal states are reset in this operating mode, and the error bit is cleared.

eCTRL_MODE_MANUAL	The value of the input value **fManSyncValue** is provided at the output (manual operation).

stParams:

The necessary parameters are passed to the function block with this structure. The variables **tTaskCycleTime** and **tCtrlCycleTime** are contained in all the parameter structures. These parameters function in the following way:

The parameter **tTaskCycleTime** specifies the cycle time with which the function block is called. If the block is called in every cycle this corresponds to the cycle time of the calling task. If it is only called in every second cycle, the time must correspondingly be doubled. The parameter **tCtrlCycleTime** indicates the control loop's sampling time. This time must be greater than or equal to the parameter tTaskCycleTime. If the sampling time is set equal to tTaskCycleTime then the block is executed with every call. If a factor of 5 greater is selected, the block is only processed in every 5th call. This makes it possible to implement slow control loops even in a fast task.

The parameters **tTaskCycleTime** and **tCtrlCycleTime** are of type TIME and therefore do not permit inputs of less than 1ms. To use the controller in a fast PLC task with a cycle time of less than 1ms, a global base time can be specified as reference for the specified cycle times. Notes on using the base time. [▶ 12]

Examples:

It is assumed that the block is called in every task cycle.

Task Configuration	Parameter: tTaskCycleTime	Parameter: tCtrlCycleTime	Method of operation:
T#10ms	T#10ms	T#10ms	The control loop is processed using a 10 ms sampling time.
T#10ms	T#10ms	T#50ms	The control loop is processed using a 50 ms sampling time.
T#100ms	T#100ms	T#100ms	The control loop is processed using a 100 ms sampling time.
T#100ms	T#100ms	T#50ms	**ERROR**, execution not possible!
T#100ms	T#50ms	T#50ms	**ERROR**, although the block has been executed, incorrect output values have been calculated!

The outputs of the function blocks

eState:

This output indicates the current internal state of the block.

eCTRL_STATE_IDLE	The block has successfully been reset, and is now waiting for selection of the operating mode.
eCTRL_STATE_PASSIVE	The block is in the passive state in which no calculations are carried out.
eCTRL_STATE_ACTIVE	The block is in the active state, which is the normal operating state.
eCTRL_STATE_RESET	A reset request is being processed, but the reset has not yet been completed.
eCTRL_STATE_MANAUL	The block is in the manual state, and the output can be manually specified at the appropriate input.
eCTRL_STATE_...	If there are any other internal states, they are described together with the corresponding blocks.
eCTRL_STATE_ERROR	An error has occurred; the block is not executed when in this state. See **eErrorId** for further information.

bError:

An error in the block is indicated by an TRUE at this boolean output.

eErrorId:

The error number [▶ 16] is provided at this output if the bError output is TRUE.

Using the global base time (only available on a PC system)

In order to be able to use the function blocks of a PLC task with a cycle time of less than 1ms, it is possible to interpret the specified cycle times as ticks of a base time. In this special parameterization, the time unit of 1ms is interpreted as 1 tick. This approach is equivalent to setting a PLC cycle time of less than 1ms in the TwinCAT System Manager.

The switchover and declaration of the base time is done with the **global** structure **stCtrl_GLOBAL_CycleTimeInterpretation** for all function blocks of the toolbox.

```
VAR_GLOBAL
    stCtrl_GLOBAL_CycleTimeInterpretation : ST_CTRL_CYCLE_TIME_INTERPRETATION;
END_VAR
```

```
TYPE ST_CTRL_CYCLE_TIME_INTERPRETATION :
STRUCT
    bInterpretCycleTimeAsTicks : BOOL;  (* e.g. 2ms -> 2ticks
*)
    fBaseTime                  : FLOAT; (* Base time in seconds,
e.g. 200µs -> 200E-6s *)
END_STRUCT
END_TYPE
```

In order to interpret the specified cycle times as ticks, the variable **bInterpretCycleTimeAsTicks** in the global structure **stCtrl_GLOBAL_CycleTimeInterpretation** is set to TRUE. Within this structure, the base time unit has to be set in variable **fBaseTime**.

By setting the flag **bInterpretCycleTimeAsTicks**, the interpretation of the parameters with the names

- tTaskCycleTime
- tCtrlCycleTime

is changed. The interpretation and effect of all other parameters of type TIME remains unaffected.

Sample:

The base time unit of the TwinCAT system is 200µs. The PLC task, and therefore the function blocks of the Toolbox, are called cyclically every 400µs.

Setting the global structure:

```
stCtrl_GLOBAL_CycleTimeInterpretation.bInterpretCycleTimeAsTicks := TRUE;
stCtrl_GLOBAL_CycleTimeInterpretation.fBaseTime          := 200E-6;
```

Parameterization of a function block from the Toolbox:

```
stParams.tTaskCycleTime    := T#2ms;  (*2*200µs=400µs *)
stParams.tCtrlCycleTime    := T#4ms;  (*4*200µs=800µs *)
stParams. ...
```

The TaskCycleTime specified on the function blocks is 2•200E-6s = 400µs and thus corresponds to the set PLC cycle time. The CtrlCycleTime is set to 800µs = 4•200E-6s, so that the control loop operates with a sampling time of 800µs, i.e. it is processed during every second PLC cycle.

4 Setting rules for the P, PI and PID controllers

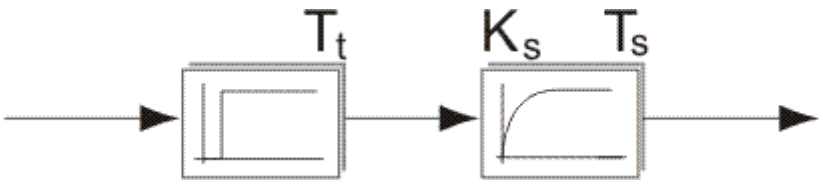
This page summarizes some of the setting rules found in the relevant literature. The setting rule to be used for a particular case has to be determined depending on the controlled system.

Ziegler and Nichols

Ziegler and Nichols

The Ziegler and Nichols setting rules can be used if the controlled system can be approximated by a dead time element and 1st order delay element.

$$G_s(s) = K_s \cdot \frac{e^{-T_t s}}{1 + s \cdot T_s}$$



T_t = dead time of the controlled system

K_s = gain factor of the controlled system

T_s = time constant of the controlled system

Controller	K_r	T_n	T_v
P-Controller	$\frac{T_s}{K_s \cdot T_t}$	-	-
PI-Controller	$0.9 \cdot \frac{T_s}{K_s \cdot T_t}$	$3.33 \cdot T_t$	-
PID-Controller	$1.2 \cdot \frac{T_s}{K_s \cdot T_t}$	$2.0 \cdot T_t$	$0.5 \cdot T_t$

Chien, Hrones and Reswick

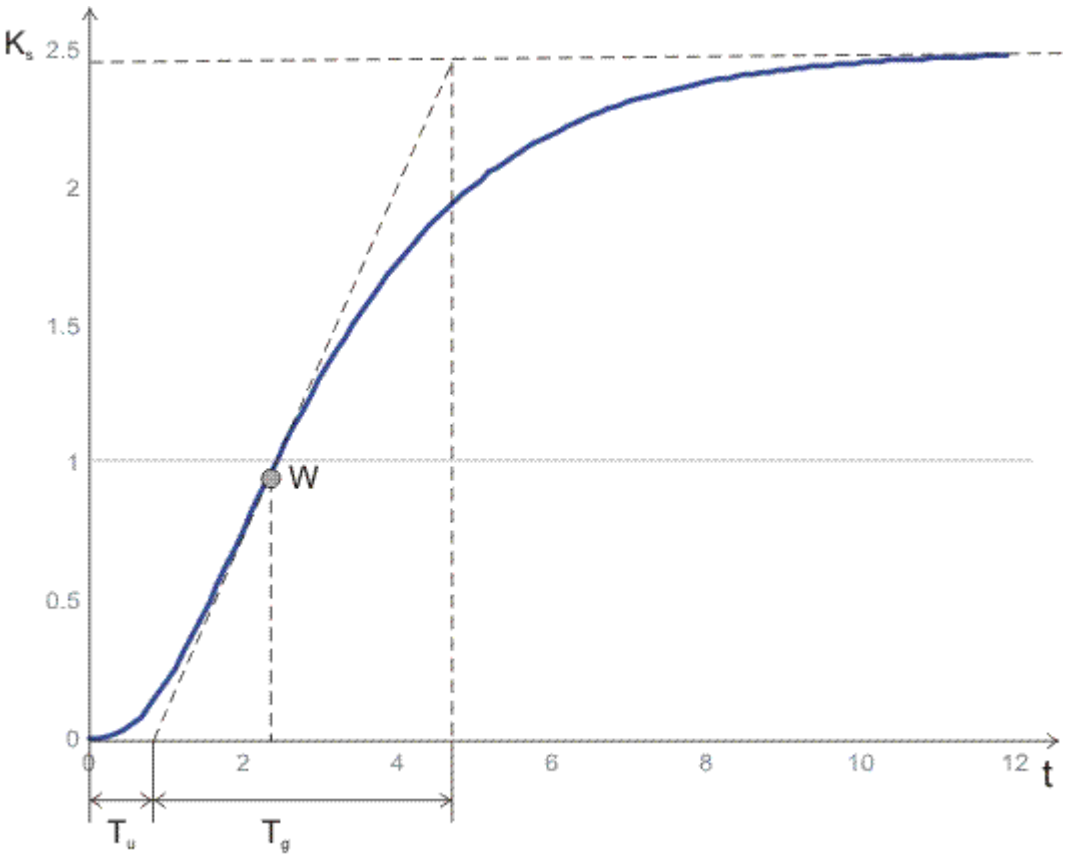
If this procedure is to be used, the step response of the system must show delay characteristics and be free from overshoot effects.

From the step response the delay time T_u, the compensation time T_g and the system gain K_s are determined.

The system gain is calculated from the following quotient:

$$K_s = \frac{\text{Step Response Height}}{\text{Controller Output Value}}$$

The setting rules listed below can only be used if Tg/Tu > 3!



Optimization of the interference behavior:

Controller		Aperiodic step response	20% Overshoot
P-Controller	K_r	$0.30 \cdot \frac{T_g}{T_u \cdot K_s}$	$0.70 \cdot \frac{T_g}{T_u \cdot K_s}$
PI-Controller	K_r	$0.60 \cdot \frac{T_g}{T_u \cdot K_s}$	$0.70 \cdot \frac{T_g}{T_u \cdot K_s}$
	T_N	$4.00 \cdot T_u$	$2.3 \cdot T_u$
PID-Controller	K_r	$0.95 \cdot \frac{T_g}{T_u \cdot K_s}$	$1.20 \cdot \frac{T_g}{T_u \cdot K_s}$
	T_N	$2.40 \cdot T_u$	$2.00 \cdot T_u$
	T_V	$0.42 \cdot T_u$	$0.42 \cdot T_u$

Optimization of the control behavior:

Controller		Aperiodic step response	20% Overshoot
P-Controller	K_r	$0.30 \cdot \frac{T_g}{T_u \cdot K_s}$	$0.70 \cdot \frac{T_g}{T_u \cdot K_s}$
PI-Controller	K_r	$0.35 \cdot \frac{T_g}{T_u \cdot K_s}$	$0.60 \cdot \frac{T_g}{T_u \cdot K_s}$
	T_N	$1.20 \cdot T_g$	$1.0 \cdot T_g$
PID-Controller	K_r	$0.60 \cdot \frac{T_g}{T_u \cdot K_s}$	$0.95 \cdot \frac{T_g}{T_u \cdot K_s}$
	T_N	$1.00 \cdot T_g$	$1.35 \cdot T_g$
	T_V	$0.50 \cdot T_u$	$0.47 \cdot T_u$

5 PLC-API

5.1 Definition of the structures and enums

All the structures and enums used in the Controller Toolbox are described in this appendix.

FLOAT:

The library is structured in such a way that it can run either on a PC, BX or on a BC system. To achieve this portability, the function blocks in the library only work with the FLOAT data type. This data type is defined as LREAL or as REAL in a supplementary library.

The supplementary library "**TcFloatPC.lib**" is linked for **PC systems**.

```
TYPE
     FLOAT : LREAL;
END_TYPE
```

The supplementary library "**TcFloatBC.lib**" is linked for **BC systems**.

```
TYPE
     FLOAT : REAL;
END_TYPE
```

The supplementary library "**TcFloatBX.lib**" is linked for **BX systems**.

```
TYPE
     FLOAT : REAL;
END_TYPE
```

E_CTRL_MODE:

```
TYPE E_CTRL_MODE :
(
    eCTRL_MODE_IDLE       := 0, (* mode idle *)
    eCTRL_MODE_PASSIVE    := 1, (* mode passive *)
    eCTRL_MODE_ACTIVE     := 2, (* mode active *)
    eCTRL_MODE_RESET      := 3, (* mode reset *)
    eCTRL_MODE_MANUAL     := 4, (* mode manual *)
    eCTRL_MODE_TUNE       := 5, (* mode tuning *)
    eCTRL_MODE_SELFTEST       := 6, (* mode selftest *)
    eCTRL_MODE_SYNC_MOVEMENT  := 7  (* mode synchronize *)
);
END_TYPE
```

E_CTRL_STATE :

```
TYPE E_CTRL_STATE :
(
    eCTRL_STATE_IDLE        := 0, (* state idle *)
    eCTRL_STATE_PASSIVE     := 1, (* state passive *)
    eCTRL_STATE_ACTIVE      := 2, (* state active *)
    eCTRL_STATE_RESET           := 3, (* state reset *)
    eCTRL_STATE_MANUAL      := 4, (* state manual *)
    eCTRL_STATE_TUNING      := 5, (* state tuning *)
    eCTRL_STATE_TUNED           := 6, (* state tuning ready - tuned *)
    eCTRL_STATE_SELFTEST    := 7, (* state selftest *)
    eCTRL_STATE_ERROR           := 8, (* state error *)
    eCTRL_STATE_SYNC_MOVEMENT       := 9  (* state synchronizing movement *)
);
END_TYPE
```

E_CTRL_ERRORCODES:

```
TYPE E_CTRL_ERRORCODES :
(
    eCTRL_ERROR_NOERROR                     := 0, (* no error *)
    eCTRL_ERROR_INVALIDTASKCYCLETIME                := 1, (* invalid task cycle time *)
    eCTRL_ERROR_INVALIDCTRLCYCLETIME                := 2, (* invalid ctrl cycle time *)
    eCTRL_ERROR_INVALIDPARAM                    := 3, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_Tv                 := 4, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_Td                 := 5, (* invalid parameter *)
```

```
    eCTRL_ERROR_INVALIDPARAM_Tn                 := 6, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_Ti                 := 7, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fHystereisisRange              := 8, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fPosOutOn_Off          := 9, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fNegOutOn_Off          := 10, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_TableDescription               := 11, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_TableData                  := 12, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_DataTableADR           := 13, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_T0                 := 14, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_T1                 := 15, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_T2                 := 16, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_T3                 := 17, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_Theta              := 18, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nOrder                 := 19, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_Tt                 := 20, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_Tu                 := 21, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_Tg                 := 22, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_infinite_slope         := 23, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fMaxIsLessThanfMin             := 24, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fOutMaxLimitIsLessThanfOutMinLimit     := 25, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fOuterWindow           := 26, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fInnerWindow           := 27, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fOuterWindowIsLessThanfInnerWindow     := 28, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fDeadBandInput             := 29, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fDeadBandOutput            := 30, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_PWM_Cycletime          := 31, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_no_Parameterset            := 32, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fOutOn                 := 33, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fOutOff                := 34, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fGain              := 35, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fOffset                := 36, (* invalid parameter *)
    eCTRL_ERROR_MODE_NOT_SUPPORTED              := 37, (* invalid mode: mode not supported *)
    eCTRL_ERROR_INVALIDPARAM_Tv_heating                 := 38, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_Td_heating                 := 39, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_Tn_heating                 := 40, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_Tv_cooling                 := 41, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_Td_cooling                 := 42, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_Tn_cooling                 := 43, (* invalid parameter *)
    eCTRL_ERROR_RANGE_NOT_SUPPORTED                 := 44, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nParameterChangeCycleTicks         := 45, (* invalid parameter *)
    eCTRL_ERROR_ParameterEstimationFailed           := 46, (* invalid parameter *)
    eCTRL_ERROR_NoiseLevelToHigh                := 47, (* invalid parameter *)
    eCTRL_ERROR_INTERNAL_ERROR_0                := 48, (* internal error *)
    eCTRL_ERROR_INTERNAL_ERROR_1                := 49, (* internal error *)
    eCTRL_ERROR_INTERNAL_ERROR_2                := 50, (* internal error *)
    eCTRL_ERROR_INTERNAL_ERROR_3                := 51, (* internal error *)
    eCTRL_ERROR_INTERNAL_ERROR_4                := 52, (* internal error *)
    eCTRL_ERROR_INTERNAL_ERROR_5                := 53, (* internal error *)
    eCTRL_ERROR_INTERNAL_ERROR_6                := 54, (* internal error *)
    eCTRL_ERROR_INTERNAL_ERROR_7                := 55, (* internal error *)
    eCTRL_ERROR_INTERNAL_ERROR_8                := 56, (* internal error *)
    eCTRL_ERROR_INTERNAL_ERROR_9                := 57, (* internal error *)
    eCTRL_ERROR_INVALIDPARAM_WorkArrayADR           := 58, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_tOnTiime                   := 59, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_tOffTiime                  := 60, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nMaxMovingPulses               := 61, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nAdditionalPulsesAtLimits          := 62, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fCtrlOutMax_Min            := 63, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fDeltaMax                  := 64, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_tMovingTime            := 65, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_tDeadTime                  := 66, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_tAdditionalMoveTimeAtLimits        := 67, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fThreshold                 := 68, (* invalid parameter *)
    eCTRL_ERROR_MEMCPY                      := 69, (* MEMCPY failed *)
    eCTRL_ERROR_MEMSET                      := 70, (* MEMSET failed *)
    eCTRL_ERROR_INVALIDPARAM_nNumberOfColumns               := 71, (* invalid parameter *)
    eCTRL_ERROR_FileClose                   := 72, (* File Close failed *)
    eCTRL_ERROR_FileOpen                    := 73, (* File Open failed *)
    eCTRL_ERROR_FileWrite                   := 74, (* File Write failed *)
    eCTRL_ERROR_INVALIDPARAM_fVeloNeg                   := 75, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fVeloPos                   := 76, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_DeadBandInput          := 77, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_DeadBandOutput         := 78, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_CycleDuration          := 79, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_tStart                 := 80, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_StepHeigthTuningToLow      := 81, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fMinLimitIsLessThanZero        := 82, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fMaxLimitIsGreaterThan100          := 83, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fStepSize                  := 84, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fOkRangeIsLessOrEqualZero          := 85, (* invalid parameter *)
```

```
    eCTRL_ERROR_INVALIDPARAM_fForceRangeIsLessOrEqualfOkRange       := 86, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPWMPeriod                := 87, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_tMinimumPulseTime              := 88, (* invalid parameter *)
    eCTRL_ERROR_FileDelete                  := 89, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nNumberOfPwmOutputs            := 90, (* File Delete failed *)
    eCTRL_ERROR_INVALIDPARAM_nPwmInputArray_SIZEOF      := 91, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nPwmOutputArray_SIZEOF         := 92, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nPwmWaitTimesConfig_SIZEOF         := 93, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nPwmInternalData_SIZEOF        := 94, (* invalid parameter *)
    eCTRL_ERROR_SIZEOF                      := 95, (* SIZEOF failed *)
    eCTRL_ERROR_INVALIDPARAM_nOrderOfTheTransferfunction        := 96, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nNumeratorArray_SIZEOF         := 97, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nDenominatorArray_SIZEOF           := 98, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_a_n_IsZero                 := 99, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_WorkArraySIZEOF            := 100, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_MOVINGRANGE_MIN_MAX            := 101, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_MOVINGTIME                 := 102, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_DEADTIME                   := 103, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fMinLimitIsGreaterThanfMaxLimit    := 104, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_DataTableSIZEOF            := 105, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_OutputVectorDescription        := 106, (* invalid parameter *)
    eCTRL_ERROR_TaskCycleTimeChanged                := 107, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nMinMovingPulses               := 108, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fAcceleration          := 109, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fDeceleration          := 110, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_StartAndTargetPos              := 111, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fVelocity                  := 112, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fTargetPos                 := 113, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fStartPos                  := 114, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fMovingLength          := 115, (* invalid parameter *)
    eCTRL_ERROR_NT_GetTime                  := 116, (* internal error NT_GetTime *)
    eCTRL_ERROR_INVALIDPARAM_No3PhaseSolutionPossible           := 117, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fStartVelo                 := 118, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fTargetVelo            := 119, (* invalid parameter *)
    eCTRL_ERROR_INVALID_NEW_PARAMETER_TYPE          := 120  (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fBaseTime                  := 121, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nOrderOfTheTransferfunction_SIZEOF     := 122, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nFilterOrder           := 124, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nCoefficientsArray_a_SIZEOF        := 125, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nCoefficientsArray_b_SIZEOF        := 126, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nDigitalFiterData_SIZEOF           := 127, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nLogBuffer_SIZEOF              := 128, (* invalid parameter *)
    eCTRL_ERROR_LogBufferOverflow               := 129, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nLogBuffer_ADR             := 130, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nCoefficientsArray_a_ADR           := 131, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nCoefficientsArray_b_ADR           := 132, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nPwmInputArray_ADR             := 133, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nPwmOutputArray_ADR            := 134, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nPwmWaitTimesConfig_ADR        := 135, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nPwmInternalData_ADR       := 136, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nDigitalFiterData_ADR      := 137, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nNumeratorArray_ADR            := 138, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nDenominatorArray_ADR      := 139, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nTransferfunction1Data_ADR         := 140, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_nTransferfunction2Data_ADR         := 141, (* invalid parameter *)
    eCTRL_ERROR_FileSeek                    := 142, (* internal error FB_FileSeek *)
    eCTRL_ERROR_INVALIDPARAM_AmbientTempMaxIsLessThanAmbientTempMin := 143, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_ForerunTempMaxIsLessThanForerunTempMin := 144, (* invalid parameter *)
    eCTRL_ERROR_INVALIDLOGCYCLETIME                 := 145, (* invalid parameter *)
    eCTRL_ERROR_INVALIDVERSION_TcControllerToolbox      := 146,
    eCTRL_ERROR_INVALIDPARAM_Bandwidth                  := 147, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_NotchFrequency             := 148, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_DampingCoefficient             := 149, (* invalid parameter *)
    eCTRL_ERROR_INVALIDPARAM_fKpIsLessThanZero              := 150  (* invalid parameter *)
);
END_TYPE
```

E_CTRL_SIGNAL_TYPE:

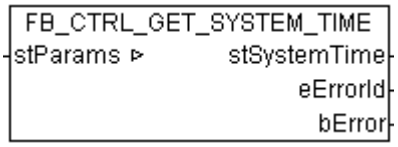
E_CTRL_SIGNAL_TYPE:

```
TYPE
E_CTRL_SIGNAL_TYPE :

    (

    eCTRL_TRIANGLE  := 0,

    eCTRL_SINUS     := 1,
```

```
    eCTRL_SAWTOOTH  := 2

    );

END_TYPE
```

5.2 Auxiliary

5.2.1 FB_CTRL_GET_SYSTEM_TIME (only on a PC system)



This function block reads the current Windows system time, making it available in SystemTimeStruct.

Description:

This function block makes the current system time available in its output structure. The resolution is specified through the tCtrlCycleTime parameter; the maximum resolution is 10 ms, and it is necessary to observe the condition tCtrlCycleTime > 2 • tTaskCycleTime. If this is not done, the resolution will be reduced to 2 • tCtrlCycleTime.

VAR_OUTPUT

```
VAR_OUTPUT
    stSystemTime    : TIMESTRUCT;
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

```
TYPE TIMESTRUCT
STRUCT
    wYear       : WORD;
    wMonth      : WORD;
    wDayOfWeek    : WORD;
    wDay    : WORD;
    wHour       : WORD;
    wMinute       : WORD;
    wSecond       : WORD;
    wMilliseconds : WORD;
END_STRUCT
END_TYPE
```

stSystemTime : Structure in which the system time is output.

wYear : The year: 1970 ~ 2106;

wMonth : The month: 1 ~ 12 (January = 1, February = 2 etc.);

wDayOfWeek : The day of the week: 0 ~ 6 (Sunday = 0, Monday = 1 etc.);

wDay : The day of the month: 1 ~ 31;

wHour : Hour: 0 ~ 23;

wMinute : Minute: 0 ~ 59;

wSecond : Second: 0 ~ 59;

wMilliseconds : Millisecond: 0 ~ 999;

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

bError : Becomes TRUE as soon as an error situation occurs.

VAR_IN_OUT

```
VAR_IN_OUT
    stParams : ST_CTRL_GET_SYSTEM_TIME;
END_VAR
```

stParams : Parameter structure of the function block. This consists of the following elements:

```
TYPE
ST_CTRL_GET_SYSTEM_TIME:
STRUCT
    tTaskCycleTime    : TIME;  (* task cycle time [TIME]
*)
    tCtrlCycleTime    : TIME;  (* controller cycle time [TIME]
*)
END_STRUCT
END_TYPE
```

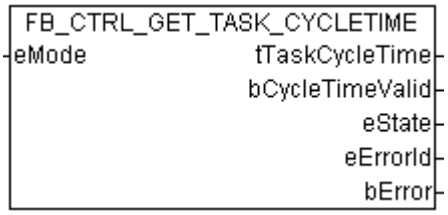
tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every task cycle this corresponds to the task cycle time of the calling task.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib

5.2.2 FB_CTRL_GET_TASK_CYCLETIME (only on a PC system)



This function block allows the task cycle time of a program to be determined with a resolution of 1 ms.

The TaskCycleTime can only be correctly determined if the program does not contain any active breakpoints.

The task cycle time is only determined once. No further measurements are taken if either of the bCycleTimeValid or bError outputs is TRUE.

This function block should not be used if cycle times of less than 1 ms, or that are not a multiple of 1 ms, are in use.

VAR_INPUT

```
VAR_INPUT
    eMode: E_CTRL_MODE;
END_VAR
```

eMode : Input that specifies the block's operating mode [▶ 16].

VAR_OUTPUT

```
VAR_OUTPUT
    tTaskCycleTime      : TIME;             (* resolution: 1ms *)
    bCycleTimeValid     : BOOL;
    eState      : E_CTRL_STATE;
```

```
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

tTaskCycleTime : This output indicates the current task cycle time, with a resolution of 1 ms.

bCycleTimeValid : The time contained in the tTaskCycleTime output is valid when this output is TRUE.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

Sample:

```
PROGRAM PRG_GET_TASK_CYCLETIME_TEST
VAR
    tTaskCycleTime          : TIME;
    bCycleTimeValid         : BOOL;
    eState              : E_CTRL_STATE;
    eErrorId            : E_CTRL_ERRORCODES;
    bError              : BOOL;
    fbCTRL_GET_TASK_CYCLETIME   : FB_CTRL_GET_TASK_CYCLETIME;
    (* control loop *)
    bInit         : BOOL := TRUE;
    fSetpointValue    : FLOAT := 45.0;
    fActualValue      : FLOAT;
    fbCTRL_PI     : FB_CTRL_PI;
    stCTRL_PI_Params  : ST_CTRL_PI_PARAMS;
    fbCTRL_PT1    : FB_CTRL_PT1;
    stCTRL_PT1_Params : ST_CTRL_PT1_PARAMS;
END_VAR
```

```
(* call fb to get the task cycle time *)
fbCTRL_GET_TASK_CYCLETIME( eMode       :=
eCTRL_MODE_ACTIVE,
               tTaskCycleTime  =>
tTaskCycleTime,
               bCycleTimeValid =>
bCycleTimeValid,
               eState      =>
eCTRL_MODE_ACTIVE,
               eErrorId    => eErrorId,
               bError      => bError
              );
(* call control loop if the cycle time is valid *)
IF fbCTRL_GET_TASK_CYCLETIME.bCycleTimeValid
THEN
    IF bInit
    THEN
    stCTRL_PT1_Params.tTaskCycleTime :=
fbCTRL_GET_TASK_CYCLETIME.tTaskCycleTime;
    stCTRL_PT1_Params.tCtrlCycleTime := T#100ms;
    stCTRL_PT1_Params.fKp        := 1.0;
    stCTRL_PT1_Params.tT1        := T#10s;
    stCTRL_PI_Params.tTaskCycleTime  :=
fbCTRL_GET_TASK_CYCLETIME.tTaskCycleTime;
    stCTRL_PI_Params.tCtrlCycleTime  := T#100ms;
    stCTRL_PI_Params.fKp         := 0.5;
    stCTRL_PI_Params.tTn         := T#5s;
    stCTRL_PI_Params.fOutMaxLimit    := 100.0;
    stCTRL_PI_Params.fOutMinLimit    := 0.0;
    bInit := FALSE;
    END_IF
    (* call controller *)
    fbCTRL_PI( fActualValue   := fbCTRL_PT1.fOut,
           fSetpointValue := fSetpointValue,
           eMode      := eCTRL_MODE_ACTIVE,
           stParams       := stCTRL_PI_Params
          );
    (* call PT1 *)
    fbCTRL_PT1( fIn      := fbCTRL_PI.fOut,
        eMode    := eCTRL_MODE_ACTIVE,
        stParams := stCTRL_PT1_Params,
        fOut     => fActualValue
          );
END_IF
```

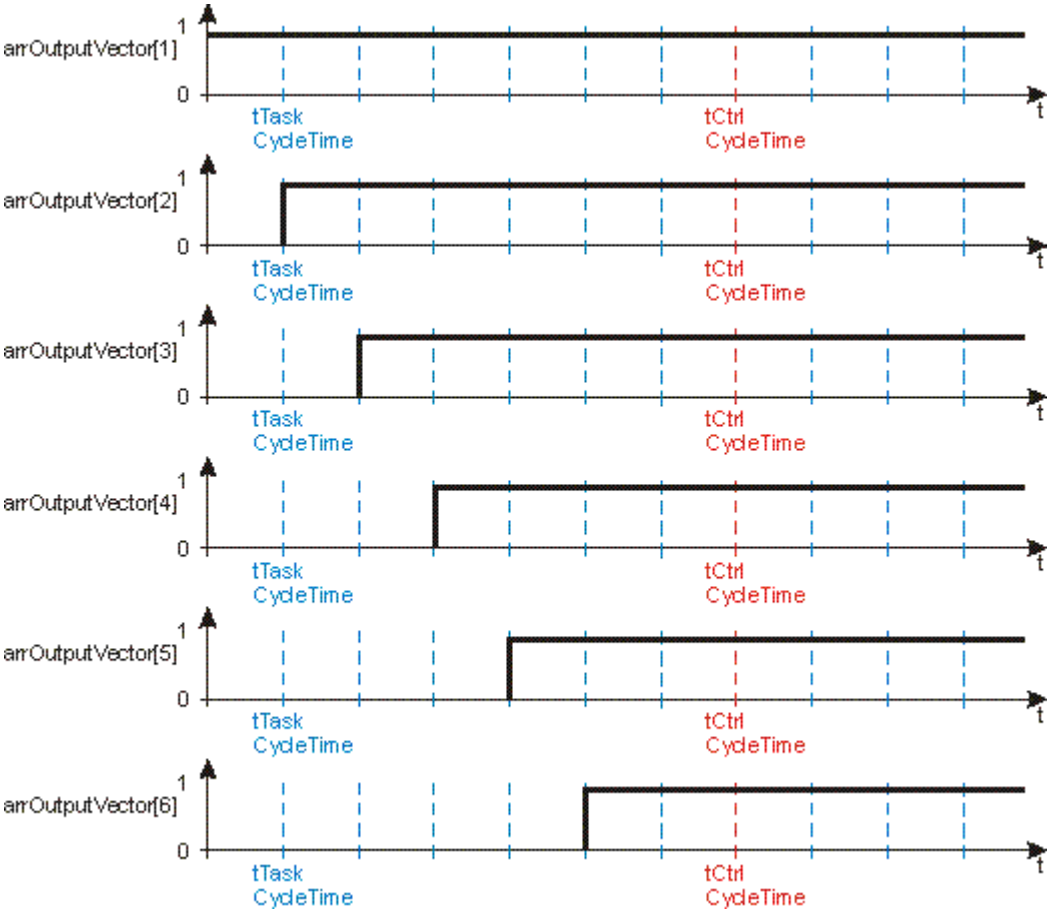
Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib

5.2.3 FB_CTRL_LOOP_SCHEDULER

This function block allows the system loading to be distributed over a number of control loops that are a) parameterised using the same tCtrlCycleTime and b) for which the condition tCtrlCycleTime > tTaskCycleTime is true. The output vector calculated by this block is used to start the individual control loops at different times, so that the system loading is distributed.

Behavior of the output vector:



6 control loops are managed in this diagram. In this case, tCtrlCycleTime = 7 · tTaskCycleTime.

The programmer must create the following array in the PLC if this function block is to be used:

```
arrOutputVector     :
ARRAY[1..nNumberOfControlLoops] OF BOOL;
```

The function block sets the bits in this vector to TRUE or FALSE. The control loops that are managed with the loop scheduler are to be switched into the eCTRL_MODE_ACTIVE state when the corresponding bit in the output vector is TRUE. See sample code below.

VAR_INPUT

```
VAR_INPUT
    nManValue : DWORD;
    eMode     : E_CTRL_MODE;
END_VAR
```

nManValue : This input allows the first 32 bits in the output vector to be set in eCTRL_MODE_MANUAL. A 1 sets the first bit, a 2 the second bit, a 3 the first and second bits, ...

eMode : Input that specifies the operation mode [▶ 16] of the function block.

VAR_OUTPUT

```
VAR_OUTPUT
    eState      : E_CTRL_STATE;
    eErrorId    : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

eState : State of the function block.

nErrorID : Returns the ADS error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_LOOP_SCHEDULER_PARAMS;
END_VAR
```

stParams : Parameter structure for the loop scheduler. This consists of the following elements:

```
TYPE
ST_CTRL_LOOP_SCHEDULER_PARAMS:
STRUCT
    tCtrlCycleTime        : TIME     := T#0ms; (*
controller cycle time [TIME] *)
    tTaskCycleTime        : TIME     := T#0ms; (* task
cycle time [TIME] *)
    nNumberOfControlLoops     : UINT;
    pOutputVector_ADR     : POINTER TO BOOL := 0;
    nOutputVector_SIZEOF      : UINT := 0;
END_STRUCT
END_TYPE
```

tCtrlCycleTime : The cycle time with which the control loops managed by the loop scheduler are processed. This must be greater than or equal to the TaskCycleTime.

tTaskCycleTime : The cycle time with which the loop scheduler and the function blocks associated with the control loops are called. If the block is called in every cycle this corresponds to the task cycle time of the calling task.

nNumberOfControlLoops : The number of control loops being managed.

pOutputVector_ADR :.Address of the output vector.

nOutputVector_SIZEOF :Size of the output vector in bytes.

Sample:

```
PROGRAM PRG_LoopScheduler
VAR
    arrOutputVector    : ARRAY[1..5] OF BOOL;
    eMode          : E_CTRL_MODE;
    stParams           : ST_CTRL_LOOP_SCHEDULER_PARAMS;
    eErrorId           : E_CTRL_ERRORCODES;
    bError         : BOOL;
    fbCTRL_LoopScheduler   : FB_CTRL_LOOP_SCHEDULER;
    bInit          : BOOL := TRUE;
    (* modes of the control loops *)
    eMode_CtrlLoop_1       : E_CTRL_MODE;
    eMode_CtrlLoop_2       : E_CTRL_MODE;
    eMode_CtrlLoop_3       : E_CTRL_MODE;
```

```
    eMode_CtrlLoop_4       : E_CTRL_MODE;
    eMode_CtrlLoop_5       : E_CTRL_MODE;
END_VAR
```

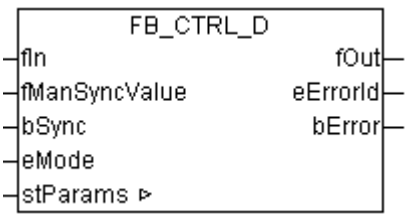
```
IF bInit
THEN
    stParams.tCtrlCycleTime     := T#10ms;
    stParams.tTaskCycleTime     := T#2ms;
    stParams.nNumberOfControlLoops  := 5;
    bInit               := FALSE;
END_IF
(* set addresses *)
stParams.nOutputVector_SIZEOF   := SIZEOF(arrOutputVector);
stParams.pOutputVector_ADR      := ADR(arrOutputVector);
(* call scheduler *)
fbCTRL_LoopScheduler( eMode     := eMode,
              stParams  := stParams,
              eErrorId  => eErrorId,
              bError    => bError);
IF arrOutputVector[ 1 ]
THEN
    (* activate control loop 1 *)
    eMode_CtrlLoop_1 := eCTRL_MODE_ACTIVE;
END_IF
IF arrOutputVector[ 2 ]
THEN
    (* activate control loop 2 *)
    eMode_CtrlLoop_2 := eCTRL_MODE_ACTIVE;
END_IF
IF arrOutputVector[ 3 ]
THEN
    (* activate control loop 3 *)
    eMode_CtrlLoop_3 := eCTRL_MODE_ACTIVE;
END_IF
IF arrOutputVector[ 4 ]
THEN
    (* activate control loop 4 *)
    eMode_CtrlLoop_4 := eCTRL_MODE_ACTIVE;
END_IF
IF arrOutputVector[ 5 ]
THEN
    (* activate control loop 5 *)
    eMode_CtrlLoop_5 := eCTRL_MODE_ACTIVE;
END_IF
```

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9, build 947 onwards	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9, build 956 onwards	BX [▶ 10]	TcControllerToolbox.lbx

5.3 Base

5.3.1 FB_CTRL_D



The function block provides a DT_1 transfer element (a real D element) in a functional diagram.

Transfer function (continuous):

$$G(s) = \frac{T_v s}{1 + T_d s}$$

VAR_INPUT

```
VAR_INPUT
    fIn             : FLOAT;
    fManSyncValue   : FLOAT;
    bSync           : BOOL;
    eMode           : E_CTRL_MODE;
END_VAR
```

fIn : Input of the D element.

fManSyncValue : Input to which the D element can be synchronized, or whose value is present at the output in Manual Mode.

bSync : A rising edge at this input sets the D element to the value fManSyncValue.

eMode : Input that specifies the operation mode [▶ 16] of the function block.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut        : FLOAT;
    eState      : E_CTRL_STATE;
    eErrorId    : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

fOut : Output of the D element.

eState : State of the function block.

nErrorID : Returns the ADS error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_D_PARAMS;
END_VAR
```

stParams : Parameter structure of the D element. This consists of the following elements:

```
TYPE ST_CTRL_D_PARAMS
:
STRUCT
    tCtrlCycleTime     : TIME     :=  T#0ms; (* controller
cycle time [TIME] *)
    tTaskCycleTime     : TIME     :=  T#0ms; (* task cycle time
[TIME] *)
    tTv        : TIME     :=  T#0ms; (* derivative
action time Tv *)
    tTd        : TIME     :=  T#0ms; (* derivative
damping time Td *)
    fOutMaxLimit       : FLOAT    :=  1E38;  (* maximum output
limit *)
    fOutMinLimit       : FLOAT    := -1E38;  (* minimum output
limit *)
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

tTv : Differentiation time constant

tTd : Damping time constant

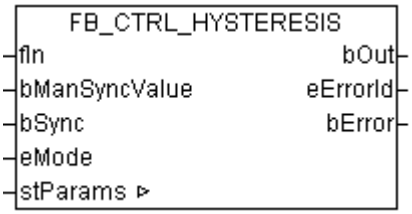
fOutMaxLimit : Upper limit at which the output of the D element is limited.

fOutMinLimit : Lower limit at which the output of the D element is limited.

Requirements

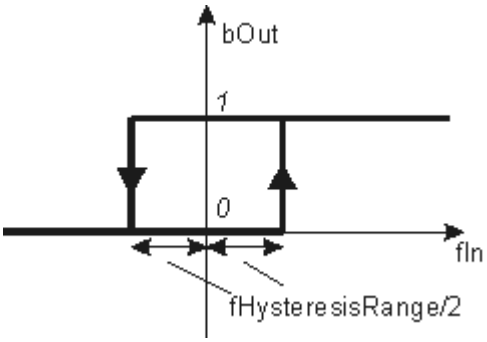
Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.3.2 FB_CTRL_HYSTERESIS



The function block provides a hysteresis transfer element in a functional diagram.

Transfer function:



VAR_INPUT

```
VAR_INPUT
    fIn             : FLOAT;
    bManSyncValue   : BOOL;
    bSync           : BOOL;
    eMode           : E_CTRL_MODE;
END_VAR
```

fIn : Input of the hysteresis element.

bManSyncValue : Input through which the hysteresis element can be set to one of the two junctions.

bSync : A rising edge at this input sets the hysteresis element to the value fManSyncValue.

eMode : Input that specifies the operation mode [▶ 16] of the function block.

VAR_OUTPUT

```
VAR_OUTPUT
    bOut        : BOOL;
    eState      : E_CTRL_STATE;
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

bOut : Output of the hysteresis element.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

bError : Becomes TRUE as soon as an error situation occurs.

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_HYSTERESIS_PARAMS;
END_VAR
```

stParams : Parameter structure of the hysteresis element. This consists of the following elements:

```
TYPE
ST_CTRL_HYSTERESIS_PARAMS :
STRUCT
    tCtrlCycleTime     : TIME := T#0ms; (* controller cycle
time [TIME] *)
    tTaskCycleTime     : TIME := T#0ms; (* task cycle time
[TIME] *)
    fHysteresisRange   : FLOAT; (* range of the hysteresis loop
*)
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

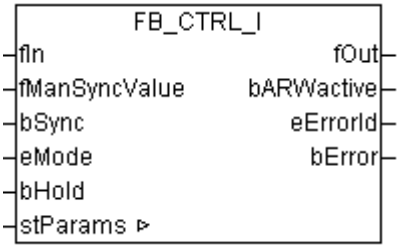
tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

fHysteresisRange : Hysteresis range, see image above.

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 ab Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 ab Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.3.3 FB_CTRL_I



The function block provides an I-transfer element in the functional diagram.

Transfer function:

$$G(s) = \frac{1}{T_I s}$$

VAR_INPUT

```
VAR_INPUT
    fIn          : FLOAT;
    fManSyncValue    : FLOAT;
    bSync        : BOOL;
    eMode        : E_CTRL_MODE;
    bHold        : BOOL;
END_VAR
```

fIn : Input of the I element.

fManSyncValue : Input to which the I-element can be synchronised, or whose value is the present at the output in Manual Mode.

bSync : A rising edge at this input sets the integrator to the value fManSyncValue.

eMode : Input that specifies the block's operating mode.

bHold : A TRUE at this input holds the integrator fixed at the current value, independently of the input fIn.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut         : FLOAT;
    bARWactive       : BOOL;
    eState       : E_CTRL_STATE;
    eErrorId         : E_CTRL_ERRORCODES;
    bError       : BOOL;
END_VAR
```

fOut : Output of the I element.

bARWactive : A TRUE at this output signals that the integrator is being restricted.

eState : State of the function block.

nErrorID : Returns the ADS error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams         : ST_CTRL_I_PARAMS;
END_VAR
```

stParams : Parameter structure of the I element. This consists of the following elements:

```
TYPE ST_CTRL_I_PARAMS
:
STRUCT
    tCtrlCycleTime : TIME  :=  T#0ms; (* controller cycle time
*)
    tTaskCycleTime : TIME  :=  T#0ms; (* task cycle time
*)
    tTi         : TIME  :=  T#0ms; (* integral action time
Ti *)
    fOutMaxLimit    : FLOAT :=  1E38;  (* maximum output limit
*)
    fOutMinLimit    : FLOAT := -1E38;  (* minimum output limit
*)
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

tTi : Integration time of the I element.

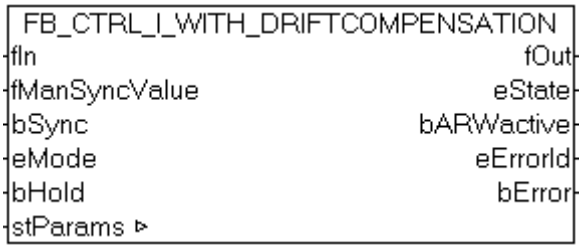
fOutMaxLimit : Upper limit at which integration is halted (ARW measure). Reaching this limit is indicated by a TRUE at the bARWActive output.

fOutMinLimit : Lower limit at which integration is halted (ARW measure). Reaching this limit is indicated by a TRUE at the bARWActive output.

Requirements

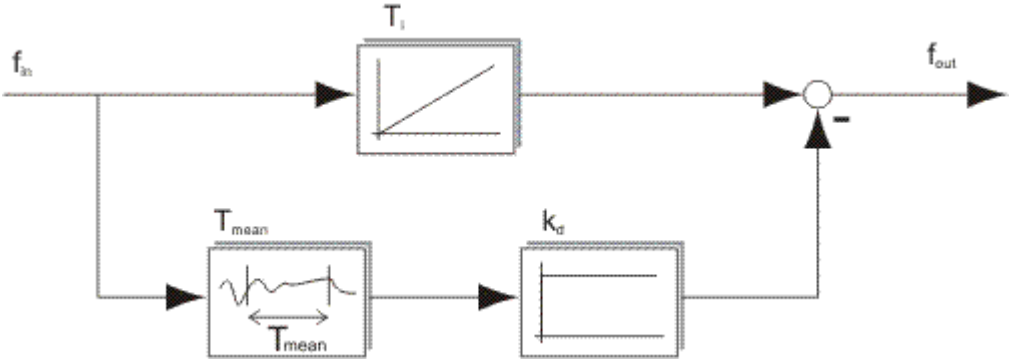
Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.3.4 FB_CTRL_I_WITH_DRIFTCOMPENSATION



The function block represents an I transfer element with drift compensation.

Functional diagram:



VAR_INPUT

```
VAR_INPUT
    fIn          : FLOAT;
    fManSyncValue   : FLOAT;
    bSync        : BOOL;
    eMode        : E_CTRL_MODE;
    bHold        : BOOL;
END_VAR
```

fIn : Input of the I-element.

fManSyncValue : Input to which the I-element can be synchronized, or whose value is the present at the output in Manual Mode.

bSync : A rising edge at this input sets the integrator to the value fManSyncValue.

eMode : Input that specifies the block's operating mode.

bHold : A TRUE at this input holds the integrator fixed at the current value, independently of the input fIn.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut        : FLOAT;
    bARWactive      : BOOL;
    eState      : E_CTRL_STATE;
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

fOut : Output of the I element.

bARWactive : A TRUE at this output signals that the integrator is being restricted.

eState : State of the function block.

nErrorID : Returns the ADS error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_I_WITH_DRIFTCOMPENSATION_PARAMS;
END_VAR
```

stParams : Parameter structure of the I element. This consists of the following elements:

```
TYPE
ST_CTRL_I_WITH_DRIFTCOMPENSATION_PARAMS:
STRUCT
     tCtrlCycleTime      : TIME := T#0ms; (* controller cycle
time [TIME] *)
     tTaskCycleTime      : TIME := T#0ms; (* task cycle time
[TIME] *)
     tTi         : TIME := T#0ms; (* integral action
time Ti *)
     fOutMaxLimit    : FLOAT := 1E38; (* maximum output
limit *)
     fOutMinLimit    : FLOAT := -1E38; (* minimum output
limit *)
     fDampingCoefficient : FLOAT := 0.0;
     tAveragingTime      : TIME := T#0ms; (* averaging time
*)
     pWorkArray_ADR      : POINTER TO FLOAT := 0;
     nWorkArray_SIZEOF   : UINT := 0;
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

tTi : Integration time of the I element.

fOutMaxLimit : Upper limit at which integration is halted (ARW measure). Reaching this limit is indicated by a TRUE at the bARWActive output.

fOutMinLimit : Lower limit at which integration is halted (ARW measure). Reaching this limit is indicated by a TRUE at the bARWActive output.

fDampingCoefficient : Factor k_d in the functional diagram.

tAveragingTime : Time across which the sliding average value filter calculates the average value.

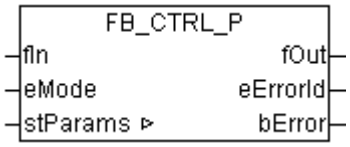
pWorkArray_ADR : address of a FLOAT array with size tAveragingTime / tCtrlCycleTime. (see description of the function block FB_CTRL_MOVING_AVERAGE)

nWorkArray_SIZEOF : size of a FLOAT array with size tAveragingTime / tCtrlCycleTime. (see description of the function block FB_CTRL_MOVING_AVERAGE)

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.3.5 FB_CTRL_P



The function block provides a P-transfer element in the functional diagram.

Transfer function:

$$G(s) = K_p$$

VAR_INPUT

```
VAR_INPUT
    fIn         :FLOAT;
    eMode       :E_CTRL_MODE;
END_VAR
```

fIn : Input of the P element.

eMode : Input that specifies the block's operating mode.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut        :FLOAT;
    eState      :E_CTRL_STATE;
    eErrorId        :E_CTRL_ERRORCODES;
    bError      :BOOL;
END_VAR
```

fOut : Output of the P element.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        :ST_CTRL_P_PARAMS;
END_VAR
```

stParams : Parameter structure of the P element. This consists of the following elements:

```
TYPE ST_CTRL_P_PARAMS
:
STRUCT
    tCtrlCycleTime : TIME    := T#0ms;  (* controller cycle
time *)
    tTaskCycleTime : TIME    := T#0ms;  (* task cycle time
*)
    fKp        : FLOAT   := 0.0;    (* proportional gain Kp
*)
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

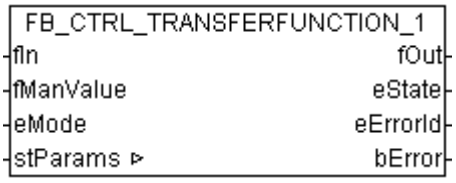
tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

fKp : Proportional gain of the P element.

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.3.6 FB_CTRL_TRANSFERFUNCTION_1



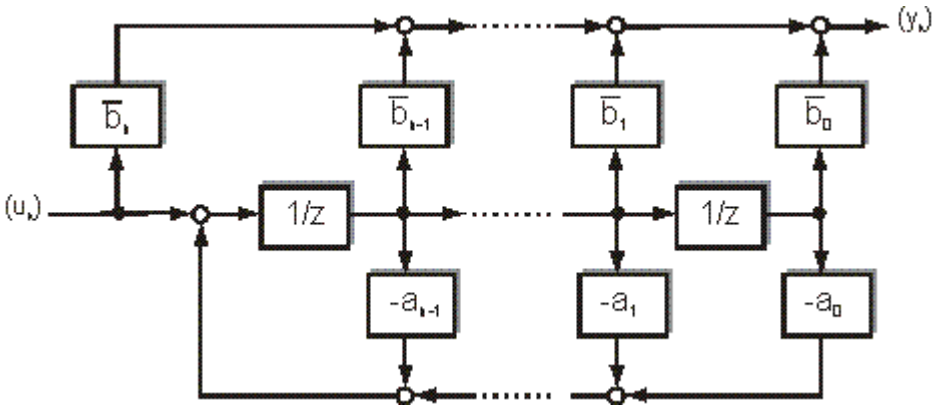
This function block calculates a discrete transfer function with the first standard form illustrated below. The transfer function here can be of any order, n.

The coefficients for the following transfer functions are stored in the parameter arrays:

$$G(z) = \frac{b_n z^n + b_{(n-1)} z^{(n-1)} + \cdots + b_1 z + b_0}{z^n + a_{(n-1)} z^{(n-1)} + \cdots + a_1 z + a_0}$$

Description of the transfer behavior:

The transfer function above is calculated with the first standard form, after some transformations, in every sampling step.



The programmer must create the following arrays in the PLC if this function block is to be used:

```
ar_fNumeratorArray     :
ARRAY[0..nOrderOfTheTransferfunction] OF FLOAT;
ar_DenominatorArray    : ARRAY[0..nOrderOfTheTransferfunction]
OF FLOAT;
ar_stTransferfunction1Data : ARRAY[0..nOrderOfTheTransferfunction]
OF ST_CTRL_TRANSFERFUNCTION_1_DATA;
```

The coefficients b_0 to b_n are stored in the array ar_fNumeratorArray. This must be organized as follows:

```
ar_fNumeratorArray[
0 ]     := b0;
ar_fNumeratorArray[ 1 ]     := b1;
...
ar_fNumeratorArray[ n-1 ]   := bn-1;
ar_fNumeratorArray[ n ]     := bn;
```

The coefficients a_0 to a_n are stored in the array ar_DenominatorArray. This must be organized as follows:

```
ar_DenominatorArray
[ 0 ]    := a0;
ar_DenominatorArray [ 1 ]    := a1;
...
ar_DenominatorArray [ n-1 ]  := an-1;
ar_DenomiantorArray [ n ]    := an;
```

The internal data required by the function block is stored in the array ar_stTransferfunction1Data. This data must never be modified from within the PLC program. This procedure is also illustrated in the sample program listed below.

VAR_INPUT

```
VAR_INPUT
    fIn          : FLOAT;
    fManValue       : FLOAT;
    eMode        : E_CTRL_MODE;
END_VAR
```

fIn : Input to the transfer function.

fManValue : Input whose value is present at the output in manual mode.

eMode : Input that specifies the block's operating mode.

VAR_OUTPUT

```
VAR_OUTPUT
 fOut         : FLOAT;
    eState       : E_CTRL_STATE;
    bError       : BOOL;
    eErrorId        : E_CTRL_ERRORCODES;
END_VAR
```

fOut : Output from the transfer function.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_TRANSFERFUNCTION_1_PARAMS;
END_VAR
```

stParams : Parameter structure of the function block. This consists of the following elements:

```
TYPE
ST_CTRL_TRANSFERFUNCTION_1_PARAMS:
STRUCT
    tTaskCycleTime        : TIME;      (* task
cycle time in seconds *)
    tCtrlCycleTime        : TIME := T#0ms; (*
controller cycle time *)
    nOrderOfTheTransferfunction   : USINT;
    pNumeratorArray_ADR       : POINTER TO FLOAT :=
0;
    nNumeratorArray_SIZEOF    : UINT;
    pDenominatorArray_ADR     : POINTER TO FLOAT :=
0;
    nDenomiantorArray_SIZEOF      : UINT;
    pTransferfunction1Data_ADR    : POINTER TO
ST_CTRL_TRANSFERFUNCTION_1_DATA;
    nTransferfunction1Data_SIZEOF : UINT;
END_TYPE
```

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

nOrderOfTheTransferfunction : Order of the transfer function [0...]

pNumeratorArray_ADR : Address of the array with the numerator coefficients.

nNumeratorArray_SIZEOF : Size of the array with the numerator coefficients in bytes.

pDenominatorArray_ADR : Address of the array with the denominator coefficients.

nDenominatorArray_SIZEOF : Size of the array with the denominator coefficients in bytes.

pTransferfunction1Data_ADR : Address of the data array.

nTransferfunction1Data_SIZEOF : Size of the data array in bytes.

```
TYPE
ST_CTRL_TRANSFERFUNCTION_1_DATA:
STRUCT
    Interne Struktur. Auf diese darf nicht schreibend
zugegriffen werden.
END_STRUCT
END_TYPE
```

Sample:

```
PROGRAM PRG_TRANSFERFUNCTION_1_TEST
VAR CONSTANT
    nOrderOfTheTransferfunction : USINT := 2;
END_VAR
VAR
    ar_fNumeratorArray     :
ARRAY[0..nOrderOfTheTransferfunction] OF FLOAT;
    ar_DenominatorArray    :
ARRAY[0..nOrderOfTheTransferfunction] OF FLOAT;
    ar_stTransferfunction1Data :
ARRAY[0..nOrderOfTheTransferfunction] OF
ST_CTRL_TRANSFERFUNCTION_1_DATA;
    eMode     : E_CTRL_MODE;
    stParams      : ST_CTRL_TRANSFERFUNCTION_1_PARAMS;
    eErrorId      : E_CTRL_ERRORCODES;
    bError    : BOOL;
```

```
    fbTansferfunction : FB_CTRL_TRANSFERFUNCTION_1;
    bInit     : BOOL := TRUE;
    fIn       : FLOAT := 0;
    fOut      : FLOAT;
    b_0, b_1, b_2   : FLOAT;
    a_0,a_1,a_2     : FLOAT;
END_VAR
IF bInit
THEN
    (* set values in the local arrays *)
    ar_fNumeratorArray[0] := 1.24906304658218E-007;
    ar_fNumeratorArray[1] := 2.49812609316437E-007;
    ar_fNumeratorArray[2] := 1.24906304658218E-007;
    ar_DenominatorArray[0] := 0.998501124344101;
    ar_DenominatorArray[1] := -1.99850062471888;
    ar_DenominatorArray[2] := 1.0;
    (* set values in the parameter struct *)
    stParams.tTaskCycleTime        := T#2ms;
    stParams.tCtrlCycleTime        := T#2ms;
    stParams.nOrderOfTheTransferfunction   :=
nOrderOfTheTransferfunction;
    (* set the mode *)
    eMode := eCTRL_MODE_ACTIVE;
    bInit := FALSE;
END_IF
(* set the addresses *)
stParams.pNumeratorArray_ADR       := ADR(
ar_fNumeratorArray);
stParams.nNumeratorArray_SIZEOF    := SIZEOF(
ar_fNumeratorArray);
stParams.pDenominatorArray_ADR     := ADR( ar_DenominatorArray
);
stParams.nDenominatorArray_SIZEOF      := SIZEOF(
ar_DenominatorArray );
stParams.pTransferfunction1Data_ADR    := ADR(
ar_stTransferfunction2Data );
stParams.nTransferfunction1Data_SIZEOF := SIZEOF(
ar_stTransferfunction2Data );
(* call the function block *)
fbTansferfunction ( fIn       := fIn,
            eMode     := eMode,
            stParams  := stParams,
            fOut      => fOut,
            eErrorId  => eErrorId,
            bError    => bError
            );
```

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.3.7 FB_CTRL_TRANSFERFUNCTION_2

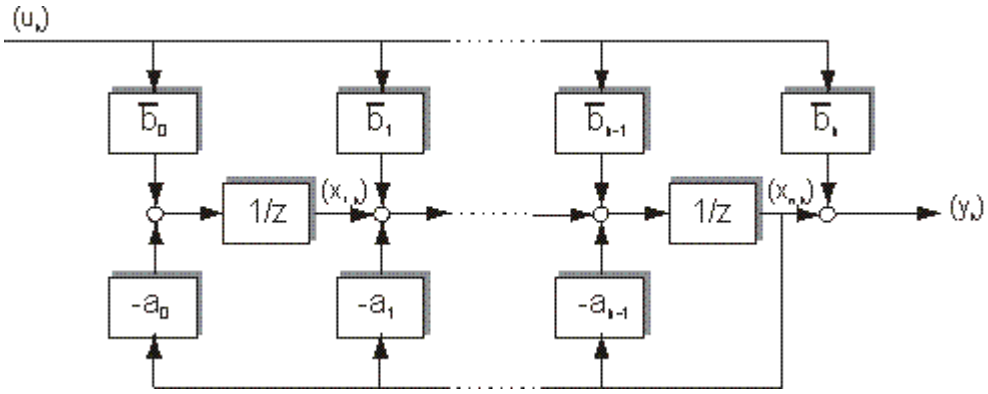
This function block calculates a discrete transfer function with the second standard form illustrated below. The transfer function here can be of any order, n.

The coefficients for the following transfer functions are stored in the parameter arrays:

$$G(z) = \frac{b_n z^n + b_{(n-1)} z^{(n-1)} + \cdots + b_1 z + b_0}{z^n + a_{(n-1)} z^{(n-1)} + \cdots + a_1 z + a_0}$$

Description of the transfer behavior:

The internal calculation is performed in each sampling step according to the second standard form, for which the following block diagram applies:



The transfer function can be brought into the form shown in the block diagram by means of some transformations:

$$G(z) = \tilde{b}_n + \frac{\tilde{b}_{(n-1)} z^{-1} + \cdots + \tilde{b}_1 z^{-(n-1)} + \tilde{b}_0 z^{-n}}{1 + a_{(n-1)} z^{-1} + \cdots + a_1 z^{-(n-1)} + a_0 z^{-n}}$$

Fig. 1: FB_CTRL_TRANSFERFUNCTION_2_G2

The coefficients of the numerator polynomial are calculated according to the following rule:

$$\begin{aligned} \tilde{b}_i &= b_i - b_n a_i \quad \forall\ 0 \le i < n \\ \tilde{b}_n &= b_n \end{aligned}$$

Fig. 2: FB_CTRL_TRANSFERFUNCTION_2_bi

The programmer must create the following arrays in the PLC if this function block is to be used:

```
ar_fNumeratorArray     :
ARRAY[0..nOrderOfTheTransferfunction] OF FLOAT;
ar_DenominatorArray    : ARRAY[0..nOrderOfTheTransferfunction]
OF FLOAT;
ar_stTransferfunction2Data : ARRAY[0..nOrderOfTheTransferfunction]
OF ST_CTRL_TRANSFERFUNCTION_2_DATA;
```

The coefficients b_0 to b_n are stored in the array ar_fNumeratorArray. This must be organized as follows:

```
ar_fNumeratorArray[
0 ]      := b0;
ar_fNumeratorArray[ 1 ]      := b1;
...
ar_fNumeratorArray[ n-1 ]    := bn-1;
ar_fNumeratorArray[ n ]      := bn;
```

The coefficients a_0 to a_n are stored in the array ar_DenominatorArray. This must be organized as follows:

```
ar_DenominatorArray
[ 0 ]     := a0;
ar_DenominatorArray [ 1 ]     := a1;
...
ar_DenominatorArray [ n-1 ]   := an-1;
ar_DenomiantorArray [ n ]     := an;
```

The internal data required by the function block is stored in the array ar_stTransferfunction2Data. This data must never be modified from within the PLC program. This procedure is also illustrated in the sample program listed below.

VAR_INPUT

```
VAR_INPUT
    fIn          : FLOAT;
    fManValue        : FLOAT;
    eMode        : E_CTRL_MODE;
END_VAR
```

fIn : Input to the transfer function.

fManValue : Input whose value is present at the output in manual mode.

eMode : Input that specifies the block's operating mode.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut         : FLOAT;
    eState       : E_CTRL_STATE;
    bError       : BOOL;
    eErrorId         : E_CTRL_ERRORCODES;
END_VAR
```

fOut : Output from the transfer function.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams         : ST_CTRL_TRANSFERFUNCTION_2_PARAMS;
END_VAR
```

stParams : Parameter structure of the function block. This consists of the following elements:

```
TYPE
ST_CTRL_TRANSFERFUNCTION_2_PARAMS:
STRUCT
    tTaskCycleTime         : TIME;      (* task
cycle time in seconds *)
    tCtrlCycleTime         : TIME := T#0ms; (*
controller cycle time *)
    nOrderOfTheTransferfunction   : USINT;
    pNumeratorArray_ADR        : POINTER TO FLOAT :=
0;
    nNumeratorArray_SIZEOF     : UINT;
    pDenominatorArray_ADR      : POINTER TO FLOAT :=
0;
    nDenomiantorArray_SIZEOF       : UINT;
    pTransferfunction2Data_ADR     : POINTER TO
ST_CTRL_TRANSFERFUNCTION_2_DATA;
    nTransferfunction2Data_SIZEOF : UINT;
END_TYPE
```

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

nOrderOfTheTransferfunction : Order of the transfer function [0...]

pNumeratorArray_ADR : Address of the array with the numerator coefficients.

nNumeratorArray_SIZEOF : Size of the array with the numerator coefficients in bytes.

pDenominatorArray_ADR : Address of the array with the denominator coefficients.

nDenominatorArray_SIZEOF : Size of the array with the denominator coefficients in bytes.

pTransferfunction2Data_ADR : Address of the data array.

nTransferfunction2Data_SIZEOF : Size of the data array in bytes.

```
TYPE
ST_CTRL_TRANSFERFUNCTION_2_DATA:
STRUCT
    Interne Struktur. Auf diese darf nicht schreibend
zugegriffen werden.
END_STRUCT
END_TYPE
```

Sample:

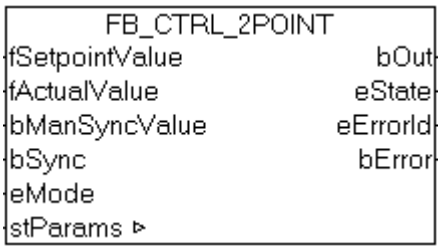
```
PROGRAM PRG_TRANSFERFUNCTION_2_TEST
VAR CONSTANT
    nOrderOfTheTransferfunction : USINT := 2;
END_VAR
VAR
    ar_fNumeratorArray     :
ARRAY[0..nOrderOfTheTransferfunction] OF FLOAT;
    ar_DenominatorArray    :
ARRAY[0..nOrderOfTheTransferfunction] OF FLOAT;
    ar_stTransferfunction2Data :
ARRAY[0..nOrderOfTheTransferfunction] OF
ST_CTRL_TRANSFERFUNCTION_2_DATA;
    eMode     : E_CTRL_MODE;
    stParams      : ST_CTRL_TRANSFERFUNCTION_2_PARAMS;
    eErrorId      : E_CTRL_ERRORCODES;
    bError    : BOOL;
    fbTansferfunction : FB_CTRL_TRANSFERFUNCTION_2;
    bInit     : BOOL := TRUE;
    fIn       : FLOAT := 0;
    fOut      : FLOAT;
    b_0, b_1, b_2   : FLOAT;
    a_0,a_1,a_2     : FLOAT;
END_VAR
IF bInit
THEN
    (* set values in the local arrays *)
    ar_fNumeratorArray[0] := 1.24906304658218E-007;
    ar_fNumeratorArray[1] := 2.49812609316437E-007;
    ar_fNumeratorArray[2] := 1.24906304658218E-007;
    ar_DenominatorArray[0] := 0.998501124344101;
    ar_DenominatorArray[1] := -1.99850062471888;
    ar_DenominatorArray[2] := 1.0;
    (* set values in the parameter struct *)
    stParams.tTaskCycleTime := T#2ms;
    stParams.tCtrlCycleTime := T#2ms;
    stParams.nOrderOfTheTransferfunction :=
nOrderOfTheTransferfunction;
    (* set the mode *)
    eMode := eCTRL_MODE_ACTIVE;
    bInit := FALSE;
END_IF
(* set the addresses *)
stParams.pNumeratorArray_ADR := ADR( ar_fNumeratorArray);
stParams.nNumeratorArray_SIZEOF := SIZEOF(
ar_fNumeratorArray);
stParams.pDenominatorArray_ADR := ADR( ar_DenominatorArray );
stParams.nDenominatorArray_SIZEOF := SIZEOF( ar_DenominatorArray
);
stParams.pTransferfunction2Data_ADR := ADR(
ar_stTransferfunction2Data );
stParams.nTransferfunction2Data_SIZEOF := SIZEOF(
ar_stTransferfunction2Data );
fbTansferfunction ( fIn       := fIn,
            eMode     := eMode,
            stParams  := stParams,
            fOut      => fOut,
            eErrorId  => eErrorId,
            bError    => bError
            );
```

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

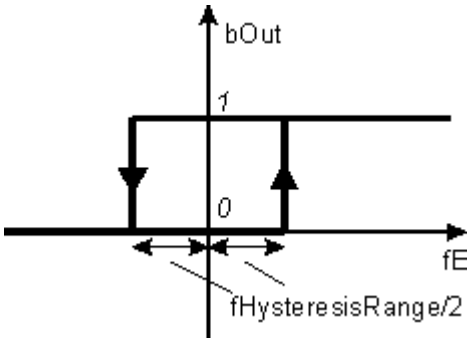
5.4 Controller

5.4.1 FB_CTRL_2POINT



The function block provides a 2-point transfer element in the functional diagram.

Behaviour of the output:



VAR_INPUT

```
VAR_INPUT
    fSetpointValue  : FLOAT;
    fActualValue    : FLOAT;
    bManSyncValue   : BOOL;
    bSync       : BOOL;
    eMode       : E_CTRL_MODE;
END_VAR
```

fSetpointValue : Set value of the controlled variable.

fActualValue : Actual value of the controlled variable.

bManSyncValue : Input through which the 2-point element can be set to one of the two branches.

bSync : A rising edge at this input sets the 2-point element to the value bManSyncValue.

eMode : Input that specifies the block's operating mode [▶ 16].

VAR_OUTPUT

```
VAR_OUTPUT
    bOut        : BOOL;
    eState      : E_CTRL_STATE;
```

```
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

bOut : Output of the 2-point element.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_2POINT_PARAMS;
END_VAR
```

stParams : Parameter structure of the 2-point element. This consists of the following elements:

```
TYPE
ST_CTRL_2POINT_PARAMS :
STRUCT
    tCtrlCycleTime     : TIME     := T#0ms; (* controller cycle
time [TIME] *)
    tTaskCycleTime     : TIME     := T#0ms; (* task cycle time
[TIME] *)
    fHysteresisRange   : FLOAT;         (* range of the
hysteresis loop *)
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

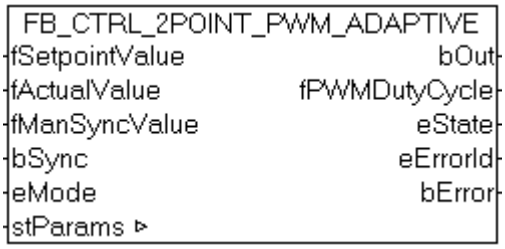
tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

fHysteresisRange: Hysteresis range, see image above.

Requirements

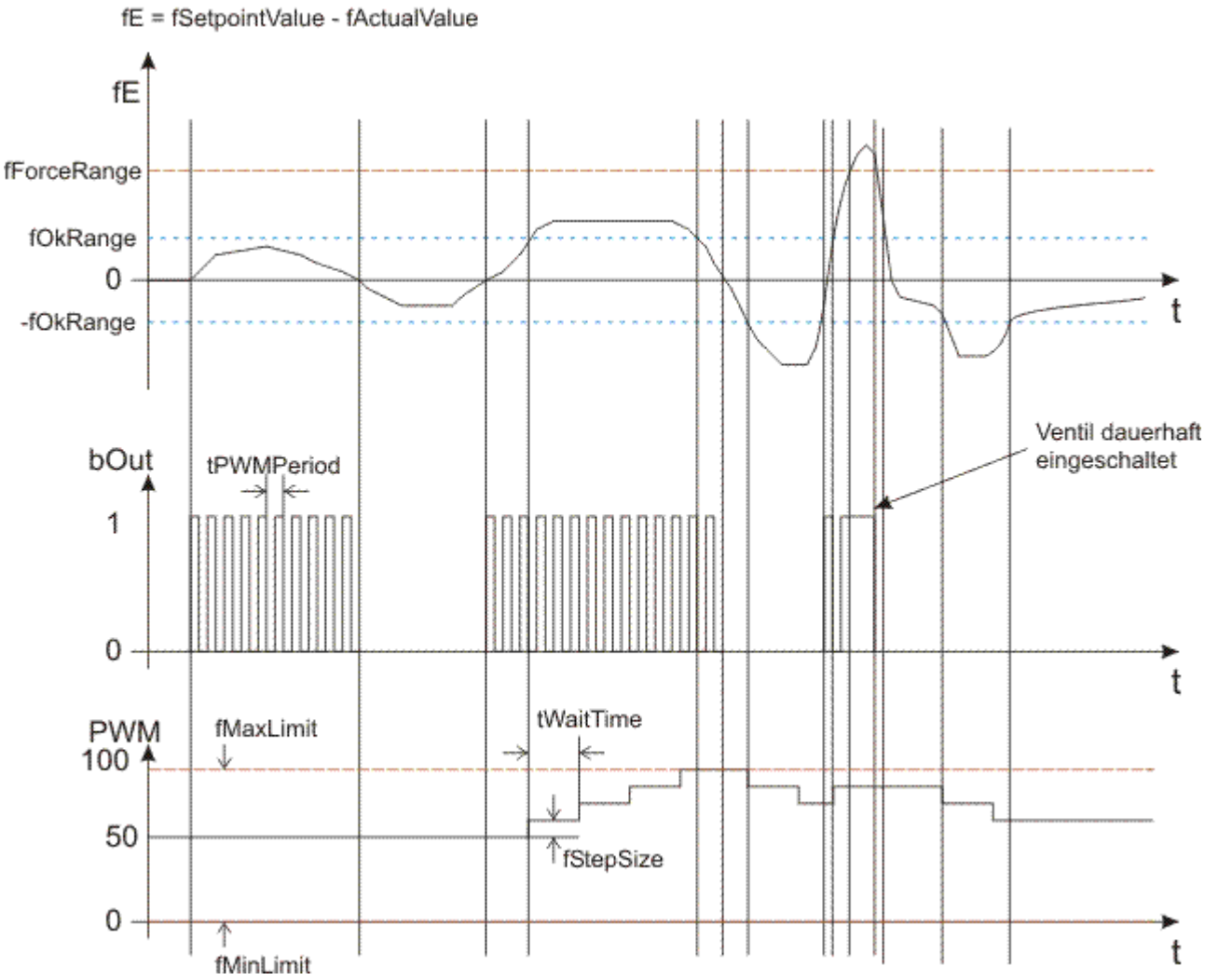
Development environment	Target platform	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.4.2 FB_CTRL_2POINT_PWM_ADAPTIVE



The function block provides an adaptive two-position controller. It is particularly suitable for single-area controllers in which high flow temperatures are present and which make use of a thermal actuator.

Behaviour of the output:



Description of the function:

Internally, the controller uses a PWM function block that is used to control the thermal actuator. The mark-to-space ratio of the PWM function block is adaptively adjusted to the behavior of the controlled system. The PWM output is switched on as soon as the system deviation, fE, which is the setpoint minus the actual value, is greater than zero, and is switched off when the system deviation is less than zero. The mark-to-space ratio is not changed as long as the system deviation remains within the range [-fOkRange ... fOkRange]. If fE > fOkRange, the mark-to-space ratio is increased by fStepSize. After such an increase, time tWaitTime must elapse before the mark-to-space ratio can be changed again. If fE falls below -fOkRange, the mark-to-space ratio is reduced by fStepSize. The mark-to-space ratio is only modified over the range [fMinLimit ... fMaxLimit]. The period of the PWM signal is specified by the parameter tPWMPeriod.

VAR_INPUT

```
VAR_INPUT
    fSetpointValue  : FLOAT;
    fActualValue    : FLOAT;
    fManSyncValue   : FLOAT;
    bSync       : BOOL;
    eMode       : E_CTRL_MODE;
END_VAR
```

fSetpointValue : Set value of the controlled variable.

fActualValue : Actual value of the controlled variable.

fManSyncValue : Input to which the controller's mark-to-space ratio can be set, or with which the output can be set in Manual Mode. The output is set in Manual Mode if fManSyncValue > 0.0.

bSync : A rising edge at this input will set the mark-to-space ratio of the internal PWM block to the value fManSyncValue.

eMode : Input that specifies the block's operating mode [▶ 16].

VAR_OUTPUT

```
VAR_OUTPUT
    bOut        : BOOL;
    fPWMDutyCycle   : FLOAT;            (* debug only *)
    eState      : E_CTRL_STATE;
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

bOut : Output of the controller.

fPWMDutyCycle : Current mark-to-space ratio of the internal PWM block.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_2POINT_PWM_ADAPTIVE_PARAMS;
END_VAR
```

stParams : Parameter structure of the 2-point element. This consists of the following elements:

```
TYPE
ST_CTRL_2POINT_PWM_ADAPTIVE_PARAMS:
STRUCT
    tCtrlCycleTime     : TIME     := T#0ms; (* controller cycle
[TIME] *)
    tTaskCycleTime     : TIME     := T#0ms; (* task cycle time
[TIME] *)
    tPWMPeriod     : TIME
    fOkRange       : FLOAT
    fForceRange    : FLOAT
    fStepSize      : FLOAT
    fMinLimit      : FLOAT           (* [0% ... 100%]
*)
    fMaxLimit      : FLOAT           (* [0% ... 100%]
*)
    tWaitTime      : TIME
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

tPMWPeriod : Period of the PWM signal.

fOkRange : The range of fE over which the mark-to-space ratio will not be modified.

fForceRange : If fE exceeds this range, the output is permanently set to TRUE.

fStepSize : Value by which the mark-to-space ratio is varied each time it is adapted. [0% ... 100%]

fMaxLimit : Maximum mark-to-space ratio in percent [0% ... 100%].

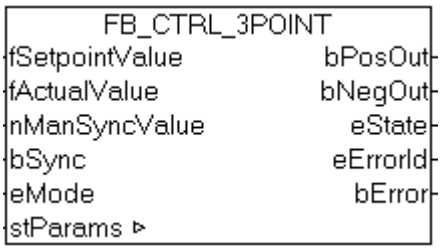
fMinLimit : Minimum mark-to-space ratio in percent [0% ... 100%].

tWaiTimet : Waiting time between individual modifications of the mark-to-space ratio.

Requirements

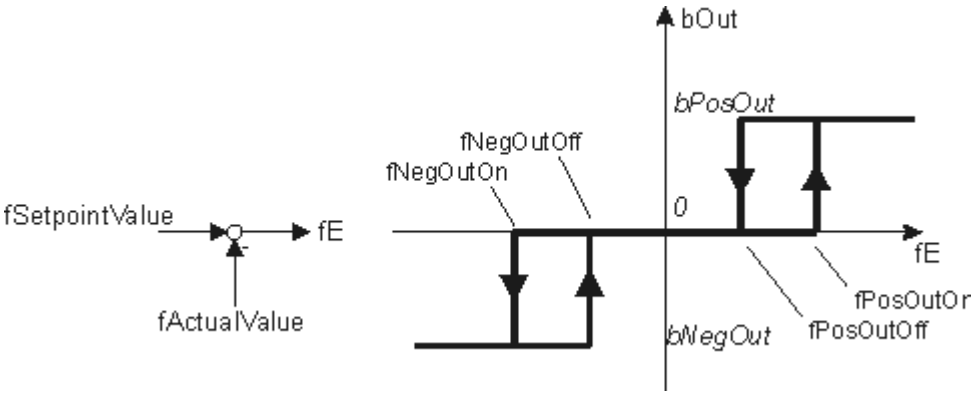
Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.4.3 FB_CTRL_3POINT



The function block provides a 3-point transfer element in the functional diagram.

Behaviour of the output:



VAR_INPUT

```
VAR_INPUT
    fSetpointValue  : FLOAT;
    fActualValue    : FLOAT;
    nManSyncValue   : INT;
    bSync       : BOOL;
    eMode       : E_CTRL_MODE;
END_VAR
```

fSetpointValue : Set value of the controlled variable.

fActualValue : Actual value of the controlled variable.

nManSyncValue : Input through which the 3-point element can be set to one of the three branches.

nManSyncValue >= 1 → bPosOut = TRUE, bNegOut = FALSE
nManSyncValue <= -1 → bPosOut = FALSE, bNegOut = TRUE
otherwise → bPosOut = FALSE, bNegOut = FALSE

bSync : A rising edge at this input sets the 3-point element to the value fManSyncValue.

eMode : Input that specifies the block's operating mode [▶ 16].

VAR_OUTPUT

```
VAR_OUTPUT
    bPosOut          : BOOL;
    bNegOut          : BOOL;
    eState       : E_CTRL_STATE;
    eErrorId         : E_CTRL_ERRORCODES;
    bError       : BOOL;
END_VAR
```

bPosOut : This output from the 3-point element is TRUE if the upper junction of the characteristic curve is active.

bNegOut : This output from the 3-point element is TRUE if the lower junction of the characteristic curve is active.

eState : State of the function block.

nErrorID : Returns the ADS error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams         : ST_CTRL_3POINT_PARAMS;
END_VAR
```

stParams : Parameter structure of the 3-point element. This consists of the following elements:

```
TYPE
ST_CTRL_3POINT_PARAMS :
STRUCT
    tCtrlCycleTime     : TIME     := T#0ms; (* controller cycle
time [TIME] *)
    tTaskCycleTime     : TIME     := T#0ms; (* task cycle time
[TIME] *)
    fPosOutOn      : FLOAT;         (* switch to
bPosOut on *)
    fPosOutOff     : FLOAT;         (* switch to
bPosOut off *)
    fNegOutOn      : FLOAT;         (* switch to
bNegOut on *)
    fNegOutOff     : FLOAT;         (* switch to
bNegOut off *)
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

fPosOutOn : Control deviation that will result in bPosOut = FALSE being switched to bPosOut = TRUE (bNegOut = FALSE).

fPosOutOff : Control deviation that will result in bPosOut = TRUE being switched to bPosOut = FALSE (bNegOut = FALSE).

fNegOutOn : Control deviation that will result in bNegOut = FALSE being switched to bNegOut = TRUE (bPosOut = FALSE).

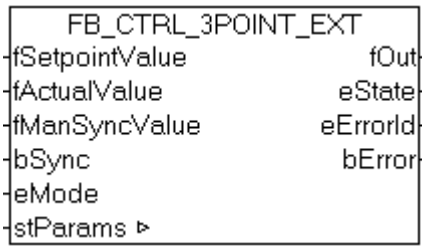
fNegOutOff : Control deviation that will result in bNegOut = TRUE being switched to bNegOut = FALSE (bPosOut = FALSE).

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6

TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

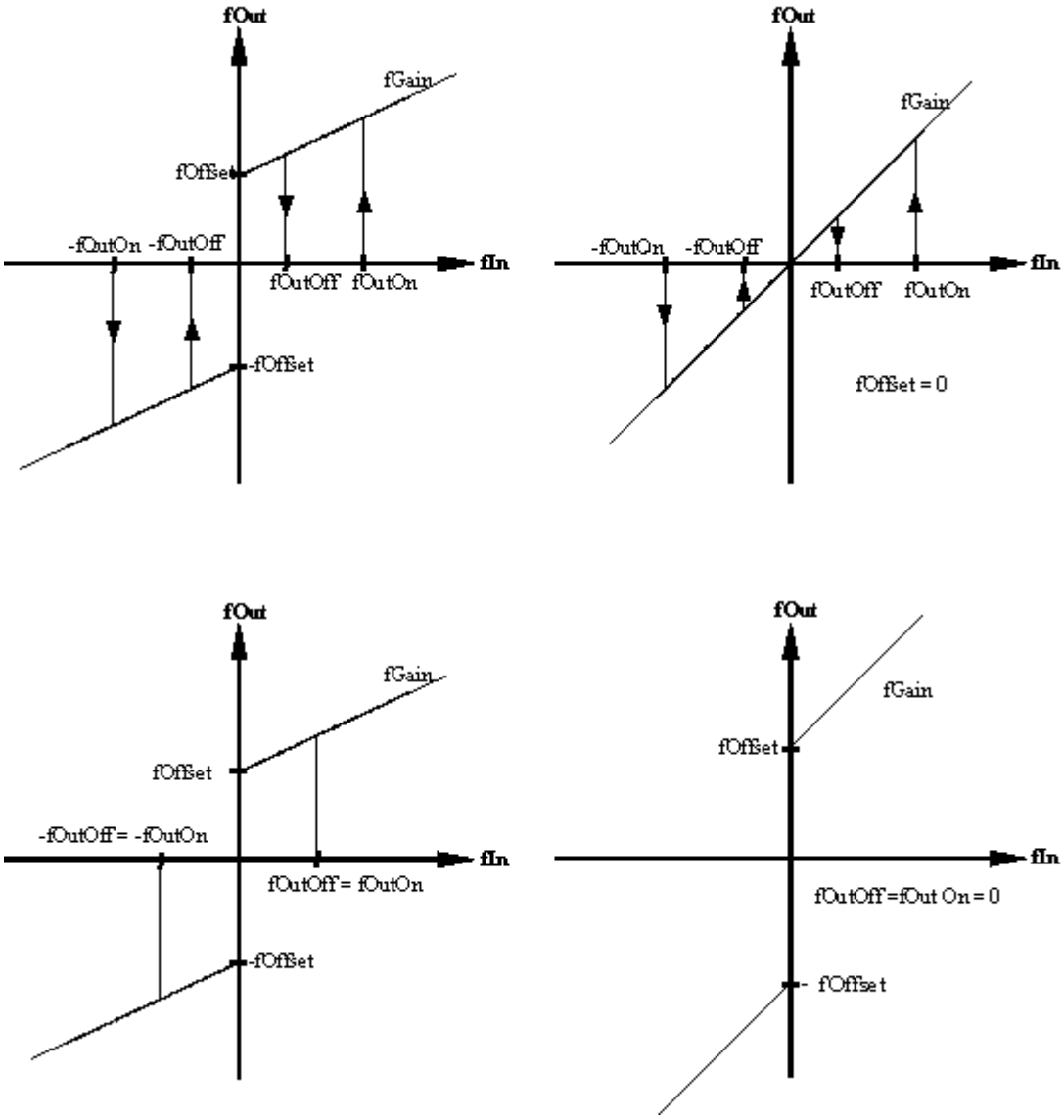
5.4.4 FB_CTRL_3POINT_EXT



The function block provides an extended 3-point element in the functional diagram.

Behaviour of the output:

fIn := fSetpointValue - fActualValue;



VAR_INPUT

```
VAR_INPUT
    fSetpointValue  : FLOAT;
    fActualValue    : FLOAT;
    fManSyncValue   : FLOAT;
    bSync           : BOOL;
    eMode           : E_CTRL_MODE;
END_VAR
```

fSetpointValue : Setpoint of the controlled variable.

fActualValue : Actual value of the controlled variable.

fManSyncValue : Input through which the extended 3-point element can be set to one of the junctions.

| fManSyncValue | < 1 → fOut = 0.0
| fManSyncValue | >= 1 → fOut = fE * fGain + fOffset

bSync : A rising edge at this input sets the 3-point element to the value fManSyncValue.

eMode : Input that specifies the operation mode [▶ 16] of the function block.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut     : FLOAT;
    eState   : E_CTRL_STATE;
    eErrorId : E_CTRL_ERRORCODES;
    bError   : BOOL;
END_VAR
```

fOut : Output of the extended 3-point element.

eState : State of the function block.

nErrorID : Returns the ADS error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_3POINT_EXT_PARAMS;
END_VAR
```

stParams : Parameter structure of the extended 3-point element. This consists of the following elements:

```
TYPE
ST_CTRL_3POINT_EXT_PARAMS :
STRUCT
    tCtrlCycleTime     : TIME := T#0ms; (* controller cycle
time [TIME] *)
    tTaskCycleTime     : TIME := T#0ms; (* task cycle time
[TIME] *)
    fOutOff        : FLOAT;     (* x-parameter for
threshold OFF *)
    fOutOn         : FLOAT;     (* x-parameter for
threshold ON *)
    fGain          : FLOAT;     (* y-parameter gain
*)
    fOffset        : FLOAT;     (* y-parameter offset
*)
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

fOutOff : If the system deviation falls below this value, the output is switched off (set to zero).

fOutOn : If the control deviation exceeds this value, the output is switched on.

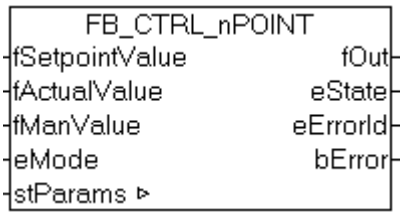
fGain : Gain factor.

fOffset : Offset.

Requirements

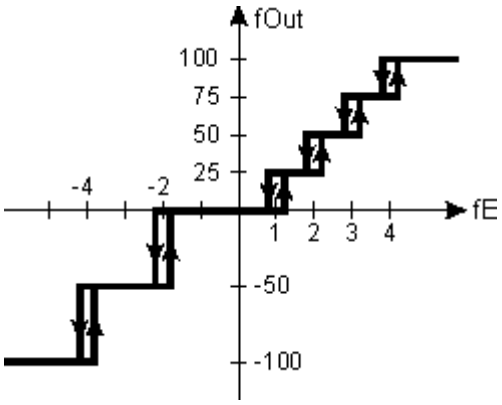
Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.4.5 FB_CTRL_nPOINT



The function block provides an n-point transfer element in the functional diagram.

Behaviour of the output:



Data array for the example:

fE	fOut
xx	-100
-4	-50
-2	0
1	25
2	50
3	75
4	100

The value of the array with index (1,1), that is the left-hand value in the first row, can be freely selected, because it is not evaluated.

VAR_INPUT

```
VAR_INPUT
    fSetpointValue  : FLOAT;
    fActualValue    : FLOAT;
    fManValue       : BOOL;
    eMode       : E_CTRL_MODE;
END_VAR
```

fSetpointValue : Set value of the controlled variable.

fActualValue : Actual value of the controlled variable.

fManValue : Input whose value is output in manual mode.

eMode : Input that specifies the block's operating mode [▶ 16].

VAR_OUTPUT

```
VAR_OUTPUT
    fOut        : nPOINT_CTRL_TABLE_ELEMENT;
    eState      : E_CTRL_STATE;
    eErrorId    : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

fOut : Output of the n-point element.

eState : State of the function block.

nErrorID : Returns the ADS error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_nPOINT_PARAMS;
END_VAR
```

stParams : Parameter structure of the n-point element. This consists of the following elements:

```
TYPE
ST_CTRL_nPOINT_PARAMS :
STRUCT
    tCtrlCycleTime        : TIME     := T#0ms; (*
controller cycle time [TIME] *)
    tTaskCycleTime        : TIME     := T#0ms; (* task
cycle time [TIME] *)
    pDataTable_ADR        : POINTER TO
nPOINT_CTRL_TABLE_ELEMENT := 0;
    nDataTable_SIZEOF     : UINT := 0;
    nDataTable_NumberOfRows   : UINT := 0;
    fHysteresisRange      : FLOAT;         (* range of
the hysteresis loop *)
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

pDataTable_ADR : Address of the data table.

nDataTable_SIZEOF : Size of the data table in bytes.

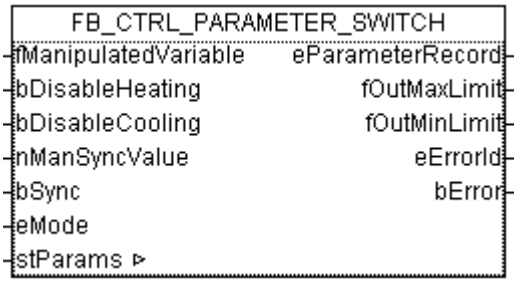
nDataTable_NumberOfRows : Number of rows in the data table.

fHysteresisRange : Hysteresis range, see image above. The hysteresis range functions as described for FB_CTRL_2POINT [▶ 39].

Requirements

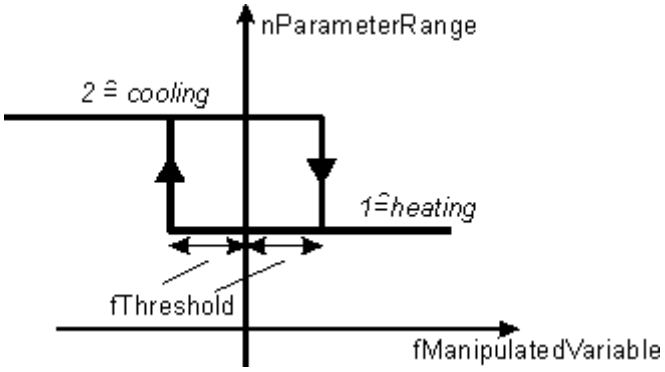
Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.4.6 FB_CTRL_PARAMETER_SWITCH



This function block can be used to switch the parameter set used by FB_CTRL_PID_SPLITRANGE.

Behaviour of the output:



Description of the function block:

This function block is used to switch over the parameter set used by **FB_CTRL_PID_SPLITRANGE**. This function block is particularly intended to switch the parameter sets of controllers that can use two actuators to heat and to cool, and to set the limits for the controller. The time **tMinWaitTime** is specified as an input parameter. At least this time must elapse when switch-over is requested to allow for the parameter range to be changed, and for the controller limits set in such a way that it is possible to switch from heating operation to cooling operation. The intention of this is to prevent the operation mode being changed immediately simply because the controller overshoots slightly.

For heating operation, the parameter range **eCTRL_PARAMETER_RECORD_HEATING = heating** is selected, while for cooling operation the parameter range is **eCTRL_PARAMETER_RECORD_COOLING = cooling**. The controller's parameter sets must be specified in accordance with this arrangement.

The request for a changeover itself is provided by a 2-point element (see image). The controller's output value, in other words the control value, should be used as the input value for the illustrated characteristic hysteresis curve. A request for changeover created by the hysteresis element must be present for at least the specified waiting time, so that the parameter range can be changed.

The **bDisableRange1** and **bDisableRange2** inputs makes it possible to prevent switching into one of the two ranges. It is therefore possible, for instance, to deactivate heating operation in summer and to deactivate cooling in winter. It would also be possible to make the change in the operation mode depend on the current control deviation. In summer, for instance, it might have to be 2°C too hot before switching into cooling operation. This can also be achieved by connecting the inputs appropriately.

Maximum and minimum limits are output in addition to providing the output of the parameter range, and these can be copied into the PID controller's parameter set. If the **FB_CTRL_PARAMETER_SWITCH** is in the **Heating** operation mode, the limits are set as follows:

$$fOutMinLimit = -1.0 \cdot stParams.fThreshold;$$
$$fOutMaxLimit = stParams.fOutMaxLimit;$$

In the **Cooling** mode, the limits are set as follows:

$$fOutMinLimit = stParams.fOutMaxLimit;$$
$$fOutMaxLimit = stParams.fThreshold;$$

VAR_INPUT

```
VAR_INPUT
    fManipulatedVariable    : FLOAT; (* fOut from the FB_CTRL_PID_SPLITRANGE *)
    nManSyncValue       : eCTRL_PARAMETER_RECORD_HEATING;
    bSync           : BOOL;
    eMode           : E_CTRL_MODE;
END_VAR
```

fManipulatedVariable : The input value of FB_Parameter_Switch. This should be the same as the controller's output value.

nManSyncValue : The input with which the function block can be set to one of the to parameter ranges.

bSync : A rising edge at this input sets the function block to the value nManSyncValue.

eMode : Input that specifies the block's operating mode [▶ 16].

VAR_OUTPUT

```
VAR_OUTPUT
    eParameterRecord        : E_CTRL_PARAMETER_RECORD;
    fOutMaxLimit        : FLOAT;
    fOutMinLimit        : FLOAT;
    eState          : E_CTRL_STATE;
    eErrorId            : E_CTRL_ERRORCODES;
    bError          : BOOL;
END_VAR
```

eParameterRecord : The output of the function block, identifying the parameter region.

fOutMaxLimit : The maximum output value of which the controller is limited. (This should be copied into the controller's parameter structure.)

fOutMinLimit : The minimum value of the output of which the controller is limited. (This should be copied into the controller's parameter structure.)

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_PARAMETER_SWITCH_PARAMS;
END_VAR
```

stParams : Parameter structure of the function block. This consists of the following elements:

```
TYPE
ST_CTRL_2POINT_PARAMS :
STRUCT
    tTaskCycleTime     : TIME; (* task cycle time [TIME]
*)
    tCtrlCycleTime     : TIME; (* controller cycle time [TIME]
*)
    fThreshold     : FLOAT;
    fOutMaxLimit       : FLOAT; (* max limit for heating
*)
    fOutMinLimit       : FLOAT; (* min limit for heating
*)
    tMinWaitTime       : TIME;
END_STRUCT
END_TYPE
```

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

fThreshold : Switching threshold, see image above.

fOutMaxLimit : Max limit, which is passed on to the controller.

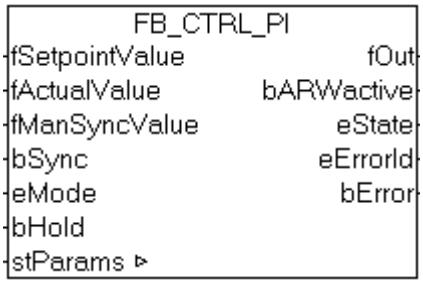
fOutMinLimit : Min limit, which is passed on to the controller.

tMinWaitTime : Waiting time (see description above)

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.4.7 FB_CTRL_PI

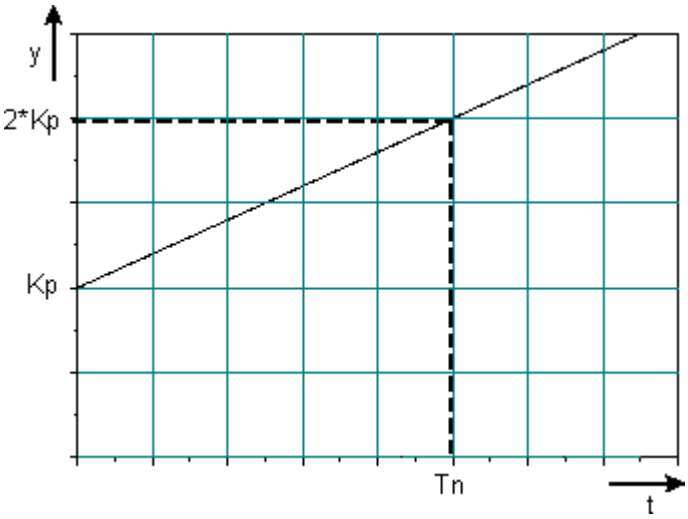


The function block provides a PI transfer element in the functional diagram.

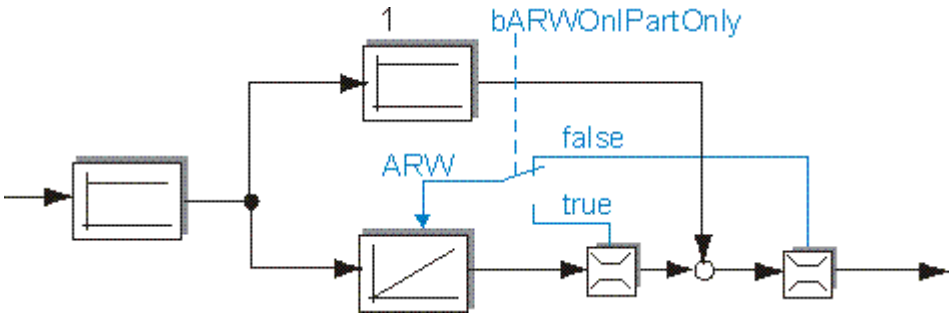
Behaviour of the output:

$$G(s) = K_p(1 + \frac{1}{T_n s})$$

Step response:



ARW:



VAR_INPUT

```
VAR_INPUT
    fSetpointValue  : FLOAT;
    fActualValue    : FLOAT;
    fManSyncValue   : FLOAT;
    bSync       : BOOL;
    eMode       : E_CTRL_MODE;
    bHold       : BOOL;
END_VAR
```

fSetpointValue : Set value of the controlled variable.

fActualValue : Actual value of the controlled variable.

fManSyncValue : Input with which the PI element can be set.

bSync : A rising edge at this input sets the PI element to the value fManSyncValue.

eMode : Input that specifies the block's operating mode [▶ 16].

bHold : A TRUE at this input will hold the internal state (and therefore also the output) constant at its current value, independently of the system deviation.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut        : FLOAT;
    bARWactive      : BOOL;
    eState      : E_CTRL_STATE;
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

fOut : Output of the PI element.

bARWactive : A TRUE at this output indicates that the PI element is being restricted.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_PI_PARAMS;
END_VAR
```

stParams : Parameter structure of the PI element. This consists of the following elements:

```
TYPE ST_CTRL_PI_PARAMS
:
STRUCT
    tCtrlCycleTime     : TIME     :=  T#0ms; (* controller
cycle time [TIME] *)
    tTaskCycleTime     : TIME     :=  T#0ms; (* task cycle time
[TIME] *)
    tTn        : TIME     :=  T#0ms; (* integral action
time Tn *)
    fKp        : FLOAT    :=  0;     (* proportional
gain *)
    fOutMaxLimit       : FLOAT    :=  1E38;  (* maximum output
limit *)
    fOutMinLimit       : FLOAT    := -1E38;  (* minimum output
limit *)
    bARWOnIPartOnly    : BOOL     := FALSE;  (* FALSE: Hold the
I-part if the entire control output reaches a limit. *)
                       (* TRUE: Hold the
I-part if only the I-part reaches a limit. *)
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

tTn : Integral action time

fKp : Controller amplification / transfer coefficient

fOutMaxLimit : Upper limit at which integration is halted and to which the output is limited (ARW measure). Reaching this limit is indicated by a TRUE at the bARWActive output.

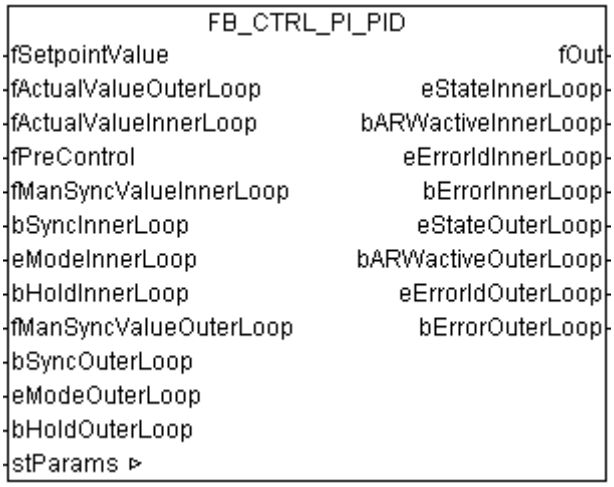
fOutMinLimit : Lower limit at which integration is halted and to which the output is limited (ARW measure). Reaching this limit is indicated by a TRUE at the bARWActive output.

bARWOnIPartOnly: If this parameter is FALSE (the default setting), the integration of the I-component is halted if the complete controller output reaches the upper or lower limit. If it is TRUE, the integration is halted if the I-component (the output of the integrator) reaches some limit. (Cf. functional diagram)

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.4.8 FB_CTRL_PI_PID



The function block provides a cascaded PI-PID controller in the functional diagram. Internally, this block uses the FB_CTRL_PI, FB_CTRL_LIMITER and FB_CTRL_PID transfer elements.

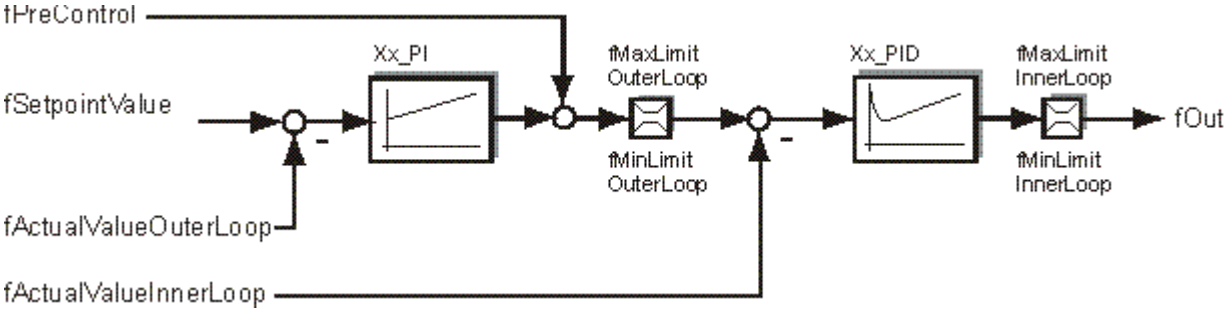
Transfer function of the PI element:

$$G(s) = K_p(1 + \frac{1}{T_n s})$$

Transfer function of the PID element:

$$G(s) = K_p(1 + \frac{1}{T_n s} + \frac{T_v s}{1 + T_d s})$$

Functional diagram for the cascaded transfer element:



VAR_INPUT

```
VAR_INPUT
    fSetpointValue      : FLOAT;        (* setpoint value *)
    fActualValueOuterLoop   : FLOAT;        (* actual value from the process to the PI-controller *)
    fActualValueInnerLoop   : FLOAT;        (* actual value from the process to the PID-
controller *)
    fPreControl         : FLOAT;        (* pre control value *)

    fManSyncValueInnerLoop  : FLOAT;        (* input value for the manual mode or the sync request *
)
    bSyncInnerLoop      : BOOL;         (* rising edge set the output to the fManSyncValue *)
    eModeInnerLoop      : E_CTRL_MODE;  (* sets the mode of the function block *)
    bHoldInnerLoop      : BOOL;         (* hold the internal integrator *)

    fManSyncValueOuterLoop  : FLOAT;        (* input value for the manual mode or the sync request *
```

```
)
    bSyncOuterLoop      : BOOL;         (* rising edge set the output to the fManSyncValue *)
    eModeOuterLoop      : E_CTRL_MODE;  (* sets the mode of the function block *)
    bHoldOuterLoop      : BOOL;         (* hold the internal integrator *)
END_VAR
```

fSetpointValue : Setpoint of the controlled variable.

fActualValueOuterLoop : The actual value of the controlled variable that is fed back to the PI controller of the outer control loop.

fActualValueInnerLoop : The actual value of the controlled variable that is fed back to the PID controller of the inner control loop.

fPreControl : Pre-control that is connected behind the PI controller.

fManSyncValueInnerLoop : Input, to whose value it is possible to set the internal state of the PID element (inner control loop).

bSyncInnerLoop : A rising edge at this input sets the PID element (inner control loop) to the value fManSyncValueInnerLoop.

eModeInnerLoop : Input that specifies the operation mode [▶ 16] of the PID element (inner control loop).

bHoldInnerLoop : A TRUE at this input holds the internal state of the PID element (inner control loop) constant at the current value.

fManSyncValueOuterLoop : Input, to whose value it is possible to set the internal state of the PI element (outer control loop).

bSyncOuterLoop : A rising edge at this input sets the PI element (the outer control loop) to the value fManSyncValueOuterLoop.

eModeOuterLoop : Input that specifies the operation mode [▶ 16] of the PI element (the outer control loop).

bHoldOuterLoop : A TRUE at this input holds the internal state of the PI element (the outer control loop) constant at the current value.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut            : FLOAT;            (* manipulated value to the process *)

    eStateInnerLoop         : E_CTRL_STATE;
    bARWactiveInnerLoop     : BOOL;
    eErrorIdInnerLoop           : E_CTRL_ERRORCODES;    (* error code *)
    bErrorInnerLoop         : BOOL;             (* TRUE if an error situation exists *)

    eStateOuterLoop         : E_CTRL_STATE;
    bARWactiveOuterLoop     : BOOL;
    eErrorIdOuterLoop           : E_CTRL_ERRORCODES;    (* error code *)
    bErrorOuterLoop         : BOOL;             (* TRUE if an error situation exists *)
END_VAR
```

fOut : Output of the PI-PID element.

eStateInnerLoop : State of the internal PID element (inner control loop).

bARWactiveInnerLoop : A TRUE at this output indicates that the PID element (inner control loop) is being restricted.

eErrorIdInnerLoop : Returns the error number [▶ 16] of the PID element (inner control loop) when the bError output is set.

bErrorInnerLoop : Becomes TRUE as soon as an error occurs in the PID element (inner control loop).
eStateOuterLoop : State of the internal PI element (outer control loop).

bARWactiveOuterLoop : A TRUE at this output indicates that the PID element (outer control loop) is being restricted.

eErrorIdOuterLoop : Returns the error number [▶ 16] of the PI element (outer control loop) when the bError output is set.

bErrorOuterLoop : Is set to TRUE as soon as an error occurs in the PI element (outer control loop).

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_PI_PID_PARAMS;
END_VAR
```

stParams : Parameter structure of the PI-PID element. This consists of the following elements:

```
TYPE
ST_CTRL_PI_PID_PARAMS :
STRUCT
    tCtrlCycleTime    : TIME  := T#0ms; (* controller cycle
time [TIME] *)
    tTaskCycleTime    : TIME  := T#0ms; (* task cycle time
[TIME] *)
    fKp_OuterLoop     : FLOAT := 0; (* proportional gain
*)
    tTn_OuterLoop     : TIME  := T#0s; (* Tn *)
    fMaxLimit_OuterLoop   : FLOAT := 1E38;
    fMinLimit_OuterLoop   : FLOAT := -1E38;
    fKp_InnerLoop     : FLOAT := 0; (* proportional gain
*)
    tTn_InnerLoop     : TIME  := T#0ms; (* Tn *)
    tTv_InnerLoop     : TIME  := T#0ms; (* Tv *)
    tTd_InnerLoop     : TIME  := T#0ms; (* Td *)
    fMaxLimit_InnerLoop   : FLOAT := 1E38;
    fMinLimit_InnerLoop   : FLOAT := -1E38;
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

fKp_OuterLoop : Controller amplification / controller coefficient of the PI element (outer control loop).

tTn_OuterLoop : Integral action time of the PI element (outer control loop). The I-component is deactivated if this is parameterized as T#0s.

fMaxLimit_OuterLoop : Upper limit at which integration of the PID element is halted and to which the output is limited (ARW measure). Reaching this limit is indicated by a TRUE at the bARWactiveOuterLoop output.

fMinLimit_OuterLoop : Lower limit at which integration of the PID element is halted and to which the output is limited (ARW measure). Reaching this limit is indicated by a TRUE at the bARWactiveOuterLoop output.

fKp_InnerLoop : Controller amplification / controller coefficient of the PID element (inner control loop).

tTn_InnerLoop : Integral action time of the PID element (inner control loop). The I-component is deactivated if this is parameterized as T#0s.

tTv_InnerLoop : Derivative action time of the PID element (inner control loop). The D-component is deactivated if this is parameterized as T#0s.

tTd_InnerLoop : Damping time of the PID element (inner control loop).

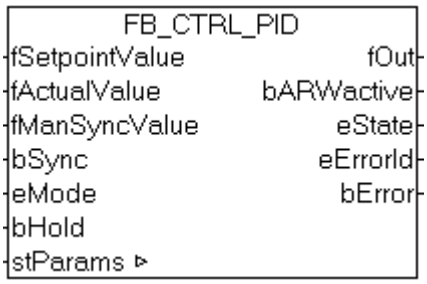
fMaxLimit_InnerLoop: Upper limit at which integration of the PID element is halted and to which the output is limited (ARW measure). Reaching this limit is indicated by a TRUE at the bARWactiveInnerLoop output.

fMinLimit_InnerLoop : Lower limit at which integration of the PID element is halted and to which the output is limited (ARW measure). Reaching this limit is indicated by a TRUE at the bARWactiveInnerLoop output.

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.4.9 FB_CTRL_PID



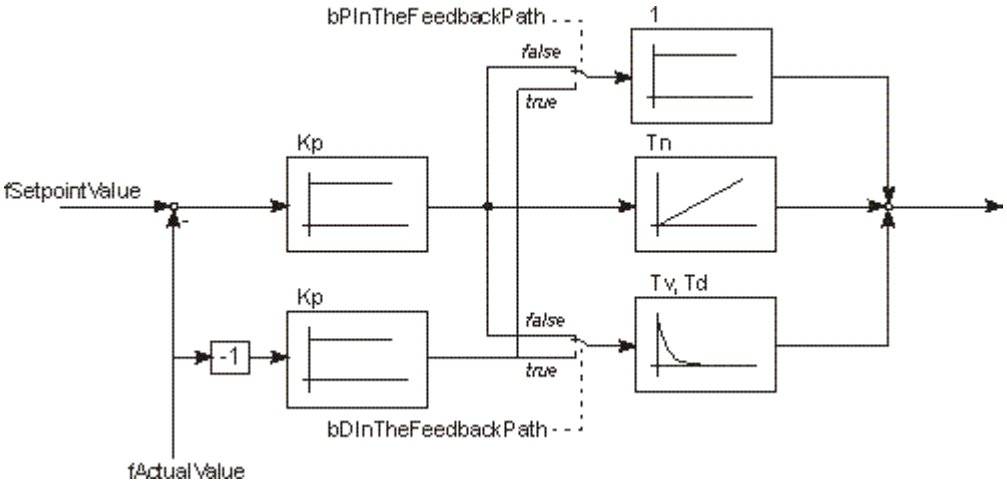
The function block provides a PID transfer element in the functional diagram.

Transfer function:

The following transfer function can be declared for this block, if the boolean inputs **bPInTheFeedbackPath** and **bDInTheFeedbackPath** are set to FALSE, otherwise this transfer function only describes a part of the blocks behaviour:

$$G_{PID}(s) = K_p\left(1 + \frac{1}{T_n s} + \frac{T_v s}{1 + T_d s}\right)$$

Functional diagram:



The standard functional diagram of a PID controller in additive form has been expanded by the two active boolean inputs **bPInTheFeedbackPath** and **bDInTheFeedbackPath** (which act as "switches"), so that a modified functional diagram can be activated.

Control background: due to the differential component of the control algorithm, large control values are generated at setpoint step-changes, which cause a strain on the control elements and may cause the control system to oscillate. A control algorithm with a differential component that is only applied to the controlled variable (bDInTheFeedbackPath := TRUE) can avoid this problem.

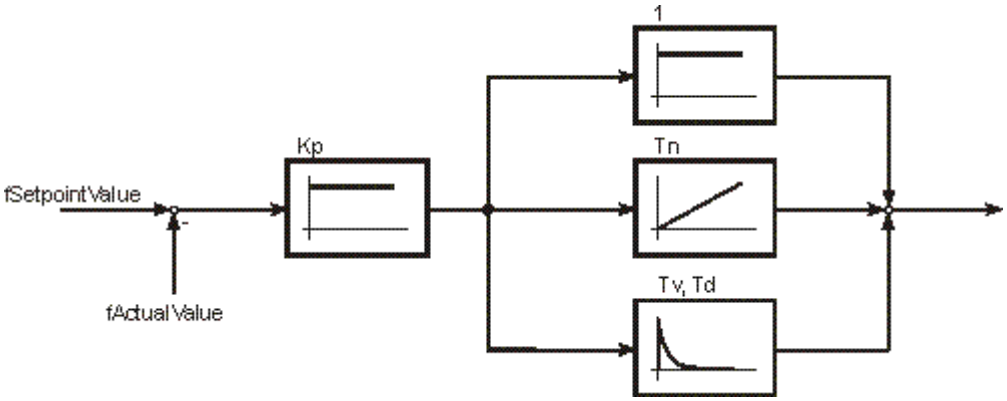
The bPInTheFeedbackPath and bDInTheFeedbackPath inputs permit the closed control loop to implement the following transfer functions:

bPInTheFeedbackPath	bDInTheFeedbackPath	$G(s)$
false	false	$G(s) = \frac{G_{PID}(s) \cdot G_S(s)}{1+G_{PID}(s) \cdot G_S(s)}$
true	false	$G(s) = \frac{G_{DI}(s) \cdot G_S(s)}{1+G_{PID}(s) \cdot G_S(s)}$
false	true	$G(s) = \frac{G_{PI}(s) \cdot G_S(s)}{1+G_{PID}(s) \cdot G_S(s)}$
true	true	$G(s) = \frac{G_I \cdot G_S(s)}{1+G_{PID}(s) \cdot G_S(s)}$

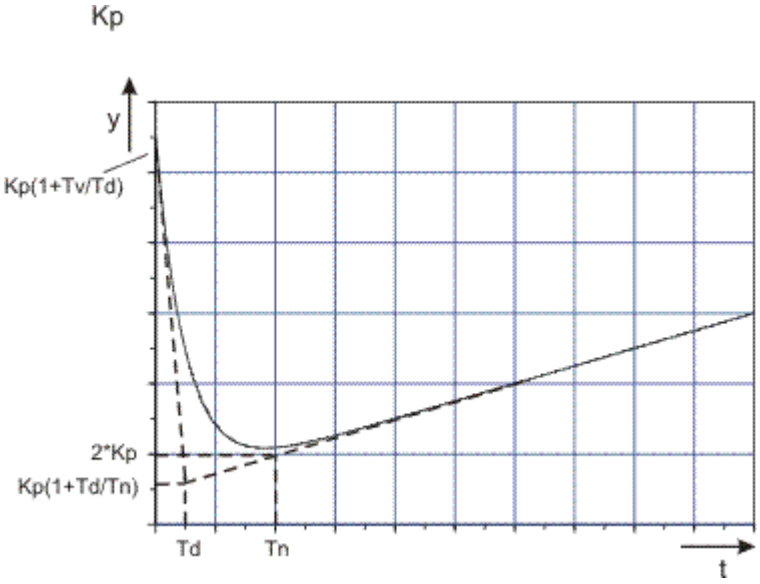
with:

$$G_{DI}(s) = K_p(\frac{1}{T_n s} + \frac{T_v s}{1+T_d s})$$
$$G_{PI}(s) = K_p(1 + \frac{1}{T_n s})$$
$$G_I(s) = K_p(\frac{1}{T_n s})$$

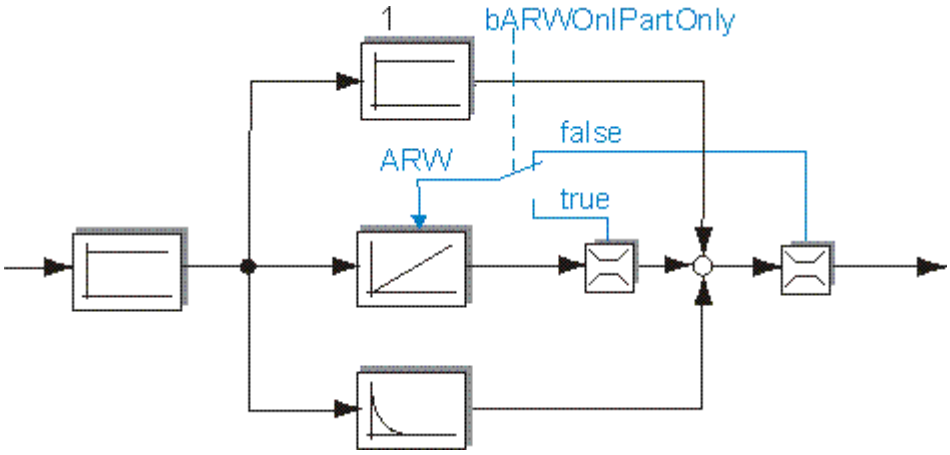
! The default setting for the two inputs **bPInTheFeedbackPath** and **bDInTheFeedbackPath** is **FALSE**. The PID controller then acts as a standard PID controller in additive form.



Step response:



ARW:



VAR_INPUT

```
VAR_INPUT
    fSetpointValue      : FLOAT;
    fActualValue        : FLOAT;
    fManSyncValue       : FLOAT;
    bSync           : BOOL;
    eMode           : E_CTRL_MODE;
    bHold           : BOOL;
END_VAR
```

fSetpointValue : Set value of the controlled variable.

fActualValue : Actual value of the controlled variable.

fManSyncValue : Input, to whose value it is possible to set the internal state of the PID element.

bSync : A rising edge at this input sets the PID element to the value fManSyncValue.

eMode : Input that specifies the block's operating mode [▶ 16].

bHold : A TRUE at this input will hold the internal state (and therefore also the output) constant at its current value, independently of the system deviation.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut        : FLOAT;
    bARWactive      : BOOL;
```

```
    eState      : E_CTRL_STATE;
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

fOut : Output of the PID element.

bARWactive : A TRUE at this output indicates that the PID element is being restricted.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_PID_PARAMS;
END_VAR
```

stParams : Parameter structure of the PID element. This consists of the following elements:

```
TYPE
ST_CTRL_PID_PARAMS :
STRUCT
    tCtrlCycleTime      : TIME      := T#0ms;      (* controller
cycle time [TIME] *)
    tTaskCycleTime      : TIME      := T#0ms;      (* task cycle
time [TIME] *)
    fKp         : FLOAT     := 0;      (* proportional
gain *)
    tTn         : TIME      := T#0ms;      (* Tn *)
    tTv         : TIME      := T#0ms;      (* Tv *)
    tTd         : TIME      := T#0ms;      (* Td *)
    fOutMaxLimit        : FLOAT     :=  1E38;      (* maximum
output limit *)
    fOutMinLimit        : FLOAT     := -1E38;      (* minimum
output limit *)
    bPInTheFeedbackPath : BOOL;
    bDInTheFeedbackPath : BOOL;
    bARWOnIPartOnly     : BOOL;
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime: Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

fKp : Controller amplification / controller coefficient

tTn : Integral action time. The I-component is deactivated if this is parameterized as T#0s.

tTv : Derivative action time. The D-component is deactivated if this is parameterized as T#0s.

tTd : Damping time

fOutMaxLimit : Upper limit at which integration is halted and to which the output is limited (ARW measure). Reaching this limit is indicated by a TRUE at the bARWActive output.

fOutMinLimit : Lower limit at which integration is halted and to which the output is limited (ARW measure). Reaching this limit is indicated by a TRUE at the bARWActive output.

bPInTheFeedbackPath : Input value of the internal P element can be selected with this input (see functional diagram). Default setting: FALSE

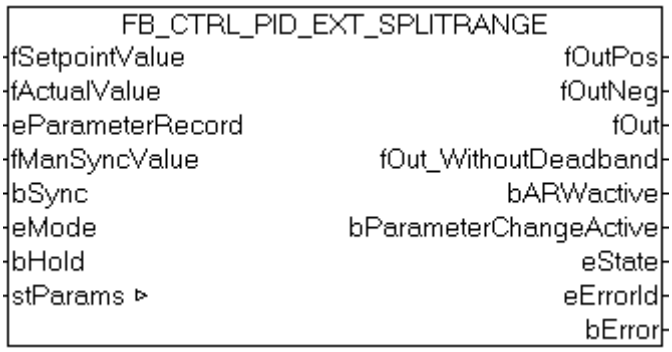
bDInTheFeedbackPath : Input value of the internal D element can be selected with this input (see functional diagram). Default setting: FALSE

bARWOnIPartOnly: If this parameter is FALSE (the default setting), the integration of the I-component is halted if the complete controller output reaches the upper or lower limit. If it is TRUE, the integration is halted if the I-component (the output of the integrator) reaches some limit. (Cf. functional diagram)

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.4.10 FB_CTRL_PID_EXT_SPLITRANGE



The function block provides an extended PID transfer element in the functional diagram. With this controller it is possible to switch between two different parameter records while the regulation is active. The functionalities of the inner and outer windows and of the input and output dead bands are available in addition.

Description:

This function block is an extension of FB_CTRL_PID_EXT, which means that the controller can be used to control systems with two controlled devices for which the transfer behaviours are different. A system with one actuator for heating and another actuator for cooling would be a typical application. To optimise the regulation of such an arrangement, it is possible to switch between two PID parameter records. Switching between the parameter records is implemented in such a way that the control value remains continuous even as the parameter records are changed.

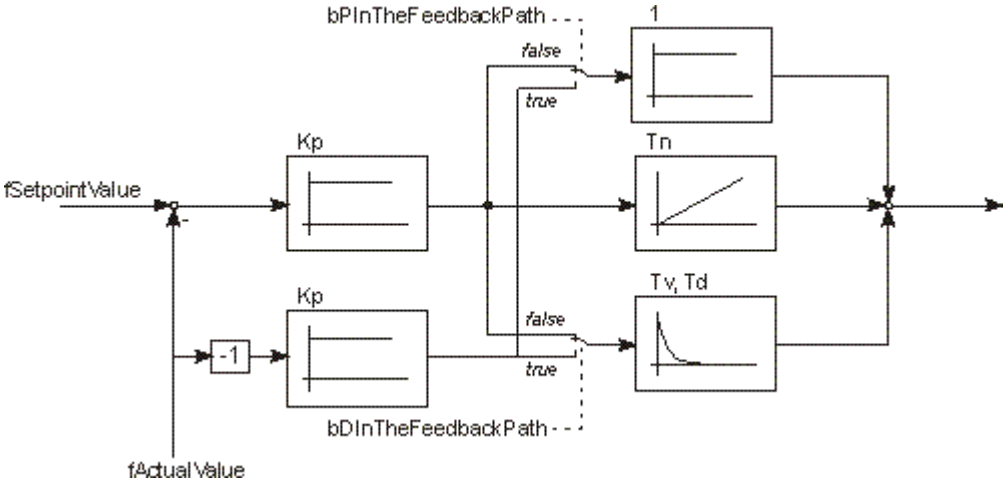
The switching algorithm calculates a linear, time-dependent transition between the two parameter records. The nParameterChangeCycleTicks parameter can be used to specify the number of task cycles over which the continuous change between the two parameter records takes place.

Transfer function:

The following transfer function can be declared for this block, if the boolean inputs **bPInTheFeedbackPath** and **bDInTheFeedbackPath** are set to FALSE, otherwise this transfer function only describes a part of the blocks behaviour:

$$G_{PID}(s) = K_p \left(1 + \frac{1}{T_n s} + \frac{T_v s}{1 + T_d s}\right)$$

Functional diagram:



The standard functional diagram of a PID controller in additive form has been expanded by the two active boolean inputs **bPInTheFeedbackPath** and **bDInTheFeedbackPath** (which act as "switches"), so that a modified functional diagram can be activated.

Control background: due to the differential component of the control algorithm, large control values are generated at setpoint step-changes, which cause a strain on the control elements and may cause the control system to oscillate. A control algorithm with a differential component that is only applied to the controlled variable (bDInTheFeedbackPath := TRUE) can avoid this problem.

The bPInTheFeedbackPath and bDInTheFeedbackPath inputs permit the closed control loop to implement the following transfer functions:

bPInTheFeedbackPath	bDInTheFeedbackPath	$G(s)$
false	false	$G(s) = \frac{G_{PID}(s) \cdot G_S(s)}{1+G_{PID}(s) \cdot G_S(s)}$
true	false	$G(s) = \frac{G_{DI}(s) \cdot G_S(s)}{1+G_{PID}(s) \cdot G_S(s)}$
false	true	$G(s) = \frac{G_{PI}(s) \cdot G_S(s)}{1+G_{PID}(s) \cdot G_S(s)}$
true	true	$G(s) = \frac{G_I \cdot G_S(s)}{1+G_{PID}(s) \cdot G_S(s)}$

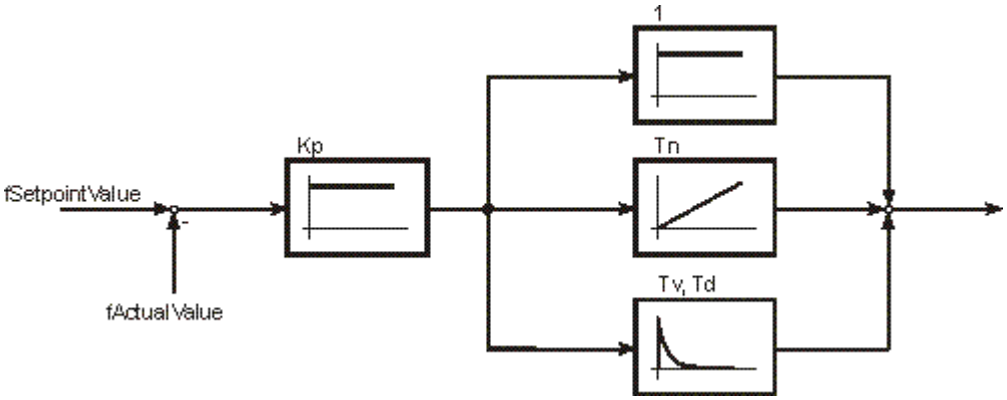
with:

$$G_{DI}(s) = K_p(\frac{1}{T_n s} + \frac{T_v s}{1 + T_d s})$$

$$G_{PI}(s) = K_p(1 + \frac{1}{T_n s})$$

$$G_I(s) = K_p(\frac{1}{T_n s})$$

! The default setting for the two inputs **bPInTheFeedbackPath** and **bDInTheFeedbackPath** is **FALSE**. The PID controller then acts as a standard PID controller in additive form.



Additional functions:

Switching off the I component in the Outer Window

Integration of the system deviation is halted if the system deviation is greater than the fOuterWindow parameter. In this way it is possible to prevent an extremely large I component from developing if the system deviation is large, since this could lead to a marked overshoot. If it is not wanted, the function can be disabled by setting fOuterWindow := 0.

Linear reduction of the I component in the Inner Window

With this function it is possible to drive the I component linearly down to zero in the range specified by the fInnerWindow parameter. If it is not wanted, the function can be disabled by setting fInnerWindow := 0.

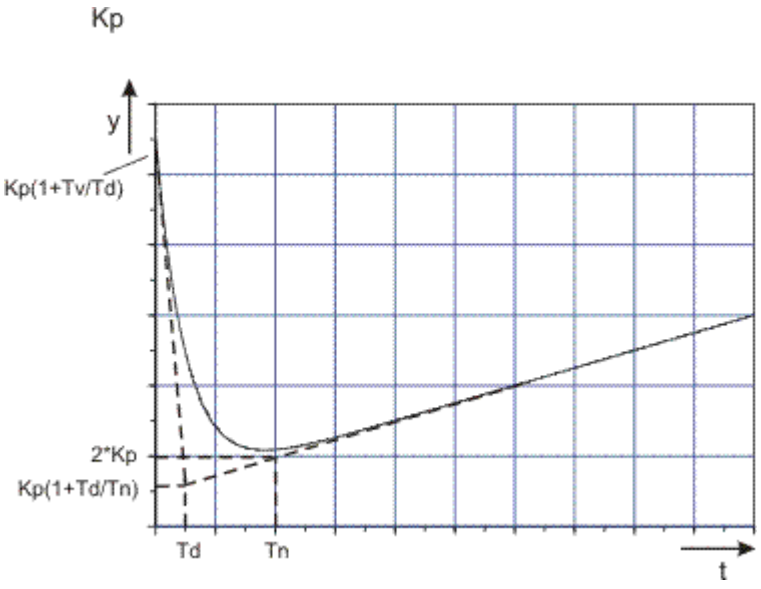
Output dead band

If the parameter fDeadBandOutput is set > 0, the output is set to zero when it is within the range of [-fDeadBandOutput ... fDeadBandOutput].

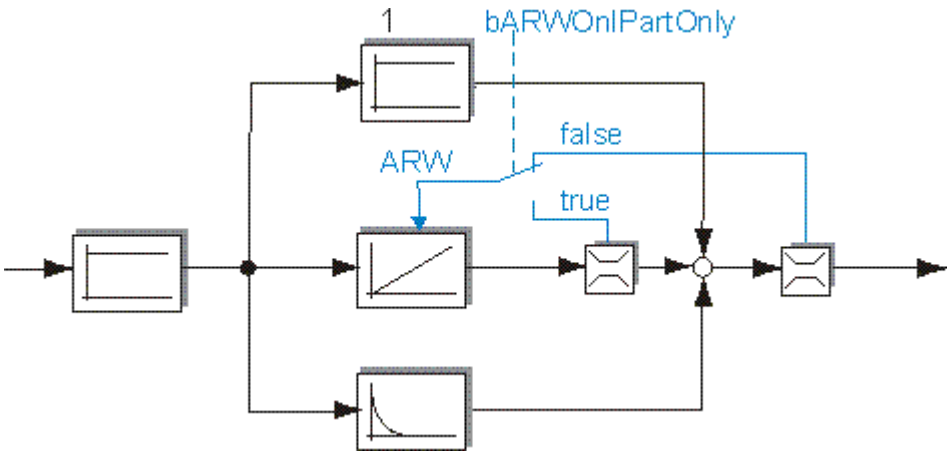
Input dead band

If the parameter fDeadBandInput is set > 0 then the output is held constant for as long as the system deviation remains within the range of [-fDeadBandInput ... fDeadBandInput].

Step response:



ARW:



VAR_INPUT

```
VAR_INPUT
    fSetpointValue      : FLOAT;
    fActualValue        : FLOAT;
    eParameterRecord    : E_CTRL_PARAMETER_RECORD;
    fManSyncValue       : FLOAT;
    bSync               : BOOL;
    eMode               : E_CTRL_MODE;
    bHold               : BOOL;
END_VAR
```

fSetpointValue : Setpoint of the controlled variable.

fActualValue : Actual value of the controlled variable.

eParameterRecord : Index of the active parameter set

fManSyncValue : Input with which the PI element can be set.

bSync : A rising edge at this input sets the PI element to the value fManSyncValue.

eMode : Input that specifies the operation mode [▶ 16] of the function block.

bHold : A TRUE at this input will hold the internal state (and therefore also the output) constant at its current value, independently of the control deviation.

VAR_OUTPUT

```
VAR_OUTPUT
    fOutPos             : FLOAT;
    fOutNeg             : FLOAT;
    fOut            : FLOAT;

    bARWActive          : BOOL  := FALSE;       (* ARW active ? [TRUE/FALSE] -> freeze I-part *)
    bParameterChangeActive  : BOOL;

    bError          : BOOL;             (* error flag *)
    eErrorId            : E_CTRL_ERRORCODES;    (* error code *)

END_VAR
```

fOutPos : Output of the PID element when the control value is positive. A zero is output otherwise.

fOutNeg : Output of the PID element when the control value is negative. A zero is output otherwise.

fOut : Output of the PID element.

bARWactive : A TRUE at this output indicates that the PID element is being restricted.

bParameterChangeActive : A TRUE at this output indicates that the change from one parameter record to the other is in progress.

fCtrlDerivation : A TRUE at this output indicates that the PID element is being restricted.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_PID_SPLITRANGE_PARAMS;
END_VAR
```

stParams : Parameter structure of the PID element. This consists of the following elements:

```
TYPE
ST_CTRL_PID_SPLITRANGE_PARAMS :
STRUCT
    tCtrlCycleTime          : TIME     := T#0ms; (*
controller cycle time [TIME] *)
    tTaskCycleTime          : TIME     := T#0ms; (* task
cycle time [TIME] *)
    fKp_heating         : FLOAT    := 0;      (*
proportional gain *)
    tTn_heating         : TIME     := T#0ms; (* Tn
*)
    tTv_heating         : TIME     := T#0ms; (* Tv
*)
    tTd_heating         : TIME     := T#0ms; (* Td
*)
    fKp_cooling         : FLOAT    := 0;      (*
proportional gain *)
    tTn_cooling         : TIME     := T#0ms; (* Tn
*)
    tTv_cooling         : TIME     := T#0ms; (* Tv
*)
    tTd_cooling         : TIME     := T#0ms; (* Td
*)
    nParameterChangeCycleTicks : INT;
    fDeadBandInput          : REAL     := 0.0; (* ctrl
deviation dead band/neutral zone for const output *)
    fDeadBandOutput         : REAL     := 0.0; (* ctrl
output dead band/neutral zone for zero output *)
    fInnerWindow            : REAL     := 0.0; (* inner
window for reduced I-part (dE-window) *)
    fOuterWindow            : REAL     := 0.0; (* outer
window for disabling I-part (dE-window) *)
    fOutMaxLimit            : FLOAT    :=  1E38; (* maximum
output limit *)
    fOutMinLimit            : FLOAT    := -1E38; (* minimum
output limit *)
    bPInTheFeedbackPath     : BOOL;
    bDInTheFeedbackPath     : BOOL;
    bARWOnIPartOnly         : BOOL;
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

Area eCTRL_PARAMETER_RECORD_**HEATING**:

fKp_heating : Controller amplification / controller coefficient

tTn_heating : Integral action time. The I-component is deactivated if this is parameterized as T#0s.

tTv_heating : Derivative action time. The D-component is deactivated if this is parameterized as T#0s.

tTd_heating : Damping time

Area eCTRL_PARAMETER_RECORD_**COOLING:**

fKp_cooling : Controller amplification / controller coefficient

tTn_cooling : Integral action time. The I-component is deactivated if this is parameterized as T#0s.

tTv_cooling : Derivative action time. The D-component is deactivated if this is parameterized as T#0s.

tTd_cooling : Damping time

nParameterChangeCycleTicks The number of task cycles over which the change from one parameter set to the other takes place.

fDeadBandInput : see above description

fDeadBandOutput: see above description

fInnerWindow : see above description

fOuterWindow : see above description

fOutMaxLimit : Upper limit at which integration is halted and to which the output is limited (ARW measure). Reaching this limit is indicated by a TRUE at the bARWActive output.

fOutMinLimit : Lower limit at which integration is halted and to which the output is limited (ARW measure). Reaching this limit is indicated by a TRUE at the bARWActive output.

bPInTheFeedbackPath : Input value of the P element can be selected with this input (see functional diagram).

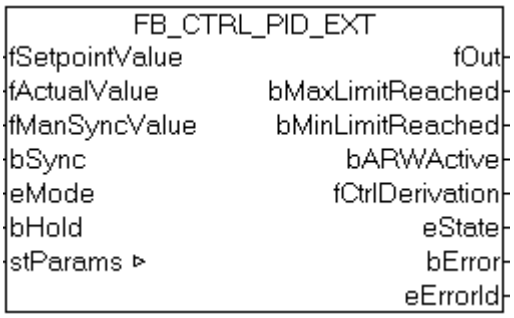
bDInTheFeedbackPath : Input value of the D element can be selected with this input (see functional diagram).

bARWOnIPartOnly: If this parameter is FALSE (default setting), integration of the I component is halted if the complete controller output reaches the upper or lower limit.
If TRUE, the integration is halted if the I component (the output of the integrator) reaches a limit. (Cf. functional diagram)

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.4.11 FB_CTRL_PID_EXT



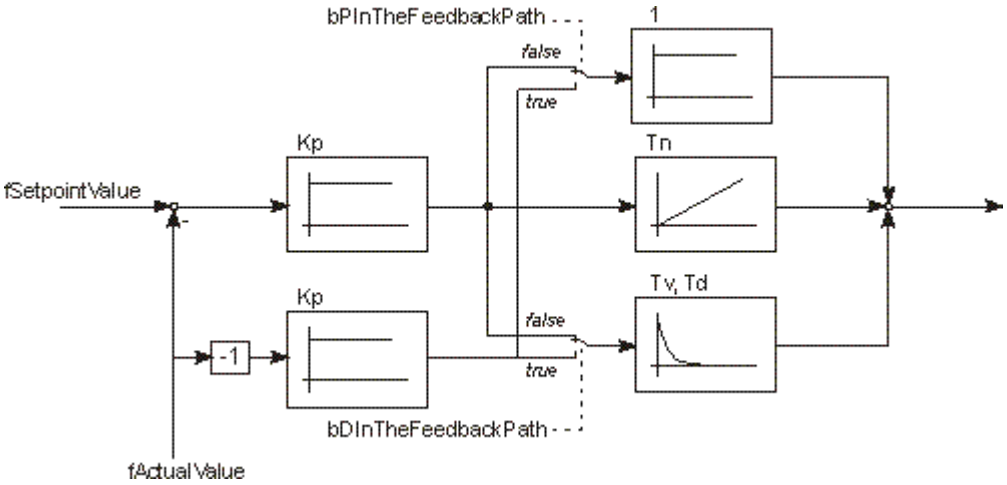
The function block provides an extended PID element in the functional diagram.

Transfer function:

The following transfer function can be declared for this block, if the boolean inputs **bPInTheFeedbackPath** and **bDInTheFeedbackPath** are set to FALSE, otherwise this transfer function only describes a part of the blocks behaviour:

$$G_{PID}(s) = K_p \left(1 + \frac{1}{T_n s} + \frac{T_v s}{1 + T_d s}\right)$$

Functional diagram:



The standard functional diagram of a PID controller in additive form has been expanded by the two active boolean inputs **bPInTheFeedbackPath** and **bDInTheFeedbackPath** (which act as "switches"), so that a modified functional diagram can be activated.

Control background: due to the differential component of the control algorithm, large control values are generated at setpoint step-changes, which cause a strain on the control elements and may cause the control system to oscillate. A control algorithm with a differential component that is only applied to the controlled variable (bDInTheFeedbackPath := TRUE) can avoid this problem.

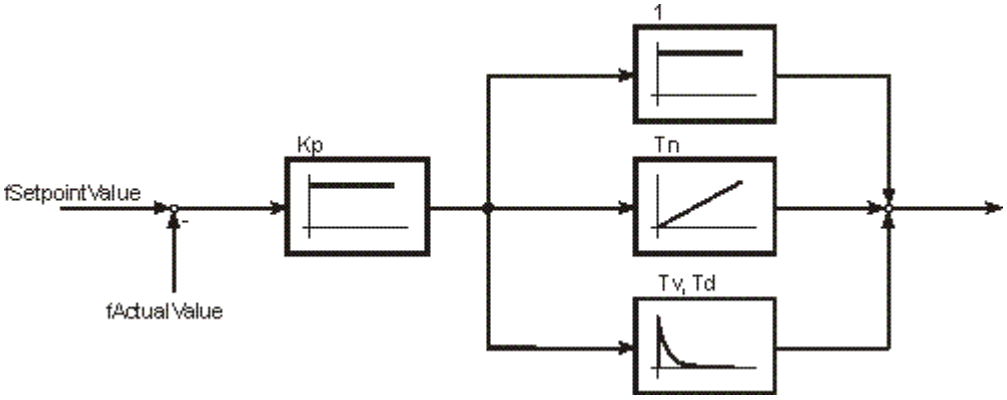
The bPInTheFeedbackPath and bDInTheFeedbackPath inputs permit the closed control loop to implement the following transfer functions:

bPInTheFeedbackPath	bDInTheFeedbackPath	$G(s)$
false	false	$G(s) = \frac{G_{PID}(s) \cdot G_S(s)}{1 + G_{PID}(s) \cdot G_S(s)}$
true	false	$G(s) = \frac{G_{DI}(s) \cdot G_S(s)}{1 + G_{PID}(s) \cdot G_S(s)}$
false	true	$G(s) = \frac{G_{PI}(s) \cdot G_S(s)}{1 + G_{PID}(s) \cdot G_S(s)}$
true	true	$G(s) = \frac{G_I \cdot G_S(s)}{1 + G_{PID}(s) \cdot G_S(s)}$

with:

$$G_{DI}(s) = K_p(\frac{1}{T_n s} + \frac{T_v s}{1 + T_d s})$$
$$G_{PI}(s) = K_p(1 + \frac{1}{T_n s})$$
$$G_I(s) = K_p(\frac{1}{T_n s})$$

! The default setting for the two inputs **bPInTheFeedbackPath** and **bDInTheFeedbackPath** is **FALSE**. The PID controller then acts as a standard PID controller in additive form.



Additional functions:

Switching off the I component in the Outer Window

Integration of the system deviation is halted if the system deviation is greater than the fOuterWindow parameter. In this way it is possible to prevent an extremely large I component from developing if the system deviation is large, since this could lead to a marked overshoot. If it is not wanted, the function can be disabled by setting fOuterWindow := 0.

Linear reduction of the I component in the Inner Window

With this function it is possible to drive the I component linearly down to zero in the range specified by the fInnerWindow parameter. If it is not wanted, the function can be disabled by setting fInnerWindow := 0.

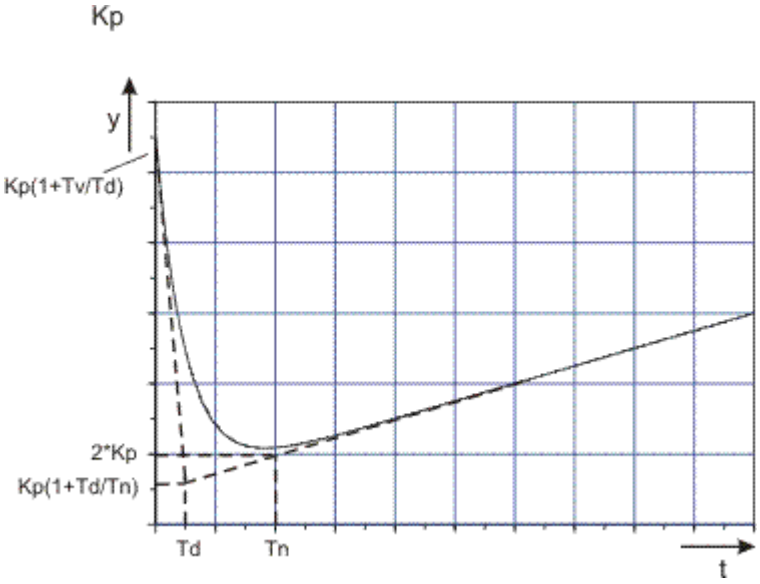
Output dead band

If the parameter fDeadBandOutput is set > 0, the output is set to zero when it is within the range of [-fDeadBandOutput ... fDeadBandOutput].

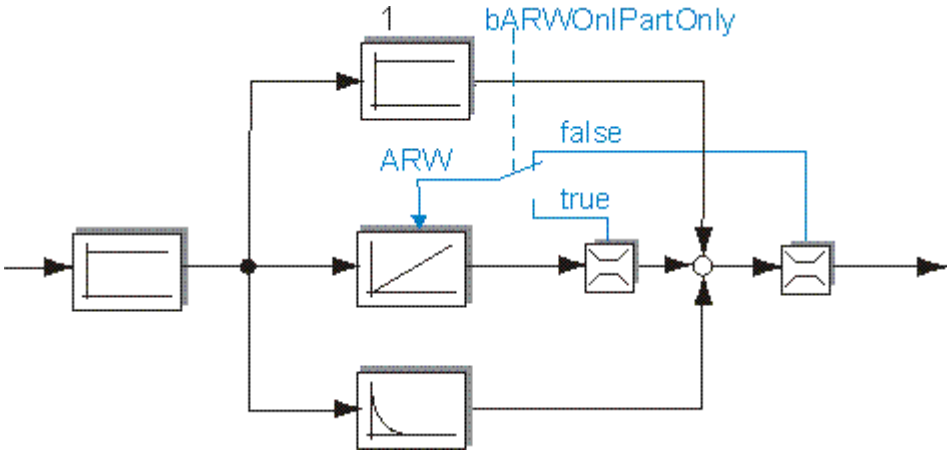
Input dead band

If the parameter fDeadBandInput is set > 0 then the output is held constant for as long as the system deviation remains within the range of [-fDeadBandInput ... fDeadBandInput].

Step response:



ARW:



VAR_INPUT

```
VAR_INPUT
    fSetpointValue  : FLOAT;
    fActualValue    : FLOAT;
    fManSyncValue   : FLOAT;
    bSync           : BOOL;
    eMode           : E_CTRL_MODE;
    bHold           : BOOL;
END_VAR
```

fSetpointValue : Setpoint of the controlled variable.

fActualValue : Actual value of the controlled variable.

fManSyncValue : Input with which the PID element can be set.

bSync : A rising edge at this input sets the PID element to the value fManSyncValue.

eMode : Input that specifies the operation mode [▶ 16] of the function block.

bHold : A TRUE at this input will hold the internal state (and therefore also the output) constant at its current value, independently of the control deviation.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut        : FLOAT;
```

```
    bMaxLimitReached    : BOOL  := FALSE;       (* minimum limitation active ? [TRUE/FALSE] -
> ARW *)
    bMinLimitReached    : BOOL  := FALSE;       (* maximum limitation active ? [TRUE/FALSE] -
> ARW *)
    bARWActive      : BOOL  := FALSE;       (* ARW active ? [TRUE/FALSE] -> freeze I-part *)

    fCtrlDerivation     : FLOAT;            (* controller command derivation dy/dt *)

    eState      : E_CTRL_STATE;
    bError      : BOOL;             (* error flag *)
    eErrorId        : E_CTRL_ERRORCODES;    (* error code *)

END_VAR
```

fOut : Output of the PID-element.

bMaxLimitReached : The output is TRUE when the block is at its upper limit.

bMinLimitReached : The output is TRUE when the block is at its lower limit.

bARWactive : A TRUE at this output indicates that the PID element is being restricted.

fCtrlDerivation : A TRUE at this output indicates that the PID element is being restricted.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_PID_EXT_PARAMS;
END_VAR
```

stParams : Parameter structure of the PID element. This consists of the following elements:

```
TYPE
ST_CTRL_PID_EXT_PARAMS :
STRUCT
    tCtrlCycleTime      : TIME      := T#0ms;      (* controller
cycle time [TIME] *)
    tTaskCycleTime      : TIME      := T#0ms;      (* task cycle
time [TIME] *)
    fKp         : FLOAT     := 0;      (* proportional
gain *)
    tTn         : TIME      := T#0ms;      (* Tn *)
    tTv         : TIME      := T#0ms;      (* Tv *)
    tTd         : TIME      := T#0ms;      (* Td *)
    fDeadBandInput      : REAL      := 0.0;        (* ctrl
deviation dead band/neutral zone for const output *)
    fDeadBandOutput     : REAL      := 0.0;        (* ctrl output
dead band/neutral zone for zero output *)
    fInnerWindow        : REAL      := 0.0;        (* inner window
for reduced I-part (dE-window) *)
    fOuterWindow        : REAL      := 0.0;        (* outer window
for disabling I-part (dE-window) *)
    fOutMaxLimit        : FLOAT     :=  1E38;      (* maximum
output limit *)
    fOutMinLimit        : FLOAT     := -1E38;      (* minimum
output limit *)
    bPInTheFeedbackPath : BOOL;
    bDInTheFeedbackPath : BOOL;
    bARWOnIPartOnly     : BOOL;
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

fKp : Controller amplification / controller coefficient

tTn : Integral action time. The I-component is deactivated if this is parameterized as T#0s.

tTv : Derivative action time. The D-component is deactivated if this is parameterized as T#0s.

tTd : Damping time

fDeadBandInput : see above description

fDeadBandOutput: see above description

fInnerWindow : see above description

fOuterWindow : see above description

fOutMaxLimit : Upper limit at which integration is halted and to which the output is limited (ARW measure). Reaching this limit is indicated by a TRUE at the bARWActive output.

fOutMinLimit : Lower limit at which integration is halted and to which the output is limited (ARW measure). Reaching this limit is indicated by a TRUE at the bARWActive output.

bPInTheFeedbackPath : Input value of the P element can be selected with this input (see functional diagram).

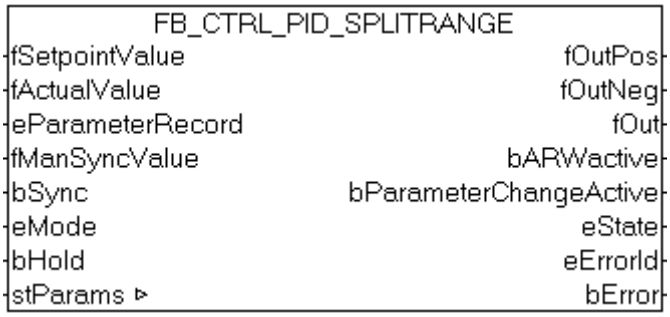
bDInTheFeedbackPath : Input value of the D element can be selected with this input (see functional diagram).

bARWOnIPartOnly: If this parameter is FALSE (default setting), integration of the I component is halted if the complete controller output reaches the upper or lower limit.
If TRUE, the integration is halted if the I component (the output of the integrator) reaches a limit. (Cf. functional diagram)

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.4.12 FB_CTRL_PID_SPLITRANGE



The function block provides an extended PID transfer element in the functional diagram. With this controller it is possible to switch between two different parameter records while the regulation is active.

Description:

This function block is an extension of FB_CTRL_PID, which means that the controller can be used to control systems with two controlled devices for which the transfer behaviours are different. A system with one actuator for heating and another actuator for cooling would be a typical application. To optimise the regulation of such an arrangement, it is possible to switch between two PID parameter records. Switching between the parameter records is implemented in such a way that the control value remains continuous even as the parameter records are changed.

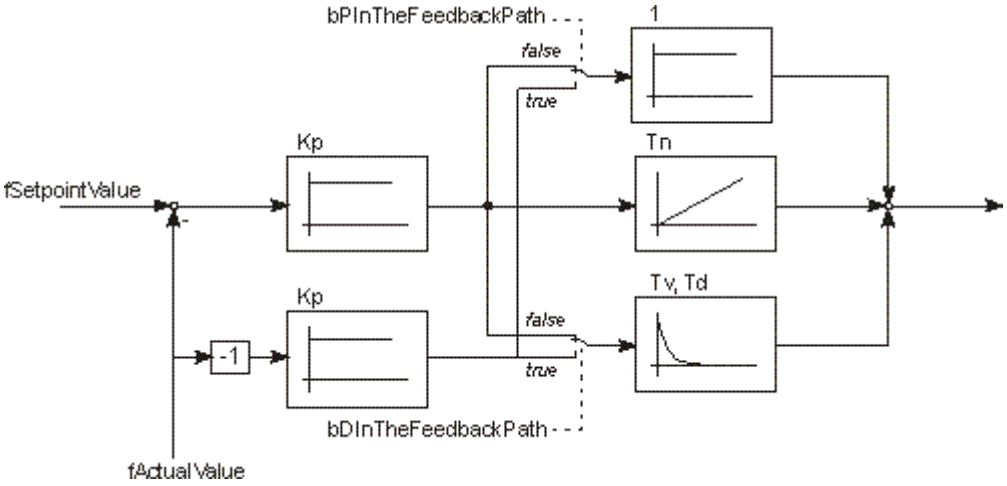
The switching algorithm calculates a linear, time-dependent transition between the two parameter records. The nParameterChangeCycleTicks parameter can be used to specify the number of task cycles over which the continuous change between the two parameter records takes place.

Transfer function:

The following transfer function can be declared for this block, if the boolean inputs **bPInTheFeedbackPath** and **bDInTheFeedbackPath** are set to FALSE, otherwise this transfer function only describes a part of the blocks behaviour:

$$G_{PID}(s) = K_p \left(1 + \frac{1}{T_n s} + \frac{T_v s}{1 + T_d s}\right)$$

Functional diagram:



The standard functional diagram of a PID controller in additive form has been expanded by the two active boolean inputs **bPInTheFeedbackPath** and **bDInTheFeedbackPath** (which act as "switches"), so that a modified functional diagram can be activated.

Control background: due to the differential component of the control algorithm, large control values are generated at setpoint step-changes, which cause a strain on the control elements and may cause the control system to oscillate. A control algorithm with a differential component that is only applied to the controlled variable (bDInTheFeedbackPath := TRUE) can avoid this problem.

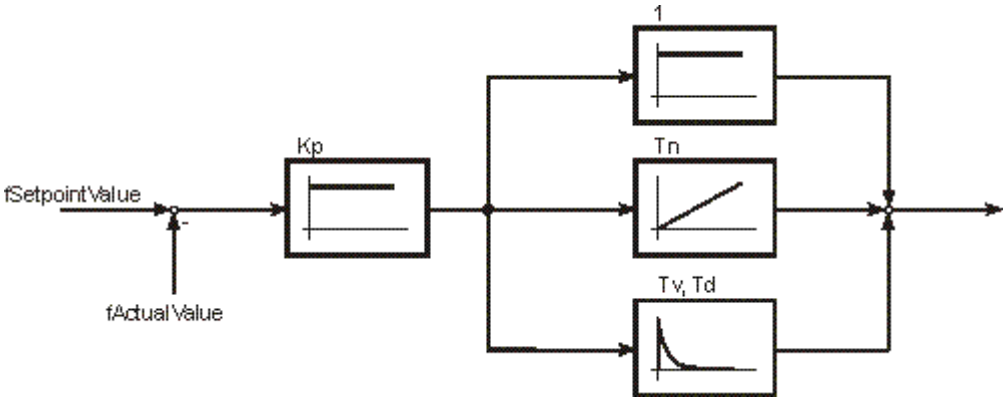
The bPInTheFeedbackPath and bDInTheFeedbackPath inputs permit the closed control loop to implement the following transfer functions:

bPInTheFeedbackPath	bDInTheFeedbackPath	$G(s)$
false	false	$G(s) = \frac{G_{PID}(s) \cdot G_S(s)}{1+G_{PID}(s) \cdot G_S(s)}$
true	false	$G(s) = \frac{G_{DI}(s) \cdot G_S(s)}{1+G_{PID}(s) \cdot G_S(s)}$
false	true	$G(s) = \frac{G_{PI}(s) \cdot G_S(s)}{1+G_{PID}(s) \cdot G_S(s)}$
true	true	$G(s) = \frac{G_I \cdot G_S(s)}{1+G_{PID}(s) \cdot G_S(s)}$

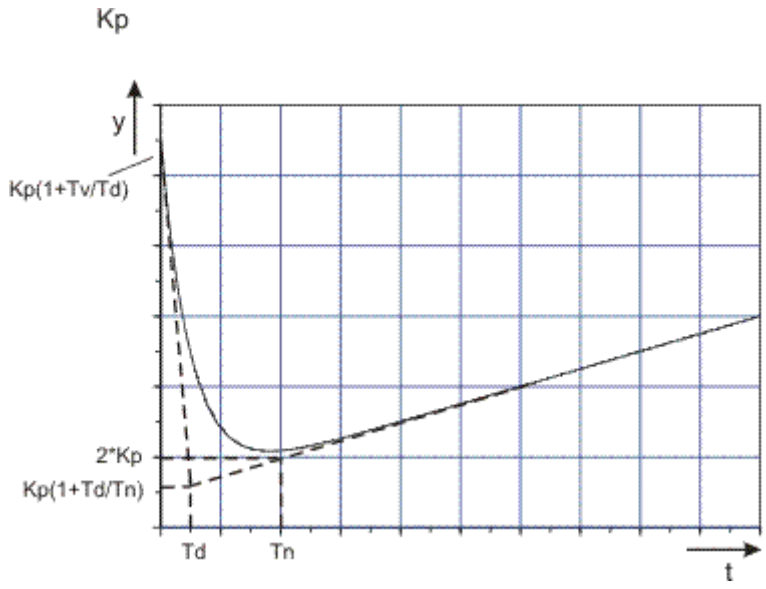
with:

$$G_{DI}(s) = K_p(\frac{1}{T_n s} + \frac{T_v s}{1+T_d s})$$
$$G_{PI}(s) = K_p(1 + \frac{1}{T_n s})$$
$$G_I(s) = K_p(\frac{1}{T_n s})$$

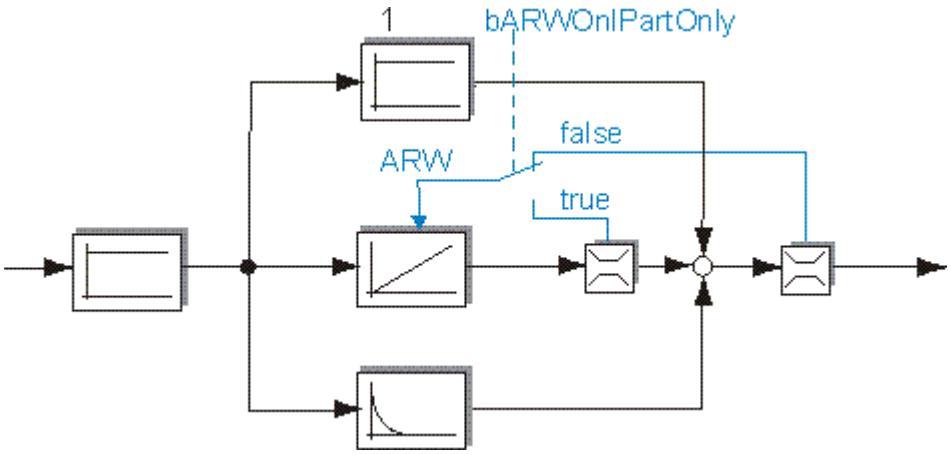
! The default setting for the two inputs **bPInTheFeedbackPath** and **bDInTheFeedbackPath** is **FALSE**. The PID controller then acts as a standard PID controller in additive form.



Step response:



ARW:



VAR_INPUT

```
VAR_INPUT
    fSetpointValue      : FLOAT;
    fActualValue        : FLOAT;
    eParameterRecord        : E_CTRL_PARAMETER_RECORD;
    fManSyncValue       : FLOAT;
    bSync           : BOOL;
    eMode           : E_CTRL_MODE;
    bHold           : BOOL;
END_VAR
```

fSetpointValue : Set value of the controlled variable.

fActualValue : Actual value of the controlled variable.

eParameterRecord : Index of the active parameter record

fManSyncValue : Input with which the PI element can be set.

bSync : A rising edge at this input sets the PI element to the value fManSyncValue.

eMode : Input that specifies the block's operating mode [▶ 16].

bHold : A TRUE at this input will hold the internal state (and therefore also the output) constant at its current value, independently of the system deviation.

VAR_OUTPUT

```
VAR_OUTPUT
    fOutPos             : FLOAT;
    fOutNeg             : FLOAT;
    fOut            : FLOAT;

    bARWActive          : BOOL  := FALSE;       (* ARW active ? [TRUE/FALSE] -> freeze I-part *)
    bParameterChangeActive  : BOOL;

    eState          : E_CTRL_STATE;
    bError          : BOOL;             (* error flag *)
    eErrorId            : E_CTRL_ERRORCODES;    (* error code *)

END_VAR
```

fOutPos : Output of the PID element when the control value is positive. A zero is output otherwise.

fOutNeg : Output of the PID element when the control value is negative. A zero is output otherwise.

fOut : Output of the PID element.

bARWactive : A TRUE at this output indicates that the PID element is being restricted.

bParameterChangeActive : A TRUE at this output indicates that the change from one parameter record to the other is in progress.

fCtrlDerivation : A TRUE at this output indicates that the PID element is being restricted.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_PID_SPLITRANGE_PARAMS;
END_VAR
```

stParams : Parameter structure of the PID element. This consists of the following elements:

```
TYPE
ST_CTRL_PID_SPLITRANGE_PARAMS :
STRUCT
    tCtrlCycleTime          : TIME      := T#0ms; (*
controller cycle time [TIME] *)
    tTaskCycleTime          : TIME      := T#0ms; (* task
cycle time [TIME] *)
    fKp_heating         : FLOAT     := 0;       (*
proportional gain *)
    tTn_heating         : TIME      := T#0ms; (* Tn
*)
    tTv_heating         : TIME      := T#0ms; (* Tv
*)
    tTd_heating         : TIME      := T#0ms; (* Td
*)
    fKp_cooling         : FLOAT     := 0;       (*
proportional gain *)
    tTn_cooling         : TIME      := T#0ms; (* Tn
*)
    tTv_cooling         : TIME      := T#0ms; (* Tv
*)
    tTd_cooling         : TIME      := T#0ms; (* Td
*)
    nParameterChangeCycleTicks : INT;
    fOutMaxLimit            : FLOAT     :=  1E38; (* maximum
output limit *)
    fOutMinLimit            : FLOAT     := -1E38; (* minimum
output limit *)
    bPInTheFeedbackPath     : BOOL;
    bDInTheFeedbackPath     : BOOL;
    bARWOnIPartOnly         : BOOL;
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

Area eCTRL_PARAMETER_RECORD_**HEATING**:

fKp_heating : Controller amplification / controller coefficient

tTn_heating : Integral action time. The I-component is deactivated if this is parameterized as T#0s.

tTv_heating : Derivative action time. The D-component is deactivated if this is parameterized as T#0s.

tTd_heating : Damping time

Area eCTRL_PARAMETER_RECORD_**COOLING**:

fKp_cooling : Controller amplification / controller coefficient

tTn_cooling : Integral action time. The I-component is deactivated if this is parameterized as T#0s.

tTv_cooling : Derivative action time. The D-component is deactivated if this is parameterized as T#0s.

tTd_cooling : Damping time

nParameterChangeCycleTicks The number of task cycles over which the change from one parameter set to the other takes place.

fOutMaxLimit : Upper limit at which integration is halted and to which the output is limited (ARW measure). Reaching this limit is indicated by a TRUE at the bARWActive output.

fOutMinLimit : Lower limit at which integration is halted and to which the output is limited (ARW measure). Reaching this limit is indicated by a TRUE at the bARWActive output.

bPInTheFeedbackPath : Input value of the P element can be selected with this input (see functional diagram).

bDInTheFeedbackPath : Input value of the D element can be selected with this input (see functional diagram).

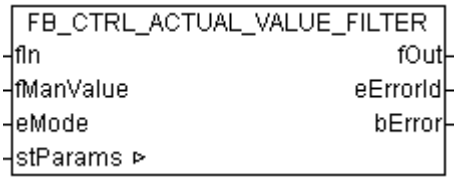
bARWOnIPartOnly: If this parameter is FALSE (the default setting), the integration of the I-component is halted if the complete controller output reaches the upper or lower limit. If it is TRUE, the integration is halted if the I-component (the output of the integrator) reaches some limit. (Cf. functional diagram)

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

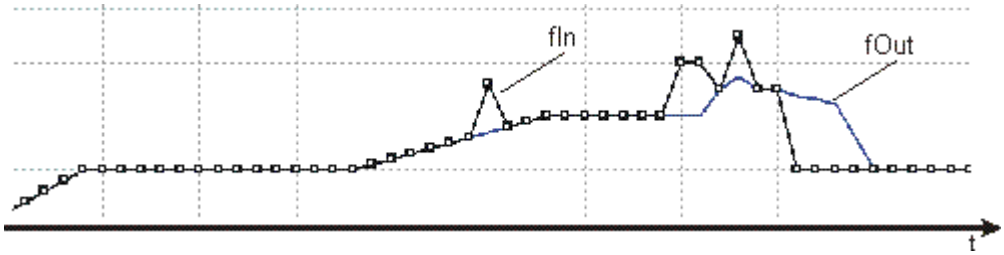
5.5 Filter / ControlledSystemSimulation

5.5.1 FB_CTRL_ACTUAL_VALUE_FILTER



The function block allows a measured input variable to be checked for plausibility and filtered.

Behaviour of the output:



This block allows a plausibility check to be carried out on a measured input variable. If the difference between two sampling values in sequence (measurements) is larger than the specified window, **fDeltaMax**, then the current input value is suppressed for a maximum of three cycles. During this time the output value is extrapolated from the previous input values. If the specified window is exceeded for more than three cycles, the output will again follow the input variable. The behaviour of the output is illustrated in the diagram above.

VAR_INPUT

```
VAR_INPUT
    fIn         : FLOAT;
    fManValue       : FLOAT;
    eMode       : E_CTRL_MODE;
END_VAR
```

fIn : Input value of the filter.

fManValue : Input variable whose value is output in manual mode.

eMode : Input that specifies the block's operating mode [▶ 16].

VAR_OUTPUT

```
VAR_OUTPUT
    fOut        : FLOAT;
    eState      : E_CTRL_STATE;
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

fOut : Output of the function block.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_ACTUAL_VALUE_FILTER_PARAMS;
END_VAR
```

stParams : Parameter structure of the actual value filter element. This consists of the following elements:

```
TYPE
FB_CTRL_ACTUAL_VALUE_FILTER:
STRUCT
    tCtrlCycleTime     : TIME     := T#0ms; (* controller cycle
time [TIME] *)
    tTaskCycleTime     : TIME     := T#0ms; (* task cycle time
[TIME] *)
    fDeltaMax      :FLOAT;
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

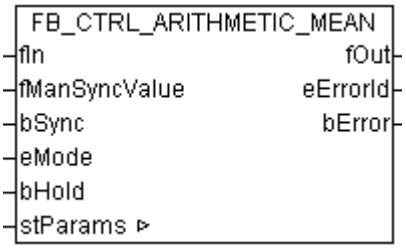
tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

fDeltaMax : Maximum difference between two input values in sequence. See description above.

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.5.2 FB_CTRL_ARITHMETIC_MEAN



Calculating the mean value:

$$\bar{x} = \frac{1}{n} \sum_{n} x_n$$

VAR_INPUT

```
VAR_INPUT
    fIn          : FLOAT;
    fManSyncValue   : FLOAT;
    bSync        : BOOL;
    eMode        : E_CTRL_MODE;
    bHold        : BOOL;
END_VAR
```

fIn : Input value for the mean value filter.

fManSyncValue : Input value to which the mean value filter can be set, or that is issued at the output in manual mode.

bSync : A rising edge at this input sets the mean value filter to the value fManSyncValue.

eMode : Input that specifies the block's operating mode [▶ 16].

bHold : A TRUE at this input will hold the internal state (and therefore also the output) constant at its current value, independently of the input value.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut          : FLOAT;
    eState        : E_CTRL_STATE;
    eErrorId          : E_CTRL_ERRORCODES;
    bError        : BOOL;
END_VAR
```

fOut : Output of the mean value filter.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams         : ST_CTRL_ARITHMETIC_MEAN_PARAMS;
END_VAR
```

stParams : Parameter structure of the average filter. This consists of the following elements:

```
TYPE
ST_CTRL_ARITHMETIC_MEAN_PARAMS:
STRUCT
    tCtrlCycleTime       : TIME     := T#0ms; (* controller
cycle time [TIME] *)
    tTaskCycleTime       : TIME     := T#0ms; (* task cycle
time [TIME] *)
END_STRUCT
END_TYPE
```

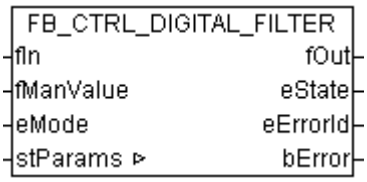
tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

Requirements

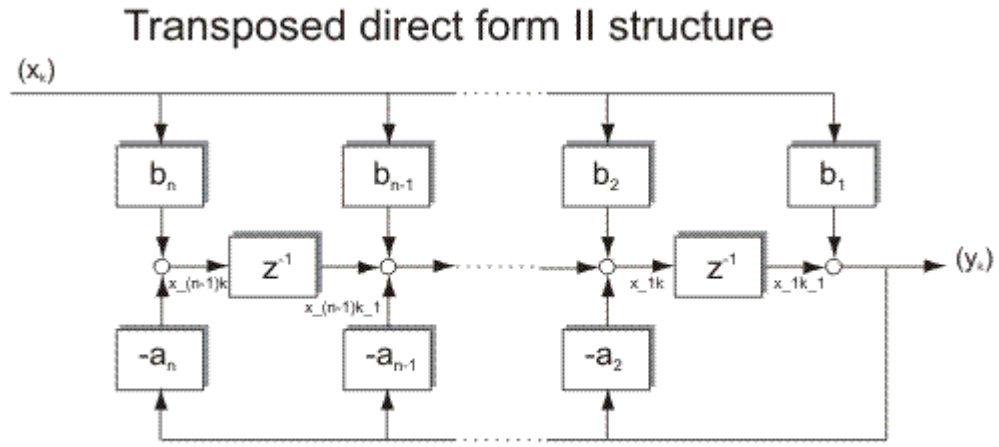
Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.5.3 FB_CTRL_DIGITAL_FILTER



This function block calculates a discrete transfer function with the structure described below. This structure allows either an FIR filter (**F**inite **I**mpulse **R**esponse) or an IIR filter (**I**nfinite **I**mpulse **R**esponse) to be implemented. The transfer function here can be of any order, n.

The coefficients for the following transfer structure are stored in the parameter arrays:



order: n-1

$$G(z) = \frac{Y(z)}{X(z)} = \frac{b_1 + b_2 z^{-1} + b_3 z^{-2} + \cdots + b_n z^{-(n-1)}}{1 + a_2 z^{-1} + a_3 z^{-2} + \cdots + a_n z^{-(n-1)}}$$

The programmer must create the following arrays in the PLC if this function block is to be used:

```
ar_CoefficientsArray_a     :
ARRAY[1..nFilterOrder+1] OF FLOAT;
ar_CoefficientsArray_b     : ARRAY[1..nFilterOrder+1] OF
FLOAT;
ar_stDigitalFilterData     : ARRAY[1..nFilterOrder  ] OF
ST_CTRL_DIGITAL_FILTER_DATA;
```

The coefficients b_1 to b_n are stored in the array ar_CoefficientsArray_b. This must be organized as follows:

```
ar_CoefficientsArray_b[ 1 ]     := b1;
ar_CoefficientsArray_b[ 2 ]     := b2;
...
ar_CoefficientsArray_b[ n-1 ]   := bn-1;
ar_CoefficientsArray_b[ n ]     := bn;
```

The coefficients a_1 to a_n are stored in the array ar_CoefficientsArray_a. This must be organized as follows:

```
ar_CoefficientsArray_a[ 1 ]    := xxx;    (* wird nicht ausgewertet
*)
ar_CoefficientsArray_a[ 2 ]    := a2;
...
ar_CoefficientsArray_a[ n-1 ]  := an-1;
ar_CoefficientsArray_a[ n ]    := an;
```

The internal data required by the function block is stored in the ar_stDigitalFilterData array. This data must never be modified from within the PLC program. This procedure is also illustrated in the sample program listed below.

VAR_INPUT

```
VAR_INPUT
    fIn          : FLOAT;
    fManValue       : FLOAT;
    eMode        : E_CTRL_MODE;
END_VAR
```

fIn : Input of the digital filter.

fManValue : Input whose value is present at the output in manual mode.

eMode : Input that specifies the block's operating mode.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut         : FLOAT;
    eState       : E_CTRL_STATE;
```

```
    bError      : BOOL;
    eErrorId        : E_CTRL_ERRORCODES;
END_VAR
```

fOut : Output of the digital filter.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_DIGITAL_FILTER_PARAMS;
END_VAR
```

stParams : Parameter structure of the function block. This consists of the following elements:

```
TYPE
ST_CTRL_DIGITAL_FILTER_PARAMS :
STRUCT
    tTaskCycleTime          : TIME;      (* task cycle
time       *)
    tCtrlCycleTime          : TIME := T#0ms; (* controller
cycle time *)
    nFilterOrder        : USINT;
    pCoefficientsArray_a_ADR    : POINTER TO FLOAT := 0;
    nCoefficientsArray_a_SIZEOF : UINT;
    pCoefficientsArray_b_ADR    : POINTER TO FLOAT := 0;
    nCoefficientsArray_b_SIZEOF : UINT;
    pDigitalFilterData_ADR      : POINTER TO
ST_CTRL_DIGITAL_FILTER_DATA;
    nDigitalFilterData_SIZEOF   : UINT;
END_STRUCT
END_TYPE
```

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

nFilterOrder : Order of the digital filter [0...]

pCoefficientsArray_a_ADR : Address of the array containing the a coefficients.

nCoefficientsArray_a_SIZEOF : Size of the array containing the a coefficients, in bytes.

pCoefficientsArray_b_ADR : Address of the array containing the b coefficients.

nCoefficientsArray_b_SIZEOF : Size of the array containing the b coefficients, in bytes.

pDigitalFilterData_ADR : Address of the data array.

nDigitalFilterData_SIZEOF : Size of the data array in bytes.

```
TYPE
ST_CTRL_DIGITAL_FILTER_DATA
STRUCT
    Interne Struktur. Auf diese darf nicht schreibend
zugegriffen werden.
END_STRUCT
END_TYPE
```

Sample:

```
PROGRAM PRG_DIGITAL_FILTER_TEST
VAR
    fbDigitalFilter : FB_CTRL_DIGITAL_FILTER;
    ar_CoefficientsArray_a : ARRAY[1..3 ] OF FLOAT;
    ar_CoefficientsArray_b : ARRAY[1..3 ] OF FLOAT;
    ar_stDigitalFilterData : ARRAY[1..2 ] OF
ST_CTRL_DIGITAL_FILTER_DATA;
```

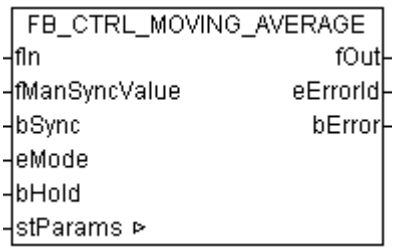
```
    eMode     : E_CTRL_MODE;
    stParams  : ST_CTRL_DIGITAL_FILTER_PARAMS;
    eErrorId  : E_CTRL_ERRORCODES;
    bError    : BOOL;
    fIn       : FLOAT := 0;
    fOut      : FLOAT;
    bInit     : BOOL := TRUE;
END_VAR
```

```
IF bInit
THEN
    (* set values in the local arrays *)
    ar_CoefficientsArray_a[1] := 0.0;    (* not used *)
    ar_CoefficientsArray_a[2] := 0.2;
    ar_CoefficientsArray_a[3] := 0.1;
    ar_CoefficientsArray_b[1] := 0.6;
    ar_CoefficientsArray_b[2] := 0.4;
    ar_CoefficientsArray_b[3] := 0.2;
    (* set values in the parameter struct *)
    stParams.tTaskCycleTime := T#2ms;
    stParams.tCtrlCycleTime := T#2ms;
    stParams.nFilterOrder   := 2;
    (* set the mode *)
    eMode := eCTRL_MODE_ACTIVE;
    bInit := FALSE;
END_IF
(* set addresses *)
stParams.pCoefficientsArray_a_ADR    := ADR(
ar_CoefficientsArray_a);
stParams.nCoefficientsArray_a_SIZEOF := SIZEOF(
ar_CoefficientsArray_a);
stParams.pCoefficientsArray_b_ADR    := ADR(
ar_CoefficientsArray_b);
stParams.nCoefficientsArray_b_SIZEOF := SIZEOF(
ar_CoefficientsArray_b);
stParams.pDigitalFilterData_ADR      := ADR( ar_stDigitalFilterData
);
stParams.nDigitalFilterData_SIZEOF   := SIZEOF(
ar_stDigitalFilterData );
fbDigitalFilter ( fIn       := fIn,
          eMode     := eMode,
          stParams  := stParams,
          fOut      => fOut,
          eErrorId  => eErrorId,
          bError    => bError);
```

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9, build 947 onwards	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9, build 956 onwards	BX [▶ 10]	TcControllerToolbox.lbx

5.5.4 FB_CTRL_MOVING_AVERAGE



The function block provides a moving mean value filter in the functional diagram.

The arithmetic mean of the last n values is calculated.

$$\bar{x}_k^n = \frac{1}{n} \sum_{i=k-n}^{k} x_i$$

The programmer must create an array, ARRAY [1.. n] of FLOAT, in which the function block can store the data that it requires internally.

VAR_INPUT

```
VAR_INPUT
    fIn          : FLOAT;
    fManSyncValue    : FLOAT;
    bSync        : BOOL;
    eMode        : E_CTRL_MODE;
    bHold        : BOOL;
END_VAR
```

fIn : Input variable for the moving average filter.

fManSyncValue : Input value to which the moving average filter can be set, or that is issued at the output in manual mode.

bSync : A rising edge at this input sets the moving average filter to the value fManSyncValue.

eMode : Input that specifies the block's operating mode [▶ 16].

bHold : A TRUE at this input will hold the internal state (and therefore also the output) constant at its current value, independently of the input value.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut          : FLOAT;
    eState        : E_CTRL_STATE;
    eErrorId          : E_CTRL_ERRORCODES;
    bError        : BOOL;
END_VAR
```

fOut : Output of the moving average filter.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams         : ST_CTRL_MOVING_AVERAGE_PARAMS;
END_VAR
```

stParams : Parameter structure of the moving average filter. This consists of the following elements:

```
TYPE
ST_CTRL_MOVING_AVERAGE_PARAMS:
STRUCT
    tCtrlCycleTime       : TIME     := T#0ms; (* controller
cycle time [TIME] *)
    tTaskCycleTime       : TIME     := T#0ms; (* task cycle
time [TIME] *)
    nSamplesToFilter     : UINT;
    pWorkArray_ADR       : POINTER TO FLOAT := 0;
    nWorkArray_SIZEOF    : UINT         := 0;
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

nSamplesToFilter : The number of values, n, whose arithmetic average value is calculated.

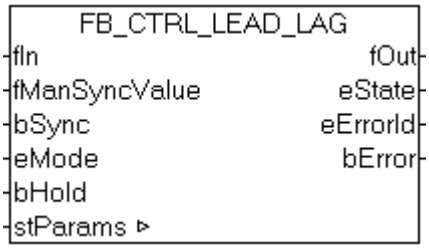
pWorkArray_ADR : The address of the array where the input values are temporarily stored.

nWorkArray_SIZEOF: Size of the WorkArray.

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.5.5 FB_CTRL_LEAD_LAG

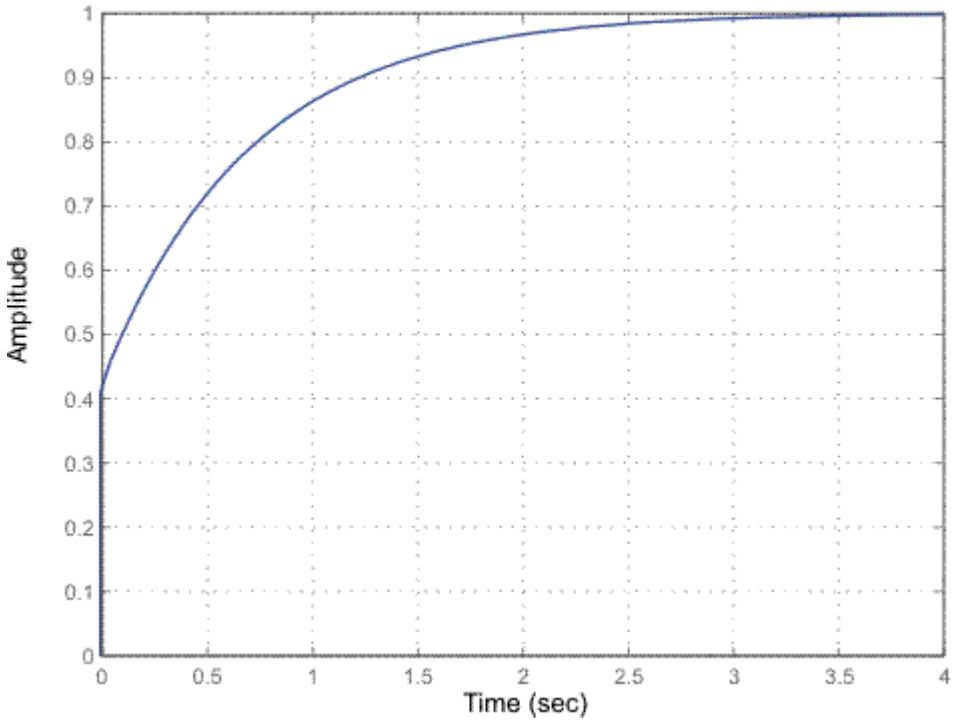


The function block represents a digital lead/lag filter.

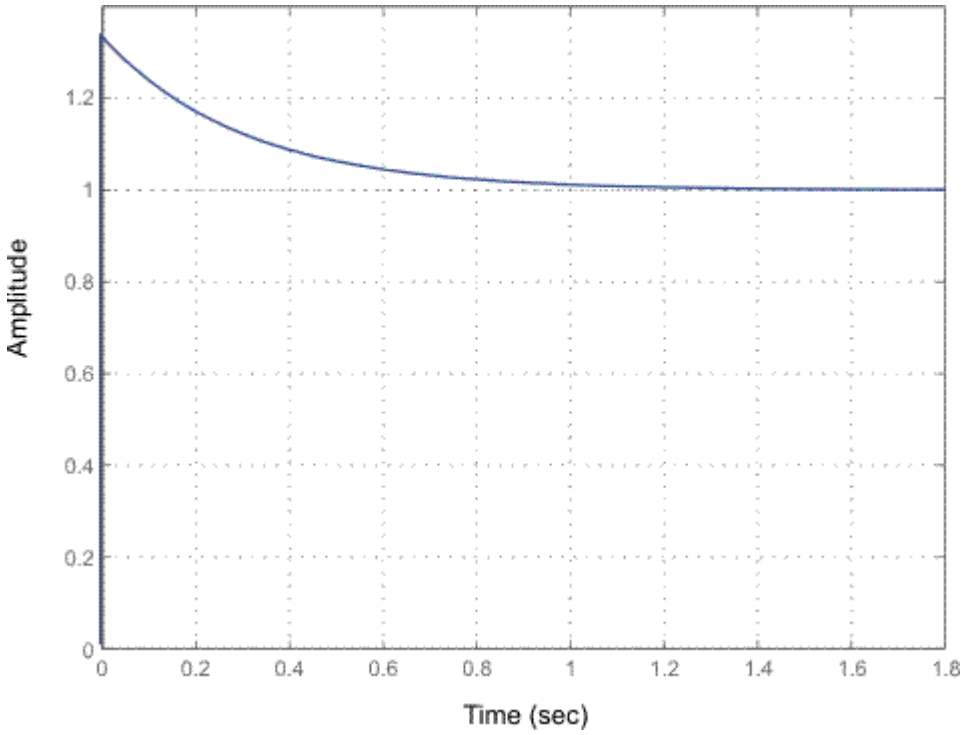
Transfer function:

$$G(s) = \frac{1 + T_1 s}{1 + T_2 s}$$

Step response with T2 > T1:



Step response with T1 > T2:



The step response at time T=0 is T1 / T2.

VAR_INPUT

```
VAR_INPUT
    fIn             : FLOAT;
    fManSyncValue   : FLOAT;
    bSync           : FLOAT;
    fManValue       : FLOAT;
    eMode           : E_CTRL_MODE;
END_VAR
```

fIn : Input value of the notch filter.

fManSyncValue : Input value that is output at the output in manual mode.

bSync : Reserve.

eMode : Input that specifies the operation mode [▶ 16] of the function block.

bHold : A TRUE at this input will hold the internal state (and therefore also the output) constant at its current value, independently of the input value.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut         : FLOAT;
    eState       : E_CTRL_STATE;
    eErrorId          : E_CTRL_ERRORCODES;
    bError       : BOOL;
END_VAR
```

fOut : lead/lag filter output.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_LEAD_LAG_FILTER_PARAMS;
END_VAR
```

stParams : Parameter structure of the lead/lag filter. This consists of the following elements:

```
TYPE
ST_CTRL_LEAD_LAG_FILTER_PARAMS:STRUCT    tCtrlCycleTime : TIME :=
T#0ms; (* controller cycle time [TIME] *)    tTaskCycleTime : TIME :=
T#0ms; (* task cycle time [TIME] *)    tT1        : TIME :=
T#0ms; (* T1 *)

    tT2        : TIME := T#0ms; (* T2 *)END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

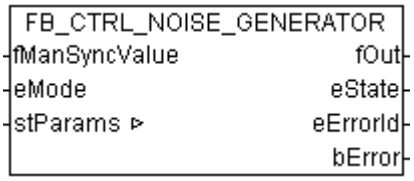
tT1 : T1, see G(s)

tT2 : T2, see G(s)

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9, build 947 onwards	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9, build 956 onwards	BX [▶ 10]	TcControllerToolbox.lbx

5.5.6 FB_CTRL_NOISE_GENERATOR (only on a PC system)



This function block generates a noise signal on the basis of the pseudo-random number in the range [-fAmplitude/2 ... fAmplitude/2] .

Output signal:

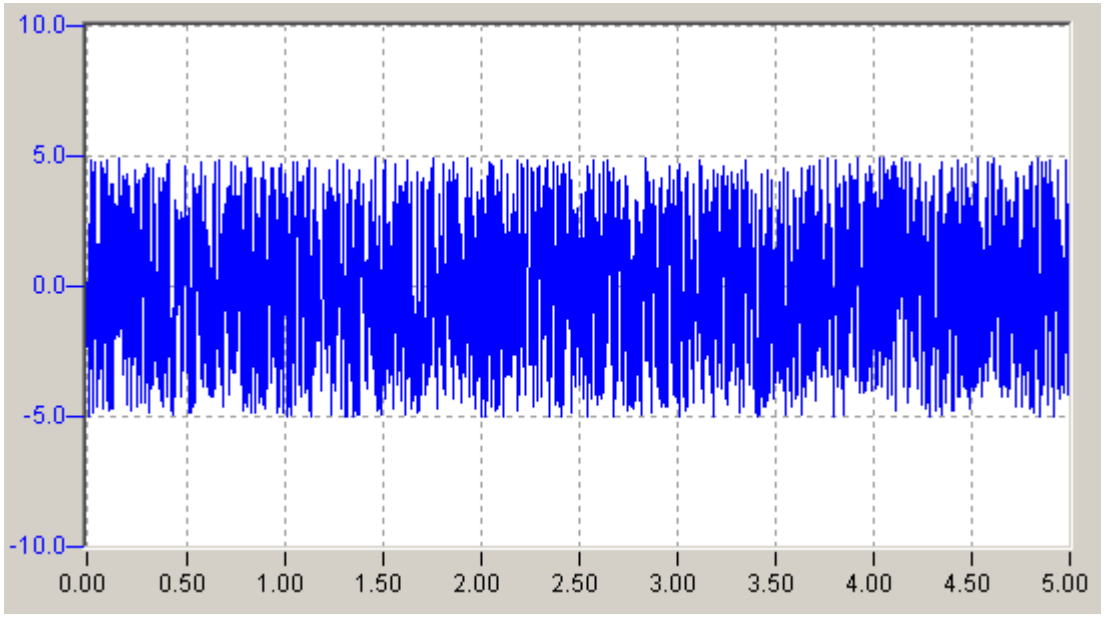


Fig. 3: FB_CTRL_NOISE_GENERATOR_OUT

Output signal with an amplitude of 5.0.

VAR_INPUT

```
VAR_INPUT
    fManSyncValue    : FLOAT;
    eMode        : E_CTRL_MODE;
END_VAR
```

fManSyncValue : Input magnitude whose value is sent to the output in manual mode.

eMode : Input that specifies the block's operating mode [▶ 16].

VAR_OUTPUT

```
VAR_OUTPUT
    fOut         : FLOAT;
    eState       : E_CTRL_STATE;
    eErrorId         : E_CTRL_ERRORCODES;
    bError       : BOOL;
END_VAR
```

fOut : Output of the noise generator.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_NOISE_GENERATOR_PARAMS;
END_VAR
```

stParams : Parameter structure of the noise generator. This consists of the following elements:

```
TYPE
ST_CTRL_NOISE_GENERATOR_PARAMS:
STRUCT
    tCtrlCycleTime : TIME     := T#0ms; (* controller cycle
time [TIME] *)
    tTaskCycleTime : TIME     := T#0ms; (* task cycle time
[TIME]       *)
    fAmplitude     : FLOAT    := 0;
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

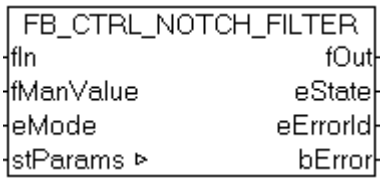
tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

fAmplitude : Amplitude of the output signal. A noise signal extending over the range [-fAmplitude/2.0 ... fAmplitude/2.0] is created at the function block's output.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib

5.5.7 FB_CTRL_NOTCH_FILTER

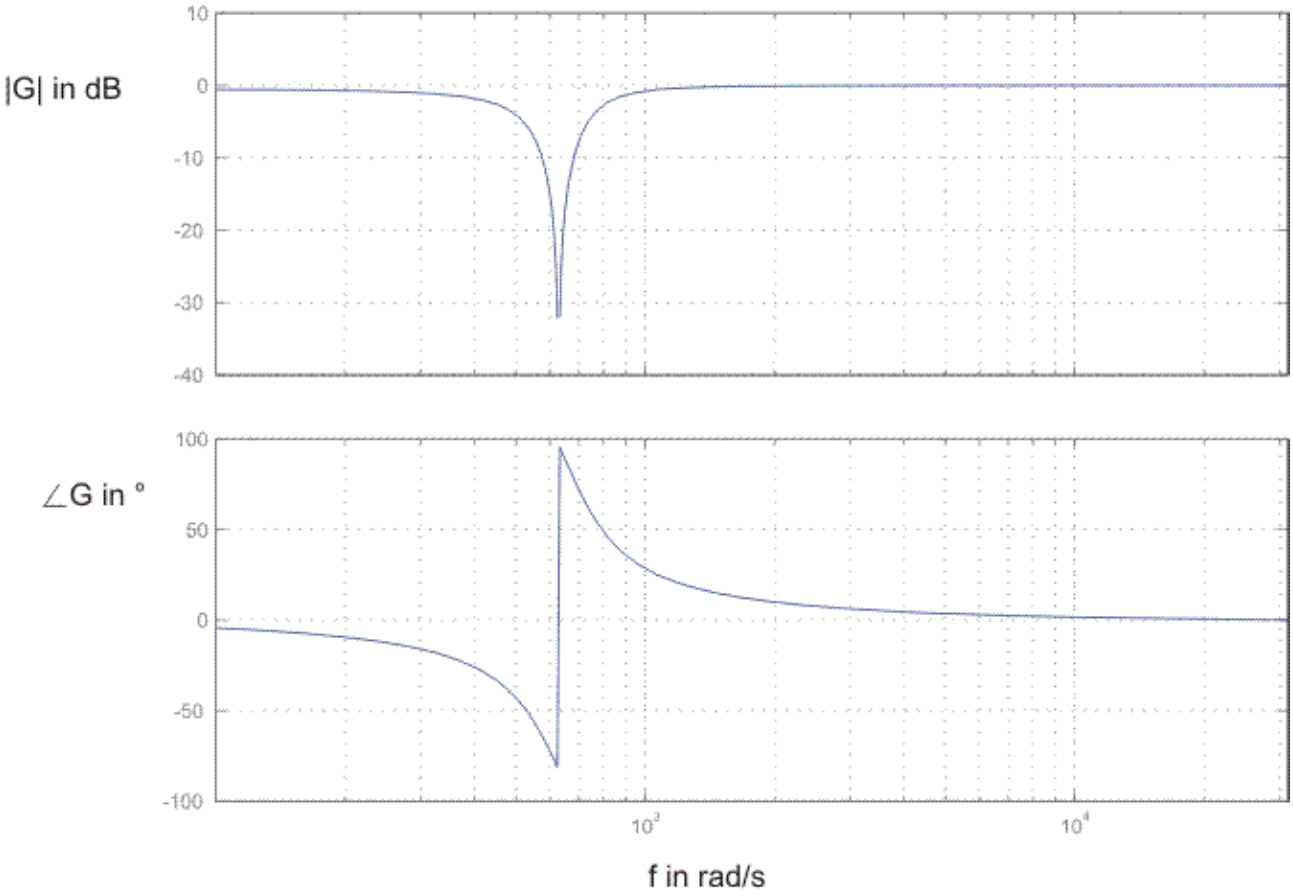


The function block represents a digital notch filter.

Transfer function:

$$G(s) = \frac{s^2 - \omega_0^2}{(s - i\epsilon\omega_0)^2 - \omega_0^2}$$

Bode diagram:



with:

$$f_0 = 100\,\mathrm{Hz}$$
$$\epsilon = 0.25$$

VAR_INPUT

```
VAR_INPUT
    fIn         : FLOAT;
    fManValue   : FLOAT;
    eMode       : E_CTRL_MODE;
END_VAR
```

fIn : Input value of the notch filter.

fManValue : Input value that is output at the output in manual mode.

eMode : Input that specifies the operation mode [▶ 16] of the function block.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut          : FLOAT;
    eState        : E_CTRL_STATE;
    eErrorId          : E_CTRL_ERRORCODES;
    bError        : BOOL;
END_VAR
```

fOut : notch filter output.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_NOTCH_FILTER_PARAMS;
END_VAR
```

stParams : Parameter structure of the notch filter. This consists of the following elements:

```
TYPE
ST_CTRL_NOTCH_FILTER_PARAMS:STRUCT    tCtrlCycleTime : TIME :=
T#0ms; (* controller cycle time [TIME] *)    tTaskCycleTime : TIME :=
T#0ms; (* task cycle time [TIME] *)    fNotchFreq : FLOAT := 0;
(* [Hz] *)    fBandwidth : FLOAT := 0;
(* Bandwidth in Hz = fBandwidth*fNotchFreq *)END_STRUCTEND_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

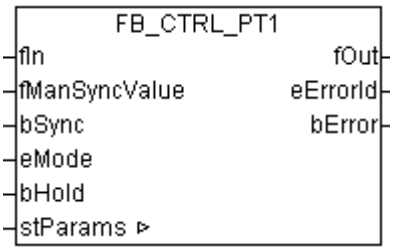
fNotchFreq : Notch frequency in Hz

fBandwidth : Bandwidth related to the notch frequency: bandwidth in Hz = fNotchFreq * fBandwidth

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9, build 947 onwards	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9, build 956 onwards	BX [▶ 10]	TcControllerToolbox.lbx

5.5.8 FB_CTRL_PT1

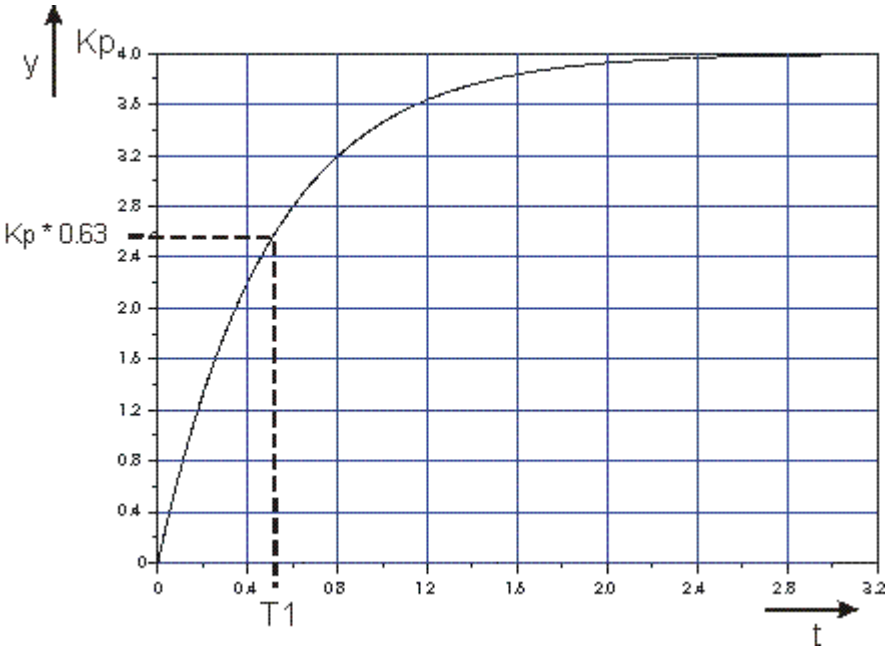


The function block provides a PT1 transfer element in the functional diagram.

Transfer function:

$$G(s) = K_p \frac{1}{1 + T_1 s}$$

Step response:



VAR_INPUT

```
VAR_INPUT
    fIn          : FLOAT;
    fManSyncValue    : FLOAT;
    bSync        : BOOL;
    eMode        : E_CTRL_MODE;
    bHold        : BOOL;
END_VAR
```

fIn : Input value of the PT1 element.

fManSyncValue : Input value to which the PT1 element can be set, or that is issued at the output in manual mode.

bSync : A rising edge at this input sets the PT1 element to the value fManSyncValue.

eMode : Input that specifies the block's operating mode [▶ 16].

bHold : A TRUE at this input will hold the internal state (and therefore also the output) constant at its current value, independently of the input value.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut         : FLOAT;
    eState       : E_CTRL_STATE;
    eErrorId         : E_CTRL_ERRORCODES;
    bError       : BOOL;
END_VAR
```

fOut : Output of the PT1 element.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams         : ST_CTRL_PT1_PARAMS;
END_VAR
```

stParams : Parameter structure of the PT1 element. This consists of the following elements:

```
TYPE
ST_CTRL_PT1_PARAMS :
STRUCT
    tCtrlCycleTime  : TIME     := T#0ms; (* controller cycle
time [TIME] *)
    tTaskCycleTime  : TIME     := T#0ms; (* task cycle time
[TIME] *)
    fKp         : FLOAT    := 0;     (* proportional gain
*)
    tT1         : TIME     := T#0ms; (* T1 *)
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

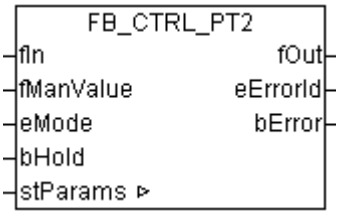
fKp : Controller amplification / transfer coefficient

tT1 : Time constant

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.5.9 FB_CTRL_PT2

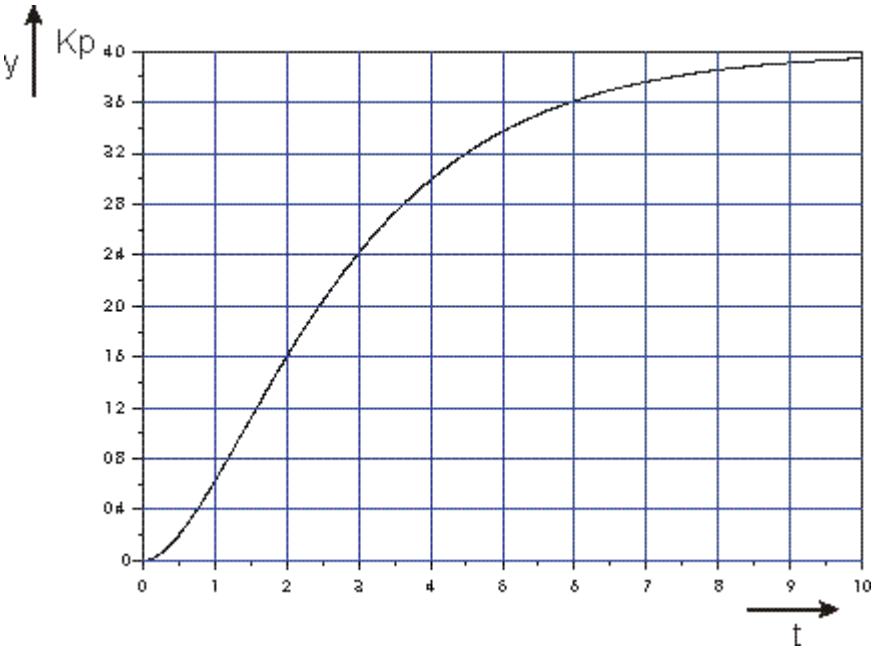


The function block provides a non-oscillating PT2 transfer element in the functional diagram.

Transfer function:

$$G(s) = K_p \frac{1}{1 + T_1 s} \frac{1}{1 + T_2 s}$$

Step response:



VAR_INPUT

```
VAR_INPUT
    fIn          : FLOAT;
    fManValue    : FLOAT;
    eMode        : E_CTRL_MODE;
    bHold        : BOOL;
END_VAR
```

fIn : Input value of the PT2 element.

fManValue : Input value that is output in manual mode.

eMode : Input that specifies the operation mode [▶ 16] of the function block.

bHold : A TRUE at this input will hold the internal state (and therefore also the output) constant at its current value, independently of the input value.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut     : FLOAT;
    eState   : E_CTRL_STATE;
    eErrorId : E_CTRL_ERRORCODES;
    bError   : BOOL;
END_VAR
```

fOut : Output of the PT2 element.

eState : State of the function block.

nErrorID : Returns the ADS error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_PT2_PARAMS;
END_VAR
```

stParams : Parameter structure of the PT2 element. This consists of the following elements:

```
TYPE
ST_CTRL_PT2_PARAMS :
STRUCT
    tCtrlCycleTime  : TIME     := T#0ms; (* controller cycle
```

```
time [TIME] *)
    tTaskCycleTime  : TIME     := T#0ms; (* task cycle time
[TIME] *)
    fKp         : FLOAT    := 0;     (* proportional gain
*)
    tT1         : TIME     := T#0ms; (* T1 *)
    tT2         : TIME     := T#0ms; (* T2 *)
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

fKp : Controller amplification / transfer coefficient

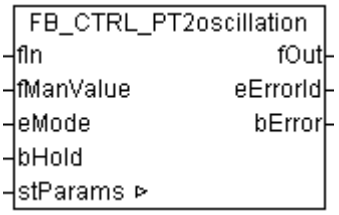
tT1 : Time constant T1

tT2 : Time constant T2

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.5.10 FB_CTRL_PT2oscillation

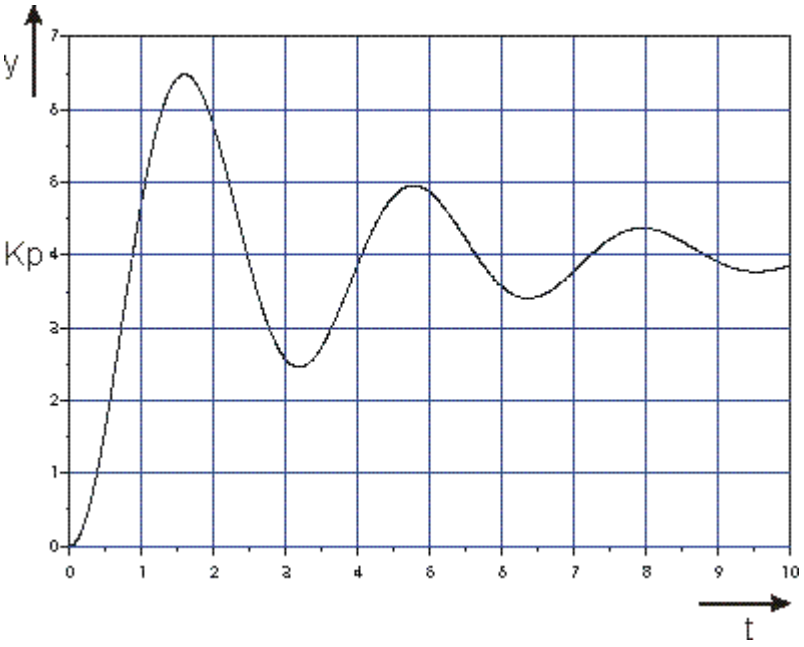


The function block provides an oscillating PT2 transfer element in the functional diagram.

Transfer function:

$$G(s) = K_p \frac{1}{1 + 2\vartheta T_0 s + T_0^2 s^2}$$

Step response:



VAR_INPUT

```
VAR_INPUT
    fIn         : FLOAT;
    fManValue       : FLOAT;
    eMode       : E_CTRL_MODE;
    bHold       : BOOL;
END_VAR
```

fIn : Input variable of the oscillating PT2 element.

fManValue : Input value that is output in manual mode.

eState : State of the function block.

eMode : Input that specifies the block's operating mode [▶ 16].

bHold : A TRUE at this input will hold the internal state (and therefore also the output) constant at its current value, independently of the input value.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut    : FLOAT;
    eState  : E_CTRL_STATE;
    eErrorId : E_CTRL_ERRORCODES;
    bError  : BOOL;
END_VAR
```

fOut : Output of the PT2 element.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_PT2oscillation_PARAMS;
END_VAR
```

stParams : Parameter structure of the oscillating PT2 element. This consists of the following elements:

```
TYPE
ST_CTRL_PT2oscillation_PARAMS :
STRUCT
    tCtrlCycleTime     : TIME     := T#0ms; (* controller cycle
```

```
time [TIME] *)
    tTaskCycleTime      : TIME      := T#0ms; (* task cycle time
[TIME] *)
    fKp         : FLOAT     := 0;      (* proportional
gain *)
    fTheta          : FLOAT     := 0;      (* damping ratio
*)
    tT0         : TIME      := T#0ms; (* T0 *)
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

fKp : Proportional coefficient

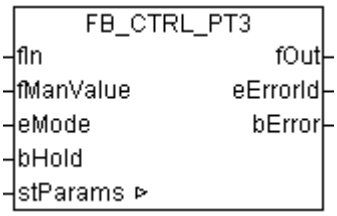
fTheta : Damping ratio

tT0 : Characteristic time

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.5.11 FB_CTRL_PT3

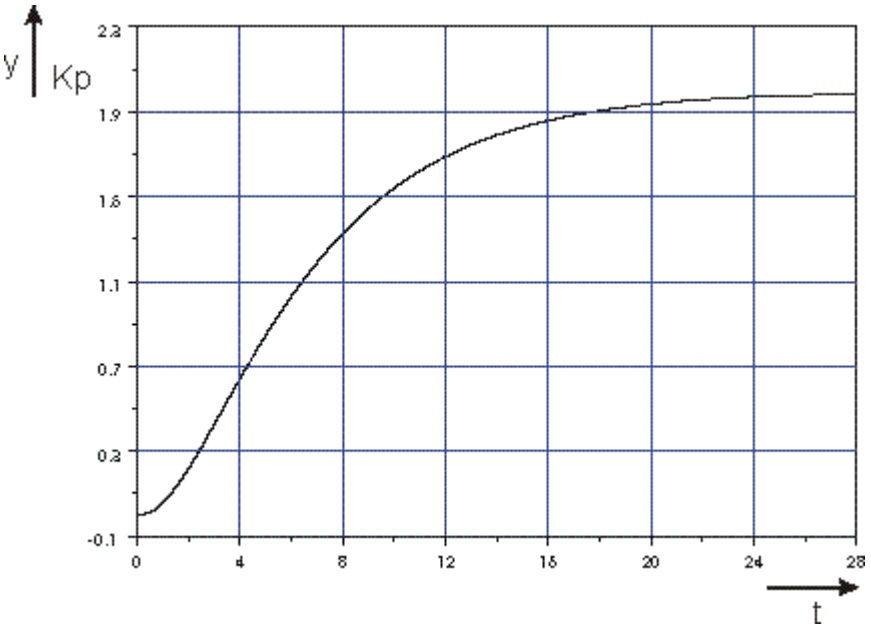


The function block provides a non-oscillating PT3 transfer element in the functional diagram.

Transfer function:

$$G(s) = K_p \frac{1}{1+T_1 s} \frac{1}{1+T_2 s} \frac{1}{1+T_3 s}$$

Step response:



VAR_INPUT

```
VAR_INPUT
    fIn          : FLOAT;
    fManValue       : FLOAT;
    eMode        : E_CTRL_MODE;
    bHold        : BOOL;
END_VAR
```

fIn : Input value of the PT3 element.

fManValue : Input value that is output in manual mode.

eMode : Input that specifies the block's operating mode [▶ 16].

bHold : A TRUE at this input will hold the internal state (and therefore also the output) constant at its current value, independently of the input value.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut    : FLOAT;
    eState  : E_CTRL_STATE;
    eErrorId : E_CTRL_ERRORCODES;
    bError  : BOOL;
END_VAR
```

fOut : Output of the PT3 element.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_PT3_PARAMS;
END_VAR
```

stParams : Parameter structure of the PT3 element. This consists of the following elements:

```
TYPE
ST_CTRL_PT3_PARAMS :
STRUCT
    tCtrlCycleTime  : TIME     := T#0ms; (* controller cycle
time [TIME] *)
```

```
    tTaskCycleTime  : TIME     := T#0ms; (* task cycle time
[TIME] *)
    fKp         : FLOAT    := 0;     (* proportional gain
*)
    tT1         : TIME     := T#0ms; (* T1 *)
    tT2         : TIME     := T#0ms; (* T2 *)
    tT3         : TIME     := T#0ms; (* T3 *)
END_STRUCT
END_TYPE>
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

fKp : Controller amplification / transfer coefficient

tT1 : Time constant T1

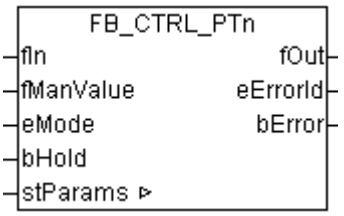
tT2 : Time constant T2p

tT3 : Time constant T3

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.5.12 FB_CTRL_PTn

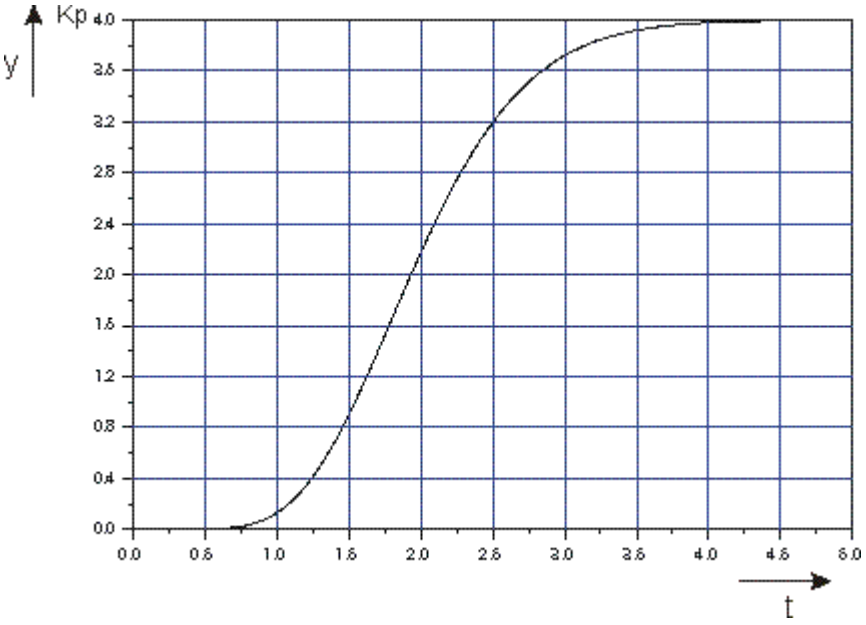


This function block provides a non-oscillating PTn transfer element with (n<= 10) and equal time constants in the functional diagram.

Transfer function:

$$G(s) = K_p \frac{1}{(1 + T_1 s)^n}$$

Step response with n=10:



VAR_INPUT

```
VAR_INPUT
    fIn         : FLOAT;
    fManValue       : FLOAT;
    eMode       : E_CTRL_MODE;
    bHold       : BOOL;
END_VAR
```

fIn : Input value of the PTn element.

fManValue : Input value that is output in manual mode.

eMode : Input that specifies the block's operating mode [▶ 16].

bHold : A TRUE at this input will hold the internal state (and therefore also the output) constant at its current value, independently of the input value.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut    : FLOAT;
    eState  : E_CTRL_STATE;
    eErrorId : E_CTRL_ERRORCODES;
    bError  : BOOL;
END_VAR
```

fOut : Output of the PTn element.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_PTn_PARAMS;
END_VAR
```

stParams : Parameter structure of the PTn element. This consists of the following elements:

```
TYPE
ST_CTRL_PTn_PARAMS :
STRUCT
    tCtrlCycleTime  : TIME     := T#0ms; (* controller cycle
time [TIME] *)
```

```
    tTaskCycleTime  : TIME     := T#0ms; (* task cycle time
[TIME] *)
    fKp         : FLOAT    := 0;     (* proportional gain
*)
    tT1         : TIME     := T#0ms; (* T1 *)
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

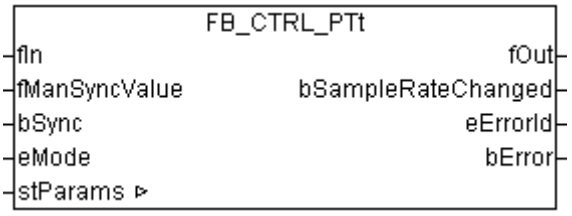
fKp : Controller amplification / transfer coefficient

tT1 : Time constant T1

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.5.13 FB_CTRL_PTt



The function block provides an PTt transfer element in the functional diagram.

Transfer function:

$$G(s) = K_p \cdot e^{-T_t s}$$

This function block contains internally an array of 500 elements with which the input values can be delayed. Using tCtrlCycleTime yields a maximum delay of 500 * tCtrlCycleTime. If this maximum delay is insufficient, the sampling time is extended internally to make it possible to reach the requested dead time. It should, however, be remembered that this process involves increasing the time between the discretisation steps. If a new sampling time has been calculated, this is indicated by a TRUE on the bSampleRateChanged output.

VAR_INPUT

```
VAR_INPUT
    fIn         : FLOAT;
    fManSyncValue   : FLOAT;
    bSync       : BOOL;
    eMode       : E_CTRL_MODE;
END_VAR
```

fIn : Input variable of the PTt element.

fManSyncValue : Input value to which the PTt element can be set, or that is issued at the output in manual mode.

bSync : A rising edge at this input sets the PTt element to the value fManSyncValue.

eMode : Input that specifies the block's operating mode [▶ 16].

VAR_OUTPUT

```
VAR_OUTPUT
    fOut          : FLOAT;
    bSampleRateChanged        : BOOL;
    eErrorId          : E_CTRL_ERRORCODES;
    bError        : BOOL;
END_VAR
```

fOut : Output of the PTt element.

bSampleRateChanged : Output that indicates whether the block has internally reduced the sampling rate because of the array being used to delay the input signal not otherwise providing sufficient room.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams         : ST_CTRL_PTt_PARAMS;
END_VAR
```

stParams : Parameter structure of the PTt element. This consists of the following elements:

```
TYPE
ST_CTRL_PTt_PARAMS :
STRUCT
    tCtrlCycleTime   : TIME      := T#0ms; (* controller cycle
time [TIME] *)
    tTaskCycleTime   : TIME      := T#0ms; (* task cycle time
[TIME] *)
    fKp          : FLOAT     := 0;      (* proportional gain
*)
    tTt          : TIME      := T#0ms; (* Tt *)
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

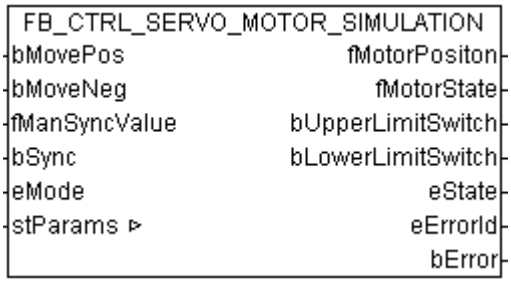
fKp : Controller amplification / transfer coefficient

tTt : Dead time

Requirements

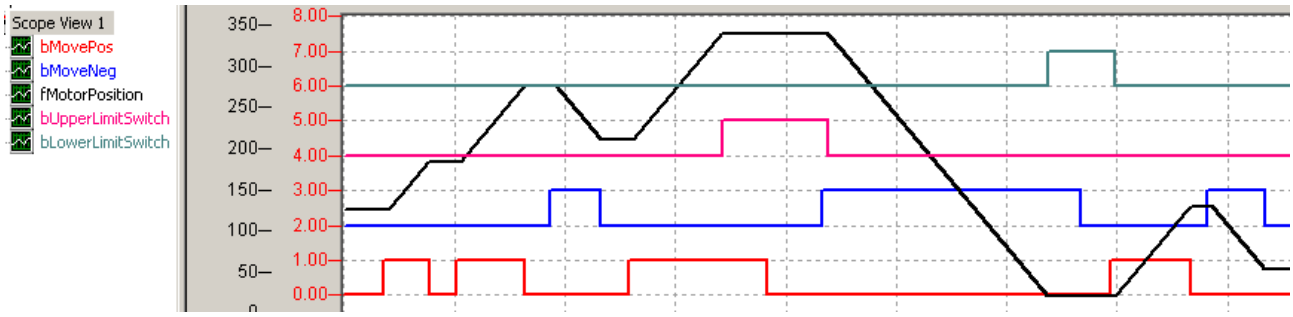
Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.5.14 FB_CTRL_SERVO_MOTOR_SIMULATION (only on a PC system)



The behavior of an actuator can be simulated with this function block.

Behavior of the output:



VAR_INPUT

```
VAR_INPUT
    bMovePos        : BOOL;
    bMoveNeg        : BOOL;
    fManSyncValue   : FLOAT;
    bSync       : BOOL;
    eMode       : E_CTRL_MODE;
END_VAR
```

bMovePos : Input that moves the simulated actuator in the positive direction.

bMoveNeg : Input that moves the simulated actuator in the negative direction.

fManSyncValue : Input with which the simulated motor position can be set, or the value to which movement takes place in manual mode.

bSync : A rising edge at this input sets the simulated motor position to the value fManSyncValue.

eMode : Input that specifies the block's operating mode [▶ 16].

VAR_OUTPUT

```
VAR_OUTPUT
    fMotorPositon   : FLOAT; (* [ fMovingRangeMin ... fMovingRangeMax ] *)
    fMotorState     : FLOAT; (* [ 0 ... 100 ] *)
    bUpperLimitSwitch       : BOOL;
    bLowerLimitSwitch       : BOOL;
    eState      : E_CTRL_STATE;
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

fMotorPosition : Simulated motor position in the range [fMovingRangeMin ... fMovingRangeMax].

fMotorState : Simulated motor position in the range [0 … 100.0].

bUpperLimitSwitch : Simulated limit switch at the actuator's positive stop.

bLowerLimitSwitch : Simulated limit switch at the actuator's negative stop.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_SERVO_MOTOR_SIMULATION_PARAMS;
END_VAR
```

stParams : Parameter structure of the function block. This consists of the following elements:

```
TYPE
ST_CTRL_SERVO_MOTOR_SIMULATION_PARAMS:
STRUCT
    tCtrlCycleTime    : TIME  := T#0ms;     (* controller
cycle time [TIME] *)
    tTaskCycleTime    : TIME  := T#0ms;     (* task cycle
time [TIME] *)
    fMovingRangeMin       : FLOAT := 0;     (* min position
in the moving range, e.g. 0° *)
    fMovingRangeMax       : FLOAT := 0;     (* max position
in the moving range, e.g. 360° *)
    tMovingTime       : TIME := T#0ms;      (* time to move
from the min to the max *)
    tDeadTime         : TIME := T#0ms;      (* wait dead
time before starting the movement *)
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

fMovingRangeMin : Minimum position of the simulated actuator.

fMovingRangeMax : Maximum position of the simulated actuator.

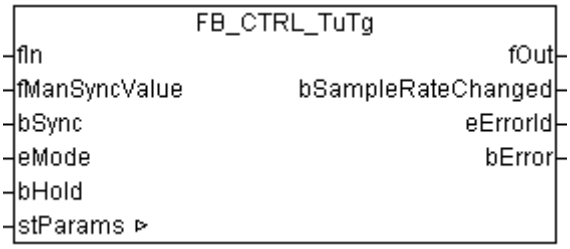
tMovingTime : The time required to move the simulated actuator from one stop to the other.

tDeadTime : Dead time of the simulated actuator.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib

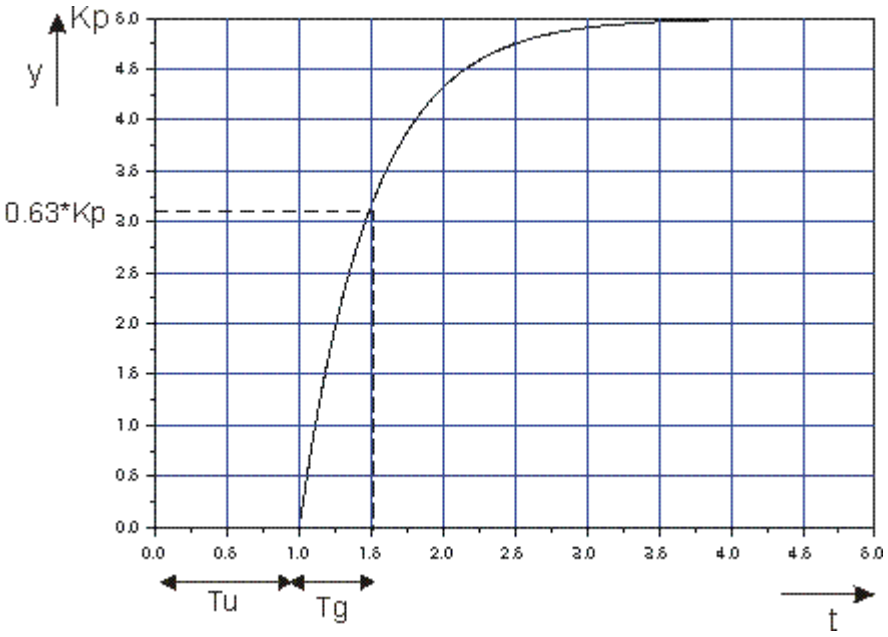
5.5.15 FB_CTRL_TuTg



The function block provides a TuTg transfer element (a dead time delay element) in the functional diagram.

Transfer function:

$$G(s) = K_p \frac{1}{1 + T_g s} \cdot e^{-T_u s}$$



VAR_INPUT

```
VAR_INPUT
    fIn         : FLOAT;
    fManSyncValue    : FLOAT;
    bSync       : BOOL;
    eMode       : E_CTRL_MODE;
    bHold       : BOOL;
END_VAR
```

fIn : Input value of the TuTg element.

fManSyncValue : Input value to which the TuTg element can be set, or that is issued at the output in manual mode.

bSync : A rising edge at this input sets the TuTg element to the value fManSyncValue.

eMode : Input that specifies the block's operating mode [▶ 16].

bHold : A TRUE at this input will hold the internal state (and therefore also the output) constant at its current value, independently of the input value.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut         : FLOAT;
    bSampleRateChanged       : BOOL;
    eState       : E_CTRL_STATE;
    eErrorId         : E_CTRL_ERRORCODES;
    bError       : BOOL;
END_VAR
```

fOut : Output of the TuTg element.

bSmpleRateChanged : Output that indicates whether the block has internally reduced the sampling rate because of the array being used to delay the input signal not otherwise providing sufficient room.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_TuTg_PARAMS;
END_VAR
```

stParams : Parameter structure of the TuTg element. This consists of the following elements:

```
TYPE
ST_CTRL_TuTg_PARAMS :
STRUCT
    tCtrlCycleTime  : TIME     := T#0ms; (* controller cycle
time [TIME] *)
    tTaskCycleTime  : TIME     := T#0ms; (* task cycle time
[TIME] *)
    fKp         : FLOAT    := 0;     (* proportional gain
*)
    tTu         : TIME     := T#0ms; (* dead time *)
    tTg         : TIME     := T#0ms; (* delay time
*)END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

fKp : Controller amplification / transfer coefficient

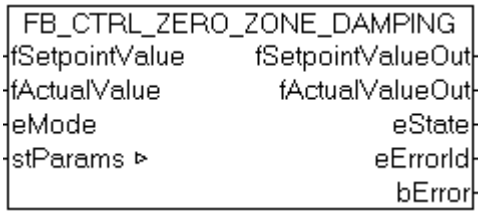
tTu : Dead time

tTg : Time constant

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.5.16 FB_CTRL_ZERO_ZONE_DAMPING

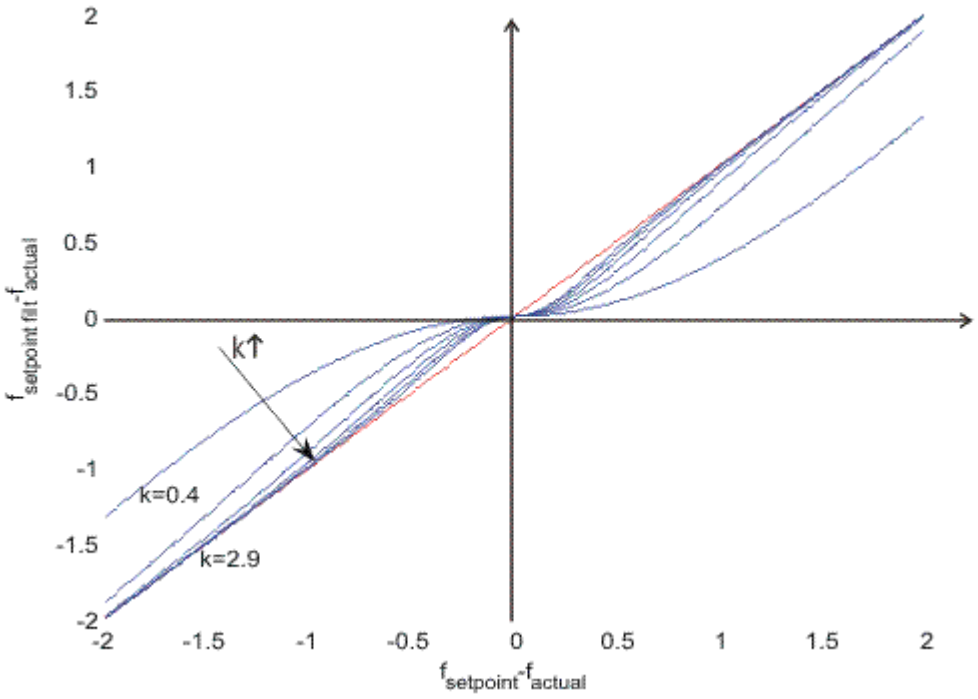


This function block enables zero point damping to be realized in order to minimize control interventions in the range | actual value - set value | < e.

Transfer behavior in the time domain:

$$f_{setpoint\_out} = (f_{setpoint\_in} - f_{actual\_in}) \cdot \tanh(|f_{setpoint\_in} - f_{actual\_in}| \cdot k_{damping}) + f_{actual\_in}$$

$$f_{actual\_out} = f_{setpoint\_in}$$



VAR_INPUT

```
VAR_INPUT
    fSetpointValue  : FLOAT;
    fActualValue    : FLOAT;
    eMode       : E_CTRL_MODE;
END_VAR
```

fSetpointValue : Set value of the controlled variable.

fActualValue : Actual value of the controlled variable.

eMode : Input that specifies the block's operating mode [▶ 16].

VAR_OUTPUT

```
VAR_OUTPUT
    fSetpointValueOut : FLOAT;
    fActualValueOut     : FLOAT;
    eState      : E_CTRL_STATE;
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

fSetpointValueOut : filtered set value to controller.

fActualValueOut : actual value to controller.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_ZERO_ZONE_DAMPING_PARAMS;
END_VAR
```

stParams : Parameter structure of the transmission element. This consists of the following elements:

```
TYPE
ST_CTRL_PI_PST_CTRL_ZERO_ZONE_DAMPING_PARAMSARAMS :
STRUCT
```

```
    tCtrlCycleTime      : TIME      :=  T#0ms; (* controller
cycle time [TIME] *)
    tTaskCycleTime      : TIME      :=  T#0ms; (* task cycle time
[TIME] *)
    fDampingCoefficient: FLOAT     :=  0.0; (*damping
coefficient *)
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

fDampingCoefficient : The parameter corresponds to $k_{damping}$ in the transfer function.

Development environment	Target platform	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

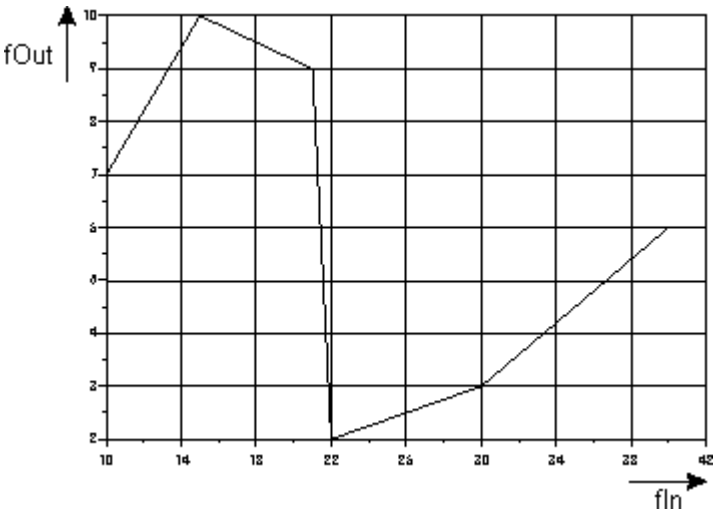
5.6 Interpolation

5.6.1 FB_CTRL_LIN_INTERPOLATION

This block performs linear interpolation to obtain values on the basis of a table of interpolation points.

Behaviour of the output:

fIn	fOut
arrTable[1,1] := 10;	arrTable[1,2] := 7;
arrTable[2,1] := 15;	arrTable[2,2] := 10;
arrTable[3,1] := 21;	arrTable[3,2] := 9;
arrTable[4,1] := 22;	arrTable[4,2] := 2;
arrTable[5,1] := 30;	arrTable[5,2] := 3;
arrTable[6,1] := 40;	arrTable[6,2] := 6;



VAR_INPUT

```
VAR_INPUT
    fIn         : FLOAT;
    fManValue       : FLOAT;
    bExtrapolate    : BOOL;
    eMode       : E_CTRL_MODE;
END_VAR
```

fIn : Input variable for the interpolation block.

bManValue : Input variable whose value is output in manual mode.

bExtrapolate : If this input is FALSE, then the value of the last interpolation point is output if the limits of the table are exceeded. If, however, it is TRUE, then extrapolation is performed on the basis of the last two interpolation points.

eMode : Input that specifies the block's operating mode [▶ 16].

VAR_OUTPUT

```
VAR_OUTPUT
    fOut            : FLOAT;
    bInIsGreaterThanMaxElement      : BOOL;
    bInIsLessThanMinElement     : BOOL;
    eState          : E_CTRL_STATE;
    eErrorId            : E_CTRL_ERRORCODES;
    bError          : BOOL;
END_VAR
```

fOut : Table value from linear interpolation.

bInIsGreaterThanMaxElement : A TRUE at this output indicates that the magnitude of the input is greater than the largest interpolation point.

bInIsLessThanMinElement : A TRUE at this output indicates that the magnitude of the input is smaller than the smallest interpolation point.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_LIN_INTERPOLATION_PARAMS;
END_VAR
```

stParams : Parameter structure of the interpolation element. This consists of the following elements:

```
TYPE
ST_CTRL_2POINT_PARAMS :
STRUCT
    tCtrlCycleTime          : TIME      := T#0ms; (*
controller cycle time [TIME] *)
    tTaskCycleTime          : TIME      := T#0ms; (* task
cycle time [TIME] *)
    pDataTable_ADR          : POINTER TO FLOAT := 0;
    nDataTable_SIZEOF       : UINT          := 0;
    nDataTable_NumberOfRows     : UINT          := 0;
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

pDataTable_ADR : Address of the n x 2 array on which linear interpolation is to be carried out

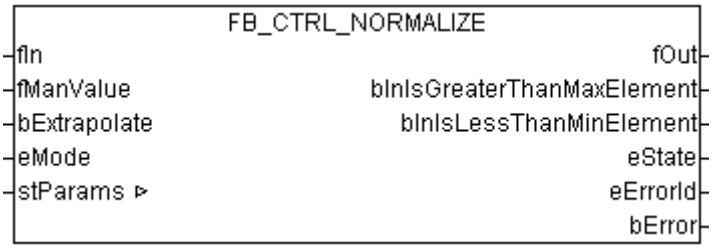
pDataTable_SIZEOF : Size of the n x 2 array.

pDataTable_NumberOfRows : Number of rows of the array.

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.6.2 FB_CTRL_NORMALIZE

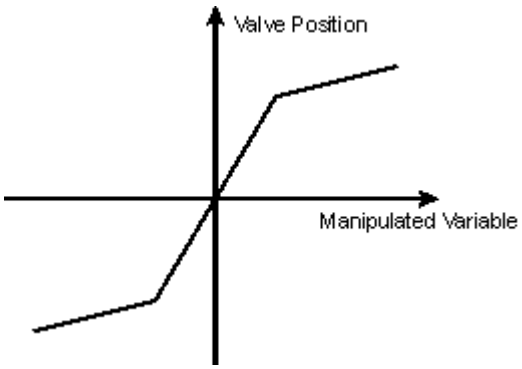


This function block can be used to linearise a non-linear transfer element, with the aid of an inverse characteristic curve.

The characteristic curve of the transfer element that is to be linearised is stored in the table associated with this block. The function block uses this to calculate the inverse characteristic curve with which the linearisation can be carried out.

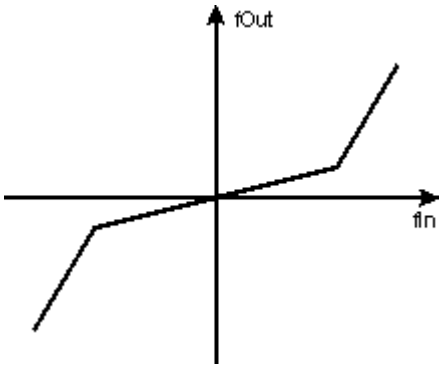
Sample:

The following valve characteristic curve is stored in the table with 4 interpolation points.

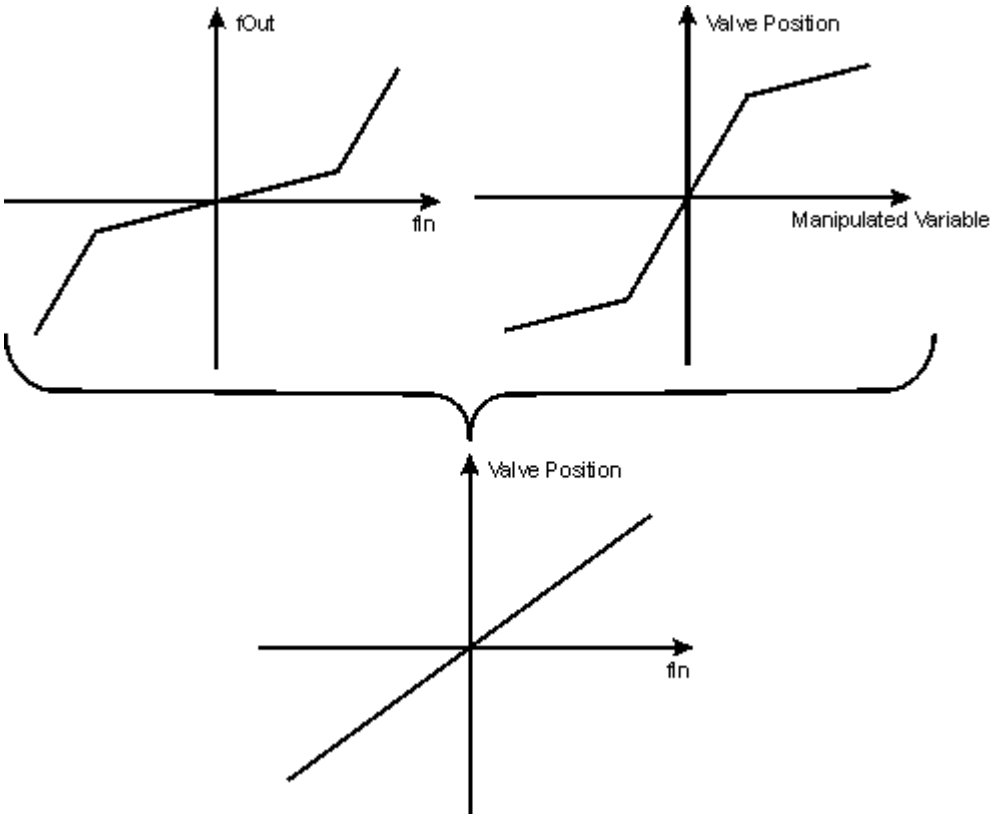


Control value	Valve position
arrTable[1,1] := -6;	arrTable[1,2] := -100;
arrTable[2,1] := -1;	arrTable[2,2] := -70;
arrTable[3,1] := 1;	arrTable[3,2] := 70;
arrTable[4,1] := 6;	arrTable[4,2] := 100;

The inverse characteristic curve is calculated from this characteristic curve:



In the ideal case, applying these two characteristic curves in series will result in a linear transfer behavior.



VAR_INPUT

```
VAR_INPUT
    fIn         : FLOAT;
    fManValue       : FLOAT;
    bExtrapolate    : BOOL;
    eMode       : E_CTRL_MODE;
END_VAR
```

fIn : Input value.

bManValue : Input variable whose value is output in manual mode.

bExtrapolate : If this input is FALSE, then the value of the last interpolation point is output if the limits of the table are exceeded. If, however, it is TRUE, then extrapolation is performed on the basis of the last two interpolation points.

eMode : Input that specifies the block's operating mode [▶ 16].

VAR_OUTPUT

```
VAR_OUTPUT
    fOut            : FLOAT;
    bInIsGreaterThanMaxElement      : BOOL;
    bInIsLessThanMinElement     : BOOL;
    eState          : E_CTRL_STATE;
    eErrorId            : E_CTRL_ERRORCODES;
    bError          : BOOL;
END_VAR
```

fOut : Table value from linear interpolation.

bInIsGreaterThanMaxElement : A TRUE at this output indicates that the magnitude of the input is greater than the largest interpolation point.

bInIsLessThanMinElement : A TRUE at this output indicates that the magnitude of the input is smaller than the smallest interpolation point.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_NORMALIZE_PARAMS;
END_VAR
```

stParams : Parameter structure of the function block. This consists of the following elements:

```
TYPE
ST_CTRL_NORMALIZE_PARAMS:
STRUCT
    tCtrlCycleTime          : TIME      := T#0ms; (*
controller cycle time [TIME] *)
    tTaskCycleTime          : TIME      := T#0ms; (* task
cycle time [TIME] *)
    pDataTable_ADR          : POINTER TO FLOAT := 0;
    nDataTable_SIZEOF       : UINT          := 0;
    nDataTable_NumberOfRows     : UINT          := 0;
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

pDataTable_ADR : Address of the n x 2 array on which linear interpolation is to be carried out

pDataTable_SIZEOF : Size of the n x 2 array.

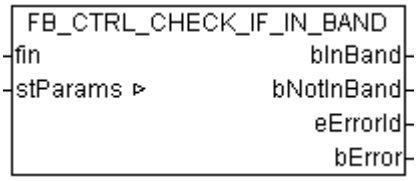
pDataTable_NumberOfRows : Number of rows of the array.

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.7 Monitoring / Alarming

5.7.1 FB_CTRL_CHECK_IF_IN_BAND



This function block monitors whether the input value is within the range [fMin ... fMax], i.e. whether the inequality

$$fMin \leq fIn \leq fMax$$

is satisfied.

VAR_INPUT

```
VAR_INPUT
    fIn         : FLOAT;
END_VAR
```

fIn : The input variable to be monitored.

VAR_OUTPUT

```
VAR_OUTPUT
    bInBand         : BOOL;
    bNotInBand      : BOOL;
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

bInBand : A TRUE at this output indicates that the input value is within the specified range.

bNotInBand : A TRUE at this output indicates that the input value is **not** within the specified range.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_CHECK_IF_IN_BAND_PARAMS;
END_VAR
```

stParams : Parameter structure of the function block. This consists of the following elements:

```
TYPE ST_CTRL_CHECK_IF_IN_BAND_PARAMS:
STRUCT
    tCtrlCycleTime     : TIME     := T#0ms; (* controller cycle
time [TIME] *)
    tTaskCycleTime     : TIME     := T#0ms; (* task cycle time
[TIME] *)
    fMin           : FLOAT;
    fMax           : FLOAT;
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input variable to calculate internally whether the state and the output variables have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the block is called in every cycle this corresponds to the task cycle time of the calling task.

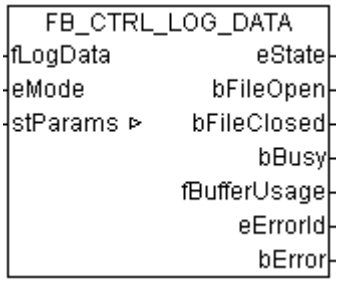
fMin : Lower limit of the range.

fMax : Upper limit of the range.

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.7.2 FB_CTRL_LOG_DATA (only on a PC system)



This function block allows a log file to be created in *.csv format (comma separated values), in which a maximum of 10 variables may be recorded. The column titles specified by the user are written in the first row of this file. The input data is written at equal time intervals in the following rows. The individual entries are separated by a comma. The time interval between the entries is specified in the **tLogCycleTime** parameter. If, for instance, tLogCycleTime := T\#2s is chosen, then an entry is made in the file every 2 s. The files that were generated can be analyzed with a spreadsheet program, for example.

The timestamp of the log entry, in s, is stored in the first column of the file. The other columns contain the data of the function block input **fLogData**.

When the mode is set to **eCTRL_MODE_ACTIVE** the log file is opened and entries are written into the file. The file remains open until the function block's mode is set to eCTRL_MODE_PASSIVE. It is essential that the file is closed by switching into **eCTRL_MODE_PASSIVE** before attempting to analyze the log file. If this is not done, it is possible that not all the entries will be written into the file.

The function block makes it possible to work with or without an external buffer. The external buffer is used if a buffer address and a buffer size different from zero are parameterized. In the absence of an external buffer, an internal buffer with the size of 255 bytes is used.

Operating without an external buffer:	The bBusy output is TRUE when the logging of a row has been started. The following data set will not be logged until the bBusy output is FALSE again. The **fBufferUsage** output indicates how full the internal buffer is.

Operating with an external buffer:	The user must create a buffer of the type *ARRAY OF BYTES* larger than 255 bytes. The individual messages are temporarily stored in the external buffer and this buffer is written to the file as soon as possible. The **fBufferUsage** output indicates how full the buffer is. The function block is stopped and an error is output if the buffer overflows.

VAR_INPUT

```
VAR_INPUT
    fLogData    : T_CTRL_LOGGER_DATA;
    eMode   : E_CTRL_MODE;
END_VAR
```

```
VAR_GLOBAL CONSTANT
    nCTRL_LOGGER_DATA_ARRAY_SIZE     :UINT := 10;
END_VAR
```

```
TYPE
    T_CTRL_LOGGER_DATA     :ARRAY [1..nCTRL_LOGGER_DATA_ARRAY_SIZE]
OF FLOAT;
END_TYPE
```

fLogData : Array containing the values that are to be written into the log file.

eMode : Input that specifies the block's operating mode [▶ 16].

VAR_OUTPUT

```
VAR_OUTPUT
    eState      : E_CTRL_STATE;
    bFileOpen       : BOOL
    bFileClosed     : BOOL
    fBufferUsage    : FLOAT     (* Buffer fill level in percent [0 ... 100] *)
    bBusy       : BOOL
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

eState : State of the function block.

bFileOpen : A TRUE at this output indicates that the file has successfully been opened.

bFileClosed : A TRUE at this output indicates that the file has successfully been closed.

fBufferUsage : Current fill level of the used buffer in percent.

bBusy : A TRUE at this output indicates that logging a row is active.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_LOG_DATA_PARAMS;
END_VAR
```

stParams : Parameter structure of the logging block. This consists of the following elements:

```
TYPE ST_CTRL_LOG_DATA_PARAMS
:
STRUCT

tLogCycleTime        : TIME := T#0ms; (* controller cycle time
[TIME] *)

tTaskCycleTime       : TIME := T#0ms; (* task cycle time [TIME]
*)


sFileName        : STRING;


sNetId           : T_AmsNetId := ''; (* ams net id
*)
```

```
tFileOperationTimeout    : TIME := T#3s; (* file operation timeout
*)


nNumberOfColumns     : INT(1..10);


arColumnHeadings     : ARRAY [1..10] OF STRING;


bAppendData          : BOOL := FALSE;


bWriteTimeStamps     : BOOL := TRUE;

bWriteColumnHeadings     : BOOL := TRUE;

bWriteAbsoluteTimeStamps : BOOL := FALSE; (* Set to true if the
NT-time form the local machine should logged in the first column.
*)

(* The log cycle time must than be greater or equal T#5s!!!
*)


pLogBuffer_ADR       : POINTER TO BYTE;

nLogBuffer_SIZEOF    : UDINT;
END_STRUCT
END_TYPE
```

tLogCycleTime : Cycle time with which entries are written into the log file. This must be greater than or equal to the TaskCycleTime.

tTaskCycleTime : Cycle time with which the function block is called. If the block is called in every cycle this corresponds to the task cycle time of the calling task.

sFileName : Name and path of the log file, e.g.: d:\Logfile.csv.

sNetId : The file is written on the system with this net id.

tFileOperationTimeout : Timeout for all file operations.

nNumberOfColums: Number of columns written into the file (maximum 10).

arColumnHeadings: Array of strings that contain the column headings.

bAppendData : If this parameter is TRUE, then new data sets are appended when a file is opened again. Otherwise the file is overwritten **without query**, and this will delete any content that already exists.

bWriteTimeStamps : If this parameter is set to TRUE, the time stamp of the measurement is written into the first column of the file.

bWriteColumnHeadings : If this parameter is set to TRUE, the column headers are written into the first row of the file.

bWriteAsoulteTimeStamps : If true, the NT-time from the local system is used as timestamp. **In this case, the minimum log cycle time is 5s!**

pLogBuffer_ADR : Address of the external LogBuffer. To be recognised, the buffer's address must be unequal to 0.

nLogBuffer_SIZEOF : Size of the LogBuffer. The buffer must be an ARRAY OF BYTE with at least 256 elements. The size of the buffer can be optimised with the aid of the fBufferUsage output.

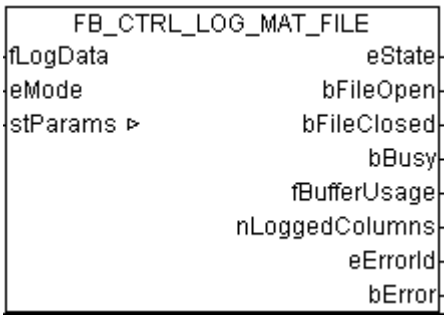
NOTE

The parameter set can only be changed when the file is closed (bFileClosed = TRUE)! Otherwise, errors can occur during file handling!

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib

5.7.3 FB_CTRL_LOG_MAT_FILE (only on a PC system)



This function block allows a log file to be created in Matlab 5 (*.mat) format, in which a maximum of 10 magnitudes may be recorded.

Two variables are created in the file, a double array and a cell array. The recorded magnitudes are recorded, line-by-line, in the double array. The identifiers of the individual rows are stored in the cell array. The user can specify the name used for the double array in the function block's parameter structure. The name of the cell array is derived from the name of the double array by appending "_Info" to the variable name.

The input data is written at equal time intervals in the columns of the data array. A time stamp for the relevant entry, in s, can be stored in the first column. The time interval between the entries is specified in the **tLogCycleTime** parameter. If, for instance, tLogCycleTime := T\#2s is chosen, then an entry is made in the file every 2 s.

When the mode is set to **eCTRL_MODE_ACTIVE** the log file is opened and entries are written into the file. The file remains open until the function block's mode is set to eCTRL_MODE_PASSIVE. It is essential that the file is closed by switching into **eCTRL_MODE_PASSIVE** before attempting to analyze the log file. If this is not done, it is possible that not all the entries will be written into the file, which will then not be consistent.

The function block makes it possible to work with or without an external buffer. The external buffer is used if a buffer address and a buffer size greater than zero are parameterized. In the absence of an external buffer, an internal buffer with the size of 1500 bytes is used.

Operating without an external buffer:	The bBusy output is TRUE when the logging of a row has been started. The following data set will not be logged until the bBusy output is FALSE again. The **fBufferUsage** output indicates how full the internal buffer is.
Operating with an external buffer:	The user must create a buffer of the type *ARRAY OF BYTES*, which is larger than 1500 bytes. The individual messages are temporarily stored in the external buffer and this buffer is written to the file as soon as possible. The **fBufferUsage** output indicates how full the buffer is. The function block is stopped and an error is output if the buffer overflows.

VAR_INPUT

```
VAR_INPUT
    fLogData    : T_CTRL_LOGGER_DATA;
    eMode   : E_CTRL_MODE;
END_VAR
```

```
VAR_GLOBAL CONSTANT
    nCTRL_LOGGER_DATA_ARRAY_SIZE     :UINT := 10;
END_VAR
```

```
TYPE
    T_CTRL_LOGGER_DATA     :ARRAY [1..nCTRL_LOGGER_DATA_ARRAY_SIZE]
OF FLOAT;
END_TYPE
```

fLogData : Array containing the values that are to be written into the log file.

eMode : Input that specifies the block's operating mode [▶ 16].

VAR_OUTPUT

```
VAR_OUTPUT
    eState      : E_CTRL_STATE;
    bFileOpen       : BOOL
    bFileClosed     : BOOL
    fBufferUsage    : FLOAT     (* Buffer fill level in percent [0 ... 100] *)
    nLoggedColumns  : DINT;
    bBusy       : BOOL
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

eState : State of the function block.

bFileOpen : A TRUE at this output indicates that the file has successfully been opened.

bFileClosed : A TRUE at this output indicates that the file has successfully been closed.

fBufferUsage : Current fill level of the external buffer as a percentage.

nLoggedColumns : Number of columns written in the file.

bBusy : A TRUE at this output indicates that logging a row is active.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_LOG_MAT_FILE_PARAMS;
END_VAR
```

stParams : Parameter structure of the logging function block. This consists of the following elements:

```
TYPE
ST_CTRL_LOG_MAT_FILE_PARAMS:
STRUCT
    tLogCycleTime    : TIME := T#0ms; (* controller cycle
time [TIME] *)
    tTaskCycleTime       : TIME := T#0ms; (* task cycle time
[TIME] *)
    sFileName        : STRING;
    nNumberOfRows    : INT(1..10);
    sMatrixName      : STRING;
    arRowDescription     : ARRAY [1..10] OF STRING(25);
    bWriteTimeStamps     : BOOL := TRUE;
    bWriteRowDescription : BOOL := TRUE;
    pLogBuffer_ADR       : POINTER TO
ST_CTRL_MULTIPLE_PWM_OUT_DATA;
    nLogBuffer_SIZEOF    : UDINT;
END_STRUCT
END_TYPE
```

tLogCycleTime : Cycle time with which entries are written into the log file. This must be greater than or equal to the TaskCycleTime.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

sFileName : Name and path of the log file, e.g.: d:\Logfile.mat.

nNumberOfRows : Number of columns written into the file (maximum 10).

sMatrixName : Name of the data array.

arRowDescription : Array of strings that contain the line headings.

bWriteTimeStamps : If this parameter is set to TRUE, the time stamp of the measurement is written into the first row of the data array.

bWriteRowDescription : If this parameter is set to TRUE a cell array is created into which the descriptions of the rows are written.

pLogBuffer_ADR : Address of the external LogBuffer. To be recognized, the buffer's address must be unequal to 0.

nLogBuffer_SIZEOF: Size of the LogBuffer. The buffer must be an ARRAY OF BYTE with at least 1501 elements. The size of the buffer can be optimized with the aid of the fBufferUsage output.

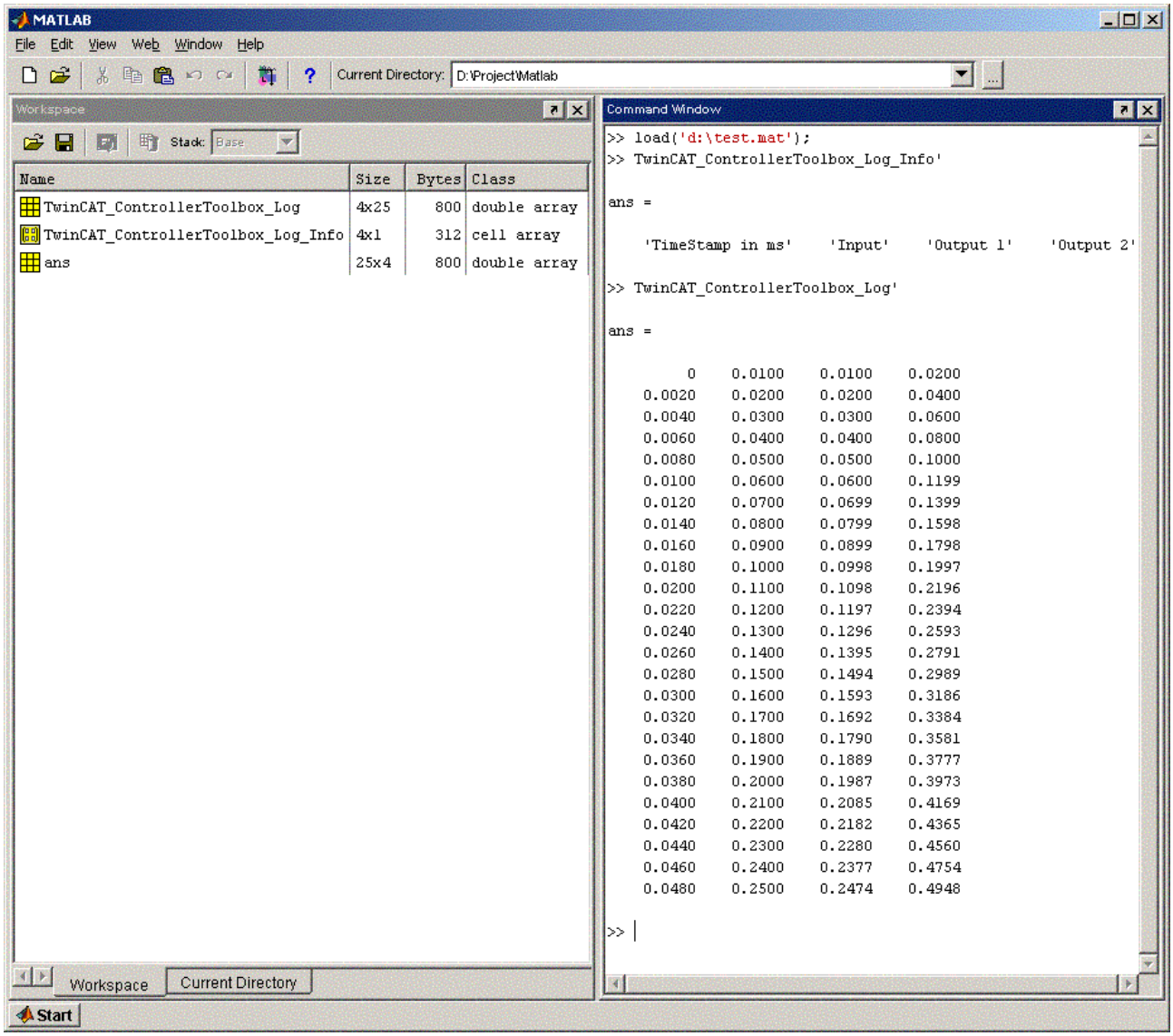
NOTE

The parameter set can only be changed when the file is closed (bFileClosed = TRUE)! Otherwise errors can occur during file handling!

Sample

```
PROGRAM PRG_LOG_MAT_FILE_TEST_BUFFERED
VAR
    eMode       : E_CTRL_MODE;
    stParams        : ST_CTRL_LOG_MAT_FILE_PARAMS;
    LoggerData      : T_CTRL_LOGGER_DATA;

    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;

    fbCtrlLogMatFile    : FB_CTRL_LOG_MAT_FILE;
    LogBuffer       : ARRAY[0..2000] OF BYTE;

    bInit       : BOOL := TRUE;

    fIn         : LREAL;
    fOut        : LREAL;

    fMaxBufferUsage     : LREAL;
END_VAR

IF bInit
THEN
    stCtrl_GLOBAL_CycleTimeInterpretation.bInterpretCycleTimeAsTicks := FALSE;
    stCtrl_GLOBAL_CycleTimeInterpretation.fBaseTime := 0;

    stParams.tLogCycleTime      := T#2ms;
    stParams.tTaskCycleTime         := T#2ms;
    stParams.nNumberOfRows      := 3;
    stParams.sFileName          := 'D:\test.mat';
    stParams.sMatrixName        := 'TwinCAT_ControllerToolbox_Log';

    stParams.arRowDescription[1]    := 'Input';
    stParams.arRowDescription[2]    := 'Output 1';
    stParams.arRowDescription[3]    := 'Output 2';

    stParams.bWriteRowDescription   := TRUE;
    stParams.bWriteTimeStamps           := TRUE;

    eMode   := eCTRL_MODE_ACTIVE;
    bInit   := FALSE;
END_IF

stParams.nLogBuffer_SIZEOF      := SIZEOF( LogBuffer );
stParams.pLogBuffer_ADR := ADR( LogBuffer );

(* creat test data *)
  fIn   := fIn + 0.01;
  fOut  := SIN(fIn);

(* copy test data to the Logger-Input *)
  LoggerData[ 1 ] := fIn;
  LoggerData[ 2 ] := fOut;
  LoggerData[ 3 ] := fOut * 2;

(* log only 1000 columns *)
  IF fbCtrlLogMatFile.nLoggedColumns >= 25
  THEN
    eMode := eCTRL_MODE_Passive;
```

```
  END_IF

(* call logger *)
  fbCtrlLogMatFile( fLogData := LoggerData,
          eMode := eMode,
          stParams :=stParams,
          eErrorId => eErrorId,
          bError => bError);

(* get max buffer usage *)
  fMaxBufferUsage := MAX(fbCtrlLogMatFile.fBufferUsage, fMaxBufferUsage);
```

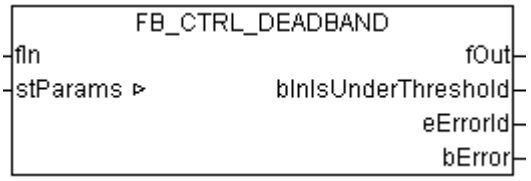


Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib

5.8 Output To Controlling Equipment

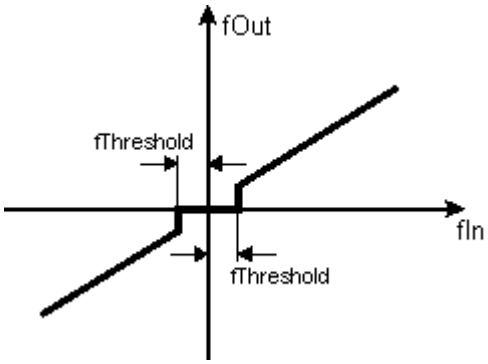
5.8.1 FB_CTRL_DEADBAND



This block provides a dead band for the input signal. If the input signal is within the dead band, this is indicated by the **bInIsUnderThreshold** output.

Description of the output behaviour:

$$f_{out} = \begin{cases} 0.0 & : \quad |f_{in}| \leq fThreshold \\ f_{in} & : \quad else \end{cases}$$



VAR_INPUT

```
VAR_INPUT
    fIn         : FLOAT;
END_VAR
```

fIn : Input value.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut            : FLOAT;
    bInIsUnderThreshold     : BOOL;
    eState          : E_CTRL_STATE;
    eErrorId            : E_CTRL_ERRORCODES;
    bError          : BOOL;
END_VAR
```

fOut : Output of the function block.

bInIsUnderThreshold : A TRUE at this output indicates that the input value is within the dead band.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_DEADBAND_PARAMS;
END_VAR
```

stParams : Parameter structure of the function block. This consists of the following elements:

```
TYPE
ST_CTRL_DEADBAND_PARAMS:
STRUCT
    tCtrlCycleTime          : TIME     := T#0ms; (*
controller cycle time [TIME] *)
    tTaskCycleTime          : TIME     := T#0ms; (* task
cycle time [TIME] *)
    fThreshold          : FLOAT;
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

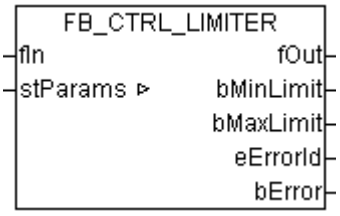
tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

fThreshold : The function block's dead band, see image.

Requirements

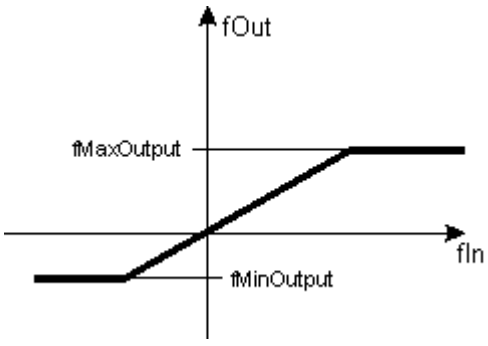
Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.8.2 FB_CTRL_LIMITER



This block limits an input signal to a parameterisable interval.

Description of the output behaviour:



VAR_INPUT

```
VAR_INPUT
    fIn         : FLOAT;
END_VAR
```

fIn : Input value for the function block.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut        : FLOAT;
    bMinLimit       : BOOL;
    bMaxLimit       : BOOL;
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

fOut : Output of the function block.

bMinLimit : A TRUE at this output indicates that the output has reached the lower limit.

bMaxLimit : A TRUE at this output indicates that the output has reached the upper limit.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_LIMITER_PARAMS;
END_VAR
```

stParams : Parameter structure of the function block. This consists of the following elements:

```
TYPE
ST_CTRL_LIMITER_PARAMS:
STRUCT
    tCtrlCycleTime         : TIME     := T#0ms; (*
controller cycle time [TIME] *)
    tTaskCycleTime         : TIME     := T#0ms; (* task
cycle time [TIME] *)
    fMinOutput          : FLOAT;
    fMaxOutput          : FLOAT;
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

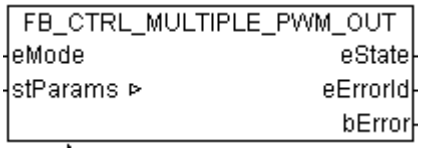
fMinOutput : Lower limit at which the output is limited.

fMaxOutput : Upper limit at which the output is limited.

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

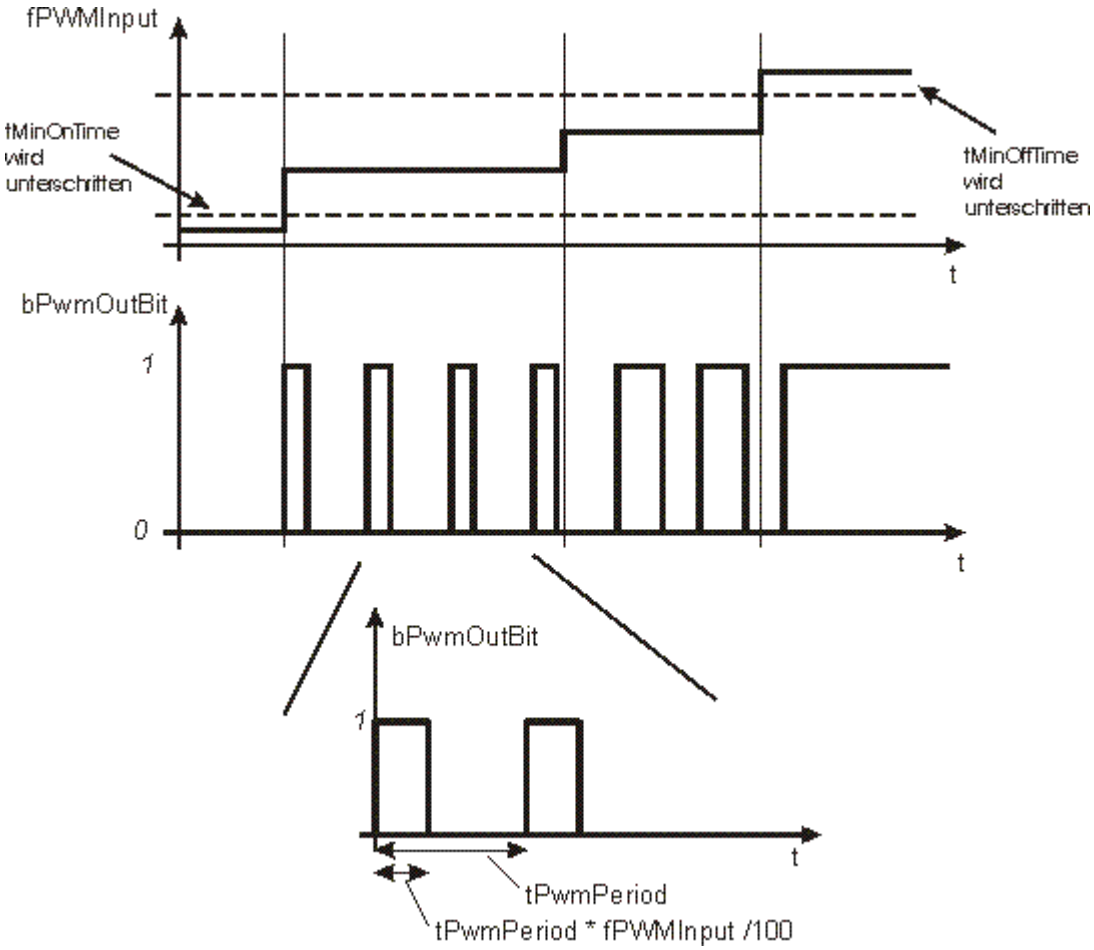
5.8.3 FB_CTRL_MULTIPLE_PWM_OUT



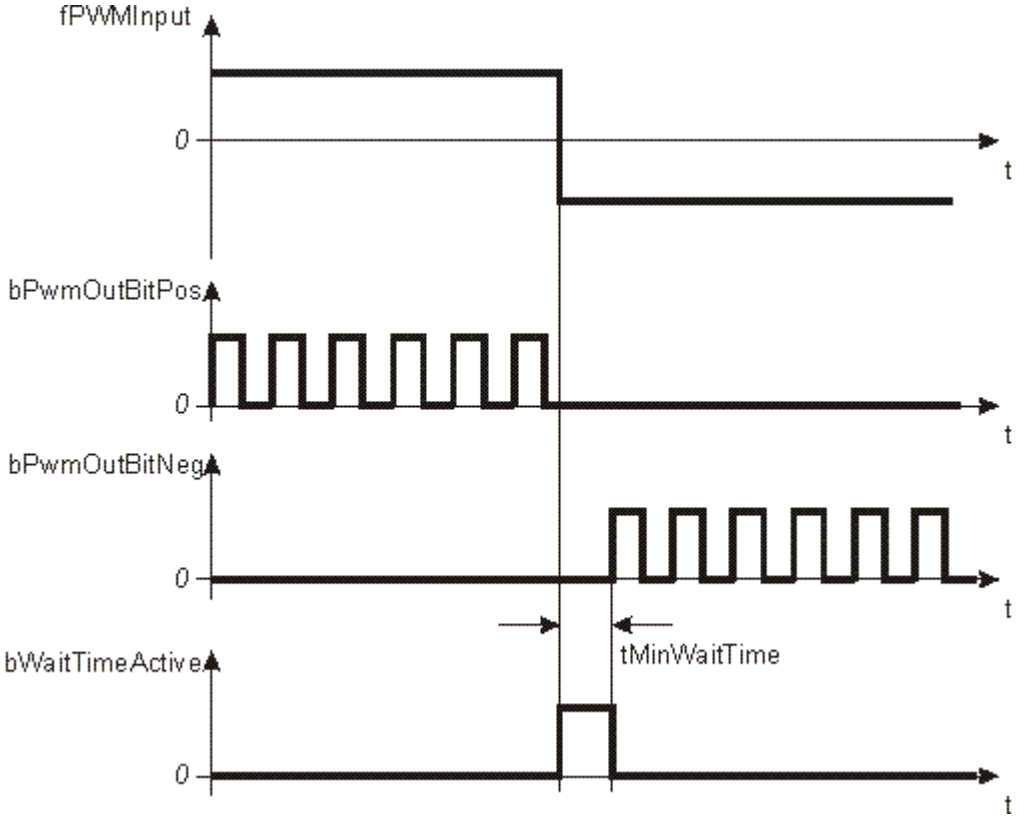
This block creates PWM modulated output signals from a number of input signals in such a way that the temporal relationships between the output signals are arranged so that as few outputs as possible are switched on at any one time. This temporal arrangement reduces the maximum power necessary required for the actuators.

Both the minimum switch on time and the minimum switch off time can be parameterised, in addition to the mark-to-space ratio, in this extended block.

Description of the output behaviour (1):



Description of the output behavior (2):



The programmer must create the following arrays in the PLC if this function block is to be used:

```
    ar_fPwmInput          :
ARRAY[1..nNumberOfPwmOutputs] OF FLOAT;
    ar_stWaitTimesConfig     : ARRAY[1..nNumberOfPwmOutputs] OF
ST_CTRL_MULTIPLE_PWM_OUT_TIMES;
    ar_stPwmOutputs       : ARRAY[1..nNumberOfPwmOutputs] OF
ST_CTRL_MULTIPLE_PWM_OUT_OUTPUTS;
    ar_stPwmDataVars      : ARRAY[1..nNumberOfPwmOutputs] OF
ST_CTRL_MULTIPLE_PWM_OUT_DATA;
```

The input values for the individual channels of the PWM function block are written into the ar_fPwmInput array. The programmer can specify the parameterizable times for the individual channels in the ar_stWaitTimesConfig array. The function block writes the output bits into the ar_stPwmOutputs array. The internal data required by the function block is stored in the ar_stPwmDataVars array. Under no circumstances the structures contained in the last-named array may not be changed in the PLC program. This procedure is also illustrated in the sample program listed below.

VAR_INPUT

```
VAR_INPUT
    eMode        : E_CTRL_MODE;
END_VAR
```

eMode : Input that specifies the block's operating mode.

VAR_OUTPUT

```
VAR_OUTPUT
    eState       : E_CTRL_STATE;
    bError       : BOOL;
    eErrorId        : E_CTRL_ERRORCODES;
END_VAR
```

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_PWM_OUT_EXT_PARAMS;
END_VAR
```

stParams : Parameter structure of the PWM element. This consists of the following elements:

```
TYPE
ST_CTRL_MULTIPLE_PWM_OUT_PARAMS :
STRUCT
    tTaskCycleTime       : TIME; (* PLC/PWM cycle time in
seconds *)
    tPWMPeriod           : TIME; (* controller cycle time
in seconds *)
    nNumberOfPwmOutputs      : USINT;
    pPwmInputArray_ADR       : POINTER TO FLOAT := 0;
    nPwmInputArray_SIZEOF    : UINT;
    pPwmTimesConfig_ADR      : POINTER TO
ST_CTRL_MULTIPLE_PWM_OUT_TIMES;
    nPwmTimesConfig_SIZEOF   : UINT;
    pPwmOutputArray_ADR      : POINTER TO
ST_CTRL_MULTIPLE_PWM_OUT_OUTPUTS;
    nPwmOutputArray_SIZEOF   : UINT;
    pPwmData_ADR         : POINTER TO
ST_CTRL_MULTIPLE_PWM_OUT_DATA;
    nPwmData_SIZEOF      : UINT;
END_STRUCT
END_TYPE
```

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

tPWMPeriod : Period of the PWM signal.

nNumberOfPwmOutputs : Number of PWM outputs. [1...n]

pPwmInputArray_ADR : Address of the PWM input array.

nPwmInputArray_SIZEOF : Size of the PWM input array in bytes.

pPwmTimesConfig_ADR : Address of the PWM times array.

nPwmTimesConfig_SIZEOF : Size of the PWM times array in bytes.

pPwmData_ADR : Address of the internal PWM data array.

pPwmData_SIZEOF : Size of the internal PWM data array in bytes.

```
TYPE
ST_CTRL_MULTIPLE_PWM_OUT_TIMES :
STRUCT
    tMinOnTime     : TIME; (* min. switch on time *)
    tMinOffTime    : TIME; (* min. switch off time *)
    tMinWaitTime   : TIME; (* min. waiting time when switching from
pos to neg or vv*)
END_STRUCT
END_TYPE
```

tMinOnTime : Minimum switch-on time of the PWM output.

tMinOffTime : Minimum switch-off time of the PWM output.

tMinWaitTime : Waiting time between switching between a positive and negative output signal.

```
TYPE
ST_CTRL_MULTIPLE_PWM_OUT_OUTPUTS :
STRUCT
    bPwmOutBitPos     : BOOL;
    bPwmOutBitNeg     : BOOL;
    bWaitTimeActive   : BOOL;
END_STRUCT
END_TYPE
```

bPwmOutBitPos : PWM signal if fPwmInput > 0.0.

bPwmOutBitNeg : PWM signal if fPwmInput < 0.0.

bWaitTimeActive : A TRUE at this output indicates that the waiting time between the switching actions of the output signals is active. This output can be used to hold an I component that may be present in the prior controller constant during this time.

```
TYPE
ST_CTRL_MULTIPLE_PWM_OUT_DATA :
STRUCT
    Interne Struktur. Auf diese darf nicht schreibend
zugegriffen werden.
END_STRUCT
END_TYPE
```

Sample:

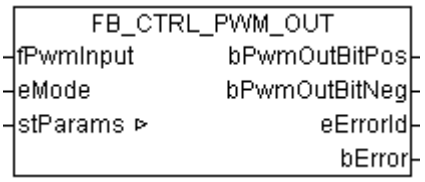
```
PROGRAM PRG_MULTIPLE_PWM_OUT_TEST
VAR CONSTANT
    nNumberOfPwmOutputs : USINT := 3;
END_VAR
VAR
    ar_fPwmInput         : ARRAY[1..nNumberOfPwmOutputs] OF
FLOAT;
    ar_stWaitTimesConfig     : ARRAY[1..nNumberOfPwmOutputs] OF
ST_CTRL_MULTIPLE_PWM_OUT_TIMES;
    ar_stPwmOutputs      : ARRAY[1..nNumberOfPwmOutputs] OF
ST_CTRL_MULTIPLE_PWM_OUT_OUTPUTS;
    ar_stPwmDataVars     : ARRAY[1..nNumberOfPwmOutputs] OF
ST_CTRL_MULTIPLE_PWM_OUT_DATA;
    eMode            : E_CTRL_MODE;
    stParams         :
ST_CTRL_MULTIPLE_PWM_OUT_PARAMS;
    eErrorId         : E_CTRL_ERRORCODES;
    bError           : BOOL;
    fbPwm_Out        : FB_CTRL_MULTIPLE_PWM_OUT;
    bInit            : BOOL := TRUE;
    fIn1             : FLOAT;
    fIn2             : FLOAT;
    fIn3             : FLOAT;
    bOut1_pos        : BOOL;
    bOut1_neg        : BOOL;
    bOut2_pos        : BOOL;
    bOut2_neg        : BOOL;
    bOut3_pos        : BOOL;
    bOut3_neg        : BOOL;
END_VAR
IF bInit
THEN
    (* set values in the local arrays *)
    ar_stWaitTimesConfig[1].tMinOnTime   := T#500ms;
    ar_stWaitTimesConfig[1].tMinOffTime  := T#300ms;
    ar_stWaitTimesConfig[1].tMinWaitTime := T#3.5s;
    ar_stWaitTimesConfig[2].tMinOnTime   := T#400ms;
    ar_stWaitTimesConfig[2].tMinOffTime  := T#250ms;
    ar_stWaitTimesConfig[2].tMinWaitTime := T#4.5s;
    ar_stWaitTimesConfig[3].tMinOnTime   := T#400ms;
    ar_stWaitTimesConfig[3].tMinOffTime  := T#200ms;
    ar_stWaitTimesConfig[3].tMinWaitTime := T#5.5s;
    (* set values in the parameter struct *)
    stParams.tTaskCycleTime     := T#2ms;
    stParams.tPWMPeriod         := T#2s;
    stParams.nNumberOfPwmOutputs    := nNumberOfPwmOutputs;
    (* set the ctrl-mode *)
    eMode := eCTRL_MODE_ACTIVE;
    bInit := FALSE;
END_IF
(* set the addresses *)
stParams.pPwmTimesConfig_ADR    := ADR(
ar_stWaitTimesConfig);
stParams.nPwmTimesConfig_SIZEOF := SIZEOF(
ar_stWaitTimesConfig);
stParams.pPwmInputArray_ADR     := ADR( ar_fPwmInput );
stParams.nPwmInputArray_SIZEOF  := SIZEOF( ar_fPwmInput );
stParams.pPwmOutputArray_ADR    := ADR( ar_stPwmOutputs );
stParams.nPwmOutputArray_SIZEOF := SIZEOF( ar_stPwmOutputs );
stParams.pPwmData_ADR       := ADR( ar_stPwmDataVars );
stParams.nPwmData_SIZEOF    := SIZEOF( ar_stPwmDataVars
);
ar_fPwmInput[1] := fIn1;
ar_fPwmInput[2] := fIn2;
ar_fPwmInput[3] := fIn3;
```

```
fbPwm_Out( eMode     := eMode,
       stParams  := stParams,
       bError    => bError,
       eErrorId  => eErrorId);
bOut1_pos := ar_stPwmOutputs[1].bPwmOutBitPos;
bOut1_neg := ar_stPwmOutputs[1].bPwmOutBitNeg;
bOut2_pos := ar_stPwmOutputs[2].bPwmOutBitPos;
bOut2_neg := ar_stPwmOutputs[2].bPwmOutBitNeg;
bOut3_pos := ar_stPwmOutputs[3].bPwmOutBitPos;
bOut3_neg := ar_stPwmOutputs[3].bPwmOutBitNeg;
```

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.8.4 FB_CTRL_PWM_OUT

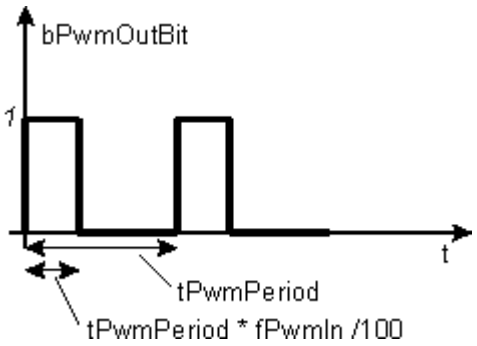


This block creates a PWM modulated signal on the basis of the input signal.

Description of the output behavior:

This function block creates a PWM signal at the outputs with a mark-to-space ratio corresponding to the **fPwmInput** input. The mark-to-space ratio is specified at the input in %; the range from -100% to 100% is available. If a positive value is specified, the pulse width modulated signal is made available at the **bPwmOutBitPos** output. If the specified mark-to-space ratio is negative, the signal is output at **bPwmOutBitNeg**. These two signals therefore make it possible to control two different actuators, depending on the arithmetic sign.

If the **bInstantPWMUpdate** is set to TRUE it is possible to adopt a new input value instantly. The new input value, in other words, is effective even in the current PWM cycle. If this parameter is FALSE, then a new input value is only adopted at the start of a new PWM cycle.



VAR_INPUT

```
VAR_INPUT
    fPwmInput       : FLOAT;        (* controller output = PMW input [-100.0 ... 100.0] *)
    eMode       : E_CTRL_MODE;
END_VAR
```

fPwmInput : Input value.

eMode : Input that specifies the block's operating mode.

VAR_OUTPUT

```
VAR_OUTPUT
    bPwmOutBitPos   : BOOL;          (* PWM output bit *)
    bPwmOutBitNeg   : BOOL;          (* PWM output bit *)
    eState      : E_CTRL_STATE;
    bError      : BOOL;
    eErrorId        : E_CTRL_ERRORCODES;
END_VAR
```

bPwmOutBitPos : PWM signal, when fPwmInput > 0.0.

bPwmOutBitNeg : PWM signal, when fPwmInput < 0.0.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_PWM_OUT_PARAMS;
END_VAR
```

stParams : Parameter structure of the PWM element. This consists of the following elements:

```
TYPE
ST_CTRL_PWM_OUT_PARAMS:
STRUCT
    tTaskCycleTime         : TIME     (* task cycle time
[TIME] *)
    tPWMPeriod         : TIME;    (* PWM period
duration [TIME] *)
    bInstantPWMUpdate      : BOOL;
END_STRUCT
END_TYPE
```

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

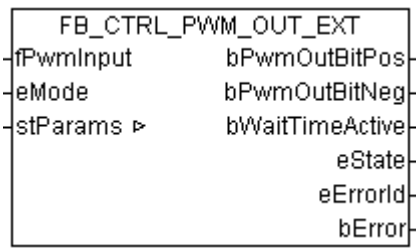
tPWMPeriod : Period of the PWM signal.

bInstantPWMUpdate : If this flag is TRUE, then a new input value is immediately adopted, even in the present PWM cycle.

Requirements

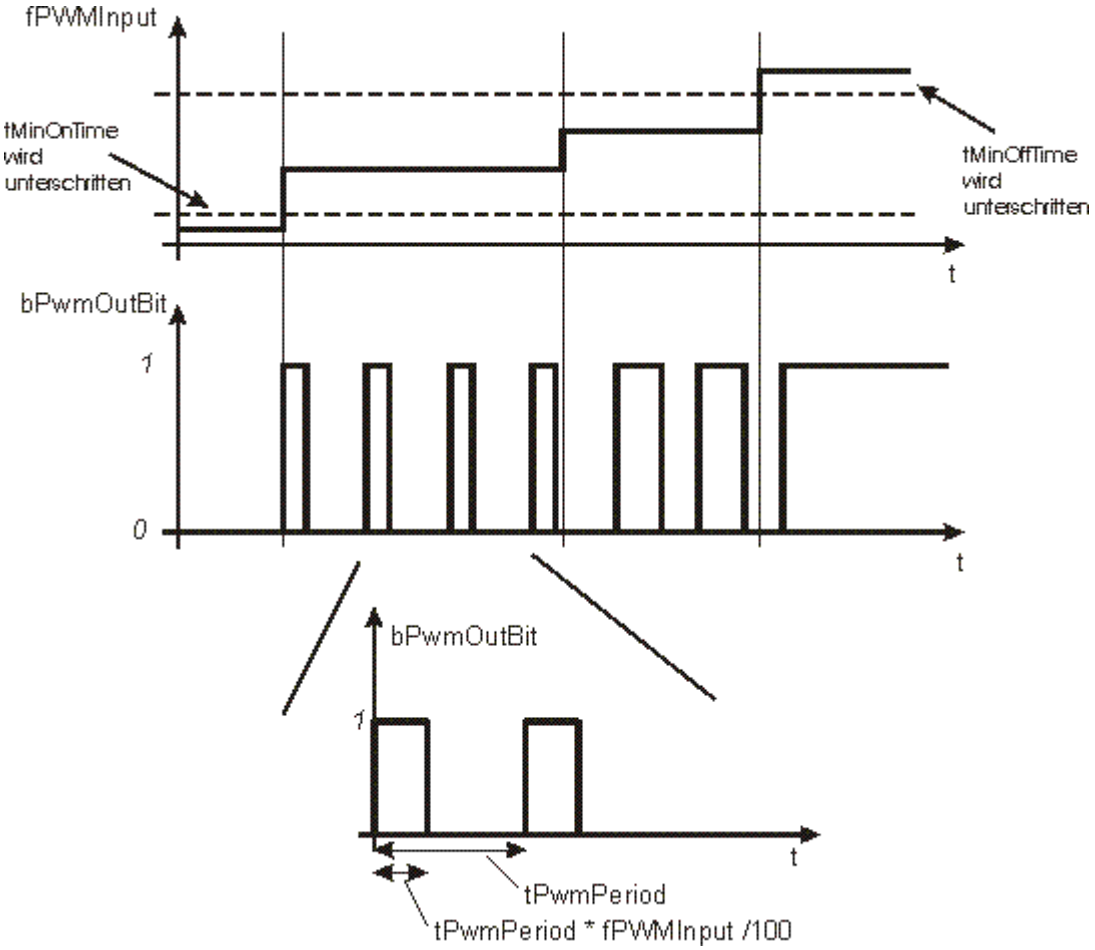
Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.8.5 FB_CTRL_PWM_OUT_EXT

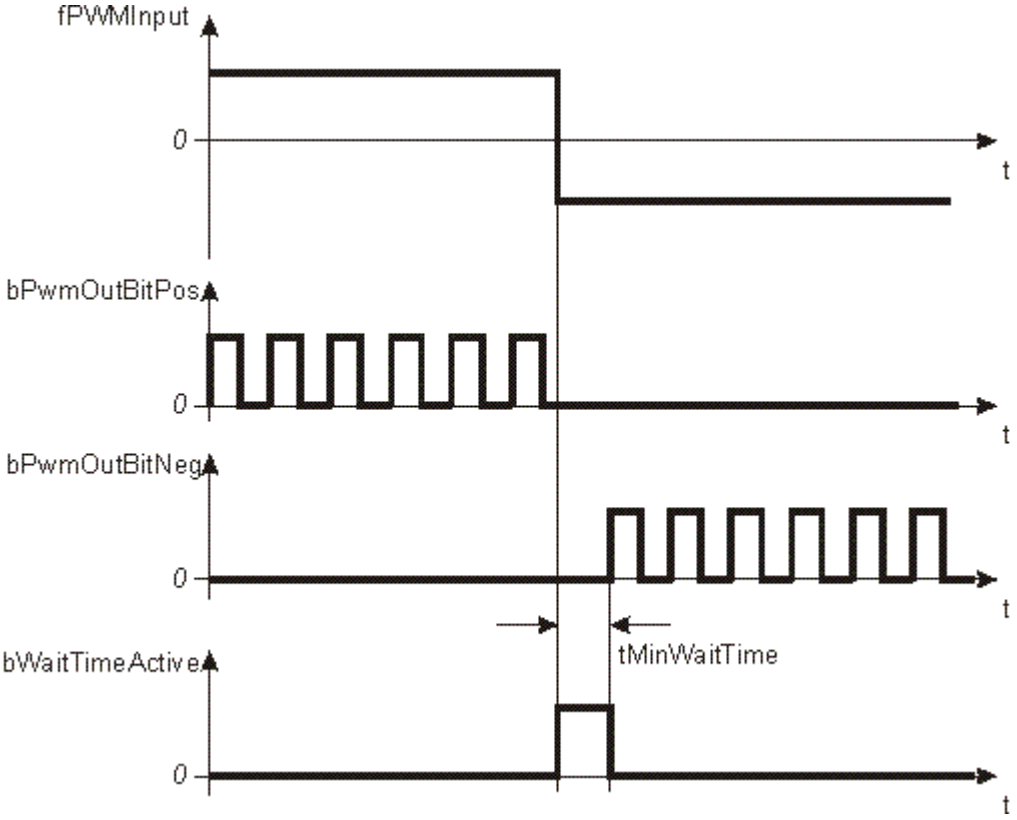


This block creates a PWM modulated signal on the basis of the input signal. Both the minimum switch on time and the minimum switch off time can be parameterised, in addition to the mark-to-space ratio, in this extended block.

Description of the output behaviour (1):



Description of the output behavior (2):



VAR_INPUT

```
VAR_INPUT
    fPwmInput       : FLOAT;        (* NEW: controller output = PMW input [-100.0 ... 100.0] *)
    eMode       : E_CTRL_MODE;
END_VAR
```

fPwmInput : Input value for the function block.

eMode : Input that specifies the block's operating mode.

VAR_OUTPUT

```
VAR_OUTPUT
    bPwmOutBitPos   : BOOL;         (* PWM output bit *)
    bPwmOutBitNeg   : BOOL;         (* PWM output bit *)
    eState      : E_CTRL_STATE;
    bError      : BOOL;
    eErrorId        : E_CTRL_ERRORCODES;
END_VAR
```

bPwmOutBitPos : PWM signal, when fPwmInput > 0.0.

bPwmOutBitNeg : PWM signal, when fPwmInput < 0.0.

bWaitTimeActive : A TRUE at this output indicates that the waiting time between the switching actions of the output signals is active. This output can be used to hold an I component that may be present in the prior controller constant during this time.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_PWM_OUT_EXT_PARAMS;
END_VAR
```

stParams : Parameter structure of the PWM element. This consists of the following elements:

```
TYPE
ST_CTRL_PWM_OUT_EXT_PARAMS :
STRUCT
    tTaskCycleTime    : TIME; (* PLC/PWM cycle time in seconds
*)
    tPWMPeriod        : TIME; (* controller cycle time in
seconds *)
    tMinOnTime        : TIME; (* min. switch on time *)
    tMinOffTime       : TIME; (* min. switch off time *)
    tMinWaitTime      : TIME; (* min. waiting time when
switching from pos to neg or vv*)
END_STRUCT
END_TYPE
```

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

tPWMPeriod : Period of the PWM signal.

tMinOnTime : Minimum switch-on time.

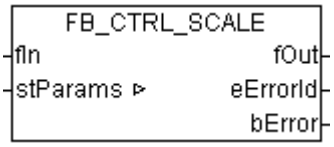
tMinOffTime : Minimum switch-off time.

tMinWaitTime : Waiting time between switching between a positive and negative output signal.

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.8.6 FB_CTRL_SCALE



This function block makes it possible to adjust the signal in a linear manner between two value ranges.

VAR_INPUT

```
VAR_INPUT
    fIn         : FLOAT;
END_VAR
```

fIn : Input value for the function block.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut        : FLOAT;
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

fOut : Scaled output value.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_SCALE_PARAMS;
END_VAR
```

stParams : Parameter structure of the function block. This consists of the following elements:

```
TYPE
ST_CTRL_SCALE_PARAMS:
STRUCT
    tCtrlCycleTime     : TIME     := T#0ms; (* controller cycle
time [TIME] *)
    tTaskCycleTime     : TIME     := T#0ms; (* task cycle time
[TIME] *)
    fInMin          : FLOAT;
    fInMax          : FLOAT;
    fOutMin         : FLOAT;
    fOutMax         : FLOAT;
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

fInMin : Minimum of the input value.

fInMax : Maximum of the input value.

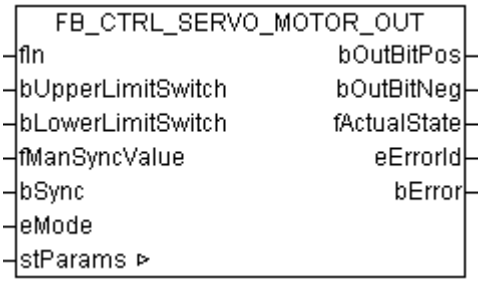
fOutMin : Minimum of the output value.

fOutMax : Maximum of the output value.

Requirements

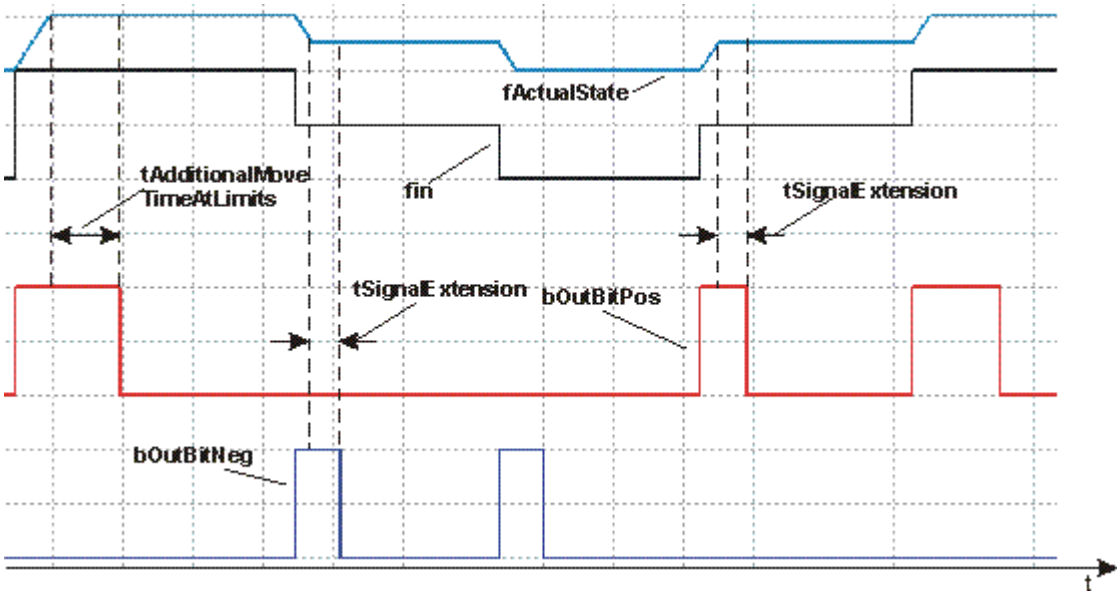
Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.8.7 FB_CTRL_SERVO_MOTOR_OUT



This function block generates pulses with which a servomotor can be driven to a defined position.

Behaviour of the output:



VAR_INPUT

```
VAR_INPUT
    fIn         : FLOAT; (* controller output = SERVO_MOTOR_OUT input [ fCtrlOutMin ... fCtrlOutMax
] *)
    bUpperLimitSwitch       : BOOL;
    bLowerLimitSwitch       : BOOL;
    fManSyncValue   : FLOAT;
    bSync       : BOOL;
    eMode       : E_CTRL_MODE;
END_VAR
```

fIn : control value of the controller in the interval [fCtrlOutMin ... fCtrlOutMax] (controller output).

bUpperLimitSwitch : limit switch: TRUE if the upper stop is reached.

bLowerLimitSwitch : limit switch: TRUE if the lower stop is reached.

fManSyncValue : input with which the internal state of the current motor position can be adjusted, or whose value is adopted in manual mode.

bSync : a rising edge at this input sets the internal motor position, which must correspond to the actual position, to the value fManSyncValue.

eMode : input that specifies the operation mode [▶ 16] of the function block.

For synchronization with the current valve position, this function block has the mode "eCTRL_MODE_SYNC_MOVEMENT". In this mode, the output for closing the valve is set until the valve is closed safely. The function block is then synchronized with this valve position.

Once the synchronization process is completed, "eCTRL_STATE_ACTIVE" mode is automatically activated!

VAR_OUTPUT

```
VAR_OUTPUT
    bOutBitPos      : BOOL; (* PWM output bit *)
    bOutBitNeg      : BOOL; (* PWM output bit *)
    fActualState    : FLOAT; (* Actual state of the motor [ 0% ... 100%] *)
    eState      : E_CTRL_STATE;
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

bOutBitPos : Output required to drive the motor in a positive direction.

bOutBitNeg : Output required to drive the motor in a negative direction.

fActualState : Current motor setting over the range [fCtrlOutMin ... fCtrlOutMax] in which the motor is currently positioned.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_SERVO_MOTOR_OUT_PARAMS;
END_VAR
```

stParams : Parameter structure of the function block. This consists of the following elements:

```
TYPE
ST_CTRL_SERVO_MOTOR_OUT_PARAMS:
STRUCT
    tCtrlCycleTime            : TIME     := T#0ms;
(* controller cycle time [TIME] *)
    tTaskCycleTime            : TIME     := T#0ms;
(* task cycle time [TIME] *)
    tMovingTime               : TIME;
(* time to move from closed (e.g. 0%) to open (e.g. 100%)*)
    tSignalExtension          : TIME;
(* output signal extension time *)
    tAdditionalMoveTimeAtLimits       : TIME;
(* add these moving time to ensure that the limits are
reached*)
    tMinWaitTimeBetweenDirectionChange: TIME;
(* wait time between pos and neg output pulses and vice versa
*)
    tMinimumPulseTime         : TIME;
(* minimum output pulse length *)
    bMoveOnLimitSwitch        : BOOL;
    bStopAdditionalMoveTimeIfInputValueIsChanged : BOOL;
(* if an additional move on the limits is active, this will be
stopped if the input value is changed *)
    fCtrlOutMax               : FLOAT := 100.0;
(* controller output to move to the max limit *)
    fCtrlOutMin               : FLOAT := 0.0;
(* controller output to move to the min limit *)
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

tMovingTime : The time required to move the actuator from one stop to the other.

tSignalExtension : Signal extension by which each output pulse is extended in order to compensate for the dead time.

tAdditionalMovingTimeAtLimits : Signal extension, which is additionally output for safe reaching of the limits, if the actuator is to be moved to +/-100%. Only effective if bMoveOnLimitSwitch is FALSE.

tMinWaitTimeBetweenDirectionChange : Minimum waiting time between positive and negative output pulses.

tMinimumPulseTime : Minimum length of an output pulse.

bMoveOnLimitSwitch : If TRUE, then when the control value is either fCtrlOutMin or fCtrlOutMax a signal will continue to be output until the corresponding limit switch is reached.

bStopAdditionalMoveTimeIfInputValueIsChanged : If this flag is TRUE, movement of the valve to the end position, which is triggered by "Additional Moving Time At Limits", is stopped, if an input value is specified that does not match the end position. If this flag is FALSE and the input value matches an end position, the valve always safely moves to the end position first, before it can move to a different valve position.

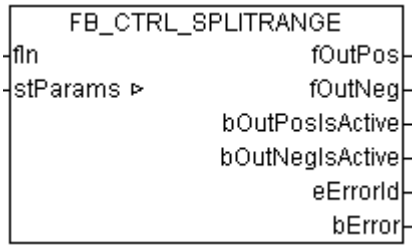
fCtrlOutMax : Control value for which the valve will be driven to 100%.

fCtrlOutMin : Control value for which the valve will be driven to 0%.

Requirements

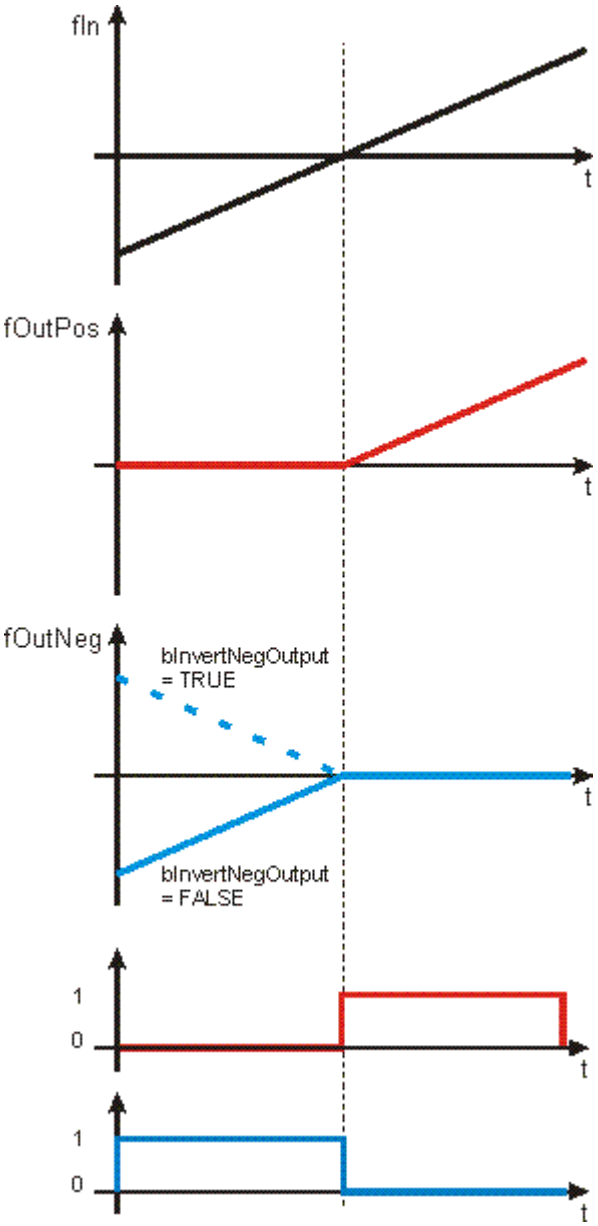
Development environment	Target platform	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.8.8 FB_CTRL_SPLITRANGE



This function block divides an input signal into positive and negative components. The parameters bDisablePosOut and bDisableNegOut can be used to deactivate the positive or negative output ? Heating operation only in winter, cooling operation only in summer. The bInvertNegOutput parameter allows the native output to be inverted.

Description of the output behaviour:



VAR_INPUT

```
VAR_INPUT
    fIn         : FLOAT;
END_VAR
```

fIn : Input value for the function block.

VAR_OUTPUT

```
VAR_OUTPUT
    fOutPos         : FLOAT;
    fOutNeg         : FLOAT;
    bOutPosIsActive     : BOOL;
    bOutNegIsActive     : BOOL;

    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

fOutPos : Positive part of fIn.

fOutNeg : Negative part of fIn.

bOutPosIsActive : TRUE indicates that fIn > 0.0,

bOutNegIsActive : TRUE indicates that fIn < 0.0,

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_SPLITRANGE_PARAMS;
END_VAR
```

stParams : Parameter structure of the function block. This consists of the following elements:

```
TYPE
ST_CTRL_SPLITRANGE_PARAMS:
STRUCT
    tCtrlCycleTime          : TIME      := T#0ms; (*
controller cycle time [TIME] *)
    tTaskCycleTime          : TIME      := T#0ms; (* task
cycle time [TIME] *)
    bInvertNegOutput        : BOOL;
    bDisablePosOut          : BOOL;
    bDisableNegOut          : BOOL;
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

bInvertNegOutput : fOutNeg is inverted when this parameter is TRUE.

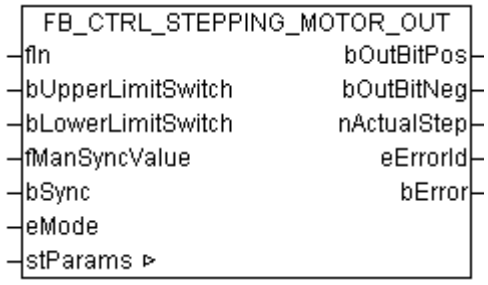
bDisablePosOut : The output fOutPos is disabled and always 0.0.

bDisableNegOut : The output fOutNeg is disabled and always 0.0.

Requirements

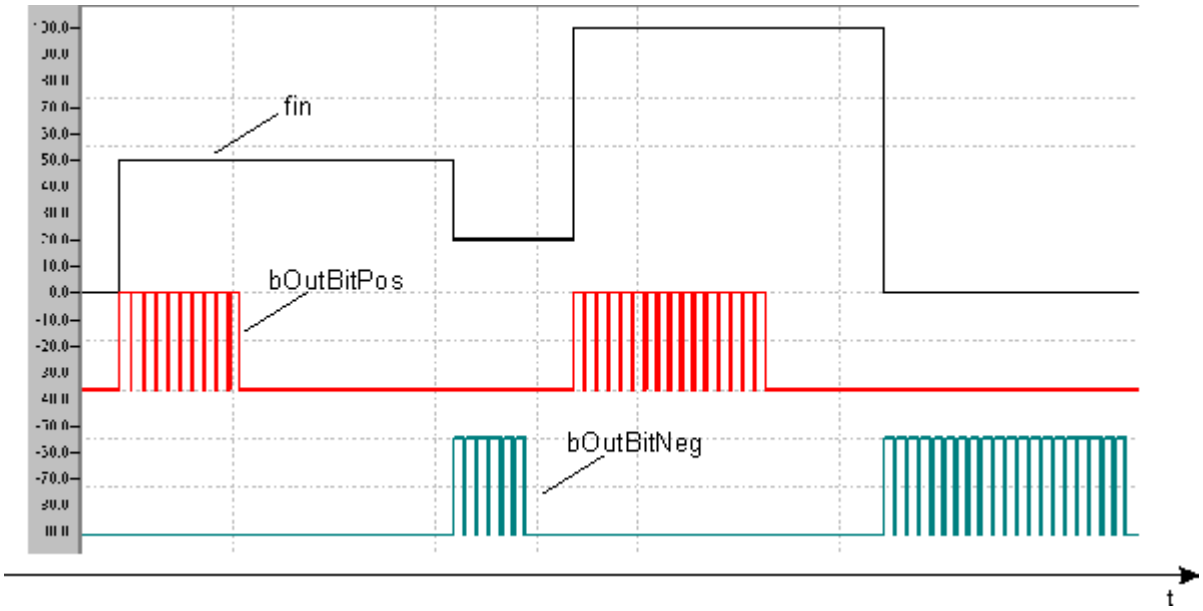
Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9, build 947 onwards	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9, build 956 onwards	BX [▶ 10]	TcControllerToolbox.lbx

5.8.9 FB_CTRL_STEPPING_MOTOR_OUT



This function block generates a control value for a stepper motor.

Behaviour of the output:



VAR_INPUT

```
VAR_INPUT
    fIn         : FLOAT; (* controller output = STEPPING_MOTOR_OUT input [ fCtrlOutMin ... fCtrlOutM
ax ] *)
    bUpperLimitSwitch       : BOOL;
    bLowerLimitSwitch       : BOOL;
    fManSyncValue   : FLOAT;
    bSync       : BOOL;
    eMode       : E_CTRL_MODE;
END_VAR
```

fIn : control value of the controller (controller output).

bUpperLimitSwitch : limit switch: TRUE if the upper stop is reached.

bLowerLimitSwitch : limit switch: TRUE if the lower stop is reached.

fManSyncValue : input with which the internal state of the motor position can be adjusted, or whose value is adopted in manual mode.

bSync : with a rising edge at this input the internal step counter is set to the step that corresponds to the value fManSyncValue.

eMode : input that specifies the operation mode [▶ 16] of the function block.

For synchronization with the current valve position, this function block has the mode "eCTRL_MODE_SYNC_MOVEMENT". In this mode, pulses for closing the valve are output until the valve is safely closed. The function block is then synchronized with this valve position.

Once the synchronization process is completed, "eCTRL_STATE_ACTIVE" mode is automatically activated!

VAR_OUTPUT

```
VAR_OUTPUT
    bOutBitPos      : BOOL; (* output signal to move to the maximum *)
    bOutBitNeg      : BOOL; (* output signal to move to the minimum*)
    nActualStep     : DINT; (* Actual state of the motor [ 0 ... nMaxMovingPulses ] *)
    eState      : E_CTRL_STATE;
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

bOutBitPos : Output required to drive the motor in a positive direction.

bOutBitNeg : Output required to drive the motor in a negative direction.

nActualStep : Actual step at which the motor is positioned.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_STEPPING_MOTOR_OUT_PARAMS;
END_VAR
```

stParams : Parameter structure of the function block. This consists of the following elements:

```
TYPE
ST_CTRL_STEPPING_MOTOR_OUT_PARAMS:
STRUCT
    tCtrlCycleTime         : TIME     := T#0ms; (*
controller cycle time [TIME] *)
    tTaskCycleTime         : TIME     := T#0ms; (* task
cycle time [TIME] *)
    tOnTime            : TIME;          (*
impulse length *)
    tOffTime               : TIME;          (*
delay between two impulses *)
    nMaxMovingPulses           : DINT;          (*
necessary pulses to move from closed to open or v. v. *)
    nMinMovingPulses           : UINT := 0;     (* min
pulses to start movement *)
    bHoldWithOutputOn          : BOOL;          (* if
TRUE then a ouput is on when no pulses are generated *)
    nAdditionalPulsesAtLimits      : DINT;          (* add
these pulses to ensure that the limits are reached*)
    bMoveOnLimitSwitch         : BOOL;
    fCtrlOutMax            : FLOAT := 100.0;    (*
controller output to move to the max limit *)
    fCtrlOutMin            : FLOAT := 0.0;      (*
controller output to move to the min limit *)
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

tOnTime : Pulse length.

tOffTime : Pause length.

nMaxMovingPulses : Pulses required to move from one limit to another.

bHoldWithOutputOn : If this parameter is set to TRUE, then an output remains set when the drive is stationary. This will result in braking.

nAdditionalPulsesAtLimits : Pulses that are additionally output to safely reach the limits.

bMoveOnLimitSwitch : If this is TRUE, then pulses are output when the control value is either 0% or 100% until a limit switch is reached.

fCtrlOutMax : Control value for which the valve will be driven to 100%.

fCtrlOutMin : Control value for which the valve will be driven to 0%.

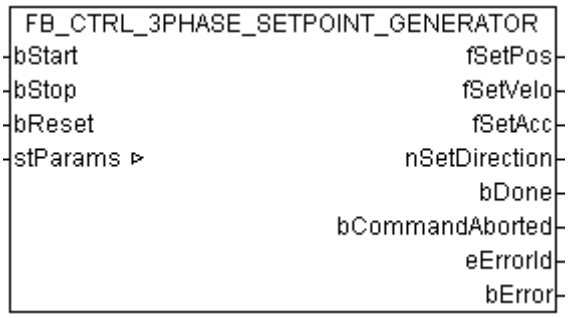
fMinCtrtlDeltaToStartMovement : Minimum difference in control value that must be exceeded in order to generate movement of the motor.

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.9 Setpointgeneration

5.9.1 FB_CTRL_3PHASE_SETPOINT_GENERATOR(only on a PC system)



This function block represents a 3-phase set value generator.

Description:

This function block generates a 3-phase setpoint profile in which the acceleration has a rectangular curve. It is possible to specify a new parameter set while the generator is active. Depending upon what type of parameter set is specified, it may become effective immediately (eNewParameterType := eCTRL_NEW_PARAMETER_TYPE_Instant); alternatively, the current movement may first be completed, after which a new movement begins with a new set of parameters. (eNewPosType := eCTRL_NEW_PARAMETER_TYPE_NotInstant).

NOTE

1. Specifying a new parameter set may mean that the previous end position is exceeded. See example.
2. It is generally recommended that end position monitoring is included after the setpoint generator.

VAR_INPUT

```
VAR_INPUT
    bStart      : BOOL;
    bStop       : BOOL;
    bReset      : BOOL;
    fOverride   : LREAL;
END_VAR
```

bStart : Generation of the set values begins when a positive edge appears at the *bStart* input, provided the generator is not already active and that the *bStop* and *bReset* inputs are FALSE.

bStop : Generation of the set values is stopped by a positive edge at the *bStop* input. Braking is carried out using the deceleration specified in the current set of parameters, and any subsequent task that may be pending is deleted.

bReset : Resets the set value generator. If a positioning process is under way it is interrupted immediately, the fSetVelo and fSetAcc outputs are set to 0.0, the set position is set to the start position, and all internal states are cleared.

fOverride : The set velocity can be scaled through an override in the range [0 .. 100.0 %]. If the override is 100%, then the generated profile uses the set velocity as specified in the parameter set. The override implemented here does not scale the current runtime table when the override is changed. Instead, an internal restart instruction is generated having a different set velocity. The override has a resolution of 0.1%.

VAR_OUTPUT

```
VAR_OUTPUT
    fSetPos         : LREAL; (* generated setpoint position *)
    fSetVelo        : LREAL; (* generated setpoint velocity *)
    fSetAcc         : LREAL; (* generated setpoint acceleration *)
    nSetDirection   : INT;   (* generated direction *)
     bCommandBuffered       : BOOL;
    bDone       : BOOL;
    bCommandAborted     : BOOL;

    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

fSetPos : Set position

fSetVelo : Set velocity

fSetAcc : Set acceleration

nSetDirection : Direction of movement [-1, 0, 1],
1 --> movement direction is positive
0 --> generator is inactive
-1 --> movement direction is negative

bGeneratorActive : Indicates whether the generator is active.

bCommandBuffered : When this output is TRUE it indicates that the buffer store contains at a transport instruction that will begin when the current instruction has completed.
An instruction in the buffer store is cleared if the following special case is specified as parameter set:

```
          fAcceleration
:= 0.0;

fDeceleration      := 0.0;

fStartPos      := 0.0;

fStartVelo     := 0.0;

fTargetPos     := 0.0;

fTargetVelo    := 0.0;

fVelocity      := 0.0;

tCtrlCycleTime     := T#0s;

tTaskCycleTime     := T#0s;

eNewParameterType  :=
eCTRL_NEW_PARAMETER_TYPE_NotInstant;
```

bDone : This output becomes TRUE when the movement has been completed and the destination position has been reached.

bCommandAborted : This output becomes TRUE when the current movement is interrupted. This can be caused, for instance, through a rising edge at the *bStop* input.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

bError : Becomes TRUE as soon as an error situation occurs.

VAR_IN_OUT

```
VAR_IN_OUT
    stParams : ST_CTRL_3PHASE_SETPOINT_GENERATOR_PARAMS;
END_VAR
```

stParams : parameter structure of the setpoint generator. This consists of the following elements:

```
TYPE
ST_CTRL_RAMP_GENERATOR_PARAMS :
STRUCT
    tTaskCycleTime     : TIME;  (* task cycle time [TIME]
*)
    tCtrlCycleTime     : TIME;  (* controller cycle time [TIME]
*)
    fStartPos      : LREAL;
    fStartVelo     : LREAL;
    fVelocity      : LREAL; (* >= 0.0 *)
    fTargetPos     : LREAL;
    fTargetVelo    : LREAL;
    fAcceleration      : LREAL; (* > 0.0 *)
    fDeceleration      : LREAL; (* > 0.0 *)
    eNewParameterType  : E_CTRL_NEW_PARAMETER_TYPE;
END_TYPE
```

tCtrlCycleTime : cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime: cycle time with which the function block is called. If the function block is called in every task cycle this corresponds to the task cycle time of the calling task.

fStartPos : start position for the motion profile.

fStartVelo : start velocity of the motion profile.

fVelocity : velocity in units/second.

fTargetPos : target position for the motion profile.

fTargetVelo : target velocity of the motion profile.

Note **The target velocity is retained when the target position has been reached (the bDone flag is set), but after this point the position is no longer calculated (constant position at velocity ≠ 0.0).**

fAcceleration : acceleration in units / second^2.

fDeceleration : deceleration in units / second^2.

eNewParameterType :

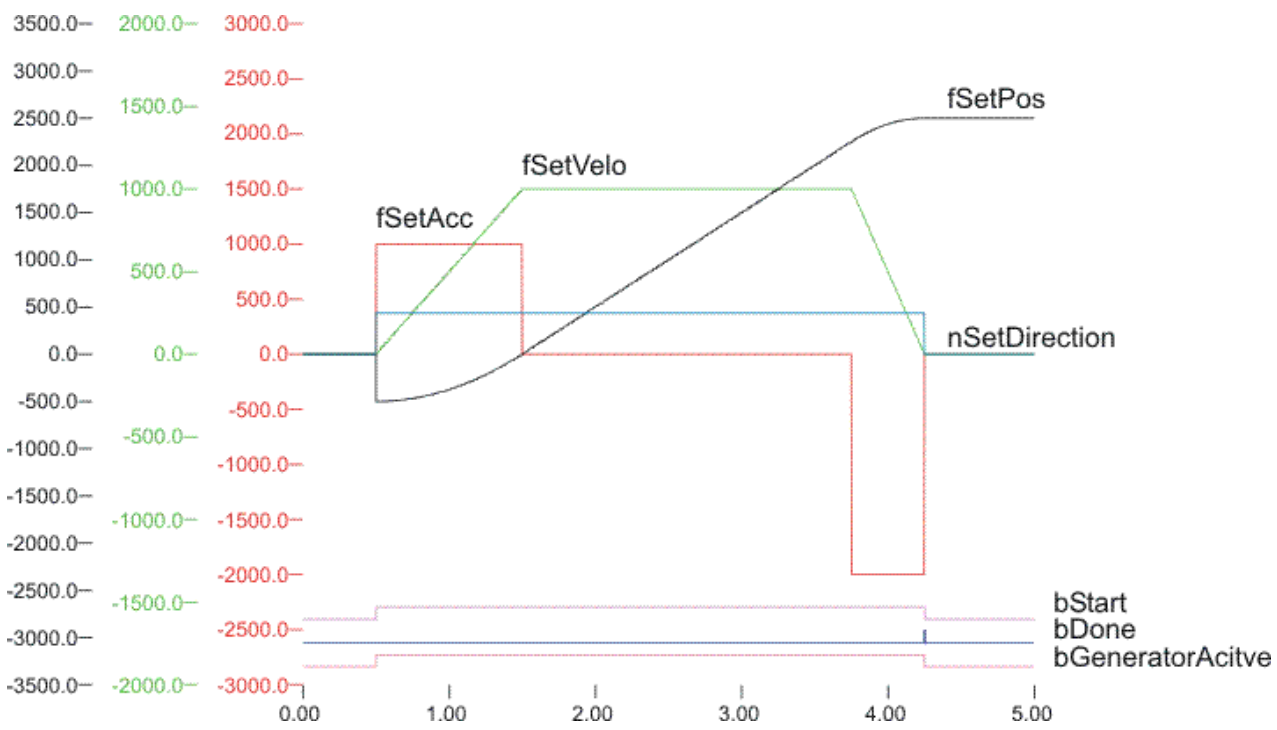
```
           TYPE E_CTRL_NEW_PARAMETER_TYPE :
(
    eCTRL_NEW_PARAMETER_TYPE_NotInstant := 0,
    eCTRL_NEW_PARAMETER_TYPE_Instant    := 1
);
END_TYPE
```

eCTRL_NEW_PARAMETER_TYPE_Instant: when a restart instruction is issued with a new parameter set, the new set is adopted immediately. In other words a transition from the current state of movement to the data represented by the new parameter set is calculated, and the old parameters are discarded.

eCTRL_NEW_PARAMETER_TYPE_NotInstant: when a restart instruction is issued with a new parameter set, the new set is not adopted immediately. In other words the current movement is first executed to completion, after which positioning to the new target is carried out with the new parameters. A TRUE at the bCommandBuffered output indicates that a non-instantaneous restart instruction is stored. An order that has already been stored can be overwritten or deleted by a further new, non-instantaneous parameter set.

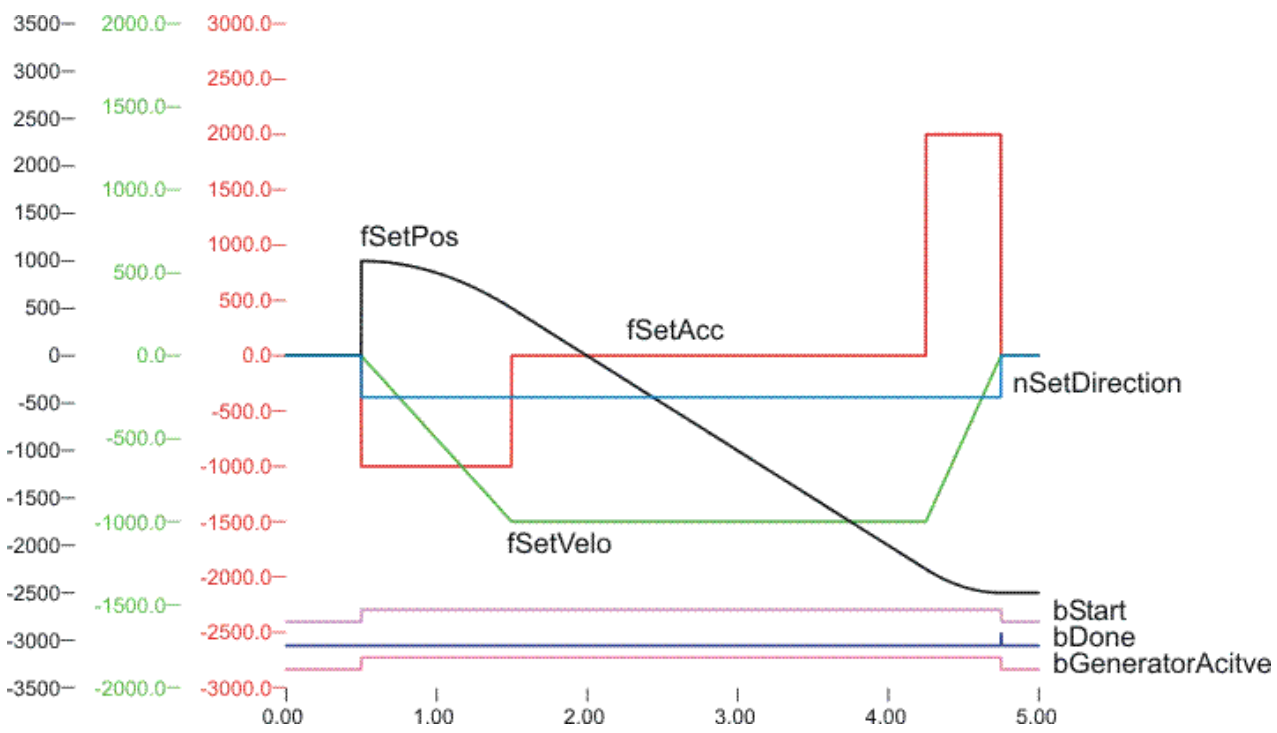
Positioning examples

Positioning example 1:	`stParams.fStartPos := -500.0;`
	`stParams.fTargetPos := 2500.0;`
	`stParams.fStartVelo := 0.0;`
	`stParams.fVelocity := 1000.0;`
	`stParams.fTargetVelo := 0.0;`
	`stParams.fAcceleration := 1000.0;`
	`stParams.fDeceleration := 2000.0;`
	`fOverride := 100.0;`



Positioning example 2:

```
stParams.fStartPos := 1000.0;
stParams.fTargetPos := -2500.0;
stParams.fStartVelo := 0.0;
stParams.fVelocity := 1000.0;
stParams.fTargetVelo := 0.0;
stParams.fAcceleration := 1000.0;
stParams.fDeceleration := 2000.0;
fOverride := 100.0;
```



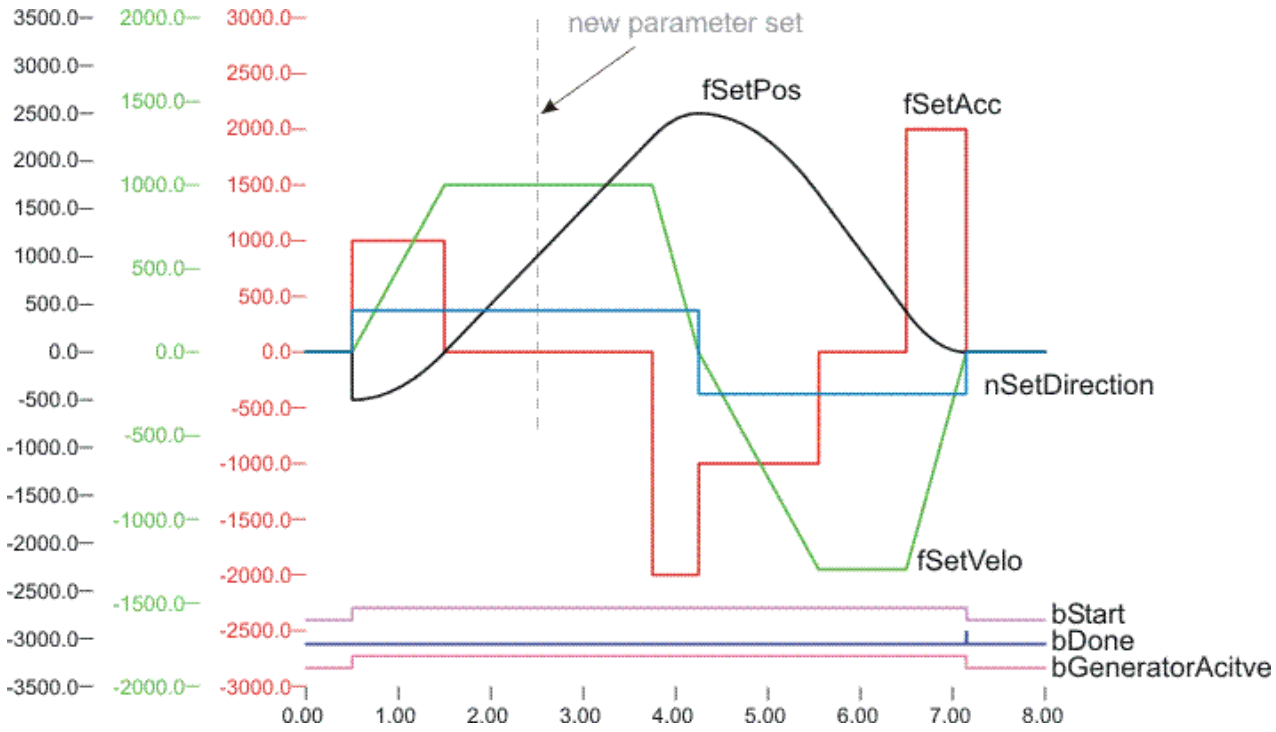
Positioning example 3:

```
stParams.fStartPos := -500.0;
stParams.fTargetPos := 2500.0;
stParams.fStartVelo := 0.0;
stParams.fVelocity := 1000.0;
stParams.fTargetVelo := 0.0;
stParams.fAcceleration := 1000.0;
```

```
stParams.fDeceleration := 2000.0; stParams.eNewParameterType :=
eCTRL_NEW_PARAMETER_TYPE_NotInstant;
fOverride := 100.0;
```

Parameter change if fSetPos > 1000.0, eNewPosType := eCTRL_NEW_POS_TYPE_NotInstant

```
stParams.fTargetPos := 0.0;
stParams.fStartVelo := 0.0;
stParams.fVelocity := 1300.0;
stParams.fTargetVelo := 0.0;
stParams.fAcceleration := 1000.0;
stParams.fDeceleration := 2000.0;
fOverride := 100.0;
```

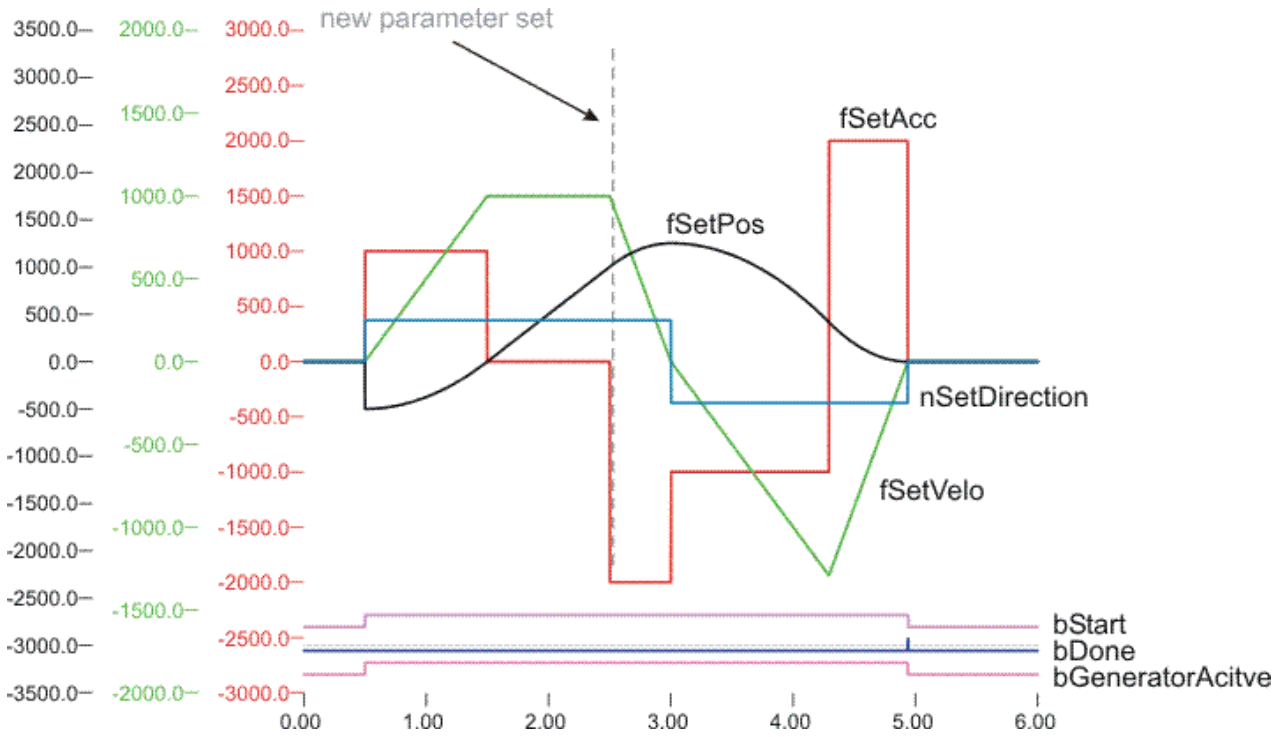


Positioning example 4:

```
stParams.fStartPos := -500.0;
stParams.fTargetPos := 2500.0;
stParams.fStartVelo := 0.0;
stParams.fVelocity := 1000.0;
stParams.fTargetVelo := 0.0;
stParams.fAcceleration := 1000.0;
stParams.fDeceleration := 2000.0;
stParams.eNewParameterType := eCTRL_NEW_PARAMETER_TYPE_NotInstant;
fOverride := 100.0;
```

Parameter change if fSetPos > 1000.0, eNewPosType := eCTRL_NEW_POS_TYPE_Instant

```
stParams.fTargetPos := 0.0;
stParams.fStartVelo := 0.0;
stParams.fVelocity := 1300.0;
stParams.fTargetVelo := 0.0;
stParams.fAcceleration := 1000.0;
stParams.fDeceleration := 2000.0;
fOverride := 100.0;
```



Positioning example 5:

Start on point 1:

```
stParams.fStartPos := -100.0;
stParams.fTargetPos := 200.0;
stParams.fStartVelo := 0.0;
stParams.fVelocity := 250.0;
stParams.fTargetVelo := 150.0;
stParams.fAcceleration := 500.0;
stParams.fDeceleration := 400.0;
stParams.eNewParameterType := eCTRL_NEW_PARAMETER_TYPE_Instant
```

Restart at point 2:

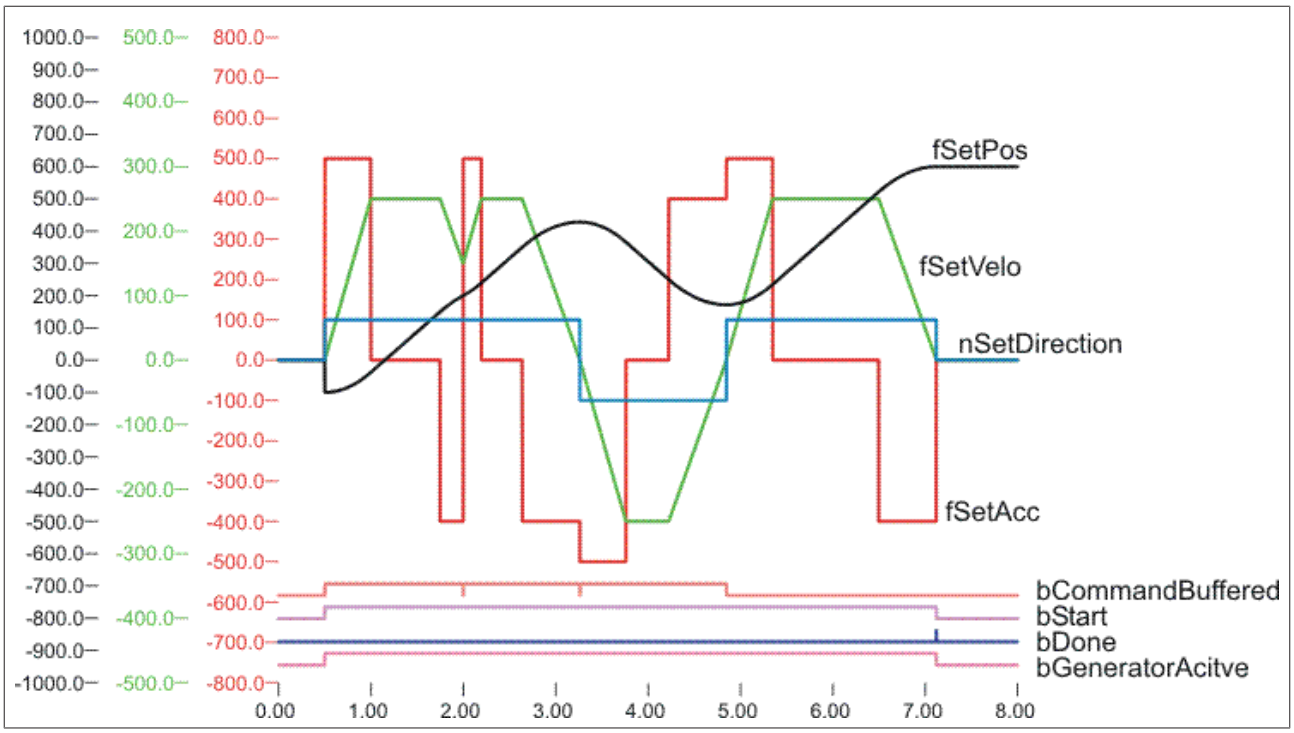
```
stParams.fTargetPos := 400.0;
stParams.eNewParameterType := eCTRL_NEW_PARAMETER_TYPE_NotInstant;
```

Restart at point 3:

```
stParams.fTargetPos := 200.0;
stParams.eNewParameterType := eCTRL_NEW_PARAMETER_TYPE_NotInstant;
```

Restart at point 4:

```
stParams.fTargetPos := 600.0;
stParams.fTargetVelo := 0.0;
stParams.eNewParameterType := eCTRL_NEW_PARAMETER_TYPE_NotInstant;
```



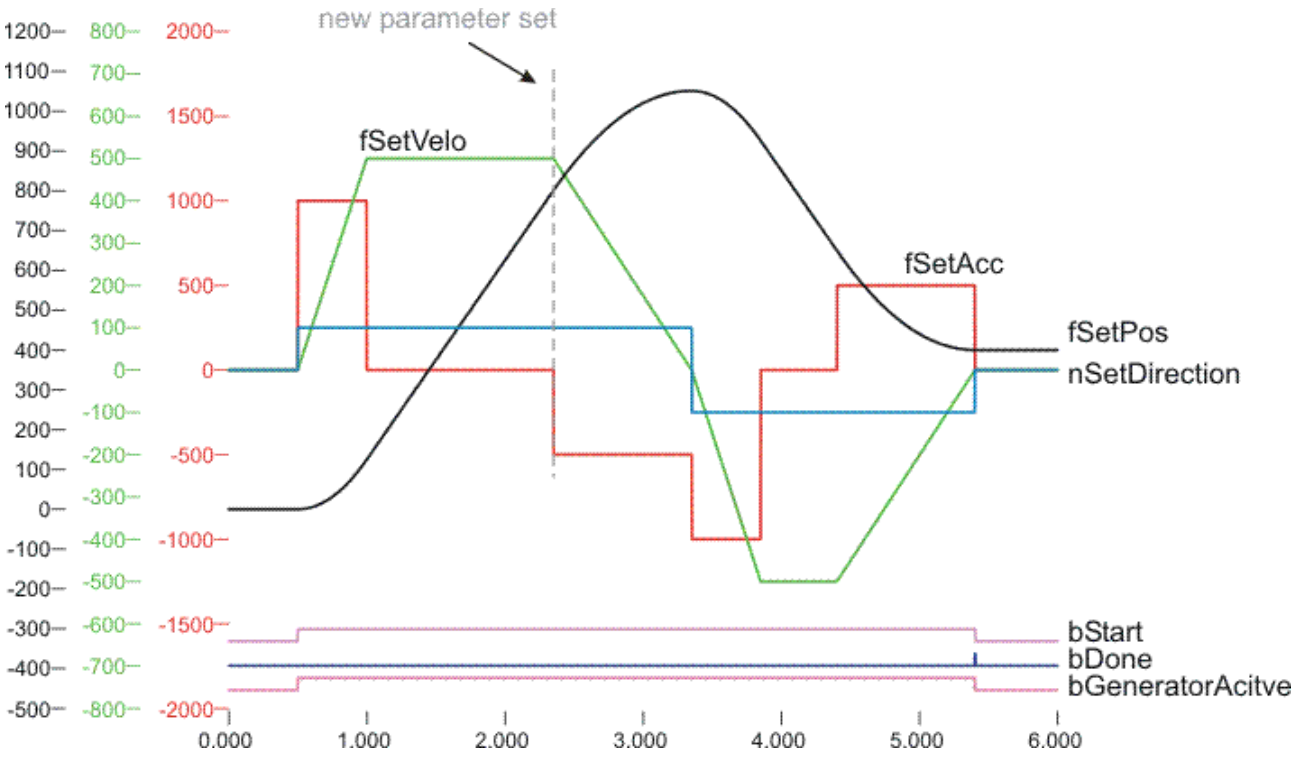
NOTE

If a new parameter set of type "`eCTRL_NEW_POS_TYPE_Instant`" in which the deceleration is reduced is given to the function block it is possible that the old target position will be exceeded.

Sample:

```
stParams.fTargetPos      := 1000.0;
stParams.fStartPos       := 0.0;
stParams.fVelocity       := 500.0;
stParams.fAcceleration   := 1000.0;
stParams.fDeceleration   := 1000.0;
IF fSetPos > 800.0
THEN
    stParams.fTargetPos    := 400.0;
    stParams.fVelocity     := 500.0;
    stParams.fAcceleration := 1_000.0;
    stParams.fDeceleration := 500.0;
    stParams.eNewPosType   := eCTRL_NEW_POS_TYPE_Instant;
END_IF
```

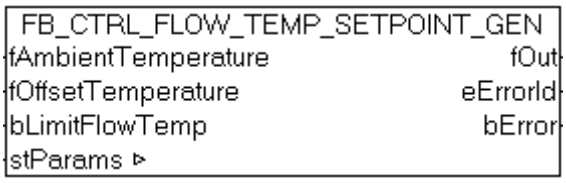
It can clearly be seen in the following scope recording that the original target position of 1000 mm is exceeded, the reason being that the new parameter set had a reduced deceleration.



Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib

5.9.2 FB_CTRL_FLOW_TEMP_SETPOINT_GEN

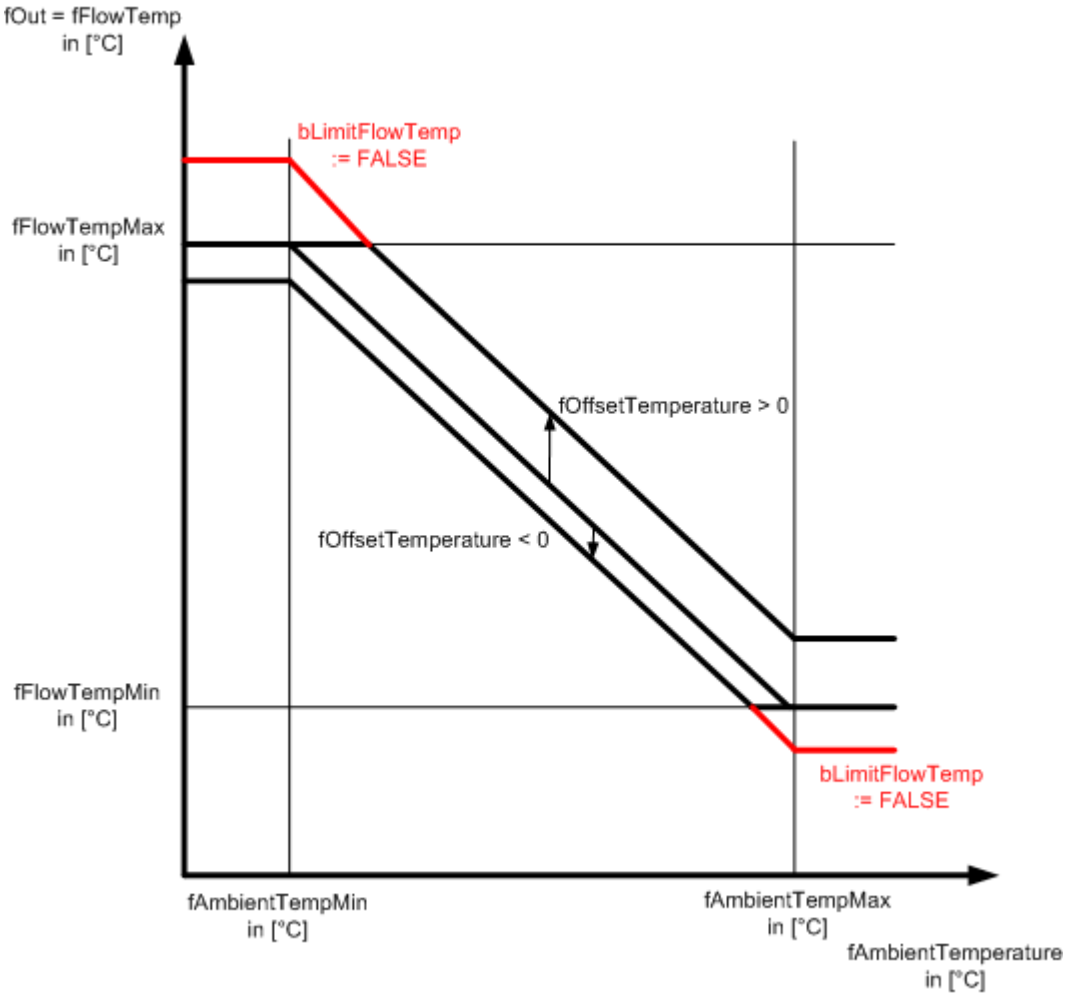


This function block enables specification of a flow temperature depending on the external temperature.

Description:

The set value for the flow temperature (**fOut**) is determined from the ambient temperature (**fAmbientTemperature**). A straight line that can be moved via an offset (**fOffsetTemperature**) is used for this purpose. The slope is determined based on the specified ambient and flow temperature corners. A flag (**bLimitFlowTemp**) is used to specify whether or not the flow temperature is restricted to its limit values. An offset temperature can be used to implement night setback or precontrol.

Behaviour of the output value:



VAR_INPUT

```
VAR_INPUT
    fAmbientTemperature     : FLOAT;
    fOffsetTemperature      : FLOAT;
    bLimitFlowTemp      : BOOL;
END_VAR
```

fAmibientTemperature : Start of ramp generation.

fOffestTemperature : Starting value for the ramp.

bLimitFlowTemp : Finishing value for the ramp.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut        : FLOAT;
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

fOut : set value of the flow temperature.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

bError : Becomes TRUE as soon as an error situation occurs.

VAR_IN_OUT

```
VAR_IN_OUT
    stParams : ST_CTRL_FLOW_TEMP_SETPOINT_GEN_PARAMS;
END_VAR
```

stParams : Parameter structure of the ramp generator. This consists of the following elements:

```
TYPE
ST_CTRL_FLOW_TEMP_SETPOINT_GEN_PARAMS:
STRUCT
    tTaskCycleTime   : TIME;  (* task cycle time [TIME]
*)
    tCtrlCycleTime   : TIME;  (* controller cycle time [TIME]
*)
    fForeRunTempMax  : FLOAT;
    fForeRunTempMin  : FLOAT;
    fAmbientTempMax  : FLOAT;
    fAmbientTempMin  : FLOAT;
END_STRUCT
END_TYPE
```

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every task cycle this corresponds to the task cycle time of the calling task.

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

fFlowTempMax : Maximum flow temperature (see diagram).

fFlowTempMin : Minimum flow temperature (see diagram).

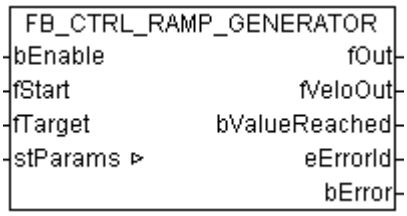
fAmbientTempMax : Outdoor temperature for which the minimum flow temperature is specified.

fAmbientTempMin : Outdoor temperature for which the maximum flow temperature is specified.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9, build 947 onwards	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9, build 956 onwards	BX [▶ 10]	TcControllerToolbox.lbx

5.9.3 FB_CTRL_RAMP_GENERATOR

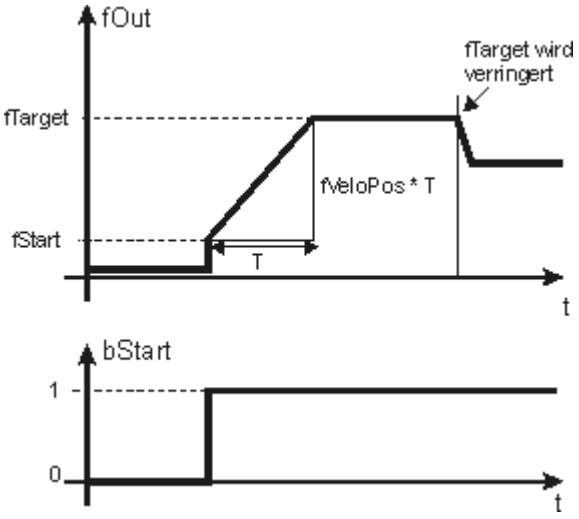


The function block provides a parameterisable ramp generator.

Description:

This function block generates a ramp connecting the starting value **fStart** and the final value **fTarget**. The slope of the ramp (i.e. the velocity) is given in units/s by means of the **fVeloPos** and **fVeloNeg** parameters. The starting value is adopted when a rising edge appears at **bEnable**; calculation of the ramp then begins. As long as the signal bEnable remains TRUE the final value can be changed, and the output value changes, taking the form of a ramp as it moves from the current value to the presently active final value.

Behaviour of the output value:



VAR_INPUT

```
VAR_INPUT
    bEnable     : BOOL;
    fStart  : FLOAT;
    fTarget     : FLOAT; (* target value *)
END_VAR
```

bEnable : Start of ramp generation.

fStart : Starting value for the ramp.

fTarget : Finishing value for the ramp.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut        : FLOAT;
    bValueReached   : BOOL;
    eState      : E_CTRL_STATE;
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

fOut : Output of the ramp generator.

bValueReached : This output indicates by going TRUE that the output fOut has reached the value fTarget.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

bError : Becomes TRUE as soon as an error situation occurs.

VAR_IN_OUT

```
VAR_IN_OUT
    stParams : ST_CTRL_RAMP_GENERATOR_PARAMS; (* RAMP_Generator parameter struct *)
END_VAR
```

stParams : Parameter structure of the ramp generator. This consists of the following elements:

```
TYPE
ST_CTRL_RAMP_GENERATOR_PARAMS :
STRUCT
    tTaskCycleTime    : TIME;   (* task cycle time [TIME]
*)
    tCtrlCycleTime    : TIME;   (* controller cycle time [TIME]
*)
    fVeloPos      : FLOAT; (* velocity ramp by time range:
> 0.0 *)
    fVeloNeg      : FLOAT; (* velocity ramp by time range:
```

```
> 0.0 *)
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every task cycle this corresponds to the task cycle time of the calling task.

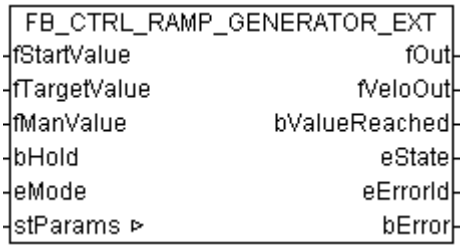
fVeloPos : Velocity in unit/s, with which the output is changed from a lower value to a higher one.

fVeloNeg : Velocity in unit/s, with which the output is changed from a higher value to a lower one.

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.9.4 FB_CTRL_RAMP_GENERATOR_EXT

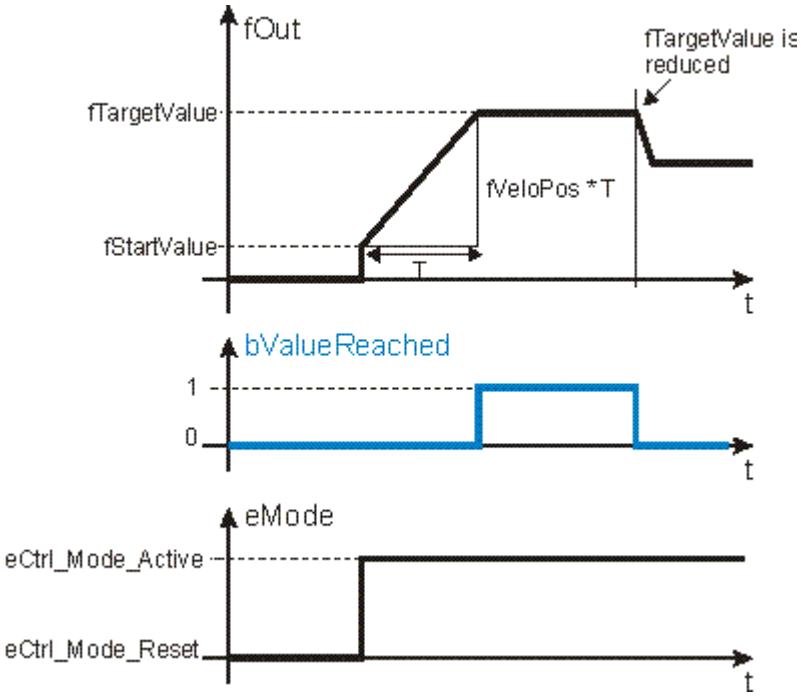


This function block represents a parameterisable ramp generator. In contrast to FB_CTRL_RAMP_GENERATOR it supports the E_CTRL_MODEs.

Description:

This function block generates a ramp connecting the starting value **fStartValue** and the final value **fTargetValue**. The slope of the ramp (i.e. the velocity) is given in units/s by means of the **fVeloPos** and **fVeloNeg** parameters. The starting value is adopted when eCTRL_MODE_RESET changes to eCTRL_MODE_ACTIVE, and calculation of the ramp begins. As long as the block remains in eCTRL_MODE_ACTIVE the target value can be changed, and the output value changes, taking the form of a ramp as it moves from the current value to the presently active target value. The current velocity is output at fVeloOut. It is possible to use this for feed forward in the control loop.

Behaviour of the output value:



VAR_INPUT

```
VAR_INPUT
    fStartValue     : FLOAT;
    fTargetValue    : FLOAT;
    fManValue           : FLOAT;
    bHold       : BOOL;
    eMode       : E_CTRL_MODE;
END_VAR
```

fStartValue : Starting value for the ramp.

fTargetValue : Finishing value for the ramp.

fManValue : Input magnitude to which the output in eCTRL_MODE_MANUAL is set.

bHold : Calculation of the ramp is halted at the current value.

eMode : Input that specifies the block's operating mode [▶ 16].

VAR_OUTPUT

```
VAR_OUTPUT
    fOut        : FLOAT;
    fVeloOut        : FLOAT;
    bValueReached   : BOOL;
    eState      : E_CTRL_STATE;
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

fOut : Output of the ramp generator.

fVeloOut : Current velocity of the ramp generator.

bValueReached : This output indicates by going TRUE that the output fOut has reached the value fTargetValue.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

bError : Becomes TRUE as soon as an error situation occurs.

VAR_IN_OUT

```
VAR_IN_OUT
    stParams : ST_CTRL_RAMP_GENERATOR_EXT_PARAMS;
END_VAR
```

stParams : Parameter structure of the ramp generator. This consists of the following elements:

```
TYPE
ST_CTRL_RAMP_GENERATOR_EXT_PARAMS :
STRUCT
    tTaskCycleTime   : TIME;  (* task cycle time [TIME]
*)
    tCtrlCycleTime   : TIME;  (* controller cycle time [TIME]
*)
    fVeloPos     : FLOAT; (* velocity ramp by time range:
> 0.0 *)
    fVeloNeg     : FLOAT; (* velocity ramp by time range:
> 0.0 *)
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime : Cycle time with which the function block is called. If the function block is called in every task cycle this corresponds to the task cycle time of the calling task.

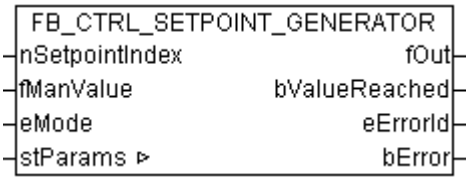
fVeloPos : Velocity (>0.0) in unit/s, with which the output is changed from a lower value to a higher one.

fVeloNeg : Velocity (>0.0) in unit/s, with which the output is changed from a higher value to a lower one.

Requirements

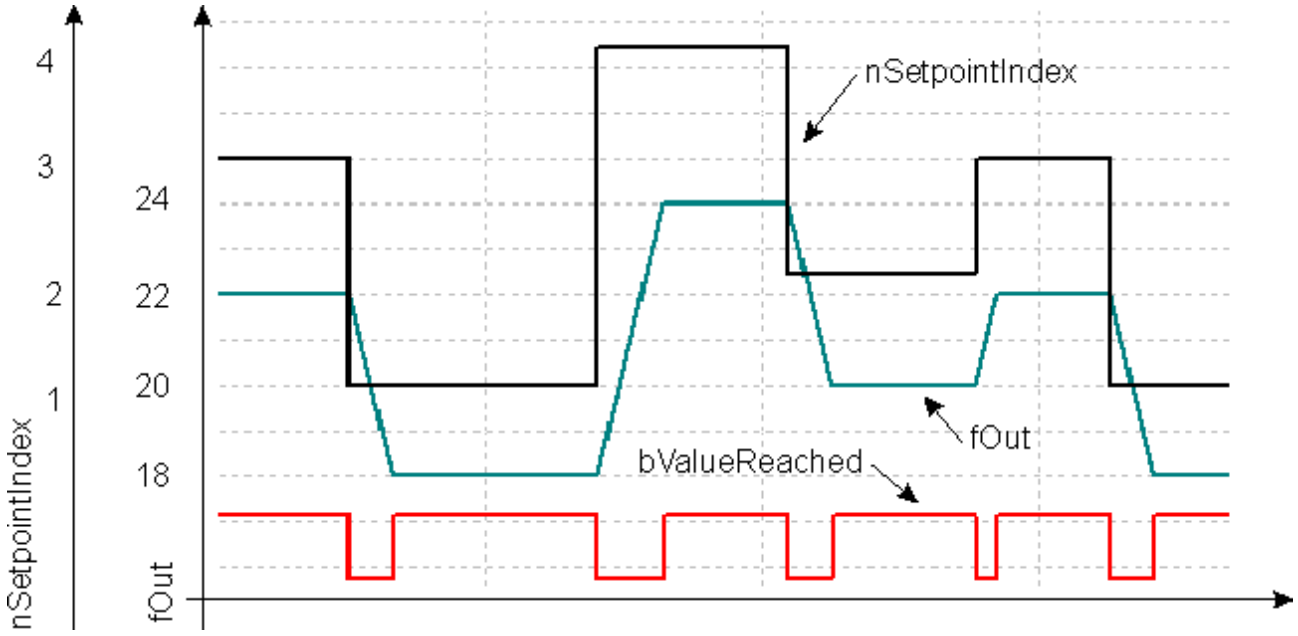
Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9, build 947 onwards	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9, build 956 onwards	BX [▶ 10]	TcControllerToolbox.lbx

5.9.5 FB_CTRL_SETPOINT_GENERATOR



The function block provides a set value generator that outputs the set value selected from a table. Changing from one set value to another can be implemented continuously or discontinuously.

Behavior of the output value:



Example table:

18
20
22
24

The first line of the table corresponds to Index 1, at the second to Index 2, and so forth.

Description:

The individual set values are stored in the array. The array must be made known to the block through the appropriate parameters. One of the set values stored in the table is selected by means of the **nSetpointIndex** input. This is then made available at the output, and can be used as the set value for the controller. Changing from one value to another can be implemented in a linear fashion or as a jump. The velocity of a continuous transition is specified by the **fVeloPos** and **fVeloNeg** parameters. The **bValueReached** output indicates that the chosen set value has been reached.

VAR_INPUT

```
VAR_INPUT
    nSetpointIndex  : INT  (*[1 ... n] *);
    fManValue       : FLOAT;
    eMode       : E_CTRL_MODE;
END_VAR
```

nSetpointIndex : Index of the selected set value.

fManValue : Input whose value is output in manual mode.

eMode : Input that specifies the block's operating mode [▶ 16].

VAR_OUTPUT

```
VAR_OUTPUT
    fOut        : SETPOINT_TABLE_ELEMENT;
    bValueReached   : BOOL;
    eState      : E_CTRL_STATE;
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

fOut : Output of the set value generator.

bValueReached : The output is TRUE when the selected set value has been reached, i.e. the ramp leading to the selected value has finished.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams : ST_CTRL_SETPOINT_GENERATOR_PARAMS; (* parameters *)
END_VAR
```

stParams : Parameter structure of the ramp generator. This consists of the following elements:

```
TYPE
ST_CTRL_SETPOINT_GENERATOR_PARAMS:
STRUCT
    tCtrlCycleTime          : TIME     := T#0ms; (*
controller cycle time [TIME] *)
    tTaskCycleTime          : TIME     := T#0ms; (* task
cycle time [TIME] *)
    pDataTable_ADR          : POINTER TO
INTERPOLATION_TABLE_ELEMENT := 0;
    nDataTable_SIZEOF       : UINT     := 0;
    nDataTable_NumberOfRows     : UINT     := 0;
    fVeloPos            : FLOAT; (* velocity ramp by
time range > 0.0 *)
    fVeloNeg            : FLOAT; (* velocity ramp by
time range > 0.0 *)
    bDisableRamping         : BOOL     := FALSE;
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and output values must be updated in the current cycle.indent: -103; margin-left: 108"> **tTaskCycleTime** : Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

pDataTable_ADR : Address of the data array.

pDataTable_SIZEOF : Size of the data array.

pDataTable_NumberOfRows: Number of rows of the data array.

fVeloPos : Velocity in unit/s, with which the output is changed from a lower value to a higher one.

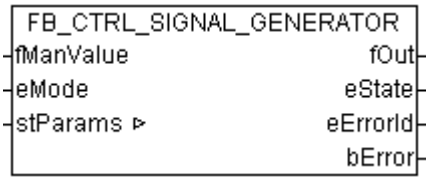
fVeloNeg : Velocity in unit/s, with which the output is changed from a higher value to a lower one.

bDisableRamping: A continuous output value is not calculated if this parameter is TRUE. There is a step change between the output values.

Requirements

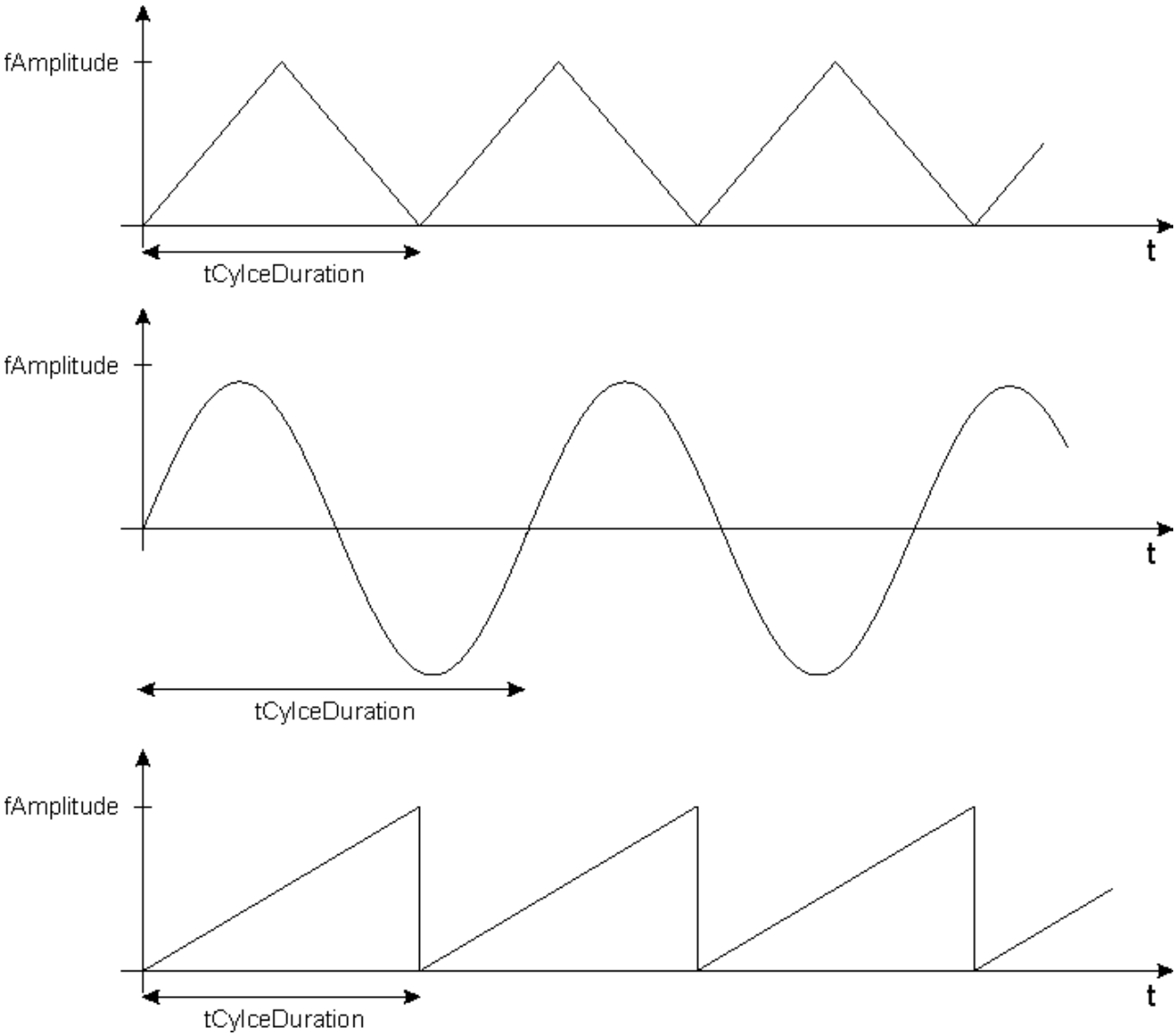
Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

5.9.6 FB_CTRL_SIGNAL_GENERATOR



The function block provides a signal generator offering **triangular, sine** and **sawtooth** signal forms.

Output signals:



VAR_INPUT

```
VAR_INPUT
    fManValue       : FLOAT;
    eMode       : E_CTRL_MODE;
END_VAR
```

fManValue : Input whose value is present at the output in manual mode.

eMode : Input that specifies the block's operating mode.

VAR_OUTPUT

```
VAR_OUTPUT
    fOut        : FLOAT;
    eState      : E_CTRL_STATE;
    eErrorId        : E_CTRL_ERRORCODES;
    bError      : BOOL;
END_VAR
```

fOut : Output of the signal generator.

eState : State of the function block.

eErrorId : Supplies the error number [▶ 16] when the bError output is set.

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

VAR_IN_OUT

```
VAR_IN_OUT
    stParams        : ST_CTRL_SIGNAL_GENERATOR_PARAMS;
END_VAR
```

stParams : Parameter structure of the function block. This consists of the following elements:

```
TYPE ST_CTRL_I_PARAMS
:
STRUCT
    tCtrlCycleTime : TIME  :=  T#0ms; (* controller cycle time
*)
    tTaskCycleTime : TIME  :=  T#0ms; (* task cycle time
*)
    eSignalType    : E_CTRL_SIGNAL_TYPE;
    tCylceDuration : TIME;
    fAmplitude     : FLOAT;
    fOffset    : FLOAT := 0.0;
    tStart     : TIME  := T#0s;
END_STRUCT
END_TYPE
```

tCtrlCycleTime : Cycle time with which the control loop is processed. This must be greater than or equal to the TaskCycleTime. The function block uses this input value to calculate internally whether the state and the output values have to be updated in the current cycle.

tTaskCycleTime: Cycle time with which the function block is called. If the function block is called in every cycle this corresponds to the task cycle time of the calling task.

eSignalType : Selection of the signal form. See E_CTRL_SIGNAL_TYPE below.

```
TYPE
E_CTRL_SIGNAL_TYPE :
    (
    eCTRL_TRIANGLE  := 0,
    eCTRL_SINUS     := 1,
    eCTRL_SAWTOOTH  := 2
    );
END_TYPE
```

tCycleDuration : Period of the generated signal curve.

fAmplitude : Amplitude of the generated signal curve.

fOffset : Offset to be added to the signal curve.

tStart : Point in time within a period at which the signal curve starts when switching to eCTRL_MODE_ACTIVE.

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.8	PC (i386)	TcControllerToolbox.lib
TwinCAT v2.9 from Build 947	BC [▶ 10]	TcControllerToolbox.lb6
TwinCAT v2.9 from Build 956	BX [▶ 10]	TcControllerToolbox.lbx

6 Example project

The ways in which the blocks in the TcControllerToolbox function are illustrated in an example project. This project contains programs using the individual function blocks, with which the basic functionality is illustrated. The illustrations include some examples and also the construction of more complex control systems. The control system is simulated for all of these programs, so that no hardware is necessary for the project.

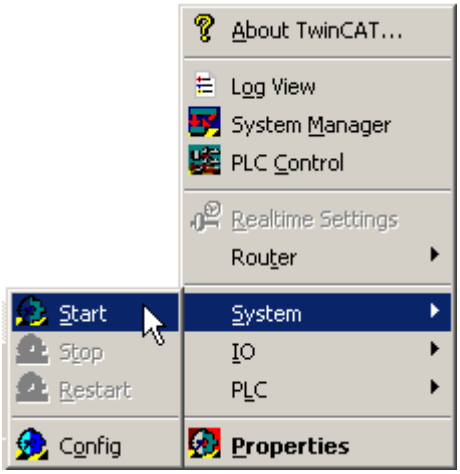
6.1 Install and start the example project

The TcControllerToolbox offers the programmer transfer elements with which a very wide variety of controllers can be implemented.

- Save and unpack the sample program from https://infosys.beckhoff.com/content/1033/tcplccontrollertoolbox/Resources/11282752267/.zip.

The sample program uses simulations of controlled systems, and can thus be run on any Windows NT PC without additional hardware.

- The TwinCAT system is started with the TwinCAT icon
  -  in the task bar.



The *TcControllerToolbox_Examples.pro* sample program is loaded into TwinCAT PLC Control, compiled and started.

- Load the PRO file.
- Compile the PLC project via the menu *Project - Rebuild All*.
- Load the PLC project into the runtime system via the menu option *Online - Login*.
- Start the program via the menu option *Online - Run*.

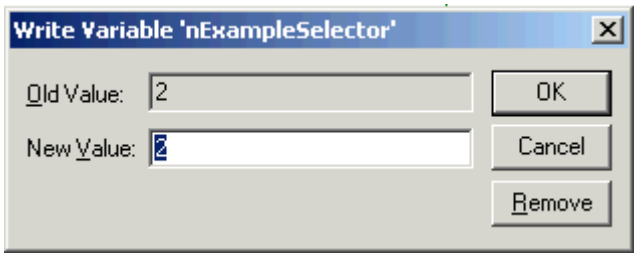
TwinCAT ScopeView allows the signal curves from the individual examples to be displayed graphically. The settings supplied in *TcControllerToolboxExamples_Scope_x.scp* can be used for this purpose. This specification of the Scope file to be used for each particular example can be found in the comments in the project's MAIN program.

- Load the SCP file into TwinCAT ScopeView.
- Start recording via the menu or via the F5 key.

Because the example project contains a number of different examples, it is necessary to set the variable nExampleSelector to the appropriate example number in the MAIN program.

- Double-click on the nExampleSelector variable.

- Enter the number of the example. OK.



- Press the F7 key, or click "Force Values" in the online menu.

6.2 Assigning the program numbers

The programs are selected through the *nExampleSelector* variable in MAIN.

Base

Function block to which the example relates	nExampleSelector	Scope File
FB_CTRL_P	1	TcControllerToolboxExamples_Scope_1.scp
FB_CTRL_I	2	TcControllerToolboxExamples_Scope_1.scp
FB_CTRL_D	3	TcControllerToolboxExamples_Scope_1.scp
FB_CTRL_HYSTERESIS	4	TcControllerToolboxExamples_Scope_2.scp
FB_CTRL_TRANSFERFUNCTION_1	5	TcControllerToolboxExamples_Scope_1.scp
FB_CTRL_TRANSFERFUNCTION_2	6	TcControllerToolboxExamples_Scope_1.scp

Controller

Function block to which the example relates	nExampleSelector	Scope File
FB_CTRL_2POINT	10	TcControllerToolboxExamples_Scope_3.scp
FB_CTRL_3POINT	11	TcControllerToolboxExamples_Scope_4.scp
FB_CTRL_3POINT_EXT	12	TcControllerToolboxExamples_Scope_5.scp
FB_CTRL_nPOINT	13	TcControllerToolboxExamples_Scope_5.scp
FB_CTRL_PI	14	TcControllerToolboxExamples_Scope_5.scp
FB_CTRL_PID	15	TcControllerToolboxExamples_Scope_5.scp
FB_CTRL_PID_EXT	16	TcControllerToolboxExamples_Scope_5.scp

Filter / Controlled System Simulation

Beispiel zu dem Funktions-baustein	nExampleSelector	Scope File
FB_CTRL_ACTUAL_VALUE_FILTER	20	TcControllerToolboxExamples_Scope_1.scp
FB_CTRL_ARITHMETIC_MEAN	21	TcControllerToolboxExamples_Scope_1.scp
FB_CTRL_MOVING_AVERAGE	22	TcControllerToolboxExamples_Scope_1.scp
FB_CTRL_PT1	23	TcControllerToolboxExamples_Scope_1.scp
FB_CTRL_PT2	24	TcControllerToolboxExamples_Scope_1.scp
FB_CTRL_PT2oscillation	25	TcControllerToolboxExamples_Scope_1.scp
FB_CTRL_PT3	26	TcControllerToolboxExamples_Scope_1.scp
FB_CTRL_PTn	27	TcControllerToolboxExamples_Scope_1.scp
FB_CTRL_PTt	28	TcControllerToolboxExamples_Scope_1.scp
FB_CTRL_TuTg	29	TcControllerToolboxExamples_Scope_1.scp

Interpolation

Function block to which the example relates	nExampleSelector	Scope File
FB_CTRL_INTERPOLATION	30	TcControllerToolboxExamples_Scope_1.scp
FB_CTRL_NORMALIZE	31	TcControllerToolboxExamples_Scope_7.scp

Monitoring / Alarming

Function block to which the example relates	nExampleSelector	Scope File
FB_CTRL_CHECK_IF_IN_BAND	40	TcControllerToolboxExamples_Scope_8.scp
FB_CTRL_LOG_DATA	41	

Output to Controlling Equipment

Function block to which the example relates	nExampleSelector	Scope File
FB_CTRL_DEADBAND	50	TcControllerToolboxExamples_Scope_1.scp
FB_CTRL_LIMITER	51	TcControllerToolboxExamples_Scope_1.scp
FB_CTRL_PWM_OUT	52	TcControllerToolboxExamples_Scope_9.scp
FB_CTRL_PWM_OUT_EXT	53	TcControllerToolboxExamples_Scope_9.scp
FB_CTRL_MULTIPLE_PWM_OUT	54	TcControllerToolboxExamples_Scope_10.scp

Function block to which the example relates	nExampleSelector	Scope File
FB_CTRL_SCALE	55	TcControllerToolboxExamples_Scope_1.scp

Setpointgeneration

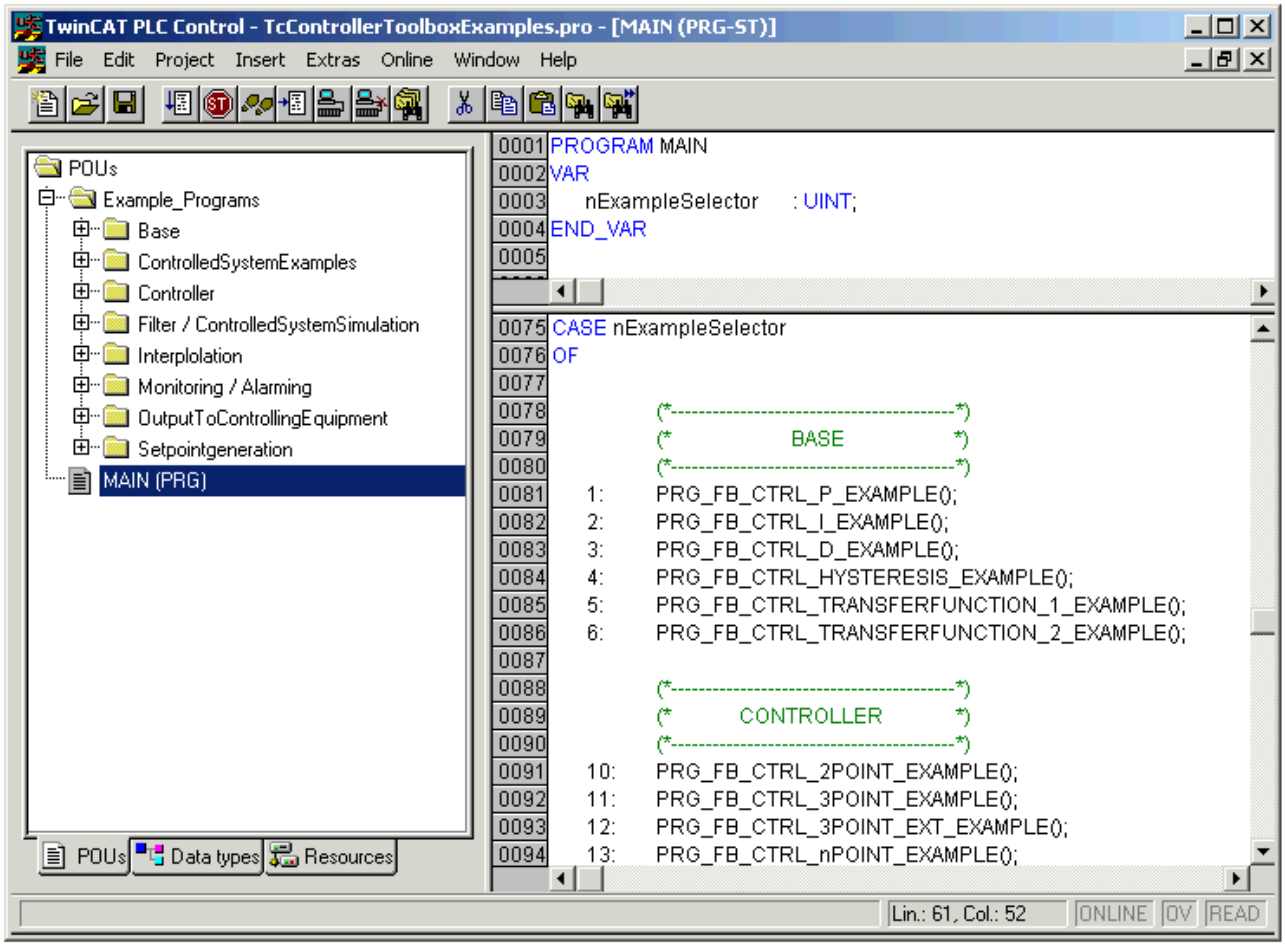
Function block to which the example relates	nExampleSelector	Scope File
FB_CTRL_RAMP_GENERATOR	60	TcControllerToolboxExamples_Scope_1.scp
FB_CTRL_SETPOINT_GENERATOR	61	TcControllerToolboxExamples_Scope_1.scp
FB_CTRL_SIGNAL_GENERATOR	62	TcControllerToolboxExamples_Scope_1.scp

Controlled System Examples

Function block to which the example relates	nExampleSelector	Scope File
FB_CTRL_PID_SPLITRANGE	80	TcControllerToolboxExamples_Scope_6.scp
FB_CTRL_PI_PID	81	TcControllerToolboxExamples_Scope_11.scp

6.3 Program Structure

The `Main` module calls up the corresponding sample program in accordance with the variable `nExampleSelector`.



The comments in the individual programs should make them clear and easy to understand.

More Information:
www.beckhoff.com/ts4100

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