

Manual | EN

TS8000

TwinCAT 2 | PLC HVAC



Table of contents

1	Foreword	9
1.1	Notes on the documentation	9
1.2	For your safety	10
1.3	Notes on information security.....	11
2	Introduction	12
2.1	Target groups	12
2.2	User requirement profile.....	12
2.3	General technical characteristics	13
2.3.1	Integration into TwinCAT.....	13
2.3.2	Hardware requirements.....	13
2.3.3	Remanent data.....	13
2.3.4	Default values	13
2.3.5	Value range monitoring	13
3	Function blocks	14
3.1	HVAC Actuators	19
3.1.1	FB_HVAC2PointActuator	19
3.1.2	FB_HVAC3PointActuator	21
3.1.3	FB_HVACCirculationPump	23
3.1.4	FB_HVACCirculationPumpEx	26
3.1.5	FB_HVACMotor1Speed	37
3.1.6	FB_HVACMotor2Speed	41
3.1.7	FB_HVACMotor3Speed	47
3.1.8	FB_HVACRedundancyCtrl.....	54
3.1.9	FB_HVACRedundancyCtrlEx.....	57
3.2	HVAC Analog modules	59
3.2.1	FB_HVACAnalogInput	59
3.2.2	FB_HVACAnalogOutput.....	62
3.2.3	FB_HVACAnalogOutputEx	64
3.2.4	FB_HVACAnalogTo3Point.....	68
3.2.5	FB_HVACConfigureKL32xx	74
3.2.6	FB_HVACScale.....	78
3.2.7	FB_HVACScale_nPoint	80
3.2.8	FB_HVACTemperatureCurve	84
3.2.9	FB_HVACTemperatureSensor.....	89
3.2.10	FB_HVACTemperatureSensorEx	92
3.2.11	FB_HVACTemperatureSensorEx	95
3.3	HVAC Room Functions	99
3.3.1	Air conditioning.....	99
3.3.2	Controller.....	112
3.3.3	Lighting.....	116
3.3.4	Sun portection	144
3.3.5	Program example.....	229
3.3.6	Structures and enumeration.....	231
3.3.7	List descriptions	237

3.4	HVAC Controller.....	246
3.4.1	FB_HVAC2PointCtrl.....	246
3.4.2	FB_HVACI_CtrlStep.....	248
3.4.3	FB_HVACI_CtrlStepEx	258
3.4.4	FB_HVACPIDCtrl.....	267
3.4.5	FB_HVACPIDCtrl_Ex.....	269
3.4.6	FB_HVACPowerRangeTable.....	275
3.4.7	Sequence-Controller	287
3.5	HVAC Setpoint modules	326
3.5.1	FB_HVACHeatingCurve.....	326
3.5.2	FB_HVACHeatingCurveEx	328
3.5.3	FB_HVACOutsideTempDamped	332
3.5.4	FB_HVACSetpointHeating.....	333
3.5.5	FB_HVACSetpointRamp.....	335
3.5.6	FB_HVACSummerCompensation.....	337
3.6	HVAC Special functions	340
3.6.1	FB_HVACAirConditioning2Speed.....	340
3.6.2	FB_HVACAlarm	344
3.6.3	FB_HVACAntiBlockingDamper	345
3.6.4	FB_HVACAntiBlockingPump	347
3.6.5	FB_HVACBlink.....	350
3.6.6	FB_HVACCmdCtrl_8	351
3.6.7	FB_HVACCmdCtrlSystem1Stage.....	361
3.6.8	FB_HVACCmdCtrlSystem2Stage	369
3.6.9	FB_HVACConvertEnum.....	383
3.6.10	FB_HVACEnthalpy.....	384
3.6.11	FB_HVACFixedLimit	386
3.6.12	FB_HVACFreezeProtectionHeater	388
3.6.13	FB_HVACMux8.....	390
3.6.14	FB_HVACMUX_INT_16.....	392
3.6.15	FB_HVACMUX_INT_8.....	396
3.6.16	FB_HVACMUX_REAL_16	400
3.6.17	FB_HVACMUX_REAL_8	404
3.6.18	FB_HVACOverwriteAnalog.....	407
3.6.19	FB_HVACOverwriteDigital	408
3.6.20	FB_HVACPowerMeasurementKL3403.....	408
3.6.21	FB_HVACPowerMeasurementKL3403Ex.....	411
3.6.22	FB_HVACPriority_INT_16.....	414
3.6.23	FB_HVACPriority_INT_8.....	419
3.6.24	FB_HVACPriority_REAL_16	423
3.6.25	FB_HVACPriority_REAL_8.....	428
3.6.26	FB_HVACOptimizedOn.....	431
3.6.27	FB_HVACOptimizedOff.....	441
3.6.28	FB_HVACTempChangeFunction	452
3.6.29	FB_HVACPWM.....	454
3.6.30	FB_HVACStartAirConditioning.....	457

3.6.31	FB_HVACSummerNightCooling	461
3.6.32	FB_HVACSummerNightCoolingEx	464
3.6.33	FB_HVACTimeCon	470
3.6.34	FB_HVACTimeConSec	471
3.6.35	FB_HVACTimeConSecMs	471
3.6.36	FB_HVACWork	472
3.7	HVAC Time schedule	474
3.7.1	FB_HVACScheduler1ch	474
3.7.2	FB_HVACScheduler7ch	477
3.7.3	FB_HVACScheduler7TCHandling	481
3.7.4	FB_HVACScheduler28ch	482
3.7.5	FB_HVACScheduler28TCHandling	486
3.7.6	FB_HVACSchedulerSpecialPeriods	486
3.7.7	FB_HVACSchedulerPublicHolidays	489
3.8	HVAC System	492
3.8.1	FB_HVACGetSystemTime	492
3.8.2	FB_HVACNOVRAMDataHandling	493
3.8.3	FB_HVACPersistentDataHandling	497
3.8.4	FB_HVACPersistentDataFileCopy	499
3.8.5	FB_HVACSetLocalTime	500
3.8.6	FB_HVACSystemTaskInfo	502
4	Backup Function blocks	503
4.1	BackupVar NOVRAM	503
4.1.1	FB_HVACNOVRAM_Bool	503
4.1.2	FB_HVACNOVRAM_Byte	503
4.1.3	FB_HVACNOVRAM_Dint	504
4.1.4	FB_HVACNOVRAM_Dword	504
4.1.5	FB_HVACNOVRAM_Int	505
4.1.6	FB_HVACNOVRAM_Lreal	505
4.1.7	FB_HVACNOVRAM_Real	505
4.1.8	FB_HVACNOVRAM_Sint	506
4.1.9	FB_HVACNOVRAM_Time	506
4.1.10	FB_HVACNOVRAM_Udint	506
4.1.11	FB_HVACNOVRAM_Uint	507
4.1.12	FB_HVACNOVRAM_Usint	507
4.1.13	FB_HVACNOVRAM_Word	507
4.2	BackupVar Persistent	508
4.2.1	FB_HVACPersistent_Bool	508
4.2.2	FB_HVACPersistent_Byte	509
4.2.3	FB_HVACPersistent_Dint	509
4.2.4	FB_HVACPersistent_Dword	509
4.2.5	FB_HVACPersistent_Int	510
4.2.6	FB_HVACPersistent_Lreal	510
4.2.7	FB_HVACPersistent_Real	510
4.2.8	FB_HVACPersistent_Sint	511
4.2.9	FB_HVACPersistent_String	511

4.2.10	FB_HVACPersistent_Struct	512
4.2.11	FB_HVACPersistent_Time.....	513
4.2.12	FB_HVACPersistent_Udint	513
4.2.13	FB_HVACPersistent_Uint	514
4.2.14	FB_HVACPersistent_Usint	514
4.2.15	FB_HVACPersistent_Word.....	514
5	Functions	516
5.1	F_RoundLREAL	516
5.2	F_RoundLREAL_EX	516
6	Enumerations and Structures	517
6.1	E_HVAC2PointActuatorMode	517
6.2	E_HVAC2PointCtrlMode	517
6.3	E_HVAC3PointActuatorMode	517
6.4	E_HVACActuatorMode	517
6.5	E_HVACAirConditioning2SpeedMode	518
6.6	E_HVACAnalogOutputMode.....	518
6.7	E_HVACAntiBlockingMode	518
6.8	E_HVACBusTerminal_KL32xx.....	518
6.9	E_HVACCtrlMode	519
6.10	E_HVACConvectionMode	520
6.11	E_HVACDataSecurityType	520
6.12	E_HVACErrorCodes	520
6.13	E_HVACExternalMode.....	520
6.14	E_HVACExternalRequestMode	521
6.15	E_HVACPlantMode.....	521
6.16	E_HVACPowerMeasurementMode.....	521
6.17	E_HVACReferencingMode	521
6.18	E_HVACRegOutsideTemp.....	521
6.19	E_HVACReqPump.....	521
6.20	E_HVACRegValve	522
6.21	E_HVACSensorType	522
6.22	E_HVACSequenceCtrlMode	522
6.23	E_HVACSetpointHeatingMode	523
6.24	E_HVACSetpointMode.....	523
6.25	E_HVACState	523
6.26	E_HVACTemperatureCurve.....	524
6.27	E_HVACTemperatureSensorMode.....	524
6.28	ST_HVAC2PointCtrlSequence.....	524
6.29	ST_HVACAggregate	524
6.30	ST_HVACCmdCtrl_8Param.....	525
6.31	ST_HVACCmdCtrl_8State	525
6.32	ST_HVACHoliday.....	525
6.33	ST_HVACI_Ctrl	526
6.34	ST_HVACPeriod	526
6.35	ST_HVACParameterScale_nPoint.....	526

6.36	ST_HVACPowerMeasurement	527
6.37	ST_HVACPowerMeasurementEx	527
6.38	ST_HVACPowerRangeTable	527
6.39	ST_HVACTimeChannel	528
6.40	ST_HVACTempChangeFunction	529
7	Appendix	530
7.1	Calculation of switching time when changing sequence	530
7.2	Example project	531
7.3	VAR_GLOBAL CONSTANT	531
7.4	Table of sequence controller operating modes	531

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

© Beckhoff Automation GmbH & Co. KG, Germany.

The reproduction, distribution and utilization of this document as well as the communication of its contents to others without express authorization are prohibited.

Offenders will be held liable for the payment of damages. All rights reserved in the event of the grant of a patent, utility model or design.

1.2 For your safety

Safety regulations

Read the following explanations for your safety.

Always observe and follow product-specific safety instructions, which you may find at the appropriate places in this document.

Exclusion of liability

All the components are supplied in particular hardware and software configurations which are 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 technology who are familiar with the applicable national standards.

Signal words

The signal words used in the documentation are classified below. In order to prevent injury and damage to persons and property, read and follow the safety and warning notices.

Personal injury warnings

DANGER

Hazard with high risk of death or serious injury.

WARNING

Hazard with medium risk of death or serious injury.

CAUTION

There is a low-risk hazard that could result in medium or minor injury.

Warning of damage to property or environment

NOTICE

The environment, equipment, or data may be damaged.

Information on handling the product



This information includes, for example:
recommendations for action, assistance or further information on the product.

1.3 Notes on information security

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

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

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

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

2 Introduction

Foreword

Comfort, energy efficiency and low operating costs as well as quick return of investment, are demands of today's building automation systems. These requirements can be reached by using integrated control system that can combine all functions of building in to one interoperable platform.

TwinCAT PLC HVAC library includes all needed functionalities for heating, cooling and ventilation systems and also for room automation controls. These functionalities are available as TwinCAT PLC function blocks that can be programmed and combined together to reach energy efficient controlling of building. By using these well tested function blocks, it is easy to realize complex building controls and hence reduce engineering time. Freely programmable controllers and modular I/O-system from Beckhoff, makes it possible to reduce life time costs of the control system when demands of the building are changing.

Room automation function blocks for air-conditioning, lighting and shading can be combined in order to reach energy efficient class A according to EN15232 standard.

This results in the following advantages for the system programmer in creating the system and for the system operator in running operation:

- Fast creation of system programs.
- Fast parameterization and commissioning of the systems.
- Guarantee of a very large range of system functions at all times.
- Transparency of programs (prerequisite for long-term maintainability and expandability of the systems)
- Once created, good reusability of templates for systems or system subassemblies.
- Easy training of staff.
- Easy expansion and modification of existing systems.
- Specifications for a clear, object-oriented structure for the creation of visualization objects in MMI and SCADA systems.
- Programs are easier to document.

2.1 Target groups

This software library is intended for building automation system partners of Beckhoff Automation GmbH. The system partners operate in the field of building automation and are concerned with the installation, commissioning, expansion, maintenance and service of measurement and control systems for the technical equipment of buildings.

2.2 User requirement profile

The user of this library is required to have basic knowledge of the following.

- TwinCAT PLC Control
- TwinCAT System Manager
- PCs and networks
- Structure and features of the Beckhoff Embedded PC and its bus terminals
- Technology of heating, ventilation, air conditioning and sanitary systems
- Relevant safety regulations for building technical equipment

2.3 General technical characteristics

2.3.1 Integration into TwinCAT

Development Environment	Target System	TwinCAT PLC libraries to include	standard TwinCAT PLC libraries to include
TwinCAT v2.10 from Build 1302	PC or CX (x86) CX (ARM)	TcHVAC.lib; TcFloatPC.lib; TcBaseMath.lib; TcMath.lib	TcBase.lib; TcIoFunctions.lib; TcPlcCoupler.lib; TcSystem.lib; TcUtilities.lib

2.3.2 Hardware requirements

The HVAC library is usable on all PC-based hardware platforms. The ideal target platforms for heating, ventilation, air conditioning and sanitary applications are the Embedded PCs from the CX series.

2.3.3 Remanent data

The controllers have either a NOVRAM and/or a flash for saving remanent data.

For automatic saving after a parameter change the IN_OUT variables are monitored for change of their values. In the event of a change, an internal variable in the library is triggered, with which the function blocks FB_HVACNOVRAMDataHandling or FB_HVACPersistentDataHandling are activated.

The IN_OUT variables are saved in a binary file in the flash if *eDataSecurityType:=eHVACDataSecurityType_Persistent*. The prerequisite for this is a single instancing of the function block FB_HVACPersistentDataHandling. Writing of the IN_OUT variables is deactivated if *eDataSecurityType:=eHVACDataSecurityType_Idle*.

Nevertheless, the variables must be allocated or addressed for remanent storage. They will then be saved in the NOVRAM in the event of a change of value. The prerequisite for this is the instantiation of the FB_HVACNOVRAMDataHandling function block, as well as an instance of the respective data type that is to be saved.

Example: an instance of FB_HVACNOVRAM_Byte must be used in the case of a variable of type BYTE.

NOTICE

Flash destruction

An IN_OUT variable that has been declared as persistent must not be written cyclically if *eDataSecurityType:=eHVACDataSecurityType_Persistent*, since otherwise the flash will be prematurely destroyed. Regarding this subject, it is strongly recommended that you read the documentation on the function blocks FB_HVACNOVRAMDataHandling or FB_HVACPersistentDataHandling and familiarize yourself with the examples contained in the documentation!

2.3.4 Default values

Default values are declared inside the function block for all system parameters. When the controller is restarted, a check is made as to whether values already exist in the NOVRAM or flash of the controller. If values exist in the binary file in flash or NOVRAM, they will be written back automatically to the IN_OUT variables during the first cycle. Hence, the controller starts automatically with the last saved remanent data.

The default values can be activated at any time by a rising edge at the input variable *bSetDefault*.

2.3.5 Value range monitoring

A value range is defined for each input parameter of the function block. After the input of a value out of its permissible range the last valid value is automatically written back by the function block. The attempt to input an invalid parameter value is indicated by a TRUE at the output variable *bInvalidParameter*. The variable *bInvalidParameter* can be reset by a rising edge at the input variable *bReset*.

3 Function blocks

Function blocks

HVAC Actuators

Name	Description
FB_HVAC2PointActuator [► 19]	Control two-point valves or two-point dampers
FB_HVAC3PointActuator [► 21]	Control three-point valves or three-point dampers
FB_HVACCirculationPump [► 23]	Control pumps
FB_HVACCirculationPumpEx [► 26]	Control pumps; switch -on conditions of bPump are different in comparison with FB_HVACCirculationPump
FB_HVACMotor1Speed [► 37]	Control a single-speed drive
FB_HVACMotor2Speed [► 41]	Control a two-speed drive
FB_HVACMotor3Speed [► 47]	Control a tree-speed drive
FB_HVACMux8 [► 390]	interpret the FIFO memory of the FB_HVACRedundancyCtrlEx
FB_HVACRedundancyCtrl [► 54]	Control a certain of number (e.g. 8 pumps) dependent on working hours
FB_HVACRedundancyCtrlEx [► 57]	Control a certain of number (e.g. 8 pumps) dependent on working hours; the working hours have to be specified externally as hourly values via VAR_INPUT

HVAC Analog modules

Name	Description
FB_HVACAnalogInput [► 59]	Acquisition analog input signal
FB_HVACAnalogOutput [► 62]	Control continuous actuators
FB_HVACAnalogOutputEx [► 64]	Control continuous actuators with a integrated scale function
FB_HVACAnalogTo3Point [► 68]	Convert an analog signal into a three-point step signal
FB_HVACConfigureKL32xx [► 74]	Parameterization the connected analog sensor via TwinCAT PLC
FB_HVACScale [► 78]	Scale function block
FB_HVACScale_nPoint [► 80]	replication characteristic curves
FB_HVACTemperatureCurve [► 84]	represent a temperature curve
FB_HVACTemperatureSensor [► 89]	Acquisition temperature values in 1/10°C, is matched to the KL320x bus terminals
FB_HVACTemperatureSensorEx [► 92]	Acquisition temperature values in 1/10°C, is matched to the KL320x bus terminals; without the second order filter, therefore with a smoothing function
FB_HVACTemperatureSensorEx2 [► 95]	Acquisition temperature values in 1/10 or 1/100°C, is matched to the KL320x bus terminals; without the second order filter, therefore with a smoothing function

HVAC Controller

Name	Description
FB_HVAC2PointCtrl [► 246]	2-point controller
FB_HVAC2PointCtrlSequence [► 290]	2-point sequence controller
FB_HVACBasicSequenceCtrl [► 298]	general sequence controller
FB_HVACI_CtrlStep [► 248]	The function block serves the sequential control of power generators

Name	Description
FB_HVACI_CtrlStepEx [▶ 258]	The function block serves the sequential control of power generators
FB_HVACMasterSequenceCtrl [▶ 301]	master controller in a AC-plant
FB_HVACPIDCooling [▶ 303]	PID cooling controller
FB_HVACPIDCtrl [▶ 267]	PID-Controller
FB_HVACPIDCtrl_Ext [▶ 269]	PID-Controller extended
FB_HVACPIDDehumidify [▶ 306]	PID dehumidification controller
FB_HVACPIDEnergyRecovery [▶ 309]	PID heat recovery controller
FB_HVACPIDHumidify [▶ 313]	PID humidification controller
FB_HVACPIDMixedAir [▶ 317]	PID mixed air chamber controller
FB_HVACPIDPreHeating [▶ 320]	PID pre-heating controller
FB_HVACPIDReHeating [▶ 323]	PID reheating controller
FB_HVACPowerRangeTable [▶ 275]	represents a power range table and serves the sequential control of power generators such as boilers or refrigeration machines

HVAC Setpoint modules

Name	Description
FB_HVACHeatingCurve [▶ 326]	Calculation the supply temperature depending on the outside temperature with four bases
FB_HVACHeatingCurveEx [▶ 328]	Calculation the supply temperature depending on the outside temperature
FB_HVACOutsideTempDamped [▶ 332]	Calculation the damped outside temperature
FB_HVACSetpointHeating [▶ 333]	Control an heating circuit with different operating modes
FB_HVACSetpointRamp [▶ 335]	Moving setpoint ramp
FB_HVACSummerCompensation [▶ 337]	summer compensation

HVAC Special functions

Name	Description
FB_HVACAirConditioning2Speed [▶ 340]	Controls AC-plant with two-speed fans
FB_HVACAlarm [▶ 344]	alarm function block
FB_HVACAntiBlockingDamper [▶ 345]	prevents the blockage of an damper
FB_HVACAntiBlockingPump [▶ 347]	prevents the blockage of a pump
FB_HVACBlink [▶ 350]	flashing sequence
FB_HVACCmdCtrl_8 [▶ 351]	With the function block can single aggregates of a plant in a certain order sequentially on or be turned off. FB_HVACCmdCtrl_8 can be used as start condition of a ventilation system.
FB_HVACCmdCtrlSystem1Stage [▶ 361]	system switch one-stage
FB_HVACCmdCtrlSystem2Stage [▶ 369]	system switch two-stage
FB_HVACConvertEnum [▶ 383]	converts an Enum into an integer value and vice versa
FB_HVACEnthalpy [▶ 384]	calculate the dew point, the specific enthalpy and the absolute humidity
FB_HVACFixedLimit [▶ 386]	Limit value switch
FB_HVACFreezeProtectionHeater [▶ 388]	freeze protection
FB_HVACMUX_INT_16 [▶ 392]	contains two different types of multiplexers
FB_HVACMUX_INT_8 [▶ 396]	contains two different types of multiplexers

Name	Description
FB_HVACMUX_REAL_16 [▶ 400]	contains two different types of multiplexers
FB_HVACMUX_REAL_8	contains two different types of multiplexers
FB_HVACOverwriteAnalog [▶ 407]	manual overwrite analog
FB_HVACOverwriteDigital [▶ 408]	manual overwrite digital
FB_HVACPowerMeasurementKL3403 [▶ 408]	control a 3-phase power measurement terminal KL/KS 3403
FB_HVACPowerMeasurementKL3403Ex [▶ 411]	compared with the FB_HVACPowerMeasurementKL3403 the results are available in LREAL format. The output is extended by the frequencies of the three phases.
FB_HVACPriority_INT_16 [▶ 414]	can be used to prioritise events or as a multiplexer.
FB_HVACPriority_INT_8 [▶ 419]	can be used to prioritise events or as a multiplexer.
FB_HVACPriority_REAL_16 [▶ 423]	can be used to prioritise events or as a multiplexer.
FB_HVACPriority_REAL_8 [▶ 428]	can be used to prioritise events or as a multiplexer.
FB_HVACOptimizedOn [▶ 431]	Turns the heating/cooling on before the building is occupied with a self adapting timetable
FB_HVACOptimizedOff [▶ 441]	Turns the heating/cooling off before the building is empty with a self adapting timetable
FB_HVACTempChangeFunctionEntry [▶ 452]	Entry function for FB_HVACOptimizedOn / FB_HVACOptimizedOff
FB_HVACPWM [▶ 454]	PWM
FB_HVACStartAirConditioning [▶ 457]	start program for an AC-plant
FB_HVACSummerNightCooling [▶ 461]	Summer night cooling
FB_HVACSummerNightCoolingEx [▶ 464]	Summer night cooling
FB_HVACTimeCon [▶ 470]	converts a TIME variable to three UDINT variables (udiSec, udiMin, udiHour)
FB_HVACTimeConSec [▶ 471]	converts a TIME variable into an UDINT variable (udiSec)
FB_HVACTimeConSecMs [▶ 471]	converts a TIME variable into two UDINT variables (udiSec, udiMs)
FB_HVACWork [▶ 472]	working hours counter

HVAC Time schedule

Name	Description
FB_HVACScheduler1ch [▶ 474]	Weekly time switch with 1 time switch channel
FB_HVACScheduler7ch [▶ 477]	Weekly time switch with 7 time switch channel
FB_HVACScheduler7TCHandling [▶ 481]	this FB can be used to select and modify an individual line from the data array of a weekly timer
FB_HVACScheduler28ch [▶ 482]	Weekly time switch with 28 time switch channel
FB_HVACScheduler28TCHandling [▶ 486]	this FB can be used to select and modify an individual line from the data array of a weekly timer
FB_HVACSchedulerSpecialPeriods [▶ 486]	Yearly timer switch with day, month and exactly time
FB_HVACSchedulerPublicHolidays [▶ 489]	Yearly timer switch with day and month

HVAC System

Name	Description
FB_HVACGetSystemTime [▶ 492]	an internal clock can be implemented in the TwinCAT PLC
FB_HVACNOVRAMDataHandling [▶ 493]	FB, it is necessary to start an instance in the main program

Name	Description
FB_HVACPersistentDataHandling [▶ 497]	FB, it is necessary to start an instance in the main program
FB_HVACPersistentDataFileCopy [▶ 499]	to copy binary data on the local TwinCAT PC or from a remote TwinCAT PC to the local TwinCAT PC
FB_HVACSetLocalTime [▶ 500]	sets the local Windows system time and the date
FB_HVACSystemTaskInfo [▶ 502]	determines system variables of the task

HVAC Backup Function blocks

Name	Description
FB_HVACNOVRAM_xyz [▶ 503]	FBs for standard data types
FB_HVACPersistent_xyz [▶ 508]	FBs for standard data types

Room function Lighting

Name	Description
FB_BARLightActuator [▶ 130]	This function block serves to control a conventional light actuator
FB_BARLightCircuit [▶ 133]	This block represents a simple light circuit without a dimming function
FB_BARLightCircuitDim [▶ 134]	This block represents a light circuit with a dimming function.
FB_BARAutomaticLight [▶ 117]	Function block for an automatic light circuit as used in corridors or sanitary facilities.
FB_BARStairwellAutomatic [▶ 138]	Function block for a stairwell light circuit.
FB_BARTwilightAutomatic [▶ 140]	Automatic twilight function.
FB_BARDaylightControl [▶ 126]	Daylight switch without dimming.
FB_BARConstantLightControl [▶ 119]	constant light control function block

Room function Shading (see also Overview)

[Overview \[▶ 144\]](#)

Name	Description
FB_BARBlindPositionEntry [▶ 160]	Shading protection: Entry of blind-positions
FB_BARSunblindEvent [▶ 197]	This function block serves to preset the position and angle for any desired event.
FB_BARSunblindWeatherProtection [▶ 218]	weather protection function
FB_BARSunblindSwitch [▶ 206]	manual operating mode
FB_BARSunblindScene [▶ 203]	manual operating mode with scenes
FB_BARSunblindTwilightAutomatic [▶ 216]	Automatic twilight function.
FB_BARSunblindThermoAutomatic [▶ 209]	Thermo automatic
FB_BARSunProtectionEx [▶ 221]	Function block for the control of glare protection with the aid of a louvered blind.
FB_BARShadingObjectsEntry [▶ 182]	Shading correction: imported data objects by FB
FB_BARReadShadingObjectsList [▶ 173]	Shading correction: imported data objects by file
FB_BARFacadeElementEntry [▶ 164]	Shading correction: imported data elements by FB
FB_BARReadFacadeElementList [▶ 168]	Shading correction: imported data elements by file
FB_BARShadingCorrection [▶ 176] / FB_BARShadingCorrectionSouth [▶ 179]	Shading correction FB
FB_BARDelayedHysteresis [▶ 163]	This function block represents a threshold switch for brightness

Name	Description
FB_BARWithinRangeAzimuth [▶ 224]	This function block checks whether the current azimuth angle (horizontal position of the sun) lies within the limits entered
FB_BARWithinRangeElevation [▶ 226]	This function block checks whether the current angle of elevation (vertical position of the sun) lies within the limits entered.
FB_BARSunblindPrioritySwitch [▶ 198]	Priority controller for up to 9 positioning telegrams
FB_BARSunblindActuator [▶ 185]/ FB_BARSunblindActuatorEx [▶ 190]	Sunblind Actuator
FB_BARSMISunblindActuator [▶ 196]	SMI Sunblind Actuator
FB_BARRollerBlind [▶ 200]	Rollerblind Actuator
FB_BARSMIRollerBlind [▶ 202]	SMI Rollerblind Actuator

Room functions controller

Name	Description
FB_BARPICtrl [▶ 112]	Simple PI controller with input via the proportional band

Air conditioning room function

Name	Description
FB_BAREnergyLevel [▶ 99]	This function block is for the adaptation of the supply of energy for the use of the building.
FB_BARFanCoil [▶ 101]	This function block maps a 3-speed fan with the corresponding switching hysteresis.
FB_BARFctSelection [▶ 104]	This function block is for enabling room heating or room cooling.
FB_BARSetpointRoom [▶ 107]	This function block assigns a setpoint for cooling operation and another for heating operation to each of the four energy levels.

Overview Library version

Date	Version	Created with TwinCAT Version	Remarks
10/29/2008	1.0.0	V2.10.0 (Build 1328)	first Release
10/29/2009	1.1.0	V2.11.0 (Build 1536)	new FBs (FB_HVACRedundancyCtrlEx; FB_HVACTemperatureSensorEx; FB_HVACEnthalpy; FB_HVACTimeCon; FB_HVACTimeConSec; FB_HVACMux8)
04/12/2010	1.2.7	V2.11.0 (Build 1539)	new FBs (FB_HVACSetLocalTime; FB_HVACAnalogOutputEx; FB_HVACConfigureKL32xx; FB_HVACScale_nPoint; FB_HVACTemperatureCurve) new Function F_HVACRoundLREAL_EX
08/04/2010	1.3.0	V2.11.0 (Build 1547)	new FBs (FB_HVAC2PointCtrlSequence; FB_HVACPowerMeasurementKL3403Ex; FB_HVACScheduler7TCHandling; FB_HVACScheduler28TCHandling)
01/13/2011	1.9.0	V2.11.0 (Build 1552)	new FBs (FB_HVACTimeConSecMs; FB_HVACI_CtrlStep; FB_HVACPowerRangeTable; FB_HVACPriority_REAL_8; FB_HVACPriority_REAL_16; FB_HVACPriority_INT_16; FB_HVACPriority_INT_8; FB_HVACMUX_INT_8;

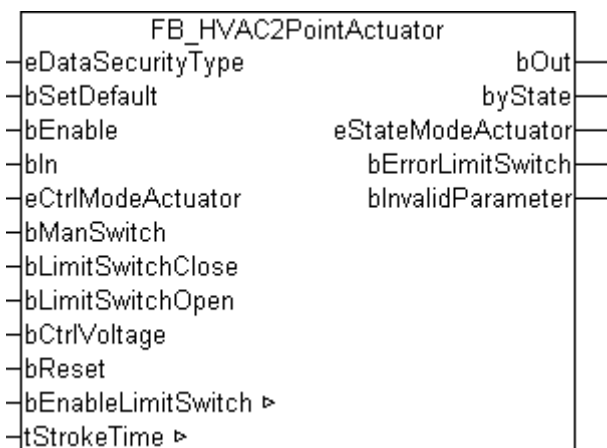
Date	Version	Created with TwinCAT Version	Remarks
			FB_HVACMUX_INT_16; FB_HVACMUX__16; FB_HVACMUX_INT_8; FB_HVACCirculationPumpEx; FB_HVACHeatingCurveEx)
02/08/2011	1.10.1	V2.11.0 (Build 1552)	new FB (FB_HVACConvertEnum)
28/07/2011	1.11.0	V2.11.0 (Build 1552)	new FB_HVACSummerNightCoolingEx
05/12/2011	1.11.12	V2.11.0 (Build 2038)	new FB_HVACTemperatureSensorEx2
28/12/2011	1.12.0	V2.11.0 (Build 2038)	new FB_HVACCmdCtrl_8
30.03.2012	1.13.0	V2.11.0 (Build 2218)	including new FBs for Room functions
30.06.2012	1.14.0	V2.11.0 (Build 2224)	new FB_HVACI_CtrlStepEx

Also see about this

📖 Overview sun protection [▶ 146]

3.1 HVAC Actuators

3.1.1 FB_HVAC2PointActuator



Application


This function block serves to control two-point valves or two-point dampers.

VAR_INPUT

```
eDataSecurityType      : E_HVACDataSecurityType;
bSetDefault            : BOOL;
bEnable                : BOOL;
bIn                    : BOOL;
eCtrlModeActuator      : E_HVAC2PointActuatorMode;
bManSwitch             : BOOL;
bLimitSwitchClose      : BOOL;
bLimitSwitchOpen       : BOOL;
bCtrlVoltage           : BOOL;
bReset                 : BOOL;
```

eDataSecurityType:if eDataSecurityType:= *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled by the PLC program with the input variable *bEnable*. The actuator always remains closed as long as the function block is not enabled. Output *bOut* is permanently FALSE.

bln: in automatic mode the actuator is closed by a FALSE and opened by a TRUE.

eCtrlModeActuator: Enum that defines the operation mode.

bManSwitch: if the two-point drive has a manual/emergency switch in the control cabinet, this can be connected to the input *bManSwitch*; the status of the manual/emergency switch will then be monitored. If *bManSwitch* = FALSE, then output *bOut* of the drive will be set to FALSE.

bLimitSwitchClose: actuator feedback TRUE when the actuator is completely closed.

bLimitSwitchOpen: actuator feedback TRUE when the actuator is completely open.

bCtrlVoltage: the parameter *bCtrlVoltage* serves to check the control voltage. The control voltage is present if the *bCtrlVoltage* variable is TRUE. The feedback control is suppressed if the control voltage fails.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
bOut           : BOOL;
byState        : BYTE;
eStateModeActuator : E_HVAC2PointActuatorMode;
bErrorLimitSwitch : BOOL;
bInvalidParameter : BOOL;
```

bOut: the actuator is connected to this output (FALSE = closing drive; TRUE = opening drive).

byState: displays the status of the control from the actuator:

```
byState.0:= Enable
byState.1:= Manual Switch
byState.2:= Enable Feedback Control
byState.3:= Control Voltage
byState.4:= Reset
byState.5:= bOut
```

eStateModeActuator: indicates in which operation mode the actuator is.

bErrorLimitSwitch: becomes TRUE if no limit switch is triggered after the preset stroke time. *bErrorLimitSwitch* is acknowledged with a positive edge on the input *bReset*.

bInvalidParameter: TRUE if an error occurs during the plausibility check. The message must be acknowledged with *bReset*.

VAR_IN_OUT

```
bEnableLimitSwitch : BOOL;
tStrokeTime        : TIME;
```

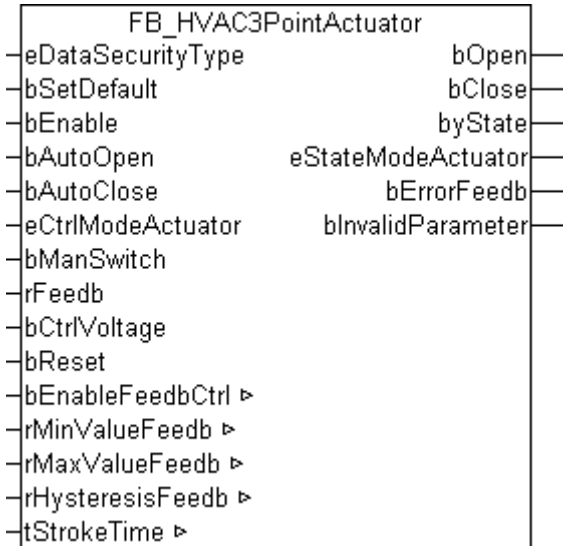
bEnableLimitSwitch: if the input is TRUE, then the function control of the drive is activated by means of the limit switches.

tStrokeTime: in order to set the function control correctly, the stroke time of the drive from fully closed to fully opened drive must be entered here (0s..3600s). The variable is saved persistently. Preset to 200 s.

Documents about this

 example_persistent_e.zip (Resources/zip/11659714827.zip)

3.1.2 FB_HVAC3PointActuator



Application

This function block is used to control three-point valves or three-point dampers with or without continuous position feedback.


The function block is often used in conjunction with the function block [FB_HVACAnalogTo3Point](#). [▶ 68]

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
bAutoOpen         : BOOL;
bAutoClose        : BOOL;
eCtrlModeActuator : E_HVAC3PointActuatorMode;
bManSwitch        : BOOL;
rFeedb            : REAL;
bCtrlVoltage      : BOOL;
bReset            : BOOL;
```

eDataSecurityType:if eDataSecurityType:= *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If eDataSecurityType:= *eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled by the PLC program with the input variable `bEnable`. The three-point actuator always remains fully closed as long as the function block is not enabled. Output `bClose` is permanently TRUE.

bAutoOpen / bAutoClose: in automatic mode, the three-point actuator is controlled by the input variables `bAutoClose` and `bAutoOpen`.

eCtrlModeActuator: Enum that defines the operation mode.

bManSwitch: if the three-point actuator has a manual/emergency switch in the control cabinet, this can be connected to the input `bManSwitch`; the status of the manual/emergency switch will then be monitored. If `bManSwitch = FALSE`, then output `bOut` of the drive will be set to FALSE.

rFeedb: analog position feedback from the actuator (0%..100%).

bCtrlVoltage: parameter to check the control voltage. The control voltage is present if the `bCtrlVoltage` variable is TRUE. The feedback control is suppressed if the control voltage fails.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
bOpen           : BOOL;
bClose          : BOOL;
byState         : BYTE;
eStateModeActuator : E_HVAC3PointActuatorMode;
bErrorFeedb    : BOOL;
bInvalidParameter : BOOL;
```

bOpen: the signal for the opening drive of the three-point actuator is connected to this output.

bClose: the signal for the closing drive of the three-point actuator is connected to this output.

byState: displays the state of the control from the actuator:

`byState.0:= Enable`

`byState.1:= Manual Switch`

`byState.2:= Enable Feedback Control`

`byState.3:= Control Voltage`

`byState.4:= Reset`

eStateModeActuator: indicates in which operation mode the actuator is.

bErrorFeedb: when setting the output `bClose`, the position of the drive must have decreased within the time `tStrokeTime` by at least the amount of `rHysteresisFeedb`.

When setting the output `bOpen` the position of the drive must have increased within the time `tStrokeTime` at least by the amount of `rHysteresisFeedb`.

If the actual position is not within the tolerance range after a positioning command within the specified time, this is signaled with TRUE at the output `bErrorFeedb`. Both outputs `bOpen` and `bClose` become FALSE. The fault is acknowledged by a positive edge on the input `bReset`.

bInvalidParameter: TRUE if an error occurs during the plausibility check. The message must be acknowledged with `bReset`.

VAR_IN_OUT

```
bEnableFeedbCtrl : BOOL;
rMinValueFeedb   : REAL;
rMaxValueFeedb   : REAL;
rHysteresisFeedb : REAL;
tStrokeTime      : TIME;
```

bEnableFeedbCtrl: if the input is TRUE, then the control of the feedback signal is enabled. The variable is saved persistently.

rMinValueFeedb: serves to scale the analog position feedback. rMinValueFeedb contains the value of the analog signal when the actuator is fully closed (0%..100%). The variable is saved persistently. Preset to 0.

rMaxValueFeedb: serves to scale the analog position feedback. rMaxValueFeedb contains the value of the analog signal when the actuator is fully open (0%..100%). The variable is saved persistently. Preset to 0.

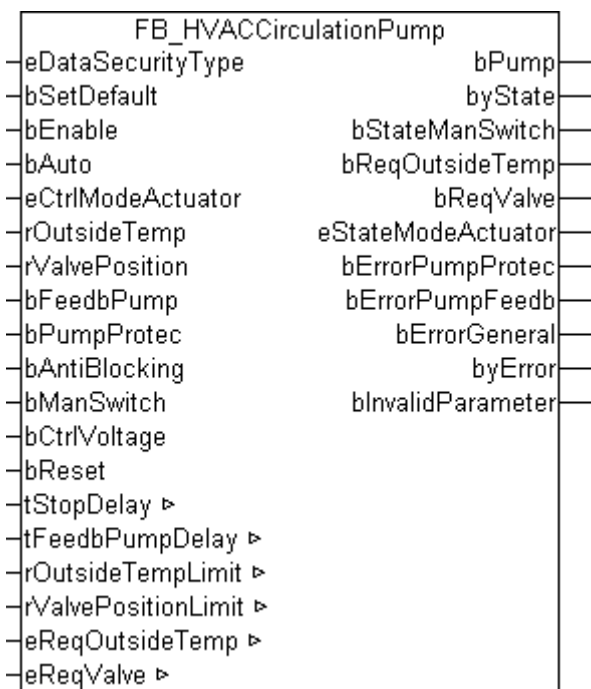
rHysteresisFeedb: due to the stroke time of the drive, the position feedback always lags in the case of a jump in the position setpoint. Using the variable *rHysteresisFeedbCtrl*, a range is specified within which the position setpoint of the actuator can deviate from the feedback signal without the feedback control (bErrorFeedb) being triggered (0%..100%). The variable is saved persistently. Preset to 10.

tStrokeTime: due to the lagging of the actual position in relation to the set position, the activation of the feedback control in the event of the maximum permissible difference being exceeded is delayed by the variable *tStrokeTime* [s]. If the actuator is fully closed and receives a setpoint step-change of 100 %, at least the stroke time of the drive over its entire travel path should be entered as a time (0s..3600s). The variable is saved persistently. Preset to 200 s.

Documents about this

example_persistent_e.zip (Resources/zip/11659714827.zip)

3.1.3 FB_HVACCirculationPump



Application

This function block serves to control pumps in HVAC systems.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
bAuto             : BOOL;
eCtrlModeActuator : E_HVACActuatorMode;
rOutsideTemp      : REAL;
rValvePosition    : REAL;
bFeedbPump        : BOOL;
bPumpProtec       : BOOL;
bAntiBlocking     : BOOL;
```


```

bManSwitch      : BOOL;
bCtrlVoltage    : BOOL;
bReset          : BOOL;

```

eDataSecurityType: if `eDataSecurityType:= eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled if `bEnable = TRUE`.

bAuto: input for the request from the automatic program. The request is overridden by `eCtrlModeActuator`.

eCtrlModeActuator: Enum that defines the operation mode.

rOutsideTemp: input for transmitting the outside temperature value.

rValvePosition: if there is a control valve in the hydraulic circuit of the pump, the position of the control valve must be applied here.

bFeedbPump: this input is for displaying the operating message in a visualization system and for monitoring the pump function.

bPumpProtec: a pump error message is connected to the input `bPumpProtec`. There is a pump error if the input `bPumpProtec` is FALSE. The output `bPump` becomes FALSE in the event of a fault. The pump can only be restarted after an acknowledgment on the input `bReset`.

bAntiBlocking: input for transferring the anti-blocking request, i.e. if TRUE the request is active.

bManSwitch: if the pump has a manual/emergency switch in the control cabinet, this can be connected to the input `bManSwitch`; the status of the manual/emergency switch will then be monitored. If `bManSwitch = FALSE`, then output `bPump` is disabled. The output `bPump` can only be switched on if `bManSwitch = TRUE` (quiescent current principle).

bCtrlVoltage: in order to suppress a torrent of messages, the error message from `bPumpProtec` is only acquired if the input `bCtrlVoltage` is TRUE.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```

bPump           : BOOL;
byState         : BYTE;
bStateManSwitch : BOOL;
bReqOutsideTemp : BOOL;
bReqValve       : BOOL;
eStateModeActuator : E_HVACActuatorMode;
bErrorPumpProtec : BOOL;
bErrorPumpFeedb : BOOL;
bErrorGeneral   : BOOL;
byError         : BYTE;
bInvalidParameter : BOOL;

```


bPump: output variable for controlling a pump.

byState: output of the pump state as byte.

byState.0 := *bEnable*

byState.1 := *bPump*

byState.2 := *bReqOutsideTemp*

byState.3 := *bReqValve*

byState.4 := *bAntiblocking*

byState.5 := *bFeedbPump*

byState.6 := NOT *bManSwitch*

byState.7 := *bCtrlVoltage*

bStateManSwitch: status message of the manual/emergency switch. A TRUE signals that the manual/emergency operating level is activated.

bReqOutsideTemp: if the condition to request the pump depending on the outside temperature is TRUE, the output variable *bReqOutsideTemp* becomes TRUE.

bReqValve: if the condition for requesting the pump depending on the valve position is reached, the variable *bReqValve* becomes TRUE.

eStateModeActuator: indicates in which operation mode the actuator is.

bErrorPumpProtec: error from the pump.

bErrorPumpFeedb: if the input *bFeedbPump* does not go TRUE within the time of *tFeedbPumpDelay* (*tFeedbPumpDelay* must be > t#0s) after setting the output *bPump*, this is recognized as a fault and this output is set to TRUE and the output *bPump* to FALSE. The error message must be acknowledged with *bReset*.

bErrorGeneral: there is a general error.

byError: output of the errors as byte.

byError.1 := *bInvalidParameter*

byError.2 := *bErrorGeneral*

byError.3 := *bErrorPumpProtec*

byError.4 := *bErrorPumpFeedb*

bInvalidParameter: TRUE if an error occurs during the plausibility check. The message must be acknowledged with *bReset*.

VAR_IN_OUT

```
tStopDelay      : TIME;
tFeedbPumpDelay : TIME;
rOutsideTempLimit : REAL;
rValvePositionLimit : REAL;
eReqOutsideTemp : E_HVACReqOutsideTemp;
eReqValve       : E_HVACReqValve;
```

tStopDelay: the time *tStopDelay* [s] delays the switching off of the pump after the switch-on conditions are no longer fulfilled. The variable is saved persistently. Preset to 0 s.

tFeedbPumpDelay: the monitoring function of the pump feedback message [s] is only active if *tFeedbPumpDelay* is > t#0s. If *tFeedbPumpDelay* = t#0s the monitoring function is deactivated (0s..3600s). The variable is saved persistently. Preset to 0 s.

rOutsideTempLimit: value [°C] above or below which the pump is switched on or off depending on the outside temperature (-60 °C..60 °C). The variable is saved persistently. Preset to 10 °C.

rValvePositionLimit: threshold value for the position of a control valve associated with a pump, from which the pump should switch on automatically, e.g. heater pump (0%..100%). The variable is saved persistently. Preset to 3%.

eReqOutsideTemp: depending on the outside temperature, the pump can be compulsorily switched on, e.g. for frost protection purposes when the temperature limit value *rOutsideTempLimit* is undershot. The prerequisite is that *bEnable* = TRUE and that the pump is in automatic mode. The variable is saved persistently.

NOTICE

Manual off overrides the frost protection function!

eReqValve: depending on the position of the valve associated with the pump, the pump can be switched on when the threshold value *rValvePositionLimit* is exceeded. The ENUM activates the switch-on via the valve position. In addition, the ENUM determines whether the temperature-dependent and valve position-dependent switch-on conditions are ORed or ANDed together.

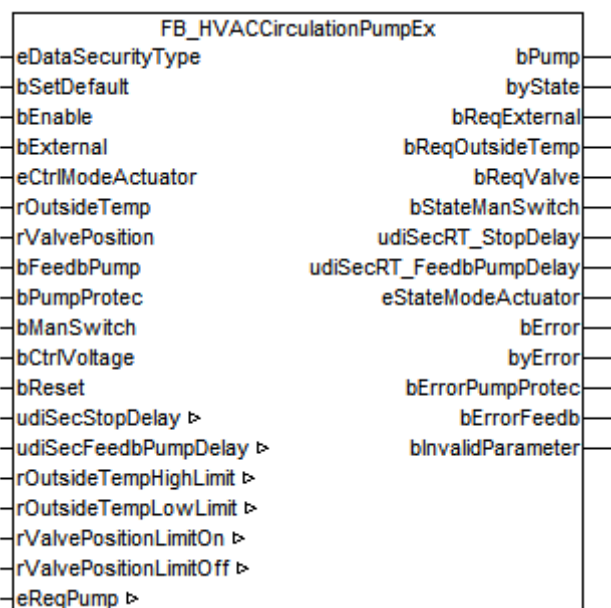
The table below shows a summary of all possible combinations:

eReqOutsideTemp	eReqValve	Function	Application
OTLowerLimit	NoRequest	outsidetemp lower limit	
OTLowerLimit	OrValvePosHigherLimit	outsidetemp OR valve higher limit	Heating circuit, air heater
OTLowerLimit	AndValvePosHigherLimit	outsidetemp AND valve higher limit	
OTHigherLimit	NoRequest	outsidetemp higher limit	
OTHigherLimit	OrValvePosHigherLimit	outsidetemp OR valve higher limit	
OTHigherLimit	AndValvePosHigherLimit	outsidetemp AND valve higher limit	Cooler pump
NoRequest	NoRequest	no request	Primary pump
NoRequest	OrValvePosHigherLimit	valve higher limit	
NoRequest	AndValvePosHigherLimit	not valid	

By means of various combinations of the two variables, *eReqOutsideTemp* and *eReqValve*, this function block can be adapted to the requirements of a heating circuit, an air heater, an air cooler or a feed pump.

Documents about this

📄 example_persistent_e.zip (Resources/zip/11659714827.zip)

3.1.4 FB_HVACCirculationPumpEx**Application**

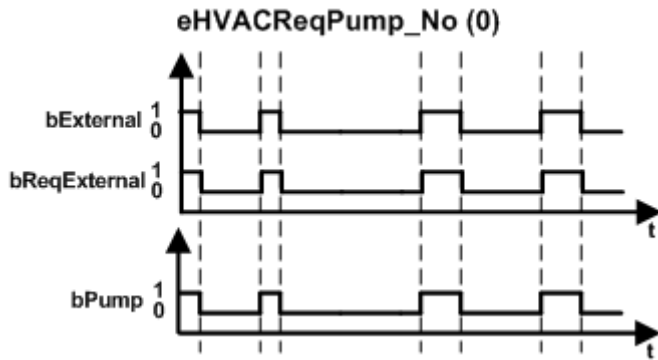
This function block serves to control pumps in HVAC systems.

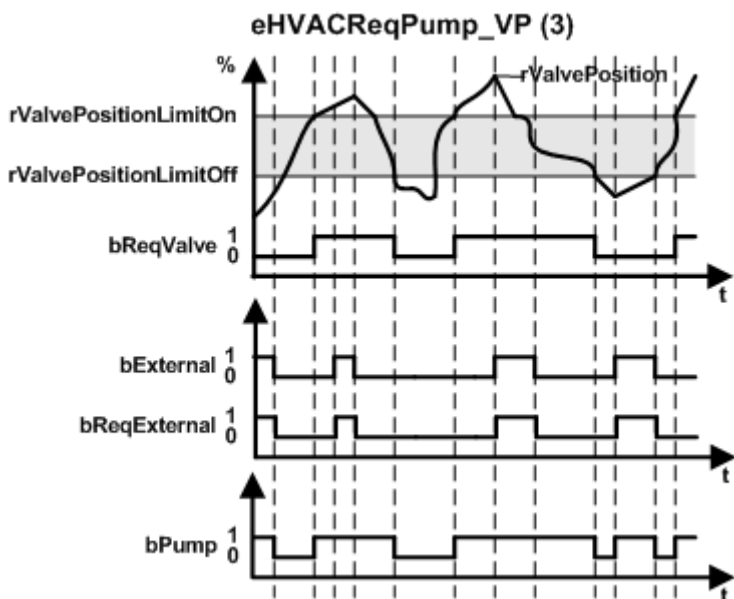
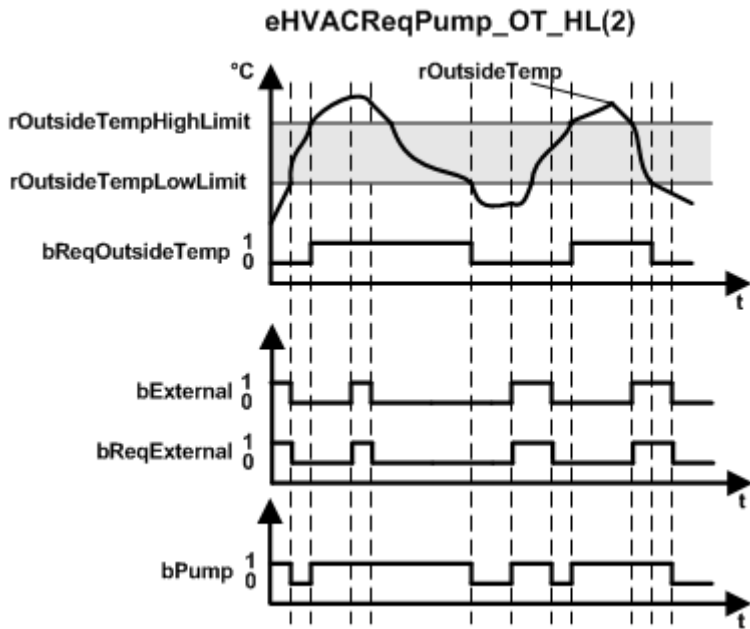
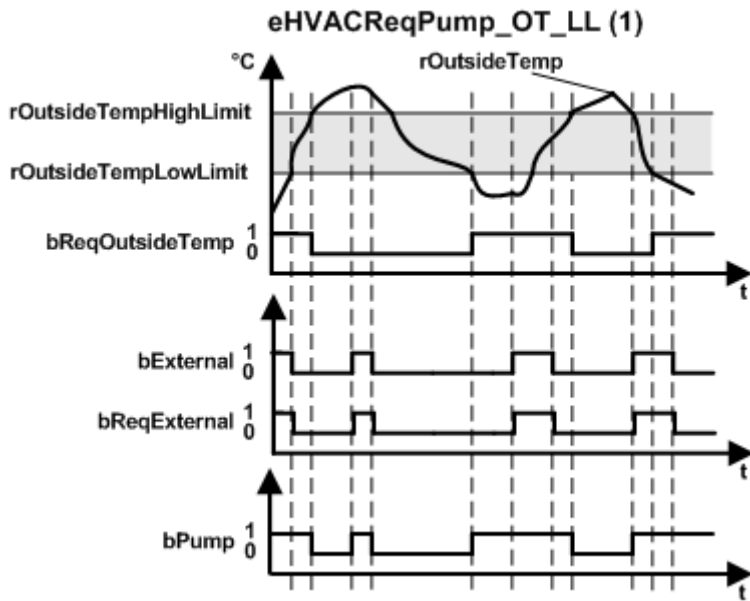
The following switch-on conditions must first be fulfilled in order to control the pump via *bPump*:

$bEnable = TRUE \wedge bErrorPumpProtec = FALSE \wedge bManSwitch = TRUE \wedge bCtrlVoltage = TRUE$

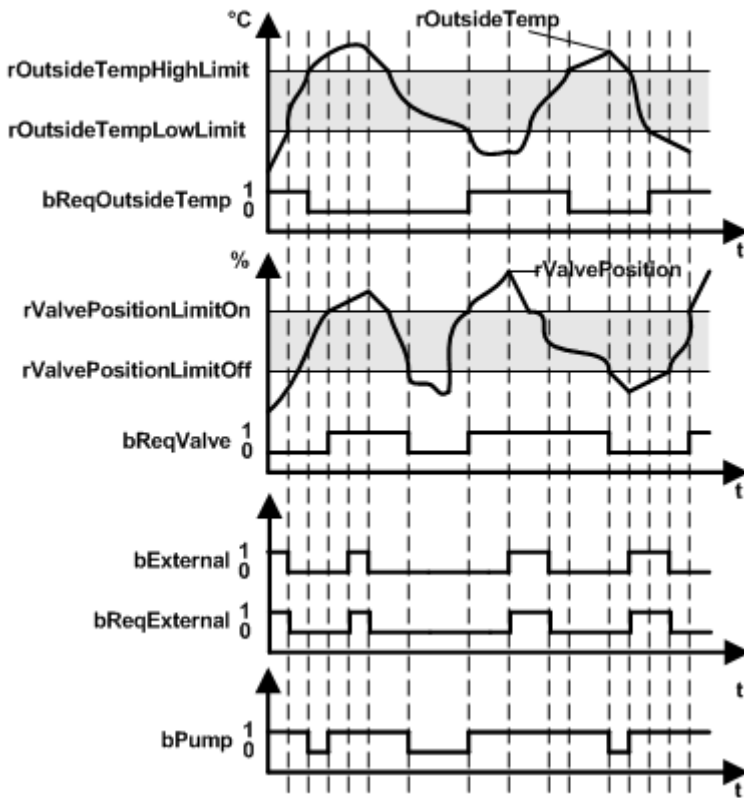
If one of the switch-on conditions is not fulfilled, then the output *bPump* is constantly FALSE.

On the basis of the following diagrams it can be seen how the output *bPump* is controlled depending on the Enum *eReqPump*, the input *bExternal*, the outside temperature *rOutSideTemp* and the valve position *rValvePosition*. The switch-on conditions mentioned above must be fulfilled for this, and one of the two operation modes *eHVACActuatorMode_Auto_BMS* or *eHVACActuatorMode_Auto_OP* must be preselected.

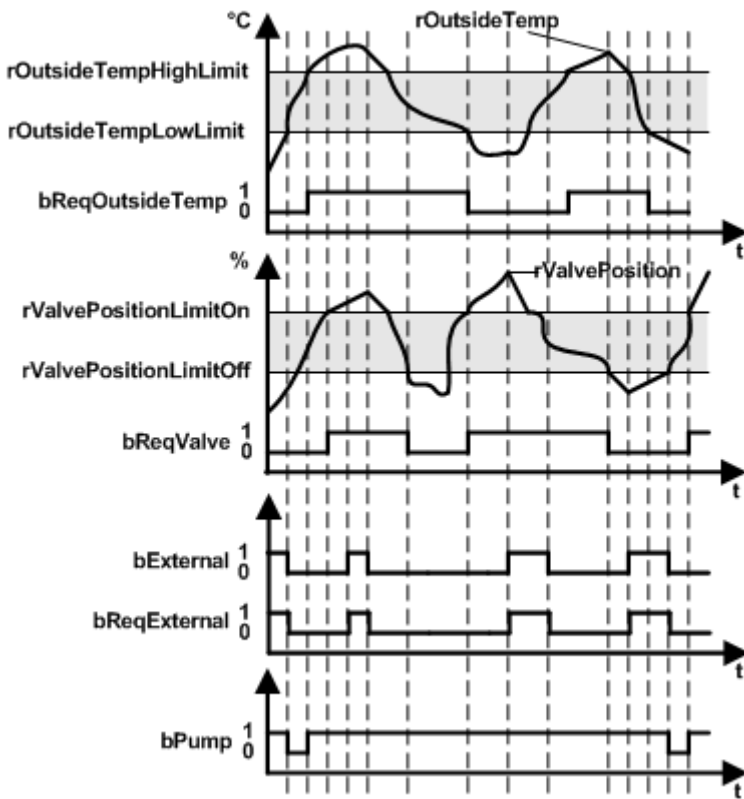




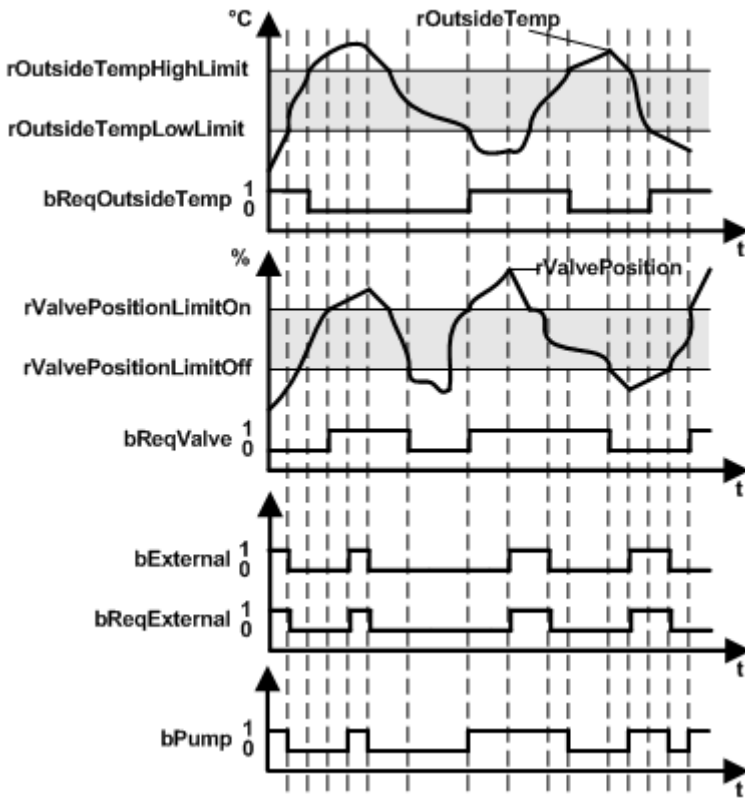
eHVACReqPump_OT_LL_OR_VP (4)



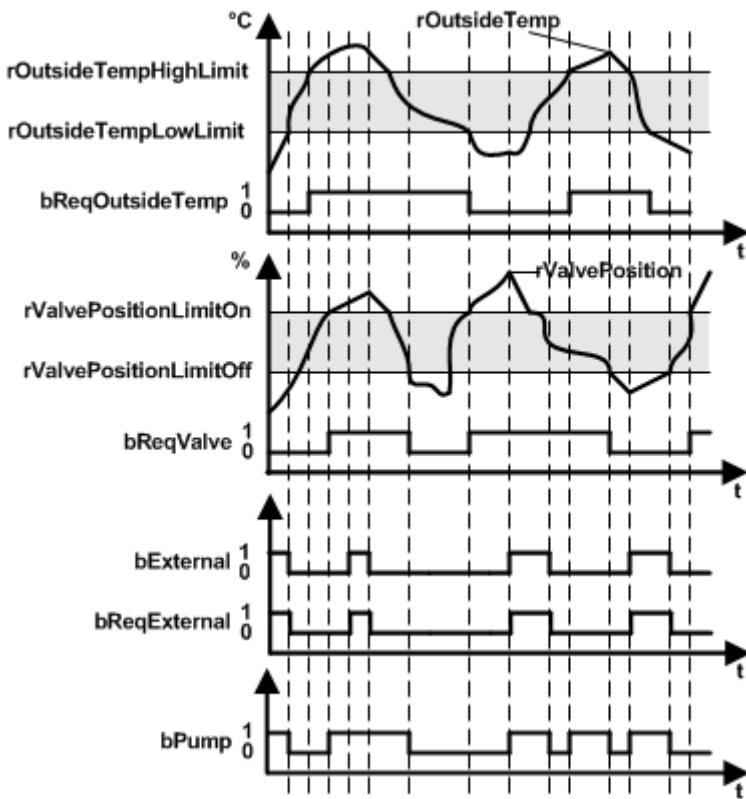
eHVACReqPump_OT_HL_OR_VP (5)



eHVACReqPump_OT_LL_AND_VP (6)



eHVACReqPump_OT_HL_AND_VP (7)



eHVACReqPump_No: there is no request on the part of the Enum to control the pump

eHVACReqPump_OT_LL: the outside temperature (OT = *rOutsideTemp*) must be lower than *rOutsideTempLowLimit* (LL = Lower Limit)

eHVACReqPump_OT_HL: the outside temperature ($OT = rOutsideTemp$) must be higher than $rOutsideTempHighLimit$ (HL = Higher Limit)

eHVACReqPump_VP: the valve position ($VP = rValvePosition$) must be larger than $rValvePositionLimitOn$

eHVACReqPump_OT_LL_OR_VP: the outside temperature ($OT = rOutsideTemp$) must be lower than $rOutsideTempLowLimit$ (LL = Lower Limit) OR the valve position ($VP = rValvePosition$) must be larger than $rValvePositionLimitOn$

eHVACReqPump_OT_HL_OR_VP: the outside temperature ($OT = rOutsideTemp$) must be higher than $rOutsideTempHighLimit$ (HL = Higher Limit) OR the valve position ($VP = rValvePosition$) must be larger than $rValvePositionLimitOn$

eHVACReqPump_OT_LL_AND_VP: the outside temperature ($OT = rOutsideTemp$) must be lower than $rOutsideTempLowLimit$ (LL = Lower Limit) AND the valve position ($VP = rValvePosition$) must be larger than $rValvePositionLimitOn$

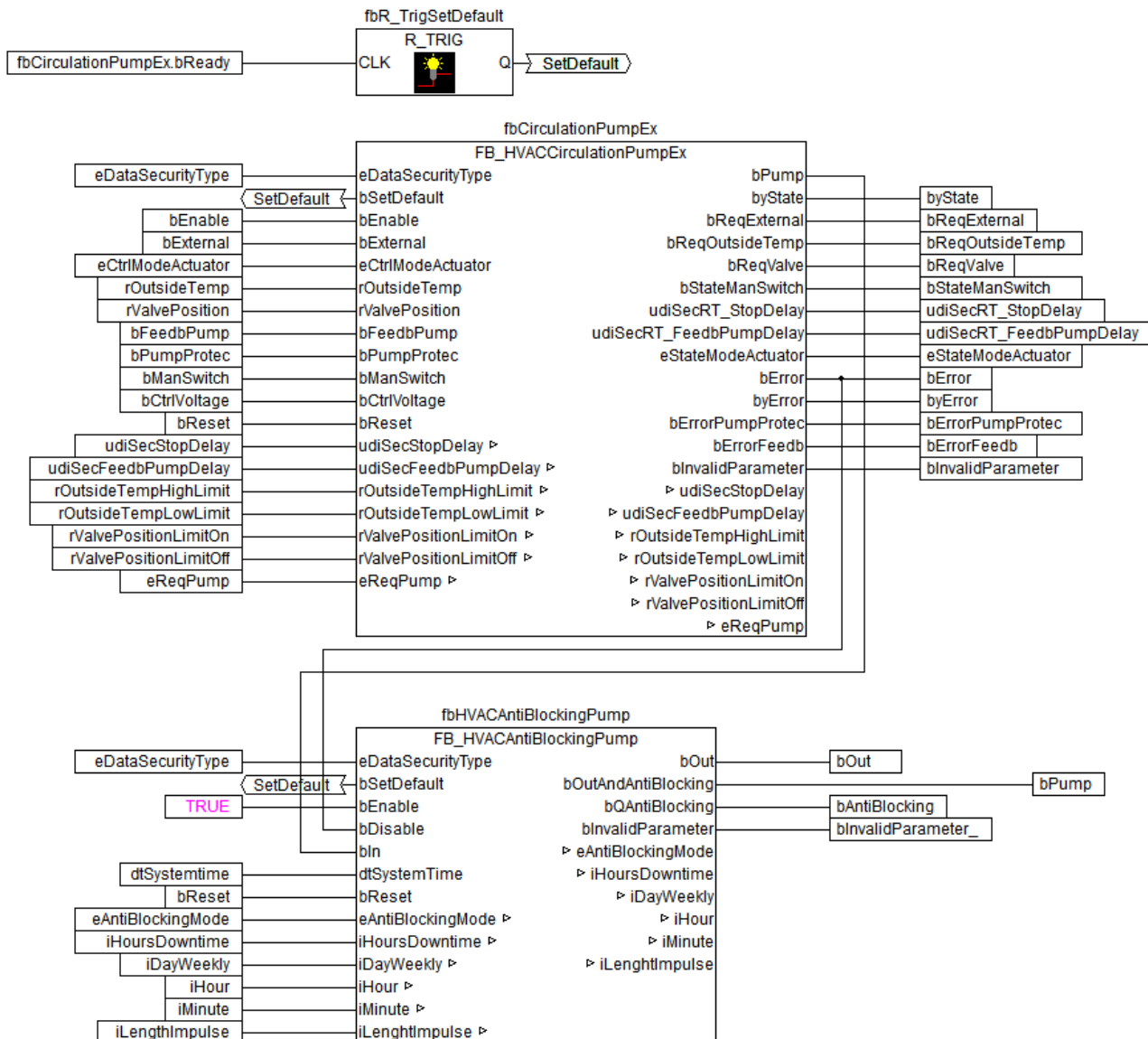
eHVACReqPump_OT_HL_AND_VP: the outside temperature ($OT = rOutsideTemp$) must be higher than $rOutsideTempHighLimit$ (HL = Higher Limit) AND the valve position ($VP = rValvePosition$) must be larger than $rValvePositionLimitOn$

The output $bPump$ switches itself off, after the switch-on conditions are no longer fulfilled, depending on the Enum $eReqPump$, the input $bExternal$, the outside temperature $rOutSideTemp$ or the valve position $rValvePosition$, with a delay set by the time $udiSecStopDelay$.



i $bError$ goes TRUE if $bErrorPumpProtec$ is TRUE. However, only the fault $bErrorPumpProtec$ leads to the deactivation of the output $bPump$. If the error message $bErrorFeedb$ is also to lead to the switch-off of the pump, then the variable must be ANDed with the output $bPump$ after calling the function block. The error message $bErrorFeedb$ is active only in the operation mode $eHVACActuatorMode_Auto_BMS$ OR $eHVACActuatorMode_Auto_OP$ and only if the time $udiSecFeedbPumpDelay$ is greater than 0.

Example anti-blocking protection



Application example

The application example shows the function block FB_HVACirculationPumpEx. The example is illustrated in the programming languages ST and CFC. The program example **P_CFC_CirculationPumpEx.PRG** for the CFC programming languages can be found in the folder **Language CFC > Actuator**, the program example **P_ST_CirculationPumpEx.PRG** for the ST programming languages in the folder **Language Structur Text > Actuator**.

Download	Required library
TcHVAC.pro [▶ 531]	TcHVAC.lib

VAR_INPUT

```

eDataSecurityType      : E_HVACDataSecurityType;
bSetDefault            : BOOL;
bEnable                : BOOL;
bExternal              : BOOL;
eCtrlModeActuator      : E_HVACActuatorMode;
rOutsideTemp           : REAL;
rValvePosition         : REAL;
bFeedbPump            : BOOL;
bPump Protec           : BOOL;
    
```




```

bManSwitch      : BOOL;
bCtrlVoltage    : BOOL;
bReset         : BOOL;

```

eDataSecurityType: if `eDataSecurityType:= eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled if `bEnable = TRUE`. If `bEnable = FALSE`, the `bPump` output is constant FALSE.

bExternal: the output `bPump` can be switched on or off directly via the input `bExternal`. The following conditions must be fulfilled for this: `bEnable = TRUE AND bErrorPump Protec = FALSE AND bManSwitch = TRUE AND bCtrlVoltage = TRUE AND eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP AND eReqPump = eHVACRequestPump_NoRequest`. Otherwise `bExternal` depending on `bReqOutsideTemp` and `bReqValve` controls the output `bPump`, see application [▶ 26].

`bExternal` is only active in the operation mode

`eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP`.

eCtrlModeActuator: Enum that defines the operation mode. The following operation modes are supported by the function block `FB_HVACCirculationPumpEx`: `eHVACActuatorMode_Auto_BMS`, `eHVACActuatorMode_Auto_OP`, `eHVACActuatorMode_Speed1_BMS`, `eHVACActuatorMode_Speed1_OP`, `eHVACActuatorMode_Off_BMS`, `eHVACActuatorMode_Off_OP`.

The two operation modes `eHVACActuatorMode_Auto_BMS`, `eHVACActuatorMode_Auto_OP` mean that the function block is in automatic mode.

The output `bPump` can be switched on directly via the two operation modes

`eHVACActuatorMode_Speed1_BMS`, `eHVACActuatorMode_Speed1_OP` if the following conditions are met: `bEnable = TRUE AND bErrorPump Protec = FALSE AND bManSwitch = TRUE AND bCtrlVoltage = TRUE`

The operation modes `eHVACActuatorMode_Off_BMS`, `eHVACActuatorMode_Off_OP` set the output `bPump` to FALSE.

If an incorrect variable value is present at `eCtrlModeActuator`, then the last valid variable value is taken. The status of the Enum `eCtrlModeActuator` is output via `eStateModeActuator`.

rOutsideTemp: input for transmitting the outside temperature value. Depending on the outside temperature, the pump can be switched on if the temperature falls below or exceeds the limit values

`rOutsideTempLowLimit / rOutsideTempHighLimit`. This depends on the Enum `eReqPump` to request the pump, see application [▶ 26].

`rOutsideTemp` is only active in `eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP` mode.

rValvePosition: input for transmitting the valve position of the control loop. Depending on the position of the valve `rValvePosition` associated with the pump, the pump can be switched on when the threshold value `rValvePositionLimitOn` is exceeded. This depends on the Enum `eReqPump` to request the pump, see application [▶ 26].

`rValvePosition` is only active in the operation mode

`eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP`.

bFeedbPump: feedback from the pump or a relay contact that the pump is switched on. If the output *bPump* = TRUE, then the input *bFeedbPump* must be TRUE within the time specification *udiSecFeedbPumpDelay* and must remain so until *bPump* = FALSE. This error is otherwise indicated via the output variable *bErrorFeedb*. After the rectification of the fault it must be acknowledged at the input *bReset*. The error message *bErrorFeedb* has no influence on the control of the output *bPump*.

bFeedbPump is only active in the operation mode

eHVACActuatorMode_Auto_BMSOReHVACActuatorMode_Auto_OP and if the time *udiSecFeedbPumpDelay* is greater than 0.

If there is no feedback on the pump status, but the function is still implemented, a TRUE must not be permanently applied to the input *bFeedbPump*. This would lead in the switched-off state to an error: *bErrorFeedb* = TRUE. In this case the output *bPump* should be applied to the input *bFeedbPump*.

bPumpProtec: the motor protection for the pump is connected to the input *bPumpProtec*. There is a pump error if the input *bPumpProtec* is FALSE. If there is an error, the output *bPump* goes FALSE and the error is displayed by means of the variable *bErrorPumpProtec*. The output *bPump* can only be switched on if *bPumpProtec* = TRUE (quiescent current principle). After the rectification of the fault it must be acknowledged at the input *bReset*.

bManSwitch: if the pump has a manual / emergency switch, this can be connected to the input *bManSwitch*. The state of the manual / emergency switch is monitored. If *bManSwitch* = FALSE, then output *bPump* is disabled. The output *bPump* can only be switched on if *bManSwitch* = TRUE (quiescent current principle). The state of *bManSwitch* is indicated by the output variable *bStateManSwitch*.

bCtrlVoltage: the control voltage monitoring is applied to the input *bCtrlVoltage*. If *bCtrlVoltage* = FALSE, the *bPump* output is constant FALSE. In order to suppress a torrent of messages, the fault messages *bPumpProtec*, *bErrorFeedb* and *bError* are only acquired if the input *bCtrlVoltage* is TRUE.

bReset: acknowledgment input in the case of a fault following its rectification.

VAR_OUTPUT

```

bPump           : BOOL;
byState         : BYTE;
bReqExternal    : BOOL;      //Request External
bReqOutsideTemp : BOOL;      //Request Outside Temperature
bReqValve       : BOOL;      //Request Valve
bStateManSwitch : BOOL;
udiSecRT_StopDelay : UDINT;  //Second Remaining Time Stop Delay
udiSecRT_FeedbPumpDelay : UDINT; //Second Remaining Time Feedback Pump Delay
eStateModeActuator : E_HVACActuatorMode;
bError          : BOOL;
byError         : BYTE;
bErrorPumpProtec : BOOL;
bErrorFeedb     : BOOL;
bInvalidParameter : BOOL;

```

bPump: output variable for controlling a pump. To control the pump via *bPump* the following conditions must first be fulfilled: *bEnable* = TRUE AND *bErrorPumpProtec* = FALSE AND *bManSwitch* = TRUE AND *bCtrlVoltage* = TRUE. If these conditions are satisfied, the pump can be switched on directly via the operation mode *eCtrlModeActuator* or via various options in automatic mode, see [Application \[▶ 26\]](#). The output *bPump* switches itself off after the switch-on conditions are no longer fulfilled depending, on the Enum *eReqPump*, the input *bExternal*, the outside temperature *rOutSideTemp* or the valve position *rValvePosition*, with a delay set by the time *udiSecStopDelay*.

byState: pump state output

```

byState.0 := bEnable;
byState.1 := bPump;
byState.2 := bReqExternal;
byState.3 := bReqOutsideTemp;
byState.4 := bReqValve;
byState.5 := bFeedbPump;
byState.6 := NOT bManSwitch;
byState.7 := bCtrlVoltage;

```

bReqExternal: If the condition *bExternal* = TRUE to switch on the pump via *bPump* is fulfilled, the output variable *bReqExternal* becomes TRUE. All switch-on conditions and the use of the output *bReqExternal* to control *bPump* are described in the [Application \[▶ 26\]](#).

bReqOutsideTemp: if the condition is achieved for requesting the pump via *bPump* depending on the outside temperature *rOutsideTemp*, the output variable *bReqOutsideTemp* becomes TRUE. All switch-on conditions and the use of the output *bReqOutsideTemp* to control *bPump* are described in the [Application](#) [[▶ 26](#)].

bReqValve: if the condition is achieved for requesting the pump via *bPump* depending on the valve position *rValvePosition*, the output variable *bReqValve* becomes TRUE. All switch-on conditions and the use of the output *bReqValve* to control *bPump* are described in the [Application](#) [[▶ 26](#)].

bStateManSwitch: status message of the manual/emergency switch. A TRUE signals that the manual/emergency operating level is activated. *bStateManSwitch* = NOT *bManSwitch*

udiSecRT_StopDelay: if the *eHVACActuatorMode_Auto_BMSOReHVACActuatorMode_Auto_OP* operation mode is selected, the output *bPump* is switched off after the switch-on conditions are no longer fulfilled, depending on the Enum *eReqPump*, the input *bExternal*, the outside temperature *rOutSideTemp* or the valve position *rValvePosition* after *udiSecRT_StopDelay* has elapsed. The output is given in seconds.

udiSecRT_FeedbPumpDelay: if the output *bPump* = TRUE, then the input *bFeedbPump* must be TRUE within the time *udiSecRT_FeedbPumpDelay* and must remain so until *bPump* = FALSE. Otherwise this error is indicated via the output variable *bErrorFeedb*.

udiSecRT_FeedbPumpDelay is only active in the operation mode *eHVACActuatorMode_Auto_BMSOReHVACActuatorMode_Auto_OP* and if the time *udiSecFeedbPumpDelay* is greater than 0. The output is given in seconds.

eStateModeActuator: indicates the operation mode of the function block. *eStateModeActuator* is equal to *eCtrlModeActuator*.

bError: *bError* becomes TRUE if *bErrorPumpProtec* is TRUE. However, only the fault *bErrorPumpProtec* leads to the deactivation of the output *bPump*. If the fault message *bErrorFeedb* is also to lead to the switch-off of the pump, then the variable must be ANDed with the output *bPump* after calling the function block.

byError: output of the errors as byte.

byError.1 := *bInvalidParameter*

byError.2 := *bError*

byError.3 := *bErrorPumpProtec*

byError.4 := *bErrorFeedb*

bErrorPumpProtec: the motor protection for the pump is connected to the input *bPumpProtec*. There is a pump error if the input *bPumpProtec* is FALSE. If there is an error, the output *bPump* goes FALSE and the error is displayed by means of the variable *bErrorPumpProtec*. The output *bPump* can only be switched on if *bPumpProtec* = TRUE (quiescent current principle). After the rectification of the fault it must be acknowledged at the input *bReset*.

bErrorFeedb: if the output *bPump* = TRUE, then the input *bFeedbPump* must be TRUE within the time *udiSecFeedbPumpDelay* and must remain so until *bPump* = FALSE. This error is otherwise indicated via the output variable *bErrorFeedb*. After the rectification of the fault it must be acknowledged at the input *bReset*. The error message *bErrorFeedb* has no influence on the control of the output *bPump*.

bErrorFeedb is only active in the operation mode

eHVACActuatorMode_Auto_BMSOReHVACActuatorMode_Auto_OP and if the time *udiSecFeedbPumpDelay* is greater than 0.

bInvalidParameter: becomes TRUE if an error occurred during the plausibility check of the following variables: *rOutsideTempHighLimit*, *rOutsideTempLowLimit*, *rValvePositionLimitOn*, *rValvePositionLimitOff*. The message must be acknowledged with *bReset*.

VAR_IN_OUT

<i>udiSecStopDelay</i>	: UDINT;
<i>udiSecFeedbPumpDelay</i>	: UDINT;
<i>rOutsideTempHighLimit</i>	: REAL;
<i>rOutsideTempLowLimit</i>	: REAL;
<i>rValvePositionLimitOn</i>	: REAL;
<i>rValvePositionLimitOff</i>	: REAL;
<i>eReqPump</i>	: E_HVACReqPump;

udiSecStopDelay: the time *udiSecStopDelay* [s] delays the switching off of the pump after the switch-on conditions are no longer fulfilled. It is input in seconds (0s..4294967s). If the operation mode *eHVACActuatorMode_Auto_BMS* OR *eHVACActuatorMode_Auto_OP* is selected, the output *bPump* is switched off after the switch-on conditions are no longer fulfilled depending on the Enum *eReqPump*, the input *bExternal*, the outside temperature *rOutsideTemp* or the valve position *rValvePosition* after the time *udiSecStopDelay* has elapsed. The data is given in seconds. The variable is saved persistently. Preset to 0.

udiSecFeedbPumpDelay: if the output *bPump* = TRUE, then the input *bFeedbPump* must be TRUE within the time *udiSecFeedbPumpDelay* and must remain so until *bPump* = FALSE. Otherwise this error is indicated via the output variable *bErrorFeedb* (0s..4294967s).
udiSecFeedbPumpDelay is only active in the operation mode *eHVACActuatorMode_Auto_BMS* OR *eHVACActuatorMode_Auto_OP* and if the time *udiSecFeedbPumpDelay* is greater than 0. The input is given in seconds. The variable is saved persistently. Preset to 0.

rOutsideTempHighLimit: value above which the pump is switched on or off depending on the outside temperature *rOutsideTemp* and the Enum *eReqPump* (-60 °C..60 °C), see [Application \[► 26\]](#).
rOutsideTempHighLimit is only active in the operation mode *eHVACActuatorMode_Auto_BMS* OR *eHVACActuatorMode_Auto_OP*.
If there is an incorrect variable value at *rOutsideTempHighLimit*, then the last valid variable value is taken. *blInvalidParameter* will be set in the event of an incorrect parameter entry. The variable is saved persistently. Preset to 4.

rOutsideTempLowLimit: value below which the pump is switched on or off depending on the outside temperature *rOutsideTemp* and the Enum *eReqPump* (-60 °C..60 °C), see [Application \[► 26\]](#).
rOutsideTempLowLimit is only active in the operation mode *eHVACActuatorMode_Auto_BMS* OR *eHVACActuatorMode_Auto_OP*.
If there is an incorrect variable value at *rOutsideTempLowLimit*, then the last valid variable value is taken. *blInvalidParameter* will be set in the event of an incorrect parameter entry. The variable is saved persistently. Preset to 1.

rValvePositionLimitOn: threshold value for the position of a control valve *rValvePosition* associated with the pump from which the pump should switch on automatically if exceeded, e.g. heater pump (0%..100%), see [Application \[► 26\]](#).
rValvePositionLimitOn is only active in the operation mode *eHVACActuatorMode_Auto_BMS* OR *eHVACActuatorMode_Auto_OP*.
rValvePositionLimitOn must not be smaller than *rValvePositionLimitOff*. Otherwise the last valid variable value is taken and *blInvalidParameter* is set.
If there is an incorrect variable value at *rValvePositionLimit*, then the last valid variable value is taken. *blInvalidParameter* will be set in the event of an incorrect parameter entry. The variable is saved persistently. Preset to 5.

rValvePositionLimitOff: threshold value for the position of a control valve *rValvePosition* associated with the pump from which the pump is to switch off automatically if the value falls below, e.g. heater pump (0%..100%), see [Application \[► 26\]](#).
rValvePositionLimitOff is only active in the operation mode *eHVACActuatorMode_Auto_BMS* OR *eHVACActuatorMode_Auto_OP*.
rValvePositionLimitOff must not be greater than *rValvePositionLimitOn*. Otherwise the last valid variable value is taken and *blInvalidParameter* is set.
If there is an incorrect variable value at *rValvePositionLimit*, then the last valid variable value is taken. *blInvalidParameter* will be set in the event of an incorrect parameter entry. The variable is saved persistently. Preset to 1.

eReqPump: using the Enum *eReqPump*, switch-on conditions or combinations of switch-on conditions can be set for switching on the pump.
The switch-on conditions are as follows:

- depending on the outside temperature, the pump can be switched on if the temperature falls below or exceeds the limit values *rOutsideTempLowLimit* / *rOutsideTempHighLimit*.
- depending on the position of the valve *rValvePosition* associated with the pump, the pump can be switched on when the threshold value *rValvePositionLimitOn* is exceeded.

In addition, the Enum combinations can be used to specify whether the temperature-dependent and valve-position-dependent switch-on conditions are ORed or ANDed together.

The following switch-on conditions or combinations of switch-on condition can be set via the Enum in order to control the output *bPump*:

eHVACReqPump_No: there is no request on the part of the Enum to control the pump

eHVACReqPump_OT_LL: the outside temperature (OT = *rOutsideTemp*) must be lower than *rOutsideTempLowLimit* (LL = Lower Limit)

eHVACReqPump_OT_HL: the outside temperature (OT = *rOutsideTemp*) must be higher than *rOutsideTempHighLimit* (HL = Higher Limit)

eHVACReqPump_VP: the valve position (VP = *rValvePosition*) must be larger than *rValvePositionLimitOn*

eHVACReqPump_OT_LL_OR_VP: the outside temperature (OT = *rOutsideTemp*) must be lower than *rOutsideTempLowLimit* (LL = Lower Limit) OR the valve position (VP = *rValvePosition*) must be larger than *rValvePositionLimitOn*

eHVACReqPump_OT_HL_OR_VP: the outside temperature (OT = *rOutsideTemp*) must be higher than *rOutsideTempHighLimit* (HL = Higher Limit) OR the valve position (VP = *rValvePosition*) must be larger than *rValvePositionLimitOn*

eHVACReqPump_OT_LL_AND_VP: the outside temperature (OT = *rOutsideTemp*) must be lower than *rOutsideTempLowLimit* (LL = Lower Limit) AND the valve position (VP = *rValvePosition*) must be larger than *rValvePositionLimitOn*

eHVACReqPump_OT_HL_AND_VP: the outside temperature (OT = *rOutsideTemp*) must be higher than *rOutsideTempHighLimit* (HL = Higher Limit) AND the valve position (VP = *rValvePosition*) must be larger than *rValvePositionLimitOn*

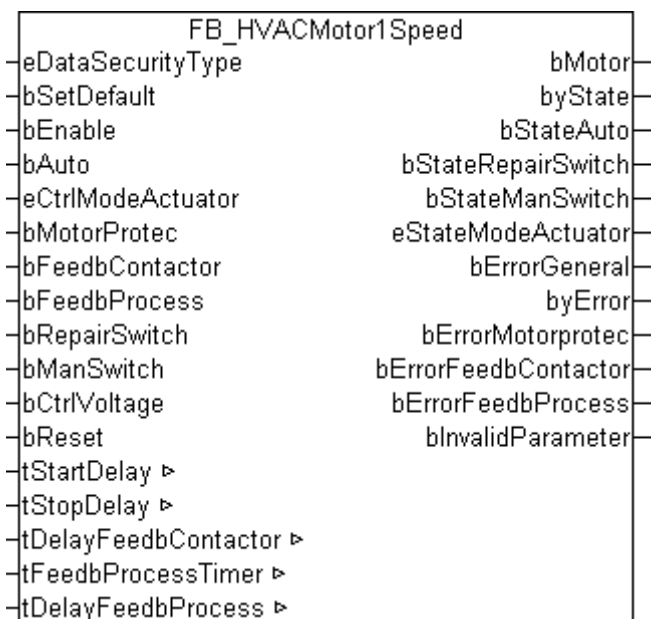
Preconditions for the use of the Enum *eReqPump* are that *bEnable* = TRUE AND *bErrorPump Protec* = FALSE AND *bManSwitch* = TRUE AND *bCtrlVoltage* = TRUE AND *eCtrlModeActuator* = *eHVACActuatorMode_Auto_BMS* OR *eHVACActuatorMode_Auto_OP*.

If there is an incorrect variable value at *eReqPump*, then the last valid variable value is taken. *blInvalidParameter* will be set in the event of an incorrect parameter entry.

Documents about this

- example_persistent_e.zip (Resources/zip/11659714827.zip)

3.1.5 FB_HVACMotor1Speed



Application

This function block serves to control a single-stage drive in HVAC systems. It is suitable for fans.

Application example


Download	Required library
TcHVAC.pro [► 531]	TcHVAC.lib

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable          : BOOL;
bAuto            : BOOL;
eCtrlModeActuator : E_HVACActuatorMode;
bMotorProtec     : BOOL;
bFeedbContactor  : BOOL;
bFeedbProcess    : BOOL;
bRepairSwitch    : BOOL;
bManSwitch       : BOOL;
bCtrlVoltage     : BOOL;
bReset           : BOOL;
```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled if `bEnable = TRUE`. If `bEnable = FALSE`, the drive will be switched off via the specified time variable `tStopDelay`.

bAuto: `bAuto` is only active if the operation mode `eCtrlModeActuator` is either `eHVACActuatorMode_Auto_BMS` or `eHVACActuatorMode_Auto_OP`.

If the input variable `bAuto = TRUE`, then the function block is instructed that the drive should run. If `bAuto = FALSE`, the drive will be switched off with a delay specified in the time variable `tStopDelay`.

eCtrlModeActuator: Enum that specifies the operation mode of the motor. In the event of an incorrect entry, operation continues internally with the last valid operating mode. This is `eHVACActuatorMode_Auto_BMS` in the case of initial commissioning. `bInvalidParameter` will be set in the event of an incorrect parameter entry.

bMotorProtec: input for the motor protection. There is a motor protection fault if the input `bMotorProtec = FALSE` (quiescent current principle). In the event of a fault, the output `bMotor = FALSE`; the fault is indicated at the output of the function block by `bErrorMotorprotec`. The motor can only be restarted if the fault has been rectified and acknowledged at the input `bReset`.

bFeedbContactor: feedback from the power section of the motor. The operating feedback is present if the input `bFeedbContactor = TRUE`. If, after switching on the motor, this feedback is not present after the time delay set by `tDelayFeedbContactor`, the output `bErrorFeedbContactor` is set in order to indicate the fault. The

output *bMotor* becomes FALSE in the event of a fault. The motor can only be restarted if the fault has been rectified and acknowledged at the input *bReset*. If no feedback from the power section of the motor is present, the output variable *bMotor* must be applied to the input *bFeedbContactor*.



If no feedback from the power section of the motor is present, the output variable *bMotor* must be applied to the input *bFeedbContactor*. See application example

bFeedbProcess: a process feedback signal, for example from a v-belt monitor or a flow monitor can be connected to the input *bFeedbProcess*. The process feedback is present if the input *bFeedbProcess* = TRUE (quiescent current principle). If, in the start-up phase after switching on the motor, the process feedback is not present after the time set by *tFeedbProcessTimer*, the drive switches off and indicates a fault at the output *bErrorFeedbProcess*. The motor can only be restarted if the fault has been rectified and acknowledged at the input *bReset*. In order to avoid undesired switching off of the drive during operation due to the process monitoring, e.g. in the event of short-term pressure fluctuations, the triggering of the input *bFeedbProcess* can be delayed by the time *tDelayFeedbProcess*.

bRepairSwitch: the state of the repair switch is monitored with the input *bRepairSwitch*. The motor can only be switched on if *bRepairSwitch*= TRUE (quiescent current principle). If the repair switch is switched off, *bRepairSwitch* = FALSE and the output *bMotor* becomes FALSE.

bManSwitch: the state of the manual/emergency switch is monitored with the input *bManSwitch*. The motor can only be switched on if *bManSwitch* = TRUE (quiescent current principle). If the manual/emergency switch is switched off, *bManSwitch*= FALSE and the output *bMotor* becomes FALSE.

bCtrlVoltage: the control voltage is monitored with the input *bCtrlVoltage*. The motor can only be switched on if *bCtrlVoltage* = TRUE (quiescent current principle). If the control voltage is switched off, *bCtrlVoltage*= = FALSE and the output *bMotor* becomes FALSE. In order to avoid a torrent of error messages if the control voltage fails, the error messages of the function block are suppressed. If the control voltage is restored, the error messages are enabled again.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
bMotor           : BOOL;
byState          : BYTE;
bStateAuto       : BOOL;
bStateRepairSwitch : BOOL;
bStateManSwitch  : BOOL;
eStateModeActuator : E_HVACActuatorMode;
bErrorGeneral    : BOOL;
byError          : BYTE;
bErrorMotorprotec : BOOL;
bErrorFeedbContactor : BOOL;
bErrorFeedbProcess : BOOL;
bInvalidParameter : BOOL;
```

bMotor: output variable for controlling a 1-stage motor.

byState: status byte indicating the operating state of the function block.

```
byState.0 := bEnable;
byState.1 := bMotor;
byState.2 := bStateAuto;
byState.5 := bStateRepairSwitch;
byState.6 := bStateManSwitch;
byState.7 := bCtrlVoltage;
```

bStateAuto: indicates the state for automatic preselection if the operation mode *eCtrlModeActuator* is either *eHVACActuatorMode_Auto_BMS* or *eHVACActuatorMode_Auto_OP* and stage 1 was activated via the input variable *bAuto*.

bStateRepairSwitch: status message of the repair switch. TRUE indicates that the repair switch is switched off.

bStateManSwitch: status message of the manual/emergency switch. A TRUE signals that the manual/emergency operating level is activated.

eStateModeActuator: Enum via which the state of the operation mode of the motor is fed back to the controller.

bErrorGeneral: the error message *bErrorGeneral* becomes TRUE as soon as one of the error messages *bErrorMotorprotec*, *bErrorFeedbContactor* or *bErrorFeedbProcess* = TRUE. The output *bMotor* is then set to FALSE and is only enabled again when the fault has been rectified and acknowledged via *bReset*.

byError: returns all error messages and warnings of the function block.

byError.1 := *blInvalidParameter*;
byError.2 := *bErrorGeneral*;
byError.3 := *bErrorMotorprotec*;
byError.4 := *bErrorFeedbContactor*;
byError.5 := *bErrorFeedbProcess*;

bErrorMotorprotec: error motor protection, see input variable *bMotorProtec*.

bErrorFeedbContactor: power section feedback error, see input variable *bFeedbContactor*.



If no feedback from the power section of the motor is present, the output variable *bMotor* must be applied to the input *bFeedbContactor*. See application example

bErrorFeedbProcess: process feedback error, see input variable *bFeedbackProcess*.

blInvalidParameter: indicates that an incorrect parameter is present at one of the variables *eCtrlModeActuator*, *tStartDelay*, *tStopDelay*, *tDelayFeedbContactor*, *tFeedbProcessTimer* or *tDelayFeedbProcess*. An incorrect parameter specification does not lead to a standstill of the function block; see description of variables. After rectifying the incorrect parameter entry, the message *blInvalidParameter* must be acknowledged via *bReset*.

VAR_IN_OUT

```
tStartDelay      : TIME;
tStopDelay       : TIME;
tDelayFeedbContactor : TIME;
tFeedbProcessTimer : TIME;
tDelayFeedbProcess : TIME;
```

tStartDelay: the start-up of the motor after enabling and switching on via the operation mode of the motor is delayed by the time *tStartDelay* [s] (0s..3600s). The variable is saved persistently. Preset to 0 s. If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *blInvalidParameter* will be set in the event of an incorrect parameter entry.

tStopDelay: the switching off of the motor via the operation mode of the motor is delayed by the time *tStopDelay* [s] (0s..3600s). The variable is saved persistently. Preset to 0 s. If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *blInvalidParameter* will be set in the event of an incorrect parameter entry.

tDelayFeedbContactor: time delay [ms] of the feedback of the power section after switching on the motor. If this time has elapsed and *bFeedbContactor* = FALSE, then this is fed back to the controller via the error message *bErrorFeedbContactor* (100ms..3600ms). The variable is saved persistently. Preset to 100 ms. If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *blInvalidParameter* will be set in the event of an incorrect parameter entry.

tFeedbProcessTimer: time delay of the process feedback *bFeedbProcess* [s] after switching on the motor. If this time has elapsed and *bFeedbProcess* = FALSE, then this is fed back to the controller via the error message *bErrorFeedbProcess* (0s..3600s). The variable is saved persistently. Preset to 0 s. If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *blInvalidParameter* will be set in the event of an incorrect parameter entry.

tDelayFeedbProcess: in order to avoid undesired switching off of the drive during operation due to the process monitoring *bFeedbProcess*, e.g. in the event of short-term pressure fluctuations, the triggering of the input *bFeedbProcess* can be delayed by the time *tDelayFeedbProcess* (0s..3600s). The variable is saved

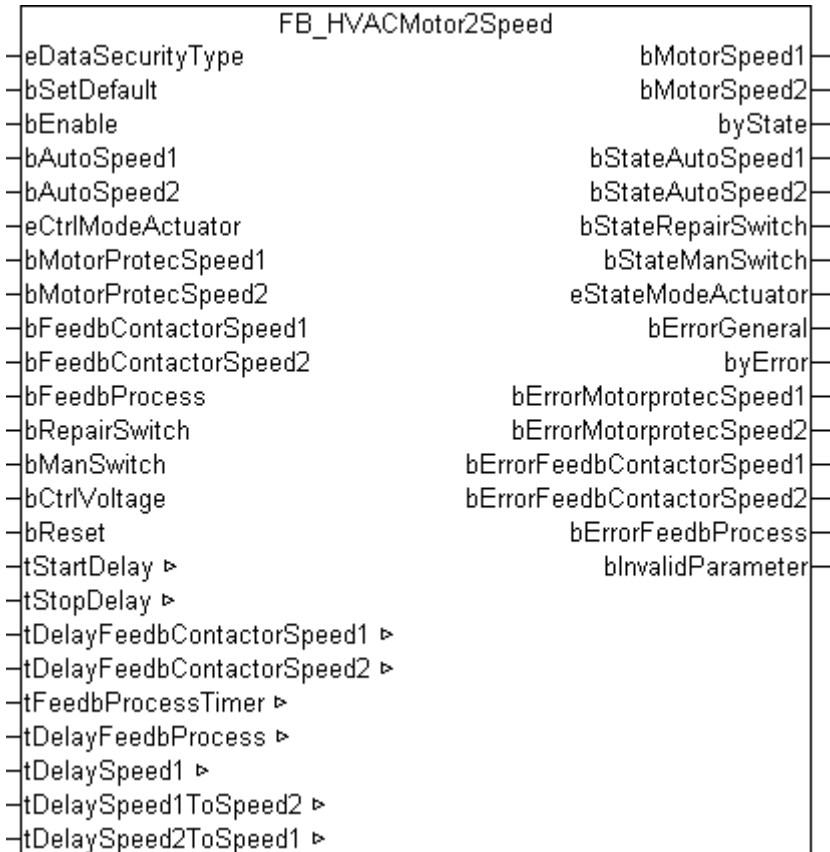
persistently. Preset to 0 s.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *blInvalidParameter* will be set in the event of an incorrect parameter entry.

Documents about this

 [example_persistent_e.zip](#) (Resources/zip/11659714827.zip)

3.1.6 FB_HVACMotor2Speed



Application

This function block serves to control a two-stage drive in HVAC systems. The function block always runs in stage 1, the lower power stage. It cannot be switched on directly in stage 2. In the event of a restart, disablement, an error or switching off of the motor via the operation mode, restart of the motor is blocked for the duration of *tDelaySpeed2ToSpeed1*.

Application example

Download	Required library
TcHVAC.pro [► 531]	TcHVAC.lib

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
bAutoSpeed1       : BOOL;
bAutoSpeed2       : BOOL;
eCtrlModeActuator : E_HVACActuatorMode;
bMotorProtecSpeed1 : BOOL;
bMotorProtecSpeed2 : BOOL;
eStateModeActuator : E_HVACActuatorMode;
bFeedbContactorSpeed1 : BOOL;
bFeedbContactorSpeed2 : BOOL;
```


```

bFeedbProcess      : BOOL;
bRepairSwitch     : BOOL;
bManSwitch        : BOOL;
bCtrlVoltage      : BOOL;
bReset            : BOOL;

```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled if `bEnable = TRUE`. If `bEnable = FALSE`, the drive will be switched off in the respective switch-on stage with a delay specified in the time variable `tStopDelay`. If an enable is present, a restart of the motor is blocked for the duration of `tDelaySpeed2ToSpeed1`.

bAutoSpeed1: `bAutoSpeed1` is only active if the operation mode `eCtrlModeActuator` is either `eHVACActuatorMode_Auto_BMS` or `eHVACActuatorMode_Auto_OP`. If the input variables `bAutoSpeed1 = TRUE` and `bAutoSpeed2 = FALSE`, the function block is instructed that the drive should run at speed 1. If `bAutoSpeed1 = FALSE`, the drive will be switched off with a delay specified in the time variable `tStopDelay`.

bAutoSpeed2: `bAutoSpeed2` is only active if the operation mode `eCtrlModeActuator` is either `eHVACActuatorMode_Auto_BMS` or `eHVACActuatorMode_Auto_OP`. If the input variable `bAutoSpeed1` and `bAutoSpeed2 = TRUE`, the function block is instructed to run the drive in at speed 2. If `bAutoSpeed1` and `bAutoSpeed2 = FALSE`, the drive is switched off with a delay specified in the time variable `tStopDelay`.

eCtrlModeActuator: Enum that specifies the operation mode of the motor. In the event of an incorrect entry, operation continues internally with the last valid operating mode. This is `eHVACActuatorMode_Auto_BMS` in the case of initial commissioning. `bInvalidParameter` will be set in the event of an incorrect parameter entry.

bMotor ProtecSpeed1: input for motor protection speed 1. There is a motor protection fault if the input `bMotor ProtecSpeed1` is FALSE (quiescent current principle). In the event of a fault, the outputs `bMotorSpeed1` and `bMotorSpeed2` are all FALSE; the fault is indicated at the output of the function block by `bErrorMotorprotecSpeed1`. The motor can only be restarted if the fault has been rectified and acknowledged at the input `bReset`.

bMotor ProtecSpeed2: input for motor protection speed 2. There is a motor protection fault if the input `bMotor ProtecSpeed2` is FALSE (quiescent current principle). In the event of a fault, the outputs `bMotorSpeed1` and `bMotorSpeed2` are all FALSE; the fault is indicated at the output of the function block by `bErrorMotorprotecSpeed2`. The motor can only be restarted if the fault has been rectified and acknowledged at the input `bReset`.

bFeedbContactorSpeed1: feedback from the power section of the motor for speed 1. The operating feedback is present if the input `bFeedbContactorSpeed1 = TRUE`. If, after switching on the motor, this feedback is not present after the time delay set by `tDelayFeedbContactorSpeed1`, the output `bErrorFeedbContactorSpeed1` is set in order to indicate a fault. In the event of a fault, the outputs `bMotorSpeed1` and `bMotorSpeed2` are all FALSE. The motor can only be restarted if the fault has been

rectified and acknowledged at the input *bReset*. If no feedback from the power section of the motor is present for speed 1, the output variable *bMotorSpeed1* must be applied to the input *bFeedbContactorSpeed1*.



If no feedback from the power section of the motor is present, the output variable *bMotorSpeed1* must be applied to the input *bFeedbContactorSpeed1*. See application example

bFeedbContactorSpeed2: feedback from the power section of the motor for speed 2. The operating feedback is present if the input *bFeedbContactorSpeed2* = TRUE. If, after switching on the motor, this feedback is not present after the time delay set by *tDelayFeedbContactorSpeed2*, the output *bErrorFeedbContactorSpeed2* is set in order to indicate a fault. In the event of a fault, the outputs *bMotorSpeed1* and *bMotorSpeed2* are all FALSE. The motor can only be restarted if the fault has been rectified and acknowledged at the input *bReset*. If no feedback from the power section of the motor is present for speed 2, the output variable *bMotorSpeed2* must be applied to the input *bFeedbContactorSpeed2*.



If no feedback from the power section of the motor is present, the output variable *bMotorSpeed2* must be applied to the input *bFeedbContactorSpeed2*. See application example

bFeedbProcess: a process feedback signal, for example from a v-belt monitor or a flow monitor can be connected to the input *bFeedbProcess*. The process feedback is present if the input *bFeedbProcess* = TRUE (quiescent current principle). If, in the start-up phase after switching on the motor, the process feedback is not present after the time set by *tFeedbProcessTimer*, the drive switches off and indicates a fault at the output *bErrorFeedbProcess*. The motor can only be restarted if the fault has been rectified and acknowledged at the input *bReset*. In order to avoid undesired switching off of the drive during operation due to the process monitoring *bFeedbProcess*, e.g. in the event of short-term pressure fluctuations, the triggering of the input *bFeedbProcess* can be delayed by the time *tDelayFeedbProcess*. The process feedback is active if either *bMotorSpeed1* or *bMotorSpeed2* = TRUE.

bRepairSwitch: the state of the repair switch is monitored with the input *bRepairSwitch*. The motor can only be switched on if *bRepairSwitch* = TRUE (quiescent current principle). If the repair switch is switched off, *bRepairSwitch* = FALSE, the outputs *bMotorSpeed1* and *bMotorSpeed2* are all FALSE. If the state of the repair switch is TRUE, a restart of the motor is blocked for the duration of *tDelaySpeed2ToSpeed1*.

bManSwitch: the state of the manual/emergency switch is monitored with the input *bManSwitch*. The motor can only be switched on if *bManSwitch* = TRUE (quiescent current principle). If the manual/emergency switch is switched off, *bManSwitch* = FALSE, the outputs *bMotorSpeed1* and *bMotorSpeed2* are both FALSE. If the state of the manual/emergency switch is TRUE, a restart of the motor is blocked for the duration of *tDelaySpeed2ToSpeed1*.

bCtrlVoltage: the control voltage is monitored with the input *bCtrlVoltage*. The motor can only be switched on if *bCtrlVoltage* = TRUE (quiescent current principle). If the control voltage is switched off, *bCtrlVoltage* = FALSE, the outputs *bMotorSpeed1* and *bMotorSpeed2* are both FALSE. In order to avoid a torrent of error messages if the control voltage fails, the error messages of the function block are suppressed. If the control voltage is restored, the error messages are enabled again. If control voltage is present, a restart of the motor is blocked for the duration of *tDelaySpeed2ToSpeed1*.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

<i>bMotorSpeed1</i>	: BOOL;
<i>bMotorSpeed2</i>	: BOOL;
<i>byState</i>	: BYTE;
<i>bStateAutoSpeed1</i>	: BOOL;
<i>bStateAutoSpeed2</i>	: BOOL;
<i>bStateRepairSwitch</i>	: BOOL;
<i>bStateManSwitch</i>	: BOOL;
<i>eStateModeActuator</i>	: E_HVACActuatorMode;
<i>bErrorGeneral</i>	: BOOL;
<i>byError</i>	: BYTE;
<i>bErrorMotorprotecSpeed1</i>	: BOOL;
<i>bErrorMotorprotecSpeed2</i>	: BOOL;
<i>bErrorFeedbContactorSpeed1</i>	: BOOL;

```

bErrorFeedbContactorSpeed2 : BOOL;
bErrorFeedbProcess          : BOOL;
bInvalidParameter          : BOOL;

```

bMotorSpeed1: output variable for controlling speed 1 of the two-speed drive.

bMotorSpeed2: output variable for controlling speed 2 of the two-speed drive.

byState: status byte indicating the operating state of the function block

```

byState.0 := bEnable;
byState.1 := bMotorSpeed1;
byState.2 := bMotorSpeed2;
byState.3 := bStateAutoSpeed1;
byState.4 := bStateAutoSpeed2;
byState.5 := bStateRepairSwitch;
byState.6 := bStateManSwitch;
byState.7 := bCtrlVoltage

```

bStateAutoSpeed1: indicates the state for automatic preselection speed 1 if the operation mode *eCtrlModeActuator* is either *eHVACActuatorMode_Auto_BMS* or *eHVACActuatorMode_Auto_OP* and speed 1 was activated via the input variable *bAutoSpeed1*.

bStateAutoSpeed2: indicates the state for automatic preselection speed 2 if the operation mode *eCtrlModeActuator* is either *eHVACActuatorMode_Auto_BMS* or *eHVACActuatorMode_Auto_OP* and speed 2 was activated via the input variable *bAutoSpeed2*.

bStateRepairSwitch: status message of the repair switch. TRUE indicates that the repair switch is switched off.

bStateManSwitch: status message of the manual/emergency switch. A TRUE signals that the manual/emergency operating level is activated.

eStateModeActuator: Enum via which the state of the operation mode of the motor is fed back to the controller.

bErrorGeneral: the error message *bErrorGeneral* becomes TRUE as soon as one of the error messages *bErrorMotorprotecSpeed1*, *bErrorMotorprotecSpeed2*, *bErrorFeedbContactorSpeed1*, *bErrorFeedbContactorSpeed2* or *bErrorFeedbProcess* = TRUE. The outputs *bMotorSpeed1* and *bMotorSpeed2* are then set to FALSE and are only enabled again when the fault has been rectified and acknowledged via *bReset*. After rectification of the fault a restart of the motor is blocked for the duration of *tDelaySpeed2ToSpeed1*.

byError: returns all error messages and warnings of the function block.

```

byError.1 := bInvalidParameter;
byError.2 := bErrorGeneral;
byError.3 := bErrorMotorprotecSpeed1;
byError.4 := bErrorMotorprotecSpeed2;
byError.5 := bErrorFeedbContactorSpeed1;
byError.6 := bErrorFeedbContactorSpeed2;
byError.7 := bErrorFeedbProcess;

```

bErrorMotorprotecSpeed1: error motor protection, see input variable *bMotorProtecSpeed1*.

bErrorMotorprotecSpeed2: error motor protection, see input variable *bMotorProtecSpeed2*.

bErrorFeedbContactorSpeed1: power section feedback error, see input variable *bFeedbContactorSpeed1*



If no feedback from the power section of the motor is present, the output variable *bMotorSpeed1* must be applied to the input *bFeedbContactorSpeed1*. See application example

bErrorFeedbContactorSpeed2: power section feedback error, see input variable *bFeedbContactorSpeed2*



If no feedback from the power section of the motor is present, the output variable *bMotorSpeed2* must be applied to the input *bFeedbContactorSpeed2*. See application example

bErrorFeedbProcess: process feedback error, see input variable *bFeedbackProcess*

blInvalidParameter: indicates that an incorrect parameter is present at one of the variables *eCtrlModeActuator*, *tStartDelay*, *tStopDelay*, *tDelayFeedbContactorSpeed1*, *tDelayFeedbContactorSpeed2*, *tFeedbProcessTimer*, *tDelayFeedbProcess*, *tDelaySpeed1* or *tDelaySpeed1ToSpeed2*. An incorrect parameter specification does not lead to a standstill of the function block; see description of variables. After rectifying the incorrect parameter entry, the message *blInvalidParameter* must be acknowledged via *bReset*.

VAR_IN_OUT

```
tStartDelay      : TIME;
tStopDelay       : TIME;
tDelayFeedbContactorSpeed1 : TIME;
tDelayFeedbContactorSpeed2 : TIME;
tFeedbProcessTimer : TIME;
tDelayFeedbProcess : TIME;
tDelaySpeed1     : TIME;
tDelaySpeed1ToSpeed2 : TIME;
tDelaySpeed2ToSpeed1 : TIME;
```

tStartDelay: the start-up of the motor after enabling and switching on via the operation mode of the motor is delayed by the time *tStartDelay* [s] (0s..3600s). The variable is saved persistently. Preset to 0 s.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *blInvalidParameter* will be set in the event of an incorrect parameter entry.

tStopDelay: the switching off of the motor in the respective switch-on stage, either by switching the enable *bEnable* to FALSE, or by switching off via the operation mode *eCtrlModeActuator*, or in automatic operation mode by switching the input variables *bAutoSpeed1* and *bAutoSpeed2* to FALSE, is delayed by the time *tStopDelay* [s]. Once the delayed switch-off of the motor has been activated it can no longer be canceled (0s..3600s). The variable is saved persistently. Preset to 0 s.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *blInvalidParameter* will be set in the event of an incorrect parameter entry.

tDelayFeedbContactorSpeed1: time delay of the feedback of the power section after switching on the motor. If this time has elapsed and *bFeedbContactorSpeed1* = FALSE, then this is fed back to the controller via the error message *bErrorFeedbContactorSpeed1* (100ms..3600s). The variable is saved persistently. Preset to 100 ms.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *blInvalidParameter* will be set in the event of an incorrect parameter entry.

tDelayFeedbContactorSpeed2: time delay of the feedback of the power section after switching on the motor. If this time has elapsed and *bFeedbContactorSpeed2* = FALSE, then this is fed back to the controller via the error message *bErrorFeedbContactorSpeed2* (100ms..3600s). The variable is saved persistently. Preset to 100 ms.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *blInvalidParameter* will be set in the event of an incorrect parameter entry.

tFeedbProcessTimer: time delay [s] of the process feedback *bFeedbProcess* [s] after start-up phase of the motor. If this time has elapsed and *bFeedbProcess* = FALSE, then this is fed back to the controller via the error message *bErrorFeedbProcess* (0s..3600s). The variable is saved persistently. Preset to 0 s.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *blInvalidParameter* will be set in the event of an incorrect parameter entry.

tDelayFeedbProcess: in order to avoid undesired switching off of the drive during operation due to the process monitoring *bFeedbProcess*, e.g. in the event of short-term pressure fluctuations, the triggering of the input *bFeedbProcess* can be delayed by the time *tDelayFeedbProcess* (0s..3600s). The variable is saved persistently. Preset to 0 s.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *blInvalidParameter* will be set in the event of an incorrect parameter entry.

tDelaySpeed1: time delay [s] for the start-up phase of the motor in speed 1 (1s..3600s). The variable is saved persistently. Preset to 3s.

After this time has elapsed, the motor can be switched from the first to the second speed if the operating mode for speed 2 is selected.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *bInvalidParameter* will be set in the event of an incorrect parameter entry.

tDelaySpeed1ToSpeed2: time delay for the motor switchover phase from speed 1 to speed 2, so that both outputs *bMotorSpeed1* and *bMotorSpeed2* are FALSE for a short while (100ms..10s). The variable is saved persistently. Preset to 250 ms. The time delay serves to protect the motor windings.

If an incorrect variable value is present, the last valid variable value is used, if available. If there is no valid last value, then the default value is used. *bInvalidParameter* will be set in the event of an incorrect parameter entry.

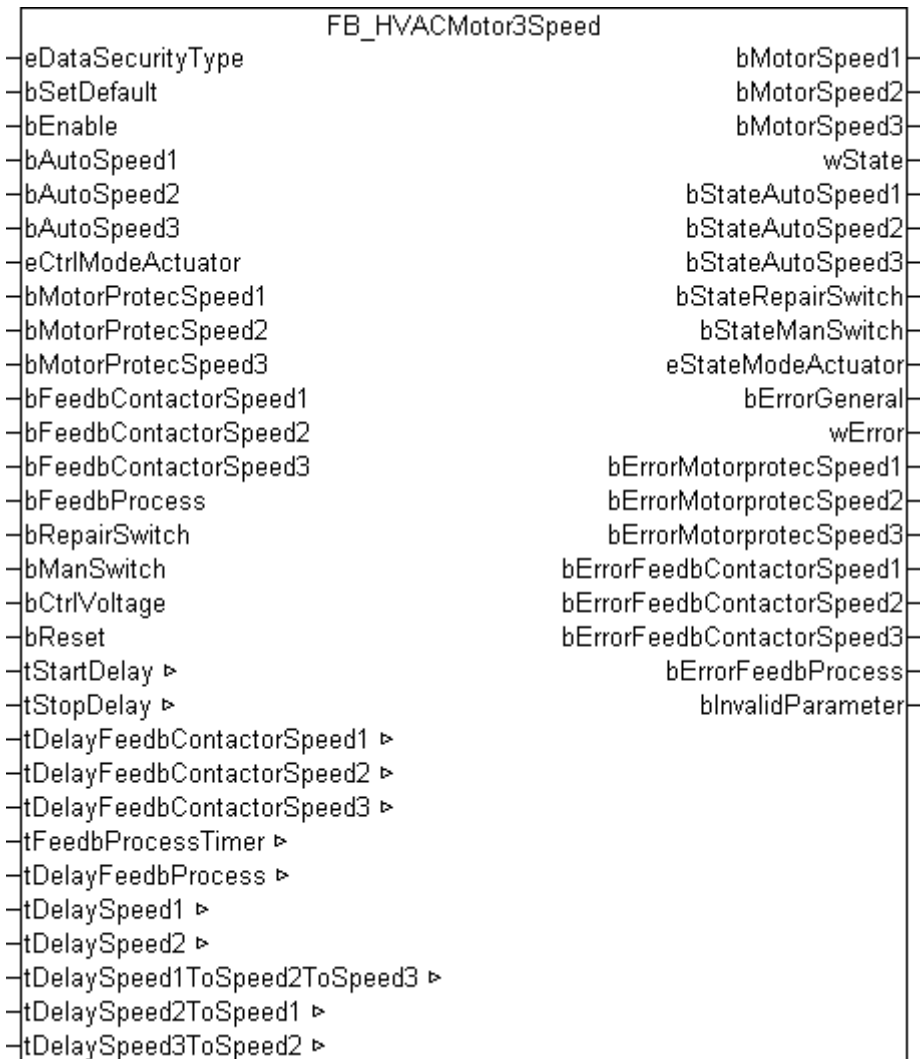
tDelaySpeed2ToSpeed1: time delay [s] for the motor switchover phase from speed 2 to speed 1 (1s..3600s). The variable is saved persistently. Preset to 10 s. In this phase both outputs *bMotorSpeed1* and *bMotorSpeed2* are FALSE for the time *tDelaySpeed2ToSpeed1* in order to reduce the speed of the motor when switching to speed 1.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *bInvalidParameter* will be set in the event of an incorrect parameter entry.

Documents about this

 [example_persistent_e.zip](#) (Resources/zip/11659714827.zip)

3.1.7 FB_HVACMotor3Speed



Application

This function block serves to control a three-speed drive in HVAC systems. The function block always runs in step 1, the lowest power step, and switches to step 2 or step 3 depending on requirements. It cannot be switched on directly in step 2 or 3. In the event of a restart, disablement, an error or switching off of the motor via the operation mode, restart of the motor is blocked for the duration of $tDelaySpeed2ToSpeed1 + tDelaySpeed3ToSpeed2$.

Application example

Download	Required library
TcHVAC.pro [► 531]	TcHVAC.lib

VAR_INPUT

```

eDataSecurityType      : E_HVACDataSecurityType;
bSetDefault            : BOOL;
bEnable                : BOOL;
bAutoSpeed1           : BOOL;
bAutoSpeed2           : BOOL;
bAutoSpeed3           : BOOL;
eCtrlModeActuator     : E_HVACActuatorMode;
bMotorProtecSpeed1    : BOOL;
bMotorProtecSpeed2    : BOOL;
bMotorProtecSpeed3    : BOOL;
bFeedbContactorSpeed1 : BOOL;
bFeedbContactorSpeed2 : BOOL;
    
```


```

bFeedbContactorSpeed3 : BOOL;
bFeedbProcess          : BOOL;
bRepairSwitch         : BOOL;
bManSwitch            : BOOL;
bCtrlVoltage          : BOOL;
bReset                : BOOL;

```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled if `bEnable = TRUE`. If `bEnable = FALSE`, the drive is switched off with a delay specified in the time variable `tStopDelay` at the respective switch-on stage. If an enable is present, a restart of the motor is blocked for the duration of `tDelaySpeed2ToSpeed1 + tDelaySpeed3ToSpeed2`.

bAutoSpeed1: `bAutoSpeed1` is only active if the operation mode `eCtrlModeActuator` is either `eHVACActuatorMode_Auto_BMS` or `eHVACActuatorMode_Auto_OP`. If the input variables `bAutoSpeed1 = TRUE`, `bAutoSpeed2 = FALSE` and `bAutoSpeed3 = FALSE`, then the function block is instructed that the drive should run at speed 1. If `bAutoSpeed1 = FALSE`, the drive will be switched off with a delay specified in the time variable `tStopDelay`.

bAutoSpeed2: `bAutoSpeed2` is only active if the operation mode `eCtrlModeActuator` is either `eHVACActuatorMode_Auto_BMS` or `eHVACActuatorMode_Auto_OP`. If the input variables `bAutoSpeed1 = TRUE`, `bAutoSpeed2 = TRUE` and `bAutoSpeed3 = FALSE`, then the function block is instructed that the drive should run at speed 2. If `bAutoSpeed1` and `bAutoSpeed2 = FALSE`, the drive will be switched off with a delay specified in the time variable `tStopDelay`.

bAutoSpeed3: `bAutoSpeed3` is only active if the operation mode `eCtrlModeActuator` is either `eHVACActuatorMode_Auto_BMS` or `eHVACActuatorMode_Auto_OP`. If the input variables `bAutoSpeed1 = TRUE`, `bAutoSpeed2 = TRUE` and `bAutoSpeed3 = TRUE`, then the function block is instructed that the drive should run at speed 3. If `bAutoSpeed1`, `bAutoSpeed2` and `bAutoSpeed3 = FALSE`, the drive will be switched off with a delay specified in the time variable `tStopDelay`.

eCtrlModeActuator: Enum that specifies the operation mode of the motor. In the event of an incorrect entry, operation continues internally with the last valid operating mode. This is `eHVACActuatorMode_Auto_BMS` in the case of initial commissioning. `bInvalidParameter` will be set in the event of an incorrect parameter entry.

bMotor ProtecSpeed1: input for motor protection speed 1. There is a motor protection fault if the input `bMotor ProtecSpeed1` is FALSE (quiescent current principle). In the event of a fault, the outputs `bMotorSpeed1`, `bMotorSpeed2` and `bMotorSpeed3` are all FALSE; the fault is indicated at the output of the function block by `bErrorMotorprotecSpeed1`. The motor can only be restarted if the fault has been rectified and acknowledged at the input `bReset`.

bMotorProtecSpeed2: input for motor protection speed 2. There is a motor protection fault if the input *bMotorProtecSpeed2* is FALSE (quiescent current principle). In the event of a fault, the outputs *bMotorSpeed1*, *bMotorSpeed2* and *bMotorSpeed3* are all FALSE; the fault is indicated at the output of the function block by *bErrorMotorprotecSpeed2*. The motor can only be restarted if the fault has been rectified and acknowledged at the input *bReset*.

bMotorProtecSpeed3: input for motor protection speed 3. There is a motor protection fault if the input *bMotorProtecSpeed3* is FALSE (quiescent current principle). In the event of a fault, the outputs *bMotorSpeed1*, *bMotorSpeed2* and *bMotorSpeed3* are all FALSE; the fault is indicated at the output of the function block by *bErrorMotorprotecSpeed3*. The motor can only be restarted if the fault has been rectified and acknowledged at the input *bReset*.

bFeedbContactorSpeed1: feedback from the power section of the motor for speed 1. The operating feedback is present if the input *bFeedbContactorSpeed1* = TRUE. If, after switching on the motor, this feedback is not present after the time delay set by *tDelayFeedbContactorSpeed1*, the output *bErrorFeedbContactorSpeed1* is set in order to indicate a fault. In the event of a fault, the outputs *bMotorSpeed1*, *bMotorSpeed2* and *bMotorSpeed3* are all FALSE. The motor can only be restarted if the fault has been rectified and acknowledged at the input *bReset*. If no feedback from the power section of the motor is present for speed 1, the output variable *bMotorSpeed1* must be applied to the input *bFeedbContactorSpeed1*.



If no feedback from the power section of the motor is present, the output variable *bMotorSpeed1* must be applied to the input *bFeedbContactorSpeed1*. See application example

bFeedbContactorSpeed2: feedback from the power section of the motor for speed 2. The operating feedback is present if the input *bFeedbContactorSpeed2* = TRUE. If, after switching on the motor, this feedback is not present after the time delay set by *tDelayFeedbContactorSpeed2*, the output *bErrorFeedbContactorSpeed2* is set in order to indicate a fault. In the event of a fault, the outputs *bMotorSpeed1*, *bMotorSpeed2* and *bMotorSpeed3* are all FALSE. The motor can only be restarted if the fault has been rectified and acknowledged at the input *bReset*. If no feedback from the power section of the motor is present for speed 2, the output variable *bMotorSpeed2* must be applied to the input *bFeedbContactorSpeed2*.



If no feedback from the power section of the motor is present, the output variable *bMotorSpeed2* must be applied to the input *bFeedbContactorSpeed2*. See application example

bFeedbContactorSpeed3: feedback from the power section of the motor for speed 3. The operating feedback is present if the input *bFeedbContactorSpeed3* = TRUE. If, after switching on the motor, this feedback is not present after the time delay set by *tDelayFeedbContactorSpeed3*, the output *bErrorFeedbContactorSpeed3* is set in order to indicate a fault. In the event of a fault, the outputs *bMotorSpeed1*, *bMotorSpeed2* and *bMotorSpeed3* are all FALSE. The motor can only be restarted if the fault has been rectified and acknowledged at the input *bReset*. If no feedback from the power section of the motor is present for speed 3, the output variable *bMotorSpeed3* must be applied to the input *bFeedbContactorSpeed3*.



If no feedback from the power section of the motor is present, the output variable *bMotorSpeed3* must be applied to the input *bFeedbContactorSpeed3*. See application example

bFeedbProcess: a process feedback signal, for example from a v-belt monitor or a flow monitor can be connected to the input *bFeedbProcess*. The process feedback is present if the input *bFeedbProcess* = TRUE (quiescent current principle). If, in the start-up phase after switching on the motor, the process feedback is not present after the time set by *tFeedbProcessTimer*, the drive switches off and indicates a fault at the output *bErrorFeedbProcess*. The motor can only be restarted if the fault has been rectified and acknowledged at the input *bReset*. In order to avoid undesired switching off of the drive during operation due to the process monitoring *bFeedbProcess*, e.g. in the event of short-term pressure fluctuations, the triggering of the input *bFeedbProcess* can be delayed by the time *tDelayFeedbProcess*. The process feedback is active if either *bMotorSpeed1*, *bMotorSpeed2* or *bMotorSpeed3* = TRUE.

bRepairSwitch: the state of the repair switch is monitored with the input *bRepairSwitch*. The motor can only be switched on if *bRepairSwitch* = TRUE (quiescent current principle). If the repair switch is switched off, *bRepairSwitch* = FALSE, the outputs *bMotorSpeed1*, *bMotorSpeed2* and *bMotorSpeed3* are all FALSE. If the state of the repair switch is TRUE, a restart of the motor is blocked for the duration of $tDelaySpeed2ToSpeed1 + tDelaySpeed3ToSpeed2$.

bManSwitch: the state of the manual/emergency switch is monitored with the input *bManSwitch*. The motor can only be switched on if *bManSwitch* = TRUE (quiescent current principle). If the manual/emergency switch is switched off, *bManSwitch* = FALSE, the outputs *bMotorSpeed1*, *bMotorSpeed2* and *bMotorSpeed3* are all FALSE. If the state of the manual/emergency switch is TRUE, a restart of the motor is blocked for the duration of $tDelaySpeed2ToSpeed1 + tDelaySpeed3ToSpeed2$.

bCtrlVoltage: the control voltage is monitored with the input *bCtrlVoltage*. The motor can only be switched on if *bCtrlVoltage* = TRUE (quiescent current principle). If the control voltage is switched off, *bCtrlVoltage* = FALSE, the outputs *bMotorSpeed1*, *bMotorSpeed2* and *bMotorSpeed3* are all FALSE. In order to avoid a torrent of error messages if the control voltage fails, the error messages of the function block are suppressed. If the control voltage is restored, the error messages are enabled again. If control voltage is present, a restart of the motor is blocked for the duration of $tDelaySpeed2ToSpeed1 + tDelaySpeed3ToSpeed2$.

bReset: input for acknowledgement of faults via a rising edge.

VAR_OUTPUT

```

bMotorSpeed1      : BOOL;
bMotorSpeed2      : BOOL;
bMotorSpeed3      : BOOL;
wState            : WORD;
bStateAutoSpeed1  : BOOL;
bStateAutoSpeed2  : BOOL;
bStateAutoSpeed3  : BOOL;
bStateRepairSwitch : BOOL;
bStateManSwitch   : BOOL;
eStateModeActuator : E_HVACActuatorMode;
bErrorGeneral     : BOOL;
wError            : WORD;
bErrorMotorprotecSpeed1 : BOOL;
bErrorMotorprotecSpeed2 : BOOL;
bErrorMotorprotecSpeed3 : BOOL;
bErrorFeedbContactorSpeed1 : BOOL;
bErrorFeedbContactorSpeed2 : BOOL;
bErrorFeedbContactorSpeed3 : BOOL;
bErrorFeedbProcess : BOOL;
bInvalidParameter : BOOL;

```

bMotorSpeed1: output variable for controlling speed 1 of the three-speed drive.

bMotorSpeed2: output variable for controlling speed 2 of the three-speed drive.

bMotorSpeed3: output variable for controlling speed 3 of the three-speed drive.

wState: statusword indicating the operating state of the function block.

```

wState.0 := bEnable;
wState.1 := bMotorSpeed1;
wState.2 := bMotorSpeed2;
wState.3 := bMotorSpeed3;
wState.4 := bStateAutoSpeed1;
wState.5 := bStateAutoSpeed2;
wState.6 := bStateAutoSpeed3;
wState.7 := bStateRepairSwitch;
wState.8 := bStateManSwitch;
wState.9 := bCtrlVoltage;

```

bStateAutoSpeed1: indicates the state for automatic preselection speed 1 if the operation mode *eCtrlModeActuator* is either *eHVACActuatorMode_Auto_BMS* or *eHVACActuatorMode_Auto_OP* and speed 1 was activated via the input variable *bAutoSpeed1*.

bStateAutoSpeed2: indicates the state for automatic preselection speed 2 if the operation mode *eCtrlModeActuator* is either *eHVACActuatorMode_Auto_BMS* or *eHVACActuatorMode_Auto_OP* and speed 2 was activated via the input variable *bAutoSpeed2*.

bStateAutoSpeed3: indicates the state for automatic preselection speed 3 if the operation mode *eCtrlModeActuator* is either *eHVACActuatorMode_Auto_BMS* or *eHVACActuatorMode_Auto_OP* and speed 3 was activated via the input variable *bAutoSpeed3*.

bStateRepairSwitch: status message of the repair switch. TRUE indicates that the repair switch is switched off.

bStateManSwitch: status message of the manual/emergency switch. A TRUE signals that the manual/emergency operating level is activated.

eStateModeActuator: Enum via which the state of the operation mode of the motor is fed back to the controller.

bErrorGeneral: the error message *bErrorGeneral* becomes TRUE as soon as one of the error messages *bErrorMotorprotecSpeed1*, *bErrorMotorprotecSpeed2*, *bErrorMotorprotecSpeed3*, *bErrorFeedbContactorSpeed1*, *bErrorFeedbContactorSpeed2*, *bErrorFeedbContactorSpeed3* or *bErrorFeedbProcess* = TRUE. The outputs *bMotorSpeed1*, *bMotorSpeed2* and *bMotorSpeed3* are then set to FALSE and are only enabled again when the fault has been rectified and acknowledged via *bReset*. After rectification of the fault a restart of the motor is blocked for the duration of $tDelaySpeed2ToSpeed1 + tDelaySpeed3ToSpeed2$.

wError: returns all error messages and warnings of the function block.

wError.1 := *bInvalidParameter*;
wError.2 := *bErrorGeneral*;
wError.3 := *bErrorMotorprotecSpeed1*;
wError.4 := *bErrorMotorprotecSpeed2*;
wError.5 := *bErrorMotorprotecSpeed3*;
wError.6 := *bErrorFeedbContactorSpeed1*;
wError.7 := *bErrorFeedbContactorSpeed2*;
wError.8 := *bErrorFeedbContactorSpeed3*;
wError.9 := *bErrorFeedbProcess*;

bErrorMotorprotecSpeed1: error motor protection, see input variable *bMotorProtecSpeed1*.

bErrorMotorprotecSpeed2: error motor protection, see input variable *bMotorProtecSpeed2*.

bErrorMotorprotecSpeed3: error motor protection, see input variable *bMotorProtecSpeed3*.

bErrorFeedbContactorSpeed1: power section feedback error, see input variable *bFeedbContactorSpeed1*



If no feedback from the power section of the motor is present, the output variable *bMotorSpeed1* must be applied to the input *bFeedbContactorSpeed1*. See application example

bErrorFeedbContactorSpeed2: power section feedback error, see input variable *bFeedbContactorSpeed2*



If no feedback from the power section of the motor is present, the output variable *bMotorSpeed2* must be applied to the input *bFeedbContactorSpeed2*. See application example

bErrorFeedbContactorSpeed3: power section feedback error, see input variable *bFeedbContactorSpeed3*



If no feedback from the power section of the motor is present, the output variable *bMotorSpeed3* must be applied to the input *bFeedbContactorSpeed3*. See application example

bErrorFeedbProcess: process feedback error, see input variable *bFeedbackProcess*

bInvalidParameter: indicates that an incorrect parameter is present at one of the variables *eCtrlModeActuator*, *tStartDelay*, *tStopDelay*, *tDelayFeedbContactorSpeed1*, *tDelayFeedbContactorSpeed2*, *tDelayFeedbContactorSpeed3*, *tFeedbProcessTimer*, *tDelayFeedbProcess*, *tDelaySpeed1*, *tDelaySpeed2*, *tDelaySpeed1ToSpeed2ToSpeed3*, *tDelaySpeed2ToSpeed1* or *tDelaySpeed3ToSpeed2*. An incorrect parameter specification does not lead to a standstill of the function block; see description of variables. After rectifying the incorrect parameter entry, the message *bInvalidParameter* must be acknowledged via *bReset*.

VAR_IN_OUT

```

tStartDelay      : TIME;
tStopDelay       : TIME;
tDelayFeedbContactorSpeed1 : TIME;
tDelayFeedbContactorSpeed2 : TIME;
tDelayFeedbContactorSpeed3 : TIME;
tFeedbProcessTimer : TIME;
tDelayFeedbProcess : TIME;
tDelaySpeed1     : TIME;
tDelaySpeed2     : TIME;
tDelaySpeed1ToSpeed2ToSpeed3 : TIME;
tDelaySpeed2ToSpeed1 : TIME;
tDelaySpeed3ToSpeed2 : TIME;

```

tStartDelay: the start-up of the motor after enabling and switching on via the operation mode of the motor is delayed by the time *tStartDelay* [s] (0s..3600s). The variable is saved persistently. Preset to 0 s.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *bInvalidParameter* will be set in the event of an incorrect parameter entry.

tStopDelay: the switching off of the motor in the respective switch-on stage, either by switching the enable *bEnable* to FALSE, or by switching off via the operation mode *eCtrlModeActuator*, or in automatic operation mode by switching the input variables *bAutoSpeed1*, *bAutoSpeed2* and *bAutoSpeed3* to FALSE, is delayed by the time *tStopDelay* [s]. Once the delayed switch-off of the motor has been activated it can no longer be canceled (0s..3600s). The variable is saved persistently. Preset to 0 s.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *bInvalidParameter* will be set in the event of an incorrect parameter entry.

tDelayFeedbContactorSpeed1: time delay of the feedback of the power section after switching on the motor. If this time has elapsed and *bFeedbContactorSpeed1* = FALSE, then this is fed back to the controller via the error message *bErrorFeedbContactorSpeed1* (100ms..3600s). The variable is saved persistently. Preset to 100 ms.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, operation continues with the default value. *bInvalidParameter* will be set in the event of an incorrect parameter entry.

tDelayFeedbContactorSpeed2: time delay of the feedback of the power section after switching on the motor. If this time has elapsed and *bFeedbContactorSpeed2* = FALSE, then this is fed back to the controller via the error message *bErrorFeedbContactorSpeed2* (100ms..3600s). The variable is saved persistently. Preset to 100 ms.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *bInvalidParameter* will be set in the event of an incorrect parameter entry.

tDelayFeedbContactorSpeed3: time delay of the feedback of the power section after switching on the motor. If this time has elapsed and *bFeedbContactorSpeed3* = FALSE, then this is fed back to the controller via the error message *bErrorFeedbContactorSpeed3* (100ms..3600s). The variable is saved persistently. Preset to 100 ms.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *bInvalidParameter* will be set in the event of an incorrect parameter entry.

tFeedbackProcessTimer: time delay [s] of the process feedback *bFeedbProcess* [s] after start-up phase of the motor. If this time has elapsed and *bFeedbProcess* = FALSE, then this is fed back to the controller via the error message *bErrorFeedbProcess* (0s..3600s). The variable is saved persistently. Preset to 0 s.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *bInvalidParameter* will be set in the event of an incorrect parameter entry.

tDelayFeedbProcess: in order to avoid undesired switching off of the drive during operation due to the process monitoring *bFeedbProcess*, e.g. in the event of short-term pressure fluctuations, the triggering of the input *bFeedbProcess* can be delayed by the time *tDelayFeedbProcess* (0s..3600s). The variable is saved persistently. Preset to 0 s.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *bInvalidParameter* will be set in the event of an incorrect parameter entry.

tDelaySpeed1: time delay [s] for the start-up phase of the motor in speed 1 (1s..3600s). The variable is saved persistently. Preset to 3s.
After this time has elapsed, the motor can be switched from the first to the second speed if the operating mode for speed 2 is selected.
If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *bInvalidParameter* will be set in the event of an incorrect parameter entry.

tDelaySpeed2: time delay [s] for the start-up phase of the motor in speed 2 (1s..3600s). The variable is saved persistently. Preset to 3s.
After this time has elapsed, the motor can be switched from the second to the third speed if the operation mode for speed 3 is selected.
If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *bInvalidParameter* will be set in the event of an incorrect parameter entry.

tDelaySpeed1ToSpeed2ToSpeed3: time delay for the motor switchover phase from speed 1 to speed 2 or from speed 2 to speed 3, so that the outputs *bMotorSpeed1*, *bMotorSpeed2* and *bMotorSpeed3* are FALSE for a short while (1s..3600s). The variable is saved persistently. Preset to 250 ms. The time delay serves to protect the motor windings.
If an incorrect variable value is present, the last valid variable value is used, if available. If there is no valid last value, then the default value is used. *bInvalidParameter* will be set in the event of an incorrect parameter entry.

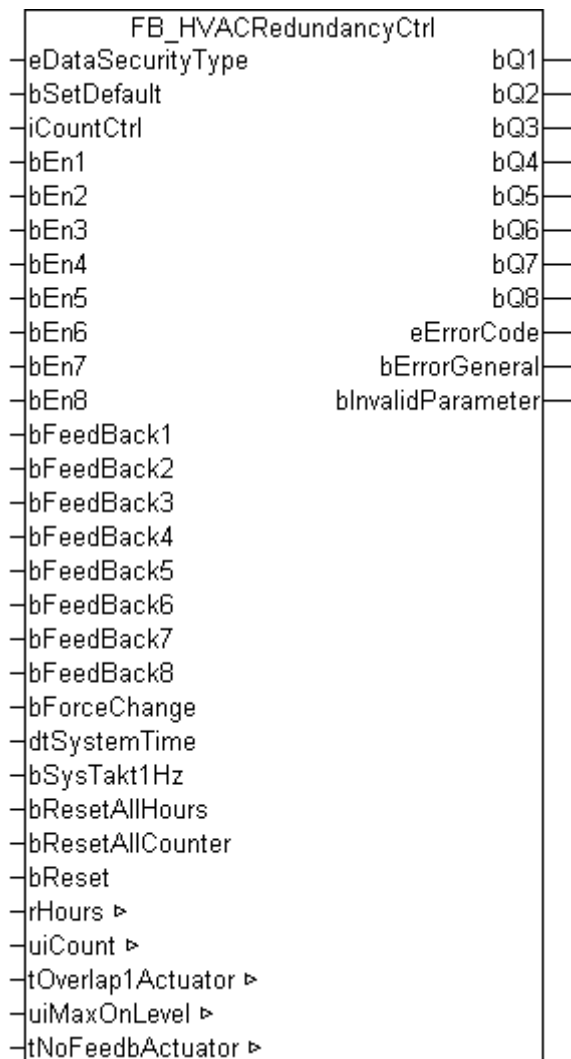
tDelaySpeed2ToSpeed1: time delay [s] for the motor switchover phase from speed 2 to speed 1 (1s..3600s). The variable is saved persistently. Preset to 10 s. In this phase the outputs *bMotorSpeed1*, *bMotorSpeed2* and *bMotorSpeed3* are set to FALSE for the duration of *tDelaySpeed2ToSpeed1* in order to reduce the speed of the motor when switching to speed 1.
If an incorrect variable value is present, the last valid variable value is used, if available. If there is no valid last value, then the default value is used. *bInvalidParameter* will be set in the event of an incorrect parameter entry.

tDelaySpeed3ToSpeed2: time delay [s] for the motor switchover phase from speed 3 to speed 2 (1s..3600s). The variable is saved persistently. Preset to 10 s. In this phase the outputs *bMotorSpeed1*, *bMotorSpeed2* and *bMotorSpeed3* are FALSE for the time *tDelaySpeed3ToSpeed2* in order to reduce the speed of the motor when switching to speed 2.
If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *bInvalidParameter* will be set in the event of an incorrect parameter entry.

Documents about this

 [example_persistent_e.zip \(Resources/zip/11659714827.zip\)](#)

3.1.8 FB_HVACRedundancyCtrl



Application

This function block serves to control a certain number of actuators, e.g. of pumps, from a pool of 8 actuators. From all enabled actuators, the logic searches for those with the shortest runtimes and switches them on according to the runtime order. This continues until the number specified via *iCountCtrl* is reached. The actuators are routed internally via a FIFO memory so that they can be switched off again in the same order. Changeover during operation can be performed with the time specification *uiMaxOnLevel* or by the trigger *bForceChnagne*. The actuator that has been switched on longest is switched off, and the one with the shortest runtime is switched on. In order to avoid a hydraulic stroke in the pumps, an overlap time can be set via *tOverlap1Actuator*. This time is only valid for the case when a change between two actuators takes place. To determine the running times, the function block [FB_HVACWork \[▶ 472\]](#) is instantiated internally and the IN-OUT variables are passed on externally (*rHours* and *uiCount*). The recording of the operating time is controlled by the **Feedback** inputs. If no feedback signal from the actuator is available, the actuator output must be fed back to the feedback input.

VAR_INPUT

VAR_INPUT

```
eDataSecurityType      : E_HVACDataSecurityType;
bSetDefault            : BOOL;
iCountCtrl             : INT;
bEn1 - bEn8           : BOOL;
bFeedBack1 - bFeedBack8 : BOOL;
bForceChange          : BOOL;
dtSystemTime          : DT;
```


```

bSysTakt1Hz      : BOOL;
bResetAllHours   : BOOL;
bResetAllCounter : BOOL;
bReset           : BOOL;

```

eDataSecurityType: if `eDataSecurityType:= eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

iCountCtrl: number of actuators to be switched on (0..8).

bEn1 - 8: enables the corresponding actuator.

bFeedBack1 - 8: operating feedback from the actuator. Evaluation only takes place if `tNoFeedbActuator > ##0s`.

bForceChange: a positive edge at the input switches off the first actuator in the FIFO and switches on the actuator from the pool that has the shortest runtime.

dtSystemTime: system time.

bSysTakt1Hz: 1 Hz clock signal as a replacement for `dtSystemTime`; if `dtSystemTime` is not available, or does not change its value for longer than 2 s, then the clock signal is used as a replacement.

bResetAllHours: resets all operating hours counters.

bResetAllCounter: resets all switch-on counters.

bReset: acknowledge input in the event of a fault.

VAR_IN_OUT

```

rHours           : REAL;
uiCount          : UINT;
tOverlap1Actuator : TIME;
uiMaxOnLevel     : UINT;
tNoFeedbActuator : TIME;

```

rHours[1..8]: operating hours [h] with a resolution of 1/100 hours (internally with 1 s). The variable is saved persistently.

uiCount[1..8]: switch-on cycle counter. The variable is saved persistently.

tOverlap1Actuator: overlap time for the case where an exchange between two actuators takes place (0ms..1min). The variable is saved persistently. Preset to 20 s.

uiMaxOnLevel: max. time in hours [h] that an actuator may be switched on (0h..1000h). Forces an actuator change only if an actuator is available to be switched on. The variable is saved persistently. Preset to 200 h.

tNoFeedbActuator: time that is allowed to elapse before the lack of an operating feedback from the actuator leads to *bErrorGeneral* = TRUE (0ms..60s). There is no evaluation if the time = 0. The variable is saved persistently. Preset to 3 s

VAR_OUTPUT

```
bQ1 - bQ8      : BOOL;  
eErrorCode     : E_HVACErrorCodes;  
bErrorGeneral  : BOOL;  
bInvalidParameter : BOOL;
```

bQ1 - 8: actuator on signal.

eErrorCode: indicates which actuator has not returned an operating feedback within the prespecified timespan. The detection of this error group is activated by a time greater than 0 in the variable *tNoFeedbActuator*. *eHVACErrorCodes_Error_NoFeedbackActuator1 := 15*,
eHVACErrorCodes_Error_NoFeedbackActuator2 := 16,
eHVACErrorCodes_Error_NoFeedbackActuator3 := 17,
eHVACErrorCodes_Error_NoFeedbackActuator4 := 18,
eHVACErrorCodes_Error_NoFeedbackActuator5 := 19,
eHVACErrorCodes_Error_NoFeedbackActuator6 := 20,
eHVACErrorCodes_Error_NoFeedbackActuator7 := 21,
eHVACErrorCodes_Error_NoFeedbackActuator8 := 22,

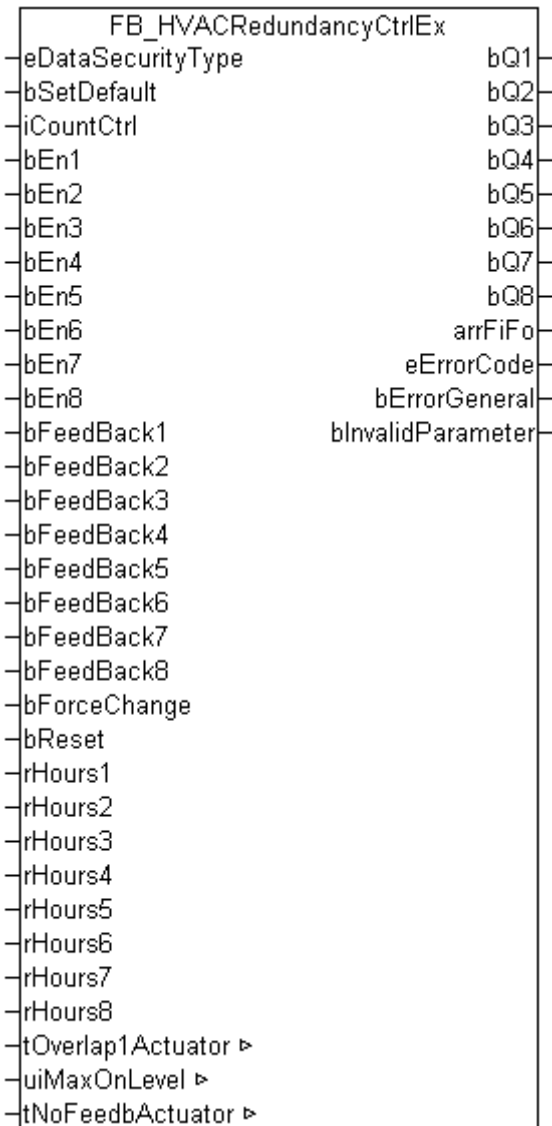
bErrorGeneral: error during evaluation of the operating feedback.

bInvalidParameter: TRUE if an error occurs during the plausibility check. The message must be acknowledged with *bReset*.

Documents about this

 [example_persistent_e.zip \(Resources/zip/11659714827.zip\)](#)

3.1.9 FB_HVACRedundancyCtrlEx



Application


This function block serves to control a certain number of actuators, e.g. of pumps, from a pool of 8 actuators. From all enabled actuators, the logic searches for those with the shortest runtimes and switches them on according to the runtime order. This continues until the number specified via *iCountCtrl* is reached. The actuators are routed internally via a FIFO memory so that they can be switched off again in the same order. Changeover during operation can be performed with the time specification *uiMaxOnLevel* or by the trigger *bForceChnagne*. The actuator that has been switched on longest is switched off, and the one with the shortest runtime is switched on. In order to avoid a hydraulic stroke in the pumps, an overlap time can be set via *tOverlap1Actuator*. This time is only valid for the case when a change between two actuators takes place. In contrast to the [FB_HVACRedundancyCtrl](#) [► 54] no internal timer is used for the determination of the running times, but the times must be applied as hourly values from outside as *Var_Input*.

VAR_INPUT

```
eDataSecurityType      : E_HVACDataSecurityType;
bSetDefault            : BOOL;
iCountCtrl             : INT;
bEn1 - bEn8           : BOOL;
bFeedBack1 - bFeedBack8 : BOOL;
bForceChange          : BOOL;
rHours1-rHours8       : REAL;
```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

iCountCtrl : number of actuators to be switched on

bEn1 - 8: enables the corresponding actuator.

bFeedBack1 - 8: operating feedback from the actuator. Evaluation only takes place if `tNoFeedbActuator > t#0s`.

bForceChange: a positive edge at the input switches off the first actuator in the FIFO and switches on the actuator from the pool that has the shortest runtime.

rHours: operating hours

VAR_IN_OUT

```
tOverlap1Actuator : TIME;
uiMaxOnLevel      : UINT;
tNoFeedbActuator  : TIME;
```

tOverlap1Actuator: overlap time for the case where an exchange between two actuators takes place (0ms..1min). The variable is saved persistently. Preset to 20 s.

uiMaxOnLevel: max. time in hours that an actuator may be switched on (0h..1000h). Forces an actuator change only if an actuator is available to be switched on. The variable is saved persistently. Preset to 200 h.

tNoFeedbActuator: time that is allowed to elapse before the lack of an operating feedback from the actuator leads to `bErrorGeneral = TRUE` (0ms..60s). There is no evaluation if the time = 0. The variable is saved persistently. Preset to 3 s

VAR_OUTPUT

```
bQ1 - bQ8          : BOOL;
arrFiFo           : Array[1..8] of INT;
eErrorCode         : E_HVACErrorCodes;
bErrorGeneral     : BOOL;
bInvalidParameter : BOOL;
```

bQ1 - 8: actuator on signal.

arrFiFo: table containing the information showing which actuator is switched on (actuator number). The order specifies the switch-off sequence. `arFifo[1]` = no. of the actuator that will be switched off next.

eErrorCode: indicates which actuator has not returned an operating feedback within the prespecified timespan. The detection of this error group is activated by a time greater than 0 in the variable `tNoFeedbActuator`. `eHVACErrorCodes_Error_NoFeedbackActuator1 := 15`,
`eHVACErrorCodes_Error_NoFeedbackActuator2 := 16`,
`eHVACErrorCodes_Error_NoFeedbackActuator3 := 17`,

eHVACErrorCodes_Error_NoFeedbackActuator4 := 18,
 eHVACErrorCodes_Error_NoFeedbackActuator5 := 19,
 eHVACErrorCodes_Error_NoFeedbackActuator6 := 20,
 eHVACErrorCodes_Error_NoFeedbackActuator7 := 21,
 eHVACErrorCodes_Error_NoFeedbackActuator8 := 22,

bErrorGeneral: error during evaluation of the operating feedback.

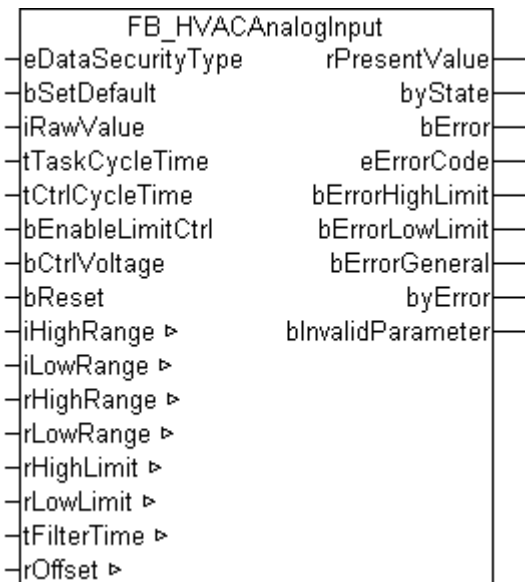
blInvalidParameter: TRUE if an error occurs during the plausibility check. The message must be acknowledged with *bReset*.

Documents about this

- example_persistent_e.zip (Resources/zip/11659714827.zip)

3.2 HVAC Analog modules

3.2.1 FB_HVACAnalogInput



Application


This function block serves the acquisition and scaling of analog input signals. Using the KL30xx, KL31xx und KL32xx terminals, the standard signals 0–20 mA, 4–20 mA, 0–10 V and 10–5000 Ohms can be acquired and converted to physical values.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
iRawValue         : INT;
tTaskCycleTime   : TIME;
tCtrlCycleTime   : TIME;
bEnableLimitCtrl : BOOL;
bCtrlVoltage     : BOOL;
bReset           : BOOL;
```

eDataSecurityType:if eDataSecurityType:= *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

iRawValue: the raw value is transferred from the terminal to the function block with the parameter `iRawValue`.

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 function block 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.

bEnableLimitCtrl: limit value monitoring is only activated if the variable `bEnableLimitCtrl` is TRUE. This way, limit value monitoring can be delayed with a timer until the heating or air conditioning system is in a controlled state. In the case of air conditioning systems, this is usually enabled by the system start program. See also `FB_HVACStartAirConditioning` regarding this point.

bCtrlVoltage: by means of checking the sensor supply voltage with the input `bCtrlVoltage`, error messages are suppressed if the supply voltage to the sensors is absent. If the sensor supply voltage is present, a TRUE is present at the input `bCtrlVoltage`.

bReset: acknowledge input in the event of an error. In addition the 2nd order filter can be synchronized via this input to the present measuring signal, so that this can be output at `rPresentValue`.

VAR_OUTPUT

```
rPresentValue      : REAL;
byState            : BYTE;
bError             : BOOL;
eErrorCode         : E_HVACErrorCodes;
bErrorHighLimit   : BOOL;
bErrorLowLimit    : BOOL;
bErrorGeneral     : BOOL;
byError           : BYTE;
bInvalidParameter : BOOL;
```

rPresentValue: determined output value.

byState: state of the function block.

`byState.1:= TRUE`, limit value monitoring is activated.

`byState.7:= TRUE`, sensor voltage supply is present.

bError: the variable `bError` becomes TRUE in the event of an internal error in the function block.

eErrorCode : contains the specific error code related to the `bError`.

bErrorHighLimit: TRUE if the upper limit value is reached.

bErrorLowLimit: TRUE if the lower limit value is reached.

bErrorGeneral: TRUE if a single error message from the process is present.

byError: output of the errors as byte.

`byError.0:= bError`

`byError.1:= bInvalidParameter`

byError.2:= bErrorGeneral

byError.3:= bErrorLowLimit is TRUE if the lower limit value is undershot.

byError.4:= bErrorHighLimit is TRUE if the upper limit value is exceeded.

InvalidParameter: TRUE if an error occurs during the plausibility check. The message must be acknowledged with *bReset*.

VAR_IN_OUT

```
iHighRange   : INT;
iLowRange    : INT;
rHighRange   : REAL;
rLowRange    : REAL;
rHighLimit   : REAL;
rLowLimit    : REAL;
tFilterTime  : TIME;
rOffset      : REAL;
```

iHighRange: upper raw value of the input variable *iRawValue*. The variable is saved persistently. Preset to 32767.

iLowRange: lower raw value of the input variable *iRawValue*. The variable is saved persistently. Preset to 0.

rHighRange: the upper scaled measured value. The variable is saved persistently. Preset to 100.

rLowRange: the lower scaled measured value. The variable is saved persistently. Preset to 0.

rHighLimit: if the scaled measured value is larger than the upper limit value *rHighLimit*, an impermissibly high measured value can be reached. The function block indicates this error by setting the variable *bErrHighLimit* to TRUE. The variable is saved persistently. Preset to 100.

rLowLimit: if the scaled measured value is smaller than the lower limit value *rLowLimit*, an impermissibly low measured value can be reached. The function block indicates this error by setting the variable *bErrLowLimit* to TRUE. The variable is saved persistently. Preset to 0.

tFilterTime: to avoid large fluctuations and jumps in the measuring signal, the function block is provided with two 1st order filters. Both filters work with the same time constant. The filter constants are determined by the variable *tFilterTime* [s] (0..3600). The variable is saved persistently. Preset to 2 s.

rOffset: with this offset the linear equation determined by means of the two conversion points is shifted parallel upwards or downwards. The variable is saved persistently. Preset to 0.

With the two value pairs *iHighRange/rHighRange* and *iLowRange/rLowRange* a linear conversion of the raw value into the physical unit takes place. *iHighRange* and *iLowRange* correspond to the raw values. *rHighLimit* and *rLowLimit* correspond to the scaled values in the physical unit of the signal to be measured.

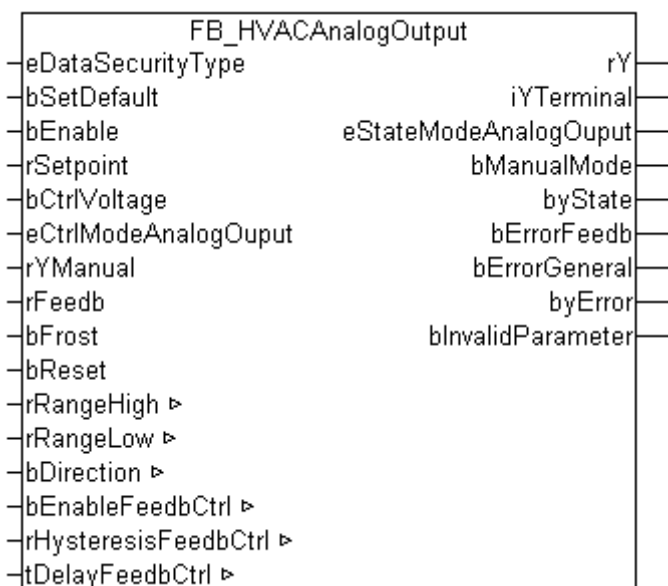
The output value is given by:

$$rPresentValue = [\{ (rHighRange - rLowRange) / (iHighRange - iLowRange) \} \times (iRawValue - iHighRange)] + rHighRange + rOffset$$

Documents about this

 [example_persistent_e.zip \(Resources/zip/11659714827.zip\)](#)

3.2.2 FB_HVACAnalogOutput



Application

This function block serves to control continuous actuators, such as valves or dampers, with positional feedback.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
rSetpoint         : REAL;
bCtrlVoltage      : BOOL;
eCtrlModeAnalogOutput : E_HVACAnalogOutputMode;
rYManual          : REAL;
rFeedb            : REAL;
bFrost           : BOOL;
bReset            : BOOL;
```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled if `bEnable = TRUE`. If the function block is not enabled the value zero is output at the output `rY`.

rSetpoint: the setpoint for the analog output is transferred to the function block with the variable *rSetpoint*.

bCtrlVoltage: the control voltage is present if the *bCtrlVoltage* variable is TRUE. The feedback control is deactivated in the event of failure of the control voltage so that no false alarms occur.

eCtrlModeAnalogOutput: Enum that defines the operation mode.

```
TYPE E_HVACAnalogOutputMode :
(
eHVACAnalogOutputMode_Auto_BMS := 0,
eHVACAnalogOutputMode_Manual_BMS := 1,
eHVACAnalogOutputMode_Auto_OP := 2,
eHVACAnalogOutputMode_Manual_OP := 3
);
END_TYPE
```

rYManual: analog input value, which is relayed to the output rY in manual mode.

rFeedb: analog position feedback from the actuator.

bFrost: this input serves to protect an air heater against frost. As soon as the input *bFrost* is TRUE, rY is set to the maximum size (*rRangeHigh*) and *iYTerminal* to 32767. The outputs rY and *iYTerminal* remain set until the input *bFrost* is FALSE again.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
rY : REAL;
iYTerminal : INT;
eStateModeAnalogOutput : E_HVACAnalogOutputMode;
bManualMode : BOOL;
byState : BYTE;
bErrorFeedb : BOOL;
bErrorGeneral : BOOL;
byError : BYTE;
bInvalidParameter : BOOL;
```

rY: indicates the current magnitude of the control signal in %.

iYTerminal: represents the size of the output signal scaled to the value range 0 to 32767.

eStateModeAnalogOutput: Enum that indicates the operation mode.

bManualMode: the analog output is in manual operation mode.

byState: status byte indicating the operating state of the function block.

```
byState.0 := bEnable;
byState.1 := bManualMode;
byState.2 := bFrost;
byState.7 := bCtrlVoltage;
```

bErrorFeedb: error feedback signal.

bErrorGeneral: this is a collective message for all function block errors.

byError: supplies all error messages and warnings.

```
byError.1 := bInvalidParameter
byError.2 := bErrorGeneral
byError.3 := bErrorFeedb
```

bInvalidParameter: TRUE if an error occurs during the plausibility check. The message must be acknowledged with *bReset*.

VAR_IN_OUT

```
rRangeHigh : REAL;
rRangeLow : REAL;
bDirection : BOOL;
bEnableFeedbCtrl : BOOL;
rHysteresisFeedbCtrl : REAL;
tDelayFeedbCtrl : TIME;
```

rRangeHigh/rRangeLow: these two variables define the value range of rY (0%..100%). The variable is saved persistently.

Example: $rSetpoint = 100$, $rRangeLow = 0$, $rRangeHigh = 200$,
 $\implies rY = 50$
 $\implies rYTerminal = 16384$

$rSetpoint = 200$, $rRangeLow = 0$, $rRangeHigh = 200$,
 $\implies rY = 100$
 $\implies rYTerminal = 32767$

bDirection: the output signal of rY is inverted by the variable $bDirection$. FALSE corresponds to the direct control direction. The variable is saved persistently.

bEnableFeedbCtrl: continuous actuators often have positional feedback. The function of the actuator is monitored by means of the positional feedback. Positional feedback monitoring is activated if the variable $bEnableFeedbCtrl$ is TRUE. The variable is saved persistently.

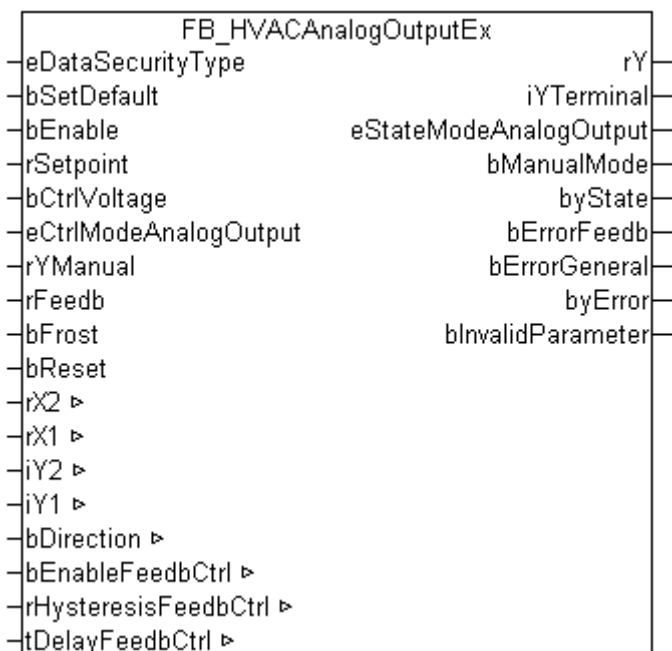
rHysteresisFeedbCtrl: due to the stroke time of typical drives used in heating and air conditioning systems, the feedback signal always lags in the case of a setpoint step-change for the position of the actuator. Using the variable $rHysteresisFeedbCtrl$, a range is specified within which the position setpoint rY of the actuator may deviate from the feedback signal (0..32767). The variable is saved persistently.

tDelayFeedbCtrl: if the difference between the set position and the actual position of the actuator is greater than $\pm rHysteresisFeedbCtrl$, then the response of the output $bErrorFeedb$ is delayed by the time of the timer $tDelayFeedbCtrl$ [s] (0s..50s). The variable is saved persistently.

Documents about this

📄 example_persistent_e.zip (Resources/zip/11659714827.zip)

3.2.3 FB_HVACAnalogOutputEx



Application

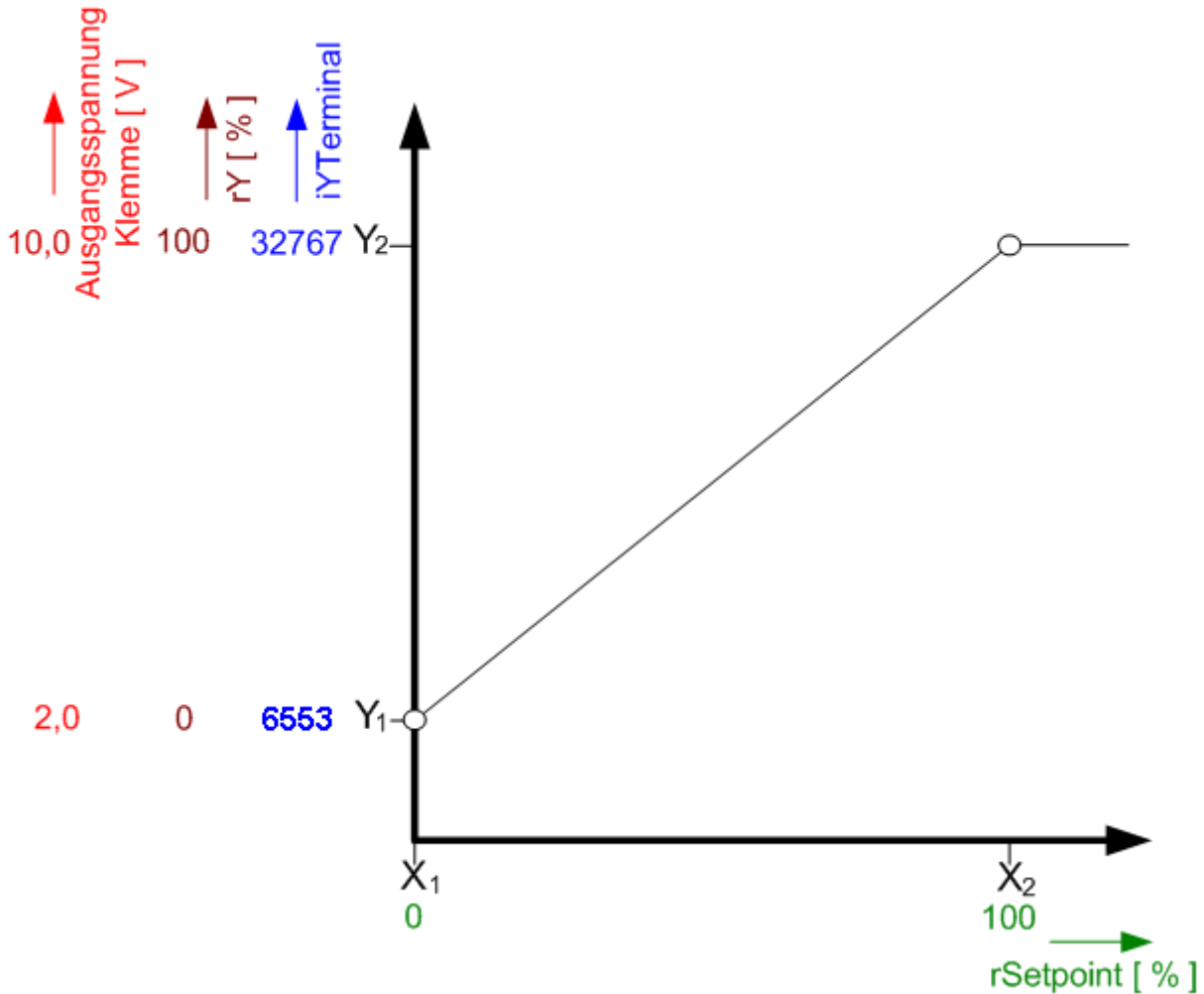
This function block is used to control continuous actuators such as valves or dampers with a positional feedback.

Compared to FB_HVACAnalogOutput, a scaling function is integrated in this function block. The function is: $y=m*x +b$.

Application example:

Damper actuator operating range 2-10 V with the KL4404 (signal voltage 0-10 V)

rX1 = 0 iY1 = 6553
rX2 = 100 iY2 = 32767




VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
rSetpoint         : REAL;
bCtrlVoltage      : BOOL;
eCtrlModeAnalogOutput: E_HVACAnalogOutputMode;
rYManual          : REAL;
rFeedb           : REAL;
bFrost           : BOOL;
bReset           : BOOL;
```

eDataSecurityType: if eDataSecurityType:= *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled if `bEnable = TRUE`. If the function block is not enabled the value zero is output at the output `rY`.

rSetpoint: the setpoint for the analog output is transferred to the function block with the variable `rSetpoint`.

bCtrlVoltage: the control voltage is present if the `bCtrlVoltage` variable is TRUE. The feedback control is deactivated in the event of failure of the control voltage so that no false alarms occur.

eCtrlModeAnalogOutput: Enum that defines the operation mode.

```
TYPE E_HVACAnalogOutputMode :
(
eHVACAnalogOutputMode_Auto_BMS := 0,
eHVACAnalogOutputMode_Manual_BMS := 1,
eHVACAnalogOutputMode_Auto_OP := 2,
eHVACAnalogOutputMode_Manual_OP := 3
);
END_TYPE
```

rYManual: analog input value, which is relayed to the output `rY` in manual mode.

rFeedb: analog position feedback from the actuator.

bFrost: this input serves to protect an air heater against frost. As soon as the input `bFrost` is TRUE, `rY` is set to the maximum size (`rRangeHigh`) and `iYTerminal` to 32767. The outputs `rY` and `iYTerminal` remain set until the input `bFrost` is FALSE again.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
rY                : REAL;
iYTerminal        : INT;
eStateModeAnalogOutput : E_HVACAnalogOutputMode;
bManualMode       : BOOL;
byState           : BYTE;
bErrorFeedb       : BOOL;
bErrorGeneral     : BOOL;
byError           : BYTE;
bInvalidParameter : BOOL;
```

rY: indicates the current magnitude of the control signal in % (0%..100%).

iYTerminal: represents the size of the output signal scaled to the value range 0 to 32767.

eStateModeAnalogOutput: Enum that indicates the operation mode.

bManualMode: the analog output is in manual operation mode.

byState: status byte indicating the operating state of the function block.

```
byState.0 := bEnable;
byState.1 := bManualMode;
byState.2 := bFrost;
byState.7 := bCtrlVoltage;
```

bErrorFeedb: error feedback signal.

bErrorGeneral: this is a collective message for all function block errors.

byError: supplies all error messages and warnings.

byError.1 := *bInvalidParameter*

byError.2 := *bErrorGeneral*

byError.3 := *bErrorFeedb*

bInvalidParameter: TRUE if an error occurs during the plausibility check. The message must be acknowledged with *bReset*.

VAR_IN_OUT

```
rX2          : REAL;
rX1          : REAL;
iY2          : INT;
iY1          : INT;
bDirection   : BOOL;
bEnableFeedbCtrl : BOOL;
rHysteresisFeedbCtrl : REAL;
tDelayFeedbCtrl : TIME;
```

rX2: upper limit value on the X axis (-32767..32767). The variable is saved persistently.

rX1: lower limit value on the X axis (-32767..32767). The variable is saved persistently.

rY2: upper limit value on the Y axis (0..32767). The variable is saved persistently.

rY1: lower limit value on the Y axis (0..32767). The variable is saved persistently.

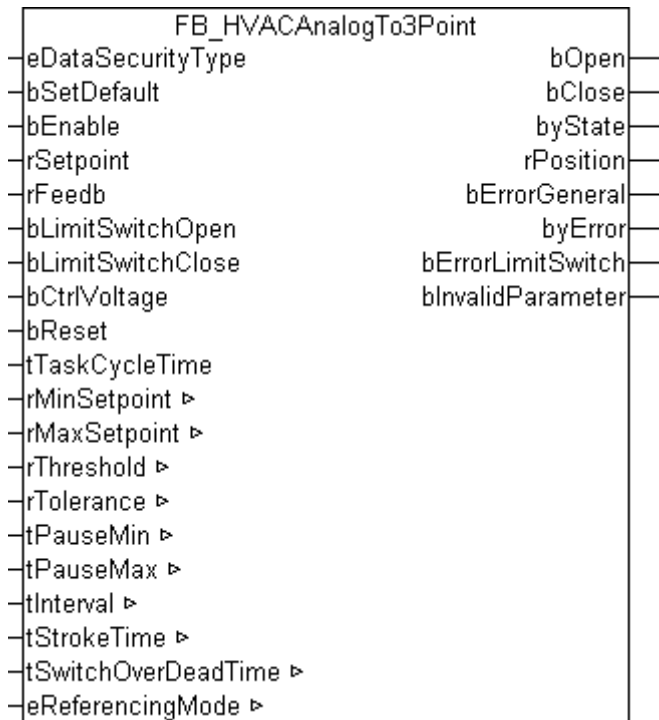
bDirection: the output signal of *rY* is inverted by the variable *bDirection*. FALSE corresponds to the direct control direction. The variable is saved persistently.

bEnableFeedbCtrl: continuous actuators often have positional feedback. The function of the actuator is monitored by means of the positional feedback. Positional feedback monitoring is activated if the variable *bEnableFeedbCtrl* is TRUE. The variable is saved persistently.

rHysteresisFeedbCtrl: due to the stroke time of typical drives used in heating and air conditioning systems, the feedback signal always lags in the case of a setpoint step-change for the position of the actuator. Using the variable *rHysteresisFeedbCtrl*, a range is specified within which the position setpoint *rY* of the actuator may deviate from the feedback signal (0..32767). The variable is saved persistently.

tDelayFeedbCtrl: if the difference between the set position and the actual position of the actuator is greater than +/- *rHysteresisFeedbCtrl*, then the response of the output *bErrorFeedb* is delayed by the time of the timer *tDelayFeedbCtrl* (0s..50s). The variable is saved persistently.

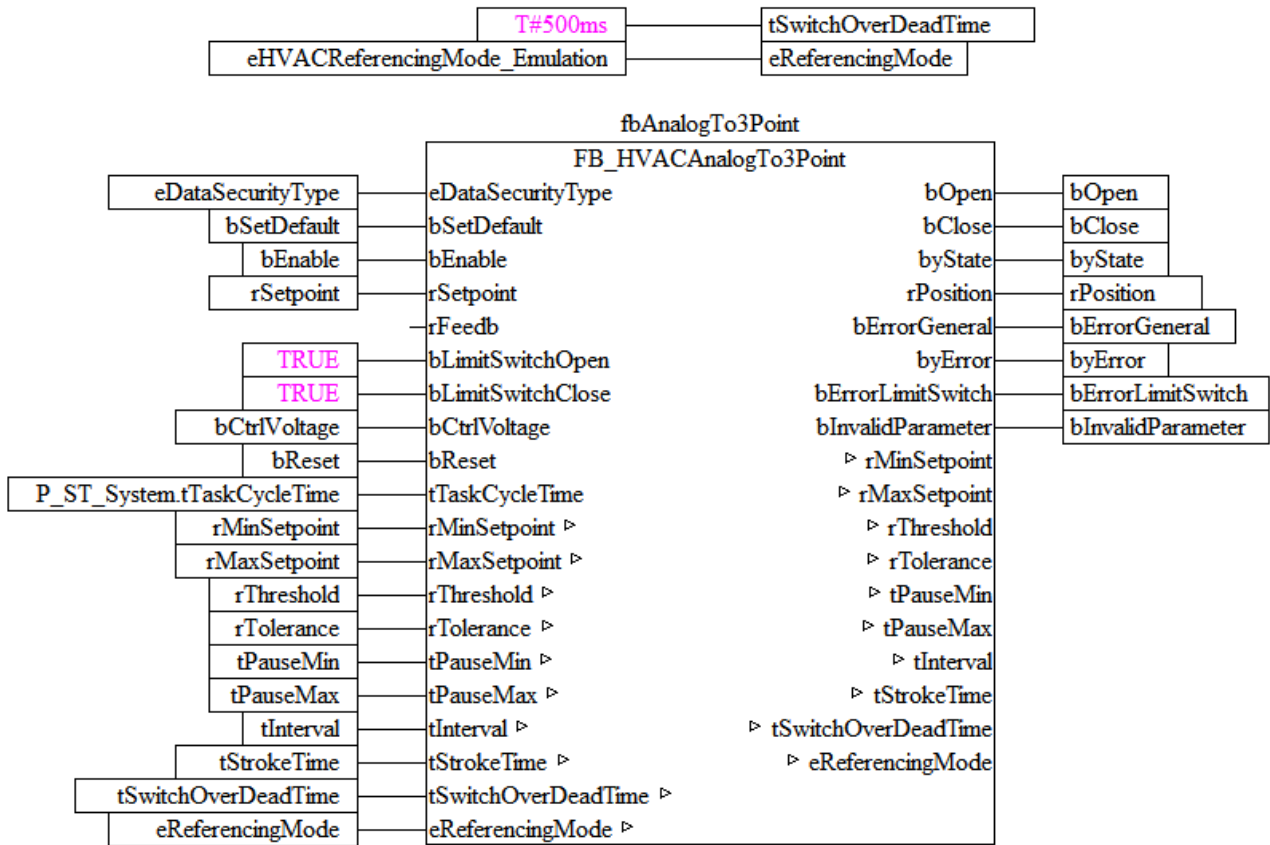
3.2.4 FB_HVACAnalogTo3Point



Application

This function block serves to convert an analog signal into a three-point step signal. Hence three-point dampers or valves can be controlled by a controller with a continuous control signal. The function block *FB_HVACAnalogTo3Point* works with or without a continuous positional feedback signal from the drive.

Example: three-point actuator without feedback and limit switch




VAR_INPUT

```

eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
rSetpoint         : REAL;
rFeedb            : REAL;
bLimitSwitchOpen : BOOL;
bLimitSwitchClose: BOOL;
bCtrlVoltage      : BOOL;
bReset            : BOOL;
tTaskCycleTime   : TIME;
rMinSetpoint     : REAL;
rMaxSetpoint     : REAL;
rThreshold       : REAL;
rTolerance       : REAL;
tPauseMin        : REAL;
tPauseMax        : REAL;
tInterval        : REAL;
tStrokeTime      : REAL;
tSwitchOverDeadTime : REAL;
eReferencingMode : EReferencingMode;
    
```

eDataSecurityType: if eDataSecurityType:= eHVACDataSecurityType_Persistent, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If eDataSecurityType:= eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType:= eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled if *bEnable* = TRUE. If control voltage is present and there are no errors and *bEnable* = FALSE, then the output *bClose* = is TRUE and the output *bOpen* is FALSE. If the reference mode *eReferencingMode* = *eHVACReferencingMode_Emulation*, then the output *bClose* remains set until either the limit switch is reached, the enable signal is present again or there is an error. If the reference mode *eReferencingMode* = *eHVACReferencingMode_AnalogFeedback*, then the output *bClose* remains set until either the enable signal is present again, or there is a fault or *rFeedback* = *rMinSetpoint*.

rSetpoint: the setpoint for the position of the damper or the valve is transferred to the function block with the variable *rSetpoint*. The parameters *rMaxSetpoint* and *rMinSetpoint* define the value range of *rSetpoint*. If an incorrect variable value is present at *rSetpoint*, then the last valid variable value is used, if available. If there is no valid, last value, then work continues internally with *rMinSetpoint*. *blInvalidParameter* is set if the variable value is incorrect, the function block continues to operate normally.

rFeedb: the input variable *rFeedb* is only active if the referencing mode of the function block *eReferencingMode* = *eHVACReferencingMode_AnalogFeedback*. The actual position of the drive is fed back with the scaled input signal *rFeedb*. The scaling of *rFeedb* must correspond to the scaling of *rSetpoint*. The parameters *rMaxSetpoint* and *rMinSetpoint* define the value range of *rFeedb*. *rFeedb* is returned to the controller via the output variable *rPosition*.

If there is an incorrect variable value at *rFeedb*, then the last valid variable value is taken, if available. If there is no valid, last value, then work continues internally with *rMinSetpoint*. *blInvalidParameter* is set if the variable value is incorrect, the function block continues to operate normally.

blLimitSwitchOpen/blLimitSwitchClose: if the three-point actuator has limit switches, these can be connected to the inputs *blLimitSwitchOpen* and *blLimitSwitchClose*.

The limit switches are used by the function block in the referencing mode *eReferencingMode* = *eHVACReferencingMode_Emulation* for referencing. Depending on the direction of driving, the currently calculated position *rPosition* is automatically set to the corresponding value of *rMaxSetpoint*/*rMinSetpoint* by a falling edge on one of the two inputs *blLimitSwitchOpen*/*blLimitSwitchClose*. This also takes place when the drive is at a standstill..

Regardless of which referencing mode the function block is in, the associated output *bOpen* or *bClose* remains TRUE after the respective limit position switch has been reached.

The limit switches must be connected to the function block as break contacts. If none exist, a TRUE must be applied to the two inputs *blLimitSwitchOpen*/*blLimitSwitchClose*.

bCtrlVoltage: a check via the input variable *bCtrlVoltage* is made of whether control voltage is present. Both outputs *bOpen* and *bClose* are set to FALSE in the event of failure of the control voltage. Since many error messages are based on the quiescent current principle, there would be a torrent of error messages following failure of the control voltage. Therefore all error messages are suppressed in the event of a control voltage failure in the *FB_HVACAnalogTo3Point*.

The control voltage is present if a TRUE is present at input *bCtrlVoltage*.

bReset: acknowledge input in the event of a fault.

tTaskCycleTime: cycle time with which the function block is called. This corresponds to the task cycle time of the calling task if the function block is called in every cycle.

If an incorrect time value of T#0s is applied to *tTaskCycleTime*, then *tTaskCycleTime* is internally set to T#1ms. *blInvalidParameter* is set if the variable value is incorrect.

VAR_OUTPUT

```
bOpen          : BOOL;
bClose         : BOOL;
byState        : BYTE;
rPosition      : REAL;
bErrorGeneral  : BOOL;
byError        : BYTE;
bErrorLimitSwitch : BOOL;
blInvalidParameter : BOOL;
```

bOpen: output for opening the three-point actuator.

bClose: output for closing the three-point actuator.

byState: status byte indicating the operating state of the function block.

byState.0 := *bEnable*

byState.1 := *bOpen*

byState.2 := *bClose*
byState.3 := *bLimitSwitchOpen*
byState.4 := *bLimitSwitchClose*
byState.7 := *bCtrlVoltage*

rPosition: with *rPosition* either the measured or the calculated position of the drive is fed back to the controller.

If *eReferencingMode* = *eHVACReferencingMode_AnalogFeedback*, then *rFeedb* is returned to the controller via the output variable *rPosition*. If *rFeedb* > *rMaxSetpoint*, then *rPosition* = *rMaxSetpoint* and *blInvalidParameter* is set. If *rFeedb* < *rMinSetpoint*, then *rPosition* = *rMinSetpoint* and *blInvalidParameter* is set.

If *eReferencingMode* = *eHVACReferencingMode_Emulation*, then the actual position of the drive will be emulated on the basis of a calculation and fed back to the controller via *rPosition*.

bErrorGeneral: the error message *bErrorGeneral* becomes TRUE as soon as *bErrorGeneralLimitSwitch* = TRUE. The outputs *bOpen* and *bClose* are then set to FALSE and are only enabled again when the error has been rectified and acknowledged via *bReset*.

byError: returns all error messages and warnings of the function block

byError.1 := *blInvalidParameter*
byError.2 := *bErrorGeneral*
byError.3 := *bErrorLimitSwitch*

bErrorLimitSwitch: becomes TRUE if both limit switches are activated simultaneously or if ((*rMaxSetpoint* - *rThreshold*) < *rSetpoint*) and *bOpen* = TRUE) or ((*rMinSetpoint* + *rThreshold*) > *rSetpoint*) and *bClose* = TRUE). The error *bErrorLimitSwitch* can only occur if *eReferencingMode* = *eEmulation*.

blInvalidParameter: indicates that an incorrect parameter is present at one of the variables *rMinSetpoint*, *rMaxSetpoint*, *rThreshold*, *rTolerance*, *tPauseMin*, *tPauseMax*, *tInterval*, *tStrokeTime*, *tSwitchOverDeadTime*, *eReferencingMode*, *rSetpoint* or *rFeedback*. An incorrect parameter entry does not lead to a standstill of the function block; see description of variables. After rectifying the incorrect parameter entry, the message *blInvalidParameter* must be acknowledged via *bReset*.

VAR_IN_OUT

VAR_IN_OUT

```
rMinSetpoint      : REAL;
rMaxSetpoint      : REAL;
rThreshold        : REAL;
rTolerance        : REAL;
tPauseMin         : TIME;
tPauseMax         : TIME;
tInterval         : TIME;
tStrokeTime       : TIME;
tSwitchOverDeadTime : TIME;
eReferencingMode  : E_HVACReferencingMode;
```

rMinSetpoint/rMaxSetpoint: the parameters *rMaxSetpoint* (0..32767) and *rMinSetpoint* (0..32767) define the value range of *rSetpoint*. *rMaxSetpoint* must be greater than *rMinSetpoint*. In addition, the two variables *rMaxSetpoint* and *rMinSetpoint* are included in the calculation of the variable pulse-pause modulation, see Figure 1.1.

If an incorrect variable value is present, then the last valid variable value is used, if available. If there is no valid last value, operation continues with the preset value. *blInvalidParameter* is set if the parameter is incorrect, the function block continues to operate normally. The variables are stored persistently. *rMinSetpoint* preset to 0. *rMaxSetpoint* preset to 100.

rThreshold/rTolerance: if the difference between the position setpoint *rSetpoint* and the calculated or measured actual position value *rPosition* of the actuator is greater than the threshold value set by the variable *rThreshold* (0.001..32767), then the function block begins to correct the position by cycling the outputs *bOpen* or *bClose*, depending on the control deviation value. Correction continues until the deviation is smaller than the value *rTolerance* (0.001..32767). A hysteresis loop for the opening and closing movement of the drive is defined by the value of *rThreshold* – *rTolerance*. This is necessary in order to prevent the drive reacting to the smallest changes of the control value. Wear on the drive and the relay is thus reduced. *rThreshold* must be greater than *rTolerance*.

rSetpoint - *rPosition* > = *rThreshold* control of *bOpen*
rPosition - *rSetpoint* > = *rThreshold* control of *bClose*

If an incorrect variable value is present, the last valid variable value is used, if available. If there is no valid last value, then the preset value is used. *blInvalidParameter* is set if the parameter is incorrect, the function block continues to operate normally. The variables are stored persistently. *rThreshold* preset to 5.0. *rTolerance* preset to 2.0.

tPauseMin/tPauseMax/tInterval: a variable pulse/pause modulation for cycling the three-point actuator can be set with the parameters *tPauseMin* [s] (0..3600), *tPauseMax* [s] (0..3600) and *tInterval* [s] (0..3600). The pause length is defined as a function of the control deviation by the parameters *tPauseMin* and *tPauseMax*. In the case of a small deviation the pause time is long in relation to the interval time. Analogously, the pause time is short if the deviation is large. The effective driving speed of the three-point actuator is thus higher for larger control deviations than it is for smaller ones; see following fig. The following condition must be fulfilled in any case: $tPauseMin < tPauseMax < tInterval$

If a wrong variable value is present, then the last valid variable value is used, if available. If there is no valid last value, then the preset value is used. *blInvalidParameter* is set if the parameter is incorrect, the function block continues to operate normally. The variables are stored persistently. *tPauseMin* preset to 2s. *tPauseMax* preset to 8s. *tInterval* preset to 10s.

tStrokeTime: the variable *tStrokeTime* specifies the complete stroke time of the drive. If the drive has no continuous position feedback, the actual position of the drive will be emulated on the basis of a calculation. For this reason the precise input of the total stroke time of the actuator is important. The following condition must be met in any case: $tStrokeTime > tInterval$

If an incorrect variable value is present, then the last valid variable value is used, if available. If there is no valid last value, then the preset value is used. *blInvalidParameter* is set if the parameter is incorrect, the function block continues to operate normally. The variable is saved persistently. Preset to 200 s.

tSwitchOverDeadTime: dwell time at a change of direction (0..3600). During this time, both outputs are reset.

If an incorrect variable value is present, the last valid variable value is used, if available. If there is no valid last value, then the preset value is used. *blInvalidParameter* is set if the parameter is incorrect, the function block continues to operate normally. The variable is saved persistently. Preset to 500 ms.

eReferencingMode: Enum via which the referencing mode of the function block is specified.

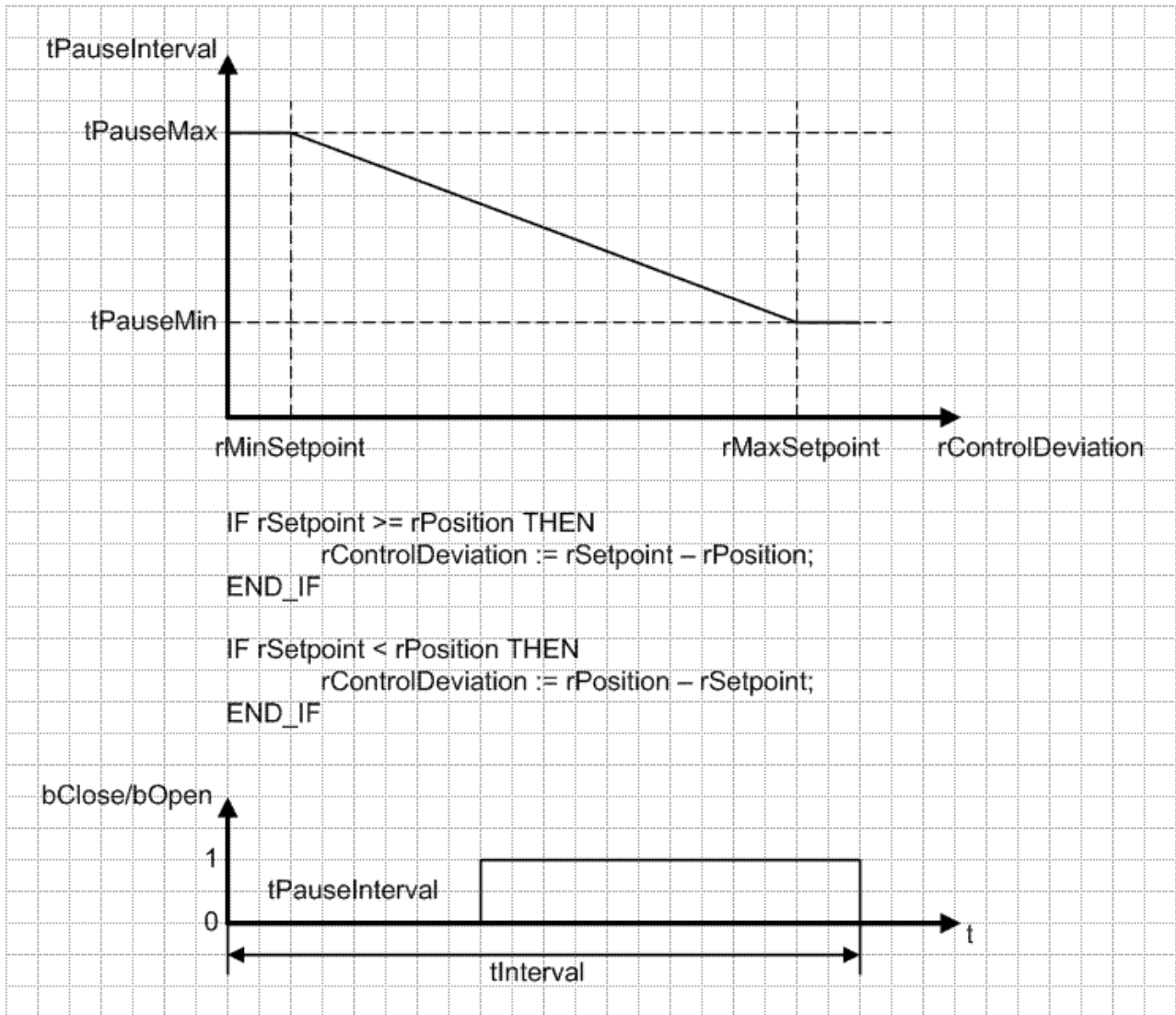
Depending on the equipment of the actuator used, the position is referenced depending on *eReferencingMode*.

If *eReferencingMode* = *eHVACReferencingMode_Emulation*, then the position of the actuator is calculated on the basis of the control of *bOpen* and *bClose* by means of *tStrokeTime*. However, as the operating hours of the three-point valve or three-point damper increase, deviations could occur due to mechanical inaccuracies in the drive. In order to achieve automatic matching of the actual and calculated positions, one of the outputs *bOpen* or *bClose* will be set to TRUE upon reaching the calculated position *rPosition* of *rMaxSetpoint* or *rMinSetpoint* and it is correspondingly given the value of the parameter *rMaxSetpoint* or *rMinSetpoint*. A falling edge on the signal inputs *bLimitSwitchOpen* and *bLimitSwitchClose* is used to reference the position of the drive to *rMaxSetpoint* or *rMinSetpoint*. After reaching the respective limit switch, the corresponding output *bOpen* or *bClose* will remain set to TRUE.

If *eReferencingMode* = *eHVACReferencingMode_AnalogFeedback* the position of the drive is transferred by means of the signal *rFeedback*. After reaching the respective limit switch, the corresponding output *bOpen* or *bClose* will remain set to TRUE.

If an incorrect variable value is present, then the last valid variable value is used, if available. If there is no valid last value, then the preset value is used. *blInvalidParameter* is set if the parameter is incorrect, the function block continues to operate normally. The variable is saved persistently. Preset to 0.

Fig. 1.1



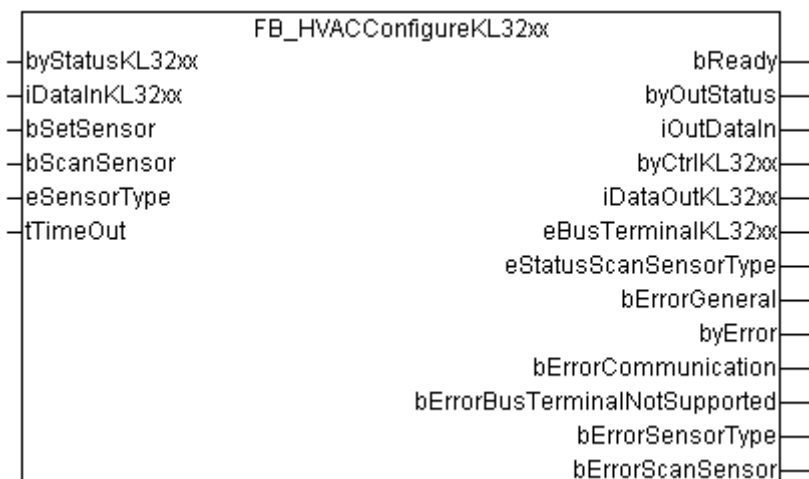
Example

tInterval	120 seconds
tPauseMin	10 seconds
tPauseMax	60 seconds
rSetpoint	100%
rPosition	50%
tPauseInterval / bClose OR bOpen	35 seconds/85 seconds
rSetpoint	50%
rPosition	25%
tPauseInterval / bClose OR bOpen	47.5 seconds/72.5 seconds

Documents about this

example_persistent_e.zip (Resources/zip/11659714827.zip)

3.2.5 FB_HVACConfigureKL32xx



Application

With the function block *FB_HVACConfigureKL32xx*, the type of sensor of a channel of the KL3201/02/04, the KL3208-0010 or the KL3228 can be set from the PLC. To do this, the variables *byStatusKL32xx*, *iDataInKL32xx*, *byCtrlKL32xx* and *iDataOutKL32xx* must be linked in the TwinCAT System Manager with the variables Status, Data In, Control and Data Out of a channel of the KL3201/02/04 or the KL3228. The resistor elements specified in Table 1 can be selected with the Enum *eSensorType*. The KS2000 configuration software is no longer needed for setting the temperature sensor.

i When measuring resistances from 10 to 5000 Ω with the KL32xx, 1 digit = 0.5 Ω, i.e. the indicated raw value must be divided by 2 in the PLC. Example: 2500 Ω would be represented in the controller by a raw value of 5000. The raw value must be divided by 2 in the PLC in order to arrive at the ohmic value of 2500 Ω.

i The measurement of resistances from 10 to 10000 Ω is possible only with the special terminal KL320x-0027. Exclusively the resistance measurement 10...10000 Ω can be performed on this special terminal.

i The EL3692 EtherCAT Terminal is a resistance measurement terminal that covers the measuring range up to 10 MΩ.

Application example

Download	Required library
TcHVAC.pro [► 531]	TcHVAC.lib

VAR_INPUT

```

byStatusKL32xx : BYTE;
iDataInKL32xx  : INT;
bSetSensor     : BOOL;
bScanSensor    : BOOL;
eSensorType    : E_HVACSensorType;
tTimeOut       : TIME;

```

byStatusKL32xx: the status byte (Status) of the Bus Terminal must be assigned here in the TwinCAT System Manager for the respective channel. If this is not the case, then this is indicated as an error with *bErrorCommunication* if the function block has been activated with a rising edge on *bSetSensor*/*bScanSensor*. If scanning or setting has been concluded with a rising edge on *bSetSensor*/*bScanSensor*, then this is indicated with a TRUE on *bReady* and *byOutStatus := byStatusKL32xx*. If scanning or setting is activated, *bReady* = FALSE, then the value of *byStatusKL32xx* present before activation is output on *byOutStatus*.

iDataInKL32xx: the raw value (Data In) of the Bus Terminal must be assigned here in the TwinCAT System Manager for the respective channel. If this is not the case, then this is indicated as an error with *bErrorCommunication* if the function block has been activated with a rising edge on *bSetSensor/bScanSensor*. If scanning or setting has been concluded with a rising edge on *bSetSensor/bScanSensor*, then this is indicated with a TRUE on *bReady* and *iOutDataIn := iDataInKL32xx*. If scanning or setting is activated, *bReady* = FALSE, then the value of *iDataInKL32xx* present before activation is output on *iOutDataIn*.

bSetSensor: the type of sensor specified on the Enum *eSensorType* is set in the Bus Terminal with a rising edge on *bSetSensor*. If the procedure has been concluded, scanning is activated internally in the function block and the type of sensor that has been set is indicated via the Enum *eStatusScanSensorType*.

bScanSensor: the scanning of the respective channel of the Bus Terminal is activated by a rising edge on *bScanSensor*. The type of sensor that has been set is indicated via the Enum *eStatusScanSensorType*.

eSensorType: Enum with which the sensor type for the respective channel is specified for setting via *bSetSensor*.

tTimeOut: specifies the time until the abortion of the function and *bErrorCommunication* goes TRUE. If no time is specified, then T#5s is used internally in the function block.

VAR_OUTPUT

```

bReady                : BOOL;
byOutStatus           : BYTE;
iOutDataIn            : INT;
byCtrlKL32xx          : BYTE;
iDataOutKL32xx        : INT;
eBusTerminalKL32xx    : E_HVACBusTerminal_KL32xx;
eStatusScanSensorType : E_HVACSensorType;
bErrorGeneral         : BOOL;
byError               : BYTE;
bErrorCommunication   : BOOL;
bErrorBusTerminalNotSupported : BOOL;
bErrorSensorType      : BOOL;
bErrorScanSensor      : BOOL;

```

bReady: if the function block is activated via a rising edge on *bSetSensor/bScanSensor*, then *bReady* goes FALSE. If scanning or the setting of the type of sensor has been concluded, then *bReady* goes TRUE.

byOutStatus: if scanning or setting has been concluded with a rising edge on *bSetSensor/bScanSensor*, then this is indicated with a TRUE on *bReady* and *byOutStatus := byStatusKL32xx*. If scanning or setting is activated, *bReady* = FALSE, then the value of *byStatusKL32xx* present before activation is output on *byOutStatus*.

iOutDataIn: if scanning or setting has been concluded with a rising edge on *bSetSensor/bScanSensor*, then this is indicated with a TRUE on *bReady* and *iOutDataIn := iDataInKL32xx*. If scanning or setting is activated, *bReady* = FALSE, then the value of *iDataInKL32xx* present before activation is output on *iOutDataIn*.

byCtrlKL32xx: the control byte (control) of the Bus Terminal must be assigned here in the TwinCAT System Manager for the respective channel. If this is not the case, then this is indicated as an error with *bErrorCommunication* if the function block has been activated with a rising edge on *bSetSensor/bScanSensor*.

iDataOutKL32xx: the data output (Data Out) of the Bus Terminal must be assigned here in the TwinCAT System Manager for the respective channel. If this is not the case, then this is indicated as an error with *bErrorCommunication* if the function block has been activated with a rising edge on *bSetSensor/bScanSensor*.

eBusTerminalKL32xx: Enum that displays the type of Bus Terminal if the function block has been activated with a rising edge on *bSetSensor/bScanSensor*.

eStatusScanSensorType: Enum that displays the type of sensor for the respective channel if the function block has been activated with a rising edge on *bSetSensor/bScanSensor*.

bErrorGeneral: goes TRUE as soon as either *bErrorCommunication*, *bErrorBusTerminalNotSupported*, *bErrorSensorType* or *bErrorScanSensor* is TRUE.

byError: returns all error messages and warnings of the function block

byError.2 := bErrorGeneral;

byError.3 := bErrorCommunication;

byError.4 := bErrorBusTerminalNotSupported;

byError.5 := bErrorSensorType;

byError.6 := bErrorScanSensor;

bErrorCommunication: goes TRUE if, for example, the variables *byCtrlKL32xx*, *iDataOutKL32xx*, *byStatusKL32xx* and *iDataInKL32xx* have not been assigned to a channel of the Bus Terminal.

Communication to the Bus Terminal is then interrupted during scanning or the setting of the type of sensor. *bErrorCommunication* likewise goes TRUE if the *tTimeOut* time is not sufficient when scanning or setting the sensor type.

bErrorBusTerminalNotSupported: becomes TRUE if the Bus Terminal is not supported by the function block.

bErrorSensorType: becomes TRUE if the type of sensor specified at *eSensorType* is not supported by the Bus Terminal.

bErrorScanSensor: becomes TRUE if, when scanning, the type of sensor is not supported by the Bus Terminals specified in the Enum *eBusTerminalKL32xx*.

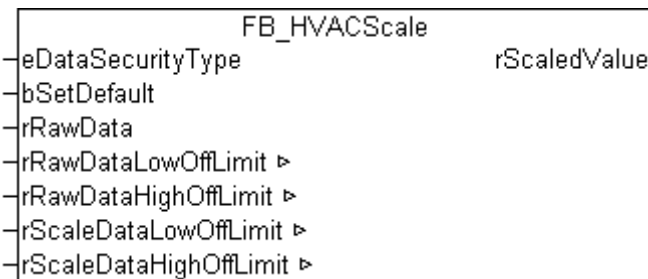
Table 1: Table 1: types of sensor selectable via the Enum *eTemperatureCharacteristic*

Sensor type	KL3201/2/4-000		KL3201/2/4-0010		KL3201/2/4-0012		KL3201/2/4-0014		KL3201/2/4-0016		KL3201/2/4-0020	
		Raw value		Raw value		Raw value		Raw value		Raw value		Raw value
PT100	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C
PT200	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C
PT500	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C
PT1000	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C
Ni100	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C
Ni120	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C
Ni1000(DIN)	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C
Ni1000(Tk500,LS)	not supported		not supported		not supported		not supported		not supported		not supported	
Resistance measurement 10 ... 1200 Ω	x		x		x		x		x		x	
Resistance measurement 10 ... 5000 Ω	x		x		x		x		x		x	

Sensor type	KL3201/2/4-000	KL3201/2/4-0010	KL3201/2/4-0012	KL3201/2/4-0014	KL3201/2/4-0016	KL3201/2/4-0020
	Raw value	Raw value	Raw value	Raw value	Raw value	Raw value
PT100	x	1/10 °C	x	1/10 °C	x	1/10 °C
PT200	x	1/10 °C	x	1/10 °C	x	1/10 °C
Resistance measurement 10 ... 10000 Ω	not supported	not supported	not supported	not supported	not supported	not supported
Sensor type	KL3201/2/4-0023	KL3201/2/4-0025	KL3201/2/4-0029	KL3201/2/4-0031	KL3228	KL3208-0010
	Raw value	Raw value	Raw value	Raw value	Raw value	Raw value
PT100	x	1/10 °C	x	1/10 °C	x	1/10 °C
PT200	x	1/10 °C	x	1/10 °C	x	1/10 °C
PT500	x	1/10 °C	x	1/10 °C	x	1/10 °C
PT1000	x	1/10 °C	x	1/10 °C	x	1/10 °C
Ni100	x	1/10 °C	x	1/10 °C	x	1/10 °C
Ni120	x	1/10 °C	x	1/10 °C	x	1/10 °C
Ni1000(DIN)	x	1/10 °C	x	1/10 °C	x	1/10 °C
Ni1000(Tk5000,LS)	not supported	not supported	x	1/10 °C	not supported	x
Resistance measurement 10 ... 1200 Ω	x	x	x	x	not supported	not supported
Resistance measurement 10 ... 5000 Ω	x	x	x	x	not supported	not supported

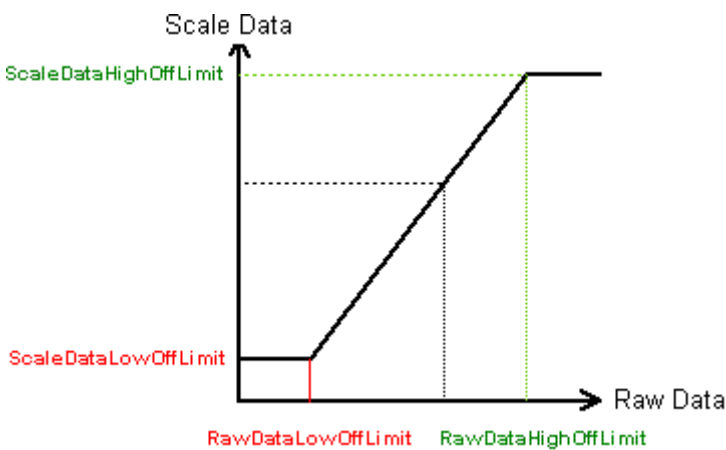
Sensor type	KL3201/2/4-000		KL3201/2/4-0010		KL3201/2/4-0012		KL3201/2/4-0014		KL3201/2/4-0016		KL3201/2/4-0020	
		Raw value		Raw value		Raw value		Raw value		Raw value		Raw value
PT100	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C
PT200	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C	x	1/10 °C
Resistance measurement 10 ... 10000 Ω	not supported		not supported		not supported		not supported		not supported		not supported	

3.2.6 FB_HVACScale



Application

A raw analog value is scaled to the specified measuring range and returned as the function value. If the raw value extends beyond the upper or lower measuring range, the corresponding limit value is output. There must be a difference of at least 0.01 between the upper and lower limit values for the raw data. If this is not the case, the lower limit value is output. The difference between the limits is necessary so as to avoid division by zero during the internal calculation of the linear equation.




VAR_INPUT

```

eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
rRawData          : REAL;
    
```

eDataSecurityType: if `eDataSecurityType` [▶ 520] := `eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType` := `eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE	
<p>A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if <code>eDataSecurityType</code> := <code>eHVACDataSecurityType_Persistent</code>. It would lead to early wear of the flash memory.</p>	

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

rRawData: raw value from the terminal.

VAR_OUTPUT

```
rScaledValue          : REAL;
```

rScaledValue: scaled value.

VAR_IN_OUT

```
rRawDataLowOffLimit  : REAL;
rRawDataHighOffLimit : REAL;
rScaleDataLowOffLimit : REAL;
rScaleDataHighOffLimit : REAL;
```

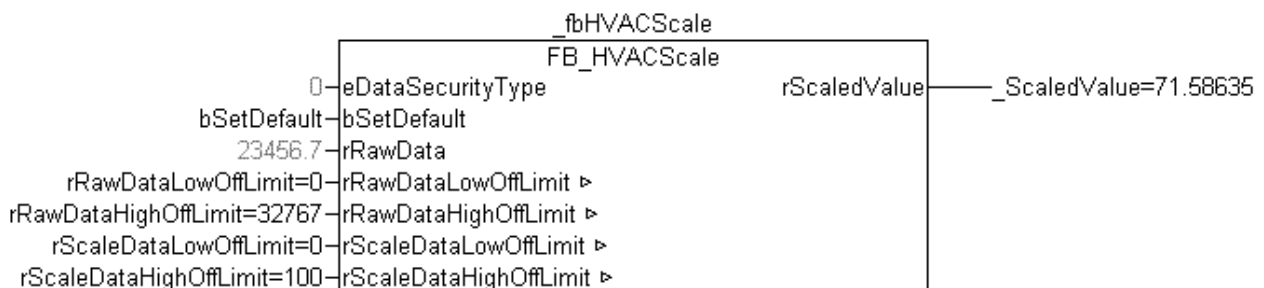
rRawDataLowOffLimit: lower limit from raw value. Preset to 0.

rRawDataHighOffLimit: upper limit from raw value. Preset to 32767.

rScaleDataLowOffLimit: lower limit of the scaled measured value. Preset to 0.

rScaleDataHighOffLimit: upper limit of the scaled measured value. Preset to 100.

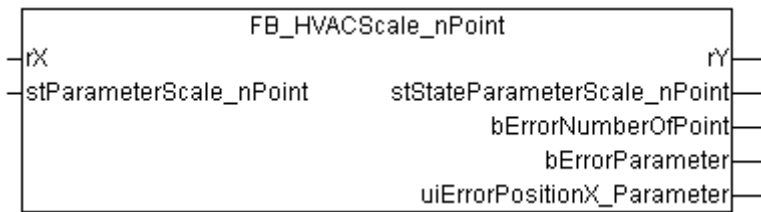
Example in FBD:



Documents about this

-  example_persistent_e.zip (Resources/zip/11659714827.zip)

3.2.7 FB_HVACScale_nPoint



Application

Curves can be reproduced in the PLC with the function block *FB_HVACScale_nPoint*. These can be, for example, positive or negative temperature coefficients (PTC/NTC). The analog raw resistance value of a PTC/NTC is applied to the input *rX* and output as a temperature value via the output *rY*. The individual parameters and the number of individual points of the X-Y axis of the characteristic curve are specified with the structure *stParameterScale_nPoint*.

The KL32xx Bus Terminals can be used to display analog raw resistance values in the PLC. Detailed information can be found in the documentation on *FB_HVACConfigureKL32xx* [▶ 74].

i When measuring resistances from 10 to 5000 Ω with the KL32xx, 1 digit = 0.5 Ω, i.e. the indicated raw value must be divided by 2 in the PLC. Example: 2500 Ω would be represented in the controller by a raw value of 5000. The raw value must be divided by 2 in the PLC in order to arrive at the ohmic value of 2500 Ω.

i The measurement of resistances from 10 to 10000 Ω is possible only with the special terminal KL320x-0027. Exclusively the resistance measurement 10...10000 Ω can be performed on this special terminal.

i The EL3692 EtherCAT Terminal is a resistance measurement terminal that covers the measuring range up to 10 MΩ.

Application example

Download	Required library
TcHVAC.pro [▶ 531]	TcHVAC.lib

VAR_INPUT

```
rX : REAL;
stParameterScale_nPoint: ST_HVACParameterScale_nPoint;
```

rX: raw value that scales to the indicated measured value of the structure *stParameterScale_nPoint* and is output via *rY*. The raw value can be, for example, the raw resistance value of the KL32xx. The terminal must be set to resistance measurement for this. Detailed information on the procedure can be found in the documentation *FB_HVACConfigureKL32xx* [▶ 74].

stParameterScale_nPoint: structure via which the individual points of the X-Y coordinates are given their valency. Regardless of which curve is to be reproduced, the following conditions must be met: either $stParameterScale_nPoint.rX[1] < stParameterScale_nPoint.rX[2] < stParameterScale_nPoint.rX[n]$ OR $stParameterScale_nPoint.rX[1] > stParameterScale_nPoint.rX[2] > stParameterScale_nPoint.rX[n]$. *stParameterScale_nPoint.iNumberOfPoint* must not be smaller than 2 OR larger than *g_iMaxNoOfScale_nPoint(60)*.

stParameterScale_nPoint.rX[1..iNumberOfPoint] - array which contains the valence of the single points of the X axis. The number of points specified depends on *iNumberOfPoint*.

stParameterScale_nPoint.rY[1..iNumberOfPoint] - array containing the valency of the individual points on the Y axis. The number of points specified depends on *iNumberOfPoint*.

VAR_OUTPUT

```

rY                : REAL;
stStateParameterScale_nPoint: ST_HVACParameterScale_nPoint;
bErrorNumberOfPoint : BOOL;
bErrorParameter     : BOOL;
uiErrorPositionX_Parameter : INT;

```

rY: *rY* is the scaled temperature value of the specified raw value *rX*.

stStateParameterScale_nPoint: state of the structure *stParameterScale_nPoint*. If *bErrorNumberOfPoint* or *bErrorParameter* = TRUE, the status of the individual parameters in the structure *stStateParameterScale_nPoint* = 0. If *stParameterScale_nPoint.iNumberOfPoint* = 20, the individual parameters of *stStateParameterScale_nPoint.rX[1..20]* / *stStateParameterScale_nPoint.rY[1..20]* are displayed, from 21 to 60 the status = 0.

bErrorNumberOfPoint: If *iNumberOfPoint* < 2 or *iNumberOfPoint* > *g_iMaxNoOfScale_nPoint*, then *bErrorNumberOfPoint* = TRUE and 0 is output at the output variable *rY*. If the error has been rectified, then *bErrorNumberOfPoint* goes FALSE.

bErrorParameter: when parameterizing the structure *stScaleTemperatureCharacteristic* care must be taken that either

stParameterScale_nPoint.rX[1] > *stParameterScale_nPoint.rX[2]* > *stParameterScale_nPoint.rX[n]* OR
stParameterScale_nPoint.rX[1] < *stParameterScale_nPoint.rX[2]* < *stParameterScale_nPoint.rX[n]*.

If this is not adhered to, *bErrorParameter* goes TRUE. If the error has been rectified, then *bErrorParameter* goes FALSE. The exact field position in the array *stParameterScale_nPoint.rX[uiErrorPositionX_Parameter]* is specified via the output variables *uiErrorPositionX_Parameter*.

uiErrorPositionX_Parameter: if *bErrorParameter* = TRUE, then the exact field position in the array *stParameterScale_nPoint.rX[uiErrorPositionX_Parameter]* at which the error has occurred is indicated by *uiErrorPositionX_Parameter*.

VAR_GLOBAL CONSTANT

```
g_iMaxNoOfScale_nPoint: INT := 60;
```

g_iMaxNoOfScale_nPoint: global constant that specifies the maximum number of points of the XY coordinates of the structure *SST_HVACParameterScale_nPoint* [► 526].

Formulas for the linear equation, two-point form:

m = gradient

$$m = (Y_2 - Y_1) / (X_2 - X_1)$$

$$m = (stParameterScale_nPoint.rY[2] - stParameterScale_nPoint.rY[1]) / (stParameterScale_nPoint.rX[2] - stParameterScale_nPoint.rX[1])$$

$$Y = Y_1 + m * (X - X_1)$$

$$rY = rY[1] + m * (rX - stParameterScale_nPoint.rX[1])$$

Example curve 1 with calculation

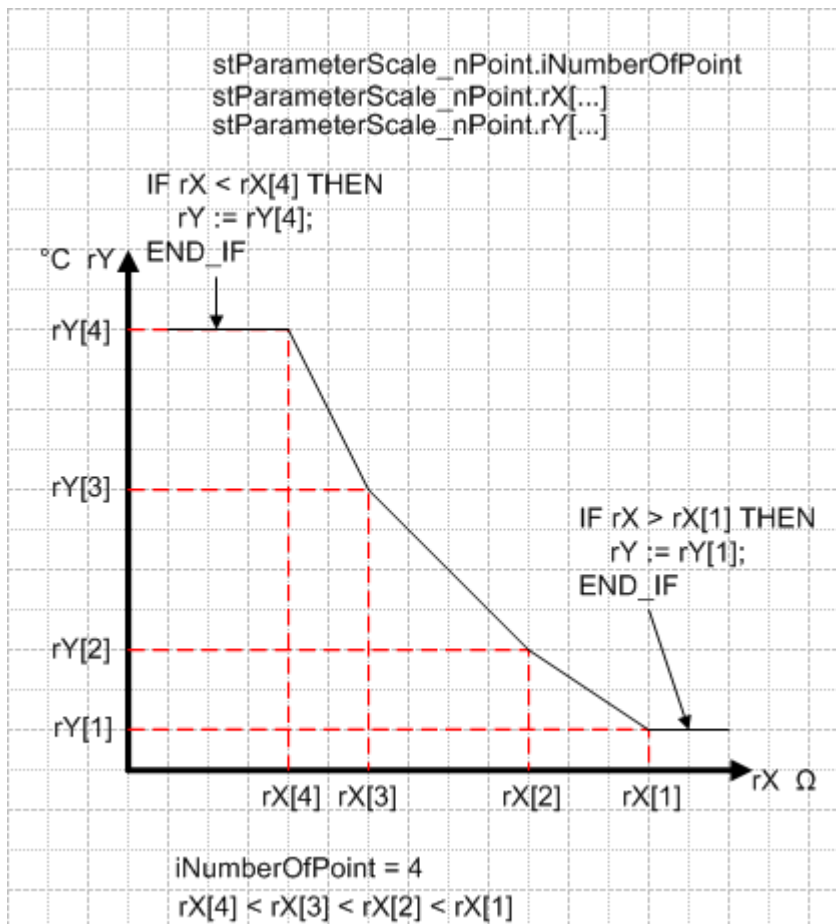


Fig. 1: FB_HVACScale_nPoint_2

m = rM = slope

X = rX = 1199.8

X2 = stParameterScale_nPoint.rX[2] = 1209

X3 = stParameterScale_nPoint.rX[3] = 1163

Y2 = stParameterScale_nPoint.rY[2] = 20

Y3 = stParameterScale_nPoint.rY[3] = 21

$m = (Y3 - Y2) / (X3 - X2)$

$rM = (stParameterScale_nPoint.rY[3] - stParameterScale_nPoint.rY[2]) / (stParameterScale_nPoint.rX[3] - stParameterScale_nPoint.rX[2])$

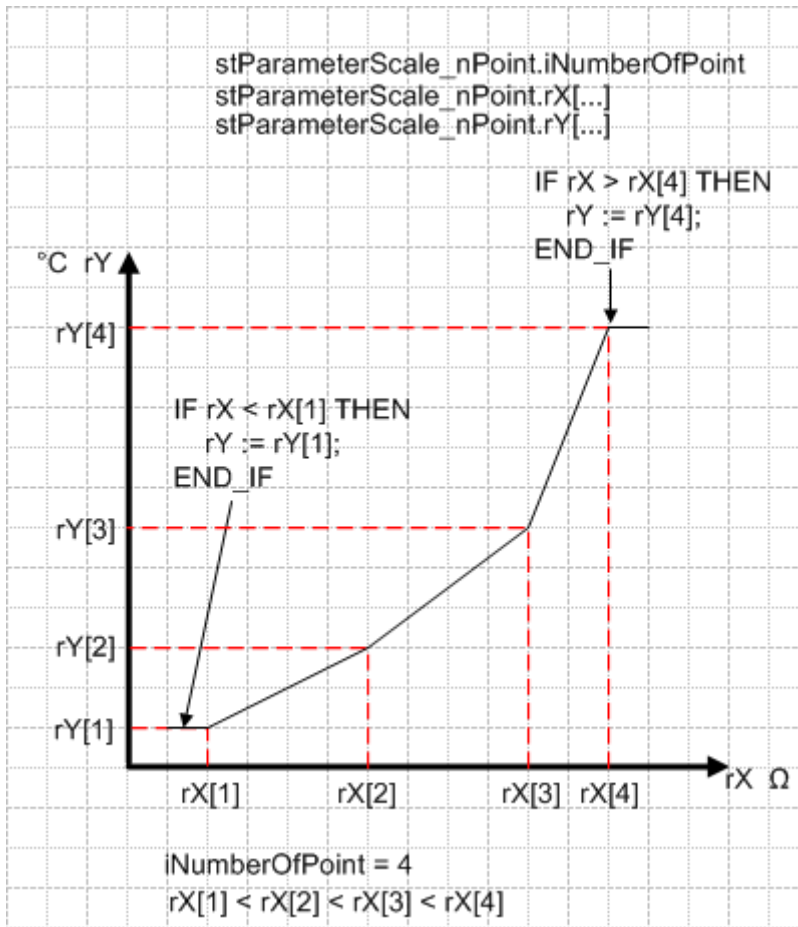
$rM = (21 - 20) / (1163 - 1209) = -0.02174$

$Y = Y2 + m * (X - X2)$

$rY = stParameterScale_nPoint.rY[2] + rM * (rX - stParameterScale_nPoint.rX[2])$

$rY = 20 + -0.02174 * (1199.8 - 1209) = 20.2$

Example curve 2 with calculation



m = rM = slope

X = rX = 1124

X3 = stParameterScale_nPoint.rX[2] = 1112

X4 = stParameterScale_nPoint.rX[3] = 1142

Y3 = stParameterScale_nPoint.rY[2] = 20

Y4 = stParameterScale_nPoint.rY[3] = 25

$$m = (Y4 - Y3) / (X4 - X3)$$

$$rM = (stParameterScale_nPoint.rY[4] - stParameterScale_nPoint.rY[3]) / (stParameterScale_nPoint.rX[4] - stParameterScale_nPoint.rX[3])$$

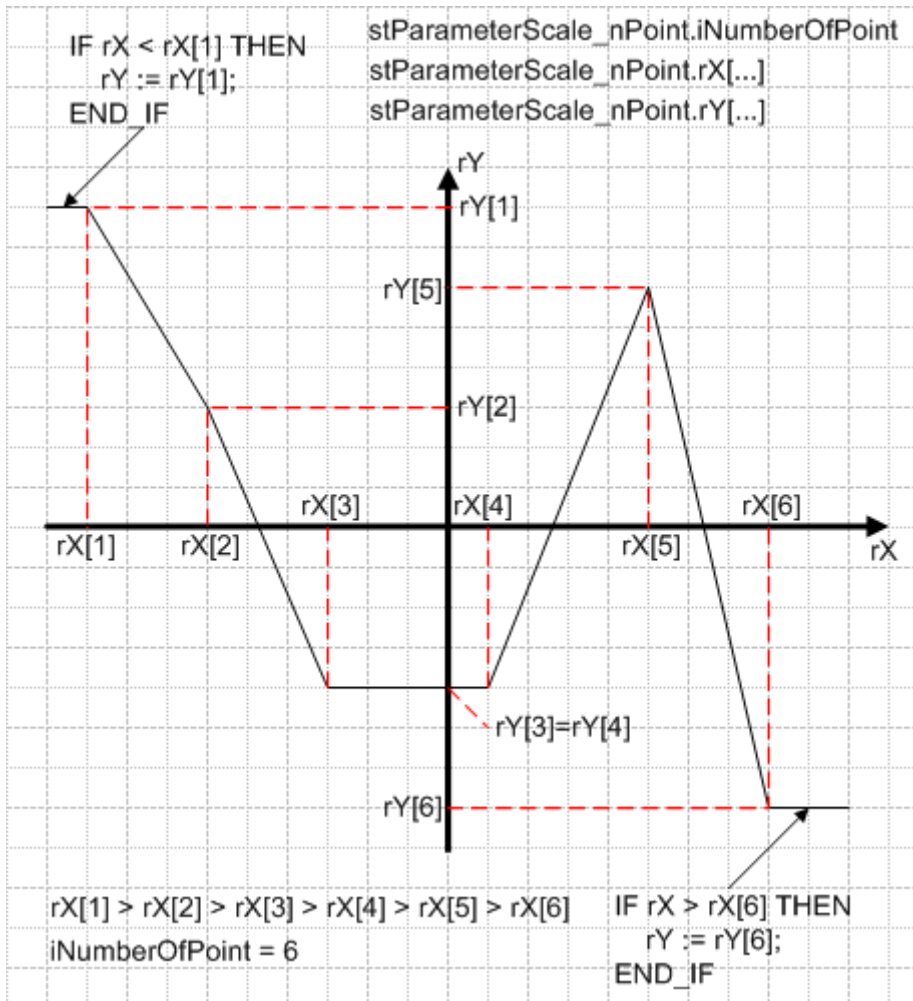
$$rM = (25 - 20) / (1142 - 1112) = 0.166$$

$$Y = Y3 + m * (X - X3)$$

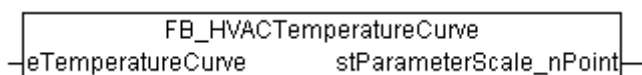
$$rY = stParameterScale_nPoint.rY[3] + rM * (rX - stParameterScale_nPoint.rX[3])$$

$$rY = 20 + 0.166 * (1124 - 1112) = 21.992$$

Example curve 3



3.2.8 FB_HVACTemperatureCurve



Application

A temperature curve stored internally in the function block is selected from Tables 1 and 2 via the Enum *eTemperatureCurve* and output via the structure *stParameterScale_nPoint*. Characteristic curves can then be reproduced with this structure in conjunction with the function block *FB_HVACScale_nPoint* [► 80]. In order to arrive at the raw resistance value of a sensor, the KL32xx Bus Terminals must be set to resistance measurement; see *FB_HVACConfigureKL32xx* [► 74].

- i** When measuring resistances from 10 to 5000 Ω with the KL32xx, 1 digit = 0.5 Ω , i.e. the indicated raw value must be divided by 2 in the PLC. Example: 2500 Ω would be represented in the controller by a raw value of 5000. The raw value must be divided by 2 in the PLC in order to arrive at the ohmic value of 2500 Ω .
- i** The measurement of resistances from 10 to 10000 Ω is possible only with the special terminal KL320x-0027. Exclusively the resistance measurement 10...10000 Ω can be performed on this special terminal.



The EL3692 EtherCAT Terminal is a resistance measurement terminal that covers the measuring range up to 10 MΩ.

Application example

Download	Required library
TcHVAC.pro [▶ 531]	TcHVAC.lib

VAR_INPUT

```
eTemperatureCurve : E_HVACTemperatureCurve;
```

eTemperatureCurve: Enum via which a temperature curve stored internally in the function block can be selected from Tables 1 and 2.

VAR_OUTPUT

```
stParameterScale_nPoint : ST_HVACParameterScale_nPoint;
```

stParameterScale_nPoint: structure containing the number of points and their valence of the X-Y coordinates. stParameterScale_nPoint contains the characteristic curves stored in the tables 1 and 2 depending on the specification via the Enum eTemperatureCurve.

Example:

```
eTemperatureCurve = eHVACTemperatureCurve_Ni1000Tk5000_TCR
stParameterScale_nPoint.iNumberOfPoint := 56;
stParameterScale_nPoint.rX[1] := 790.8;
stParameterScale_nPoint.rY[1] := -50.0;
stParameterScale_nPoint.rX[2] := 826.8;
stParameterScale_nPoint.rY[2] := -40.0;
.
.
.
stParameterScale_nPoint.rX[56] := 1625.4;
stParameterScale_nPoint.rY[56] := 120.0;
stParameterScale_nPoint.rX[57] := 0;
stParameterScale_nPoint.rY[57] := 0;
.
.
.
stParameterScale_nPoint.rX[60] := 0;
stParameterScale_nPoint.rY[60] := 0;
```

Table 2: Table 1: specified temperature curves, selectable via the Enum eTemperatureCurve

	S+S Sensor type 1K3 A1 NTC 1 kOhm	S+S Sensor type 1.8K3 A1 NTC 1.8 kOhm	S+S Sensor type 2.2K3 A1 NTC 2.2 kOhm	S+S Sensor type 3.3K3 A1 NTC 3 kOhm	S+S Sensor type NI1000 DIN	S+S Sensor type PT1000 DIN	S+S Sensor type Ni1000/ TCR (LAN1) Ni1000Tk5 000
eHVACTe mpera- tureChar- acteristic_	NTC1k_3_ A1	NTC1k8_3_ _A1	NTC2k2_3_ _A1	NTC3k3_3_ _A1	Ni1000_DI N	Pt1000_DI N	Ni1000Tk5 000_TCR
°C	Ω	Ω	Ω	Ω	Ω	Ω	Ω
- 50	32886				743	803	790.8
- 40	18641				791	843	826.8

	S+S Sensor type 1K3 A1 NTC 1 kOhm	S+S Sensor type 1.8K3 A1 NTC 1.8 kOhm	S+S Sensor type 2.2K3 A1 NTC 2.2 kOhm	S+S Sensor type 3.3K3 A1 NTC 3 kOhm	S+S Sensor type NI1000 DIN	S+S Sensor type PT1000 DIN	S+S Sensor type Ni1000/ TCR (LAN1) Ni1000Tk5 000
eHVACTe mpera- tureChar- acteristic_ °C	NTC1k_3_ A1 Ω	NTC1k8_3_ _A1 Ω	NTC2k2_3_ _A1 Ω	NTC3k3_3_ _A1 Ω	Ni1000_DI N Ω	Pt1000_DI N Ω	Ni1000Tk5 000_TCR Ω
- 30	11130	21695	27886	53093	842	882	871.7
- 20	6777	12987	16502	29125	893	922	913.4
- 15	5341	10153	12844	21887	920	941	934.7
- 10	4247	8011	10070	16599	946	961	956.2
- 5	3390	6347	8134	12698	973	980	978
0	2728	5071	6452	9795	1000	1000	1000
1	2613	4851	6164	9309			1004.4
2	2503	4640	5891	8849			1008.9
3	2399	4441	5631	8415			1013.3
4	2300	4252	5384	8005			1017.8
5	2205	4071	5150	7617	1028	1020	1022.3
6	2115	3899	4927	7251			1026.7
7	2030	3738	4715	6905			1031.2
8	1948	3582	4513	6575			1035.8
9	1870	3434	4321	6265			1040.3
10	1796	3294	4138	5971	1056	1039	1044.8
11	1724	3158	3964	5691			1049.3
12	1656	3029	3797	5427			1053.9
13	1590	2905	3639	5177			1058.4
14	1528	2788	3488	4938			1063
15	1469	2677	3345	4713	1084	1058	1067.6
16	1412	2570	3207	4500			1072.2
17	1358	2468	3076	4298			1076.8
18	1306	2371	2952	4104			1081.4
19	1256	2277	2832	3922			1086
20	1209	2189	2719	3747	1112	1078	1090.7
21	1163	2103	2610	3582			1095.3
22	1120	2023	2506	3426			1100
23	1078	1945	2407	3277			1104.6
24	1038	1871	2289	3135			1109.3
25	1000	1800	2200	3000	1142	1098	1114
26	963	1732	2115	2872			1120
27	928	1667	2034	2750			1123.4
28	894	1604	1957	2634			1128.1
29	862	1545	1883	2522			1132.9
30	831	1488	1812	2417	1171	1117	1137.6
35	694	1235	1500	1960	1200	1136	1161.5
40	583	1031	1248	1597	1230	1155	1185.7

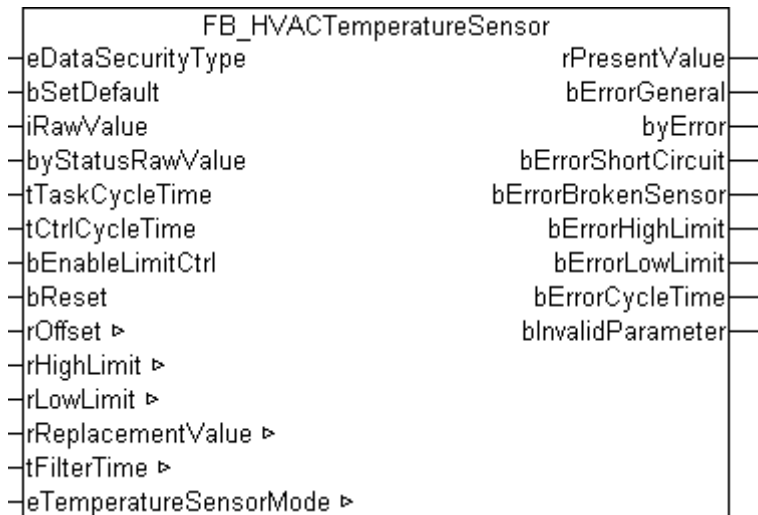
	S+S Sensor type 1K3 A1 NTC 1 kOhm	S+S Sensor type 1.8K3 A1 NTC 1.8 kOhm	S+S Sensor type 2.2K3 A1 NTC 2.2 kOhm	S+S Sensor type 3.3K3 A1 NTC 3 kOhm	S+S Sensor type NI1000 DIN	S+S Sensor type PT1000 DIN	S+S Sensor type Ni1000/ TCR (LAN1) Ni1000Tk5 000
eHVACTe mpera- tureChar- acteristic_ °C	NTC1k_3_ A1 Ω	NTC1k8_3_ _A1 Ω	NTC2k2_3_ _A1 Ω	NTC3k3_3_ _A1 Ω	Ni1000_DI N Ω	Pt1000_DI N Ω	Ni1000Tk5 000_TCR Ω
45	491	865	1043	1310	1261	1175	1210.2
50	416	729	876	1081	1291	1194	1235
55	354	616	738	896	1322	1213	1260.1
60	302	524	626	746	1353	1232	1285.4
65	259	447	532	625	1385	1252	1311.1
70	223	383	454	526	1417	1271	1337.1
75	192	329	390	444	1450	1290	1363.5
80	167	284	335	346	1483	1309	1390.1
85	145	246	289	321	1516	1328	1417.1
90	127	214	251	275	1549	1347	1444.4
95	111	187	218	236	1584	1366	1472
100	97	163	190	204	1618	1385	1500
105	88	143	167	176			1528.3
110	76	126	146	138	1688	1423	1557
115		111		120			1586
120		99		105	1760	1461	1625.4
125		88		92			
130		80			1833	1498	
140		62		64	1909	1536	
150		50		50	1987	1573	
160					2066	1611	
170					2148	1648	
180					2232	1685	
190						1722	
200						1758	
210						1795	
220						1832	
230						1868	
240						1905	
250						1941	
260						1977	
270						2013	
280						2049	
290						2085	
300						2121	
310						2156	
320						2191	
330						2227	

	S+S Sensor type 1K3 A1 NTC 1 kOhm	S+S Sensor type 1.8K3 A1 NTC 1.8 kOhm	S+S Sensor type 2.2K3 A1 NTC 2.2 kOhm	S+S Sensor type 3.3K3 A1 NTC 3 kOhm	S+S Sensor type NI1000 DIN	S+S Sensor type PT1000 DIN	S+S Sensor type Ni1000/ TCR (LAN1) Ni1000Tk5 000
eHVACTe mpera- tureChar- acteristic_ °C	NTC1k_3_ A1 Ω	NTC1k8_3_ _A1 Ω	NTC2k2_3_ _A1 Ω	NTC3k3_3_ _A1 Ω	Ni1000_DI N Ω	Pt1000_DI N Ω	Ni1000Tk5 000_TCR Ω
340						2262	
350						2297	
360						2332	
370						2367	
380						2401	
390						2436	
400						2470	

Table 3: Table 2: specified temperature curves, selectable via the Enum eTemperatureCurve

	Thermokon Sensor type NTC 1.8 kOhm	Thermokon Sensor type Ni1000 Tk5000
eHVACTemperatureCharacteris- tic_ °C	NTC1K8 Ω	Ni1000Tk5000 Ω
- 50		790.88
- 40		830.83
- 30	24500	871.69
- 20	14000	913.48
- 10	8400	956.24
0	5200	1000
10	3330	1044.79
20	2200	1090.65
25	1800	1113.99
30	1480	1137.61
40	1040	1185.71
50	740	1234.97
60	540	1285.44
70	402	1337.14
80	306	1390.12
90	240	1444.39
100	187	1500
110	149	1556.98
120	118	1615.36
130	95	1675.18
140	77	1736.47
150	64	1799.26

3.2.9 FB_HVACTemperatureSensor



Application


This function block serves the acquisition and further processing of temperature values for sensor types PT100, PT200, PT1000, NI100, NI120 and NI1000. The function block *FB_HVACTemperatureSensor* is matched to the KL320x Bus Terminals. These Bus Terminals can either be ordered preconfigured or set to the corresponding sensor types in the software.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
iRawValue         : INT;
byStatusRawValue  : BYTE;
tTaskCycleTime    : TIME;
tCtrlCycleTime    : TIME;
bEnableLimitCtrl  : BOOL;
bReset            : BOOL;
```

eDataSecurityType: if *eDataSecurityType := eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block *FB_HVACPersistentDataHandling* must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If *eDataSecurityType := eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if *eDataSecurityType := eHVACDataSecurityType_Persistent*. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

iRawValue: raw value of the temperature sensor in 1/10 °C from the Bus Terminal.

byStatusRawValue: status byte of the temperature sensor from the Bus Terminal. Serves for error diagnosis, e.g. wire break or short circuit.

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. *tTaskCycleTime* must be longer than $T\#0ms$.

tCtrlCycleTime: the variable *tCtrlCycleTime* specifies the cycle time with which the 2nd order filter is processed internally. The cycle time *tCtrlCycleTime* must be greater than or equal to *tTaskCycleTime*. If this is not the case, an error *bErrorCycleTime* occurs and either the replacement value *rReplacementValue* or the last valid measured value will be output at *rPresentValue*, depending on the mode *eTemperatureSensorMode* that is set.

bEnableLimitCtrl: enables *rHighLimit* and *rLowLimit* for limit monitoring

bReset: acknowledge input in the event of an error. In addition the 2nd order filter can be synchronized via this input to the present measuring signal, so that this can be output at *rPresentValue*.

VAR_OUTPUT

```
rPresentValue      : REAL;
bErrorGeneral      : BOOL;
byError            : BYTE;
bErrorShortCircuit : BOOL;
bErrorBrokenSensor : BOOL;
bErrorHighLimit    : BOOL;
bErrorLowLimit     : BOOL;
bErrorCycleTime    : BOOL;
bInvalidParameter  : BOOL;
```

rPresentValue: temperature output variable with one decimal place, $rPresentValue = (iRawValue / 10.0) + rOffset$, if *bErrorShortCircuit* or *bErrorBrokenSensor* = TRUE, then *rPresentValue* depends on the operation mode *eTemperatureSensorMode*.

bErrorGeneral: the error message *bErrorGeneral* becomes TRUE as soon as one of the error messages *bErrorHighLimit*, *bErrorLowLimit*, *bErrorCycleTime*, *bErrorShortCircuit* or *bErrorBrokenSensor* = TRUE. The value of the output variable *rPresentValue* is then dependent on the operation mode *eTemperatureSensorMode* and is enabled once the error has been rectified and, depending on the operation mode, acknowledged with *bReset*.

byError: returns all error messages and warnings,

```
byError.1 := bInvalidParameter
byError.2 := bErrorGeneral
byError.3 := bErrorHighLimit
byError.4 := bErrorLowLimit
byError.5 := bErrorShortCircuit
byError.6 := bErrorBrokenSensor
byError.7 := bErrorCycleTime
```

bErrorShortCircuit: error, short circuit at the connected temperature sensor. After rectification of the error, acknowledgment depends on the operation mode.

bErrorBrokenSensor: error, wire break in the connected temperature sensor. After rectification of the error, acknowledgment depends on the operation mode.

bErrorHighLimit: warning upper limit value exceeded; becomes TRUE if $rPresentValue \geq rHighLimit$. The warning that the upper limit value has been exceeded can only be acknowledged if $rPresentValue \leq rHighLimit - 1.0$ for a time duration of 5 seconds.

bErrorLowLimit: warning lower limit value undershot; becomes TRUE if $rPresentValue \leq rLowLimit$. The warning that the lower limit value has been undershot can only be acknowledged if $rPresentValue \geq rLowLimit + 1.0$ for a time duration of 5 seconds

bErrorCycleTime: error caused by an incorrect time input at the input variables *tTaskCycleTime* and *tCtrlCycleTime*, which must be acknowledged after rectification of the error.

bInvalidParameter: indicates that an incorrect parameter is present at one of the variables *rOffset*, *rHighLimit*, *rLowLimit*, *rReplacementValue*, *tFilterTime* and *eTemperatureSensorMode*. An incorrect parameter specification does not lead to a standstill of the function block; see description of variables. After rectifying the incorrect parameter entry, the message *bInvalidParameter* must be acknowledged via *bReset*.

VAR_IN_OUT

```

rOffset          : REAL;
rHighLimit       : REAL;
rLowLimit        : REAL;
rReplacementValue : REAL;
tFilterTime      : TIME;
eTemperatureSensorMode : E_HVACTemperatureSensorMode;

```

rOffset: temperature compensation in Kelvin (-50..+50), $rPresentValue = (iRawValue / 10.0) + rOffset$. The variable is saved persistently. Preset to 0.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used.

bInvalidParameter will be set in the event of an incorrect parameter entry.

rHighLimit: upper limit (-250..+850), if $rPresentValue \geq rHighLimit$, then the output *bErrorHighLimit* is set. *rHighLimit* must be greater than *rLowLimit*. The variable is saved persistently. Preset to 120.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *bInvalidParameter* will be set in the event of an incorrect parameter entry.

rLowLimit: lower limit (-250..+850), if $rPresentValue \leq rLowLimit$, then the output *bErrorLowLimit* is set. The variable is saved persistently. Preset to -60.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *bInvalidParameter* will be set in the event of an incorrect parameter entry.

rReplacementValue: replacement value (-250..+850), that is output at *rPresentValue* in the case of the errors *bErrorShortCircuit* and *bErrorBrokenSensor*, if the selected operation mode *eTemperatureSensorMode* = *eHVACTemperatureSensorMode_ReplacementValue* or *eTemperatureSensorMode* = *eHVACTemperatureSensorMode_AutoResetReplacementValue*. The variable is saved persistently. Preset to 0.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *bInvalidParameter* will be set in the event of an incorrect parameter entry.

tFilterTime: filter constant (1ms..100s). To avoid large fluctuations and jumps in the measuring signal, the function block is provided with a 2nd order filter. Upon a restart of the controller, or following correction of the error *bErrorShortCircuit* or *bErrorBrokenSensor*, the 2nd order filter is synchronized immediately and additionally after 2 seconds with the present measuring signal, so that the latter is output at *rPresentValue*. The measuring signal can be synchronized via the input *bReset* during running operation. The variable is saved persistently. Preset to 10 s.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *bInvalidParameter* will be set in the event of an incorrect parameter entry.

eTemperatureSensorMode: Enum that specifies the operation mode of the function block. The variable is saved persistently. Preset to 3%.

eTemperatureSensorMode = *eHVACTemperatureSensorMode_ReplacementValue*: if *bErrorShortCircuit* or *bErrorBrokenSensor* = TRUE, then $rPresentValue = rReplacementValue$. After correction of the error, the function block must be acknowledged with a rising edge at the input variable *bReset*.

eTemperatureSensorMode = *eHVACTemperatureSensorMode_LastValue*: if *bErrorShortCircuit* or *bErrorBrokenSensor* = TRUE, then the last valid temperature value that was present 10 seconds previously is output at *rPresentValue*. After correction of the error, it must be acknowledged with a rising edge at the input variable *bReset*.

eTemperatureSensorMode = *eHVACTemperatureSensorMode_AutoResetReplacementValue* : if *bErrorShortCircuit* or *bErrorBrokenSensor* = TRUE, then $rPresentValue = rReplacementValue$. The function block acknowledges itself automatically following correction of the error.

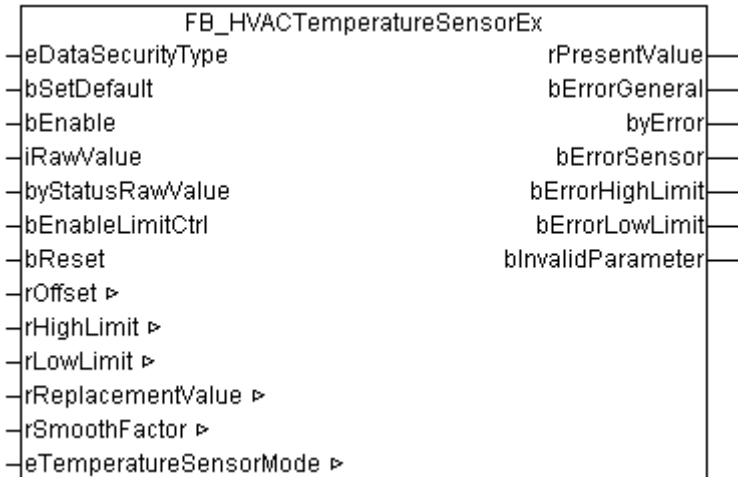
eTemperatureSensorMode = *eHVACTemperatureSensorMode_AutoResetLastValue*: if *bErrorShortCircuit* or *bErrorBrokenSensor* = TRUE, then the last valid temperature value that was present 10 seconds previously is output at *rPresentValue*. The function block acknowledges itself automatically following correction of the error.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. `blInvalidParameter` will be set in the event of an incorrect parameter entry.

Documents about this

 `example_persistent_e.zip` (Resources/zip/11659714827.zip)

3.2.10 FB_HVACTemperatureSensorEx



Application

This function block is used for the acquisition and subsequent processing of temperature values, e.g. for the sensor types PT100, PT200, PT1000, NI100, NI120, NI1000, NI1000Tk5000. The function block `FB_HVACTemperatureSensorEx` is tailored to the Bus Terminals KL3201/02/04 and KL3228. These Bus Terminals can either be ordered preconfigured or set to the corresponding sensor types by software. The raw temperature value is transferred to the function block in 1/10 °C via the input variable `iRawValue` and output as a floating point number via `rPresentValue`. `iRawValue` can, for example, be linked directly to the raw temperature value of the following Bus Terminals: KL3201/02/04 and KL3228.

The output value `rPresentValue` depends on the following smoothing function:

$:= ((/ 10 +) - rPresentValueOld) / + rPresentValueOld; rPresentValue iRawValue rOffset rSmoothFactor$

`rPresentValueOld` is the value of `rPresentValue` that was output in the previous PLC cycle. If `bEnable` goes TRUE, then `rPresentValue` = `rPresentValueOld` for one PLC cycle. If `bErrorSensor` = TRUE, the error has been corrected and `bErrorSensor` = FALSE, then `rPresentValue` = `rPresentValueOld` for one PLC cycle.

The status of the connected temperature sensor is monitored via the input variable `byStatusRawValue` and returned to the controller via the variable `bErrorSensor` in the event of an error. `byStatusRawValue` can, for example, be linked directly to the status byte of the following Bus Terminals: KL3201/02/04 and KL3228. `rHighLimit/rLowLimit` can be used to specify temperature limit values.

Unlike `FB_HVACTemperatureSensor` [▶ 89], this function block has the input variable `bEnable`, which is useful when the sensor characteristic curves in the Bus Terminals KL3201/02/04 and KL3228 are to be adjusted from the PLC via the function block `FB_HVACConfigureKL32xx` [▶ 74]. In this function block the second order filter in `FB_HVACTemperatureSensor` [▶ 89] is replaced by the smoothing function described above. The output `bErrorSensor` is new and replaces the two outputs `bErrorShortCircuit/bErrorBrokenSensor`. These outputs continue to be available in the error byte `byError`.

Application example


Download	Required library
TcHVAC.pro [▶ 531]	TcHVAC.lib

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
iRawValue         : INT;
byStatusRawValue : BYTE;
bEnableLimitCtrl : BOOL;
bReset            : BOOL;
```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled if `bEnable = TRUE`. If `bEnable = FALSE`, the value of `rReplacementValue` is output at the `rPresentValue` output. All error messages and `bInvalidParameter` are set to FALSE. If `bEnable` goes TRUE, then `rPresentValue = rPresentValueOld` for one PLC cycle.

iRawValue: raw value of the temperature sensor in 1/10 °C from the Bus Terminal.

byStatusRawValue: status byte of the temperature sensor from the Bus Terminal. Serves for error diagnosis, e.g. wire break or short circuit. If the KL32xx is set to resistance measurement (Ω) there is no error diagnosis.

bEnableLimitCtrl: enables `rHighLimit` and `rLowLimit` for limit monitoring

bReset: acknowledgement input in the event of an error with rising edge from `bReset`. Depending on operation mode `eTemperatureSensorMode`, errors are acknowledged either with `bReset` or automatically.

VAR_OUTPUT

```
rPresentValue      : REAL;
bErrorGeneral      : BOOL;
byError            : BYTE;
bErrorSensor       : BOOL;
bErrorHighLimit    : BOOL;
bErrorLowLimit     : BOOL;
bInvalidParameter  : BOOL;
```

rPresentValue: temperature output variable with one decimal place.

The value for `rPresentValue` is calculated and output according to the following formula:

$$rPresentValue := ((iRawValue / 10 + rOffset) - rPresentValueOld) / rSmoothFactor + rPresentValueOld;$$

`rPresentValueOld` is the value of `rPresentValue` that was output in the previous PLC cycle. If `bEnable` goes TRUE, then `rPresentValue = rPresentValueOld` for one PLC cycle. If `bErrorSensor = TRUE`, the error is rectified and `bErrorSensor = FALSE`, then for one PLC cycle `rPresentValue = rPresentValueOld`.

If `bErrorSensor = TRUE`, then the value of `rPresentValue` depends on the operation mode `eTemperatureSensorMode`.

bErrorGeneral: the error message *bErrorGeneral* becomes TRUE as soon as one of the error messages *bErrorHighLimit*, *bErrorLowLimit* or *bErrorSensor* = TRUE. The value of the output variable *rPresentValue* is then dependent on the operation mode *eTemperatureSensorMode* and is enabled once the error has been rectified and, depending on the operation mode *eTemperatureSensorMode*, acknowledged with *bReset*.

byError: returns all error messages and warnings,

byError.1 := *bInvalidParameter*

byError.2 := *bErrorGeneral*

byError.3 := *bErrorHighLimit*

byError.4 := *bErrorLowLimit*

byError.5 := *bErrorShortCircuit*

byError.6 := *bErrorBrokenSensor*

byError.7 := *bErrorSensor*

byError.5 := *bErrorShortCircuit*: error, short circuit at the connected temperature sensor. After the error has been rectified, the message is either acknowledged with *bReset* or automatically, depending on the operation mode *eTemperatureSensorMode*.

byError.6 := *bErrorBrokenSensor*: error, wire break at the connected temperature sensor. Once the fault has been corrected, the message is acknowledged either with *bReset* or automatically, depending on mode *eTemperatureSensorMode*.

bErrorSensor: becomes TRUE, if *byError.5/bErrorShortCircuit* or *byError.6/bErrorBrokenSensor* = TRUE. Once the fault has been corrected, the message is acknowledged either with *bReset* or automatically, depending on mode *eTemperatureSensorMode*. If *bErrorSensor* = TRUE, the error has been corrected and *bErrorSensor* = FALSE, then *rPresentValue* = *rPresentValueOld* for one PLC cycle.

bErrorHighLimit: warning upper limit value exceeded; becomes TRUE if *rPresentValue* >= *rHighLimit*. The warning that the upper limit value has been exceeded can only be acknowledged if *rPresentValue* <= *rHighLimit* - 1.0 for a time duration of 5 seconds. Depending on mode *eTemperatureSensorMode*, the warning is acknowledged either with *bReset* or automatically.

bErrorLowLimit: warning lower limit value undershot; becomes TRUE if *rPresentValue* <= *rLowLimit*. The warning that the lower limit value has been undershot can only be acknowledged if *rPresentValue* >= *rLowLimit* + 1.0 for a time duration of 5 seconds. Depending on mode *eTemperatureSensorMode*, the warning is acknowledged either with *bReset* or automatically.

bInvalidParameter: indicates that an incorrect parameter is present at one of the variables *rHighLimit*, *rLowLimit*, *rSmoothFactor* or *eTemperatureSensorMode*. An incorrect parameter specification does not lead to a standstill of the function block; see description of variables. Once the incorrect parameter specification has been corrected, the message *bInvalidParameter* is acknowledged either with *bReset* or automatically, depending on mode *eTemperatureSensorMode*.

VAR_IN_OUT

```
rOffset           : REAL;
rHighLimit        : REAL;
rLowLimit         : REAL;
rReplacementValue : REAL;
rSmoothFactor     : REAL;
eTemperatureSensorMode: E_HVACTemperatureSensorMode;
```

rOffset: temperature compensation in Kelvin, $rPresentValue = (iRawValue / 10.0) + rOffset$. The variable is saved persistently. Preset to 0.

rHighLimit: upper limit value. If *rPresentValue* >= *rHighLimit*, then the output *bErrorHighLimit* is set.

rHighLimit must be greater than *rLowLimit*. The variable is saved persistently. Preset to 120.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *bInvalidParameter* is set if the parameter is incorrect.

rLowLimit: lower limit. The output *bErrorLowLimit* is set if *rPresentValue* <= *rLowLimit*. The variable is saved persistently. Preset to -60.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *bInvalidParameter* is set if the parameter is incorrect.

rReplacementValue: replacement value that is output at *rPresentValue* in the case of the errors *bErrorShortCircuit* and *bErrorBrokenSensor*, if the selected operation mode *eTemperatureSensorMode* = *eHVACTemperatureSensorMode_ReplacementValue* or *eTemperatureSensorMode* = *eHVACTemperatureSensorMode_AutoResetReplacementValue*. The variable is saved persistently. Preset to 0.

rSmoothFactor: smoothing factor (>=1) for the output value *rPresentValue*. The variable is saved persistently. Preset to 100.

According to the following formula the value for *rPresentValue* is calculated and output:

$$rPresentValue := ((iRawValue / 10 + rOffset) - rPresentValueOld) / rSmoothFactor + rPresentValueOld;$$

rPresentValueOld is the value of *rPresentValue*, which was output one PLC cycle before.

If *rSmoothFactor* = 1, then *rPresentValue* := ((*iRawValue* / 10 + *rOffset*)

If there is an incorrect variable value at *rSmoothFactor*, then the last valid variable value is used, if available.

If there is no valid last value, then the default value is used. *bInvalidParameter* is set if the parameter is incorrect.

eTemperatureSensorMode: Enum, via which the operation mode of the function block is specified.

eTemperatureSensorMode = *eHVACTemperatureSensorMode_ReplacementValue*: if *bErrorSensor* =

TRUE, then *rPresentValue* = *rReplacementValue*. After the error has been corrected, the function block must be acknowledged by a rising edge at the input variable *bReset*.

eTemperatureSensorMode = *eHVACTemperatureSensorMode_LastValue*: if *bErrorSensor* = TRUE, then the last valid temperature value that was present 10 seconds before is output at the output variable *rPresentValue*. After the error has been corrected, it must be acknowledged by a rising edge at the input variable *bReset*.

eTemperatureSensorMode = *eHVACTemperatureSensorMode_AutoResetReplacementValue*: if *bErrorShortCircuit* or *bErrorBrokenSensor* = TRUE, then *rPresentValue* = *rReplacementValue*. After the error has been corrected, the function block acknowledges itself automatically.

eTemperatureSensorMode = *eHVACTemperatureSensorMode_AutoResetLastValue*: if *bErrorSensor* = TRUE, then the last valid temperature value that was present 10 seconds before is output at the output variable *rPresentValue*. After the error has been corrected, the function block acknowledges itself automatically.

If there is an incorrect variable value at *eTemperatureSensorMode*, the default value is used.

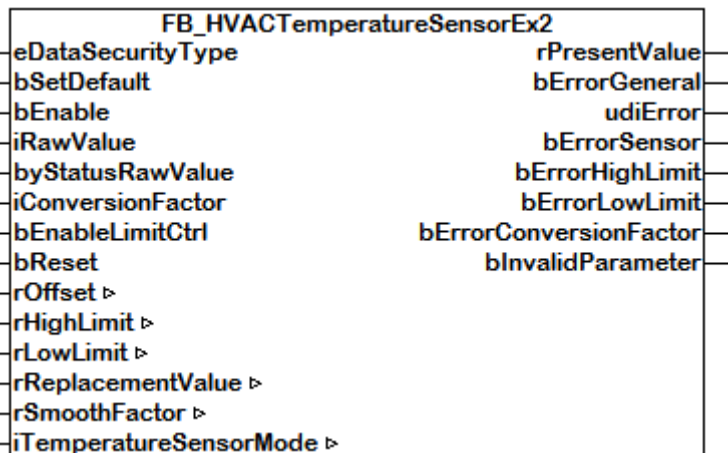
bInvalidParameter is set if an incorrect parameter is specified.

The variable is saved persistently. Preset to 3%.

Documents about this

- example_persistent_e.zip (Resources/zip/11659714827.zip)

3.2.11 FB_HVACTemperatureSensorEx



Application

This function block is used for the acquisition and subsequent processing of temperature values, e.g. for the sensor types PT100, PT200, PT1000, NI100, NI120, NI1000, NI1000Tk5000. The function block *FB_HVACTemperatureSensorEx2* is tailored to the Bus Terminals KL3201/02/04, KL3222, KL3228 and KL3208-0010. These Bus Terminals can either be ordered preconfigured or set to the corresponding sensor types by software.

The raw temperature value is transferred to the function block in 1/10 °C or 1/100 °C via the input variable *iRawValue* and output as a floating point number via *rPresentValue*. *iRawValue* can, for example, be linked directly to the raw temperature value of the following Bus Terminals: KL3201/02/04, KL3222, KL3228 and KL3208-0010.

The output value *rPresentValue* depends on one of the following two smoothing functions:

= 0: := ((/ *iConversionFactor* *rPresentValue* *iRawValue* 10 +) - *rPresentValueOld*) / + *rPresentValueOld*;
rOffset *rSmoothFactor*

= 1: := ((/ *iConversionFactor* *rPresentValue* *iRawValue* 100 +) - *rPresentValueOld*) / + *rPresentValueOld*;
rOffset *rSmoothFactor*

rPresentValueOld is the value of *rPresentValue* that was output in the previous PLC cycle. If *bEnable* goes TRUE, then *rPresentValue* = *rPresentValueOld* for one PLC cycle. If *bErrorSensor* = TRUE, the error has been corrected and *bErrorSensor* = FALSE, then *rPresentValue* = *rPresentValueOld* for one PLC cycle.

The status of the connected temperature sensor is monitored via the input variable *byStatusRawValue* and returned to the controller via the variable *bErrorSensor* in the event of an error. *byStatusRawValue* can, for example, be linked directly to the status byte of the following Bus Terminals: KL3201/02/04, KL3222, KL3228 and KL3208-0010.

rHighLimit/rLowLimit can be used to specify temperature limit values.

Unlike [FB_HVACTemperatureSensor](#) [► 89], this function block has the input variable *bEnable*, which is useful when the sensor characteristic curves in the Bus Terminals KL3201/02/04, KL3222, KL3228 and KL3208-0010 are to be adjusted from the PLC via the function block [FB_HVACConfigureKL32xx](#) [► 74]. In this function block the second order filter in [FB_HVACTemperatureSensor](#) [► 89] is replaced by the smoothing function described above. The output *bErrorSensor* is new and replaces the two outputs *bErrorShortCircuit*/*bErrorBrokenSensor*. These outputs continue to be available in the error byte *udiError*.

Unlike [FB_HVACTemperatureSensorEx](#) [► 92], sensors with the raw temperature value 1/10 or 1/100 °C can be transferred on this function block.

Application example


Download	Required library
TcHVAC.pro [► 531]	TcHVAC.lib

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
iRawValue         : INT;
byStatusRawValue : BYTE;
iConversionFactor : INT;           0..1
bEnableLimitCtrl : BOOL;
bReset            : BOOL;
```

eDataSecurityType: if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block [FB_HVACPersistentDataHandling](#) must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled if `bEnable = TRUE`. If `bEnable = FALSE`, the value of `rReplacementValue` is output at the `rPresentValue` output. All error messages and `bInvalidParameter` are set to FALSE. If `bEnable` goes TRUE, then `rPresentValue = rPresentValueOld` for one PLC cycle.

iRawValue: raw value of the temperature sensor either in 1/10 or 1/100 °C from the Bus Terminal.

byStatusRawValue: status byte of the temperature sensor from the Bus Terminal. Serves for error diagnosis, e.g. wire break or short circuit. If the KL32xx is set to resistance measurement (Ω) there is no error diagnosis.

iConversionFactor: conversion factor for the output value `rPresentValue`.

`iConversionFactor = 0: rPresentValue := ((iRawValue / 10 + rOffset) - rPresentValueOld) / rSmoothFactor + rPresentValueOld;`

`iConversionFactor = 1: rPresentValue := ((iRawValue / 100 + rOffset) - rPresentValueOld) / rSmoothFactor + rPresentValueOld;`

If a value outside its range of 0 to 1 is specified at `iConversionFactor`, this is signaled by `bErrorConversionFactor = TRUE`.

bEnableLimitCtrl: enables `rHighLimit` and `rLowLimit` for limit monitoring

bReset: acknowledgement input in the event of an error with rising edge from `bReset`. Depending on mode `iTemperatureSensorMode`, errors are acknowledged either with `bReset` or automatically.

VAR_OUTPUT

```
rPresentValue      : REAL;
bErrorGeneral      : BOOL;
udiError           : UDINT;
bErrorSensor       : BOOL;
bErrorHighLimit    : BOOL;
bErrorLowLimit     : BOOL;
bErrorConversionFactor: BOOL;
bInvalidParameter  : BOOL;
```

rPresentValue: temperature output variable with two decimal places.

The value for `rPresentValue` is calculated and output according to the following formula:

`iConversionFactor = 0: rPresentValue := ((iRawValue / 10 + rOffset) - rPresentValueOld) / rSmoothFactor + rPresentValueOld;`

`iConversionFactor = 1: rPresentValue := ((iRawValue / 100 + rOffset) - rPresentValueOld) / rSmoothFactor + rPresentValueOld;`

`rPresentValueOld` is the value of `rPresentValue` that was output in the previous PLC cycle. If `bEnable` goes TRUE, then `rPresentValue = rPresentValueOld` for one PLC cycle. If `bErrorSensor = TRUE`, the error is corrected and `bErrorSensor = FALSE`, then for one PLC cycle `rPresentValue = rPresentValueOld`. If `bErrorSensor = TRUE`, then the value of `rPresentValue` depends on the operation mode `iTemperatureSensorMode`.

bErrorGeneral: the error message *bErrorGeneral* becomes TRUE as soon as one of the error messages *bErrorHighLimit*, *bErrorLowLimit* or *bErrorSensor* = TRUE. The value of the output variable *rPresentValue* is then dependent on the operation mode *iTemperatureSensorMode* and is enabled once the error has been rectified and, depending on the operation mode *iTemperatureSensorMode*, acknowledged with *bReset*.

udiError: returns all error messages and warnings,

```

udiError.1 := bInvalidParameter
udiError.2 := bErrorGeneral
udiError.3 := bErrorHighLimit
udiError.4 := bErrorLowLimit
udiError.5 := bErrorShortCircuit
udiError.6 := bErrorBrokenSensor
udiError.7 := bErrorSensor
udiError.8 := bErrorConversionFactor

```

byError.5 := bErrorShortCircuit: error, short circuit at the connected temperature sensor. After the error has been corrected, the message is either acknowledged with *bReset* or automatically, depending on the operation mode *iTemperatureSensorMode*. *byError.6 := bErrorBrokenSensor:* error, wire break at the connected temperature sensor. Once the fault has been corrected, the message is acknowledged either with *bReset* or automatically, depending on mode *iTemperatureSensorMode*.

bErrorSensor: becomes TRUE, if *byError.5 / bErrorShortCircuit* or *byError.6 / bErrorBrokenSensor* = TRUE. Once the fault has been corrected, the message is acknowledged either with *bReset* or automatically, depending on mode *iTemperatureSensorMode*. If *bErrorSensor* = TRUE, the error has been corrected and *bErrorSensor* = FALSE, then *rPresentValue* = *rPresentValueOld* for one PLC cycle.

bErrorHighLimit: warning upper limit value exceeded; becomes TRUE if *rPresentValue* >= *rHighLimit*. The warning that the upper limit value has been exceeded can only be acknowledged if *rPresentValue* <= *rHighLimit* - 1.0 for a time duration of 5 seconds. Depending on mode *iTemperatureSensorMode*, the warning is acknowledged either with *bReset* or automatically.

bErrorLowLimit: warning lower limit value undershot; becomes TRUE if *rPresentValue* <= *rLowLimit*. The warning that the lower limit value has been undershot can only be acknowledged if *rPresentValue* >= *rLowLimit* + 1.0 for a time duration of 5 seconds. Depending on mode *iTemperatureSensorMode*, the warning is acknowledged either with *bReset* or automatically.

bErrorConversionFactor: if a value outside its range of 0 to 1 is specified at *iConversionFactor*, this is signaled by *bErrorConversionFactor* = TRUE. The message need not be acknowledged after rectifying the cause.

bInvalidParameter: indicates that an incorrect parameter is present at one of the variables *rHighLimit*, *rLowLimit*, *rSmoothFactor*, *iConversionFactor* or *iTemperatureSensorMode*. An incorrect parameter specification does not lead to a standstill of the function block; see description of variables. Once the incorrect parameter specification has been corrected, the message *bInvalidParameter* is acknowledged either with *bReset* or automatically, depending on mode *iTemperatureSensorMode*.

VAR_IN_OUT

```

rOffset           : REAL;
rHighLimit        : REAL;
rLowLimit         : REAL;
rReplacementValue : REAL;
rSmoothFactor     : REAL;
iTemperatureSensorMode: INT;

```

rOffset: temperature compensation in Kelvin. The variable is saved persistently. Preset to 0.

rHighLimit: upper limit value. The variable is saved persistently. Preset to 120.

If *rPresentValue* >= *rHighLimit*, then the output *bErrorHighLimit* is set. *rHighLimit* must be greater than *rLowLimit*.

If an incorrect variable value is present, then the last valid variable value is used, if available. If there is no valid last value, then the default value is used. *bInvalidParameter* is set if the parameter is incorrect.

rLowLimit: lower limit. The variable is saved persistently. Preset to -60.

If *rPresentValue* <= *rLowLimit*, then the output *bErrorLowLimit* is set.

If an incorrect variable value is present, then the last valid variable value is taken, if available. If there is no valid last value, then the default value is used. *bInvalidParameter* is set if the parameter is incorrect.

rReplacementValue: replacement value that is output at *rPresentValue* in the case of the errors *bErrorShortCircuit* and *bErrorBrokenSensor*, if the selected operation mode *iTemperatureSensorMode* = 0 or *iTemperatureSensorMode* = 2.

The variable is saved persistently. Preset to 0.

rSmoothFactor: smoothing factor (>=1) for the output value *rPresentValue*.

The variable is saved persistently. Preset to 100.

The value for *rPresentValue* is calculated and output according to the following formula:

iConversionFactor = 0: $rPresentValue := ((iRawValue / 10 + rOffset) - rPresentValueOld) / rSmoothFactor + rPresentValueOld;$

iConversionFactor = 1: $rPresentValue := ((iRawValue / 100 + rOffset) - rPresentValueOld) / rSmoothFactor + rPresentValueOld;$

rPresentValueOld is the value of *rPresentValue* that was output in the previous PLC cycle.

If an incorrect variable value is present at *rSmoothFactor*, then the last valid variable value is taken, if available. If there is no valid last value, then the default value is used. *bInvalidParameter* is set if the parameter is incorrect.

iTemperatureSensorMode: specifies the operation mode of the function block.

iTemperatureSensorMode = 0: if *bErrorSensor* = TRUE, then *rPresentValue* = *rReplacementValue*. After the error has been corrected, the function block must be acknowledged by a rising edge at the input variable *bReset*.

iTemperatureSensorMode = 1: if *bErrorSensor* = TRUE, then the last valid temperature value that was present 10 seconds before is output at the output variable *rPresentValue*. After the error has been corrected, it must be acknowledged by a rising edge at the input variable *bReset*.

iTemperatureSensorMode = 2: if *bErrorShortCircuit* or *bErrorBrokenSensor* = TRUE, then *rPresentValue* = *rReplacementValue*. After the error has been corrected, the function block acknowledges itself automatically.

iTemperatureSensorMode = 3: if *bErrorSensor* = TRUE, then the last valid temperature value that was present 10 seconds before is output at the output variable *rPresentValue*. After the error has been corrected, the function block acknowledges itself automatically.

If there is an incorrect variable value at *iTemperatureSensorMode*, the default value is used.

bInvalidParameter is set if the parameter is incorrect.

The variable is saved persistently. Preset to 3%.

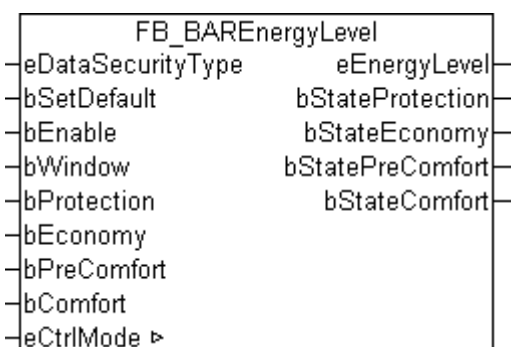
Documents about this

 example_persistent_e.zip (Resources/zip/11659714827.zip)

3.3 HVAC Room Functions

3.3.1 Air conditioning

3.3.1.1 FB_BAREnergyLevel



Application

This function block is for the adaptation of the supply of energy for the use of the building. The type of room utilization is set by the BMS. The longer a building or room is not used, the further the energy level can be lowered. The energy level currently selected by the function block is transferred to the room temperature controller.

Protection:

This operation mode is activated in the case of long absence times e.g. during works holidays or also when a window is open. The energy level is very low and serves only to protect the building from damage caused by frost or overheating.

Economy:

The Economy energy level is used for the economy mode. Economy mode is activated, for example, at night by a timer switch schedule.

PreComfort:

The PreComfort energy level is for an unused room which, however, can be occupied again shortly. The standby mode is frequently activated by a timer schedule.

Comfort:

If the room is occupied, it is in Comfort mode. Comfort mode can be activated by a timer switch schedule or by presence recognition.

VAR_INPUT

```
eDataSecurityType      : E_HVACDataSecurityType;
bSetDefault            : BOOL;
bEnable                : BOOL;
bWindow                : BOOL;
bProtection            : BOOL;
bEconomy               : BOOL;
bPreComfort            : BOOL;
bComfort               : BOOL;
```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: The function block is activated by a TRUE at this input.

bWindow: The window contact is connected to this input. TRUE means that the window is OPEN. FALSE means that the window is CLOSED.

bProtection: Protection mode is activated with the input bProtection. Protection mode is active if the input is TRUE.

bEconomy: economy mode is activated with the input bEconomy. Economy mode is active if the input is TRUE.

bPreComfort: The Pre-comfort level is activated with this input. The Pre-comfort level is active if the input is TRUE.

bComfort: The Comfort level is activated with this input if the room is occupied.

VAR_OUTPUT

```
eEnergyLevel      : E_BAREnergyLevel;
bStateProtection  : BOOL;
bStateEconomy     : BOOL;
bStatePreComfort  : BOOL;
bStateComfort     : BOOL;
```

eEnergyLevel: this output contains the current energy level.

bStateProtection: the state of the *bProtection* input is relayed to the outside in the operation modes *eBAREnergyLevel_AUTO_I* and *eBAREnergyLevel_AUTO_II*.

bStateEconomy: the state of the *bEconomy* input is relayed to the outside in the operation modes *eBAREnergyLevel_AUTO_I* and *eBAREnergyLevel_AUTO_II*.

bStatePreComfort: The state of the *bPreComfort* input is relayed to the outside in the operation modes *eBAREnergyLevel_AUTO_I* and *eBAREnergyLevel_AUTO_II*.

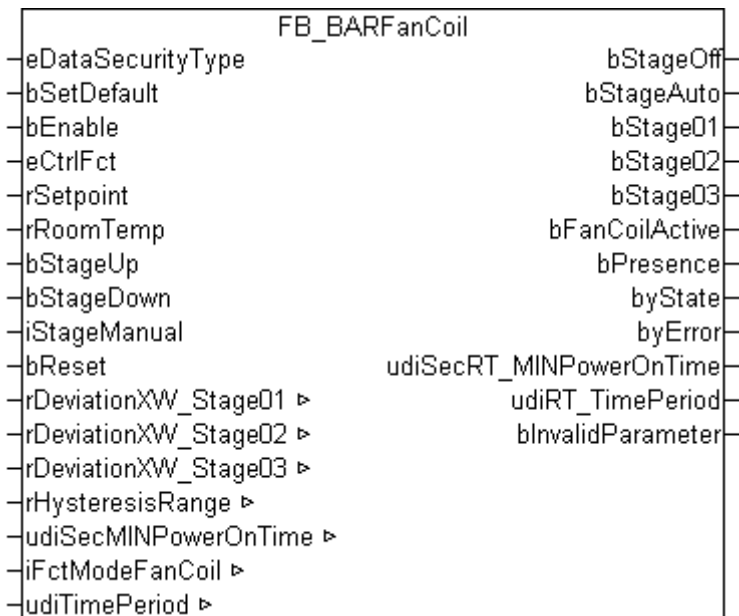
bStateComfort: the state of the *bComfort* input is relayed to the outside in the operation modes *eBAREnergyLevel_AUTO_I* and *eBAREnergyLevel_AUTO_II*.

VAR_IN_OUT

```
eCtrlMode      : E_BAREnergyLevel;
```

eCtrlMode: Using this ENUM the operation mode can be preselected from the building management level. The variable is saved persistently. Preset to automatic.

3.3.1.2 FB_BARFanCoil



Application

The function block maps a 3-speed fan with the corresponding switching hysteresis, which is the same for all three speeds. The speed is set stepwise via the control deviation of the actual room temperature value from the set room temperature value. Furthermore there is a possibility to manually override the fan controller via the *iStageManual* or *bStageUp* or *bStageDown* input. A minimum switch-on time can be set via the *udiSecMINPowerOnTime* input, which is then valid for each stage.

VAR_INPUT

```

eDataSecurityType      : E_HVACDataSecurityType;
bSetDefault            : BOOL;
bEnable                : BOOL;
eCtrlFct               : E_BARCtrlFct;
rSetpoint              : REAL;
rRoomTemp              : REAL;
bStageUp               : BOOL;
bStageDown             : BOOL;
iStageManual           : INT;
bReset                 : BOOL;

```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: The function block is activated if the variable `bEnable` is TRUE. No fan stage is activated with a FALSE.

eCtrlFct: this input is connected to the `eCtrlFct` output of the **FB_BARFctSelection**. This information is important in order to know whether the plant is in heating or cooling mode. In automatic mode the fan stages are only activated, for example, if according to the control deviation the requirement for heating is active and the system is in heating mode, or if according to the control deviation the requirement for cooling is active and the system is in cooling mode.

rSetpoint: input for the set temperature.

rRoomTemp: input for the room temperature.

bStageUp: Local adjustment of the fan stage, stepwise increase by push button.

bStageDown: Local adjustment of the fan stage, stepwise decrease by push button.

iStageManual: The manual fan stage can/is set centrally via this input.

iStageManual: = 0 corresponds to fan stage OFF

iStageManual: = 1 corresponds to fan stage AUTO

iStageManual: = 2 corresponds to fan stage01 active

iStageManual: = 3, corresponds to fan stage02 active

iStageManual: = 4, corresponds to fan stage03 active

bReset: Acknowledge input in case of a fault or an incorrect parameter.

VAR_OUTPUT

```

bStageOff              : BOOL;
bStageAuto             : BOOL;
bStage01               : BOOL;
bStage02               : BOOL;
bStage03               : BOOL;
bFanCoilActive        : BOOL;
bPresence              : BOOL;
byState                : BYTE;
byError                : BYTE;

```

```

udiSecRT_MINPowerOnTime: UDINT;
udiRT_TimePeriod       : UDINT;
bInvalidParameter      : BOOL;

```

bStageOff: TRUE, fan stages are switched off.

bStageAuto: TRUE, fan controller is in automatic mode.

bStage01: TRUE, fan stage01 active.

bStage02: TRUE, fan stage02 active.

bStage03: TRUE, fan stage03 active.

bFanCoilActive: TRUE if one of the three fan stages is active. This output can be/is used to enable controllers in order to avoid a build up of heat or cold.

bPresence: TRUE means that presence was detected via the *bStageUp*, *bStageDown* or *iStageManual* inputs.

byState: indicates the state of the fan controller.

byState.0:= function block is activated

byState.3:= manual fan stage setting is active

byState.4:= bReset

byState.5:= fan stage01 active

byState.6:= fan stage02 active

byState.7:= fan stage03 active

byError: output of the errors as byte.

byError.1:= bInvalidParameter

udiSecRT_MINPowerOnTime: Indicates the remaining time of the minimum switch-on time.

udiRT_TimePeriod: Indicates the remaining time of the manual override.

bInvalidParameter: Indicates that an incorrect input parameter is present. *bInvalidParameter* must be acknowledged with *bReset*.

VAR_IN_OUT

```

rDeviationXW_Stage01 : REAL;
rDeviationXW_Stage02 : REAL;
rDeviationXW_Stage03 : REAL;
rHysteresisRange      : REAL;
udiSecMINPowerOnTime : UDINT;
iFctModeFanCoil       : INT;
udiTimePeriod         : UDINT;

```

rDeviationXW_Stage01: limit value of the control deviation for fan stage01. The variable is saved persistently. Preset to 0.7.

rDeviationXW_Stage02: limit value of the control deviation for fan stage02. The variable is saved persistently. Preset to 1.7.

rDeviationXW_Stage03: limit value of the control deviation for fan stage03. The variable is saved persistently. Preset to 2.1.

rHysteresisRange: hysteresis range that is placed around the limit value.

Example: a limit value of 0.7 and a hysteresis range of 0.2 results in the fan stage01 being switched on at a control deviation > 0.8.

And at a control deviation < 0.6 the fan stage01 is switched off.

The variable is saved persistently. Preset to 0.2.

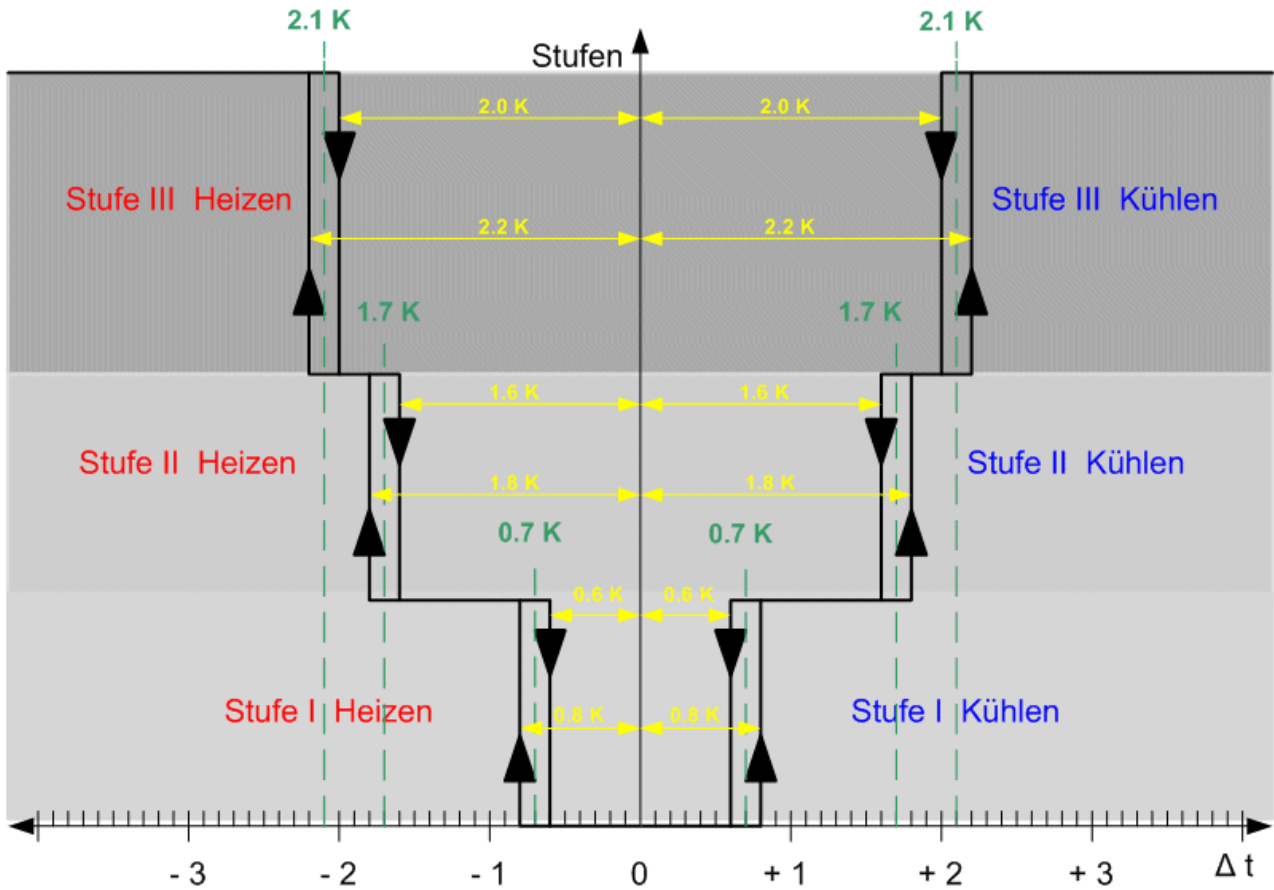
udiSecMINPowerOnTime: Minimum switch-on time that a fan must run for in a stage before switching to a different stage or switching off. Input in seconds (e.g. 120 corresponds to 120 s). The variable is saved persistently. Preset to 120 s.

iFctModeFanCoil: The user has the possibility to activate the fan controller for heating mode or cooling mode or both modes via the valence of this variable. Valid values are 1, 2 or 3. Other values are invalid and *bInvalidParameter* is set to TRUE. The variable is saved persistently. Preset to 3%.

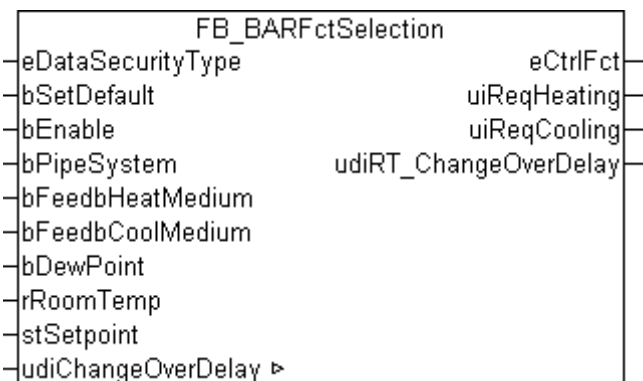
Cooling	Heating	Valence
0	1	1 (= fan controller active in heating mode)
1	0	2 (= fan controller active in cooling mode)
1	1	3 (= fan controller active in heating mode and cooling mode)

udiTimePeriod: Timeframe during which the manual override is active in case of presence. Specified in minutes

Fig. 1: representation of the fan control with the default parameters



3.3.1.3 FB_BARFctSelection

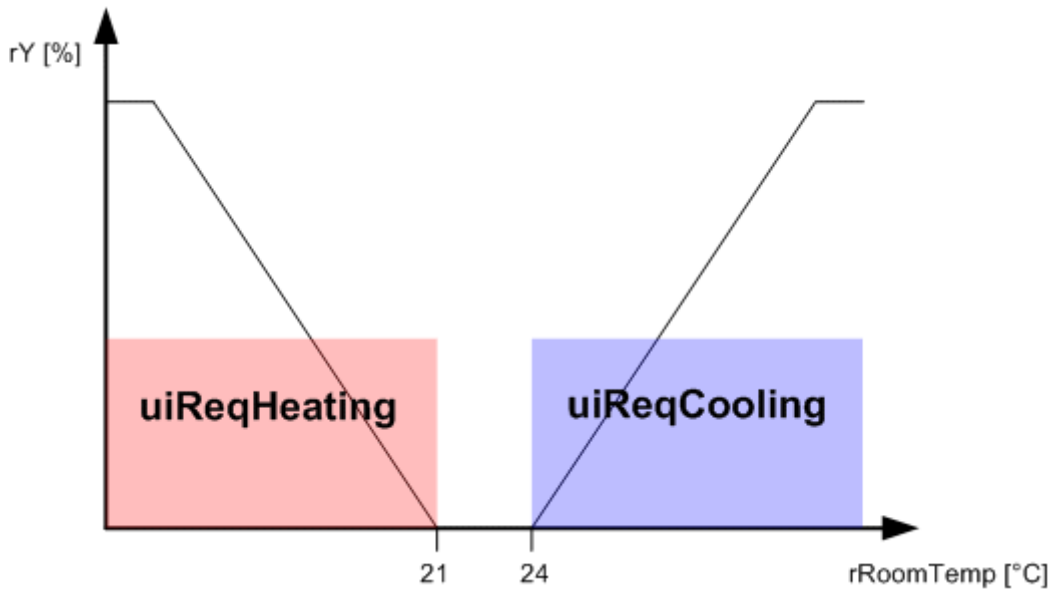


Application

This function block is for enabling room heating or room cooling. It can be used for 2-pipe systems (changeover) or 4-pipe systems.

In the case of a 4-pipe system the changeover from heating to cooling operation takes place automatically on the basis of a comparison of the setpoint for the room temperature with the actual value for the room temperature.

Sketch:



In the case of a 2-pipe system the heating operation or cooling operation may only be enabled when heating or cooling medium is present. The room temperature controller gets this information from the primary units.

In both 2-pipe and 4-pipe systems the changeover between heating operation and cooling operation can be delayed by a timer. The input variable *udiChangeOverDelay* must be greater than zero for this.

The following tables describe the interrelationship between the inputs and the eCtrlFct output of the FB_BARFctSelection function block.

in the 2-pipe system

bEnable	bPipeSystem	bFeedbHeat Medium	bFeedbCool Medium	interim result	bDewPoint	eCtrlFct
0	0	0	0	OFF	TRUE / FALSE	OFF
1	0	0	0	Heating	TRUE / FALSE	Heating
1	0	0	1	Cooling	TRUE / FALSE	OFF / Cooling
1	0	1	0	Heating	TRUE / FALSE	Heating
1	0	1	1	Heating	TRUE / FALSE	Heating

in the 4-pipe system

bEnable	bPipeSystem	T_Room <= Tsetpoint	T_Room > Tsetpoint	interim result	bDewPoint	eCtrlFct
0	1	0	0	OFF	TRUE / FALSE	OFF
1	1	0	1	Cooling	TRUE / FALSE	OFF / Cooling

1	1	1	0	Heating	TRUE / FALSE	Heating
1	1	1	1	Heating	TRUE / FALSE	Heating

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
bPipeSystem       : BOOL;
bFeedbHeatMedium : BOOL;
bFeedbCoolMedium : BOOL;
bDewPoint         : BOOL;
rRoomTemp         : REAL;
stSetpoint        : ST_BARSetpointRoom;
```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: TRUE enables the function block. If FALSE the function block is disabled and `eCtrlFct := eHVACCtrlFct_Off`.

bPipeSystem: FALSE means that a 2-pipe system is present. TRUE means that a 4-pipe system is present.

bFeedbHeatMedium: Signal from the power generation or distribution that heating medium is available.

bFeedbCoolMedium: Signal from the power generation or distribution that cooling medium is available.

bDewPoint: the dew point sensor is connected to this input. If this is triggered the cooling control function is deactivated and `eCtrlFct := eHVACCtrlFct_Off` is set.

rRoomTemp: This input variable transfers the current room temperature to the function block.

stSetpoint: STRUCTURE containing the setpoints for the individual energy levels.

VAR_OUTPUT

```
eCtrlFct          : E_BARCtrlFct;
uiReqHeating      : UINT;
uiReqCooling      : UINT;
udiRT_ChangeOverDelay: UDINT;
```

eCtrlFct: This output contains the current control function.

uiReqHeating: is 1 if the room/zone requests heating energy. It is 0 if there is no heating requirement.

uiReqCooling: is 1 if the room/zone requests cooling energy. It is 0 if there is no cooling requirement.

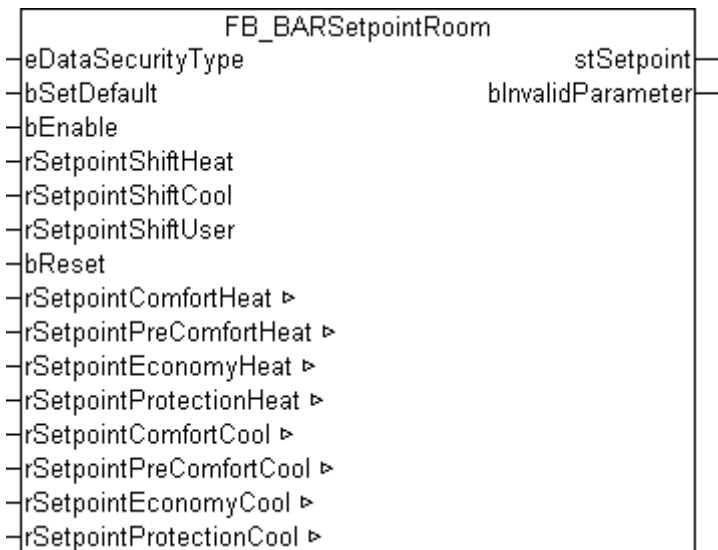
udiRT_ChangeOverDelay: Indicates the time remaining until the active control function is changed over.

VAR_IN_OUT

```
uiChangeOverDelay : UINT;
```

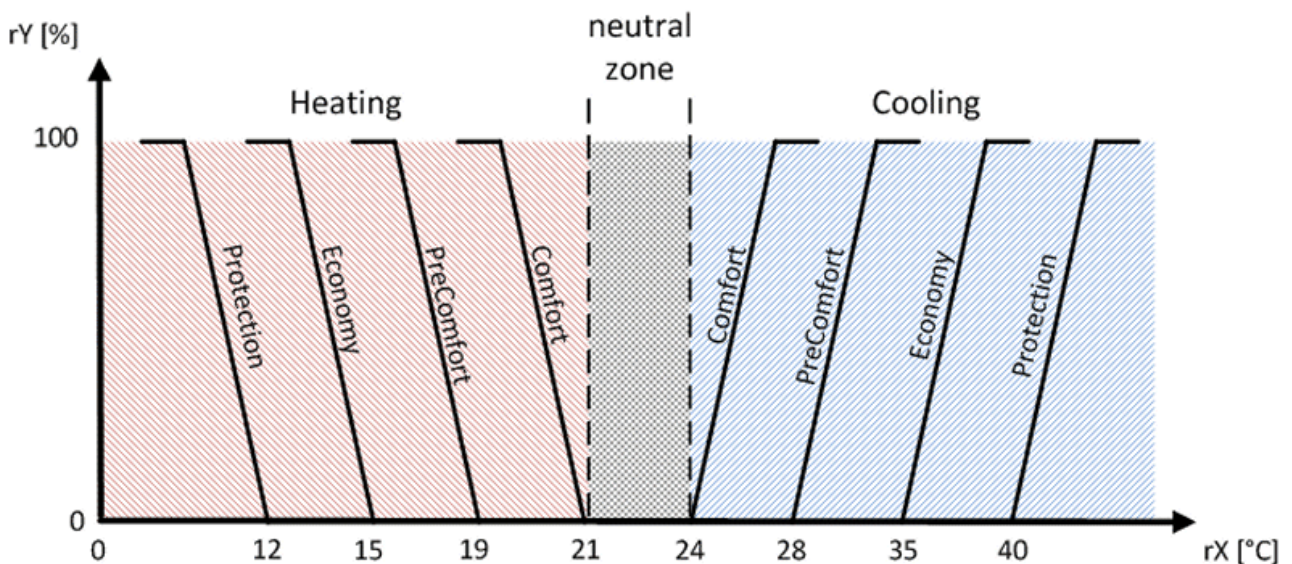
uiChangeOverDelay: changeover time between the control functions. Must be specified in seconds. If the input is greater than 0 it will always be observed. The variable must be 0 if there is to be no changeover time between the control functions. The variable is saved persistently. Preset to 0.

3.3.1.4 FB_BARSetpointRoom



Application

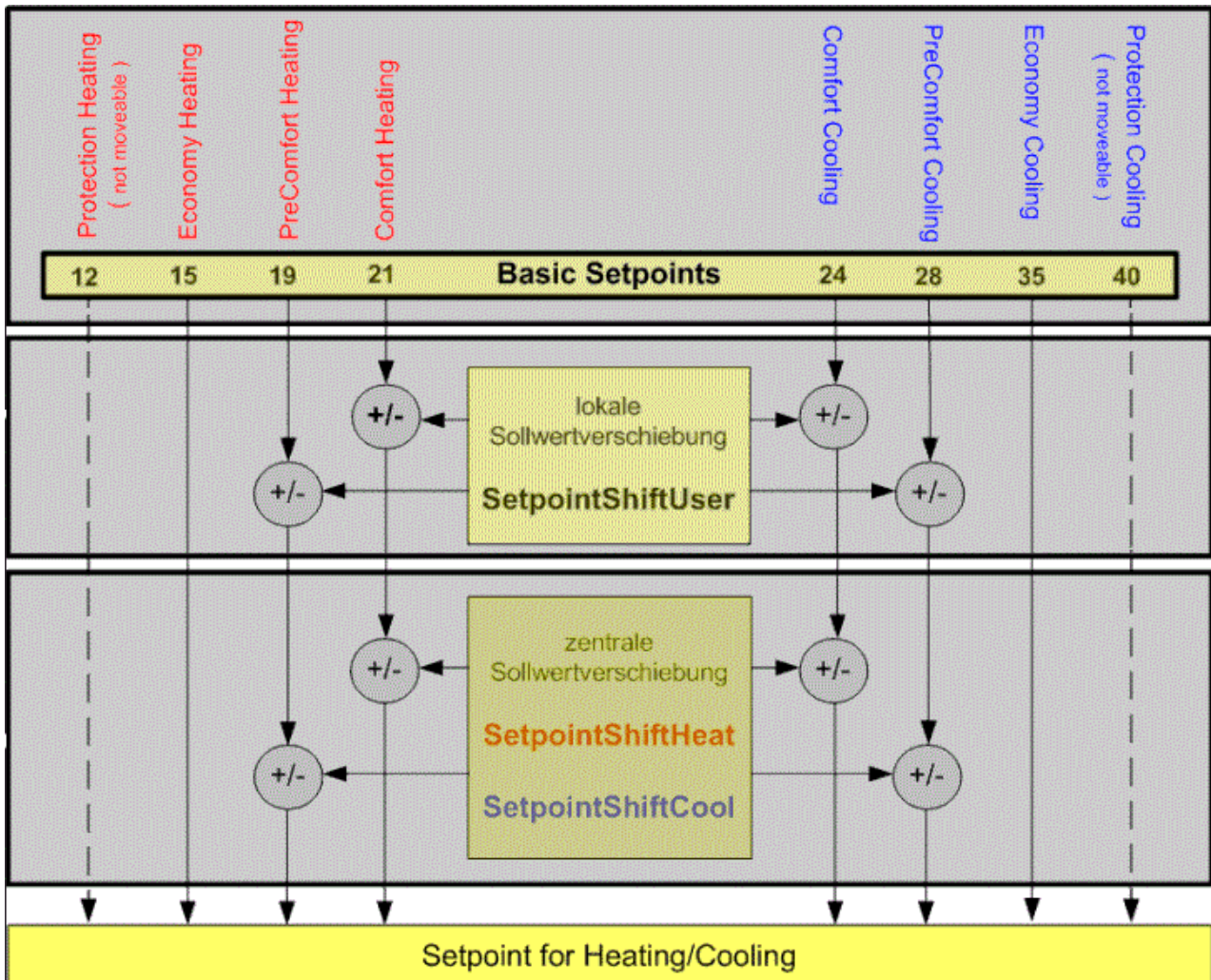
The function block FB_BARSetpointRoom assigns setpoints for cooling and heating operation to each of the energy levels Protection, Economy, PreComfort and Comfort. In connection with the function block FB_BAREnergyLevel the room temperature controllers are assigned an energetically optimum setpoint in accordance with the use of the room and the selected heating or cooling operation.



The resulting setpoint for the different energy levels is made up of:

1. the base setpoint value
2. the local setpoint value shift (not in the case of the Protection setpoints)
3. the central setpoint value shift (not in the case of the Protection setpoints)

The local shift due to a room setpoint generator and also the remote adjustment of the setpoints by a building management system only affect the Comfort and PreComfort energy levels.



VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable          : BOOL;
rSetpointShiftHeat: REAL;
rSetpointShiftCool: REAL;
rSetpointShiftUser: REAL;
bReset           : BOOL;
```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instantiated once in the main program, which is called cyclically. Otherwise the instantiated FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: The function block is activated if the variable `bEnable` is TRUE.

rSetpointShiftHeat: The rSetpointShiftHeat variable is used for the adaptation of the ComfortHeating setpoint of the building management system.

If the ComfortHeating setpoint is raised, then the setpoints for the ComfortCooling and PreComfortCooling are also increased.

Example:

Energy levels	ProtectionHeating	Economy Heating	PreComfortHeating	Comfort Heating	Comfort Cooling	PreComfortCooling	Economy Cooling	ProtectionCooling
Base setpoint value [°C]	12	15	19	21	24	28	35	40
Setpoint ShiftHeat [K]	-	-	-	+3	+3	+3	-	-
Resulting setpoint [°C]	12	15	19	24	27	31	35	40

A lowering of the ComfortHeating setpoint affects only the ComfortHeating and PreComfortHeating setpoints.

Example:

Energy levels	ProtectionHeating	Economy Heating	PreComfortHeating	Comfort Heating	Comfort Cooling	PreComfortCooling	Economy Cooling	ProtectionCooling
Base setpoint value [°C]	12	15	19	21	24	28	35	40
Setpoint ShiftHeat [K]	-	-	-3	-3	-	-	-	-
Resulting setpoint [°C]	12	15	16	18	24	28	35	40

rSetpointShiftCool: The rSetpointShiftCoolvariable is used for the adaptation of the ComfortCooling setpoint of the building management system.

If the ComfortCooling setpoint is raised, then the setpoint for the PreComfortCooling is also raised.

Example:

Energy levels	ProtectionHeating	Economy Heating	PreComfortHeating	Comfort Heating	Comfort Cooling	PreComfortCooling	Economy Cooling	ProtectionCooling
Base setpoint value [°C]	12	15	19	21	24	28	35	40
Setpoint ShiftCool [K]	-	-	-	-	+3	+3	-	-
Resulting setpoint [°C]	12	15	19	21	27	31	35	40

A lowering of the ComfortCooling setpoint only affects the ComfortCooling. The PreComfortCooling is not changed.

Example:

Energy levels	ProtectionHeating	Economy Heating	PreComfortHeating	Comfort Heating	Comfort Cooling	PreComfortCooling	Economy Cooling	ProtectionCooling
Base setpoint value [°C]	12	15	19	21	24	28	35	40
Setpoint ShiftCool [K]	-	-	-	-	-3	-	-	-
Resulting setpoint [°C]	12	15	19	21	21	28	35	40

If the setpoint of the PreComfort energy level is shifted beyond the setpoint of the Economy level, then the setpoint of the Economy level adopts the value of the PreComfort level.

Example:

Energy levels	ProtectionHeating	Economy Heating	PreComfortHeating	Comfort Heating	Comfort Cooling	PreComfortCooling	Economy Cooling	ProtectionCooling
Base setpoint value [°C]	12	15	19	21	24	28	35	40
Setpoint ShiftCool [K]	-	-	-	-	+8	+8	-	-
Resulting setpoint [°C]	12	15	19	21	32	36	36	40

rSetpointShiftUser: The variable rSetpointShiftUser is used for local setpoint shifting of the user. A positive value of rSetpointShiftUser affects the setpoint of ComfortHeating, ComfortCooling and PreComfortCooling.

Example:

Energy levels	ProtectionHeating	Economy Heating	PreComfortHeating	Comfort Heating	Comfort Cooling	PreComfortCooling	Economy Cooling	ProtectionCooling
Base setpoint value [°C]	12	15	19	21	24	28	35	40
Setpoint ShiftUser [K]	-	-	-	+3	+3	+3	-	-
Resulting setpoint [°C]	12	15	19	24	27	31	35	40

A negative value of rSetpointShiftUser affects the setpoints of PreComfortHeating, ComfortHeating and ComfortCooling.

Example:

Energy levels	Protection Heating	Economy Heating	PreComfort Heating	Comfort Heating	Comfort Cooling	PreComfort Cooling	Economy Cooling	Protection Cooling
Base setpoint value [°C]	12	15	19	21	24	28	35	40
Setpoint ShiftUser [K]	-3	-	-3	-3	-3	-	-	-
Resulting setpoint [°C]	12	15	16	18	21	28	35	40

bReset: Acknowledge input in case of a fault or an incorrect parameter.

VAR_OUTPUT

```
stSetpoint : ST_BARSetpointRoom;
bInvalidParameter: BOOL;
```

stSetpoint: Structure containing the setpoints for all energy levels.

bInvalidParameter: Indicates that an incorrect input parameter is present. *bInvalidParameter* must be acknowledged with *bReset*.

VAR_IN_OUT

```
rSetpointComfortHeat : REAL;
rSetpointPreComfortHeat : REAL;
rSetpointEconomyHeat : REAL;
rSetpointProtectionHeat : REAL;
rSetpointComfortCool : REAL;
rSetpointPreComfortCool : REAL;
rSetpointEconomyCool : REAL;
rSetpointProtectionCool : REAL;
```

rSetpointComfortHeat: Setpoint for the Comfort heating energy level. The variable is saved persistently. Preset to 21.0.

rSetpointPreComfortHeat: Setpoint for the PreComfort heating energy level. The variable is saved persistently. Preset to 19.0.

rSetpointEconomyHeat: Setpoint for the Economy heating energy level. The variable is saved persistently. Preset to 15.0.

rSetpointProtectionHeat: Setpoint for the Protection heating energy level. The variable is saved persistently. Preset to 12.0.

rSetpointComfortCool: Setpoint for the Comfort cooling energy level. The variable is saved persistently. Preset to 24.0.

rSetpointPreComfortCool: Setpoint for the PreComfort cooling energy level. The variable is saved persistently. Preset to 28.0.

rSetpointEconomyCool: Setpoint for the Economy cooling energy level. The variable is saved persistently. Preset to 35.0.

rSetpointProtectionCool: Setpoint for the Protection cooling energy level. The variable is saved persistently. Preset to 40.0.

3.3.2 Controller

3.3.2.1 FB_BARPICtrl

Simple PI controller. The control gain has no influence on the I-component.

FB_BARPICtrl	
eDataSecurityType	rY
bSetDefault	rE
bEnable	rEMin
rW	rEMax
rX	bARW
tTaskCycleTime	bMaxLimit
uiCtrlCycleCall	bMinLimit
bSync	bError
bDirection ▸	udiErrorId
rXp ▸	▸ bDirection
tTn ▸	▸ rXp
rYMin ▸	▸ tTn
rYMax ▸	▸ rYMin
rSyncValue ▸	▸ rYMax
	▸ rSyncValue

This PI controller does not work directly with a gain factor K_p , but instead calculates this internally from the so-called proportional band (input rXp) in relation to the control value limits ($rYmin$ and $rYmax$), from which K_p is then determined internally.


Inputs/outputs

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
rW                : REAL;
rX                : REAL;
tTaskCycleTime   : TIME;
uiCtrlCycleCall  : UINT;
bSync            : BOOL;
```

eDataSecurityType: if $eDataSecurityType := eHVACDataSecurityType_Persistent$, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If $eDataSecurityType := eHVACDataSecurityType_Idle$ the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: controller activation

rW : setpoint.

rX : actual value.

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.

uiCtrlCycleCall : call cycle of the function block as a multiple of the cycle time. . A zero entry is automatically interpreted as `uiCtrlCycleCall =1`.

Example: `tTaskCycleTime = 20ms, uiCtrlCycleCall =10` -> The control algorithm is called every 200 ms. Thus the outputs are also updated only every 200 ms.

bSync: a rising edge at this input sets the (internal) I-component such that `rSyncValue` is output at the control value output. If the I-component is deactivated by `tTn=0ms`, however, then this command is ignored.

VAR_OUTPUT

```
rY      : REAL;
rE      : REAL;
bARW    : BOOL;
bMaxLimit : BOOL;
bMinLimit : BOOL;
bError   : BOOL;
udiErrorId: UDINT;
```

rY : control value.

rE : control deviation (calculation dependent on [control direction](#) [▶ 116])

rEMin : lower control deviation limit value, which results from the input proportional band.

rEMax : upper control deviation limit value, which results from the input proportional band.

bARW: anti-Reset-Windup function is active.

bMaxLimit : the control value has reached its upper limit value.

bMinLimit : the control value has reached its lower limit value.

bError: this output is switched to TRUE if the parameters entered are erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes](#) [▶ 237].

VAR_IN_OUT

```
bDirection : BOOL;
rXp        : REAL;
tTn        : TIME;
rYMin      : REAL;
rYMax      : REAL;
rSyncValue : REAL;
```

bDirection: the [control direction](#) [▶ 116] of the controller can be changed with the parameter `bDirection`. If `bDirection` is TRUE, then the direct control direction is active for cooling operation of the controller. If `bDirection` is FALSE, then the indirect control direction of the controller is activated for heating operation. The variable is saved persistently. Preset to FALSE.

rXp: proportional band. This defines the internal proportional factor, see below. The proportionality factor or gain affects only the P-part. The variable is saved persistently. Preset to 100.0.

tTn: integral action time in seconds. The I-part corrects the residual control deviation following correction of the P-part. The smaller tTi is set, the faster the controller corrects. The control loop becomes unstable if the time is too short. Larger tTi -times must be entered in order to reduce the integration component. The integral action time should be selected to be longer than the stroke time of the valve or damper drive. A zero value at this input deactivates the I-component. The variable is saved persistently. Preset to 30 s.

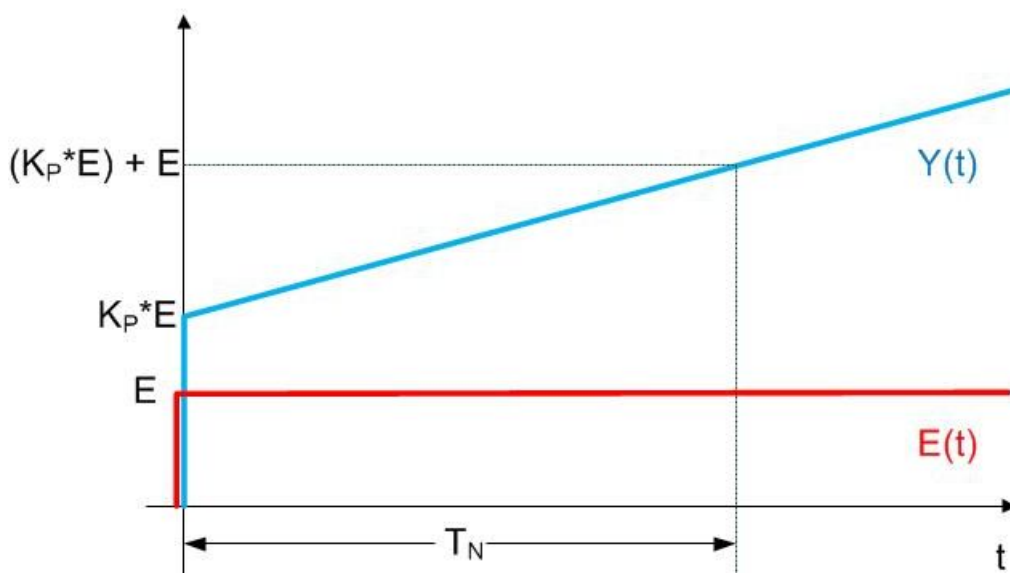
rYMin / rYMax: limiting the working range of the controller. Several other function blocks, e.g. sequencers, require a symmetrical control range (-100 to +100). In the case of a cascade structure, the working range of the master controller determines the setpoint of the slave controller. For example, 15° to 25° as the limitation of the supply air set value for an exhaust/supply air cascade control. The variable is saved persistently. Preset to 0.0 or 100.0 respectively.

rSyncValue: a rising edge at this input $bSync$ sets the control value rY to this value. In addition, the I-component is changed internally. If the I-component doesn't exist (PD controller), the D-component is changed. The variable is saved persistently. Preset to 0.0.

Functional description

Step response of a simple PI controller, where the control gain has no influence on the integral component. Response of output $Y(t)$ to a control deviation jump by E : when the control deviation jumps by E , output Y first jumps to $K_p \cdot E$ by the proportional component and then grows by a further E in each interval T_N .

Note: the controller is designed in such a way that the controller starts at 0, i.e. without the $K_p \cdot E$ jump, after a PLC reset or restart.



Basic function

A TRUE signal at the $bEnable$ input activates the function block. The internal control algorithm is now processed. The input value $uiCtrlCycleCall$ specifies the number of PLC cycles after which the internal control algorithm is processed. If $uiCtrlCycleCall = 1$, then the new calculation takes place in each PLC cycle; if, conversely, it is set to 100, then a new calculation of the output values takes place only every 100 PLC cycles. The PLC cycle time is also accounted for in the control value calculation. An incorrect input value leads to incorrect calculation.

The inputs rW (setpoint), rX (actual value), rXp (proportional band) and rTn (integral action time) are the input values of the PI controller. They are used in each calculation cycle for the determination of the output values rY (control value) and rE (control deviation). The control value can additionally be limited by the inputs $rYMin$ and $rYMax$.

Setting via the proportional band

The adjustment of the gain factor K_p of a controller often harbors the difficulty for the user that there is no size reference to the application. If a heating controller normally operates within the two-figure range, then a flow rate controller can accept values in the five-figure range. It therefore makes sense to represent the K_p

factor in such a way that it has a reference to the possible control deviation and change of control value. The P-part of the controller is regarded for the dimensioning of the K_p factor. The equation for this is:

- Control value = control deviation x gain factor $\rightarrow Y = E \cdot K_p$

this relationship also applies to the changes in the control deviation and the control value:

- Change in control value = change in control deviation • gain factor $\rightarrow \Delta Y = \Delta E \cdot K_p$

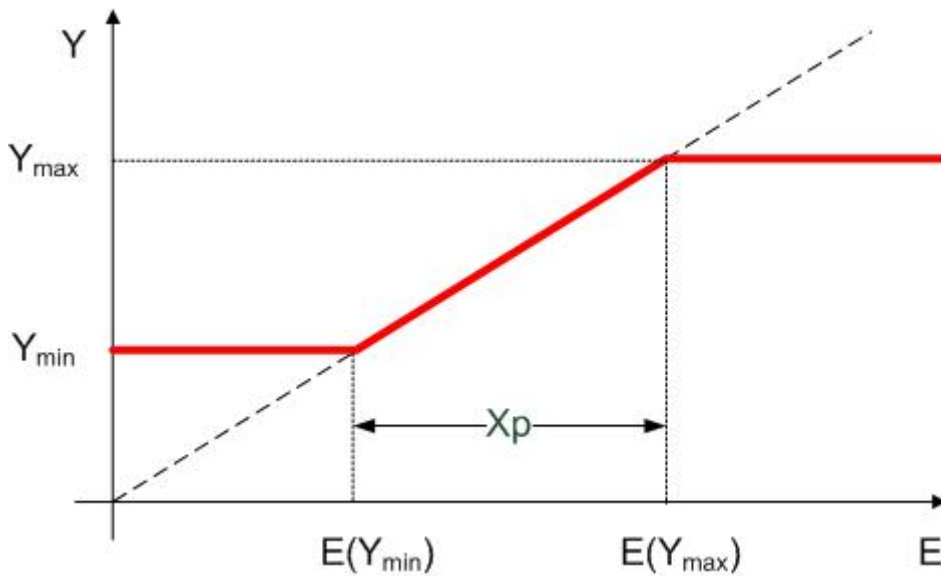
Referenced to the minimum and maximum value of the control value, Y_{min} and Y_{max} :

- $Y_{max} - Y_{min} = (E(Y_{max}) - E(Y_{min})) \cdot K_p$

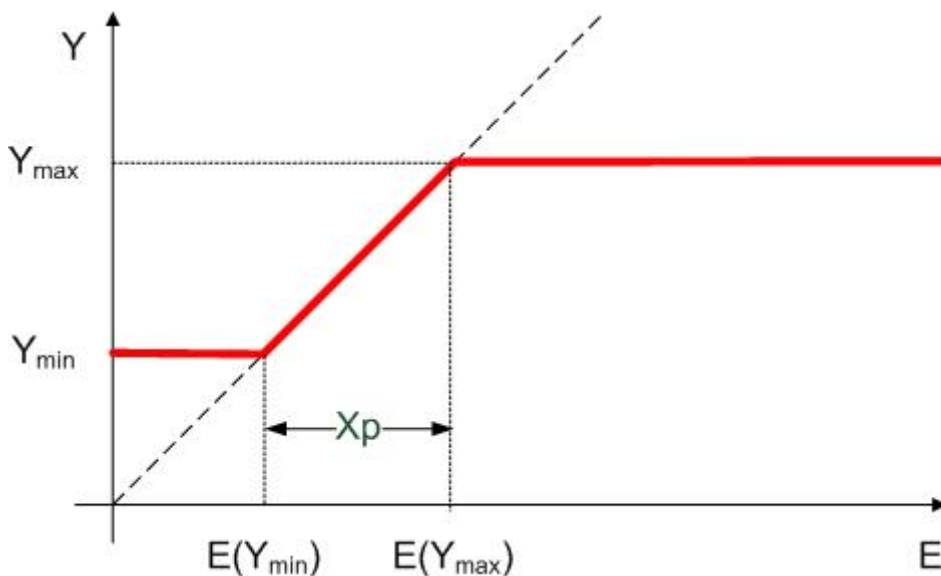
This difference, $E(Y_{max}) - E(Y_{min})$, is called the proportional band (X_p). Transposed, the equation is then:

- $K_p = (Y_{max} - Y_{min}) / X_p$

The following diagram clarifies the functional interrelationship:



The proportional band X_p therefore indicates the size of the range of the control deviation that leads to an output of Y_{min} to Y_{max} from the controller. A smaller X_p leads to a steeper function and thus to an increase of the K_p factor. However, the control deviation limit values $E(Y_{max}) - E(Y_{min})$ are shifted:



Control direction

Control direction

If $bDirection = FALSE$, the control direction of the controller is reversed so that a control deviation of less than 0 causes a change in the control value in the positive direction. This is achieved by a negative calculation of the control deviation:

bDirection	rE (control deviation)	Control direction
TRUE	$IrX - IrW$ (actual value-set value)	direct (cooling)
FALSE	$IrW - IrX$ (set value-actual value)	indirect (heating)

Anti-Reset-Windup (ARW)

If the controller “runs” into this limit, then the I-component is held internally at the final value. If this were not done, then the I-component could adopt very large values without hindrance during the limit case, which would first have to be eliminated again in case of reversal of the direction of action of the controller. This function is called “Anti-Reset-Windup” (ARW). The output $bARW$ is set if this function is active.

Special case: $T_n=0$ as switch-off of the I-component

From the above step response diagram it can be seen that the influence of the I-component becomes all the weaker, the larger the integral action time T_n is set. As the integral action time approaches infinity, the influence of the I-component is virtually zero. Conversely, an increasingly smaller integral action time allows the influence of the I-component to grow; at $T_n=0$ the control value would approach infinity. However, this special case is used to **cut off** the I-component. This is an internally formed exception, since the integral action time belongs directly to the I-component and should also figuratively result in switch-off due to the zero entry.

Synchronization

A positive edge on $bSync$ sets the controller output rY directly to $rSyncValue$, provided that the controller has been activated by a TRUE signal on $bEnable$. If this is not the case the positive edge on $bSync$ is ignored.

Error case/function block not activated

If the controller is incorrectly parameterized processing is stopped, the $bError$ output is set and the corresponding error ID is output at $udiErrorID$, see [error codes \[► 237\]](#). The function block is also stopped if the input $bEnable$ is not set. In both cases the outputs are set as follows:

rY	0.0
rE	0.0
$bARW$	FALSE
$bMaxLimit$	FALSE
$bMinLimit$	FALSE

Documents about this

 [example_persistent_e.zip \(Resources/zip/11659714827.zip\)](#)

3.3.3 Lighting

3.3.3.1 Lighting functions – overview

The lighting is subdivided into two different function groups, which can be combined to form arbitrary solutions:

- User functions
- Actuator functions

The **user functions** are control and closed loop circuits, which, served by the sensor functions, each output a light value.

[FB_BARLightCircuit \[▶ 133\]](#) - simple light switching circuit without dimming function.

[FB_BARLightCircuitDim \[▶ 134\]](#) - simple light switching circuit with dimming function.

[FB_BARAutomaticLight \[▶ 117\]](#) - presence-controlled automatic light with switch-off delay.

[FB_BARStairwellAutomatic \[▶ 138\]](#) - stairwell lighting with early warning sequence.

[FB_BARTwilightAutomatic \[▶ 140\]](#) - twilight automatic.

[FB_BARDaylightControl \[▶ 126\]](#) - daylight control without dimming procedures

[FB_BARConstantLightControl \[▶ 119\]](#) - constant light regulation with continuous output of analog values.

The group of **actuator functions** is currently represented by only one function block.

[FB_BARLightActuator \[▶ 130\]](#) - output of a preset proportional dimming value via a ramp function. Output is alternatively in percent, INTEGER or BOOL. This function block likewise encompasses a light scene memory of 21 adjustable dimming values.

3.3.3.2 FB_BARAutomaticLight

Function block for an automatic light circuit as used in corridors or sanitary facilities.

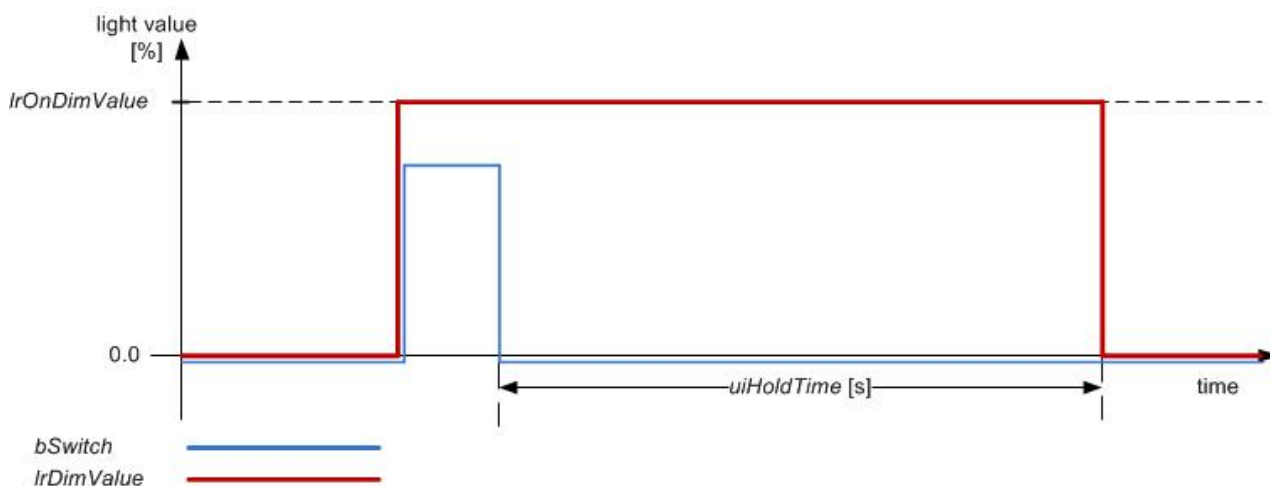


Fig. 2: FB_BARAutomaticLight

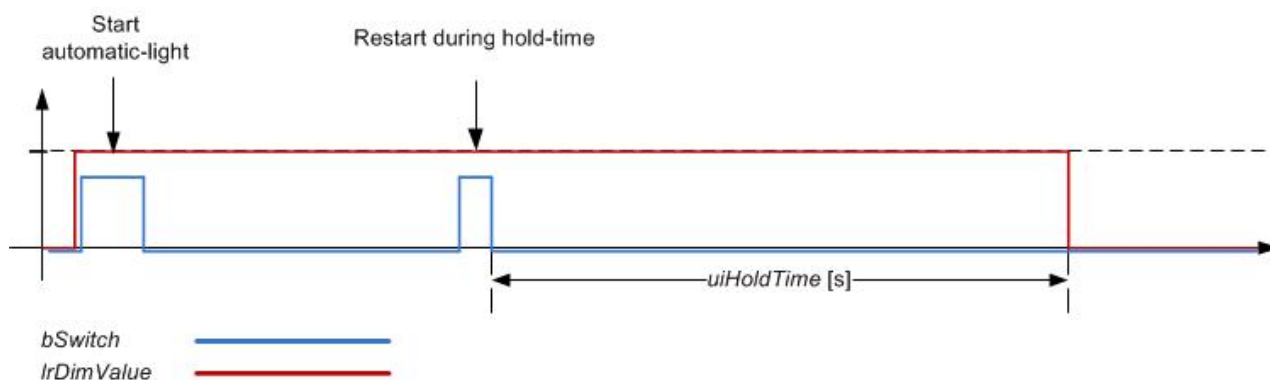
The function block knows 3 different modes, which can be set via the input *uiLightCtrlMode*:

- Automatic mode
- Manual On mode
- Manual Off mode

The automatic light circuit is active in automatic mode (*uiLightCtrlMode*=0). A positive edge on *bSwitch* sets the output *IrDimValue* to the value entered under *IrManualDimValue*. A negative edge starts the holding time generator. If the hold time *uiHoldTime* [s] has expired, the output *IrDimValue* is reset to 0.0.



The sequence can be restarted at any time:



In the manual operation modes the input $bSwitch$ has no function: with $uiLightCtrlMode=1$ the output value $IrDimValue$ is constantly set to $IrManualDimValue$ and with $uiLightCtrlMode=2$ constantly set to 0.0.

Changing to manual mode resets any hold time that had started up to that point.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
uiLightCtrlMode   : UINT;
bSwitch           : BOOL;
```

eDataSecurityType: if $eDataSecurityType:=eDataSecurityType_Persistent$, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instantiated once in the main program, which is called cyclically. Otherwise the instantiated FB is not internally released.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If $eDataSecurityType:=eDataSecurityType_Idle$ the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if $eDataSecurityType:=eHVACDataSecurityType_Persistent$. It would lead to early wear of the flash memory.

uiLightCtrlMode : operation mode.

- 0: automatic mode; the automatic light circuit is active and reacts to the input *bSwitch*.
- 1: Manual On; the automatic light circuit is inactive - the output *lrDimValue* is set constantly to *lrManualDimValue*.
- 2: Manual Off; the automatic light circuit is inactive - the output *lrDimValue* is set constantly to 0.0.

bSwitch: a rising edge switches the light on in automatic mode (*uiLightCtrlMode*=0), a falling edge starts the holding time generator. This input has no function in manual operation mode (*uiLightCtrlMode*=1 or 2).

VAR_OUTPUT

```
lrDimValue      : LREAL;
uiRemainingHoldTime: UINT;
bError          : BOOL;
udiErrorId      : UDINT;
```

lrDimValue : output dimming value for the lighting in percent.


uiRemainingTimeHold : remaining hold time in seconds. If the light is off or if manual operation mode is active, then this output is "0". With a rising edge on *bSwitch* in automatic mode, this output initially indicates the complete number of seconds of the hold time (*uiHoldTime*), in order to illustrate the countdown of the hold time, starting with a falling edge on *bSwitch*. This output is 0 as long as no countdown of the time is taking place.

bError: this output is switched to TRUE if the parameters entered are erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes](#) [▶ 237].

VAR_IN_OUT

In order for the registered parameters to be retained beyond a controller failure, it is necessary to declare them as In-Out variables. A reference variable is then assigned to them in the program. Each change of the value of these reference variables is persistently stored in the function block and written back to the reference variable following a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

```
uiHoldTime      : UINT;
lrOnDimValue    : LREAL;
lrManualDimValue : LREAL;
```

uiHoldTime : hold time [s] of the automatic light controller after a falling edge on *bSwitch*.

lrOnDimValue : on dimming value in automatic mode (*uiLightCtrlMode*=0).

lrManualDimValue : output dimming value in Manual On mode (*uiLightCtrlMode*=1).

3.3.3.3 FB_BARConstantLightControl

The constant light regulation function block controls the lighting when the room is occupied such that the lighting intensity does not fall below a preset minimum. This ensures adequate lighting whilst at the same time minimizing energy consumption.

The constant light regulation is enabled when the room is entered or by a rising edge on the input *bPresence*. Optionally the constant light regulation can also be operated by a button. A short pulse on the input *bToggle* enables the constant light regulation or switches it off, depending on the current state. If many constant light controllers are to be switched on by a central command, for example on an office floor, this can be realized with the input *bCentralOn*. A central command to switch off can be connected to the input *bCentralOff*.

Actuating the button on *bToggle* for a longer period manually changes the setpoint value for the room brightness. Upon a falling edge on *bToggle* the current room brightness value is saved as a setpoint. The rate of change when manually increasing or decreasing the room brightness can be set by means of a parameter.

The manually set setpoint value for the constant light regulation is retained until the next time the lighting is switched off. On restarting the controller by a rising edge on *bPresence*, a rising edge on *bCentralOn* or a pulse on *bToggle*, the last manually selected setpoint or the value of *uiSetpointValue* is adopted, depending on the setting of the parameter *bInitialMode*.

In automatic mode the lighting is switched on only if the room brightness lies below an adjustable hysteresis value (*uiTargetRange*). As the outdoor brightness increases, the constant light controller reduces the artificial light portion until a minimum load control value *lrMinDimValue* lies at the output of the controller *lrDimValue*. Subsequently, the controller switches the lighting off following a time delay with the timer *uiOffDelay*. The switching on again of the lighting as the outdoor brightness decreases is delayed with the timer *uiOnDealy*.

In order to avoid unpleasant visible changes in the brightness, the rate of change of the control signal is retarded with the parameter *uiControlRampTime*.

For maintenance and test purposes the automatic constant light regulation can be deactivated and the lighting can be switched on and off in manual mode. This deactivates the edge detection at the inputs *bCentralOn*, *bCentralOff* and *bToggle*.



Fig. 3: FB_BARConstantLightControl

The function block knows 3 different modes, which can be set via the input *uiLightCtrlMode*:

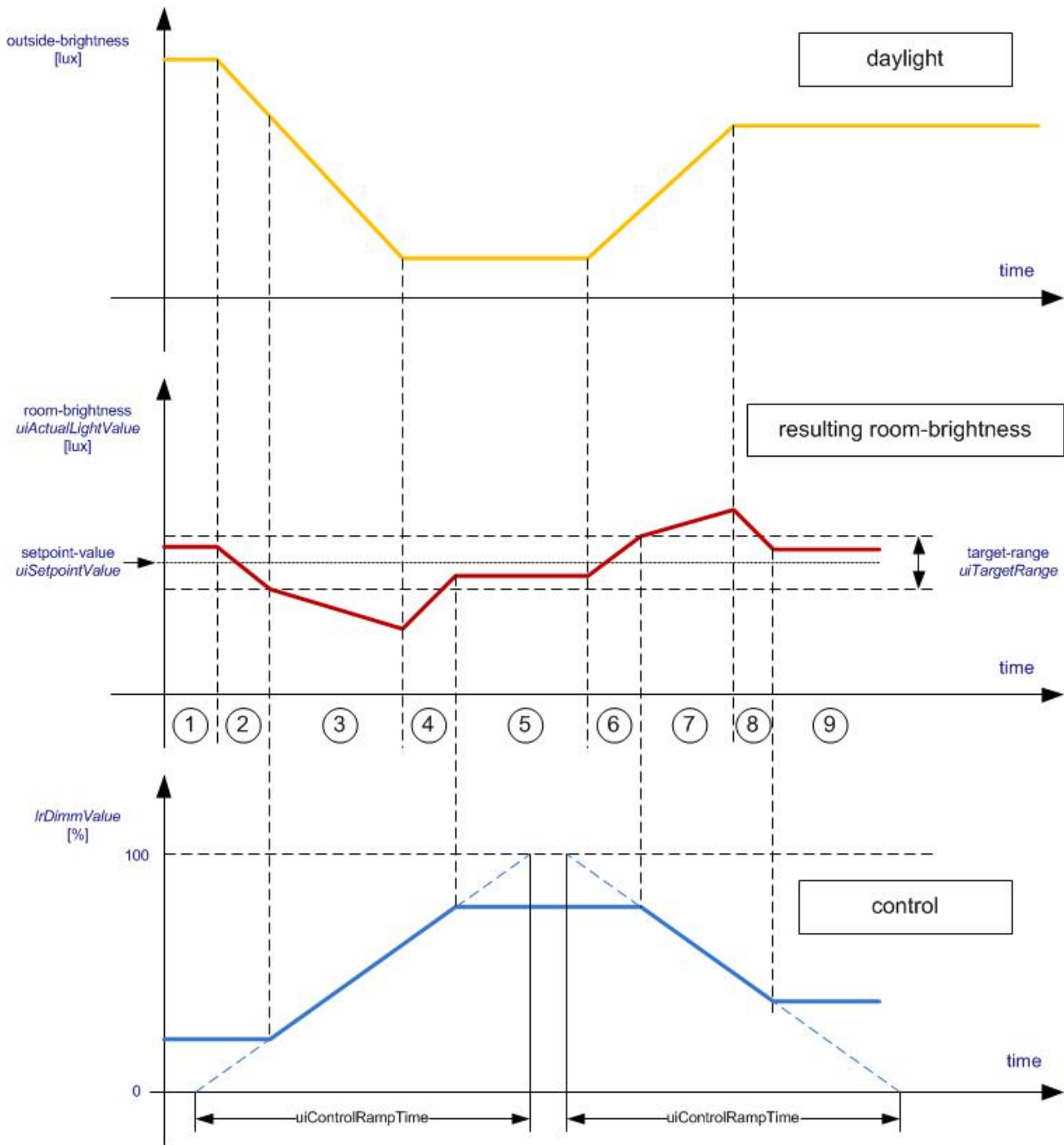
- Automatic mode
- Manual On mode
- Manual Off mode

Automatic mode

In automatic mode the control can be activated or deactivated in three different ways:

1. Via the input *bSwitch*: a short TRUE signal (shorter than *uiSwitchOverTime* in milliseconds) at *bSwitch* activates the constant light regulation if it had been inactive beforehand. Another short signal at *bSwitch* deactivates the constant light regulation again. A long TRUE signal on *bSwitch* switches the function block to setpoint adjustment mode; this is described below.
2. Via the input *bPresence*: switching on via this input must be explicitly enabled by a TRUE signal on *bPresenceOnActive* (VAR_IN_OUT). A rising edge then activates the control, while a falling edge always deactivates the control independent of the state of *bPresenceOnActive*.
3. The control is explicitly switched on and off via TRUE edges on the inputs **bCentralOn** and *bCentralOff*, independent of the previous state of the control. These inputs do not have an override function: for example, if switching off has taken place via *bCentralOff*, *bSwitch* can be used to switch on again at any time.

The light output value *IrDimValue* is initially set in the active state to the start value *IrStartDimValue*. After that an actual value/setpoint value comparison takes place continuously. If the actual room value *uiBrightness* [lux] thereby leaves the target range *uiTargetRange* [lux] around the setpoint value *uiSetpointValue* [lux], then this is counteracted by dimming the lighting up or down. The control always operates with a constant, parameterizable dimming ramp *uiRamptime* [s], which indicates the time of a complete dimming process from 0% to 100%. The control target values are likewise constant: 0% for dimming down and 100% for dimming up. In the inactive state the light output value *IrDimValue* is set to 0.0.



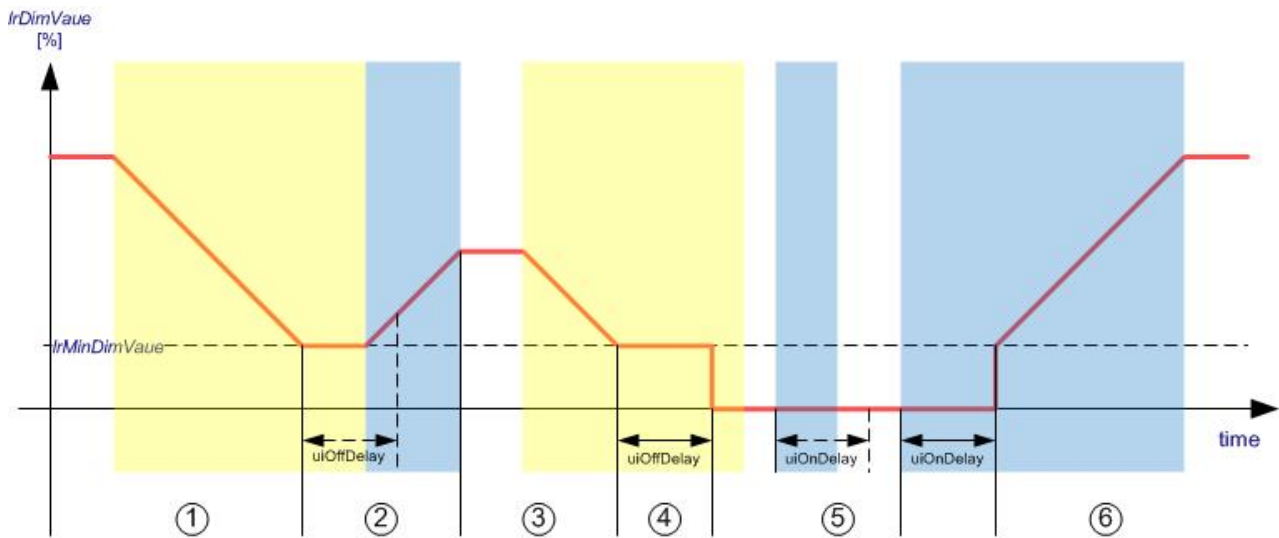
- ① Initial situation: the room-brightness is well-regulated.
- ② The outside-brightness decreases and thus the room-brightness. Because the room-brightness is still within the target-range, the control will not react.
- ③ The outside-brightness decreases further and the room-brightness leaves the target range. The control starts its ramp – due to the lower slope of the ramp, the control only succeeds in damping the loss of room-brightness.
- ④ The outside-brightness does not decrease any more. The control-ramp is now able to compensate the room-brightness.
- ⑤ The target-range is reached again.
- ⑥ The outside-brightness rises and thus the room-brightness. Because the room-brightness is still within the target-range, the control will not react.
- ⑦ The outside-brightness continues to rise and the measured room-brightness leaves the target-range. The control starts its ramp – due to the lower slope of the ramp, the control only succeeds in damping the gain of room-brightness.
- ⑧ The outside-brightness does not increase any more. The control-ramp is now able to compensate the room-brightness.
- ⑨ The target-range is reached again. Because the outside-brightness is now lower than in section 1, the electrical dim-value is higher.

Switch-on and switch-off delay, minimum output value

If the outdoor brightness increases, then less and less artificial light is necessary in order to obtain the desired total brightness. If the outdoor brightness is sufficient, the lighting can also be switched off completely.

However, switch-on and switch-off processes can be perceived as distracting, as can very low output dimming values. Therefore a switch-on and switch-off delay $uiOnDelay/uiOffDelay$ can be defined in the function block around a lower limit value $IrMinDimValue$. If the internally determined output value sinks below this minimum value, then the output remains at this minimum for the time $uiOffDelay$ [s]. Only after that is 0.0 output at the output $IrDimValue$. If in the reverse case switching off to 0.0 has taken place, then if artificial light is required this is only switched on after the expiry of $uiOnDelay$ [s] and then to the value $IrMinDimValue$. The following diagram is intended to clarify the behavior:

The yellow areas indicate the ranges where the outdoor brightness permits the lighting to be dimmed down; in the blue areas, conversely, the outdoor brightness alone is not sufficient to obtain the desired room brightness. In the white areas the output control value $IrDimValue$ is appropriate for the lighting conditions.



- ① The light value outside is sufficient, so the light inside the room will be dimmed.
- ② At the beginning of this section, the waiting time to turn off the light begins. But before this time is elapsed it gets dark outside. Thus the light-value inside the room will be increased.
- ③ The light outside is sufficient again. After a while, it gets brighter. The light inside will be dimmed.
- ④ The waiting time to turn off the light inside is started again. This time, it is bright enough outside for such a long time, that the lamp is turned off.
- ⑤ A short span of time with lower brightness outside will not let the lamp turn on again.
- ⑥ Only if the time of lower brightness outside, $uiOnDelay$, is exceeded, the light inside the room will be turned on again.

Manual adjustment of the setpoint

In order to be able to adapt the light control to personal brightness needs, there is an option to increase or decrease the setpoint value. A long TRUE signal on $bSwitch$ (longer than $uiSwitchOverTime$ in milliseconds) switches the function block to the dimming mode and the light is dimmed up if it had been dimmed down in the preceding dimming mode and vice versa. If $bSwitch$ resets to FALSE, then the brightness value now measured on $uiBrightness$ is adopted as the new setpoint value to which control is to take place.

Manual mode

In the manual operation modes the inputs $bSwitch$, $bCentralOn$ and $bCentralOff$ have no function: with $uiLightCtrlMode=1$ the output value $IrDimValue$ is constantly set to $IrManualDimValue$ and with $uiLightCtrlMode=2$ it is set constantly to 0.0.


Changing to manual mode resets any control process that had previously started. On re-entering automatic mode the output value is 0.0 and the control must be restarted. $IrDimValue$

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
uiLightCtrlMode   : UINT;
bSwitch           : BOOL;
bPresence         : BOOL;
bCentralOn       : BOOL;
bCentralOff      : BOOL;
uiBrightness      : UINT; (*lux*)
uiSetpoint        : UINT; (*lux*)
```

eDataSecurityType: if `eDataSecurityType:= eDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instantiated once in the main program, which is called cyclically. Otherwise the instantiated FB is not internally released.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

uiLightCtrlMode : operation mode.

- 0: Automatic mode; the commands `bSwitch`, `bCentralOn` and `bCentralOff` are executable and can be used to switch the controller on and off.
- 1: Manual On mode; constant light regulation is inactive – the value `IrManualDimValue` is output without a delay at the output `IrDimValueOut`, all other command inputs are ineffective.
- 2: Manual Off mode; constant light regulation is inactive – the value 0.0 is output without a delay at the output `IrDimValueOut`, all other command inputs are ineffective.

bSwitch : positive edges on this input switch the control on and off alternately. When switching off the output `IrDimValue` is set to 0.0. This command input is active only in automatic mode (`uiLightCtrlMode=0`).

bPresence : a continuous TRUE signal on this input activates the control if the presence function is activated by `bPresenceOnActive=TRUE` (VAR_IN_OUT). Conversely, a falling edge on this input always deactivates the control. This command input is active only in automatic mode (`uiLightCtrlMode=0`).

bCentralOn : a positive edge on this input switches the control on. This command input is active only in automatic mode (`uiLightCtrlMode=0`).

bCentralOff : a positive edge on this input switches the control off and sets the output `IrDimValue` to 0.0. This command input is active only in automatic mode (`uiLightCtrlMode=0`).

uiBrightness : actual light value [lux].

uiSetpointValue : light setpoint [lux].

VAR_OUTPUT

```
IrDimValue         : LREAL;
bControlActive     : BOOL;
bAdjustedSetpointActive: BOOL;
uiAdjustedSetpoint : UINT;
diActDeviation     : DINT;
bError             : BOOL;
udiErrorId         : UDINT;
```

IrDimValue : light output value, 0..100%.

bControlActive : this output is TRUE if the function block is in automatic mode and the control is activated. This is intended to serve as an additional feedback signal if switch-on has taken place but the control outputs a light value of $IrDimValue=0.0$.

bAdjustedSetpointActive : if the control is active and the setpoint value has been manually adjusted (see above), then the state of this output changes to TRUE in order to indicate that the setpoint value on the input $uiSetpointValue$ is no longer active.

uiAdjustedSetpoint : this output indicates the active setpoint value if this has been manually adjusted ($bAdjustedSetpointActive = TRUE$). This output is set to 0 if no manually adjusted setpoint value is active.


diActDeviation : current control deviation in lux. This output indicates a valid value only if the function block is in automatic mode and activated. Otherwise 0.0 is output.

bError: this output is switched to TRUE if the parameters entered are erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes](#) [► 237].

VAR_IN_OUT

In order for the registered parameters to be retained beyond a controller failure, it is necessary to declare them as In-Out variables. A reference variable is then assigned to them in the program. Each change of the value of these reference variables is persistently stored in the function block and written back to the reference variable following a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>.

```
uiSwitchOverTime : UINT;
uiAdjustRampTime : UINT;
bPresenceOnActive : BOOL;
bInitialMode : BOOL;
uiTargetRange : UINT;
uiControlRampTime : UINT;
uiOnDelay : UINT;
uiOffDelay : UINT;
lrMinDimValue : LREAL;
lrMaxDimValue : LREAL;
lrStartDimValue : LREAL;
lrManualDimValue : LREAL;
```

uiSwitchOverTime : switching time in milliseconds for the input $bSwitch$ for the recognition of short and long signals. (short signal: switch-on/switch-off function; long signal: dimming function)

uiAdjustRampTime : ramp time in seconds with which the setpoint value is changed during manual adjustment.

bPresenceOnActive : if this input is TRUE, then the control is activated by a positive edge on $bPresence$ if the function block is in automatic mode ($uiLightCtrlMode = 0$).

bInitialMode : a TRUE signal on this input makes the function block begin with the setpoint value on $uiSetpoint$ each time it is activated. If on the other hand this input is FALSE, then the setpoint value that was last active – i.e. including manually adjusted setpoint values – is adopted on activation of the function block.

uiTargetRange : target range around the setpoint value in which no further control is to take place.

uiControlRampTime : ramp time in seconds (time required to dim from 0% to 100%).

uiOnDelay : switch-on delay in seconds around the minimum value $lrMinDimValue$.

uiOffDelay : switch-off delay in seconds around the minimum value $lrMinDimValue$.

lrMinDimValue : lower limit value for dimming, see [introduction](#) [► 119].

lrMaxDimValue : upper limit to which the output $IrDimValue$ can be controlled.

lrStartDimValue : value to which the light should jump on activating the control.

lrManualDimValue : output dimming value in Manual On mode ($uiLightCtrlMode=1$).

3.3.3.4 FB_BARDaylightControl

Daylight switch. Unlike the constant light regulation [► 119], this automatic control operates not with dimming values, but merely switches the light on or off depending on the measured brightness.

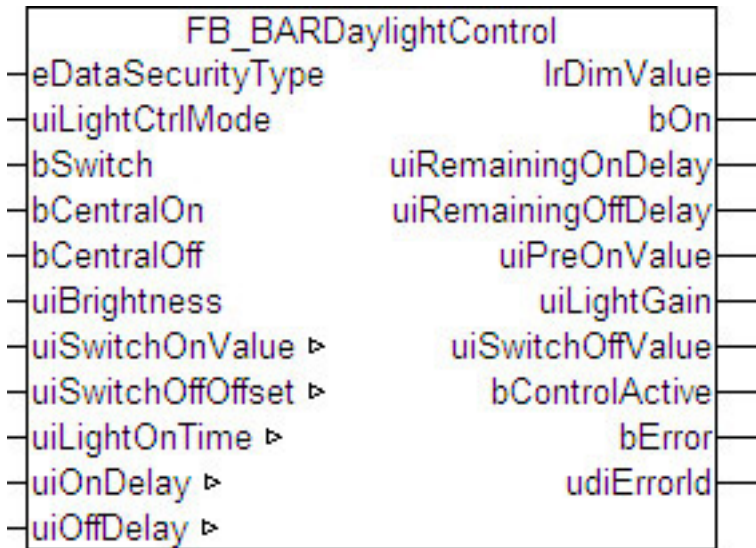


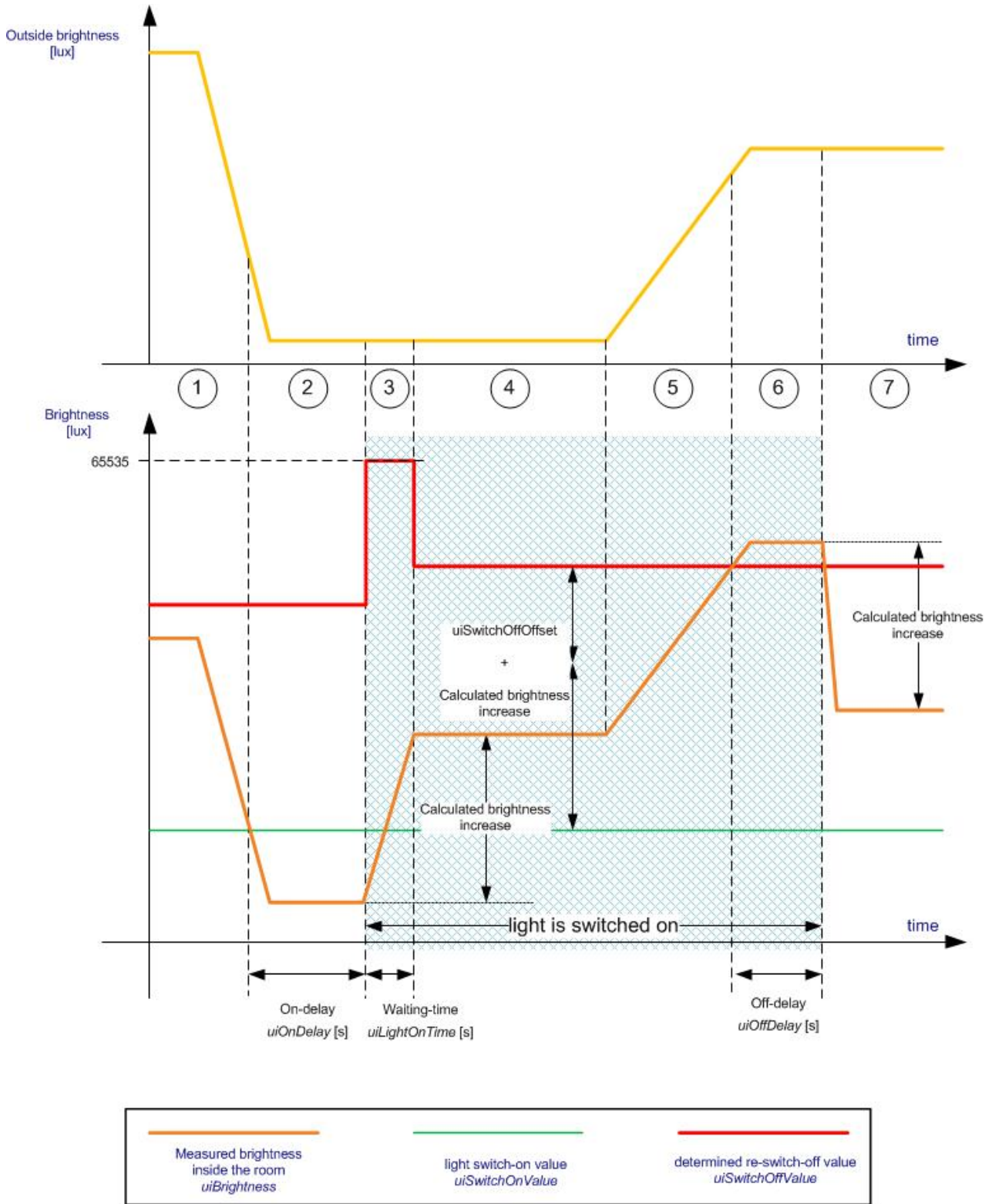
Fig. 4: FB_BARDaylightControl

The function block knows 3 different modes, which can be set via the input *uiLightCtrlMode*:

- Automatic mode
- Manual On mode
- Manual Off mode

Automatic mode

If the brightness in the room, *uiBrightness*, falls below the value *uiSwitchOnValue* [lux] for the time *uiOnDelay* [s], then the light is switched on. There is now a delay, *uiLightOnTime*, to allow the lamps reach their full luminosity. The measured brightness in the room – **assuming constant outdoor brightness** – is then higher than before switching on by the amount of the increase in brightness. Switching the lighting off again makes sense only after the outdoor brightness has significantly increased again. This limit value is calculated from the light switch-on value plus the brightness increase plus the parameter *uiSwitchOffOffset*. If an increase in the outdoor brightness causes the room brightness to exceed this determined re-switch-off value for the duration of *uiOffDelay* [s], then the lighting is switched off again.



- 1 The outside-brightness decreases and with it the measured brightness inside the room. The value falls below *uiSwitchOnValue*.
- 2 The measured brightness inside the room remains below the light switch-on value *uiSwitchOnValue* for the time *uiOnDelay* [s].

First the function-block memorises the measured brightness inside the room (without lighting). Afterwards the light is turned on. In order to judge the increase in light intensity, a time delay of *uiLightOnTime* in milliseconds is allowed to elapse.
During this phase, the external brightness is assumed constant.
Before the re-switch-off value *uiSwitchOffValue* is recalculated in the next phase, it will be set to 65535, the maximum value for its data-type. This prevents the lighting to be turned off by its own light-increase.
- 3
- 4 After the waiting-time is elapsed, the light gain will be calculated as the difference between the actual measured brightness and the light-value, which was memorised before the light was turned on. The „determined re-switch-off value“ *uiSwitchOffValue* is then calculated according to the formula:
$$\text{light switch-on value} + \text{calculated brightness increase} + \text{uiswitchOffOffset}$$
- 5 Outside the room it gets brighter again. Thus the measured room-brightness increases as well and exceeds the previously determined re-switch-off value.
- 6 The brightness inside the room remains above the re-switch-off value for the time *uiOffDelay* [s].
- 7 The Lighting is switched off. Thus the brightness inside the room is decreased by the light-gain.

Manual mode

In the manual operation modes, the *bSwitch*, *bCentralOn* and *bCentralOff* inputs have no function: with *uiLightCtrlMode*=1, the output value *bOn* is set to *TRUE* and with *uiLightCtrlMode*=2 to *FALSE*.


A change to manual mode resets a previously started control. On re-entering automatic mode the output value *bOn* is *FALSE* and the control must be restarted.

VAR_INPUT

```
eDataSecurityTyp: E_HVACDataSecurityType;
uiLightCtrlMode : UINT;
bSwitch         : BOOL;
bCentralOn      : BOOL;
bCentralOff     : BOOL;
uiBrightness    : UINT;
```

eDataSecurityType:if *eDataSecurityType*:= *eDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not internally released.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235.zip>

If *eDataSecurityType*:= *eDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if *eDataSecurityType*:= *eHVACDataSecurityType_Persistent*. It would lead to early wear of the flash memory.

uiLightCtrlMode : operation mode.

- 0: Automatic mode; the commands *bSwitch*, *bCentralOn* and *bCentralOff* are executable and can be used to switch the controller on and off.
- 1: Manual On mode; constant light regulation is inactive – the value *IrManualDimValue* is output without a delay at the output *IrDimValueOut*, all other command inputs are ineffective.
- 2: Manual Off mode; constant light regulation is inactive – the value 0.0 is output without a delay at the output *IrDimValueOut*, all other command inputs are ineffective.

bSwitch : positive edges on this input switch the control on and off alternately. When switching off the output *IrDimValue* is set to 0.0. This command input is active only in automatic mode (*uiLightCtrlMode*=0).

bCentralOn : a positive edge on this input switches the control on. This command input is active only in automatic mode (*uiLightCtrlMode=0*).

bCentralOff : a positive edge on this input switches the control off and sets the output *IrDimValue* to 0.0. This command input is active only in automatic mode (*uiLightCtrlMode=0*).

uiBrightness : actual light value [lux].

VAR_OUTPUT

```
IrDimValue      : LREAL;
bOn             : BOOL;
uiRemainingOnDelay: UINT;
uiRemainingOffDel : UINT;
uiPreOnValue    : UINT;
uiLightGain     : UINT;
uiSwitchOffValue : UINT;
bControlActive  : BOOL;
bError          : BOOL;
udiErrorId      : UDINT;
```

IrDimValue : in order to keep this function the same as the other light user functions, a light output value exists here too in the form of a floating point number in percent, even though the light is only switched on or off. This means: light off: *IrDimValue* = 0.0, light on: *IrDimValue* = 100.0.

bOn : switching output for the lighting.

uiRemainingOnDelay : countdown of the switch-on delay in seconds. This output is 0 as long as no countdown of the time is taking place.

uiRemainingOffDelay : countdown of the switch-off delay in seconds. This output is 0 as long as no countdown of the time is taking place.

uiPreOnValue : measured light value immediately before switching the lighting on. This output is "0" if the controller is switched off or in manual mode.

uiLightGain : calculated brightness increase after switching on the lighting and waiting for the expiry of the waiting time *uiFullLightTime*. This output is "0" if the controller is switched off or in manual mode.


uiSwitchOffValue : determined re-switch-off value, wherein the measured brightness must be larger. During the waiting phase (*uiLightOnTime*) this value jumps to 65535 in order to avoid the light switching off during this time. This output is "0" if the controller is switched off or in manual mode.

bError: this output is switched to TRUE if the parameters entered are erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes](#) [▶ 237].

VAR_IN_OUT

In order for the registered parameters to be retained beyond a controller failure, it is necessary to declare them as In-Out variables. A reference variable is then assigned to them in the program. Each change of the value of these reference variables is persistently stored in the function block and written back to the reference variable following a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

```
uiSwitchOnValue : UINT;
uiSwitchOffOffset: UINT;
uiLightOnTime   : UINT;
uiOnDelay       : UINT;
uiOffDelay      : UINT;
```

uiSwitchOnValue : light switch-on value. The lighting is switched on after the expiry of the switch-on delay if the outdoor brightness is lower than this value.

uiSwitchOffOffset : if the light is switched on for the time *uiLightOnTime* (see above), the light switch-off value is calculated from the currently measured luminous intensity plus this value.

uiLightOnTime : the lighting does not reach its true switch-on value immediately. In order to judge the increase in light intensity, this time delay in milliseconds is allowed to elapse before the increase and the light switch-off value that depends on it are calculated.

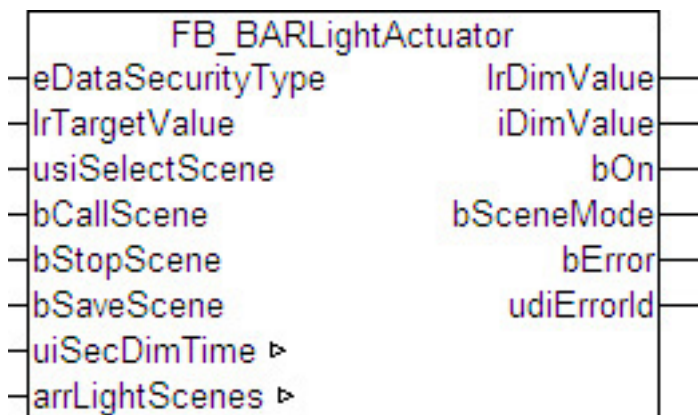
uiOnDelay : switch-on delay in seconds. Constant switching on and off of the lighting is perceived as very distracting. If the outdoor brightness sinks in such a manner that the constant light regulation needs to be switched on for support, then this should take place only after this time delay.

uiOffDelay : switch-off delay in seconds. If the outdoor brightness increases in such a manner that the constant light regulation needs to be switched off, then this should take place only after this time delay in order to mask out short-term fluctuations.

3.3.3.5 FB_BARLightActuator

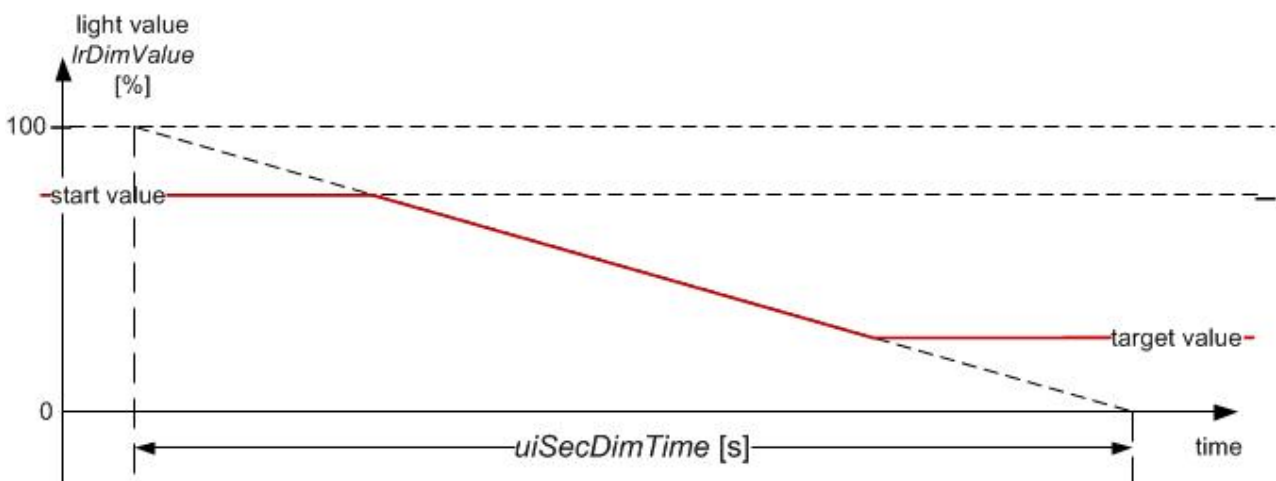
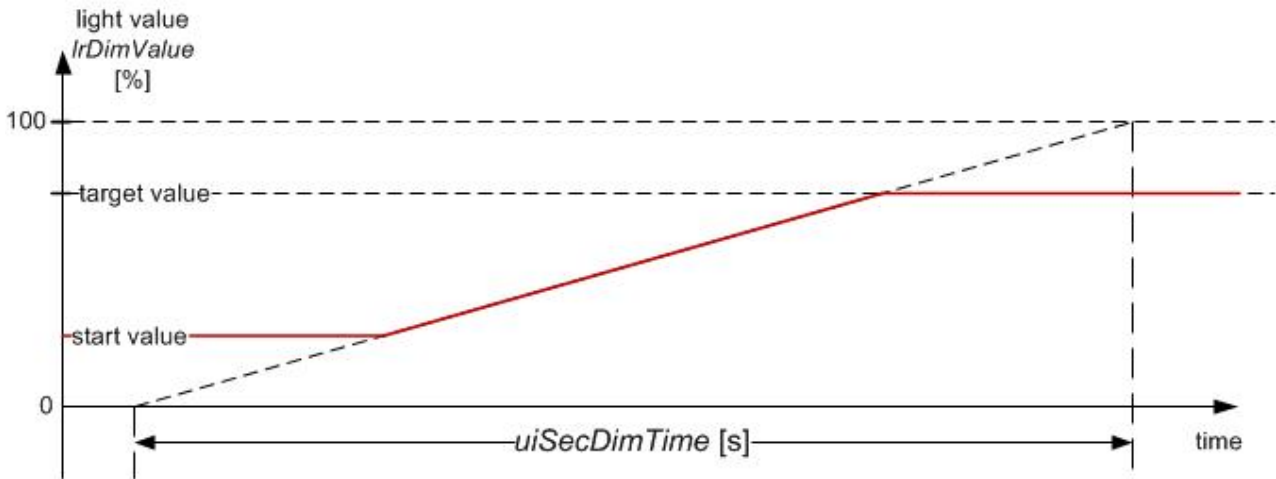
This function block serves to control a conventional light actuator. The outputs cover the value ranges 0..100%, 0..32767 and On/Off.

On top of that the function block contains a scene memory in which up to 21 different light values can be saved.



In principle the function block passes the values on the input *IrTargetValue* through to the output *IrDimValue*. Conversely, a positive edge on the input *bCallScene* sets the output to the light value that is saved in the scene table *arrLightScenes* under the index *usiSelectedScene*. The output *bSceneMode* then changes to TRUE. If the scene index *usiSelectedScene* changes, then the newly selected scene value is adopted only on another positive edge on *bCallScene*. The scene mode is quit again by a change of value on the input *IrTargetValue* or a positive edge on *bStopScene*. The output *bSceneMode* goes back to FALSE and the output *IrDimValue* once again follows the input *IrTargetValue*. A TRUE signal on *bSaveScene* saves the current light output value into the scene table *arrLightScenes* under the index *usiSelectedScene*.

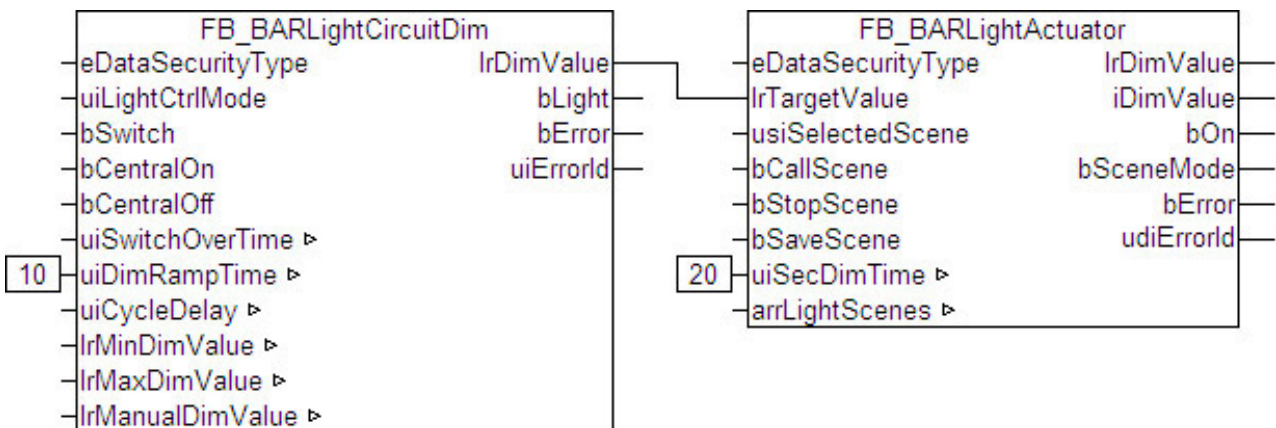
The light output value *IrDimValue* always follows the target values specified to it via a ramp. As in the case of the user function [FB_BARLightCircuitDim](#) [▶ 134], this is defined by a ramp time – in this case *uiSecDimTime* – which indicates the time interval in seconds that the light output value should require in order to change by 100%.



Since the light sensor function block [FB_BARLightCircuitDim \[► 134\]](#) also contains a ramp function for presetting the target value, the light actuator function described here is called the "secondary dimtime" - *uiSecDimTime*.

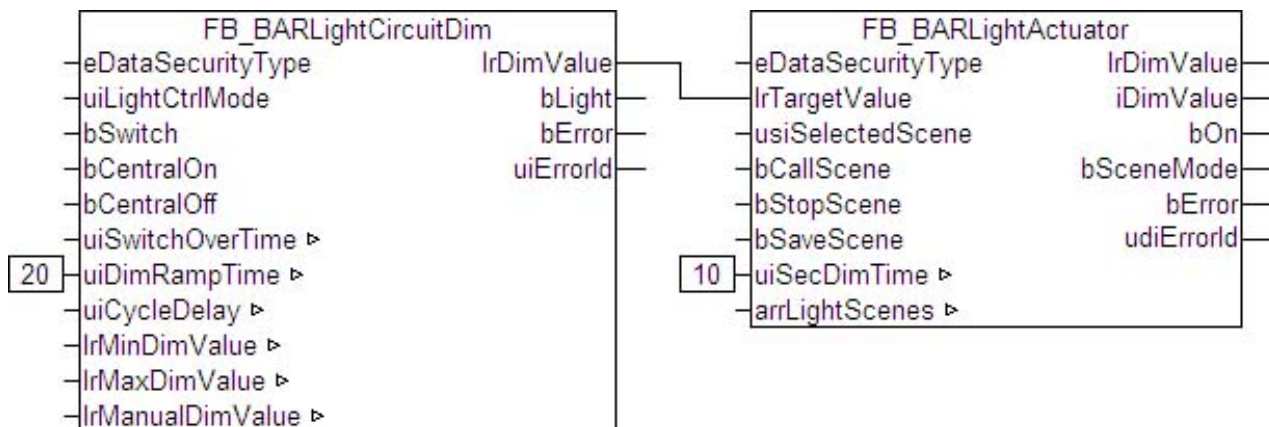
If both have different values, then the longer ramp time is always the relevant one:

Example1:



In this constellation the function block [FB_BARLightCircuitDim \[► 134\]](#) will change its output value *IrDimValue* from 0% to 100% in 10 s. However, the function block [FB_BARLightActuator](#) can follow these continuously changing setpoint values only with a ramp time of 20 s (related to a change from 0% to 100%). Therefore this is ultimately the resulting ramp time.

Example2:



Conversely the function block **FB_BARLightActuator** could follow much faster here with a ramp time of 10 s. However, since the dimming function block **FB_BARLightCircuitDim** [► 134] presets its setpoints in this example only with a ramp time of 20 s, this longer time is also relevant in this case.


It must be noted in this example that a short button press on the function block **FB_BARLightCircuitDim** immediately changes the output *IrDimValue* to the internally-saved value (see **FB_BARLightCircuitDim** [► 134]). Hence, the ramp time on the **FB_BARLightActuator** is then relevant.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
lrTargetValue    : LREAL;
usiSelectedScene : USINT;
bCallScene      : BOOL;
bStopScene     : BOOL;
bSaveScene     : BOOL;
```

eDataSecurityType: if *eDataSecurityType := eDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block **FB_HVACPersistentDataHandling** must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not internally released.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If *eDataSecurityType := eDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if *eDataSecurityType := eHVACDataSecurityType_Persistent*. It would lead to early wear of the flash memory.

lrTargetValue : target value of light output in 0..100%;

usiSelectedScene : selected lighting scene, 0..20;

bCallScene : sets the output value, controlled by a ramp, to the light value entered in the index *usiSelectedScene* (*arrLightScenes*).

bStopScene : resets the output value to the value *lrTargetValue*. The change is likewise controlled by a ramp.

bSaveScene : saves the current light value at the output *lrDimvalue* in the light value table *arrLightScenes* under the index *usiSelectedScene*.

VAR_OUTPUT

```
lrDimValue: LREAL;
iDimValue : INT;
bOn       : BOOL;
bSceneMode: BOOL;
bError    : BOOL;
udiErrorId: UDINT;
```

lrDimValue : output light value in 0..100%.

iDimValue : output light value in 0..32767.

bOn : output light status: *lrDimValue*=0.0 => *bOn*=FALSE - *lrDimValue*>0.0 => *bOn*=TRUE.


bSceneMode : the function block presently outputs a scene value and not the value *lrTargetValue* on the input.

bError: this output is switched to TRUE if the parameters entered are erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes](#) [▶ 237].

VAR_IN_OUT

In order for the registered parameters to be retained beyond a controller failure, it is necessary to declare them as In-Out variables. A reference variable is then assigned to them in the program. Each change of the value of these reference variables is persistently stored in the function block and written back to the reference variable following a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>.

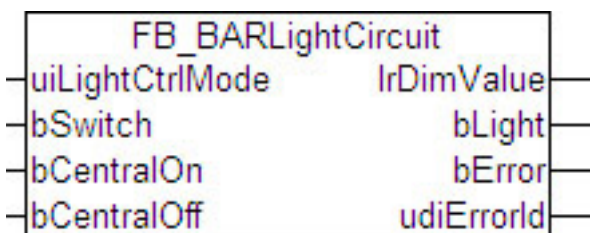
```
uiSecDimTime : UINT;
arrLightScenes : ARRAY[0..20] OF LREAL;
```

uiSecDimTime : ramp time in seconds. This is the time that the light actuator function requires in order to regulate from 0..100%.

arrLightScenes : table of saved light values.

3.3.3.6 FB_BARLightCircuit

This function block represents a simple light switching circuit without a dimming function.



The function block knows 3 different modes, which can be set via the input *uiLightCtrlMode*:

- Automatic mode
- Manual On mode
- Manual Off mode

In automatic mode (*uiLightCtrlMode*=0) the function block can be controlled via the inputs *bSwitch*, *bCentralOff* and *bCentralOn*. A rising edge on *bCentralOff* switches the output *IrDimValue* to 0.0, while a rising edge on *bCentralOn* sets the output to 100.0. Rising edges on *bSwitch* make the output *IrDimValue* change each time between 0.0 and 100.0.

In manual operation modes, when *uiLightCtrlMode*=1, the output value *IrDimValueOut* is constantly set to the value 100.0 and when *uiLightCtrlMode*=2, the output value is constantly set to the value 0.0.

VAR_INPUT

```
uiLightCtrlMode: UINT;
bSwitch         : BOOL;
bCentralOn     : BOOL;
bCentralOff    : BOOL;
```

uiLightCtrlMode : operation mode.

- 0: automatic mode, the output value *IrDimValue* can be influenced by the command inputs *bSwitch*, *bCentralOn* and *bCentralOff*.
- 1: Manual On mode; the value 100.0 is output without a delay at the output *IrDimValueOut*, all other command inputs are ineffective.
- 2: Manual Off mode; the value 0.0 is output without a delay at the output *IrDimValueOut*, all other command inputs are ineffective.

bSwitch: rising edges on *bSwitch* make the output *IrDimValue* change each time between 0.0 and 100.0.

bCentralOn: switches the output *IrDimValueOut* to 100.0.

bCentralOff: switches the output *IrDimValueOut* to 0.0.



All switching commands, *bSwitch*, *bCentralOn* and *bCentralOff* are effective only in automatic mode.

VAR_OUTPUT

```
IrDimValue     : LREAL;
bLight         : BOOL;
bError         : BOOL;
udiErrorId     : UDINT;
```

IrDimValue : light output value in percent 0.0, if the light is switched off and 100.0 if the light is switched on.

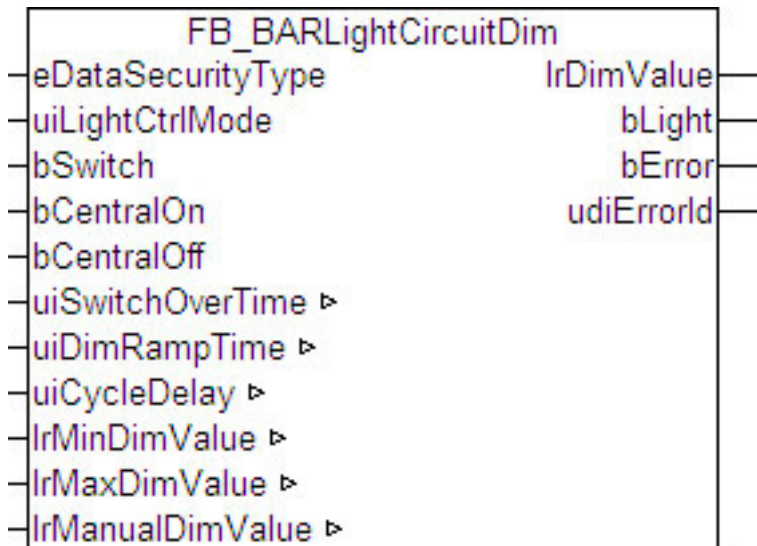
bLight : FALSE, if *IrDimValue* = 0.0, otherwise TRUE.

bError: this output is switched to TRUE if the parameters entered are erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes \[► 237\]](#).

3.3.3.7 FB_BARLightCircuitDim

This function block represents a light switching circuit with a dimming function.

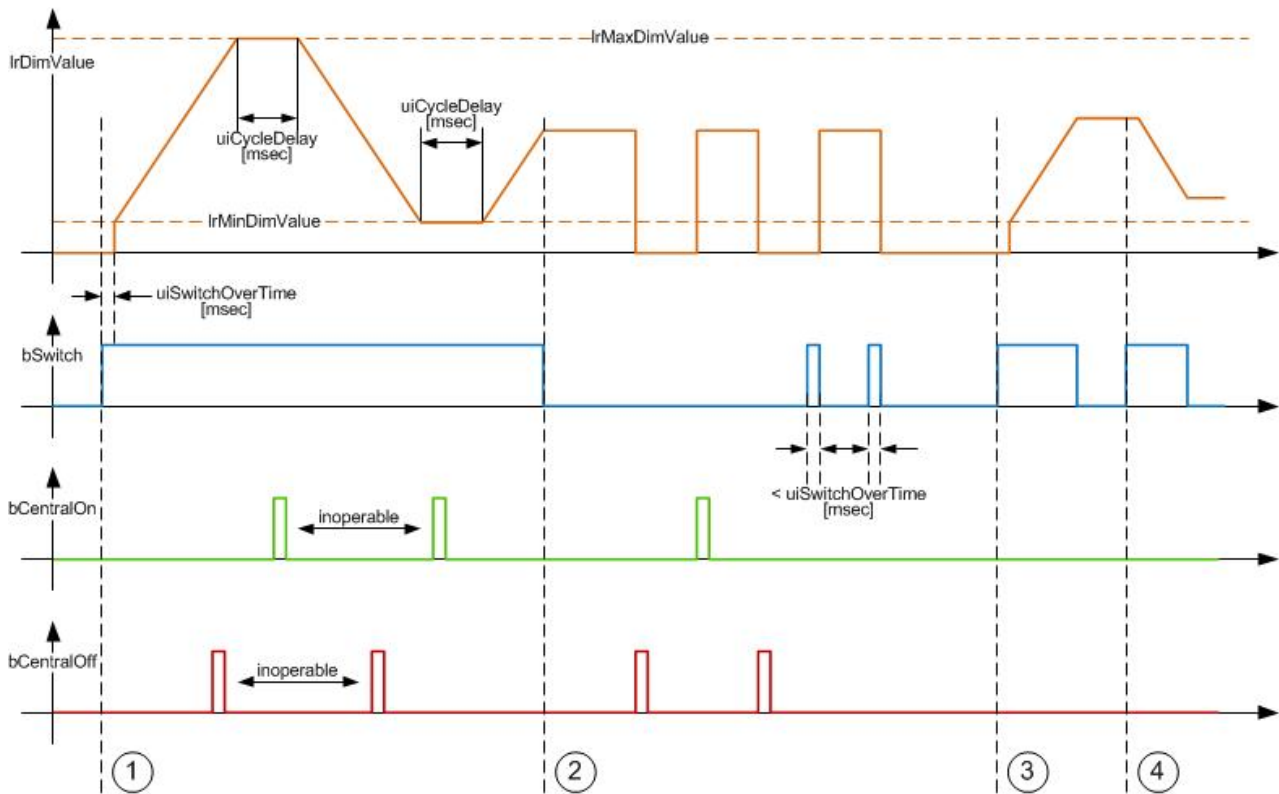


The function block knows 3 different modes, which can be set via the input *uiLightCtrlMode*:

- Automatic mode
- Manual On mode
- Manual Off mode

In automatic mode (*uiLightCtrlMode*=0) the function block can be controlled via the inputs *bSwitch*, *bCentralOff* and *bCentralOn*. Whereas the output *lrDimValue* is set to 0.0 or to the value reached before the last switch-off by rising edges on *bCentralOff* and *bCentralOn*, the behavior of the input *bSwitch* depends on the duration of the signal. A short TRUE signal that is shorter than *uiSwitchOverTime* in milliseconds switches the output *lrDimValue*. If the output *lrDimValue* is initially larger than 0.0, then it is switched to 0 and the previous value is saved. If on the other hand it is 0.0, then the output is set to the previously internally saved value. This saved value is set to the maximum value in the switch-on state of the program, see below. Unlike in the case of the inputs *bCentralOff* and *bCentralOn*, the falling edge triggers the switching event here. A long TRUE signal on *bSwitch* starts the dimming process. In principle dimming takes place only between the set minimum and maximum values (*lrMinDimValue* and *lrMaxDimValue*). The dimming ramp is defined in seconds by *uiRampTime*. This dimming time refers to the dimming range from 0 to 100 per cent, even if the limits *lrMinDimValue* and *lrMaxDimValue* are set differently. If the output value *lrDimValue* reaches one of the limits without *bSwitch* resetting to FALSE, then it remains there for the time *uiCycleDelay* in milliseconds, before dimming takes place again to the other limit value.

Exception: a *uiCycleDelay* value of 0 is not interpreted as an immediate dimming direction reversal, but instead deactivates this. Only another long TRUE signal on *bSwitch* starts the dimming in the opposite direction.



- ① **Dimming:**
With a TRUE-signal at the input *bSwitch* longer than *uiSwitchOverTime*, the output *IrDimmValue* will be initially set to *IrMinValue*. Afterwards the dimming-cycle begins and *IrDimmValue* will be continuously increased until it reaches the maximum value *IrMaxValue*. With *bSwitch* still set to TRUE it will remain at this level for *uiCycleDelay* in milliseconds before the output is decreased. When *IrDimmValue* reaches *IrMinValue*, it will remain there again for *uiCycleDelay* in milliseconds. Then the dimming cycle starts again. A falling edge at *bSwitch* stops the dimming cycle. During the whole dimming-cycle, which means, that *bSwitch* is set to TRUE, the inputs *bCentralOn* and *bCentralOff* will be inoperable.
- ② **Switching:**
Rising edges at *bCentralOff* and *bCentralOn* will switch the output *IrDimValue* to 0 respectively to the last saved value. Rising edges at *bSwitch* will also toggle the output *IrDimValue* between 0 and the last saved value, but only, if the signal is shorter than *uiSwitchOverTime* in milliseconds. Otherwise the dimming-cycle will begin.
- ③ Another long-signal at *bSwitch* lets the dimming-cycle start again. Because the light was previously switched off (*IrDimmValue*=0.0) the output *IrDimmValue* will be set to the minimum-value first, before the cycle begins. The dimming will be stopped with a falling edge at *bSwitch*.
- ④ If the dimming is started again, the cycle will begin with the last saved value. The dim-direction will be inverted.


In manual operation modes, when *uiLightCtrlMode*=1, the output value *IrDimValueOut* is constantly set to the value *IrManualDimValue* and when *uiLightCtrlMode*=2, the output value is constantly set to 0.0.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
uiLightCtrlMode   : UNT;
bSwitch           : BOOL;
bCentralOn        : BOOL;
bCentralOff       : BOOL;
```

eDataSecurityType: if *eDataSecurityType*:= *eDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not internally released.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

uiLightCtrlMode : operation mode.

- 0: automatic mode, the output value *IrDimValue* can be influenced by the command inputs *bSwitch*, *bCentralOn* and *bCentralOff*
- 1: Manual On mode; the value *IrManualDimValue* is output without a delay at the output *IrDimValueOut*, all other command inputs are ineffective.
- 2: Manual Off mode; the value 0.0 is output without a delay at the output *IrDimValueOut*, all other command inputs are ineffective.

bSwitch: a short TRUE signal that is shorter than *uiSwitchOverTime* in milliseconds switches the output *IrDimValue*. If the output *IrDimValue* is initially larger than 0.0, then it is switched to 0 and the previous value is saved. If on the other hand it is 0.0, then the output is set to the previously internally saved value. This saved value is set to the maximum value in the switch-on state of the program, see below.

A long TRUE signal that is longer than *uiSwitchOverTime* in milliseconds starts the dimming process, wherein the dimming direction changes if two long signals follow one another on the input *bSwitch*.

bCentralOn: switches the output *IrDimValueOut* to the previously saved (at the last switch-off) dimming value, see *bSwitch*.

bCentralOff: switches the output *IrDimValueOut* to 0.0.



All switching commands, *bSwitch*, *bCentralOn* and *bCentralOff* are effective only in automatic mode.

VAR_OUTPUT

```
IrDimValue : LREAL;
bLight     : BOOL;
bError     : BOOL;
udiErrorId : UDINT;
```

IrDimValue : light output value; can adopt values between *IrMinDimValue* and *IrMaxDimValue*, see VAR_IN_OUT, but maximally 0..100%.

bLight : FALSE, if *IrDimValue* = 0.0, otherwise TRUE.


bError: this output is switched to TRUE if the parameters entered are erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes](#) [▶ 237].

VAR_IN_OUT

In order for the registered parameters to be retained beyond a controller failure, it is necessary to declare them as In-Out variables. A reference variable is then assigned to them in the program. Each change of the value of these reference variables is persistently stored in the function block and written back to the reference variable following a controller failure and restart. If the parameters were only declared as input

variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>.

```
uiSwitchOverTime : UINT;
uiDimRampTime    : UINT;
uiCycleDelay     : UINT;
lrMinDimValue    : LREAL;
lrMaxDimValue    : LREAL;
lrManualDimValue : LREAL;
```

uiSwitchOverTime : switching time in milliseconds from button to dimming mode.

uiDimRampTime : ramp time in seconds (time required to dim from 0% to 100%).

uiCycleDelay : switching time in milliseconds of the automatic change between dimming up and dimming down. The automatic change is inactive if this value is zero.

lrDimMinValue : minimum light value in % that can be reached by dimming.

lrDimMaxValue : maximum light value in %.

lrManualDimValue : output dimming value in Manual On mode (`uiLightCtrlMode=1`).

3.3.3.8 FB_BARStairwellAutomatic

Function block for a stairwell light circuit.

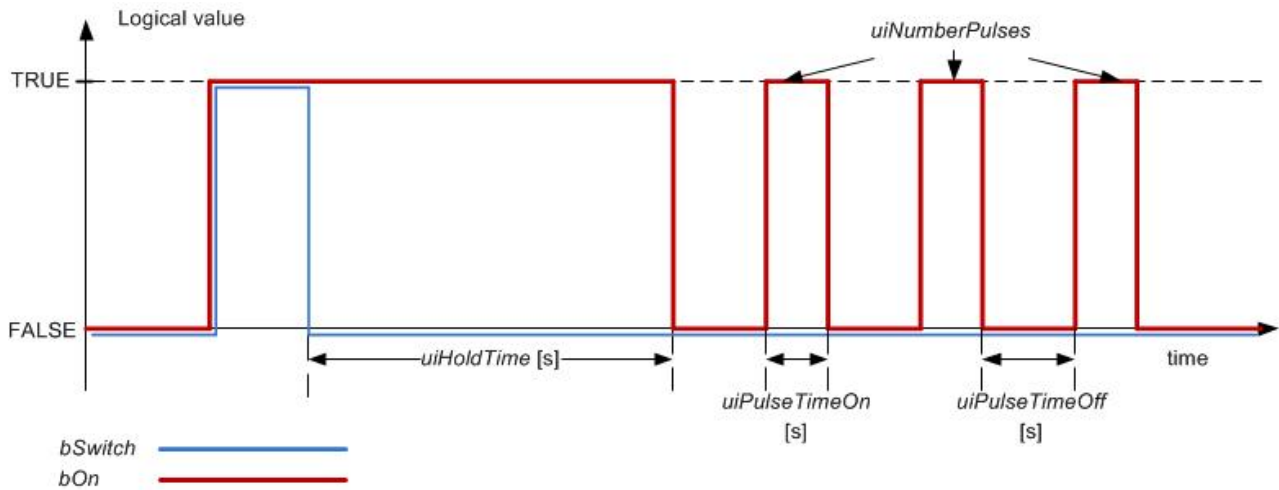


Fig. 5: FB_BARStairwellAutomatic

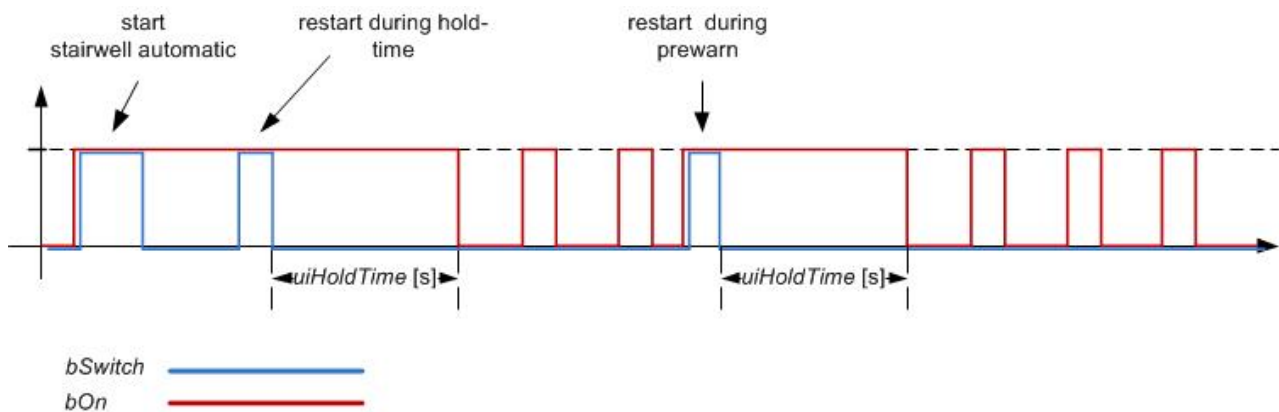
The function block knows 3 different modes, which can be set via the input `uiLightCtrlMode`:

- Automatic mode
- Manual On mode
- Manual Off mode

The stairwell control is active in automatic mode (`uiLightCtrlMode=0`). A positive edge on `bSwitch` initially only switches the light on (output `bOn`). A negative edge starts the holding time generator. If the hold time `uiHoldTime` [s] has expired, a flash sequence with a number of flash pulses programmable by `uiNumberPulses` begins as a warning of the impending switch-off. These pulses have an on-time of `uiPulseTimeOn` [ms] and an off-time of `uiPulseTimeOff` [ms].



The sequence can be restarted at any time:



In the manual operation modes the input *bSwitch* has no function: with *uiLightCtrlMode*=1 the output value *bOn* is constantly set to TRUE and with *uiLightCtrlMode*=2 it is constantly set to FALSE.


Changing to manual mode resets any lighting sequence that had started up to that point.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
uiLightCtrlMode   : UINT;
bSwitch           : BOOL;
```

eDataSecurityType: if *eDataSecurityType*:= *eDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instantiated once in the main program, which is called cyclically. Otherwise the instanced FB is not internally released.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If *eDataSecurityType*:= *eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if *eDataSecurityType*:= *eHVACDataSecurityType_Persistent*. It would lead to early wear of the flash memory.

uiLightCtrlMode : operation mode.

- 0: automatic mode; the stairwell circuit is active and reacts to the input *bSwitch*.
- 1: Manual On mode; the stairwell circuit is inactive - the output *bOn* is set constantly to TRUE.
- 2: Manual Off mode; the stairwell circuit is inactive - the output *bOn* is set constantly to FALSE.

bSwitch: a rising edge switches the light on in automatic mode (*uiLightCtrlMode*=0), a falling edge starts the holding time generator. This input has no function in manual operation mode (*uiLightCtrlMode*=1 or 2).

VAR_OUTPUT

```
lrDimValue      : LREAL;
bOn             : BOOL;
uiRemainingHoldTime: UINT;
bError         : BOOL;
udiErrorId     : UDINT;
```

lrDimValue : in order to keep this function the same as the other light user functions, a light output value exists here too in the form of a floating point number in percent, even though the light is only switched on or off. This means: light off: *lrDimValue* = 0.0, light on: *lrDimValue* = 100.0.

bOn : switching output for the lighting.

uiRemainingTimeHold: remaining hold time in seconds. If the light is off or if manual operation mode is active, then this output is "0". With a rising edge on *bSwitch* in automatic mode, this output initially indicates the complete number of seconds of the hold time (*uiHoldTime*), in order to illustrate the countdown of the hold time, starting with a falling edge on *bSwitch*. This output is 0 as long as no countdown of the time is taking place.

bError: this output is switched to TRUE if the parameters entered are erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes](#) [► 237].

VAR_IN_OUT

In order for the registered parameters to be retained beyond a controller failure, it is necessary to declare them as In-Out variables. A reference variable is then assigned to them in the program. Each change of the value of these reference variables is persistently stored in the function block and written back to the reference variable following a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

```
uiHoldTime      : UINT;
uiPulseTimeOn   : UINT;
uiPulseTimeOff  : UINT;
uiNumberPulses  : UINT;
```

uiHoldTime : hold time [s] of the stairwell controller after a falling edge on *bSwitch*.

uiPulseTimeOn: ON-time of the early warning pulses in milliseconds.

uiPulseTimeOff: OFF-time of the early warning pulses in milliseconds.

uiNumberPulses: number of early warning pulses.

3.3.3.9 FB_BARTwilightAutomatic

Twilight automatic

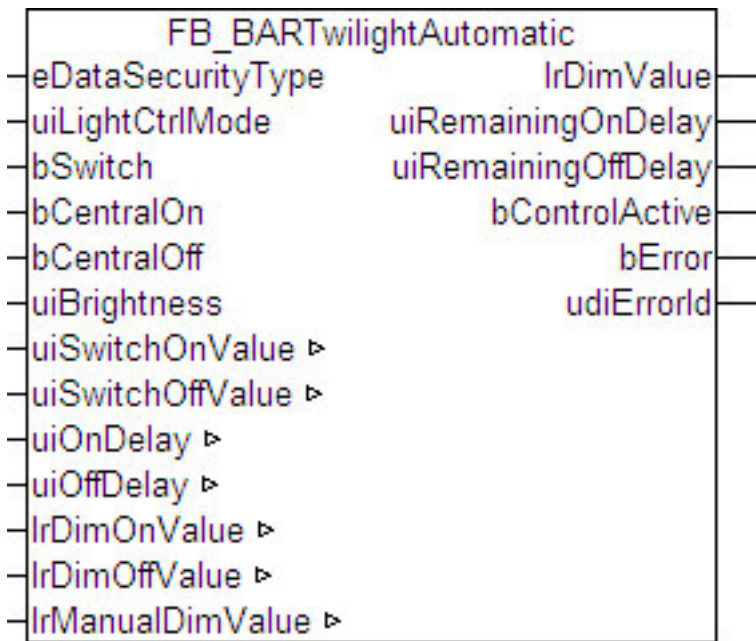


Fig. 6: FB_BARTwilightAutomatic

The function block knows 3 different modes, which can be set via the input *uiLightCtrlMode*:

- Automatic mode
- Manual On mode
- Manual Off mode

Automatic mode

In automatic mode a positive edge on *bSwitch* activates the twilight automatic, if it had been inactive beforehand. A further edge on *bSwitch* deactivates the twilight automatic again. The automatic function is explicitly switched on and off with *bCentralOn* and *bCentralOff*, independent of the previous state of the automatic function. If the twilight automatic is activated, then the function block switches the output *lrDimValue* to the switch-on value *lrDimValueOn* when the brightness falls below a switch-on threshold value *uiSwitchOnValue* for an entered delay time *uiOnDelay*. Conversely, if a switch-off threshold value *uiSwitchOffValue* is exceeded for an entered time delay *uiOffDelay*, then the output is switched to the value *lrDimValueOff*. In the inactive state the light output value *lrDimValue* is set to 0.0.

Manual mode

In the manual operation modes the inputs *bSwitch*, *bCentralOn* and *bCentralOff* have no function: with *uiLightCtrlMode*=1 the output value *lrDimValue* is constantly set to *lrManualDimValue* and with *uiLightCtrlMode*=2 it is constantly set to 0.0.

Switch-on and switch-off delay


The switch-on and switch-off delays written in automatic mode are always run through irrespective of the state of the automatic function (active or inactive) and the operating mode, i.e. the timers are not reset by these operating states.

VAR_INPUT

```
eDataSecurityTyp: E_HVACDataSecurityType;
uiLightCtrlMode : UINT;
bSwitch         : BOOL;
bCentralOn      : BOOL;
bCentralOff     : BOOL;
uiBrightness    : UINT;
```

eDataSecurityType: if `eDataSecurityType:= eDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not internally released.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

uiLightCtrlMode : operation mode.

- 0: Automatic mode; the commands `bSwitch`, `bCentralOn` and `bCentralOff` are executable and can be used to switch the twilight automatic on and off.
- 1: Manual On mode; twilight automatic is inactive – the value `IrManualDimValue` is output without a delay at the output `IrDimValueOut`, all other command inputs are ineffective.
- 2: Manual Off mode; twilight automatic is inactive – the value 0.0 is output without a delay at the output `IrDimValueOut`, all other command inputs are ineffective.

bSwitch : positive edges on this input switch the twilight automatic on and off alternately. When switching off the output `IrDimValue` is set to 0.0. This command input is active only in automatic mode (`uiLightCtrlMode=0`).

bCentralOn : a positive edge on this input switches the twilight automatic on. This command input is active only in automatic mode (`uiLightCtrlMode=0`).

bCentralOff : a positive edge on this input switches the twilight automatic off and sets the output `IrDimValue` to 0.0. This command input is active only in automatic mode (`uiLightCtrlMode=0`).

uiBrightness : actual light value [lux].

VAR_OUTPUT

```
IrDimValue      : LREAL;
uiRemainingOnDelay : UINT;
uiRemainingOffDelay: UINT;
bControlActive  : BOOL;
bError          : BOOL;
udiErrorId      : UDINT;
```

IrDimValue : light output value, 0..100%.

uiRemainingOnDelay : countdown of the switch-on delay in seconds. This output is 0 as long as no countdown of the time is taking place.

uiRemainingOffDelay : countdown of the switch-off delay in seconds. This output is 0 as long as no countdown of the time is taking place.

bControlActive : this output is TRUE if the function block is in automatic mode and the twilight automatic is activated. This is intended to serve as an additional feedback signal if switch-on has taken place but the control outputs a light value of `IrDimValue=0.0`.

bError: this output is switched to TRUE if the parameters entered are erroneous.


udiErrorId : contains the error code if the values entered should be erroneous. See [error codes](#) [▶ 237].



Note If an error should occur, then this automatic function is deactivated and position and angle are set to 0. This means that if a priority controller is in use, another function with a lower priority (see Overview) automatically takes over control of the blind. In the case of a direct connection, conversely, the blind will drive to position/angle 0.

VAR_IN_OUT

In order for the registered parameters to be retained beyond a controller failure, it is necessary to declare them as In-Out variables. A reference variable is then assigned to them in the program. Each change of the value of these reference variables is persistently stored in the function block and written back to the reference variable following a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>.

```
uiSwitchOnValue : UINT;
uiSwitchOffValue : UINT;
uiOnDelay : UINT;
uiOffDelay : UINT;
lrDimOnValue : LREAL;
lrDimOffValue : LREAL;
lrManualDimValue : LREAL;
```

uiSwitchOnValue: switch-on threshold. This is compared with the brightness value at the input *uiBrightness*. This value must be greater than the output threshold value *uiSwitchOffValue*.

uiSwitchOffValue: switch-off threshold. This is compared with the brightness value at the input *uiBrightness*.

uiOnDelay: switch-on delay in seconds.



uiOffDelay: switch-off delay in seconds.

lrDimOnValue : switch-on light value in %.

lrDimOffValue : switch-off light value in %.

lrManualDimValue : output dimming value in Manual On mode (*uiLightCtrlMode*=1).

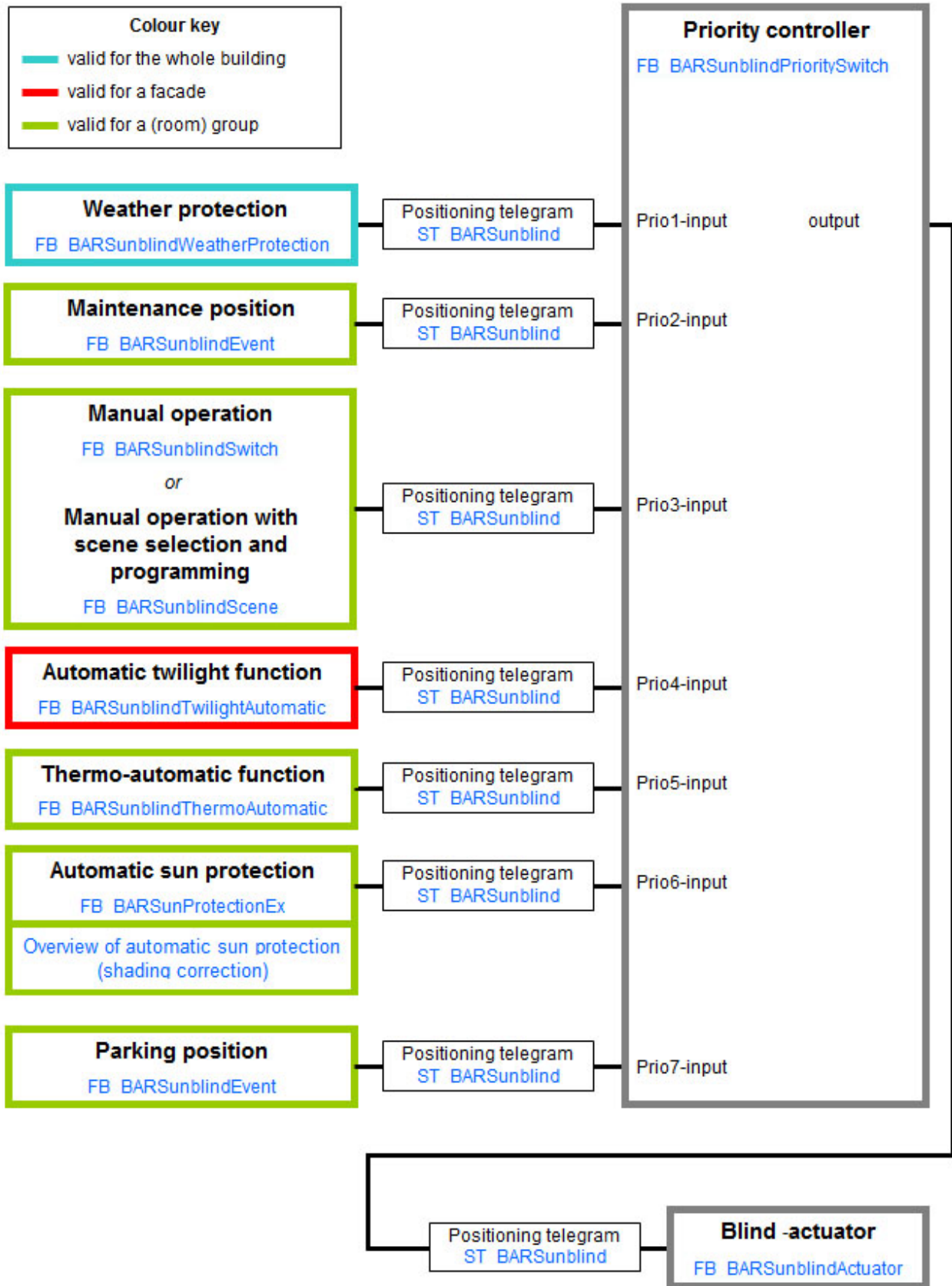
Documents about this

-  [example_persistent_e.zip \(Resources/zip/11659714827.zip\)](#)
-  [example_persistent_e.zip \(Resources/zip/11659714827.zip\)](#)

3.3.4 Sun portection

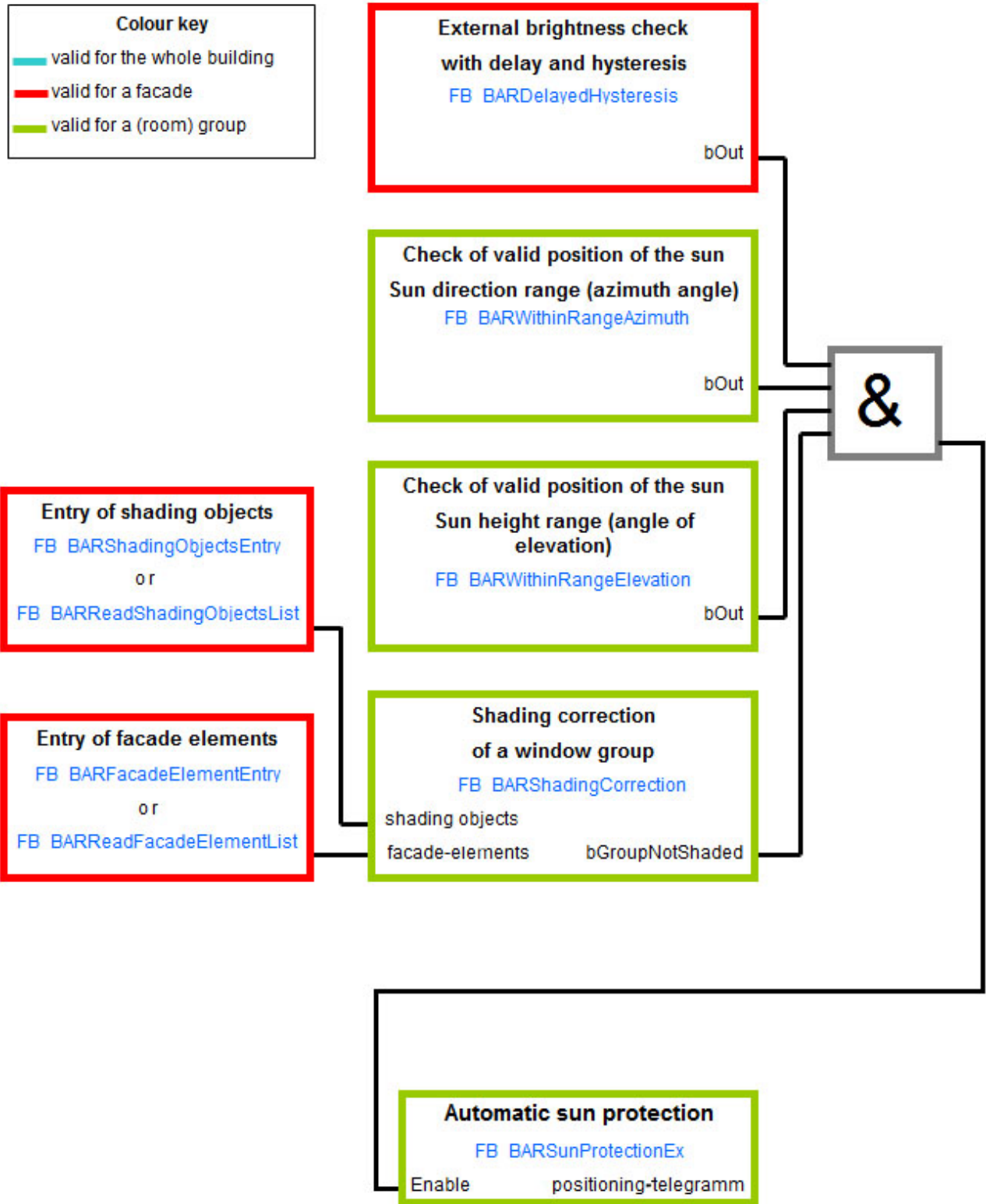
3.3.4.1 Overview shading

Shading - overview



3.3.4.2 Overview sun protection

Overview of automatic sun protection



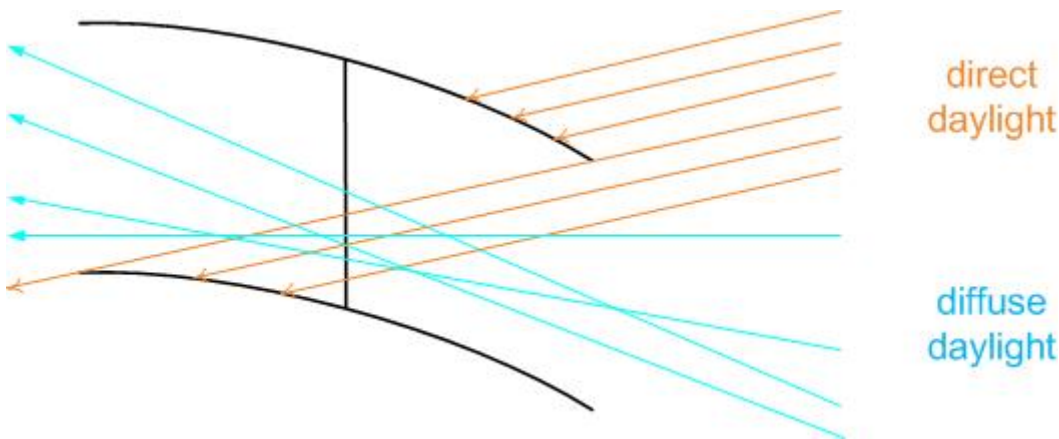
3.3.4.3 Sun protection: basic principles and definitions

The direct incidence of daylight is regarded as disturbing by persons in rooms. On the other hand, however, people perceive natural light to be more pleasant in comparison with artificial light. Two options for glare protection are to be presented here:

- Lamella setpoint tracing
- Height adjustment

Lamella setpoint tracing

A louvered blind that can be adjusted offers the option of intelligent sun protection here. The position of the slats is cyclically adapted to the current position of the sun, so that no direct daylight enters through the blinds, but as much diffuse daylight can be utilized as possible.



The illustration shows that diffuse light can still enter from underneath, whereas no further direct daylight, or theoretically only a single ray, can enter. The following parameters are necessary for the calculation of the slat angle:

- the current sun elevation (angle of elevation)
- the position of the sun, i.e. the azimuth angle
- the facade orientation
- the slat width
- the slat spacing

The effective angle of elevation is calculated from the first three of the parameters listed above:

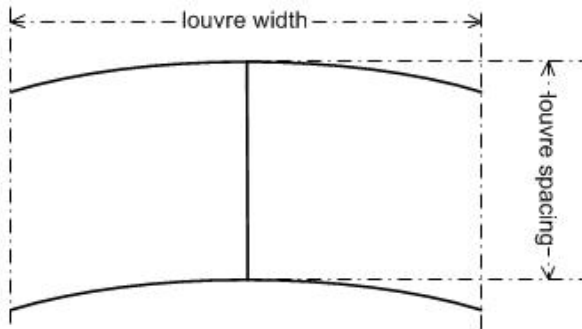
If the facade orientation and the position of the sun (azimuth) are equal, then the effective angle is equal to the current angle of elevation.

However, if the sunlight falls at an angle onto the facade as seen from the sun direction, the effective angle is larger for the same angle of elevation.

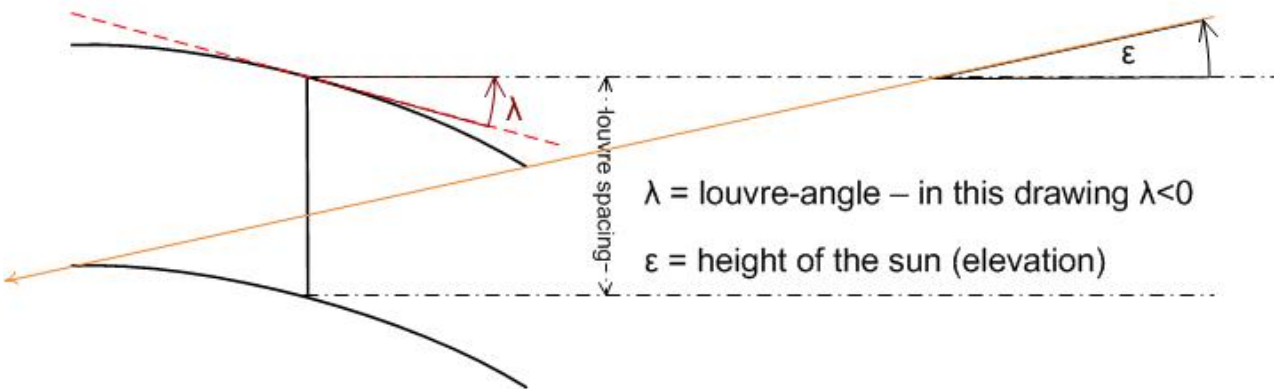
The following three pictures illustrate the relationship between the effective angle of elevation and the dimensions of the blind and how the resulting slat angle λ changes:

Lamellenwinkel

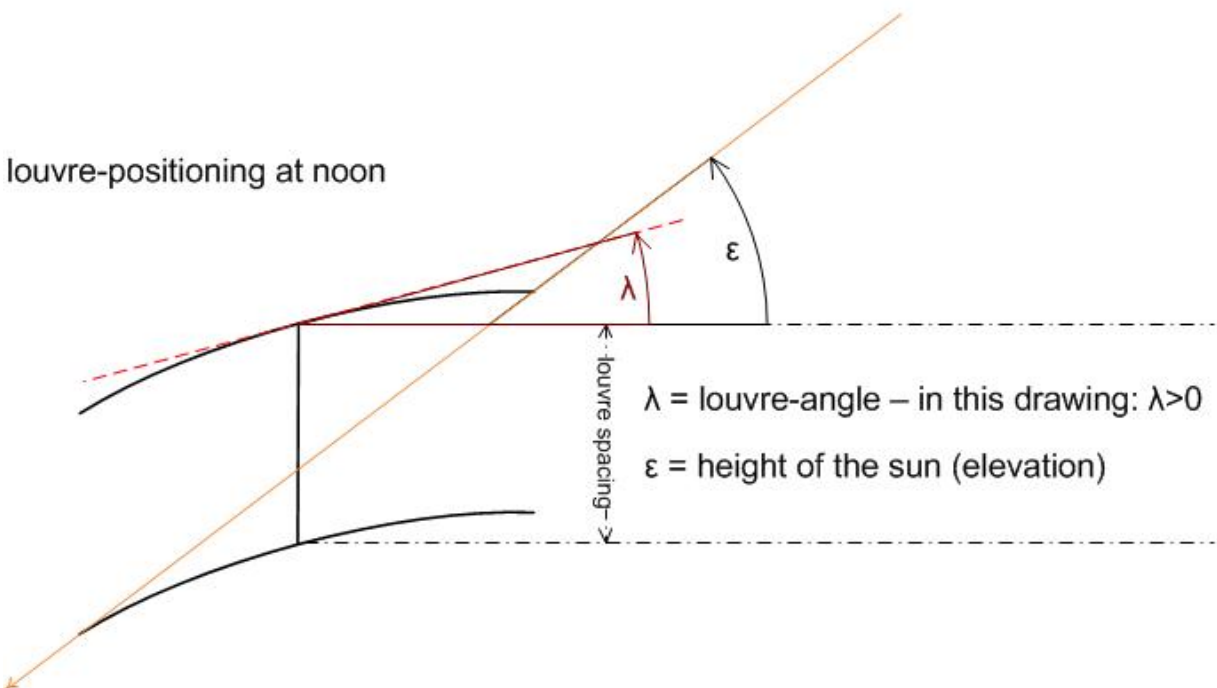
louvre at an angle of $\lambda=0$



louvre-positioning at in the morning/evening

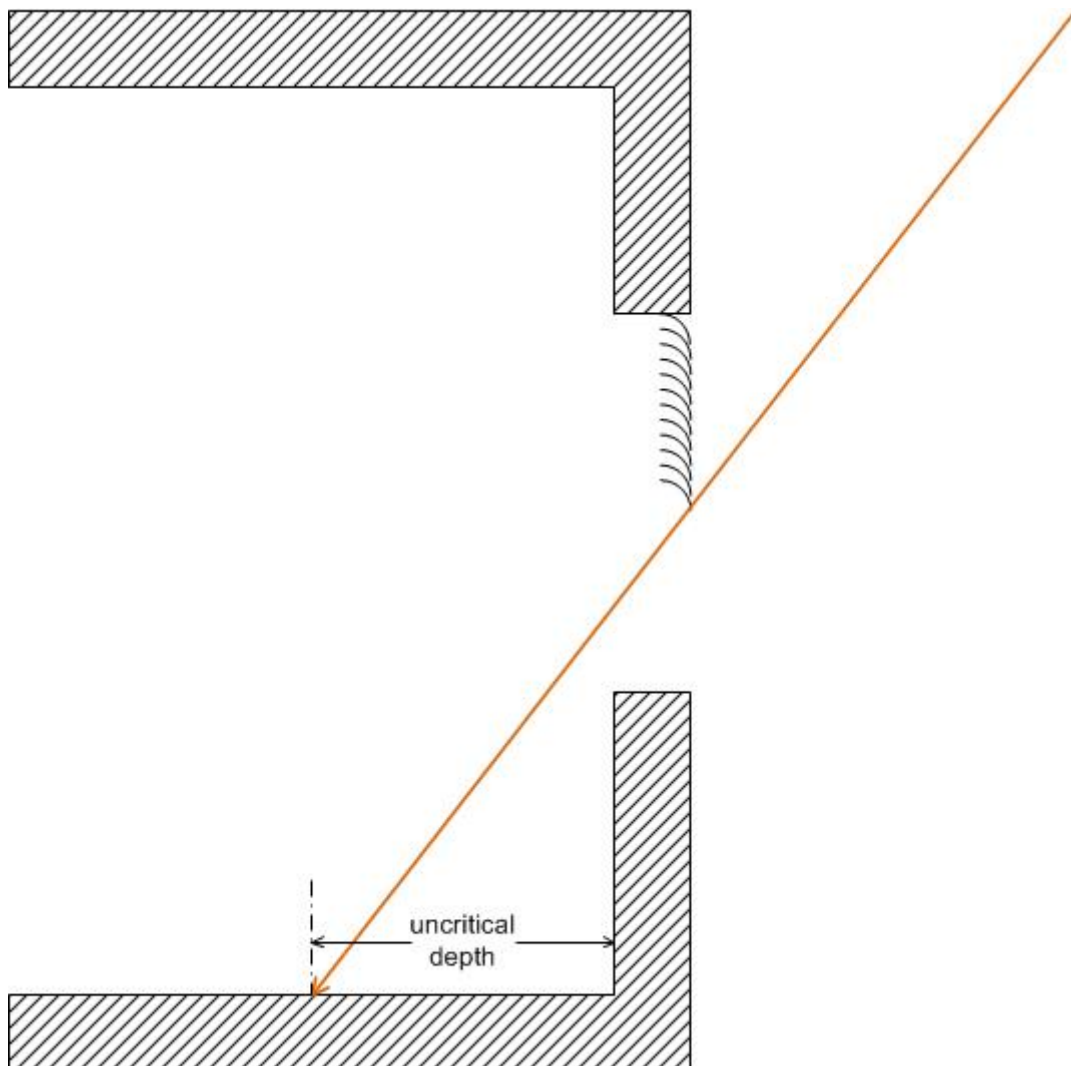


louvre-positioning at noon



Height adjustment

With a high position of the sun at midday, the direct rays of sunlight do not penetrate into the full depth of the room. If direct rays of sunlight in the area of the window sill are regarded as uncritical, the height of the sun protection can be adapted automatically in such a way that the rays of sunlight only ever penetrate into the room up to an uncritical depth.

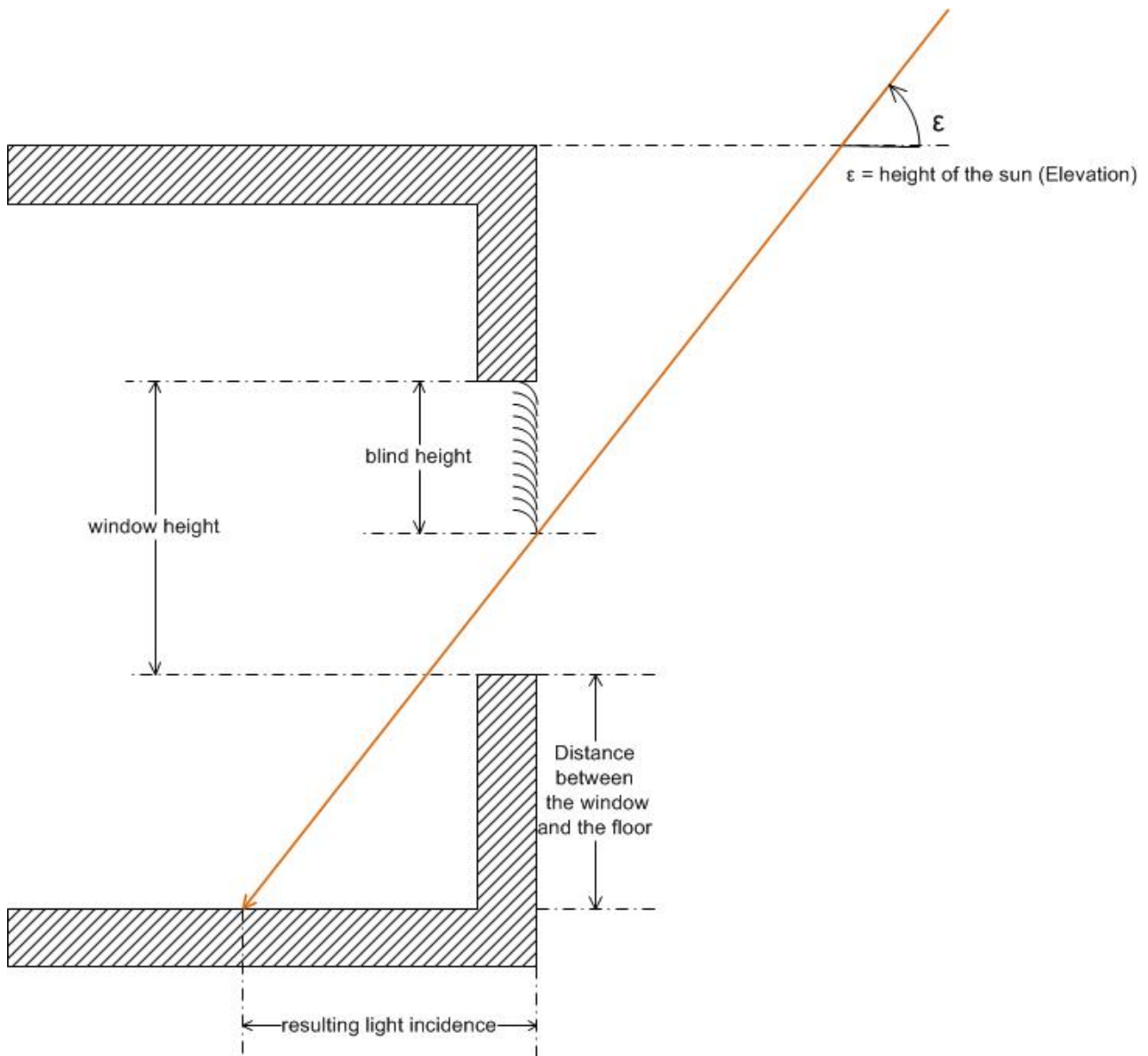


In order to be able to calculate at any time the appropriate blind height that guarantees that the incidence of sunlight does not exceed a certain value, the following values are necessary.

Required for the calculation of the respective blind height:

- Sun elevation
- Window height
- Distance between the window and the floor

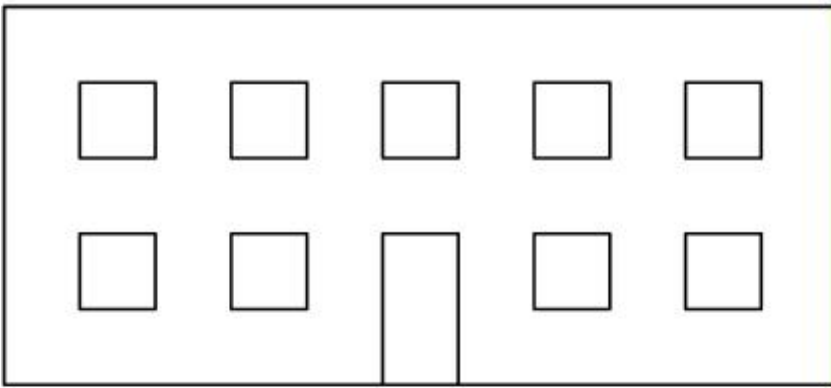
The following illustration shows where these parameters are to be classified:



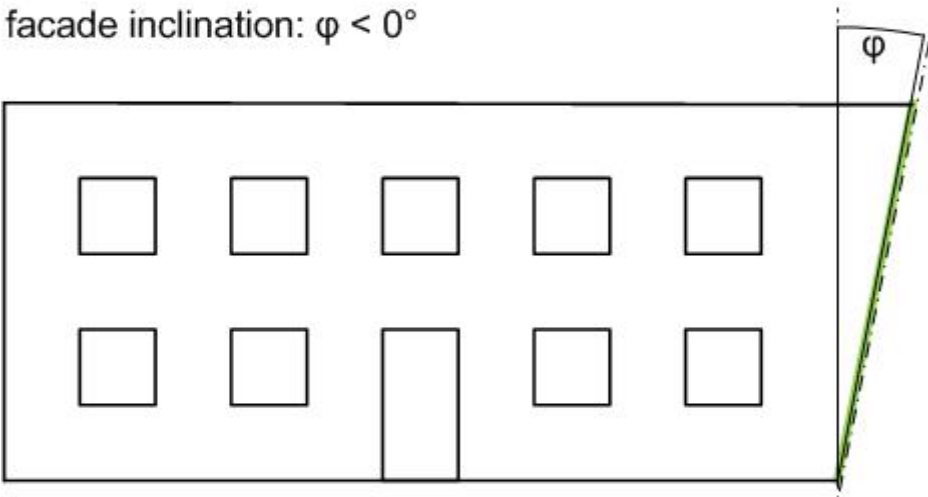
Influence of the facade inclination

In both of the methods of sun protection described, it was assumed that the facade and thus the windows are perpendicular to the ground. In the case of an inclined facade, however, the incidence of light changes such that this influence will also be taken into account. The facade inclination is defined as follows:

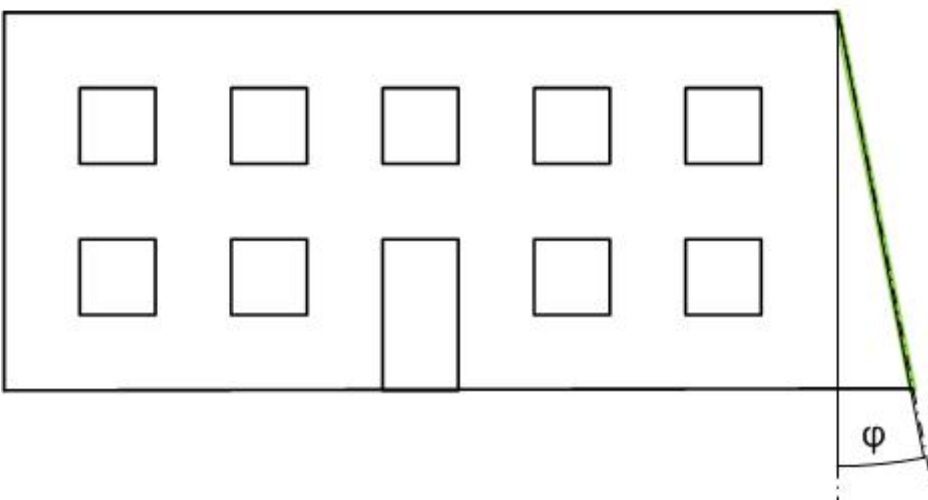
facade inclination: $\varphi = 0^\circ$



facade inclination: $\varphi < 0^\circ$



facade inclination: $\varphi > 0^\circ$



3.3.4.4 Shading correction: basic principles and definitions

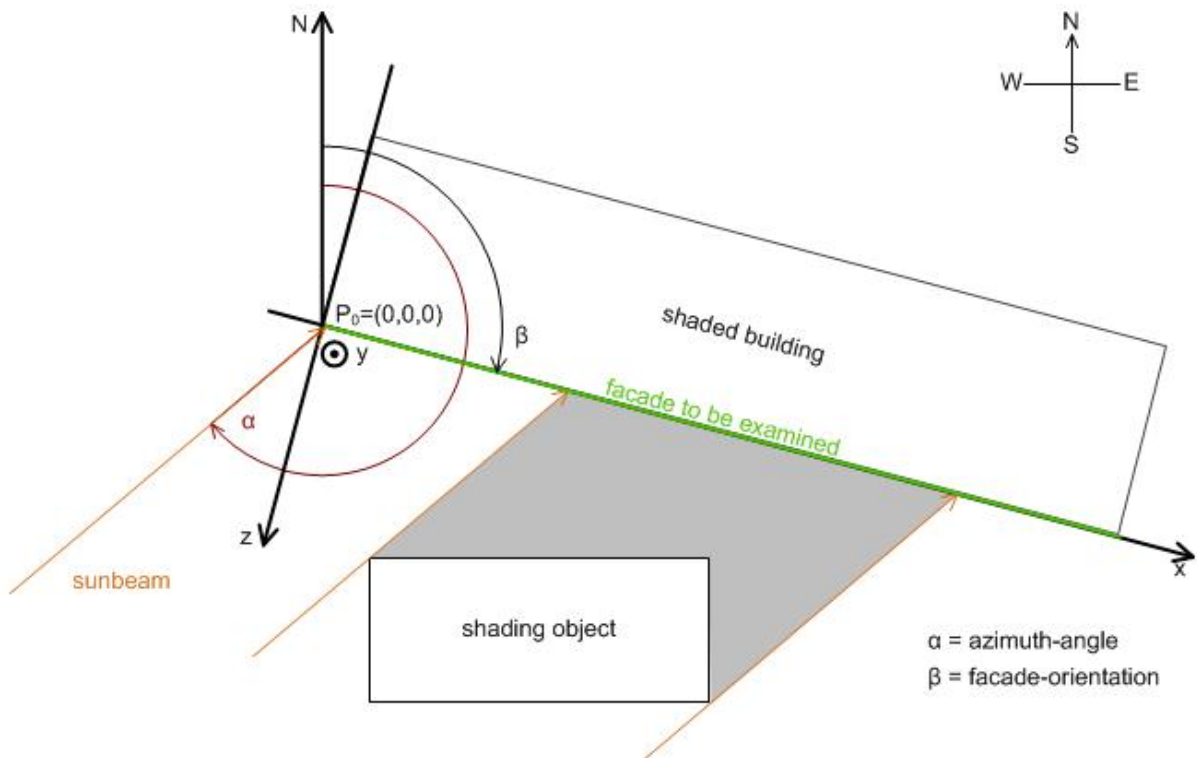
The shading correction can be used in conjunction with the automatic sun function or lamella setpoint tracing. The function checks whether a window or a window group that is assigned to a room, for example, is temporarily placed in the shade by surrounding buildings or parts of its own building. Sun shading for windows that stand in the shadow of surrounding buildings or trees is not necessary and may even be disturbing under certain circumstances. On the basis of data entered regarding the facade and its surroundings, the shading correction determines which parts of the front are in the shade. Hence, it is then possible to decide whether the sun protection should be active for individual windows or window groups. Apart from the current position of the sun, the shading of the individual windows depends on three things:

- the orientation of the facade
- the position of the windows
- the positioning of the shading objects

The following illustrations are intended to describe these interrelationships and to present the parameters to be entered.

Orientation of the facade

Observation from above



For the pure observation of the shadow thrown on the facade, a two-dimensional coordinate system is ultimately required, therefore the X and Y axis were placed on the facade. The zero point is thereby at the bottom left on the base, as if one were regarding the facade from the front. For the calculation of the shading objects the Z component is then also added. Its axis points from away the facade and has the same zero point as the X and Y axis.

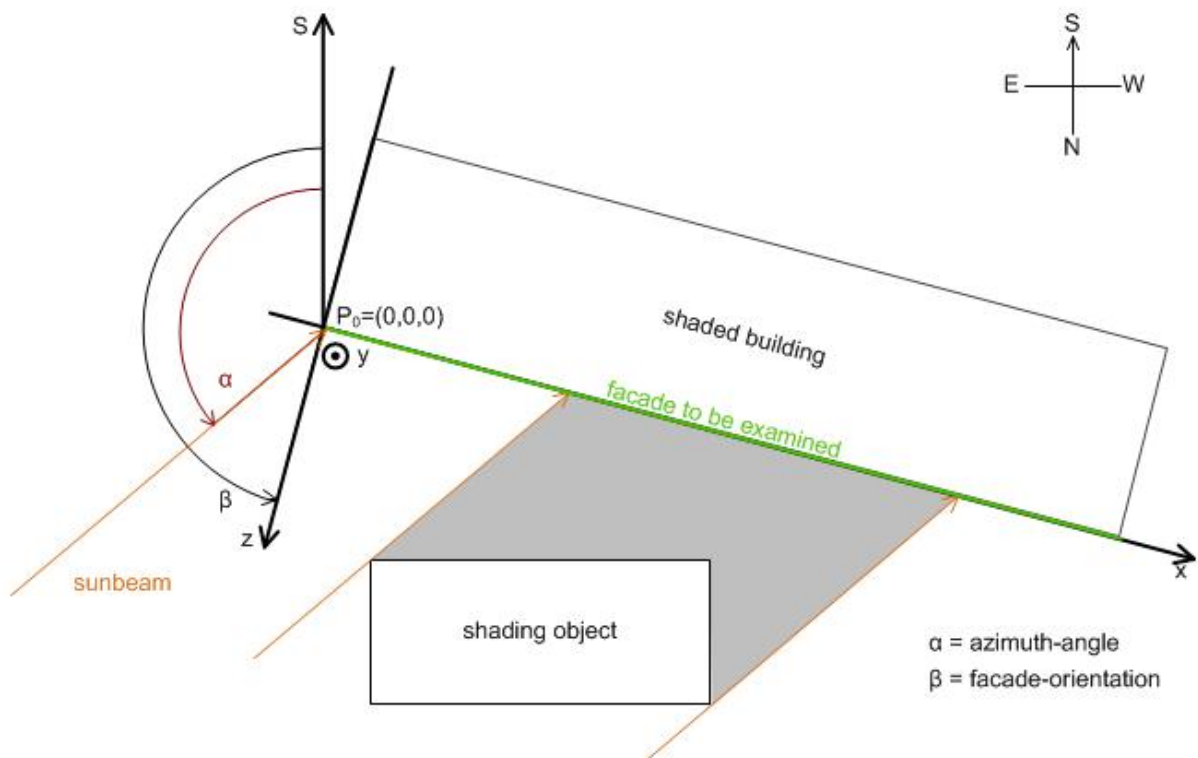
In the northern hemisphere, the horizontal sun position (azimuth angle) is determined from the north direction by definition. The facade orientation is likewise related to north, wherein the following applies to the line of sight from a window in the facade:

Line of sight	Facade orientation
North	$\beta=0^\circ$
East	$\beta=90^\circ$

Line of sight	Facade orientation
South	$\beta=180^\circ$
West	$\beta=270^\circ$

In the southern hemisphere the sun path is the other way round: although it also rises in the east, at midday it is in the north. The facade orientation is adjusted to this path:

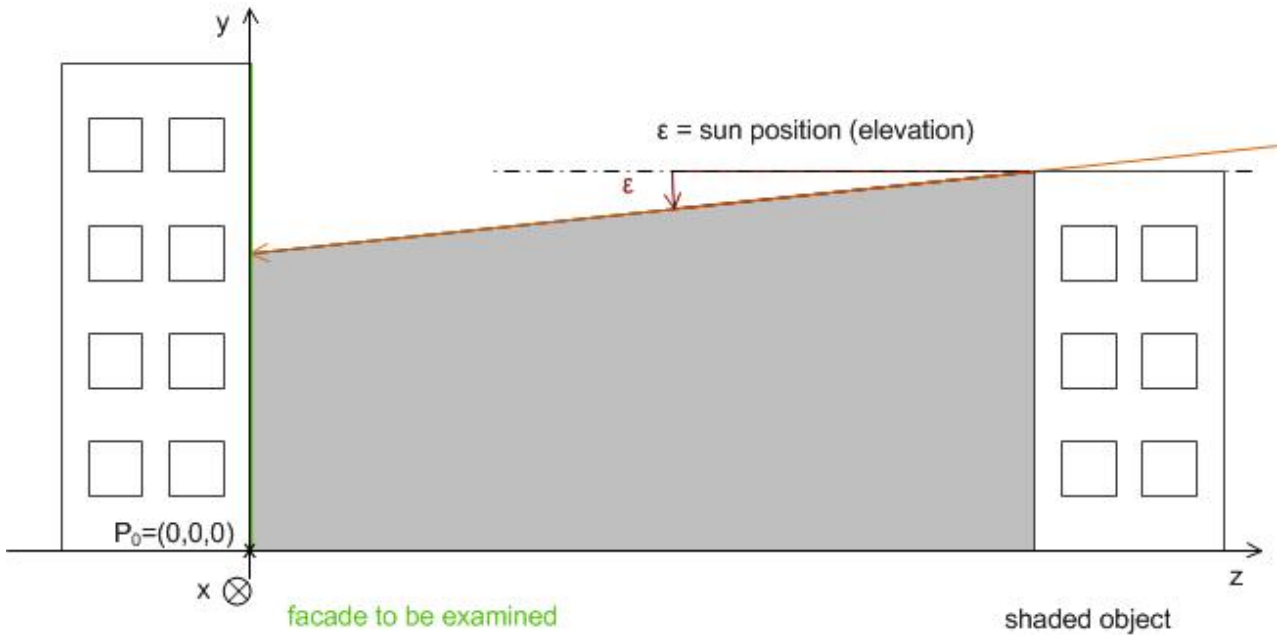
Line of sight	Facade orientation
South	$\beta=0^\circ$
East	$\beta=90^\circ$
North	$\beta=180^\circ$
West	$\beta=270^\circ$



For convenience, the other explanations refer to the northern hemisphere. For the later parameterization (FB_BARShadingCorrection [▶ 176] / FB_BARShadingCorrectionSouth [▶ 179]) only the corresponding facade orientation is necessary anyway, which can be taken from the respective valid table - northern or southern hemisphere.

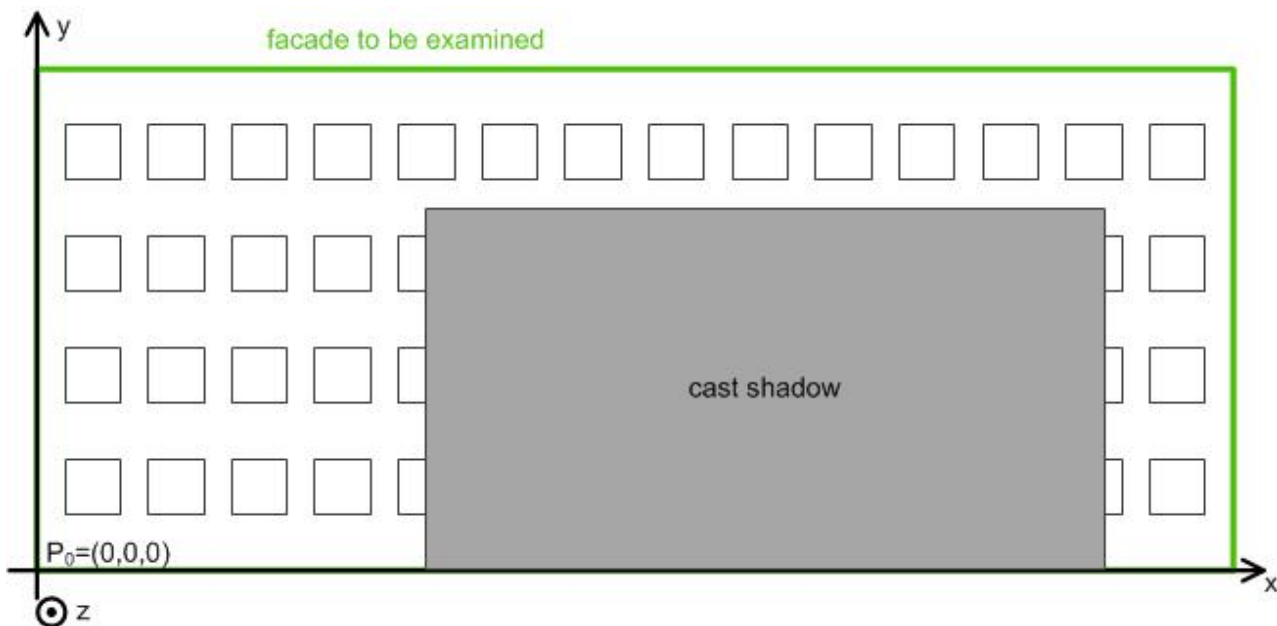
The following two illustrations are intended to further clarify the position of the point of origin P_0 as well as the orientation of the coordinate system:

Observation from the side



The sun elevation can be represented using this illustration: by definition this is 0° at sunrise (horizontal incidence of light) and can reach maximally 90° , but this applies only to places within the Tropic of Cancer and the Tropic of Capricorn.

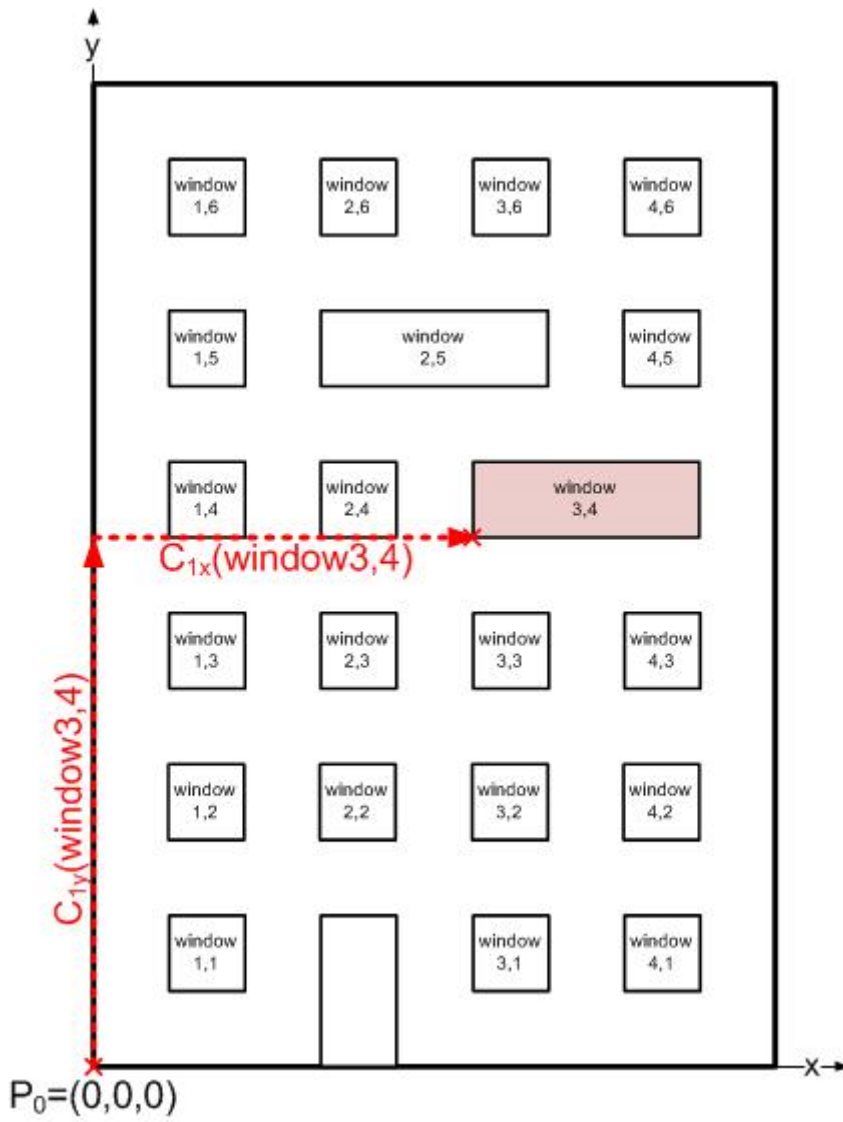
Observation from the front



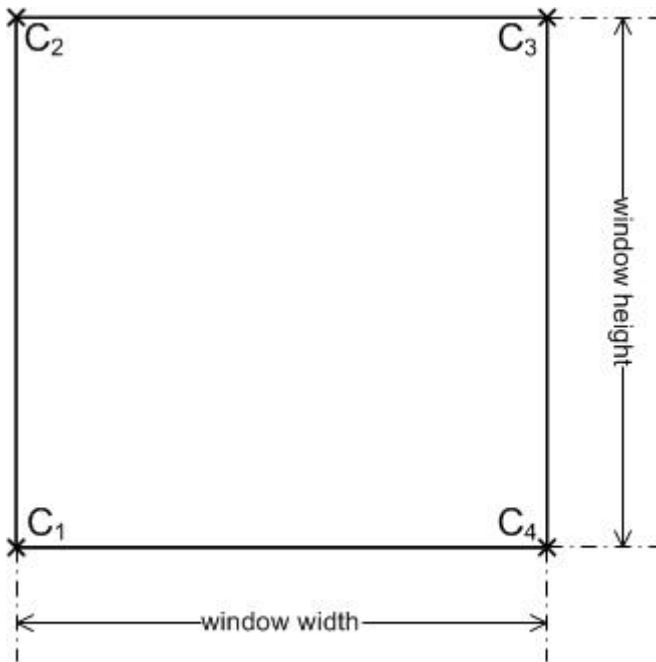
Here, the position of the point of origin, P_0 , at the bottom left base point of the facade is once more very clear. Beyond that the X-Y orientation is illustrated, which is important later for the entry of the window elements.

Position of the windows

The position of the windows is defined by the specification of their bottom left corner in relation to the facade coordinate system. Since a window lies flat on the facade, the entry is restricted to the X and Y coordinates.



The width and height must additionally be specified.



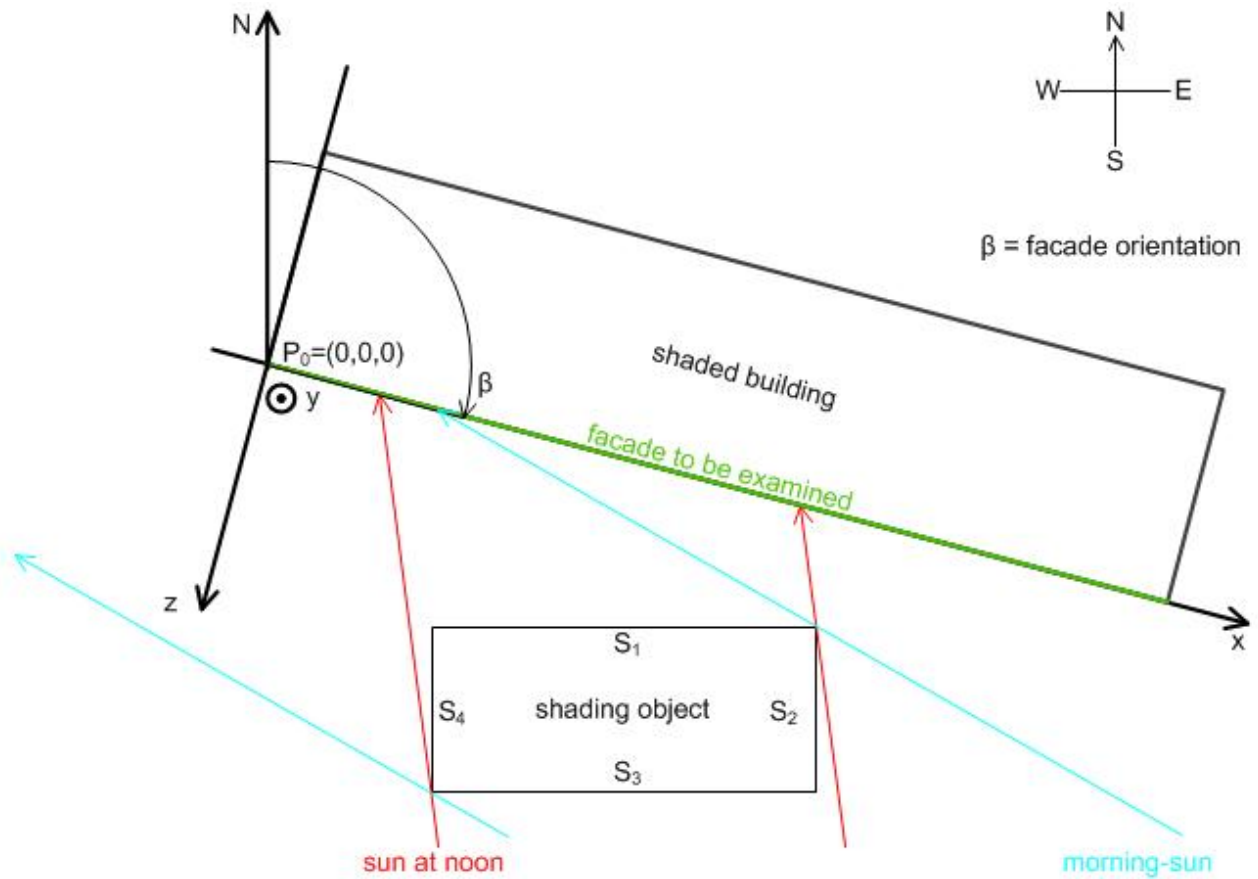
C_n =corner n

The position of each window corner on the facade is determined internally from the values entered. A window is considered to be in the shade if all corners lie in the shade.

Positioning of the shading objects

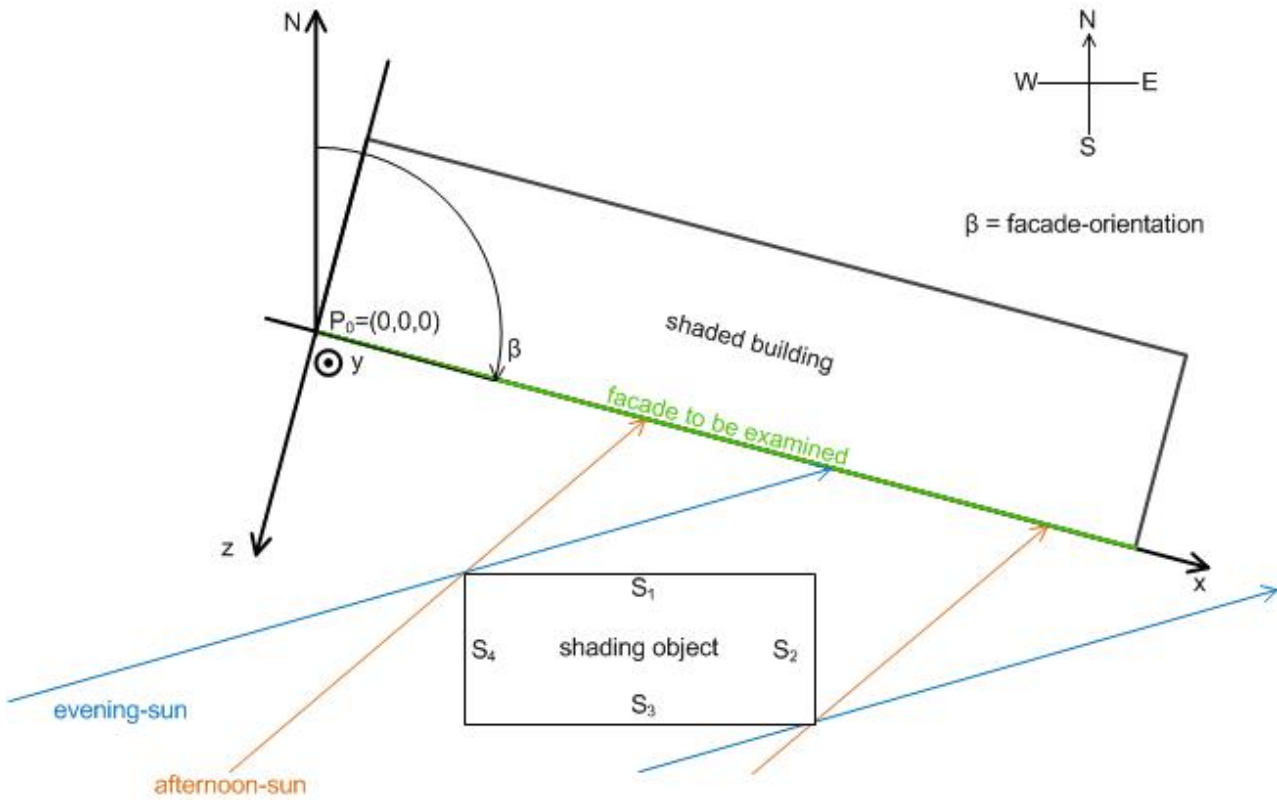
When describing the shading objects, distinction is made between angular objects (building, column) and objects that are approximately spherical (e.g. trees). Angular objects can be categorized according to the shadow they cast into square, shadow-casting facades, wherein one must consider which ones cast the main shadows over the course of the day:

Morning/noon

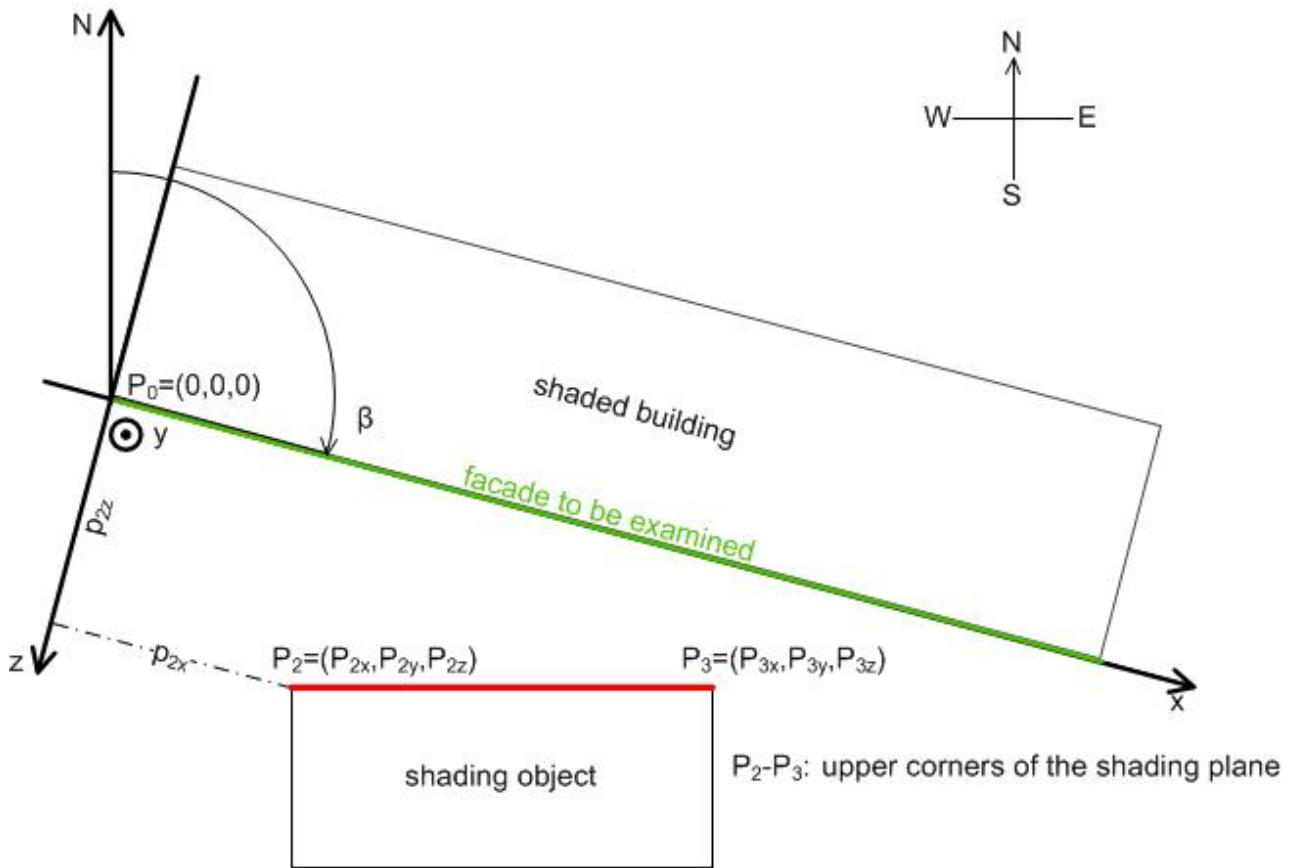


In the morning and around noon, the shadows are mainly cast by the sides S_1 and S_4 ; S_2 and S_3 need not be considered if they are not higher.

Afternoon/evening



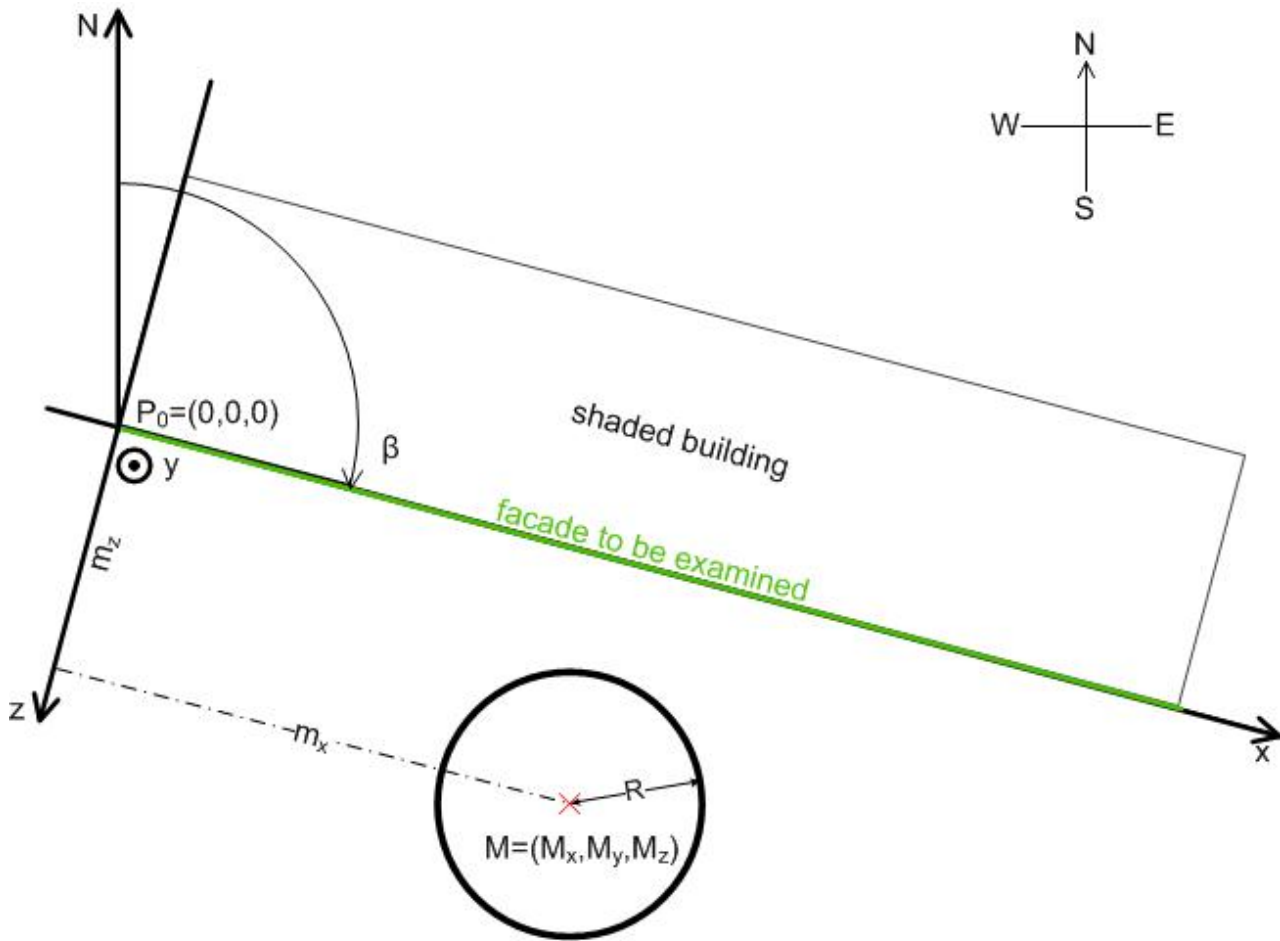
In the afternoon and in the evening too, the total shade can be determined alone by the observation of S_1 and S_3 . In this case it is therefore sufficient to specify S_1 and S_3 as shadow casters. The entry is made on the basis of the four corners or their coordinates in relation to the zero point of the facade:



In this sketch only the upper points, P_2 and P_3 , are illustrated due to the plan view. The lower point P_1 lies underneath P_2 and P_4 lies underneath P_3 .

The input of shadow-casting sphere elements is done by entering the center of the sphere and its radius:

Sphere elements



A "classification" of the sphere element as in the case of the angular building is of course unnecessary, since the shadow cast by a sphere changes only its direction, but not its size.

3.3.4.5 FB_BARBlindPositionEntry

This function block serves for the input of interpolation points for the function block [FB_BARSunProtectionEx](#) [► 221], if this should be operated in the height positioning mode with the help of a table, see [E_BARPosMode](#) [► 233].

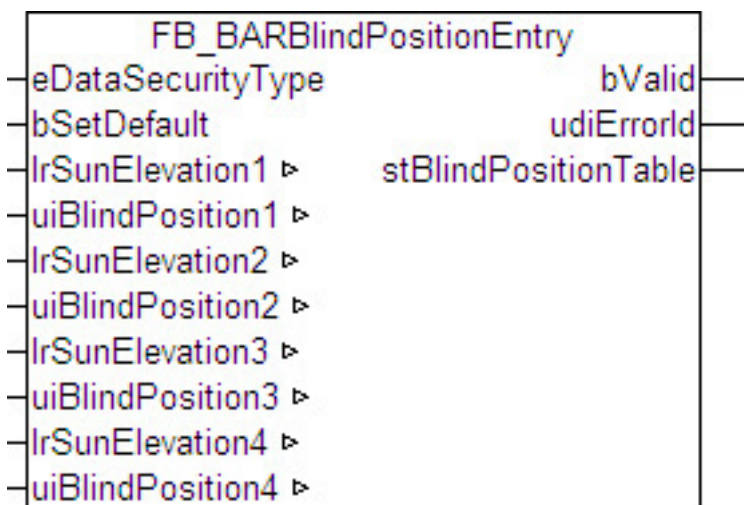


Fig. 7: FB_BARBlindPositionEntry

In addition to the operation modes "Fixed blind height" and "Maximum light incidence", the function block [FB_BARSunProtectionEx \[► 221\]](#) also offers the possibility to control the blind height in relation to the position of the sun by means of table entries. By entering several interpolation points, the blind height relative to the respective sun position is calculated by linear interpolation. However, since incorrectly entered values can lead to malfunctions in [FB_BARSunProtectionEx \[► 221\]](#), this function block is to be preceded by the function block [FB_BARBlindPositionEntry](#). Four interpolation points can be parameterized on this function block, whereby a missing entry is evaluated as a zero entry.

The function block does not sort the values entered independently, but instead ensures that the positions of the sun entered in the respective interpolation points are entered in ascending order. Unintentional erroneous entries are noticed faster as a result.

The values chosen for *rSunElevation1... rSunElevation4* must be unique, for example, the following situation must be avoided:

[*rSunElevation1* = 10 ; *uiBlindPosition1* = 50] and simultaneously [*rSunElevation2* = 10 ; *uiBlindPosition2* = 30].

This would mean that there would be two different target values for one and the same value, which does not allow a unique functional correlation to be established.

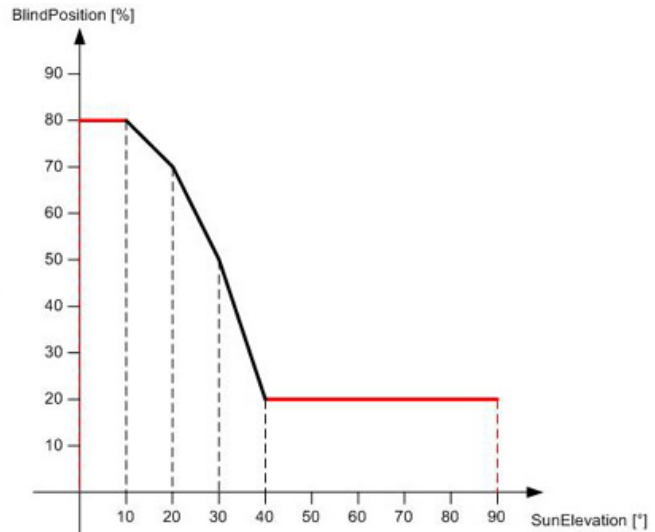
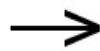
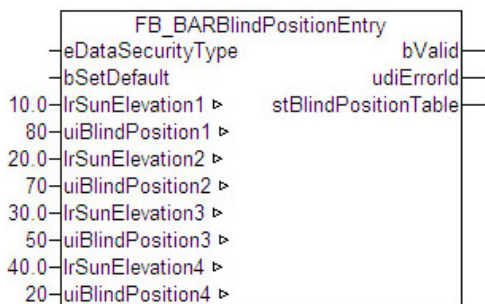
In addition, the entries for the position of the sun and blind height must lie within the valid range.

Mathematically this means that the following conditions must be satisfied:

- $rSunElevation1 < rSunElevation2 < rSunElevation3 < rSunElevation4$ - (values ascending and unequal)
- $0 \leq rSunElevation \leq 90$ (in degrees - range of validity of source values)
- $0 \leq uiBlindPosition \leq 100$ (in percent - range of validity of target values)

The function block checks the values entered for these conditions and outputs an [error code \[► 237\]](#) if they are not met. In addition, the output *bValid* is set to FALSE.

Furthermore the function block independently ensures that the boundary areas are filled out: internally, another interpolation point is set up at *rSunElevation* = 0 with *uiBlindPosition1* and another one above *rSunElevation4* at *rSunElevation* = 90 with *uiBlindPosition4*. This ensures that a sensible target value exists for all valid input values $0 \leq rSunElevation \leq 90$ **without** the user having to assign an entry for *rSunElevation* = 0 and *rSunElevation* = 90:



The actual number of interpolation points transferred to the function block [FB_BARSunProtectionEx \[► 221\]](#) thus increases to 6, see [ST_BARBlindPositionTable \[► 234\]](#).


The interpolation of the values takes place in the glare protection function block.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
```

eDataSecurityType: if *eDataSecurityType* := *eDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block [FB_HVACPersistentDataHandling](#) must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not internally released.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

VAR_OUTPUT

```
bValid           : BOOL;
uiErrorId       : UDINT;
stBlindPositionTable: ST_BARBlindPositionTable;
```

bValid: this output will be TRUE as long as the entries correspond to the criteria listed above.

uiErrorId: contains the error code, if the entries should not correspond to the criteria listed above. See [error codes \[► 237\]](#).

stBlindPositionTable : transfer structure of the interpolation points, see [ST_BARBlindPositionTable \[► 234\]](#).

VAR_IN_OUT

In order for the registered parameters to be retained beyond a controller failure, it is necessary to declare them as In-Out variables. A reference variable is then assigned to them in the program. Each change of the value of these reference variables is persistently stored in the function block and written back to the reference variable following a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

```
rSunElevation1   : REAL;;
uiBlindPosition1 : UINT;
rSunElevation2   : REAL;
uiBlindPosition2 : UINT;
rSunElevation3   : REAL;
uiBlindPosition3 : UINT;
rSunElevation4   : REAL;
uiBlindPosition4 : UINT;
```

rSunElevation1: position of the sun of the 1st interpolation point (0°..90°).

uiBlindPosition1: blind position (closing degree) of the 1st interpolation point (0%..100%).

rSunElevation2: position of the sun of the 2nd interpolation point (0°..90°).

uiBlindPosition2: blind position (closing degree) of the 2nd interpolation point (0%..100%).

rSunElevation3: position of the sun of the 3rd interpolation point (0°..90°).

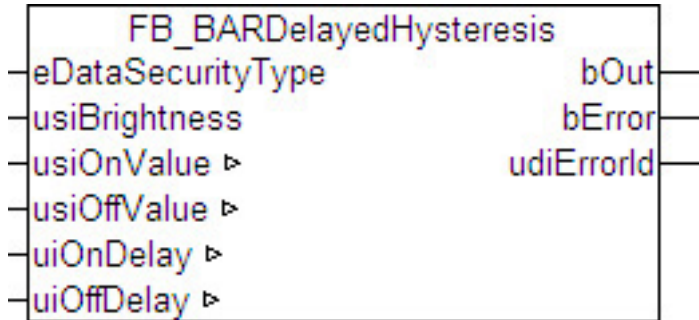
uiBlindPosition3: blind position (closing degree) of the 3rd interpolation point (0%..100%).

rSunElevation4: position of the sun of the 4th interpolation point (0°..90°).

uiBlindPosition4: blind position (closing degree) of the 4th interpolation point (0%..100%).

3.3.4.6 FB_BARDelayedHysteresis

This function block represents a threshold switch for brightness. The switch-on and switch-off behavior can additionally be delayed.




VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
uiBrightness      : UINT;
```

eDataSecurityType: if `eDataSecurityType := eDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not internally released.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

uiBrightness: outdoor brightness in lux.

VAR_OUTPUT

```
bOut      : BOOL;
bError    : BOOL;
udiErrorId : UDINT;
```


bOut: binary delayed output of the threshold switch.

bError: this output is switched to TRUE if the parameters entered are erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes \[▶ 237\]](#).

VAR_IN_OUT

In order for the registered parameters to be retained beyond a controller failure, it is necessary to declare them as In-Out variables. A reference variable is then assigned to them in the program. Each change of the value of these reference variables is persistently stored in the function block and written back to the reference variable following a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

```

uiOnValue : UINT;
uiOffValue : UINT;
uiOnDelay : UINT;
uiOffDelay : UINT;

```

uiOnValue: switch-on threshold value in lux. This must be greater than the switch-off threshold value *usiOffValue*.

uiOffValue: switch-off threshold value in lux. This must be smaller than the switch-on threshold value *usiOnValue*.

uiOnDelay: switch-on delay in seconds.

uiOffDelay: switch-off delay in seconds.

3.3.4.7 FB_BARFacadeElementEntry

This function block serves the administration of all facade elements (windows) in a facade, which are saved globally in a [list of facade elements](#) [► 237]. It is intended to facilitate the input of the element information - also with regard to the use of the target visualization. A schematic representation of the objects with description of the coordinates is shown in [Shading correction: principles and definitions](#) [► 152].



The facade elements are declared in the global variables as a two-dimensional field above the window columns and rows:

```

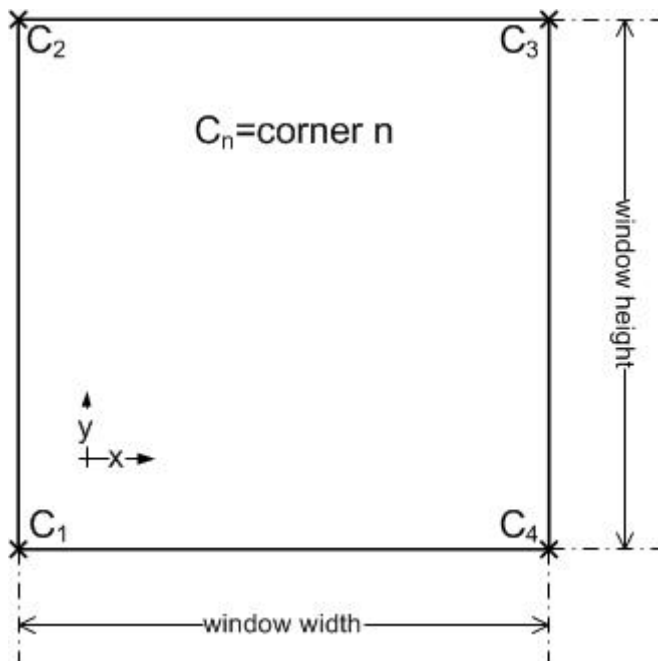
VAR_GLOBAL
    arrFacadeElement : ARRAY[1..iColumnsPerFacade, 1..iRowsPerFacade] OF ST_BARFacadeElement;
END_VAR

```

Each individual element *arrFacadeElement* carries the information for one facade element (*ST_BARFacadeElement* [► 235]). The information includes the group assignment, the dimensions (width, height) and the coordinates of the corners. The function block thereby accesses this field directly via the IN-OUT variable *arrFacadeElement*.



The fact that the coordinates of corners 2 to 4 are output values arises from the fact that they are formed from the input parameters and are to be available for use in a visualization.



All data in meters!

$lrCorner2X = lrCorner1X$
 $lrCorner2Y = lrCorner1Y + lrWindowHeight$ (window height)
 $lrCorner3X = lrCorner1X + lrWindowWidth$ (window width)
 $lrCorner3Y = lrCorner2Y$
 $lrCorner4X = lrCorner1X + lrWindowWidth$ (window width)
 $lrCorner4Y = lrCorner1Y$

The function block is used in three steps:

- Read
- Change
- Write

Read

With the entries in *iColumn* and *iRow* the corresponding element is selected from the list *arrFacadeElement[iColumn,iRow]*. A rising edge on *bRead* reads the following data from the list element:

- *usiGroup* Group membership,
- *lrCorner1X* X-coordinate of corner 1 in meters
- *lrCorner1Y* Y-coordinate of corner 1 in meters
- *lrWindowWidth* Window width in meters
- *lrWindowHeight* Window height in meters

These are then assigned to the corresponding input variables of the function block, which uses them to calculate the coordinates of corners 2-4 as output variables in accordance with the correlation described above. It is important here that the input values are not overwritten in the reading step. Hence, all values can initially be displayed in a visualization.

Change

In a next program step the listed input values can then be changed. The values entered are constantly checked for plausibility. The output *bValid* indicates whether the values are valid (*bValid*=TRUE). If this is not the case, a corresponding error code [► 237] is output at the output *udiErrorId*. See also below "Errors (bValid=FALSE)".

Write

The parameterized data are written to the list element with the index *nId* upon a positive edge on *bWrite*, regardless of whether they represent valid values or not. Therefore the element structure [ST_BARFacadeElement \[▸ 235\]](#) also contains a plausibility bit, *bValue*, that relays precisely this information to the function block [FB_BARShadingCorrection \[▸ 176\]](#) / [FB_BARShadingCorrectionSouth \[▸ 179\]](#) and prevents incorrect calculations there.

Error (bValid=FALSE)

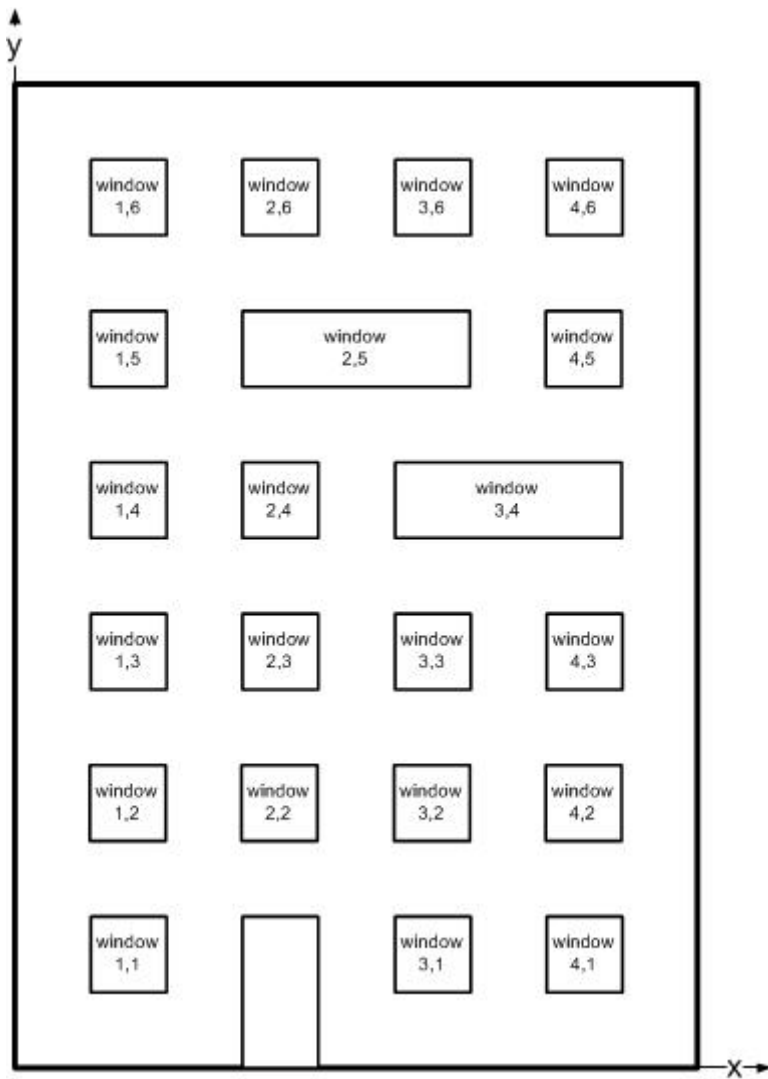
The function block [FB_BARShadingCorrection \[▸ 176\]](#) / [FB_BARShadingCorrectionSouth \[▸ 179\]](#), which judges whether all windows in a group are shaded, will only perform its task if all windows in the examined group have valid entries.

This means:

- *usiGroup* must be greater than 0
- *IrCorner1X* must be greater than or equal to 0.0
- *IrCorner1Y* must be greater than or equal to 0.0
- *IrWindowWidth* must be greater than 0
- *IrWindowHeight* must be greater than 0

If one of these criteria is not met, this is interpreted as an incorrect entry and *bValid* is set to FALSE at the function block output of [FB_BARFacadeElementEntry](#) and in the window element [ST_BARFacadeElement \[▸ 235\]](#).

If, on the other hand, all entries of a facade element are zero, it is regarded as a valid, deliberately omitted facade element:



In the case of a facade of 6x4 windows, the elements window (2.1), window (3.5) and window (4.4) would be empty elements here.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
iColumn           : INT;
iRow              : INT;
bWrite           : BOOL;
bRead            : BOOL;
usiGroup         : USINT;
lrCorner1X       : LREAL;
lrCorner1Y       : LREAL;
lrWindowWidth    : LREAL;
lrWindowHeight   : LREAL;
```

iColumn: column index of the selected component on the facade. This refers to the selection of a field element of the array stored in the IN-OUT variable *arrFacadeElement*.

iRow: ditto row index. **iRow and iColumn must not be zero!** This arises from the field definition, see above.

bRead: with a positive edge at this input, the information of the selected element, *arrFacadeElement[iColumn,iRow]* is read into the function block and assigned to the input variables *usiGroup* to *rWindowHeight*. This gives rise to the output variables *rCorner2X* to *rCorner4Y*. If data are already present on the inputs *usiGroup* to *rWindowHeight* at time of reading, then the data previously read are immediately overwritten with these data.

bWrite: a positive edge writes the entered as well as calculated values into the selected field element *arrFacadeElement[iColumn,iRow]*.

usiGroup: group membership.

lrCorner1X: X-coordinate of corner 1 in meters.

lrCorner1Y: Y-coordinate of corner 1 in meters.

lrWindowWidth: window width in meters.

lrWindowHeight: window height in meters.

VAR_OUTPUT

```
lrCorner2X : LREAL;
lrCorner2Y : LREAL;
lrCorner3X : LREAL;
lrCorner3Y : LREAL;
lrCorner4X : LREAL;
lrCorner4Y : LREAL;
bValid    : BOOL;
udiErrorId : UDINT;
```

lrCorner2X: determined X-coordinate of corner 2 of the window in meters. See "Info" above.

lrCorner2Y: determined Y-coordinate of corner 2 of the window in meters. See "Info" above.

lrCorner3X: determined X-coordinate of corner 3 of the window in meters. See "Info" above.

lrCorner3Y: determined Y-coordinate of corner 3 of the window in meters. See "Info" above.

lrCorner4X: determined X-coordinate of corner 4 of the window in meters. See "Info" above.

lrCorner4Y: determined Y-coordinate of corner 4 of the window in meters. See "Info" above.

bValid: result verification for the entered values.

udiErrorId: contains the error code if the values entered are not OK. See [error codes \[► 237\]](#).

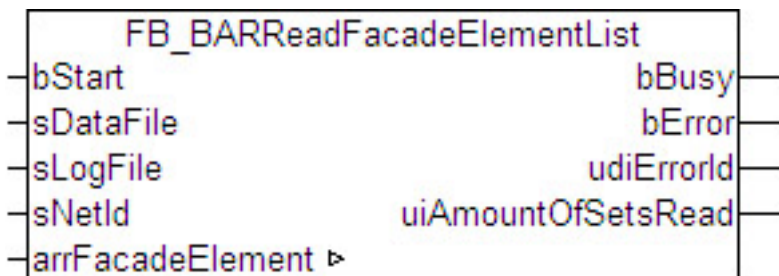
VAR_IN_OUT

```
arrFacadeElement : ARRAY[1..iColumnsPerFacade, 1..iRowsPerFacade] OF ST_BARFacadeElement;
```

arrFacadeElement: [list of facade elements \[► 237\]](#).

3.3.4.8 FB_BARReadFacadeElementList

With the help of this function block, data for facade elements (windows) can be imported from a pre-defined Excel table in csv format into the [List of facade elements \[► 237\]](#). In addition the imported data are checked for plausibility and errors are written to a log file.



VAR_INPUT

```
bStart    : BOOL;
sDataFile : STRING;
sLogFile  : STRING;
sNetId    : STRING;
```

bStart: a TRUE edge on this input starts the reading process.

sDataFile: contains the path and file name of the file to be opened. This must have been saved in Excel as file type "CSV (comma-separated values) (*.csv)". If the file is opened with a simple text editor, then the values must be displayed separated by semicolons. Example of an entry: *sDataFile:= 'C:\Projects\FacadeElements.csv'*

sLogFile: ditto log file for the accumulating errors. This file is overwritten each time the function block is activated, so that only current errors are contained.

sNetId: a string can be entered here with the AMS Net ID of the TwinCAT computer on which the files are to be written/read. If it is to be run on the local computer, an empty string can be entered.



The data can be saved only on the control computer itself and on the computers that are connected by ADS to the control computer. Links to local hard disks in this computer are possible, but not to connected network hard drives.

VAR_OUTPUT

```
bBusy          : BOOL;
bError         : BOOL;
udiErrorId    : UDINT;
uiAmountOfSetsRead: UINT;
```

bBusy: this output is TRUE as long as elements are being read from the file.

bError: this output is switched to TRUE if the parameters entered are erroneous or if an error has occurred while writing to or reading from the file.

udiErrorId: contains the error code of the error that occurred last. See [Error codes \[► 237\]](#) or [ADS_Error codes](#).

uiAmountOfSetsRead: number of data sets read

VAR_IN_OUT

```
arrFacadeElement : ARRAY[1..iColumnsPerFacade, 1..iRowsPerFacade] OF ST_BARFacadeElement;
```

arrFacadeElement: [list of facade elements \[► 237\]](#).

Excel table

The following example shows the Excel table with the entries of the window elements. All text fields are freely writable, important are the green marked fields, where each line indicates a data set. The following rules are to be observed:

- A data set must always start with a '@'.
- The indices *IndexColumn* and *IndexRow* must lie within the defined limits, see [List of facade elements \[► 237\]](#). These indices directly describe the facade element in the list *arrFacadeElements* to which the data from the set are saved.
- Window width and window height must be greater than zero
- The corner coordinates P1x and P1y must be greater than or equal to zero.
- Each window element must be assigned to a group 1...255.
- For system-related reasons the total size of the table may not exceed 65534 bytes.
- This must have been saved in Excel as file type "CSV (comma-separated values) (*.csv)".

It is not necessary to describe all window elements that would be possible by definition or declaration. Before the new list is read in, the function block deletes the entire old list in the program. All elements that are not described by entries in the Excel table then have pure zero entries and are thus marked as non-existent and also non-evaluable, since the function block for shading correction, [FB BARSunProtectionEx \[► 221\]](#), does not accept elements with the group entry '0'.

EN_FacadeElements.xls										
	A	B	C	D	E	F	G	H	I	J
1	Number	Description		IndexColumn	IndexRow	Window-Width	Window-Height	P1x	P1y	Group
2				(Axis)	(Floor)	[m]	[m]	[m]	[m]	
3		Text								
4	1	Description	@	1	1	1,2	1,3	1,5	1	2
5	2	Description	@	0	1	1,2	1,3	2,7	1	2
6	3	Description	@	3	1	1,2	1,3	4,4	1	2
7	4	Description	@	4	1	1,2	1,3	6,1	1	2
8	5	Description	@	5	1	1,2	1,3	7,8	1	2
9	6	Description	@	6	1	1,2	1,3	9,5	1	2
10	7	Description	@	7	1	1,2	1,3	11,2	1	2
11	8	Description	@	8	1	1,2	1,3	12,9	1	2
12	9	Description	@	9	1	1,2	1,3	14,6	1	2
13	10	Description	@	10	1	1,2	1,3	16,3	1	2
14	11	Description	@	1	1	1,2	1,3	1,5	4	3
15	12	Description	@	0	1	1,2	1,3	2,7	4	3
16	13	Description	@	3	1	1,2	1,3	4,4	4	3
17	14	Description	@	4	1	1,2	1,3	6,1	4	3
18	15	Description	@	5	1	1,2	1,3	7,8	4	3
19	16	Description	@	6	1	1,2	1,3	9,5	4	3
20	17	Description	@	7	1	1,2	1,3	11,2	4	3
21	18	Description	@	8	1	1,2	1,3	12,9	4	3
22	19	Description	@	9	1	1,2	1,3	14,6	4	3
23	20	Description	@	10	1	1,2	1,3	16,3	4	3
24	21	Description	@	1	1	1,2	1,3	1,5	7	4
25	22	Description	@	0	1	1,2	1,3	2,7	7	4
26	23	Description	@	3	1	1,2	1,3	4,4	7	4
27	24	Description	@	4	1	1,2	1,3	6,1	7	4
28	25	Description	@	5	1	1,2	1,3	7,8	7	4
29	26	Description	@	6	1	1,2	1,3	9,5	7	4
30	27	Description	@	7	1	1,2	1,3	11,2	7	4
31	28	Description	@	8	1	1,2	1,3	12,9	7	4
32	29	Description	@	9	1	1,2	1,3	14,6	7	4
33	30	Description	@	10	1	1,2	1,3	16,3	7	4
34	31	Description	@	1	1	1,2	1,3	1,5	10	5
35	32	Description	@	0	1	1,2	1,3	2,7	10	5
36	33	Description	@	3	1	1,2	1,3	4,4	10	5
37	34	Description	@	4	1	1,2	1,3	6,1	10	5
38	35	Description	@	5	1	1,2	1,3	7,8	10	5
39	36	Description	@	6	1	1,2	1,3	9,5	10	5
40	37	Description	@	7	1	1,2	1,3	11,2	10	5
41	38	Description	@	8	1	1,2	1,3	12,9	10	5
42	39	Description	@	9	1	1,2	1,3	14,6	10	5
43	40	Description	@	10	1	1,2	1,3	16,3	10	5
44										

Log file

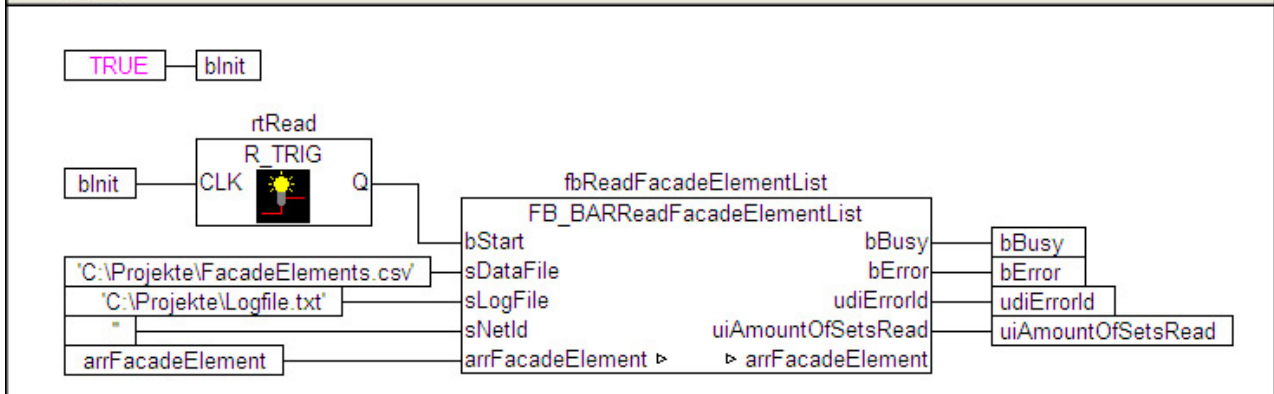
Each time the reading function block is restarted, the log file is rewritten and the old contents are deleted. If there is no log file, it will be automatically created first. The log file then contains either an OK message or a list of all errors that have occurred. Errors connected with the opening, writing or closing of the log file itself cannot be written at the same time. Therefore the output *udiErrorId* of the reading function block must also always be observed, since it displays the last error code. Since the log file is always closed last during the reading process, a corresponding alarm is ensured in the event of an error.

Program sample

```

0001 PROGRAM ReadFacadeElements
0002 VAR
0003     blnit                : BOOL;
0004     rtRead               : R_TRIG;
0005     fbReadFacadeElementList : FB_BARReadFacadeElementList;
0006     arrFacadeElement     : ARRAY[1..iColumnsPerFacade, 1..iRowsPerFacade] OF ST_BARFacadeElement;
0007
0008     bBusy                : BOOL;
0009     bError               : BOOL;
0010     udiErrorId          : UDINT;
0011     uiAmountOfSetsRead  : UINT;
0012 END_VAR
0013

```



In this sample the variable *blnit* is initially set to TRUE when the PLC starts. Hence, the input *bStart* on the function block *fbReadFacadeElementList* receives a once-only rising edge that triggers the reading process. The file "FacadeElements.csv" is read, which is located in the folder "C:\Projects\". The log file "Logfile.txt" is then saved in the same folder. If this log file does not yet exist it will be created, otherwise the existing contents are overwritten. Reading and writing take place on the same computer on which the PLC is located. This is defined by the input *sNetID* = "" (=local). All data are written to the list *arrFacadeElement* declared in the program. The output *bBusy* is set to TRUE as long as reading and writing is in progress. The error that occurred last is indicated on *udiErrorId*; *bError* is then TRUE. The number of found and read data rows is displayed at *uiAmountOfSetsRead* for verification purposes.

The errors marked were "built into" the following Excel list. This gives rise to the log file shown:

EN_FacadeElements.csv

	A	B	C	D	E	F	G	H	I	J
1	Number	Description		IndexColumn	IndexRow	Window-Width	Window-Height	P1x	P1y	Group
2				(Axis)	(Floor)	[m]	[m]	[m]	[m]	
3		Text								
4	1	Description	@	1	1	1,2	1,3	1,5	1,4	2
5	2	Description	@	0	1	1,2	1,3	2,7	1	2
6	3	Description	@	3	1	1,2	1,3	4,4	1	2
7	4	Description	@	4	1	1,2	1,3	6,1	1	2
8	5	Description	@	5	1	1,2	1,3	7,8	1	2
9	6	Description	@	6	1	0	1,3	9,5	1	2
10	7	Description	@	7	1	1,2	1,3	11,2	1	2
11	8	Description	@	8	1	1,2	1,3	12,9	1	2
12	9	Description	@	9	1	1,2	1,3	14,6	1	2
13	10	Description	@	10	1	1,2	1,3	16,3	1	5
14	11	Description	@	1	2	1,2	1,3	1	1	5
15	12	Description	@	2	2	1,2	1,3	2,7	3	5
16	13	Description	@	3	2	1,2	1,3	4,4	4	5
17	14	Description	@	4	2	1,2	1,3	4,4	4	5
18	15	Description	@	5	2	1,2	1,3	7,8	4	5
19	16	Description	@	6	2	1,2	1,3	9,5	4	5
20	17	Description	@	7	2	1,2	1,3	11,2	4	5
21	18	Description	@	8	2	1,2	1,3	12,9	4	5
22	19	Description	@	9	2	1,2	1,3	14,6	4	3
23	20	Description	@	10	2	1,2	1,3	16,3	4	3
24										
25	31	Description	@	1	3	1,2	1,3	1	7	3
26	32	Description	@	2	3	1,2	1,3	-1	6	3
27	33	Description	@	3	3	1,2	1,3	4,4	7	3
28	34	Description	@	4	3	1,2	1,3	6,1	7	0
29	35	Description	@	5	3	1,2	1,3	7,8	7	3
30	36	Description	@	6	3	1,2	1,3	9,5	7	3
31	37	Description	@	7	3	1,2	1,3	11,2	7	3
32	38	Description	@	8	3	1,2	1,3	12,9	7	7
33	39	Description	@	9	3	1,2	1,3	14,6	7	7
34	40	Description	@	10	3	1,2	1,3	16,3	7	7
35										
36										
37										
38										
39										
40										
41										
42										
43										
44										
45										
46										

LogFacade.txt - Editor

Datei Bearbeiten Format Ansicht ?

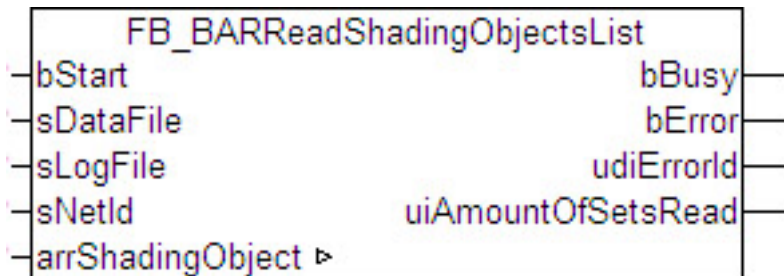
```

Index-Error in Data-Set #2
Validation-Error in Data-Set #6, ErrorId=32796
Validation-Error in Data-Set #22, ErrorId=32794
Validation-Error in Data-Set #24, ErrorId=32793
    
```

The first error is in data set 2 and is an index error, since "0" is not permitted. The next error in data set 6 was found after validation of the data with the internally used function block `FB_BARShadingObjectsEntry` [182] and therefore was assigned an error number, which is broken down in more detail in [error codes](#) [237]. The third and the fourth errors likewise occurred after the internal validation. Important here it that the data set numbers (in this case 22 and 24) do not go by the numbers entered in the list, but by the actual sequential numbers: only 30 data sets were read in here.

3.3.4.9 FB_BARReadShadingObjectsList

With the help of this function block, data for shading objects can be imported from a pre-defined Excel table in csv format into the [list of shading objects](#) [► 237]. In addition the imported data are checked for plausibility and errors are written to a log file.



VAR_INPUT

```
bStart      : BOOL;
sDataFile   : STRING;
sLogFile    : STRING;
sNetId      : STRING;
```

bStart: a TRUE edge on this input starts the reading process.

sDataFile: contains the path and file name of the file to be opened. This must have been saved in Excel as file type "CSV (comma-separated values) (*.csv)". If the file is opened with a simple text editor, then the values must be displayed separated by semicolons. Example of an entry: *sDataFile:= 'C:\Projects\ShadingObjects.csv'*

sNetId: a string can be entered here with the AMS Net ID of the TwinCAT computer on which the files are to be written/read. If it is to be run on the local computer, an empty string can be entered.



The data can be saved only on the control computer itself and on the computers that are connected by ADS to the control computer. Links to local hard disks in this computer are possible, but not to connected network hard drives.

VAR_OUTPUT

```
bBusy       : BOOL;
bError      : BOOL;
udiErrorId  : UDINT;
uiAmountOfSetsRead: UINT;
```

bBusy: this output is TRUE as long as elements are being read from the file.

bError: this output is switched to TRUE if the parameters entered are erroneous or if an error has occurred while writing to or reading from the file.

udiErrorId: contains the error code of the error that occurred last. See [Error codes](#) [► 237] or [ADS_Error codes](#).

uiAmountOfSetsRead: number of data sets read.

VAR_IN_OUT

```
arrShadingObject : ARRAY[1..iShadingObjects] OF ST_BARShadingObject;
```

arrShadingObject: [list of shading objects](#) [► 237].

Excel table

The following example shows the Excel table with the entries of the window elements.

All text fields are freely writable, important are the fields marked in green, where each row indicates a data set. The columns G to J have a different meaning depending on whether the type rectangle or sphere is concerned. The columns K to M are to be left empty in the case of spheres. With regard to the rectangle

coordinates, only the relevant data are entered and the remainder are internally calculated, see [FB BARShadingObjectsEntry \[► 182\]](#).

The following rules are to be observed:

- A data set must always start with a '@'.
- The month entries must not be 0 and not be greater than 12, all other combinations are possible.

Examples:

Start=1, End=1: shading in January.

Start=1, End=5: shading from the beginning of January to the end of May.

Start=11, End=5: shading from the beginning of November to the end of May (of the following year).

- Window width and window height must be greater than zero
- The z-coordinates P1z and P3z or Mz must be greater than zero.
- The radius must be greater than zero.
- For system-related reasons the total size of the table may not exceed 65534 bytes.
- This must have been saved in Excel as file type "CSV (comma-separated values) (*.csv)".

It is not necessary to describe all shading objects that are possible per facade. Only those contained in the list ultimately take effect.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Number	Description		Type	Begin	End	P1x/Mx	P1y/My	P1z/Mz	P2y/R	P3x	P3y	P3z
2				0 - Tetragon	(Month)	(Month)	[m]	[m]	[m]	[m]	[m]	[m]	[m]
3				1 - Globe									
4		Text											
5	1	Description	@	0	1	2	-94,75	0	36,06	11	-70,71	11	68,59
6	2	Description	@	0	1	2	-23,33	0	9,9	10,5	-3,54	10,5	22,62
7	3	Description	@	0	1	2	62,23	0	0	14,47	62,23	14,47	8
8	4	Description	@	0	1	2	46	0	13	14,47	62,23	14,47	8
9	5	Description	@	0	1	2	46	0	13	14,47	46	14,47	38,89
10	6	Description	@	0	1	2	0	0	14	9	35	9	14
11	7	Description	@	0	1	2	0	0	14	9,8	16	9,8	14
12	8	Description	@	0	1	2	23,6	0	14	9,8	25	9,8	14
13	9	Description	@	0	1	2	27,8	0	14	9,8	35	9,8	14
14													
15	10	Description	@	1	1	2	27	15	40	6			
16	11	Description	@	1	1	2	38	15	36	6			
17	12	Description	@	1	1	2	-14	4	4	1,5			
18	13	Description	@	1	1	2	-6,5	6	6	3,2			
19	14	Description	@	1	1	2	-7	9	6	1,2			
20	15	Description	@	1	1	2	-1	6	8	3,2			
21	16	Description	@	1	1	2	-1	9	8	1,2			

Log file

Each time the reading function block is restarted, the log file is rewritten and the old contents are deleted. If there is no log file, it will be automatically created first. The log file then contains either an OK message or a list of all errors that have occurred. However, errors connected with the opening, writing or closing of the log file itself cannot be written at the same time. Therefore the output *udiErrorId* of the reading function block must also always be observed, since it displays the last error code. Since the log file is always closed last during the reading process, a corresponding alarm is ensured in the event of an error.

Program sample

```

0001 PROGRAM ReadShadingObjects
0002 VAR
0003     blnit           : BOOL;
0004     rtRead          : R_TRIG;
0005     fbReadShadingObjectsList : FB_BARReadShadingObjectsList;
0006     arrShadingObject : ARRAY[1..iShadingObjects] OF ST_BARShadingObject;
0007
0008     bBusy           : BOOL;
0009     bError          : BOOL;
0010     udiErrorId     : UDINT;
0011     uiAmountOfSetsRead : UINT;
0012 END_VAR
0013

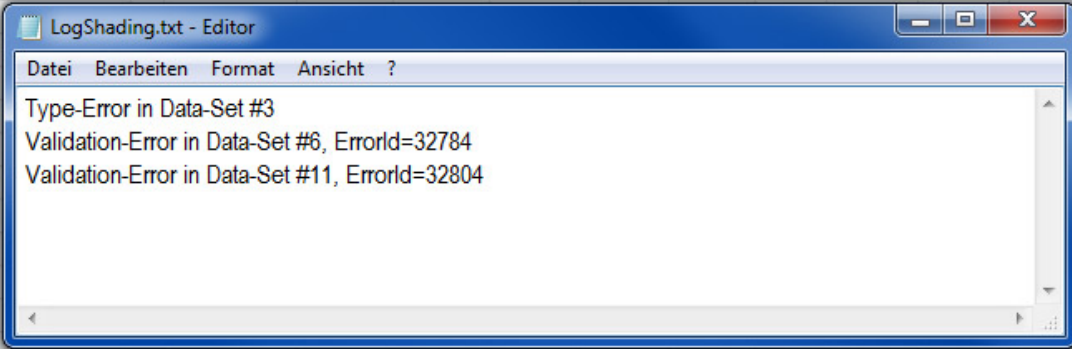
```

In this sample the variable *blnit* is initially set to TRUE when the PLC starts. Hence, the input *bStart* on the function block *fbReadShadingObjectsList* receives a once-only rising edge that triggers the reading process. The file "ShadingObjects.csv" is read, which is located in the folder "C:\Projects\". The log file "Logfile.txt" is then saved in the same folder. If this log file does not yet exist it will be created, otherwise the existing contents are overwritten. Reading and writing take place on the same computer on which the PLC is located. This is defined by the input *sNetID* = " (=local). All data are written to the list *arrShadingObject* declared in the program. The output *bBusy* is set to TRUE as long as reading and writing is in progress. The error that occurred last is indicated on *udiErrorId*; *bError* is then TRUE. The number of found and read data rows is displayed at *uiAmountOfSetsRead* for verification purposes.

The errors marked were built into the following Excel list. This gives rise to the log file shown:

EN_ShadingObjects.csv

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Number	Description	Type	Begin	End	P1x/Mx	P1y/My	P1z/Mz	P2y/R	P3x	P3y	P3z	
2			0 - Tetragon	(Month)	(Month)	[m]	[m]	[m]	[m]	[m]	[m]	[m]	
3			1 - Globe										
4		Text											
5	1	Description	@	0	1	2	-94,75	0	36,06	11	-70,71	11	68,59
6	2	Description	@	0	1	2	-23,33	0	9,9	10,5	-3,54	10,5	22,62
7	3	Description	@	2	1	2	62,23	0	0	14,47	62,23	14,47	8
8	4	Description	@	0	1	2	46	0	13	14,47	62,23	14,47	8
9	5	Description	@	0	1	2	46	0	13	14,47	46	14,47	38,89
10	6	Description	@	0	1	2	0	0	14	9	35	9	-14
11	7	Description	@	0	1	2	0	0	14	9,8	16	9,8	14
12	8	Description	@	0	1	2	23,6	0	14	9,8	25	9,8	14
13	9	Description	@	0	1	2	27,8	0	14	9,8	35	9,8	14
14													
15	11	Description	@	1	1	2	27	15	40	6			
16	12	Description	@	1	1	13	38	15	36	6			
17	13	Description	@	1	1	2	-14	4	4	1,5			
18	14	Description	@	1	1	2	-6,5	6	6	3,2			
19	15	Description	@	1	1	2	-7	9	6	1,2			
20	16	Description	@	1	1	2	-1	6	8	3,2			
21	17	Description	@	1	1	2	-1	9	8	1,2			
22													
23													
24													
25													
26													
27													
28													
29													
30													
31													
32													
33													
34													



The first error is in data set 3 and is a type error, since "2" is not defined.

The next error in data set 6 was found after validation of the data with the internally used function block [FB_BARShadingObjectsEntry](#) [► 182] and therefore was assigned an error number, which is broken down in more detail in [error codes](#) [► 237]. The third error likewise occurred after the internal validation. Important here it that the data set number (in this case 11) does not go by the numbers entered in the list, but by the actual sequential number: only 16 data sets were read in here.

3.3.4.10 FB_BARShadingCorrection

Function block for the shading evaluation of a window group on a facade.

This function block is valid only for the northern hemisphere. The valid function block for the southern hemisphere is [FB_BARShadingCorrectionSouth](#) [► 179].



The function block `FB_BARShadingCorrection` calculates whether a window group lies in the shadow of surrounding objects. The result, which is output at the output `bGroupNotShaded`, can be used to judge whether sun shading makes sense for this window group.

The function block thereby accesses two lists, which are to be defined:

- The data of the elements (window) of the facade in which the group to be regarded is located. This list of facade elements [► 237] is accessed via the IN-OUT variable `arrFacadeElement`, which itself is globally defined.
- The parameters that describe the shading elements that are relevant to the facade on which the window group is located. This list of shading objects [► 237] is likewise globally defined. The IN-OUT variable `arrShadingObject` accesses it directly.

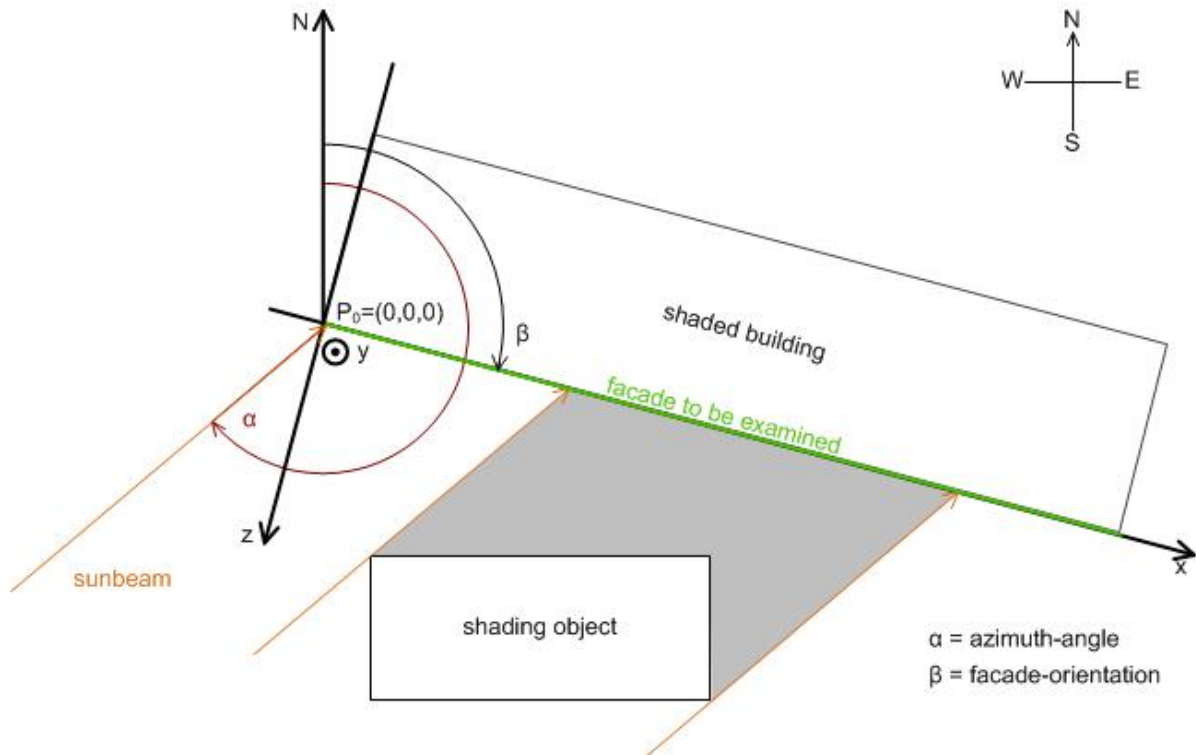
On the basis of the facade orientation (`lrFacadeOrientation`), the direction of the sun (`lrAzimuth`) and the sun elevation (`lrElevation`), a calculation can be performed for each corner of a window to check whether this lies in a shaded area. A window group is considered to be completely shaded if all corners are shaded.

In the northern hemisphere, the following applies for the facade orientation (looking out of the window):

Line of sight	Facade orientation
North	$\beta=0^\circ$
East	$\beta=90^\circ$
South	$\beta=180^\circ$
West	$\beta=270^\circ$

The function block performs its calculations only if the sun is actually shining on the facade. If one regards the drawing presented in the introduction, then this is the case if the following is true:

$$\text{Facade orientation} < \text{azimuth angle} < \text{facade orientation} + 180^\circ$$




In addition, a calculation is also not required, if the sun has not yet risen, i.e. the sun elevation is below 0° . In both cases the output *bFacadeSunlit* is set to FALSE.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
stTimeAct       : Timestruct;
lrAzimuth       : LREAL;
lrElevation     : LREAL;
```

eDataSecurityType: if *eDataSecurityType:= eDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instantiated once in the main program, which is called cyclically. Otherwise the instanced FB is not internally released.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If *eDataSecurityType:= eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if *eDataSecurityType:= eHVACDataSecurityType_Persistent*. It would lead to early wear of the flash memory.

stTimeAct: entry of the current time in GMT (Greenwich Mean Time).

rAzimuth: direction of the sun at the time of observation in degrees.

rElevation: sun elevation at the time of observation in degrees.

VAR_OUTPUT

```
bGroupNotShaded: BOOL;
bFacadeSunlit : BOOL;
bError : BOOL;
udiErrorId : UDINT;
```

bGroupNotShaded : is TRUE as long as the window group is not calculated as shaded.


bFacadeSunlit: this output is set to TRUE when the sun shines on the facade. See description above.

bError : this output is set to TRUE if an error is detected during the execution of the function block.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes \[▶ 237\]](#).

VAR_IN_OUT

In order for the registered parameters to be retained beyond a controller failure, it is necessary to declare them as In-Out variables. A reference variable is then assigned to them in the program. Each change of the value of these reference variables is persistently stored in the function block and written back to the reference variable following a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>.

```
arrFacadeElement : ARRAY[1..iColumnsPerFacade, 1..iRowsPerFacade] OF ST_BARFacadeElement;
arrShadingObject : ARRAY[1..iShadingObjects] OF ST_BARShadingObject;
lrFacadeOrientation : LREAL;
usiGroupID : USINT;
```

arrFacadeElement: [list of facade elements \[▶ 237\]](#).

arrShadingObject: [list of shading objects \[▶ 237\]](#).

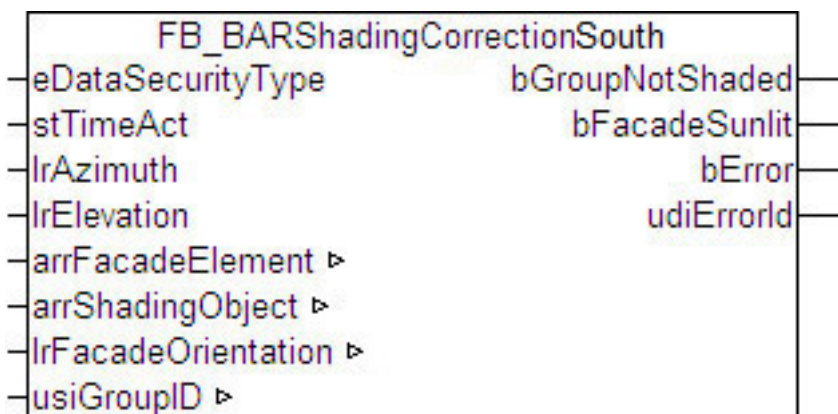
lrFacadeOrientation: facade orientation, see figure above.

usiGroupId: window group regarded. The group 0 is reserved here for unused window elements (see [FB_BARFacadeElementEntry \[▶ 164\]](#)). A 0-entry would lead to an error output (bError=TRUE). The function block is then not executed any further and *bGroupNotShaded* is set to FALSE.

3.3.4.11 FB_BARShadingCorrectionSouth

Function block for the shading evaluation of a window group on a facade.

This function block is valid only for the southern hemisphere. The valid function block for the northern hemisphere is [FB_BARShadingCorrection \[▶ 176\]](#).



The function block FB_BARShadingCorrectionSouth calculates whether a window group lies in the shadow of surrounding objects. The result, which is output at the output *bGroupNotShaded*, can be used to judge whether sun shading makes sense for this window group.

The function block thereby accesses two lists, which are to be defined:

- The data of the elements (window) of the facade in which the group to be regarded is located. This [list of facade elements](#) [► 237] is accessed via the IN-OUT variable *arrFacadeElement*, which itself is globally defined.
- The parameters that describe the shading elements that are relevant to the facade on which the window group is located. This [list of shading objects](#) [► 237] is likewise globally defined. The IN-OUT variable *arrShadingObject* accesses it directly.

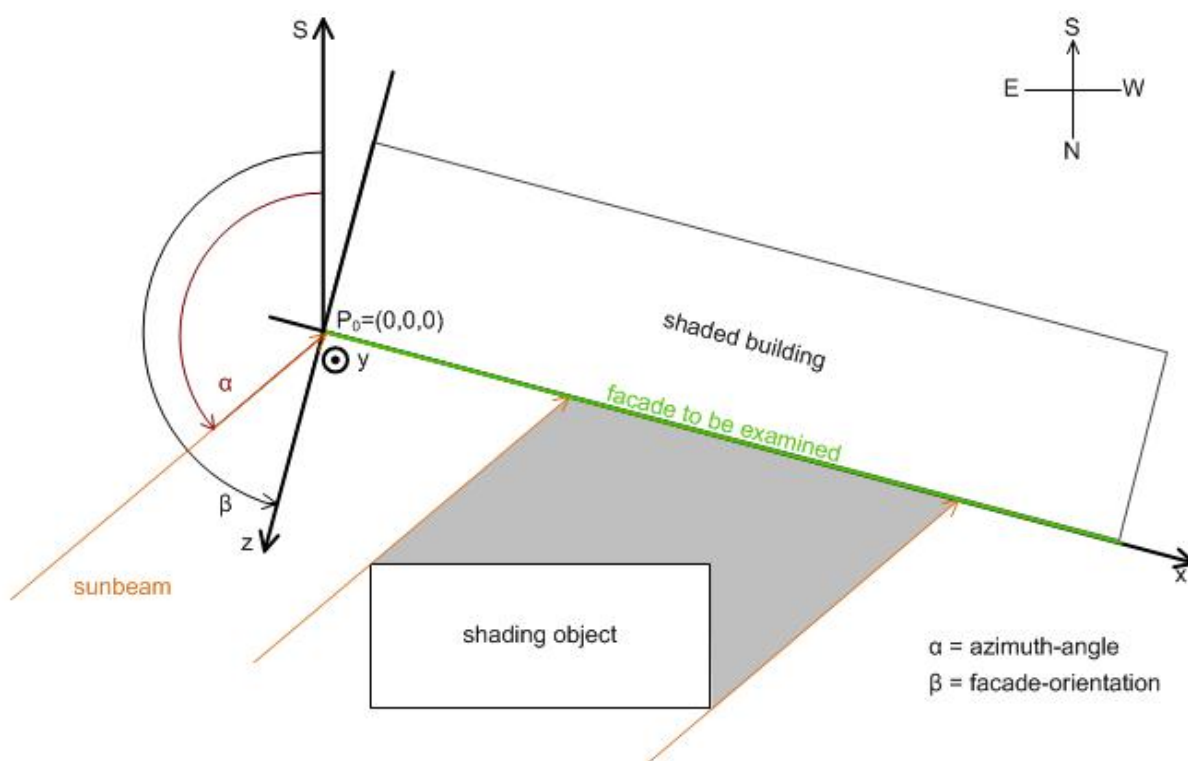
On the basis of the facade orientation (*lrFacadeOrientation*), the direction of the sun (*lrAzimuth*) and the sun elevation (*lrElevation*), a calculation can be performed for each corner of a window to check whether this lies in a shaded area. A window group is considered to be completely shaded if all corners are shaded.

In the southern hemisphere, the following applies for the facade orientation (looking out of the window):

Line of sight	Facade orientation
South	$\beta=0^\circ$
East	$\beta=90^\circ$
North	$\beta=180^\circ$
West	$\beta=270^\circ$

The function block performs its calculations only if the sun is actually shining on the facade. If one regards the drawing presented in the introduction, then this is the case if the following is true:

Facade orientation < azimuth angle < facade orientation + 180°




In addition, a calculation is also not required, if the sun has not yet risen, i.e. the sun elevation is below 0°. In both cases the output *bFacadeSunlit* is set to FALSE.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
stTimeAct         : TIMESTRUCT;
lrAzimuth         : LREAL;
lrElevation       : LREAL;
```

eDataSecurityType: if *eDataSecurityType* := *eDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not internally released.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

stTimeAct: entry of the current time in GMT (Greenwich Mean Time).

rAzimuth: direction of the sun at the time of observation in degrees.

rElevation: sun elevation at the time of observation in degrees.

VAR_OUTPUT

```
bGroupNotShaded: BOOL;
bFacadeSunlit   : BOOL;
bError          : BOOL;
udiErrorId      : UDINT;
```

bGroupNotShaded : is TRUE as long as the window group is not calculated as shaded.


bFacadeSunlit: this output is set to TRUE when the sun shines on the facade. See description above.

bError : this output is set to TRUE if an error is detected during the execution of the function block.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes \[► 237\]](#).

VAR_IN_OUT

In order for the registered parameters to be retained beyond a controller failure, it is necessary to declare them as In-Out variables. A reference variable is then assigned to them in the program. Each change of the value of these reference variables is persistently stored in the function block and written back to the reference variable following a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

```
arrFacadeElement : ARRAY[1..iColumnsPerFacade, 1..iRowsPerFacade] OF ST_BARFacadeElement;
arrShadingObject : ARRAY[1..iShadingObjects] OF ST_BARShadingObject;
lrFacadeOrientation : LREAL;
usiGroupID        : USINT;
```

arrFacadeElement: [list of facade elements \[► 237\]](#).

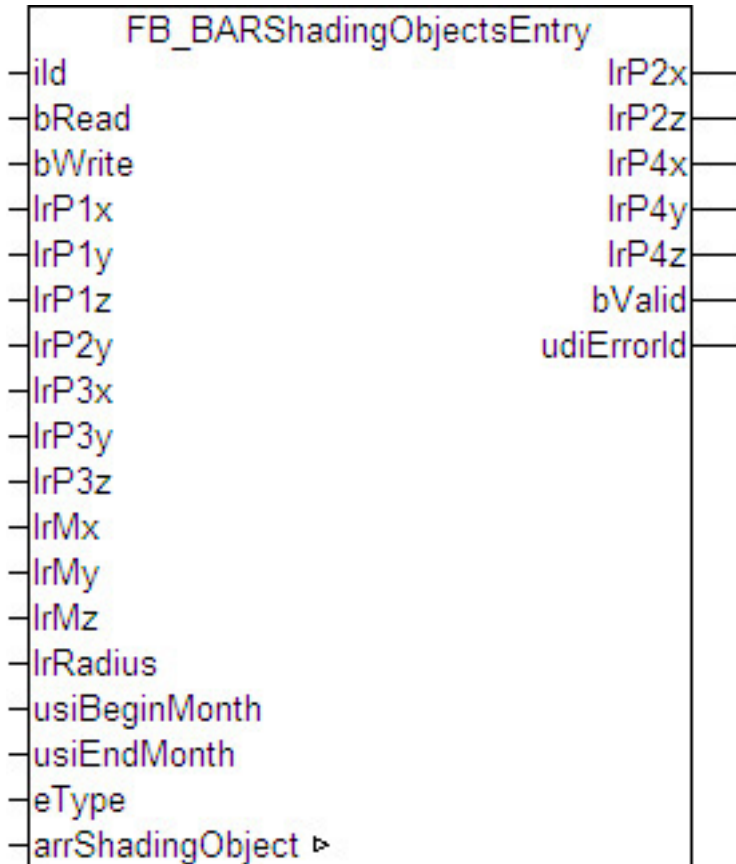
arrShadingObject: [list of shading objects \[► 237\]](#).

lrFacadeOrientation: facade orientation, see figure above.

usiGroupId: window group regarded. The group 0 is reserved here for unused window elements (see [FB BARFacadeElementEntry \[► 164\]](#)). A 0-entry would lead to an error output (`bError=TRUE`). The function block is then not executed any further and `bGroupNotShaded` is set to FALSE.

3.3.4.12 FB_BARShadingObjectsEntry

This function block serves for the administration of all shading elements in a facade, which is globally saved in a [list of shading elements](#) [▶ 237]. It is intended to facilitate the input of the element information - also with regard to the use of the target visualization. A schematic representation of the objects with description of the coordinates is shown in [Shading correction: principles and definitions](#) [▶ 152].



The shading elements are declared in the global variables:

```
VAR_GLOBAL
    arrShadingObject : ARRAY[1..iShadingObjects] OF ST_BARShadingObject;
END_VAR
```

Each individual element *arrShadingObject[1]* to *arrShadingObject[iShadingObjects]* carries the information for one shading element ([ST_BARShadingObject](#) [▶ 235]). This information consists of the selected type of shading (rectangle or sphere) and the respectively associated coordinates. For a rectangle, these are the corner points (*lrP1x*, *lrP1y*, *lrP1z*), (*lrP2x*, *lrP2y*, *lrP2z*), (*lrP3x*, *lrP3y*, *lrP3z*) and (*lrP4x*, *lrP4y*, *lrP4z*), for a sphere this are the center point (*lrMx*, *lrMy*, *lrMz*) and the radius *lrRadius*. In addition, the phase of the shading can be defined via the inputs *usiBeginMonth* and *usiEndMonth*, which is important in the case of objects such as trees that bear no foliage in winter.

The function block thereby directly accesses the field of this information via the IN-OUT variable *arrShadingObject*.



The fact that the rectangle coordinates *lrP2x*, *lrP2z*, *lrP4x*, *lrP4y* and *lrP4z* are output values arises from the fact that they are formed from the input parameters.

$lrP2x = lrP1x$; $lrP2z = lrP1z$; $lrP4x = lrP3x$; $lrP4y = lrP1y$; $lrP4z = lrP3z$;

That limits the input of a rectangle to the extent that the lateral edges stand vertically on the floor ($lrP2x = lrP1x$ and $lrP4x = lrP3x$), that the rectangle has no inclination ($lrP2z = lrP1z$ and $lrP4z = lrP3z$) and can only have a different height "upwards", i.e. in the positive y-direction ($lrP4y = lrP1y$).

The function block is used in three steps:

- Read
- Change
- Write

Read

With the entry to *ild* the appropriate element is selected from the list *arrShadingObject[ild]*. A rising edge on *bRead* reads the data. These values are assigned to the input and output variables of the function block. These are the input values *lrP1x*, *lrP1y*, *lrP1z*, *lrP2y*, *lrP3x*, *lrP3y*, *lrP3z*, *lrMx*, *lrMy*, *lrMz*, *rRadius*, the object enumerator *eType* and the output values *lrP2x*, *lrP2z*, *lrP4x*, *lrP4y* and *lrP4z*. It is important here that the input values are not overwritten in the reading step. Hence, all values can initially be displayed in a visualization.

Change

In a next program step the listed input values can then be changed. If a rectangle is preselected at input *eType* [► 234] via the value "*eObjectTypeTetragon*", the output values *rP2x*, *rP2z*, *rP4x*, *rP4y* and *rP4z* result from the rectangle coordinates that were entered (see above).

The values entered are constantly checked for plausibility. The output *bValid* indicates whether the values are valid (*bValid*=TRUE). If this is not the case, a corresponding *error code* [► 237] is output at the *udiErrorId* output.

If a rectangle is defined, only the inputs *lrP1x*, *lrP1y*, *lrP1z*, *lrP2y*, *lrP3x*, *lrP3y* and *lrP3z* have to be described; the inputs *lrMx*, *lrMy*, *lrMz* and *lrRadius* do not have to be linked. For a sphere definition, only *lrMx*, *lrMy*, *lrMz* and *lrRadius* have to be described; the rectangle coordinates can remain unlinked

The parameterized data are written to the list element with the index *ild* upon a positive edge on *bWrite*, regardless of whether they represent valid values or not. Therefore the element structure *ST_BARShadingObject* [► 235] also contains a plausibility bit, *bValid*, that relays precisely this information to the function block *FB_BARShadingCorrection* [► 176] / *FB_BARShadingCorrectionSouth* [► 179] and prevents incorrect calculations there.

This approach is to be regarded only as a proposal. It is naturally also possible to parameterize the function block quite normally in one step and to write the values entered to the corresponding list element with a rising edge on *bWrite*.

VAR_INPUT

```
iId      : INT;
bRead    : BOOL;
bWrite   : BOOL;
lrP1x    : LREAL;
lrP1y    : LREAL;
lrP1z    : LREAL;
lrP2y    : LREAL;
lrP3x    : LREAL;
lrP3y    : LREAL;
lrP3z    : LREAL;
lrMx     : LREAL;
lrMy     : LREAL;
lrMz     : LREAL;
lrRadius : LREAL;
usiBeginMonth: USINT;
usiEndMonth : USINT;
eType    : E_BARShadingObjectType;
```

ild: index of the selected element. This refers to the selection of a field element of the array stored in the IN-OUT variable *arrShadingObject*. **ild may not be zero!** This results from the *field definition* [► 182], *arrShadingObject* : *ARRAY[1..iShadingObjects] OF ST_BARShadingObject*

bRead: the information of the selected element, *arrShadingObject[iId]*, is read into the function block with a positive edge at this input and assigned to the input variables *lrP1x* to *eType* and the output variables *lrP2x* to *lrP4z*. If at this time data have already been applied to the inputs *lrP1x* to *eType*, the previously read data are immediately overwritten with these.

bWrite: a positive edge writes the values applied to inputs *rP1x* to *eType* and the values determined and assigned to outputs *lrP2x* to *lrP4z* to the selected field element *arrShadingObject[iId]*.

lrP1x: X-coordinate of point 1 of the shading element (rectangle) in meters.

lrP1y: Y-coordinate of point 1 of the shading element (rectangle) in meters.

lrP1z: Z-coordinate of point 1 of the shading element (rectangle) in meters.

lrP2y: Y-coordinate of point 2 of the shading element (rectangle) in meters.

lrP3x: X-coordinate of point 3 of the shading element (rectangle) in meters.

lrP3y: Y-coordinate of point 3 of the shading element (rectangle) in meters.

lrP3z: Z-coordinate of point 3 of the shading element (rectangle) in meters.

lrMx: X-coordinate of the center of the shading element (sphere) [m].

lrMy: Y-coordinate of the center of the shading element (sphere) in meters.

lrMz: Z-coordinate of the center of the shading element (sphere) in meters.

lrRadius: radius of the shading element (sphere) in meters.

usiBeginMonth: beginning of the shading period (month).

usiEndMonth: end of the shading period (month).

eType: selected element type: rectangle or sphere. See [E_BARShadingObjectType \[► 234\]](#).

Comment on shading period:

The month entries must not be 0 and not be greater than 12, all other combinations are possible.

Examples:

Start=1, End=1: shading in January.

Start=1, End=5: shading from the beginning of January to the end of May.

Start=11, End=5: shading from the beginning of November to the end of May (of the following year).

VAR_OUTPUT

```
lrP2x      : LREAL;
lrP2z      : LREAL;
lrP4x      : LREAL;
lrP4y      : LREAL;
lrP4z      : LREAL;
bValid     : BOOL;
udiErrorId : UDINT;
```

lrP2x: determined X-coordinate of point 2 of the shading element (rectangle) in meters. See "Info" above.

lrP2z: determined Z-coordinate of point 2 of the shading element (rectangle) in meters. See "Info" above.

lrP4x: determined X-coordinate of point 4 of the shading element (rectangle) in meters. See "Info" above.

lrP4y: determined Y-coordinate of point 4 of the shading element (rectangle) in meters. See "Info" above.

lrP4z: determined Z-coordinate of point 4 of the shading element (rectangle) in meters. See "Info" above.

bValid: result of the plausibility check for the values entered. With respect to a square it is required that the internal angle is 360° and that the points lie in one plane and *in front* of the facade regarded. In the case of a sphere the center must likewise lie in front of the facade and the radius must be greater than zero.

udiErrorId: contains the error code, if the entries should not correspond to the mentioned criteria. See [error codes \[► 237\]](#).

VAR_IN_OUT

```
arrShadingObject : ARRAY[1..iShadingObjects] OF ST_BARShadingObject;
```

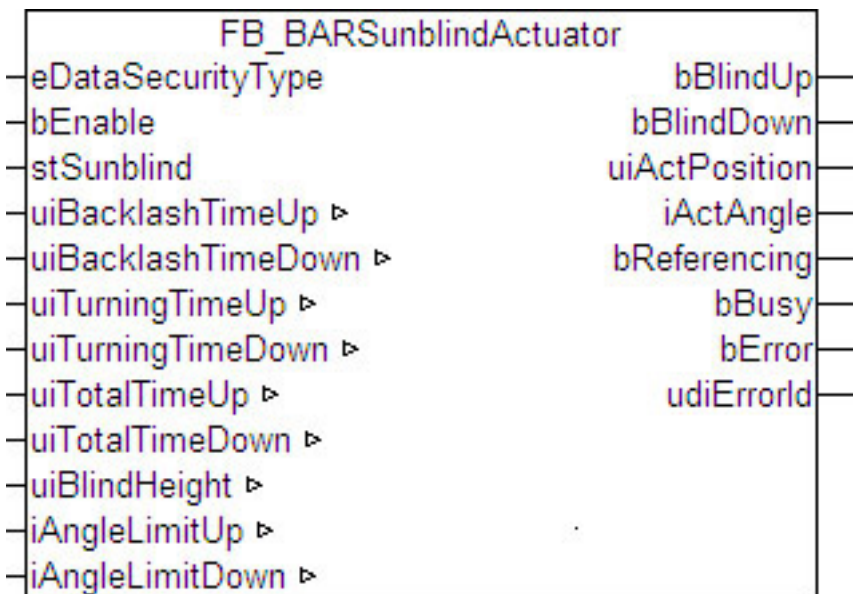
arrShadingObject: [list of shading objects \[▶ 237\]](#). Values are persistently saved.

3.3.4.13 FB_BARSunblindActuator

This function block is used for positioning of a slatted blind via two outputs: up and down. The blind can be driven to any desired (height) position and slat angle via the positioning telegram [stSunblind \[▶ 236\]](#). On top of that, the positioning telegram [stSunblind \[▶ 236\]](#) also contains manual commands with which the blind can be moved individually to certain positions. These manual commands are controlled by the function block [FB_BARSunblindSwitch \[▶ 206\]](#).

The total stroke time in milliseconds is limited here by the UINT format to 65535 ms.

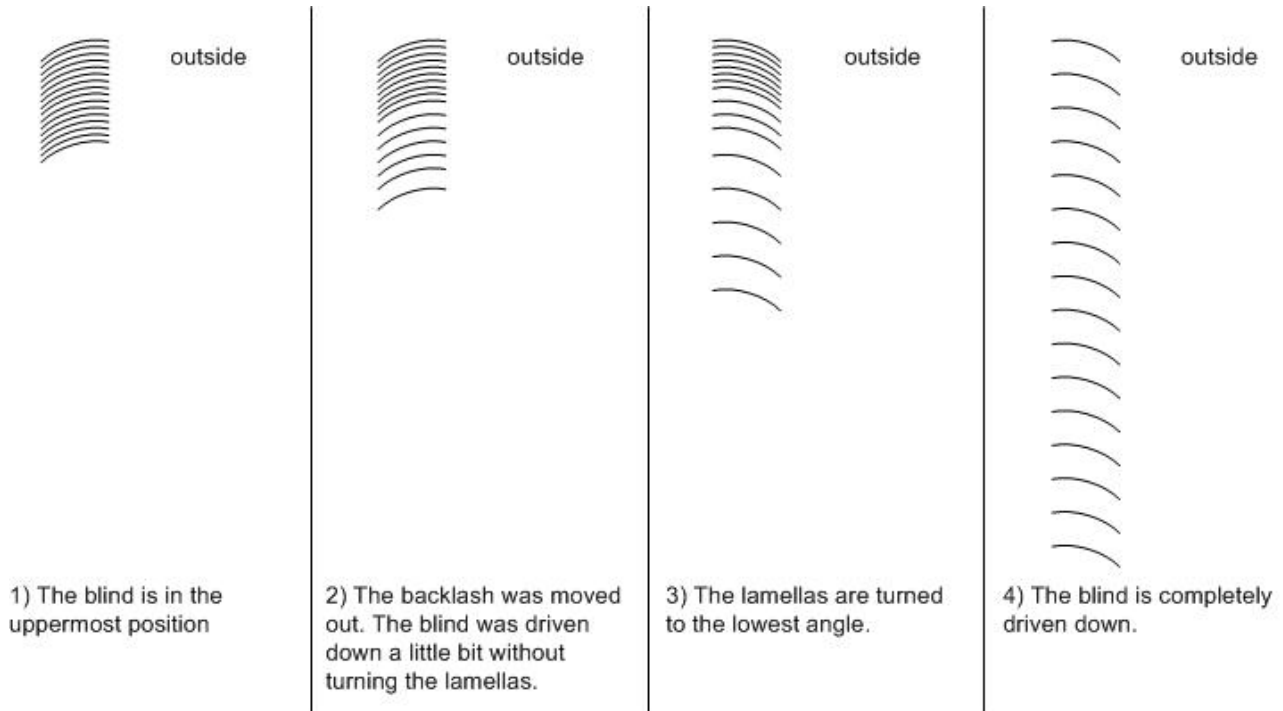
[FB_BARSunblindActuatorEx \[▶ 190\]](#) should be used for longer stroke times as it represents these times in UDINT format (*udiTotalTimeUp/udiTotalTimeDown*).



Structure of the blind positioning telegram [stSunblind \[▶ 236\]](#).

```
TYPE ST_BARSunblind:
STRUCT
    uiPosition : UINT;
    iAngle     : INT;
    bManUp     : BOOL;
    bManDown   : BOOL;
    bManualMode : BOOL;
    bActive    : BOOL;
END_STRUCT
END_TYPE
```

The current height position and the slat angle are not read in by an additional encoder, but determined internally by the travel time of the blind. The calculation is based on the following travel profile (regarded from the highest and lowest position of the blind):

Downward travel profile:

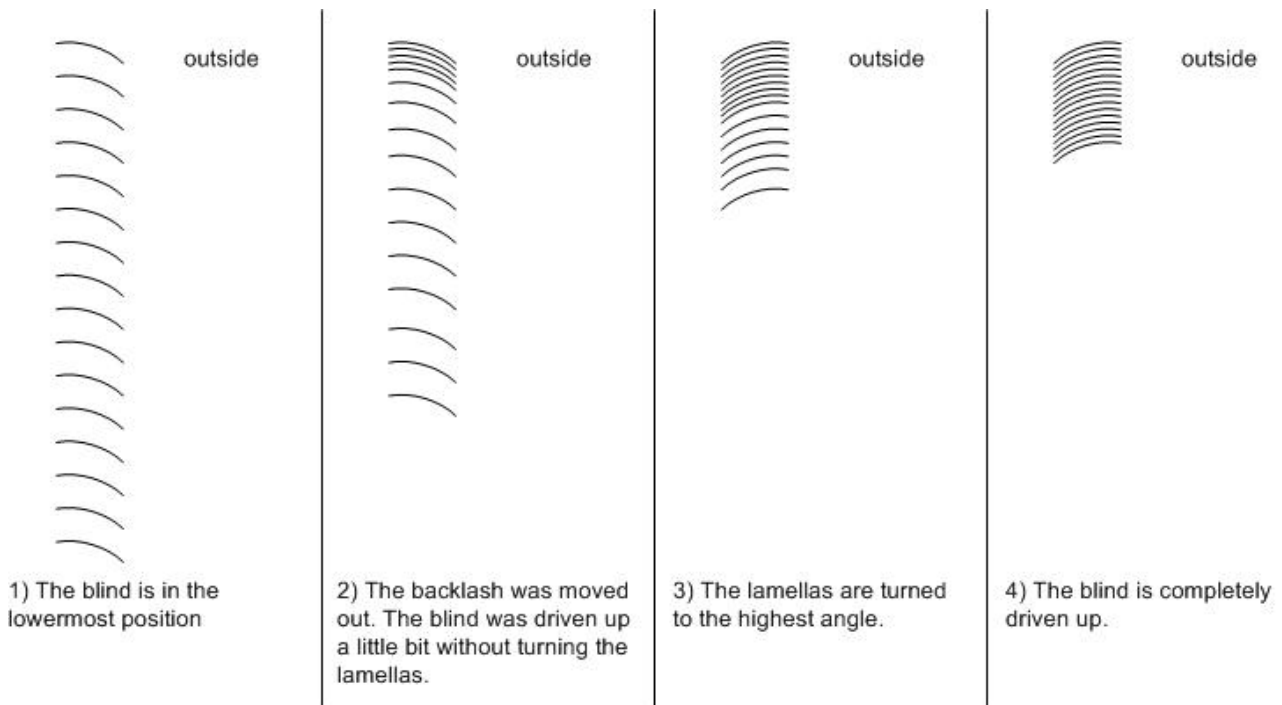
More detailed explanations of the terms "backlash" and "turning" are given here in the downward movement:

The blind normally describes its downward movement with the slat low point directed outwards, as in fig. 3.

If the blind is in an initial position with the low point directed inwards (i.e. after the conclusion of an upward movement), then a certain time elapses after a new downward movement begins before the slats start to turn from the "inward low point" to the "outward low point". During this time the slat angle does not change; the blind only drives downward (fig. 1 and fig. 2). This time is an important parameter for the movement calculation and is entered in the function block in ms under *uiBacklashTimeDown*. Since it is not known at an arbitrary point after a blind movement of an arbitrary length whether part of the backlash has already been traveled, the most secure way to measure the backlash of the downward movement or its travel time is when the blind has initially

been driven fully upward. A further important parameter is the timespan of the subsequent turning of the slats from the "Inward low point" to the "Outward low point". This time is to be entered as *uiTurningTimeDown* in ms in the function block.

Upward travel profile:



More detailed explanations of the terms "backlash" and "turning" are given here in the upward movement:

The circumstances are similar to the downward movement described above: the blind normally describes its upward movement with the slat low point directed inwards, as in fig. 3. If the blind is in an initial position with the low point directed outwards (i.e. after the conclusion of a downward movement), then a certain time elapses after a new upward movement begins before the slats start to turn from the "Outward low point" to the "Inward low point". During this time the slat angle does not change; the blind only drives upward (fig. 1 and fig. 2). Also this time is an important parameter for the movement calculation and is entered in the function block in ms under *uiBacklashTimeUp*. Since it is not known at an arbitrary point after a blind movement of an arbitrary length whether part of the backlash has already been traveled, the most secure way to measure the backlash of the upward movement or its travel time is when the blind has initially been driven fully downward. A further important parameter is the time interval of the subsequent turning of the slats from the "Outward low point" to the "Inward low point". This time is to be entered as *uiTurningTimeUp* in ms in the function block.

Parameterization

For the calculation of the (height) position and the slat angle, the following times now have to be determined for both the upward and downward movement:

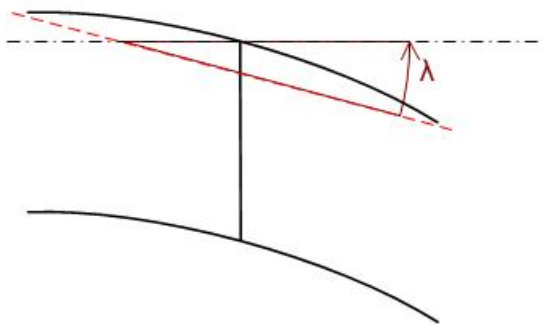
- the travel time of the backlash (*uiBacklashTimeUp* / *uiBacklashTimeDown* in ms)
- the turning duration (*uiTurningTimeUp* / *uiTurningTimeDown* in ms)
- the total travel time (*uiTotalUpTime* / *uiTotalDownTime* in ms)

Furthermore the following are required for the calculation:

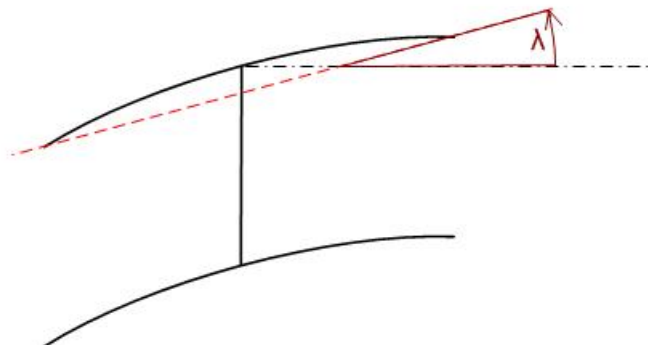
- the total extended blind height (*uiBlindHeight* in mm)
- the highest slat angle after turning upwards (*iAngleLimitUp* in degrees)
- the lowest slat angle after turning downwards (*iAngleLimitDown* in degrees)

The slat angle λ is defined by a notional straight line through the end points of the slat to the horizontal.

louvre angle $\lambda < 0$



louvre angle $\lambda > 0$



Functioning

The function block fundamentally controls the blind via the information from the positioning telegram `stSunblind` [► 236]. If automatic mode is active (`bManualMode=FALSE`), the current position and slat angle are always approached, and changes are immediately taken into account. The positioning to the height has priority: the entered height is approached first and then the slat angle. For reasons of the simplicity the position error due to the angle movement is disregarded. In manual mode (`bManualMode=TRUE`) the commands `bManUp` and `bManDown` control the blind.

Referencing

Secure referencing is ensured if the blind is driven upward for longer than its complete travel-up time. The position is then in any case "0" and the slat angle is at its maximum. Since a blind positioning without encoder is naturally always error-prone, it is important to reference automatically as often as possible: every time the position "0" is to be approached (the angle does not matter), the blind first moves up normally with continuous position calculation. Once the calculated position value 0% is reached, the output `bBlindUp` continues to be held for the complete travel-up time + 5s.

For reasons of flexibility there are now two possibilities to interrupt the referencing procedure: until the calculated 0% position is reached, a change in position continues to be assumed and executed. Once this 0% position is reached, the blind can still be moved with the manual "blind down" command. These two sensible limitations make it necessary for the user to ensure that the blind is securely referenced as often as possible.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bEnable           : BOOL;
stSunblind       : ST_BARSunblind;
```

eDataSecurityType: if `eDataSecurityType:= eDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not internally released.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bEnable: enable input for the function block. As long as this input is TRUE, the actuator function block accepts and executes commands as described above. A FALSE signal on this input resets the control outputs `bBlindUp` and `bBlindDown` and the function block remains in a state of rest.

stSunblind: positioning telegram, see [ST_BARSunblind \[► 236\]](#).

VAR_OUTPUT

```
bBlindUp      : BOOL;
bBlindDown    : BOOL;
uiActPosition : UINT;
uiActAngle    : UINT;
bReferencing  : BOOL;
bBusy         : BOOL;
bError        : BOOL;
udiErrorId    : UDINT;
```

bBlindUp: control output blind up.

bBlindDown: control output blind down.

uiActPosition: current position in percent.

uiActAngle: current slat angle in degrees.

bReferencing: the blind is referencing, i.e. the output `bBlindUp` is set for the complete travel-up time + 5 s. Only a manual "down" command can move the blind in the opposite direction and terminate this mode.


bBusy: a positioning or a referencing procedure is in progress.

bError: this output is switched to TRUE if the parameters entered are erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes \[► 237\]](#).

VAR_IN_OUT

In order for the registered parameters to be retained beyond a controller failure, it is necessary to declare them as In-Out variables. A reference variable is then assigned to them in the program. Each change of the value of these reference variables is persistently stored in the function block and written back to the reference variable following a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>.

```
uiBacklashTimeUp   : UINT;
uiBacklashTimeDown : UINT;
uiTurningTimeUp    : UINT;
uiTurningTimeDown  : UINT;
uiTotalTimeUp      : UINT;
uiTotalTimeDown    : UINT;
uiBlindHeight      : UINT;
iAngleLimitUp      : INT;
iAngleLimitDown    : INT;
```

uiBacklashTimeUp: time to traverse the backlash in the upward direction in ms.

uiBacklashTimeDown: time to traverse the backlash in the downward direction in ms.

uiTurningTimeUp: time for turning the slats in the upward direction in ms.

uiTurningTimeDown: time for turning the slats in the downward direction in ms.

uiTotalTimeUp: total time for driving up in ms.

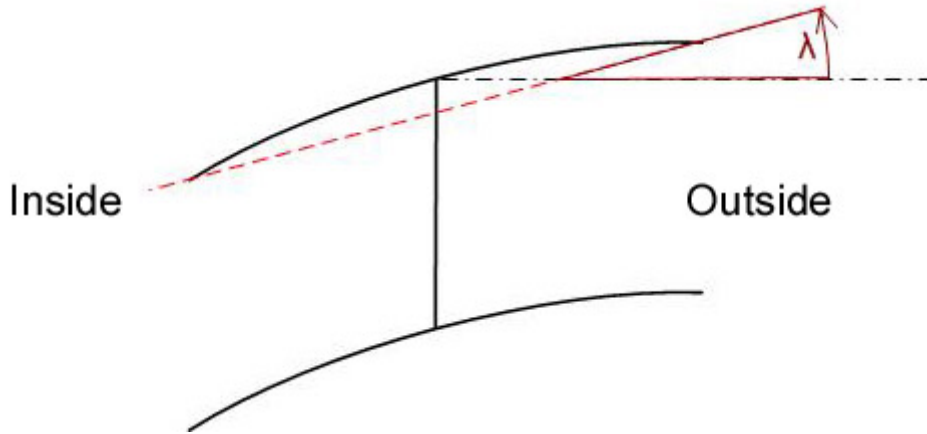
uiTotalTimeDown: total time for driving down in ms.

uiBlindHeight: blind height in mm.

iAngleLimitUp: highest position of the slats in degrees.

This position is reached once the blind has moved to the top position.

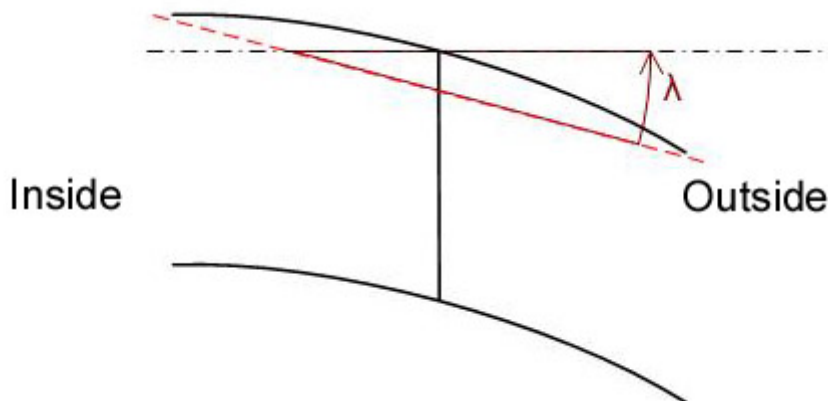
The slat angle λ , as defined above, is then typically greater than zero.



iAngleLimitDown: lowest position of the slats in degrees.

This position is reached once the blind has moved to the bottom position.

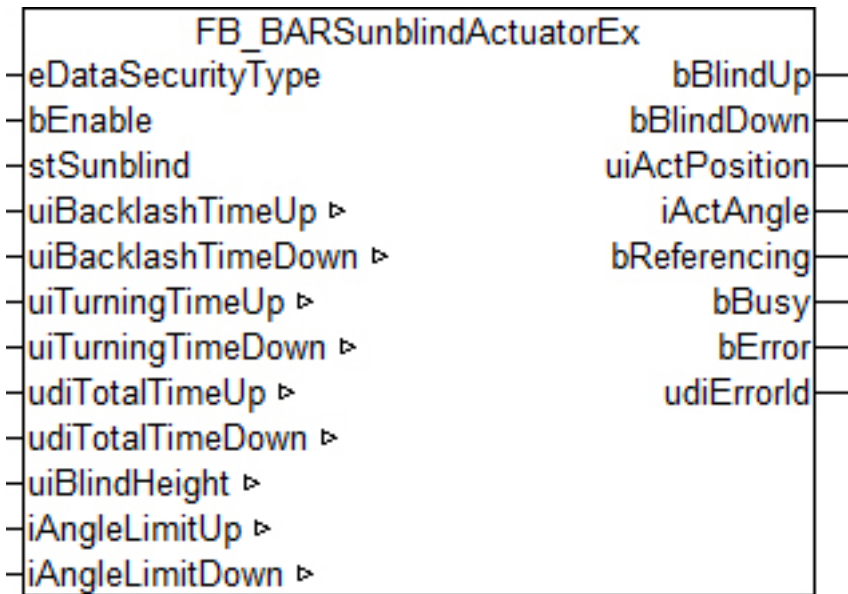
The slat angle λ , as defined above, is then typically less than zero.



3.3.4.14 FB_BARSunblindActuatorEx

This function block is used for positioning of a slatted blind via two outputs: up and down. The blind can be driven to any desired (height) position and slat angle via the positioning telegram `stSunblind` [► 236]. On top of that, the positioning telegram `stSunblind` [► 236] also contains manual commands with which the blind can be moved individually to certain positions. These manual commands are controlled by the function block `FB_BARSunblindSwitch` [► 206].

As opposed to the function block `FB_BARSunblindActuator` [► 185], this allows the input of longer stroke times because they are represented here not in UINT, but in the UDINT format (`udiTotalTimeUp/udiTotalTimeDown`).



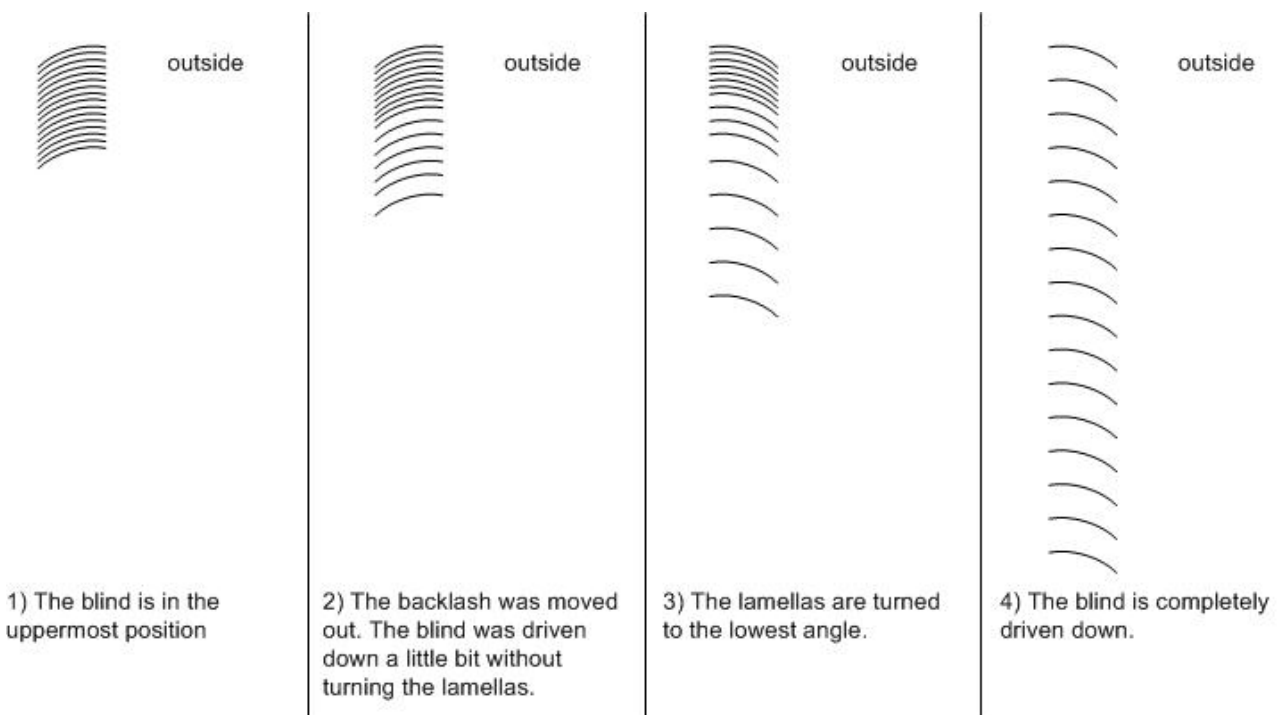
Structure of the blind positioning telegram `stSunblind` [▶ 236].

```

TYPE ST_BARSunblind:
STRUCT
    uiPosition : UINT;
    iAngle     : INT;
    bManUp     : BOOL;
    bManDown   : BOOL;
    bManualMode : BOOL;
    bActive    : BOOL;
END_STRUCT
END_TYPE
    
```

The current height position and the slat angle are not read in by an additional encoder, but determined internally by the travel time of the blind. The calculation is based on the following travel profile (regarded from the highest and lowest position of the blind):

Downward travel profile:

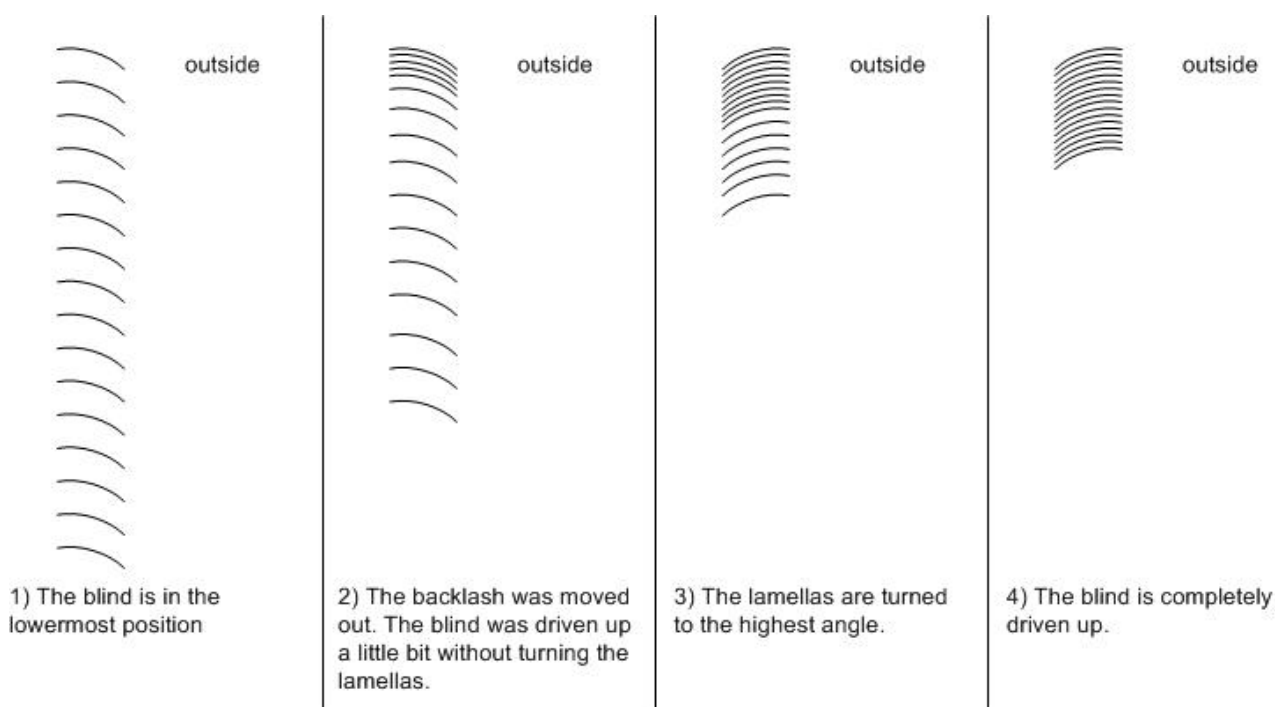


More detailed explanations of the terms "backlash" and "turning" are given here in the downward movement:

The blind normally describes its downward movement with the slat low point directed outwards, as in fig. 3.

If the blind is in an initial position with the low point directed inwards (i.e. after the conclusion of an upward movement), then a certain time elapses after a new downward movement begins before the slats start to turn from the "inward low point" to the "outward low point". During this time the slat angle does not change; the blind only drives downward (fig. 1 and fig. 2). This time is an important parameter for the movement calculation and is entered in the function block in ms under *uiBacklashTimeDown*. Since it is not known at an arbitrary point after a blind movement of an arbitrary length whether part of the backlash has already been traveled, the most secure way to measure the backlash of the downward movement or its travel time is when the blind has initially been driven fully upward. A further important parameter is the timespan of the subsequent turning of the slats from the "Inward low point" to the "Outward low point". This time is to be entered as *uiTurningTimeDown* in ms in the function block.

Upward travel profile:



More detailed explanations of the terms "backlash" and "turning" are given here in the upward movement:

The circumstances are similar to the downward movement described above: the blind normally describes its upward movement with the slat low point directed inwards, as in fig. 3.

If the blind is in an initial position with the low point directed outwards (i.e. after the conclusion of a downward movement), then a certain time elapses after a new upward movement begins before the slats start to turn from the "Outward low point" to the "Inward low point". During this time the slat angle does not change; the blind only drives upward (fig. 1 and fig. 2). Also this time is an important parameter for the movement calculation and is entered in the function block in ms under *uiBacklashTimeUp*. Since it is not known at an arbitrary point after a blind movement of an arbitrary length whether part of the backlash has already been traveled, the most secure way to measure the backlash of the upward movement or its travel time is when the blind has initially been driven fully

downward. A further important parameter is the time interval of the subsequent turning of the slats from the "Outward low point" to the "Inward low point". This time is to be entered as *uiTurningTimeUp* in ms in the function block.

Parameterization

For the calculation of the (height) position and the slat angle, the following times now have to be determined for both the upward and downward movement:

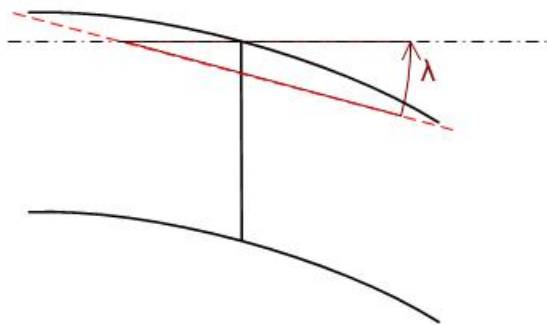
- the travel time of the backlash (*uiBacklashTimeUp* / *uiBacklashTimeDown* in ms)
- the turning duration (*uiTurningTimeUp* / *uiTurningTimeDown* in ms)
- the total travel time (*udiTotalTimeUp* / *udiTotalTimeDown* in ms)

Furthermore the following are required for the calculation:

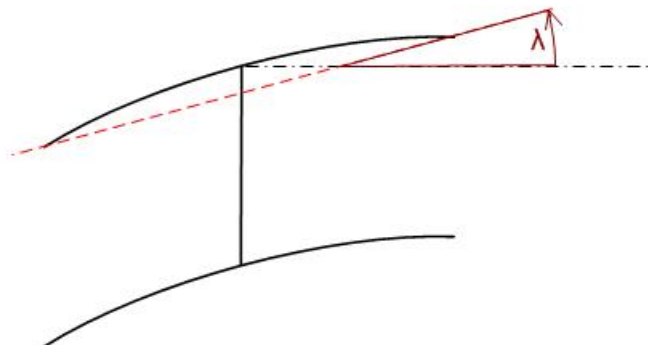
- the total extended blind height (*uiBlindHeight* in mm)
- the highest slat angle after turning upwards (*iAngleLimitUp* in degrees)
- the lowest slat angle after turning downwards (*iAngleLimitDown* in degrees)

The slat angle λ is defined by a notional straight line through the end points of the slat to the horizontal.

louvre angle $\lambda < 0$



louvre angle $\lambda > 0$



Functioning

The function block fundamentally controls the blind via the information from the positioning telegram [stSunblind](#) [► 236]. If automatic mode is active (*bManualMode*=FALSE), the current position and slat angle are always approached, and changes are immediately taken into account. The positioning to the height has priority: the entered height is approached first and then the slat angle. For reasons of the simplicity the position error due to the angle movement is disregarded. In manual mode (*bManualMode*=TRUE) the commands *bManUp* and *bManDown* control the blind.

Referencing

Secure referencing is ensured if the blind is driven upward for longer than its complete travel-up time. The position is then in any case "0" and the slat angle is at its maximum. Since a blind positioning without encoder is naturally always error-prone, it is important to reference automatically as often as possible: every time the position "0" is to be approached (the angle does not matter), the blind first moves up normally with continuous position calculation. Once the calculated position value 0% is reached, the output *bBlindUp* continues to be held for the complete travel-up time + 5s.


For reasons of flexibility there are now two possibilities to interrupt the referencing procedure: until the calculated 0% position is reached, a change in position continues to be assumed and executed. Once this 0% position is reached, the blind can still be moved with the manual "blind down" command. These two sensible limitations make it necessary for the user to ensure that the blind is securely referenced as often as possible.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bEnable           : BOOL;
stSunblind       : ST_BARSunblind;
```

eDataSecurityType: if `eDataSecurityType:= eDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not internally released.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bEnable: enable input for the function block. As long as this input is TRUE, the actuator function block accepts and executes commands as described above. A FALSE signal on this input resets the control outputs `bBlindUp` and `bBlindDown` and the function block remains in a state of rest.

stSunblind: positioning telegram, see [ST_BARSunblind](#) [► 236].

VAR_OUTPUT

```
bBlindUp      : BOOL;
bBlindDown    : BOOL;
uiActPosition : UINT;
uiActAngle    : UINT;
bReferencing  : BOOL;
bBusy         : BOOL;
bError        : BOOL;
udiErrorId    : UDINT;
```

bBlindUp: control output blind up.

bBlindDown: control output blind down.

uiActPosition: current position in percent.

uiActAngle: current slat angle in degrees.

bReferencing: the blind is referencing, i.e. the output `bBlindUp` is set for the complete travel-up time + 5 s. Only a manual "down" command can move the blind in the opposite direction and terminate this mode.

bBusy: a positioning or a referencing procedure is in progress.

bError: this output is switched to TRUE if the parameters entered are erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes](#) [► 237].

VAR_IN_OUT

In order for the registered parameters to be retained beyond a controller failure, it is necessary to declare them as In-Out variables. A reference variable is then assigned to them in the program. Each change of the value of these reference variables is persistently stored in the function block and written back to the reference variable following a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

```
uiBacklashTimeUp   : UINT;
uiBacklashTimeDown : UINT;
uiTurningTimeUp    : UINT;
```

```

uiTurningTimeDown : UINT;
udiTotalTimeUp : UDINT;
udiTotalTimeDown : UDINT;
uiBlindHeight : UINT;
iAngleLimitUp : INT;
iAngleLimitDown : INT;
    
```

uiBacklashTimeUp: time to traverse the backlash in the upward direction in ms.

uiBacklashTimeDown: time to traverse the backlash in the downward direction in ms.

uiTurningTimeUp: time for turning the slats in the upward direction in ms.

uiTurningTimeDown: time for turning the slats in the downward direction in ms.

udiTotalTimeUp: total time for driving up in ms.

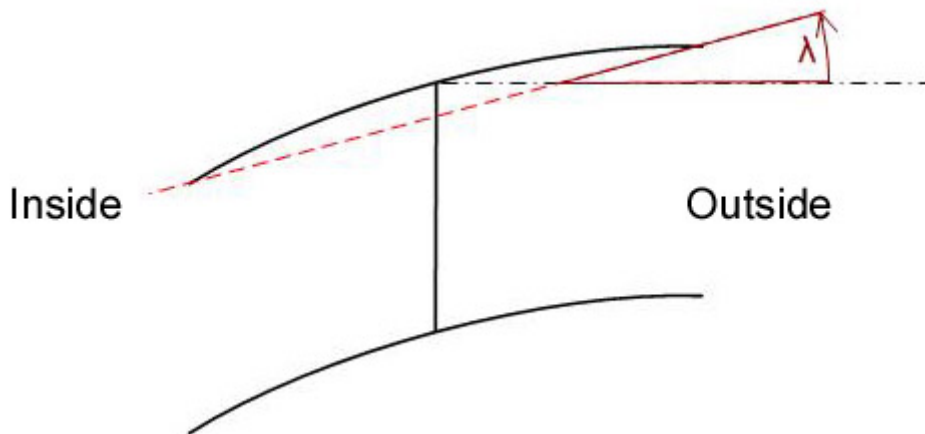
udiTotalTimeDown: total time for driving down in ms.

uiBlindHeight: blind height in mm.

iAngleLimitUp: highest position of the slats in degrees.

This position is reached once the blind has moved to the top position.

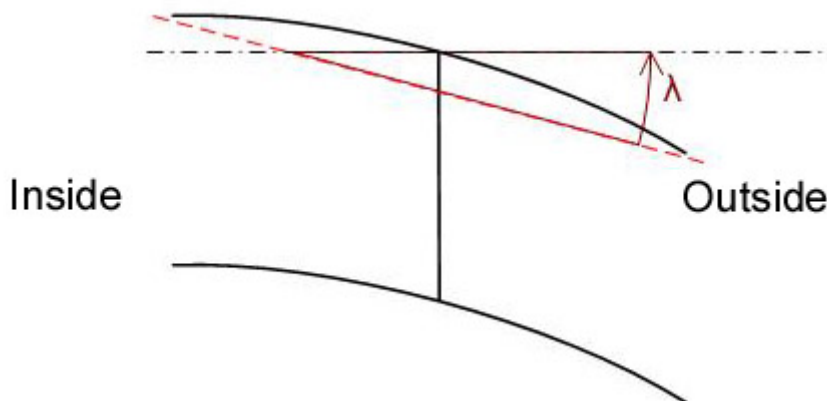
The slat angle λ , as defined above, is then typically greater than zero.



iAngleLimitDown: lowest position of the slats in degrees.

This position is reached once the blind has moved to the bottom position.

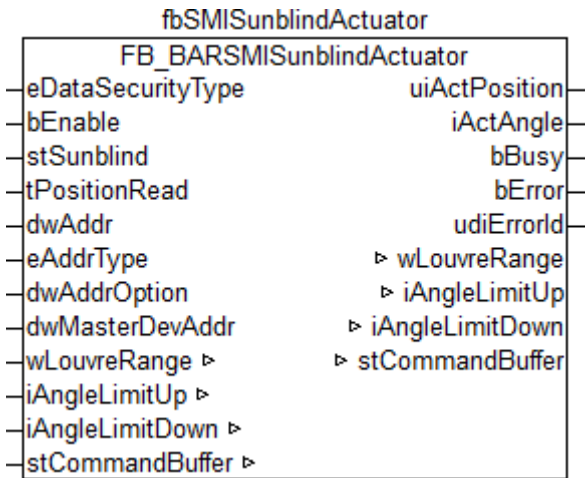
The slat angle λ , as defined above, is then typically less than zero.



3.3.4.15 FB_BARSMISunblindActuator

This function block works like the [FB_BARSunblindActuator](#) [► 185] only with the difference that an SMI motor is controlled directly.

Required library is the [TwinCAT 2 PLC Lib: TcSMI](#).



Download PLC export file: [https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/](https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659719179.zip)

11659719179.zip  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659719179.zip>

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bEnable           : BOOL;
stSunblind       : ST_BARSunblind;
tPositionRead    : TIME;
dwAddr           : DWORD;
eAddrType        : E_SMIAddrType;
dwAddrOption     : DWORD;
dwMasterDevAddr  : DWORD;
```

eDataSecurityType: if `eDataSecurityType := eDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instanced once in the main program, which is called cyclically. Otherwise, the instanced FB is not internally released.

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bEnable: enable input for the function block. If this input is TRUE, the actuator function block accepts and executes commands as described above. A FALSE signal at this input causes the function block to remain in an idle state.

stSunblind: positioning telegram, see [ST_BARSunblind](#) [► 236].

tPositionRead: interval at which the current position is read from the SMI motor when positioning commands are processed.

dwAddr: manufacturer code (0-15), address of a device (0-15), bit field (16 bits) for the group addressing or slave ID (32-bit key ID). This input has no meaning if a collective call (broadcast) is sent.

eAddrType: defines whether the `dwAddr` input is to be evaluated as a manufacturer code, the address of a device, for group addressing or as a slave ID.

dwAddrOption: if the SMI device is addressed by slave ID (*eAddrType = eSMIAAddrTypeSlaveId*), then the manufacturer code must be specified via this input.

dwMasterDevAddr: address (0-15) of the SMI motor from which the current position is to be read if *eAddrType != eSMIAAddrTypeAddress*.

VAR_OUTPUT

```
uiActPosition: UINT;
iActAngle    : INT;
bBusy        : BOOL;
bError       : BOOL;
udiErrorId   : UDINT;
```

uiActPosition: current position in percent.

iActAngle: current slat angle in degrees.

bBusy: a positioning process is taking place.

bError: this output is switched to TRUE if the parameters entered are erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes](#).

VAR_IN_OUT

```
wLouvreRange : WORD;
iAngleLimitUp : INT;
iAngleLimitDown : INT;
stCommandBuffer : ST_SMICommandBuffer;
```

wLouvreRange: number of steps of the SMI motor from the slats "low point inside" to the "low point outside".

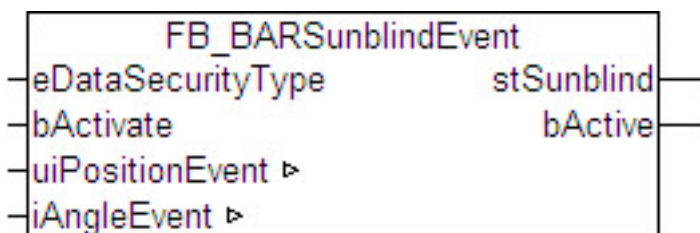
iAngleLimitUp: highest position of the slats in degrees (-360..360).

iAngleLimitDown: lowest position of the slats in degrees (-360..360).

stCommandBuffer: reference to the structure for communication (buffer) with the [FB_KL6831KL6841Communication\(\)](#) function block.

3.3.4.16 FB_BARSunblindEvent

This function block serves to preset the position and angle for any desired event. It can be used, for example, in order to drive to a parking position or to drive the blind upward for maintenance.




The function is activated via the input *bActivate*. If this is the case, then the active flag is set in the positioning telegram (*bActive* in *stSunblind*) at the output *stSunblind* [► 236] and the values *uiPositionEvent* for the blind height in % and *iAngleEvent* for the slat angle in degrees, which are entered in the In-Out variables, are passed on in this telegram. If the function is no longer active due to the resetting of *bActive*, then the active flag in the positioning telegram *stSunblind* [► 236] is reset and the positions for height and angle are set to "0". If the priority function block is used, then a function with a lower priority can take over the control.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bActivate         : BOOL;
```

eDataSecurityType: if `eDataSecurityType:= eDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not internally released.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bActivate: a TRUE signal on this input activates the function block and transfers the entered setpoints together with the active flag in the positioning telegram `ST_BARSunblind` [► 236]. A FALSE signal resets the active flag again and sets position and angle to zero.

VAR_OUTPUT


```
stSunblind : ST_BARSunblind;
bActive    : BOOL;
```

bActive : corresponds to the boolean value `bActive` in the blind telegram `ST_BARSunblind` [► 236] and is solely used to indicate whether the function block sends an active telegram.

stSunblind: output structure of the blind positions, see `ST_BARSunblind` [► 236]

VAR_IN_OUT

In order for the registered parameters to be retained beyond a controller failure, it is necessary to declare them as In-Out variables. A reference variable is then assigned to them in the program. Each change of the value of these reference variables is persistently stored in the function block and written back to the reference variable following a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.



Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>.

```
uiPositionPark : UINT;
iAnglePark     : INT;
```

uiPositionEvent: height position of the blind in % in case of activation.

iAngleEvent: slat angle of the blind in degrees in case of activation.

Documents about this

-  [example_persistent_e.zip \(Resources/zip/11659714827.zip\)](#)
-  [example_persistent_e.zip \(Resources/zip/11659714827.zip\)](#)

3.3.4.17 FB_BARSunblindPrioritySwitch

Priority controller for up to 9 positioning telegrams (`stSunblind_Prio1` ... `stSunblind_Prio9`) of the type `ST_BARSunblind` [► 236].

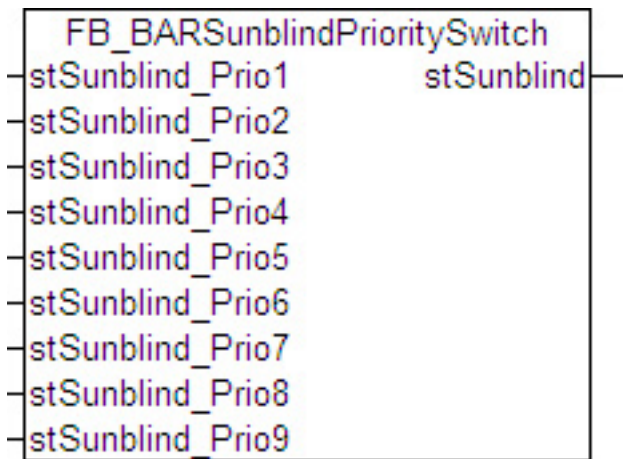


Fig. 8: FB_BARSunblindPrioritySwitch

Structure of the blind positioning telegram [ST_BARSunblind](#) [► 236].

```

TYPE ST_BARSunblind:
STRUCT
    uiPosition    : UINT;
    iAngle        : INT;
    bManUp        : BOOL;
    bManDown      : BOOL;
    bManualMode   : BOOL;
    bActive       : BOOL;
END_STRUCT
END_TYPE

```

Up to 9 positioning telegrams from different control function blocks can be applied to this function block. The telegram on *stSunblind_Prio1* has the highest priority and that on *stSunblind_Prio9* the lowest. The active telegram with the highest priority is output at *stSunblind*. "Active" means that the variable *bActive* is set within the structure of the positioning telegram.



This function block is to be programmed in such a way that one of the applied telegrams is always active. If no telegram is active, an empty telegram is output, i.e. *uiPosition=0*, *iAngle=0*, *bManUp=FALSE*, *bManDown=FALSE*, *bManualMode=FALSE*, *bActive=FALSE*. Since the blind function block [FB_BARSunblindActuator](#) [► 185] or the roller blind function block [FB_BARRollerblind](#) [► 200] does not take account of the flag *bActive*, this telegram would be interpreted as movement command to position "0", i.e. fully open. The absence of an active telegram therefore does not represent a safety risk for the blind.

VAR_INPUT

```

stSunblind_Prio1 : ST_BARSunblind;
stSunblind_Prio2 : ST_BARSunblind;
stSunblind_Prio3 : ST_BARSunblind;
stSunblind_Prio4 : ST_BARSunblind;
stSunblind_Prio5 : ST_BARSunblind;
stSunblind_Prio6 : ST_BARSunblind;
stSunblind_Prio7 : ST_BARSunblind;
stSunblind_Prio8 : ST_BARSunblind;
stSunblind_Prio9 : ST_BARSunblind;

```

stSunblind_Prio1..stSunblind_Prio9 : positioning telegrams available for selection. *stSunblind_Prio1* has the highest priority and *stSunblind_Prio9* the lowest.

VAR_OUTPUT

```

stSunblind : ST_BARSunblind;

```

stSunblind: resulting positioning telegram.

3.3.4.18 FB_BARRollerBlind

This function block is used for positioning of a blind via two outputs: up and down. The blind can be driven to any desired position with the positioning telegram `stSunblind` [▶ 236]. On top of that, the positioning telegram `stSunblind` [▶ 236] also contains manual commands with which the blind can be moved individually to certain positions. These manual commands are controlled by the function block `FB_BARSunblindSwitch` [▶ 206].



Structure of the blind positioning telegram `stSunblind` [▶ 236].

```

TYPE ST_BARSunblind:
STRUCT
    uiPosition      : UINT;
    iAngle          : INT;
    bManUp          : BOOL;
    bManDown        : BOOL;
    bManualMode     : BOOL;
    bActive         : BOOL;
END_STRUCT
END_TYPE

```

The current height position and the slat angle are not read in by an additional encoder, but are determined internally by the runtime of the blind.

The two different runtime parameters `udiTotalTimeUp` (runtime blind up in ms) and `udiTotalTimeDown` (runtime blind down in ms) take into account the different travel characteristics.

Functioning

The function block fundamentally controls the blind via the information from the positioning telegram `stSunblind` [▶ 236]. If automatic mode is active (`bManualMode=FALSE`), the current position is always approached, and changes are immediately taken into account. In manual mode (`bManualMode=TRUE`) the commands `bManUp` and `bManDown` control the blind.

Referencing

Secure referencing is ensured if the blind is driven upward for longer than its complete travel-up time. The position is then always "0". Since a blind positioning without encoder is always error-prone by nature, it is important to reference automatically as often as possible: every time the position "0" is to be approached, the blind first moves up normally with continuous position calculation. Once the calculated position value 0% is reached, the output `bBlindUp` continues to be held for the complete travel-up time + 5s.

For reasons of flexibility there are now two possibilities to interrupt the referencing procedure: until the calculated 0% position is reached, a change in position continues to be assumed and executed. Once this 0% position is reached, the blind can still be moved with the manual "blind down" command. These two sensible limitations make it necessary for the user to ensure that the blind is securely referenced as often as possible.

VAR_INPUT


```

eDataSecurityType: E_HVACDataSecurityType;
bEnable          : BOOL;
stSunblind       : ST_BARSunblind;

```


eDataSecurityType: if `eDataSecurityType:= eDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not internally released.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bEnable: enable input for the function block. As long as this input is TRUE, the actuator function block accepts and executes commands as described above. A FALSE signal on this input resets the control outputs `bBlindUp` and `bBlindDown` and the function block remains in a state of rest.

stSunblind: positioning telegram, see [ST_BARSunblind](#) [► 236].

VAR_OUTPUT

```
bBlindUp      : BOOL;
bBlindDown    : BOOL;
uiActPosition : UINT;
bReferencing  : BOOL;
bBusy         : BOOL;
bError        : BOOL;
udiErrorId    : UDINT;
```

bBlindUp: control output blind up.

bBlindDown: control output blind down.

uiActPosition: current position in percent.

bReferencing: the blind is referencing, i.e. the output `bBlindUp` is set for the complete travel-up time + 5 s. Only a manual "down" command can move the blind in the opposite direction and terminate this mode.


bBusy: a positioning or a referencing procedure is in progress.

bError: this output is switched to TRUE if the parameters entered are erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes](#) [► 237].

VAR_IN_OUT

In order for the registered parameters to be retained beyond a controller failure, it is necessary to declare them as In-Out variables. A reference variable is then assigned to them in the program. Each change of the value of these reference variables is persistently stored in the function block and written back to the reference variable following a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

```
udiTotalTimeUp   : UDINT;
udiTotalTimeDown : UDINT;
```

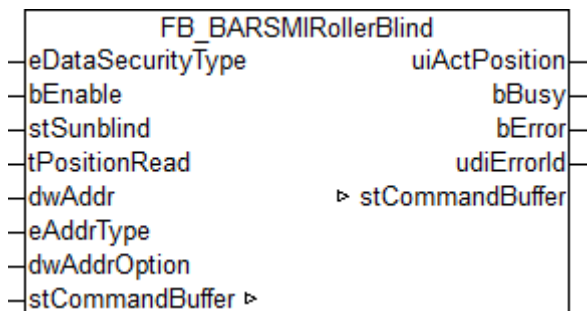
udiTotalTimeUp: total time for driving up in ms.

udiTotalTimeDown: total time for driving down in ms.

3.3.4.19 FB_BARSMIRollerBlind

This function block works like the [FB_BARRollerBlind \[► 200\]](#) only with the difference that an SMI motor is controlled directly.

Required library is the [TwinCAT 2 PLC Lib: TcSMI](#).



Download PLC export file: <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659720587/.zip>

<https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659720587/.zip>

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bEnable           : BOOL;
stSunblind       : ST_BARSunblind;
tPositionRead    : TIME;
dwAddr           : DWORD;
eAddrType        : E_SMIAddrType;
dwAddrOption     : DWORD;
dwMasterDevAddr  : DWORD;
```

eDataSecurityType: if eDataSecurityType:= *eDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise, the instanced FB is not internally released.

If eDataSecurityType:= *eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType:= *eHVACDataSecurityType_Persistent*. It would lead to early wear of the flash memory.

bEnable: enable input for the function block. As long as this input is TRUE, the actuator function block accepts and executes commands as described above. A FALSE signal at this input causes the function block to remain in an idle state.

stSunblind: positioning telegram, see [ST_BARSunblind \[► 236\]](#).

tPositionRead: interval at which the current position is read from the SMI motor when positioning commands are processed.

dwAddr: [manufacturer code](#) (0-15), address of a device (0-15), bit field (16 bits) for the group addressing or slave ID (32-bit key ID). This input has no meaning if a collective call (broadcast) is sent.

eAddrType: defines whether the *dwAddr* input is to be evaluated as a manufacturer code, the address of a device, for group addressing or as a slave ID.

dwAddrOption: if the SMI device is addressed by slave ID (*eAddrType* = *eSMIAddrTypeSlaveId*), then the manufacturer code must be specified via this input.

dwMasterDevAddr: address (0-15) of the SMI motor from which the current position is to be read if *eAddrType* != *eSMIAddrTypeAddress*.

VAR_OUTPUT

```
uiActPosition: UINT;
bBusy        : BOOL;
bError       : BOOL;
udiErrorId   : UDINT;
```

uiActPosition: current position in percent.

bBusy: a positioning process is taking place.

bError: this output is switched to TRUE if the parameters entered are erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See error codes.

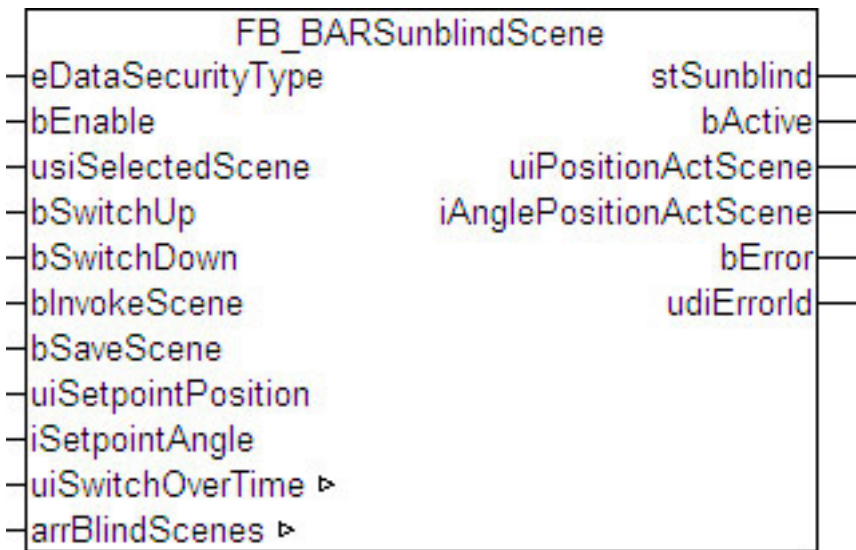
VAR_IN_OUT

```
stCommandBuffer : ST_SMICommandBuffer;
```

stCommandBuffer: reference to the structure for communication (buffer) with the FB_KL6831KL6841Communication() function block.

3.3.4.20 FB_BARSunblindScene

This function block represents an extension of the manual operation FB_BARSunblindSwitch [▶ 206] by a scene memory and a call function. The blind actuator FB_BARSunblindActuator [▶ 185] or the roller blind actuator FB_BARRollerblind [▶ 200] can thus be controlled in manual operation mode and can also drive directly to previously saved positions (scenes). Up to 21 scenes can be saved.



Structure of the blind positioning telegram stSunblind [▶ 236].

```
TYPE ST_BARSunblind:
STRUCT
    uiPosition : UINT;
    iAngle     : INT;
    bManUp     : BOOL;
    bManDown   : BOOL;
    bManualMode : BOOL;
    bActive    : BOOL;
END_STRUCT
END_TYPE
```

Mode of operation

In manual mode, the function block controls the blind function block FB_BARSunblindActuator [▶ 185] or the roller blind function block FB_BARRollerblind [▶ 200] via the command inputs *bSwitchUp* and *bSwitchDown*; *bSwitchUp* has priority. The commands are passed on to the respective commands *bManUp* and *bManDown* of the positioning telegram. If a command input is activated for longer than the entered time

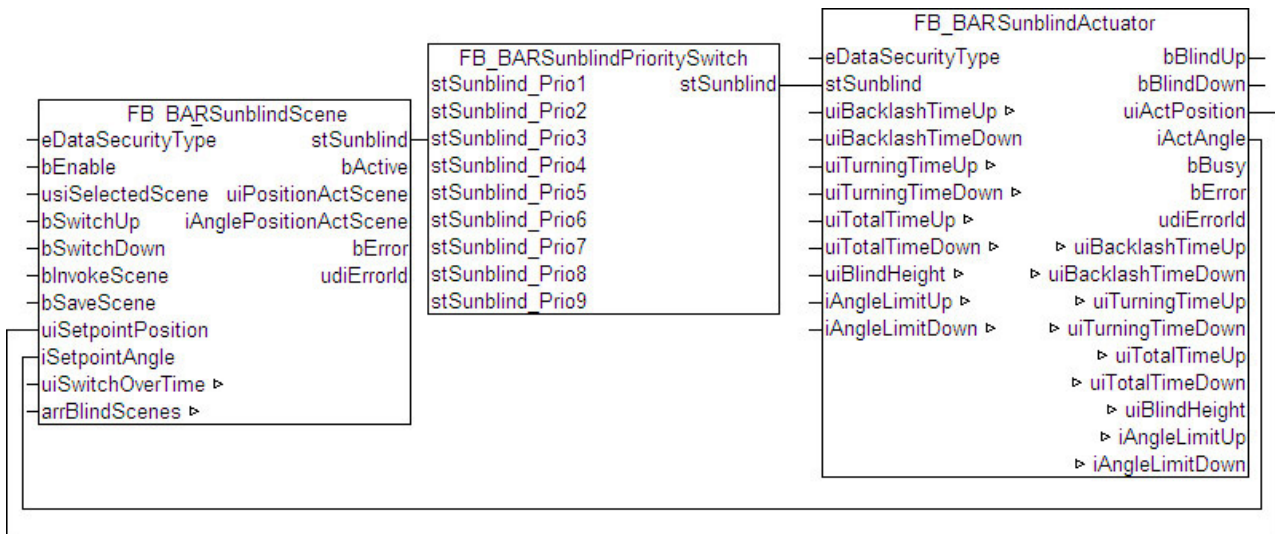
uiSwitchOverTime (in ms), the corresponding control command latches. Activating a command input again clears this latching.

A rising edge at *bSaveScene* saves the current position and the slat angle in the scene selected at *usiSelectedScene*. This procedure is possible at any time, even during active positioning. The selected scene is called with *bInvokeScene*, i.e. the saved position and angle values are approached.

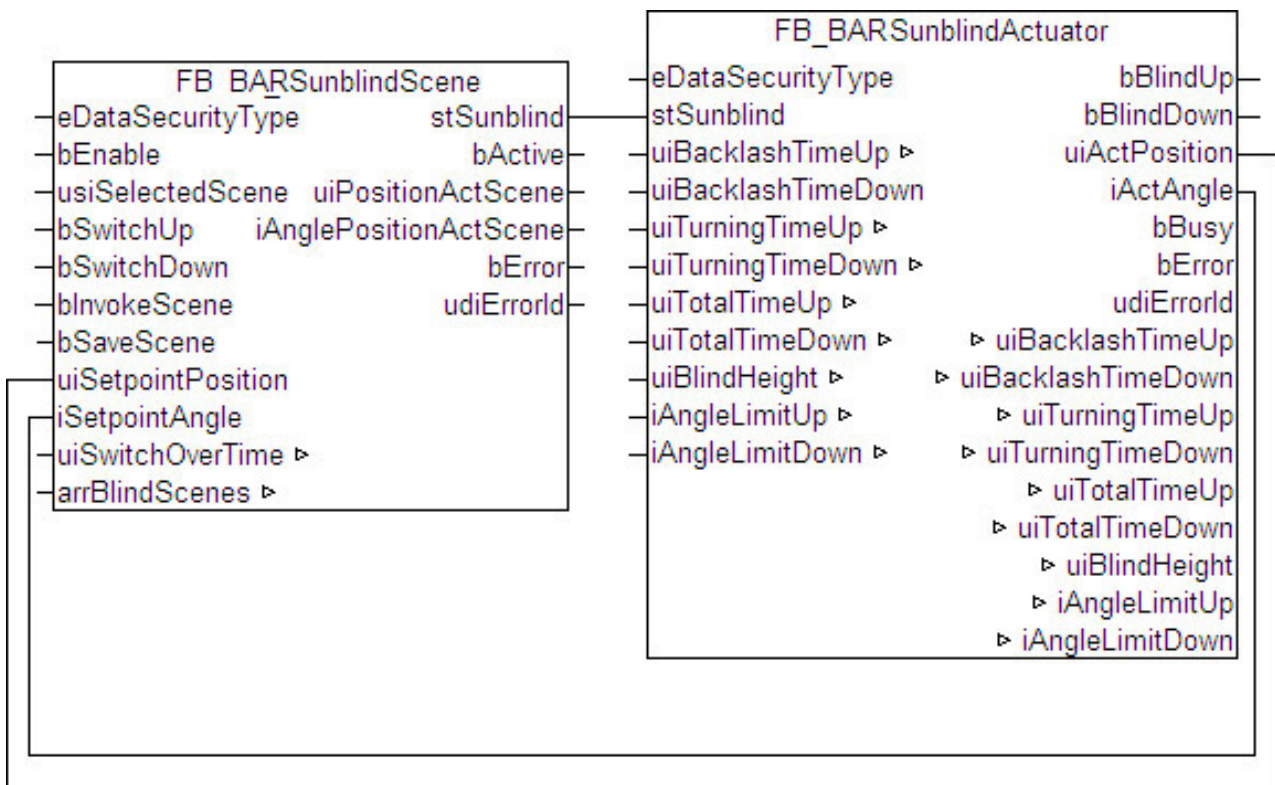
Linking to the blind function block

Like the "normal" manual control function block, [FB_BARSunblindSwitch \[▶ 206\]](#), the scene selection function block can be connected to the blind function block either directly or via an upstream priority controller [FB_BARSunblindPrioritySwitch \[▶ 198\]](#). The connection is made via the positioning telegram [stSunblind \[▶ 236\]](#). Furthermore the scene function block requires the current positions from the blind function block for the reference blind:

Use of a priority controller:



Direct connection:




VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
usiSelectedScene  : USINT;
bSwitchUp        : BOOL;
bSwitchDown      : BOOL;
bInvokeScene     : BOOL;
bSaveScene       : BOOL;
uiSetpointPosition: UINT;
iSetpointAngle   : INT;
```

eDataSecurityType: if `eDataSecurityType:= eDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not internally released.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bEnable : the function block has no function if this input is FALSE. 0 is output for the position and the angle in the positioning telegram `stSunblind` [► 236] - `bManualMode` and `bActive` are set to FALSE. For a connection with priority controller this means that another functionality takes over control of the blind. Conversely, a direct connection allows the blind to drive directly to the 0 position, i.e. fully up, since the actuator function block does not evaluate the bit `bActive` itself.

usiSelectedScene : selected scene which should either be saved (`bSaveScene`) or called (`bInvokeScene`).

bSwitchUp: command input blind up.

bSwitchDown: command input blind down.

bInvokeScene : call selected scene.

bSaveScene: save selected scene.

uiSetpointPosition: set position in % that is to be saved in the selected scene. This must be linked to the actual position of the actuator function block `FB_BARSunblindActuator` [► 185] or `FB_BARRollerblind` [► 200] of the reference blind/roller blind, in order to be able to save a position that was previously approached manually.

iSetpointAngle : ditto slat angle in degrees.

VAR_OUTPUT

```
stSunblind       : ST_BARSunblind;
bActive          : BOOL;
uiPositionActScene : UINT;
iAnglePositionActScene: INT;
bError           : BOOL;
udiErrorId       : UDINT;
```

stSunblind: positioning telegram, see `ST_BARSunblind` [► 236].

bActive : corresponds to the boolean value `bActive` in the blind telegram `ST_BARSunblind` [► 236] and is solely used to indicate whether the function block sends an active telegram.

uiPositionActScene : indicates the saved relative blind height position in % for the currently selected scene.

iAnglePositionActScene : ditto slat angle in degrees.


bError: this output is switched to TRUE if the parameters entered are erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes \[▶ 237\]](#).

i If an error should occur, then this automatic function is deactivated and position and angle are set to 0. This means that if a priority controller is in use, another function with a lower priority (see [Overview](#)) automatically takes over control of the blind. In the case of a direct connection, conversely, the blind will drive to position/angle 0.

VAR_IN_OUT

In order for the registered parameters to be retained beyond a controller failure, it is necessary to declare them as In-Out variables. A reference variable is then assigned to them in the program. Each change of the value of these reference variables is persistently stored in the function block and written back to the reference variable following a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.



Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>.

```
uiSwitchOverTime : UINT;
arrBlindScenes   : ARRAY[0..20] OF ST_BARSunblindScene;
```

uiSwitchOverTime : time in milliseconds until the corresponding manual command in the positioning telegram [stSunblind \[▶ 236\]](#) switches to latching mode, if the command input is activated permanently.

arrBlindScenes : table with the scene entries of the type [ST_BARSunblindScene \[▶ 236\]](#). Up to 21 scenes can be saved: 0..20.

Documents about this

-  [example_persistent_e.zip \(Resources/zip/11659714827.zip\)](#)
-  [example_persistent_e.zip \(Resources/zip/11659714827.zip\)](#)

3.3.4.21 FB_BARSunblindSwitch

With the help of this function block the blind controller [FB_BARSunblindActuator \[▶ 185\]](#) or the roller blind controller [FB_BARRollerblind \[▶ 200\]](#) can be controlled in manual operation mode. The connection takes place via the positioning telegram [stSunblind \[▶ 236\]](#) either directly or with an additional priority controller.

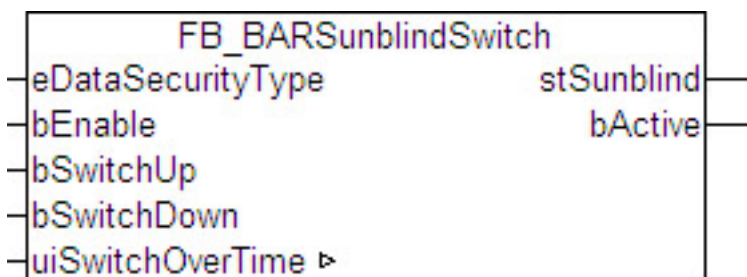


Fig. 9: FB_BARSunblindSwitch

Structure of the blind positioning telegram [stSunblind \[▶ 236\]](#).

```
TYPE ST_BARSunblind:
STRUCT
  uiPosition : UINT;
  iAngle     : INT;
  bManUp     : BOOL;
  bManDown   : BOOL;
  bManualMode : BOOL;
```

```

    bActive      : BOOL;
END_STRUCT
END_TYPE
    
```

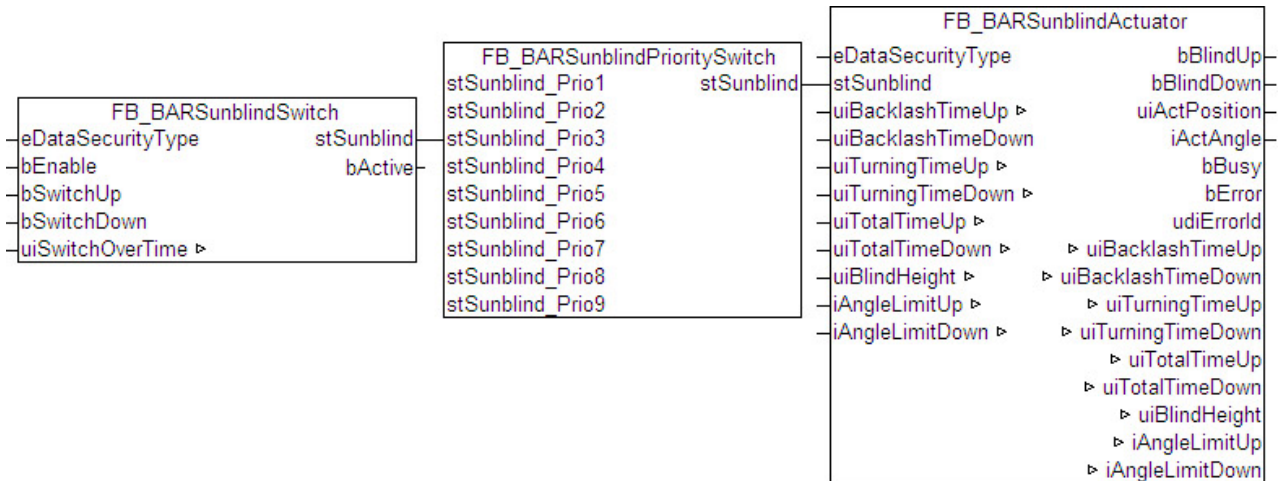
Operation

In manual mode, the function block controls the blind function block [FB_BARSunblindActuator \[▸ 185\]](#) or the roller blind function block [FB_BARRollerblind \[▸ 200\]](#) via the command inputs *bSwitchUp* and *bSwitchDown*; *bSwitchUp* has priority. The commands are passed on to the respective commands *bManUp* and *bManDown* of the positioning telegram. If a command input is activated for longer than the entered time *uiSwitchOverTime* (in ms), the corresponding control command latches. Activating a command input again releases this latch.

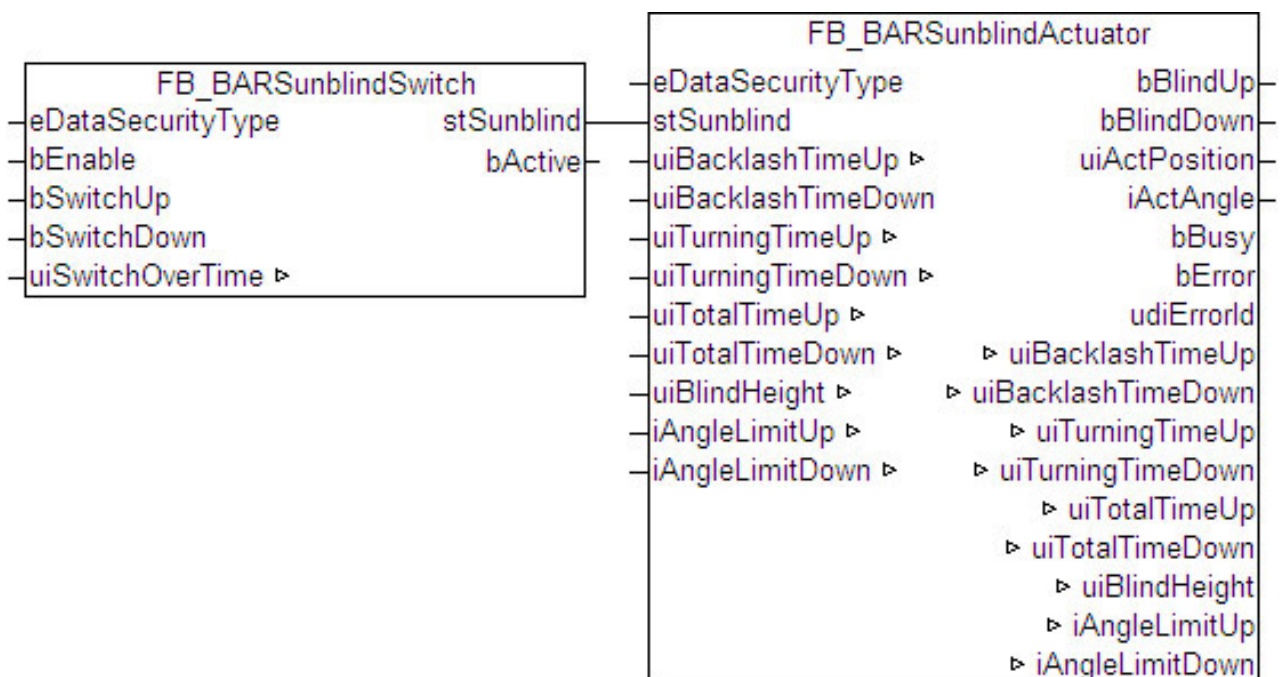
Linking to the blind function block

The manual control function block can be connected to the blind function block either directly or via an upstream priority controller [FB_BARSunblindPrioritySwitch \[▸ 198\]](#). The connection is made via the positioning telegram [stSunblind \[▸ 236\]](#).

Use of a priority controller:



Direct connection:




VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bEnable           : BOOL;
bSwitchUp        : BOOL;
bSwitchDown      : BOOL;
```

eDataSecurityType: if `eDataSecurityType:= eDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instantiated once in the main program, which is called cyclically. Otherwise the instantiated FB is not internally released.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bEnable : the function block has no function if this input is FALSE. 0 is output for the position and the angle in the positioning telegram `stSunblind` [► 236] - `bManualMode` and `bActive` are set to FALSE. For a connection with priority controller this means that another functionality takes over control of the blind. Conversely, a direct connection allows the blind to drive directly to the 0 position, i.e. fully up, since the actuator function block does not evaluate the bit `bActive` itself.

bSwitchUp: command input blind up.

bSwitchDown: command input blind down.

VAR_OUTPUT

```
stSunblind : ST_BARSunblind;
bActive    : BOOL;
```

stSunblind: positioning telegram, see `ST_BARSunblind` [► 236].

bActive : corresponds to the boolean value `bActive` in the blind telegram `ST_BARSunblind` [► 236] and is solely used to indicate whether the function block sends an active telegram.

VAR_IN_OUT



In order for the registered parameters to be retained beyond a controller failure, it is necessary to declare them as In-Out variables. A reference variable is then assigned to them in the program. Each change of the value of these reference variables is persistently stored in the function block and written back to the reference variable following a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

```
uiSwitchOverTime : UINT;
```

uiSwitchOverTime : time in milliseconds until the corresponding manual command in the positioning telegram `stSunblind` [► 236] switches to latching mode, if the command input is activated permanently.

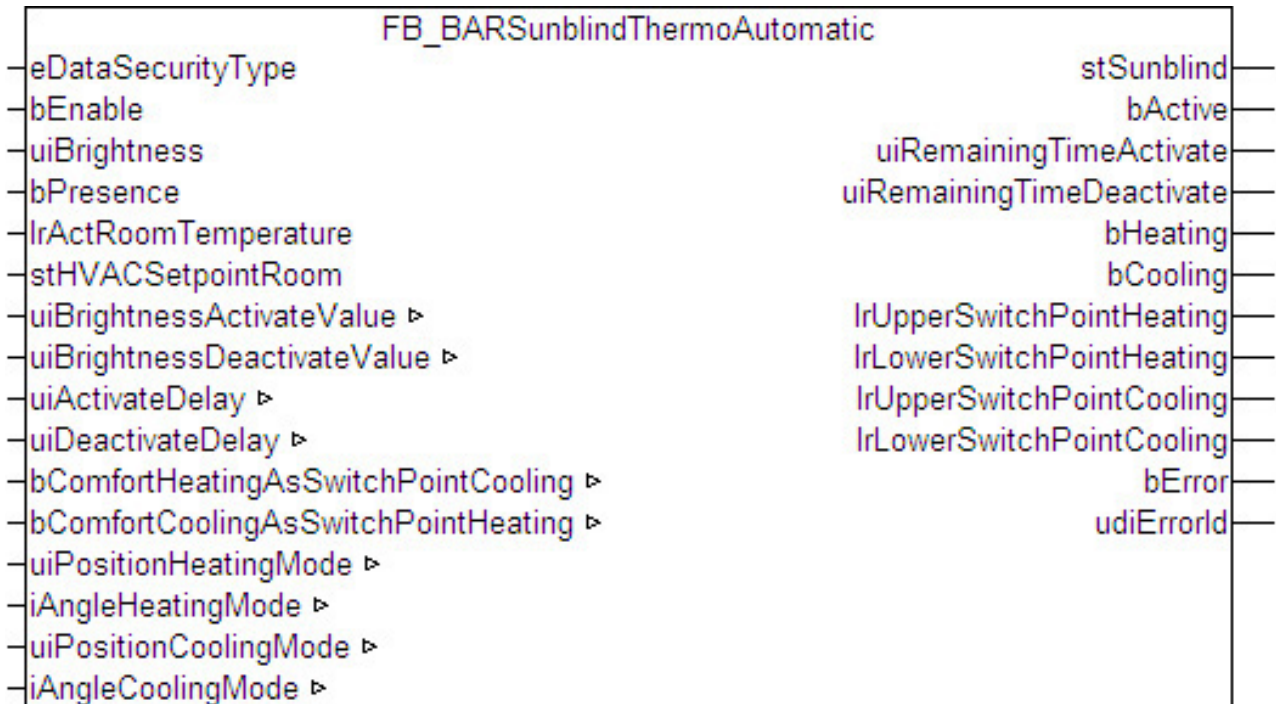
Documents about this

-  [example_persistent_e.zip \(Resources/zip/11659714827.zip\)](#)
-  [example_persistent_e.zip \(Resources/zip/11659714827.zip\)](#)

3.3.4.22 FB_BARSunblindThermoAutomatic

This function block controls the blind in relation to the temperature inside the room if the room is **unoccupied**.

The objective is to block radiant heat when cooling and to increase the input of heat by opening the blinds when heating.



The thermal automatic is active only if there is sufficient sunshine and the room concerned is unoccupied. To this end, the external brightness value *uiBrightness* [lux] must have exceeded the activation limit value *uiBrightnessActivateValue* for the time *uiActivateDelay* (in seconds) and the presence input *bPresence* must be FALSE. The automatic function is deactivated again if either presence is signaled (*bPresence* = TRUE) or the external brightness value *uiBrightness* falls below the value *uiBrightnessDeactivateValue* for the time *uiDeactivateDelay*.

A structure of the type *ST_BARSetpointRoom*, which carries the information about the temperature switching points for the air conditioning of the room, is to be applied at the input *stBARSetpointRoom*:

```

TYPE ST_BARSetpointRoom :
STRUCT
    stBARSetpointRoom_ComfortHeat      : REAL:= 21.0;
    stBARSetpointRoom_PreComfortHeat   : REAL:= 19.0;
    stBARSetpointRoom_EconomyHeat      : REAL:= 15.0;
    stBARSetpointRoom_ProtectionHeat   : REAL:= 12.0;
    stBARSetpointRoom_ComfortCool      : REAL:= 24.0;
    stBARSetpointRoom_PreComfortCool   : REAL:= 28.0;
    stBARSetpointRoom_EconomyCool      : REAL:= 35.0;
    stBARSetpointRoom_ProtectionCool   : REAL:= 40.0;
END_STRUCT
END_TYPE
    
```

Once activated, the thermo-automatic function distinguishes between 3 cases:

- Remain in the current position if the room temperature *rActRoomTemperature* lies between the values *stBARSetpointRoom_ComfortHeat* and *stBARSetpointRoom_ComfortCool*.
- Heating case if the room temperature *rActRoomTemperature* falls below the value *stBARSetpointRoom_ComfortHeat*.
- Cooling case if the room temperature *rActRoomTemperature* exceeds the value *stBARSetpointRoom_ComfortCool*.

Once the heating or the cooling case has been reached, the thermal automatic remembers this until it becomes inactive itself once again due to renewed presence detection or insufficient outdoor brightness (see above).

In both cases, however – heating or cooling – the mechanism operates like a on-off controller.

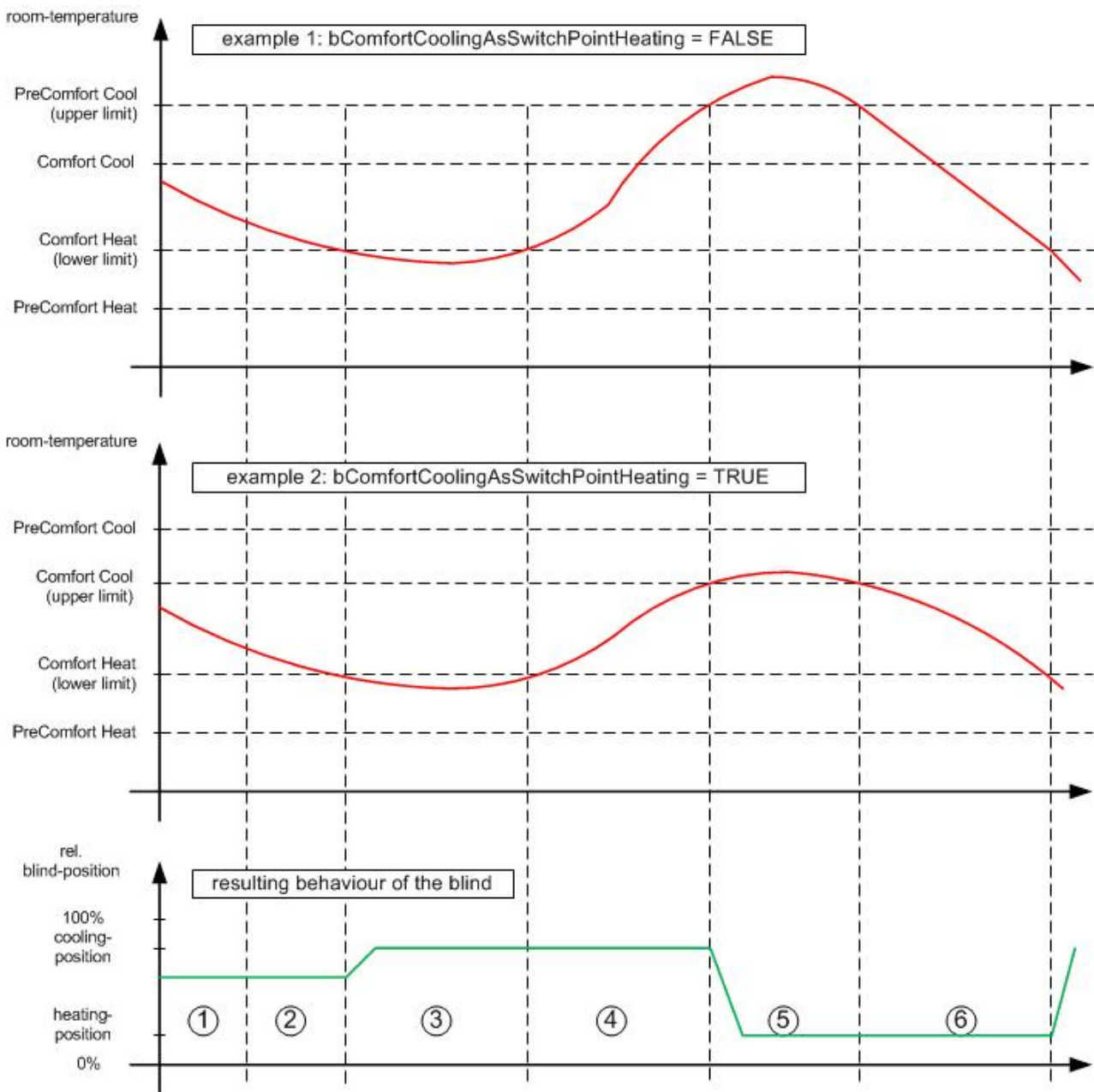
Heating case

In the heating case, the blind drives to the heating position *uiPositionHeatingMode* and *iAngleHeatingMode* if the room temperature falls below the lower limit value. If the room temperature exceeds the upper limit value, then the blind drives to the cooling position *uiPositionCoolingMode* and *iAngleCoolingMode*.

The upper limit value is selectable by the parameter *bComfortCoolingAsSwitchPointHeating* [BOOL], the lower limit value is not selectable:

	<i>bComfortCoolingAsSwitchPointHeating</i> = FALSE	<i>bComfortCoolingAsSwitchPointHeating</i> = TRUE
upper limit value	<i>stBARSetpointRoom_PreComfortCool</i>	<i>stBARSetpointRoom_ComfortCool</i>
lower limit value	<i>stBARSetpointRoom_ComfortHeat</i>	<i>stBARSetpointRoom_ComfortHeat</i>

Diagram



- ① Initial situation:
The thermo-automatic is not active yet. Another function is controlling the blind.
- ② The thermo-automatic was activated due to non-presence-detection and enough outside-brightness. But the room-temperature is still between both comfort-values, so a decision between heating or cooling cannot be made yet. Thus the blind remains in its actual position.
- ③ The temperature falls below the comfort-heat-value. The thermo-automatic switches over to the heating-mode and the blind will be driven to heating-position.
- ④ The temperature rises above the comfort-heat-value. The position of the blind will not be changed.
- ⑤ Only after reaching the upper limit (example1: PreComfortCool and example2: ComfortCool) the blind will be driven to the cooling-position.
- ⑥ The blind will stay in the cooling-position until the temperature falls below the lower setpoint (ComfortHeat) again.

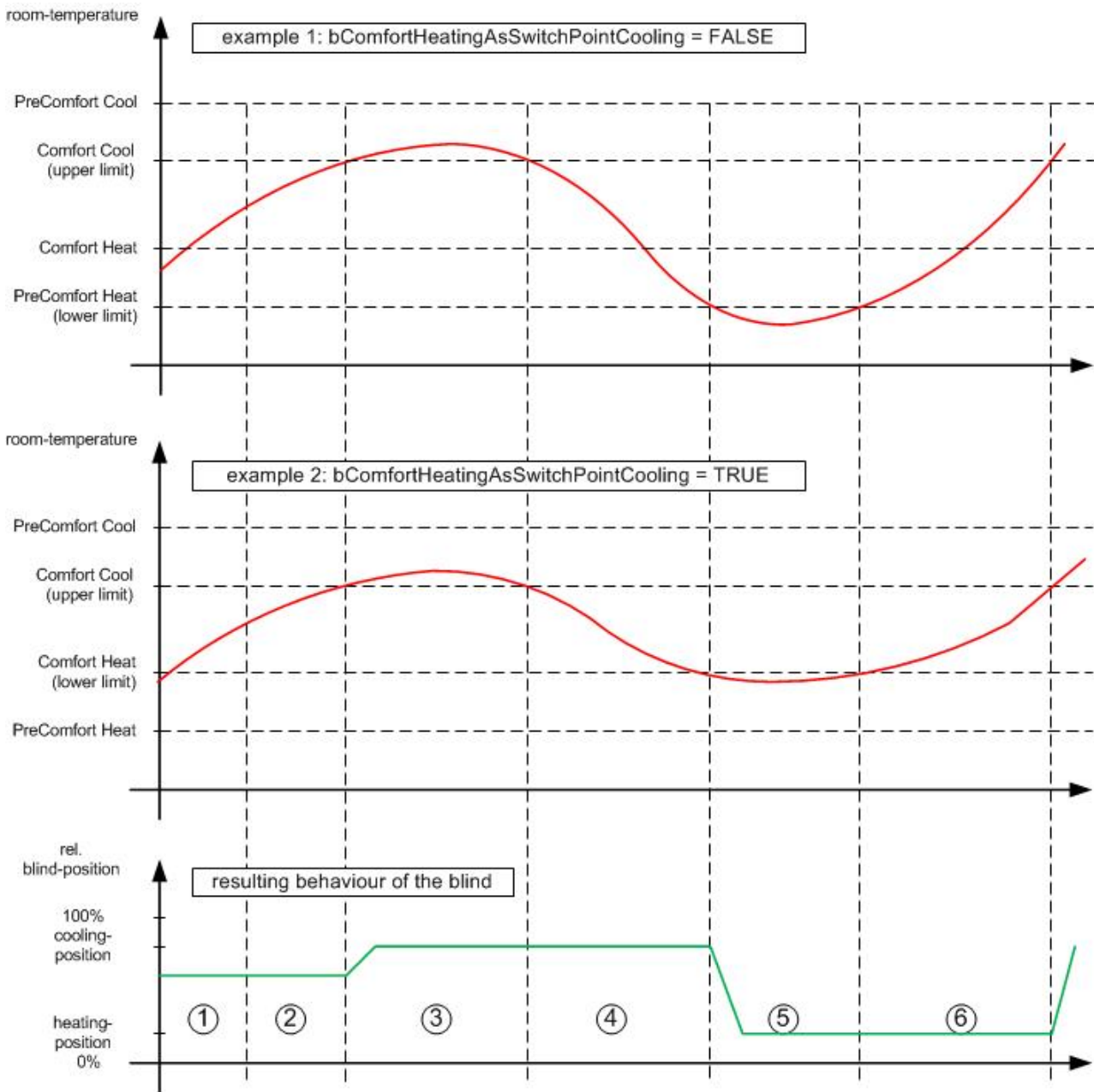
Cooling case

In the cooling case the blind drives to the cooling position *uiPositionCoolingMode* and *iAngleCoolingMode* if the room temperature exceeds the upper limit value. If the room temperature falls below the lower limit value, then the blind drives to the heating position *uiPositionHeatingMode* and *iAngleHeatingMode*.

The lower limit value is selectable by the parameter *bComfortHeatingAsSwitchPointCooling* [BOOL], the upper limit value is not selectable:

	bComfortHeatingAsSwitchPointCooling = FALSE	bComfortHeatingAsSwitchPointCooling = TRUE
upper limit value	stBARSetpointRoom_ComfortCool	stBARSetpointRoom_ComfortCool
lower limit value	stBARSetpointRoom_PreComfortHeat	stBARSetpointRoom_ComfortHeat

Diagram



- ① Initial situation:
The thermo-automatic is not active yet. Another function is controlling the blind.
- ② The thermo-automatic was activated due to non-presence-detection and enough outside-brightness. But the room-temperature is still between both comfort-values, so a decision between heating or cooling cannot be made yet. Thus the blind remains in its actual position.
- ③ The temperature rises above the comfort-cool-value. The thermo-automatic switches over to the cooling-mode and the blind will be driven to cooling-position.
- ④ The temperature falls below the comfort-cool-value. The position of the blind will not be changed.
- ⑤ Only after reaching the lower limit (example1: PreComfortHeat and example2: ComfortHeat) the blind will be driven to the heating-position.
- ⑥ The blind will stay in the heating-position until the temperature falls below the lower setpoint (ComfortCool) again.


A lower average room temperature is obtained by selecting the *Precomfort Cool* value instead of the *Comfort Cool* value as the upper hysteresis point. This could be perceived as more pleasant when re-entering the room.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
uiBrightness      : UINT;
bPresence         : BOOL;
lrActRoomTemperature: LREAL;
stBARSetpointRoom : ST_BARSetpointRoom;
```

eDataSecurityType: if `eDataSecurityType := eDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bEnable : the function block has no function if this input is FALSE. In the positioning telegram `stSunblind` [► 236], 0 is output for the position and the angle, and `bActive` is FALSE. This means that another function takes over control of the blind via the priority controller.

uiBrightness: outdoor brightness in lux.

bPresence: the automatic function is active only if the room is unoccupied, since otherwise the sun protection takes priority. If this input is FALSE, the room is considered to be unoccupied and the thermal automatic is active. A TRUE signal at `bPresence` deactivates the thermal automatic **immediately**.

lrActRoomTemperature: input for the room temperature in °C.

stBARSetpointRoom: structure of the type `ST_BARSetpointRoom` [► 209], which contains the information about the temperature switching points for the air conditioning of the room, see above.

VAR_OUTPUT

```
stSunblind          : ST_BARSunblind;
bActive             : BOOL;
uiRemainingTimeActivate : UINT;
uiRemainingTimeDeactivate: UINT;
bHeating            : BOOL;
bCooling            : BOOL;
lrUpperSwitchPointHeating: LREAL;
lrLowerSwitchPointHeating: LREAL;
lrUpperSwitchPointCooling: LREAL;
lrLowerSwitchPointCooling: LREAL;
bError              : BOOL;
udiErrorId          : UDINT;
```

stSunblind: output structure of the blind positions, see `ST_BARSunblind` [► 236].

bActive : corresponds to the boolean value `bActive` in the blind telegram `ST_BARSunblind` [► 236] and is solely used to indicate whether the function block sends an active telegram.

uiRemainingTimeActivate: the thermal automatic is active if no presence is detected AND the brightness lies above the limit value *uiBrightnessActivateValue* for the time *uiActivateDelay* (switching delay). This output indicates the remaining duration of the switching delay in seconds. This output is 0 as long as no countdown of the time is taking place.

uiRemainingTimeDeactivate: the thermal automatic is not active if presence is detected OR if the brightness lies below the limit value *uiBrightnessDeactivateValue* for the time *uiDeactivateDelay* (switching delay). This output indicates the remaining duration of the switching delay in seconds. This output is 0 as long as no countdown of the time is taking place.

bHeating : indicates whether the heating case is active.

bCooling : indicates whether the cooling case is active.

IrUpperSwitchPointHeating : indicates the selected upper switch point in heating mode at which the blind is driven to the position *uiPositionCoolingMode* for cooling. This is either *stBARSetpointRoom_ComfortCool* or *stBARSetpointRoom_PreComfortCool*, depending on the preselection (see diagrams above).

IrUpperSwitchPointHeating : indicates the lower switch point in heating mode at which the blind is driven to the position *uiPositionHeatingMode* for heating. This is permanently set as the value *stBARSetpointRoom_ComfortHeat* (see diagrams above).

IrUpperSwitchPointCooling : indicates the upper switch point in cooling mode at which the blind is driven to the position *uiPositionCoolingMode* for cooling. This is permanently set as the value *stBARSetpointRoom_ComfortCool* (see diagrams above).

IrLowerSwitchPointCooling : indicates the selected lower switch point in cooling mode at which the blind is driven to the position *uiPositionHeatingMode* for heating. This is either *stBARSetpointRoom_ComfortHeat* or *stBARSetpointRoom_PreComfortHeat*, depending on the preselection (see diagrams above).

bError: this output is switched to TRUE if the parameters entered are erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes \[▶ 237\]](#).



If an error should occur, then this automatic function is deactivated and position and angle are set to 0. This means that if a priority controller is in use, another function with a lower priority (see Overview) automatically takes over control of the blind. In the case of a direct connection, conversely, the blind will drive to position/angle 0.

VAR_IN_OUT

The need for entered parameters to be preserved across a control failure makes it necessary for them to be declared as IN-OUT variables. A reference variable is then assigned to them in the program. Each change in the value of this reference variable is persistently saved in the function block and written back to the reference variable after a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

```

uiBrightnessActivateValue      : UINT; (*lux*)
uiBrightnessDeactivateValue    : UINT; (*lux*)
uiActivateDelay                : UINT; (*sec*)
uiDeactivateDelay              : UINT; (*sec*)
bComfortHeatingAsSwitchPointCooling: BOOL;
bComfortCoolingAsSwitchPointHeating: BOOL;
uiPositionHeatingMode          : UINT;
iAngleHeatingMode              : INT;
uiPositionCoolingMode          : UINT;
iAngleCoolingMode              : INT;
    
```

uiBrightnessActivateValue / uiActivateDelay : activation limit value. The thermal automatic is active only if the outdoor brightness is adequate. Otherwise there would be no radiant heat from the sun which could be utilized or which would need to be blocked. If the outdoor brightness *uiBrightness* [lux] exceeds the value *uiBrightnessActivateValue* [lux] for the time *uiActivateDelay* [s], then the automatic function becomes active if in addition no presence is detected (*bPresence* = FALSE).

uiBrightnessDeactivateValue / uiDeactivateDelay : brightness deactivation limit value: if the outdoor brightness *uiBrightness* [lux] falls below the value *uiBrightnessDeactivateValue* [lux] for the time *uiDeactivateDelay* [s], the automatic system becomes inactive. The value *uiBrightnessDeactivateValue* must be smaller than the value *uiBrightnessActivateValue*. Otherwise an error is output.

bComfortHeatingAsSwitchPointCooling : if this input is TRUE, then the value *stBARSetpointRoom_ComfortHeat* is considered to be the lower value in cooling mode from which the blinds are opened again. If the input is set to FALSE, then the value *stBARSetpointRoom_PreComfortHeat* applies (see introductory description).

bComfortCoolingAsSwitchPointHeating : if this input is TRUE, then the value *stBARSetpointRoom_ComfortCool* is considered to be the lower value in heating mode from which the blinds are closed again. If the input is set to FALSE, then the value *stBARSetpointRoom_PreComfortCool* applies (see introductory description).

uiPositionHeatingMode : height position of the blind in % in the heating case (intended heat irradiation). The following applies to the height position: 0% = fully open, 100% = fully closed.

iAngleHeatingMode : slat angle of the blind in degrees in the heating case (intended heat irradiation).

uiPositionCoolingMode : height position of the blind in % in the cooling case (sun protection). The following applies to the height position: 0% = fully open, 100% = fully closed. The value of the height position must be higher than in the heating case, i.e. the blind must be closed more in the cooling case than in the heating case. Otherwise the supply of heat from the sun cannot be meaningfully controlled according to the above description. In this case an error is output.

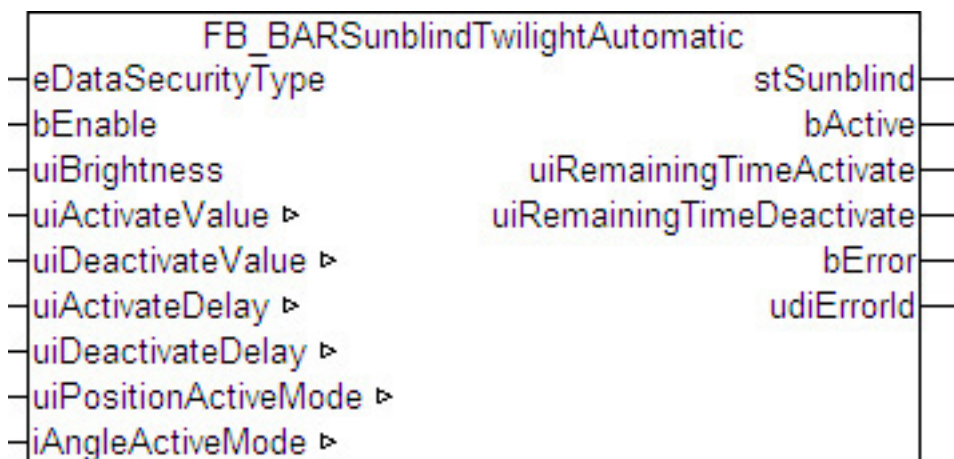
iAngleCoolingMode : slat angle of the blind in degrees in the cooling case (sun protection).

Documents about this

- 📄 example_persistent_e.zip (Resources/zip/11659714827.zip)
- 📄 example_persistent_e.zip (Resources/zip/11659714827.zip)

3.3.4.23 FB_BARSunblindTwilightAutomatic

This function block controls the blind if the outdoor brightness has fallen below a limit value.



The twilight automatic operates with both a value hysteresis and a temporal hysteresis: if the outdoor brightness value *usiBrightness* [lux] falls below the value *usiActivateValue* [lux] for the time *uiActivateDelay* [s], then the function block is active and will supply the blind positions *uiPositionActiveMode* (height in %) and *uiAngleActiveMode* (slat angle in degrees), which are specified on the IN-OUT variables, at the output in the positioning telegram *stSunblind* [▶ 236]. Conversely, if the outdoor brightness exceeds the value *usiDeactivateValue* [lux] for the time *uiDeactivateDelay* [s], then the automatic function is no longer active. The active flag in the positioning telegram *stSunblind* [▶ 236] is reset and the positions for height and angle are set to "0". A function with a lower priority can then take over control.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bEnable           : BOOL;
uiBrightness      : UINT;
```

eDataSecurityType: if `eDataSecurityType := eDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bEnable : the function block has no function if this input is FALSE. In the positioning telegram `stSunblind` [▶ 236], 0 is output for the position and the angle, and `bActive` is FALSE. This means that another function takes over control of the blind via the priority controller.

uiBrightness: outdoor brightness in lux.

VAR_OUTPUT

```
stSunblind      : ST_BARSunblind;
bActive         : BOOL;
uiRemainingTimeActivate : UINT;
uiRemainingTimeDeactivate: UINT;
bError          : BOOL;
udiErrorId     : UDINT;
```

stSunblind: output structure of the blind positions, see `ST_BARSunblind` [▶ 236].

bActive : corresponds to the boolean value `bActive` in the blind telegram `ST_BARSunblind` [▶ 236] and is solely used to indicate whether the function block sends an active telegram.

uiRemainingTimeActivate: indicates the time remaining in seconds after falling below the switch value `usiActivateValue` until automatic mode is activated. This output is 0 as long as no countdown of the time is taking place.

uiRemainingTimeDeactivate: indicates the time remaining in seconds after exceeding the switch value `usiDeactivateValue` until automatic mode is deactivated. This output is 0 as long as no countdown of the time is taking place.

bError: this output is switched to TRUE if the parameters entered are erroneous.


udiErrorId : contains the error code if the values entered should be erroneous. See `error codes` [▶ 237].



If an error should occur, then this automatic function is deactivated and position and angle are set to 0. This means that if a priority controller is in use, another function with a lower priority (see Overview) automatically takes over control of the blind. In the case of a direct connection, conversely, the blind will drive to position/angle 0.

VAR_IN_OUT

The need for entered parameters to be preserved across a control failure makes it necessary for them to be declared as IN-OUT variables. A reference variable is then assigned to them in the program. Each change in the value of this reference variable is persistently saved in the function block and written back to the reference variable after a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>.

```
uiActivateValue      : UINT;
uiDeactivateValue   : UINT;
uiActivateDelay     : UINT;
uiDeactivateDelay   : UINT;
uiPositionActiveMode : UINT;
iAngleActiveMode    : INT;
```

uiActivateValue: activation limit value in lux.

uiDeactivateValue: deactivation limit value in lux.



uiActivateDelay: activation delay in seconds.

uiDeactivateDelay: deactivation delay in seconds.

uiPositionActiveMode: height position of the blind in % if the twilight automatic is active.

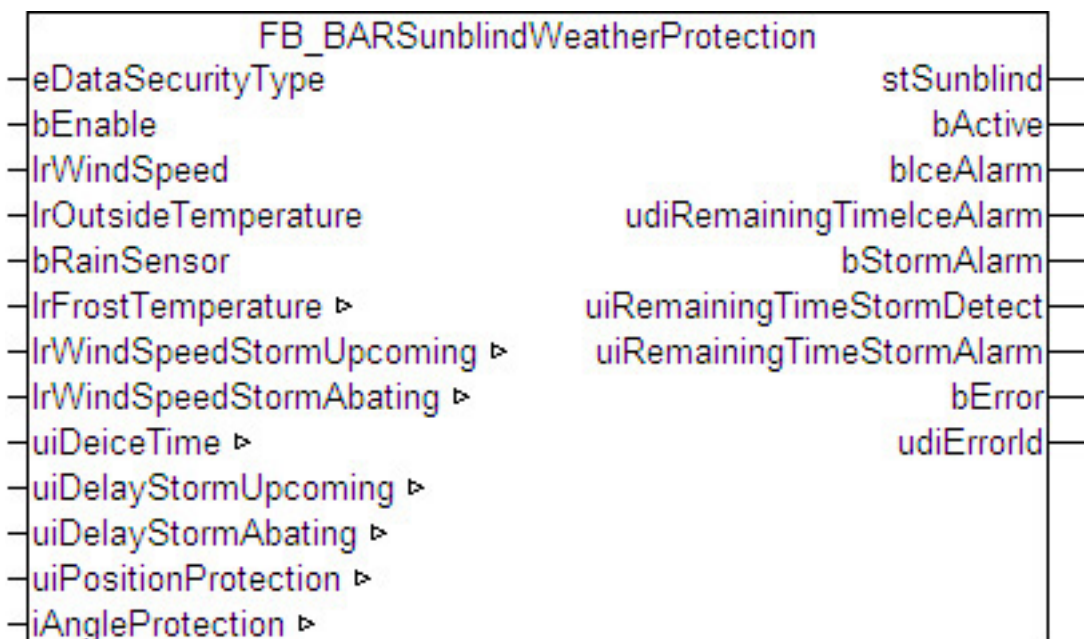
iAngleActiveMode: slat angle of the blind in degrees if the twilight automatic is active.

Documents about this

-  example_persistent_e.zip (Resources/zip/11659714827.zip)
-  example_persistent_e.zip (Resources/zip/11659714827.zip)

3.3.4.24 FB_BARSunblindWeatherProtection

The weather protection has the highest priority in the blind controller (see overview) and is intended to ensure that the blind is not damaged by ice or wind.



The task of the automatic weather protection function is to protect the blind against two impending dangers and, in order to do so, to drive it to a safe position:

- **Icing up:** impending icing up is detected when the measured outside temperature *rOutsideTemperature* falls below the frost limit value *rFrostTemperature* while at the same time rain is detected on *bRainSensor*. This event is saved internally and remains active until it is ensured that the ice has melted again. In addition, the outside temperature must have exceeded the frost limit value for the entered deicing time *uiDeiceTime* (in minutes). For safety reasons the icing event is persistently saved, i.e. also beyond a PLC failure. Thus, if the controller fails during the icing up or deicing period, the blind is considered to be newly iced up when then the controller restarts and the deicing timer starts from the beginning again.
- **Storm:** if the measured wind speed lies above the value *rWindSpeedStormUpcoming* for the time *uiDelayStormUpcoming* [s], then it is assumed that a storm is directly impending. Only if the wind speed falls below the value *rWindSpeedStormAbating* for the time *uiDelayStormAbating* [s] is the storm considered to have abated and the driving of the blind considered to be safe. For safety reasons the storm event is also persistently saved. Thus, if the controller fails during a storm, the storm abating timer is started again from the beginning when the controller is restarted.


In both cases of danger the blind is driven to the protection position specified by *uiPositionProtection* (height position in percent) and *iAngleProtection* (slat angle in degrees).

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
lrWindSpeed       : LREAL;
lrOutsideTemperature: LREAL;
bRainSensor       : BOOL;
```

eDataSecurityType: if *eDataSecurityType:= eDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If *eDataSecurityType:= eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if *eDataSecurityType:= eHVACDataSecurityType_Persistent*. It would lead to early wear of the flash memory.

bEnable : the function block has no function if this input is FALSE. In the positioning telegram *stSunblind* [► 236], 0 is output for the position and the angle, and *bActive* is FALSE. This means that another function takes over control of the blind via the priority controller.

lrWindSpeed: wind speed. The unit of the entry is arbitrary, but it is important that no value is smaller than 0 and that the values become larger with increasing speed.

rOutsideTemperature: outside temperature in degrees Celsius.

bRainSensor: input for a rain sensor.

VAR_OUTPUT

```
stSunblind          : ST_BARSunblind;
bIceAlarm           : BOOL;
udiRemainingTimeIceAlarm : UDINT;
bStormAlarm         : BOOL;
uiRemainingTimeStormDetect: UINT;
uiRemainingTimeStormAlarm : UINT;
bError              : BOOL;
udiErrorId          : UDINT;
```

stSunblind: output structure of the blind positions, see [ST_BARSunblind](#) [► 236].

bActive : corresponds to the boolean value *bActive* in the blind telegram [ST_BARSunblind](#) [► 236] and is solely used to indicate whether the function block sends an active telegram.

bIceAlarm: displays the icing-up alarm.

uiRemainingTimeIceAlarm: in the case of impending icing up (*bIceAlarm* = TRUE), this second counter is set to the deicing time. As soon as the temperature lies above the frost point entered (*rFrostTemperature*), the remaining number of seconds until the 'all-clear' signal is given (*bIceAlarm* = FALSE) is displayed here. This output is 0 as long as no countdown of the time is taking place.

bStormAlarm: displays the storm alarm.

uiRemainingTimeStormDetect: in an uncritical case this second counter constantly displays the alarm delay time *uiDelayStormUpcoming*. If the measured wind speed *rWindSpeed* is above the activation limit value *rWindSpeedStormUpcoming*, the seconds to the alarm are counted down. This output is 0 as long as no countdown of the time is taking place.

uiRemainingTimeStormAlarm: as soon as the storm alarm is triggered, this second counter first constantly displays the deactivation time delay of the storm alarm *uiDelayStormAbating*. If the measured wind speed *rWindSpeed* falls below the deactivation limit value *rWindSpeedStormAbating*, the seconds to the all-clear signal (*bStormAlarm*=FALSE) are counted down. This output is 0 as long as no countdown of the time is taking place.

bError: this output is switched to TRUE if the parameters entered are erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes](#) [► 237].



If an error should occur, then this automatic function is deactivated and position and angle are set to 0. This means that if a priority controller is in use, another function with a lower priority (see Overview) automatically takes over control of the blind. In the case of a direct connection, conversely, the blind will drive to position/angle 0.

VAR_IN_OUT

The need for entered parameters to be preserved across a control failure makes it necessary for them to be declared as IN-OUT variables. A reference variable is then assigned to them in the program. Each change in the value of this reference variable is persistently saved in the function block and written back to the reference variable after a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

```
lrFrostTemperature      : LREAL;
lrWindSpeedStormUpcoming : LREAL;
lrWindSpeedStormAbating : LREAL;
uiDeiceTime            : UINT;
uiDelayStormUpcoming   : UINT;
uiDelayStormAbating    : UINT;
uiPositionProtection   : UINT;
iAngleProtection       : INT;
```

lrFrostTemperature: icing-up temperature limit value in degrees Celsius. This value may not be greater than 0. Otherwise an error is output.

lrWindSpeedStormUpcoming: wind speed limit value for the activation of the storm alarm. This value may not be smaller than 0 and must lie above the value for the deactivation. Otherwise an error is output. The unit of the entry must be the same as that of the input *rWindSpeed*. A value greater than this limit value triggers the alarm after the entered time *uiDelayStormUpcoming*.

lrWindSpeedStormAbating: wind speed limit value for the deactivation of the storm alarm. This value may be not smaller than 0 and must lie below the value for the activation. Otherwise an error is output. The unit of the entry must be the same as that of the input *rWindSpeed*. A value smaller than or equal to this limit value resets the alarm after the entered time *uiDelayStormAbating*.

uiDeiceTime: time until the deicing of the blind after icing up (in minutes). After that the icing up alarm is reset.

uiDelayStormUpcoming: time delay until the storm alarm is triggered in seconds.

uiDelayStormUpcoming: time delay until the storm alarm is reset in seconds.

uiPositionProtection: height position of the blind in % in the case of protection.

iAngleProtection: slat angle of the blind in degrees in the case of protection.

Documents about this

- 📄 example_persistent_e.zip (Resources/zip/11659714827.zip)
- 📄 example_persistent_e.zip (Resources/zip/11659714827.zip)

3.3.4.25 FB_BARSunProtectionEx

Function block for the control of glare protection with the aid of a slatted blind.

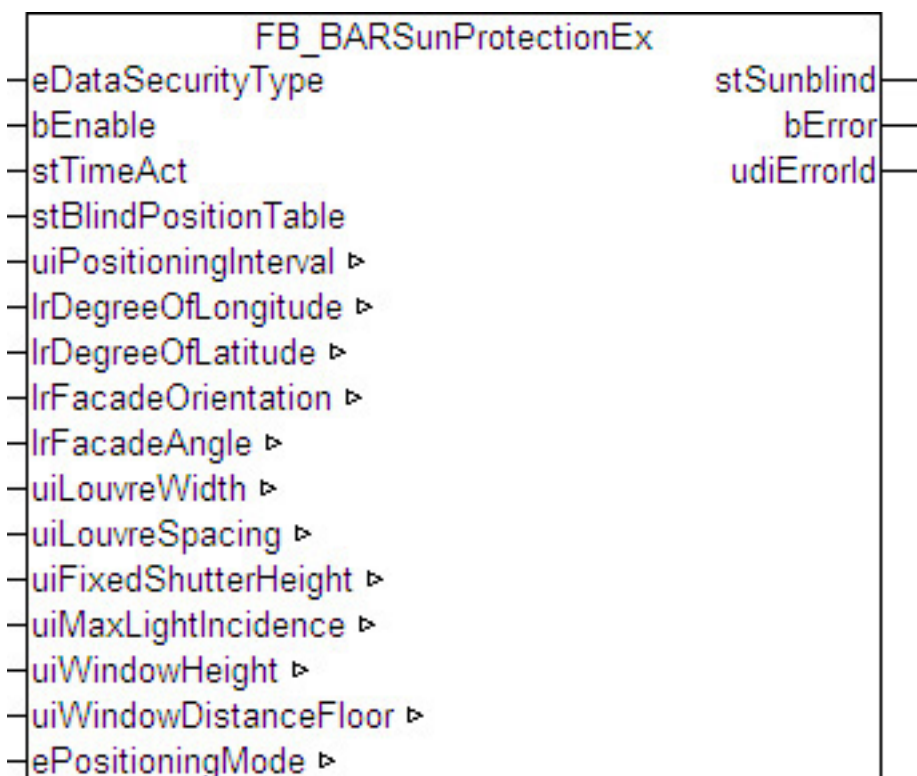


Fig. 10: FB_BARSunProtectionEx

The function block FB_BARSunProtectionEx enables glare protection in two ways that operate in parallel:

- Lamella setpoint tracing, so that the direct incident light cannot quite enter through the lowered part of the blind.
- Control of the blind height, with 3 different possibilities (Adjustable via the enumerator at *ePositioningMode*):
 - 1) fixed blind height, i.e. no change (preset)
 - 2) blind height depending on the sun position, defined via a table ([stBlindPosTable](#) [▶ 234]), see also description of [FB_BARBlindPositionEntry](#) [▶ 160]
 - 3) maximum desired light incidence

On the basis of the parameters entered, which are described further below, the function block calculates the necessary slat angle and blind position and transfers them to the output structure [stSunblind](#) [▶ 236]. Of course, the output does not take place continuously since a constant blind movement would be perceived as extremely distracting. At the input *uiPositioningInterval* it is possible to set in seconds the interval at which new position values are to be output.

However, the shading criteria must always be fulfilled between two positioning times: no direct light may pass through the slats and the desired light incidence through the blind height must remain limited, assuming initially that the blind height is controlled via the "maximum desired light incidence" mode. Therefore, two blind and slat positions are calculated internally: the one for the current switching point and the one for the next switching point. The position in which the blind is more closed is then the valid position.

The positioning in intervals starts precisely when the following three conditions are satisfied:

- the input *bEnable* must be TRUE.
- the function block must not be in an error state due to incorrect parameterization (*bError*=TRUE).
- the sun must have risen, i.e. the sun elevation must be greater than 0°. This is an internal safety query since the limitation to at least 0° should be done by the user by programming the input *bEnable*; see Overview of automatic sun protection (shading correction).

If these three conditions are not satisfied, then the active bit (*bActive*) is set to FALSE, the blind height to 0% and the slat angle to 0% in the positioning structure.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
stTimeAct        : TIMESTRUCT;
stBlindPositionTable: ST_BARBlindPositionTable;
```

eDataSecurityType: if *eDataSecurityType*:= *eDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235.zip>

If *eDataSecurityType*:= *eDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if *eDataSecurityType*:= *eDataSecurityType_Persistent*. It would lead to early wear of the flash memory.

bEnable: if this input is set to FALSE, the positioning is inactive, i.e. the active bit (*bActive*) is reset in the positioning structure *stSunblind* of the type [ST_BARSunblind](#) [▶ 236] and the function block itself remains in a standstill mode. If on the other hand the function block is activated, then the active bit is TRUE and the function block outputs its control values (*uiShutterHeight*, *iLouvreAngle*) in the positioning structure at the appropriate times.

stTimeAct: entry of the current time in GMT (Greenwich Mean Time).

stBlindPositionTable: table of 6 interpolation points, 4 of which are parameterizable, from which a blind position is then given in relation to the position of the sun by linear interpolation. Valid if *ePositioningMode* = *ePosModeFixed* (see enumerator [E_BARPosMode](#) [▶ 233]). For a more detailed description please refer to [FB_BARBlindPositionEntry](#) [▶ 160].

VAR_OUTPUT

```
stSunblind: ST_BARSunblind;
bError     : BOOL;
udiErrorId: UDINT;
```

stSunblind: output structure of the blind positions, see [ST_BARSunblind](#)

bError: this output is switched to TRUE if the parameters entered are erroneous.


udiErrorId : contains the error code if the values entered should be erroneous. See [error codes \[► 237\]](#).



If an error should occur, then this automatic function is deactivated and position and angle are set to 0. This means that if a priority controller is in use, another function with a lower priority (see Overview) automatically takes over control of the blind. In the case of a direct connection, conversely, the blind will drive to position/angle 0.

VAR_IN_OUT

The need for entered parameters to be preserved across a control failure makes it necessary for them to be declared as IN-OUT variables. A reference variable is then assigned to them in the program. Each change in the value of this reference variable is persistently saved in the function block and written back to the reference variable after a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>.

```
uiPositioningInterva : UINT;
lrDegreeOfLongitude  : LREAL;
lrDegreeOfLatitude   : LREAL;
lrFacadeOrientation  : LREAL;
lrFacadeAngle        : LREAL;
uiLouvreWidth        : UINT;
uiLouvreSpacing      : UINT;
uiFixedShutterHeigh : UINT;
uiMaxLightIncidence  : UINT;
uiWindowHeight       : UINT;
uiWindowDistanceFloor : UINT;
ePositioningMode     : E_BARPOSMODE;
```

uiPositioningInterval: positioning interval in seconds - time between two blind position outputs. Valid range: 1s..7200s.

lrDegreeOfLongitude: longitude in degrees. Valid range: -180°..180°.

lrDegreeOfLatitude: latitude in degrees. Valid range: -90°..90°.

lrFacadeOrientation: facade orientation in degrees:

In the northern hemisphere, the following applies for the facade orientation (looking out of the window):

Line of sight	Facade orientation
North	$\beta=0^\circ$
East	$\beta=90^\circ$
South	$\beta=180^\circ$
West	$\beta=270^\circ$

The following applies for the southern hemisphere:

Line of sight	Facade orientation
South	$\beta=0^\circ$
East	$\beta=90^\circ$
North	$\beta=180^\circ$
West	$\beta=270^\circ$

lrFacadeAngle: facade inclination in degrees. See [Facade inclination \[► 150\]](#).

uiLouvreWidth: width of the slats in mm, see [sketch \[► 147\]](#).

uiLouvreSpacing: slat spacing in mm, see [sketch \[► 147\]](#).

uiFixedShutterHeight: fixed (constant) shutter height [0..100%]. Valid if *ePositioningMode* = *ePosModeFixed* (see enumerator [E_BARPosMode \[► 233\]](#)).

uiMaxLightIncidence: maximum desired incidence of light in mm measured from the outside of the wall (see Height adjustment). With the aid of the parameters *uiWindowHeight* and *uiWindowDistanceFloor*, a calculation is performed in relation to the position of the sun to determine how high the blind must be so that the incidence of light does not exceed the value *uiMaxLightIncidence*. Valid if *ePositioningMode* = *ePosModeMaxIncidence* (see enumerator [E_BARPosMode](#) [► 233]).

uiWindowHeight: window height in mm for the calculation of the blind height if the mode "maximum desired incidence of light" is selected.

uiWindowDistanceFloor: distance between the floor and the window sill in mm for the calculation of the blind height if the mode "maximum desired incidence of light" is selected.

ePositioningMode: selection of the positioning mode, see enumerator [E_BARPosMode](#) [► 233].

Also see about this

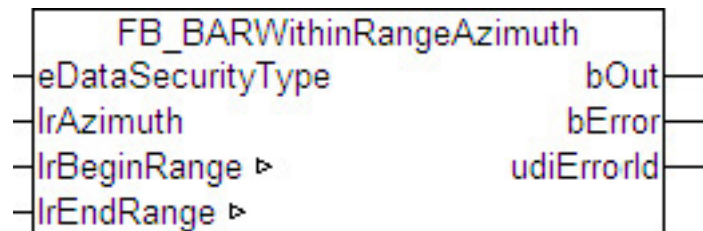
- ☰ Sun protection: basic principles and definitions [► 147]

Documents about this

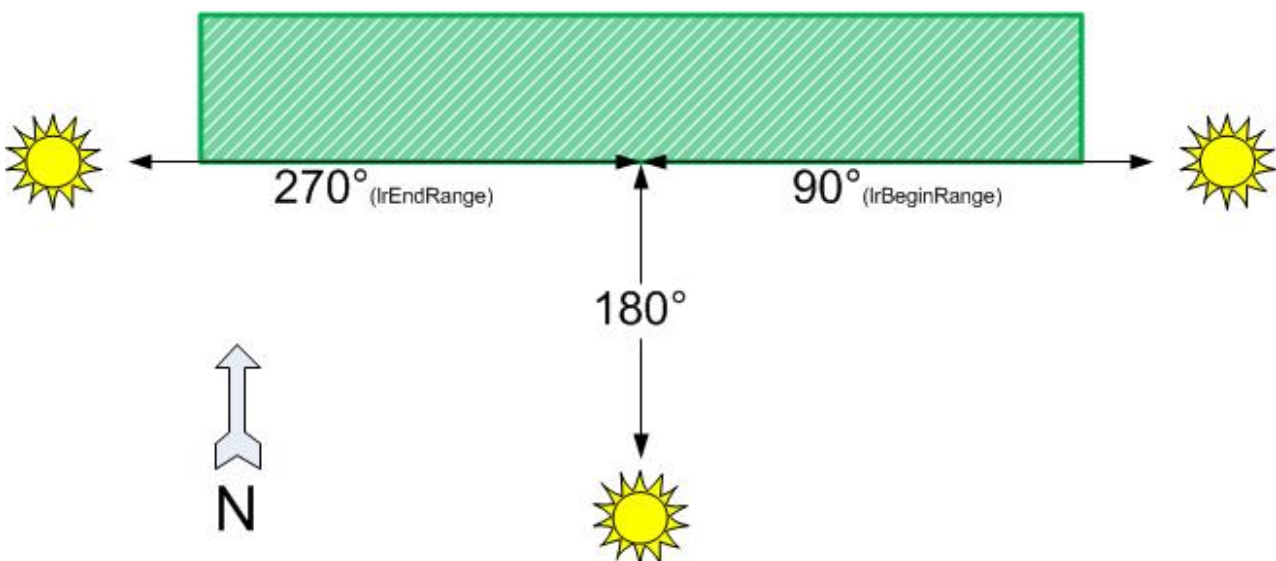
- ☰ example_persistent_e.zip (Resources/zip/11659714827.zip)
- ☰ example_persistent_e.zip (Resources/zip/11659714827.zip)

3.3.4.26 FB_BARWithinRangeAzimuth

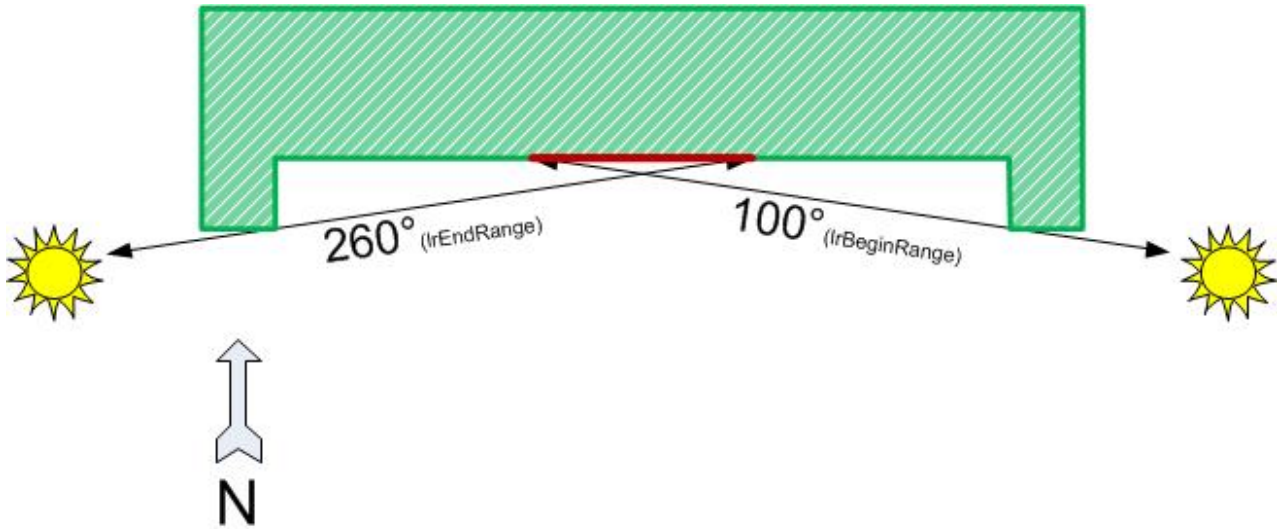
This function block checks whether the current azimuth angle (horizontal position of the sun) lies within the limits entered. As can be seen in the [overview](#) [► 146], the function block provides an additionally evaluation as to whether the sun protection of a window group should be activated. Therefore the observations in the remainder of the text always apply to one window group.



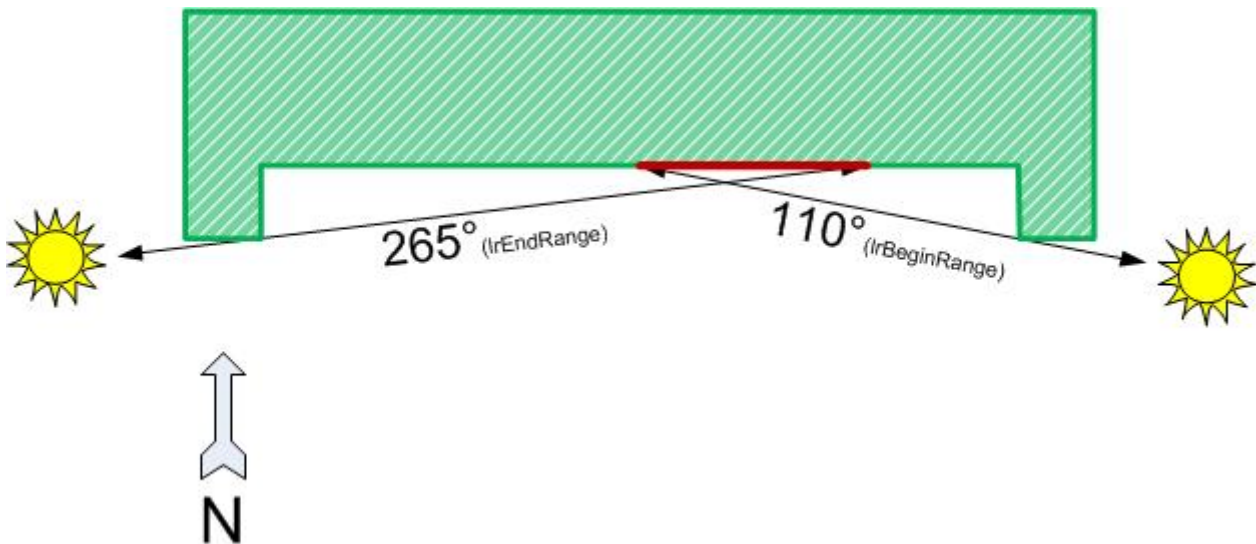
The sun incidence azimuth angle on a smooth facade will always be *facade orientation-90°... facade orientation+90°*.



If the facade has lateral projections, however, this range is limited. This limitation can be checked with the help of this function block. However, the position of the window group on the facade also plays a role. If it lies centrally, this gives rise to the following situation (the values are only examples):



The values change for a group at the edge:



The beginning of the range *rBeginRange* may thereby be larger than the end *rEndRange*; it is then regarded beyond 0°:

Sample:

lrAzimuth	10.0°
lrBeginRange	280.0°
lrEndRange	20.0°
bOut	TRUE


However, the range regarded may not be greater than 180° or equal to 0° – this would be unrealistic. Such entries result in an error on the output *bError* – the test output *bOut* is then additionally set to FALSE.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
lrAzimuth          : LREAL;
```

eDataSecurityType: if `eDataSecurityType:= eDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

IrAzimuth: current azimuth angle.

VAR_OUTPUT

```
bOut      : BOOL;
bError    : BOOL;
udiErrorId : UDINT;
```


bOut: binary delayed output of the threshold switch.

bError: this output is switched to TRUE if the parameters entered are erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes \[► 237\]](#).

VAR_IN_OUT

In order for the registered parameters to be retained beyond a controller failure, it is necessary to declare them as In-Out variables. A reference variable is then assigned to them in the program. Each change of the value of these reference variables is persistently stored in the function block and written back to the reference variable following a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

```
lrBeginRange : LREAL;
lrEndRange   : LREAL;
```

lrBeginRange: begin of range in degrees.

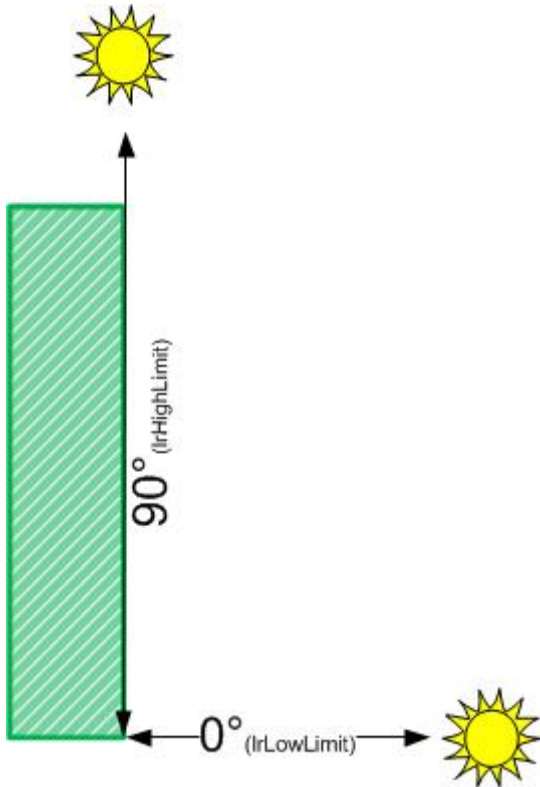
lrEndRange: end of range in degrees.

3.3.4.27 FB_BARWithinRangeElevation

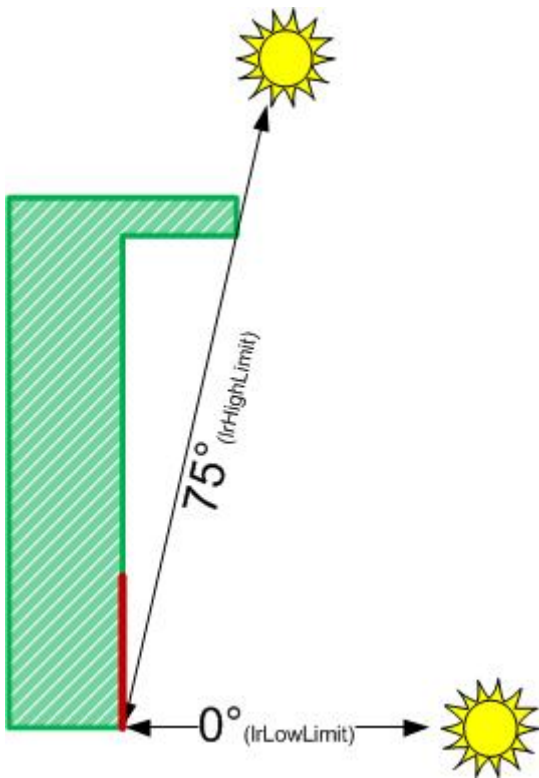
This function block checks whether the current angle of elevation (vertical position of the sun) lies within the limits entered. As can be seen in the [overview \[► 146\]](#), the function block provides an additionally evaluation as to whether the sun protection of a window group should be activated. Therefore the observations in the remainder of the text always apply to one window group.



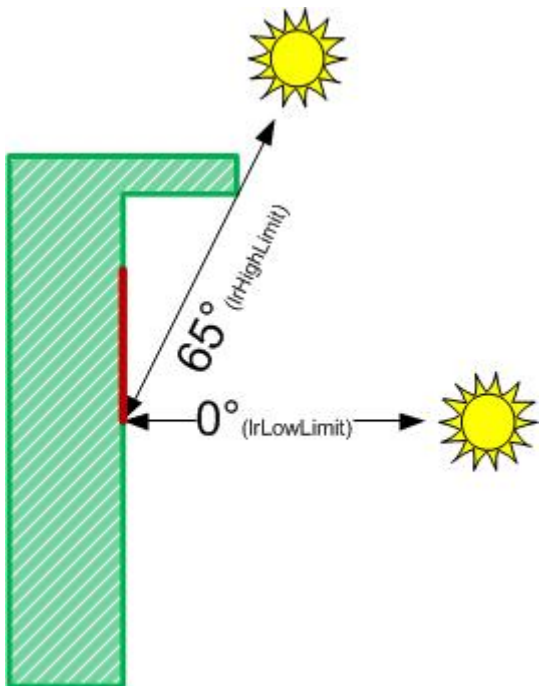
A normal vertical facade is irradiated by the sun at an angle of elevation of 0° to maximally 90°.



If the facade has projections, however, this range is limited. This limitation can be checked with the help of this function block. However, the position of the window group on the facade also plays a role. If it lies in the lower range, this gives rise to the following situation (the values are only examples):



The values change for a group below the projection:




The lower observation limit, *rLowLimit*, may thereby not be greater than or equal to the upper limit, *rHighLimit*. Such entries result in an error on the output *bError* – the test output *bOut* is then additionally set to FALSE.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
lrElevation       : LREAL;
```

eDataSecurityType: if `eDataSecurityType:= eDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

IrElevation: current elevation angle in degrees.

VAR_OUTPUT

```
bOut      : BOOL;
bError    : BOOL;
udiErrorId : UDINT;
```


bOut: binary delayed output of the threshold switch.

bError: this output is switched to TRUE if the parameters entered are erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes](#) [▶ 237].

VAR_IN_OUT

In order for the registered parameters to be retained beyond a controller failure, it is necessary to declare them as In-Out variables. A reference variable is then assigned to them in the program. Each change of the value of these reference variables is persistently stored in the function block and written back to the reference variable following a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

```
lrLowLimit  : LREAL;
lrHighLimit : LREAL;
```

lrLowLimit: lower limit in degrees.

lrHighLimit: upper limit in degrees.

3.3.5 Program example

3.3.5.1 Program example FB_BARSunblindSwitch

This example is intended to show how a possible activation (input `bEnable`) of the manual blind controller might look. The input and output variables (bordered) have the following meaning:

tAutoResetTime : time after which the activation is to be automatically reset.

bSwitchUp : blind up button.

bSwitchDown : blind down button.

bPauseClock : pause signal of the property, e.g. lunch break: 12:00-13:00. *bPauseClock* = TRUE: pause.

bPresenceDetection : occupied message of the room in which the blinds are controlled.
bPresenceDetection= TRUE: room is occupied.

uiSwitchOverTime : time in milliseconds until the corresponding manual command latches in the case of a continuous active command input.

stSunblind: positioning telegram, see [ST_BARSunblind \[► 236\]](#). For further linking to a priority selection, see [FB_BARSunblindPrioritySwitch \[► 198\]](#).

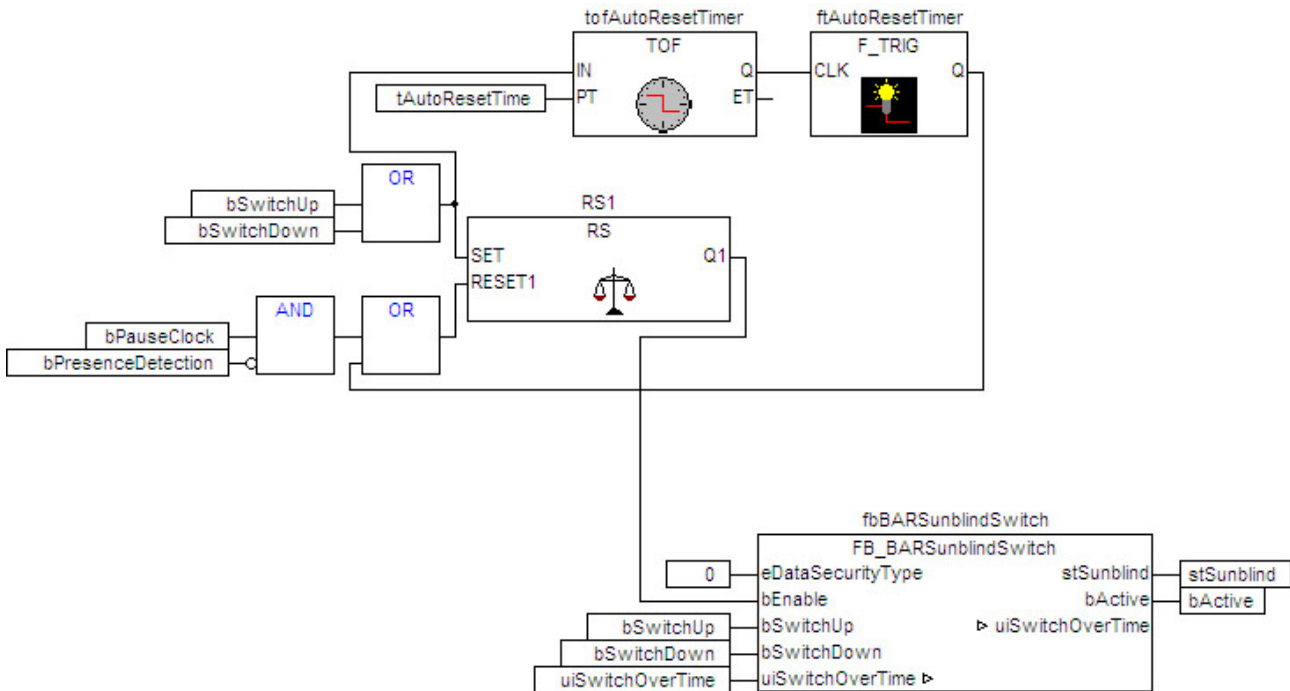


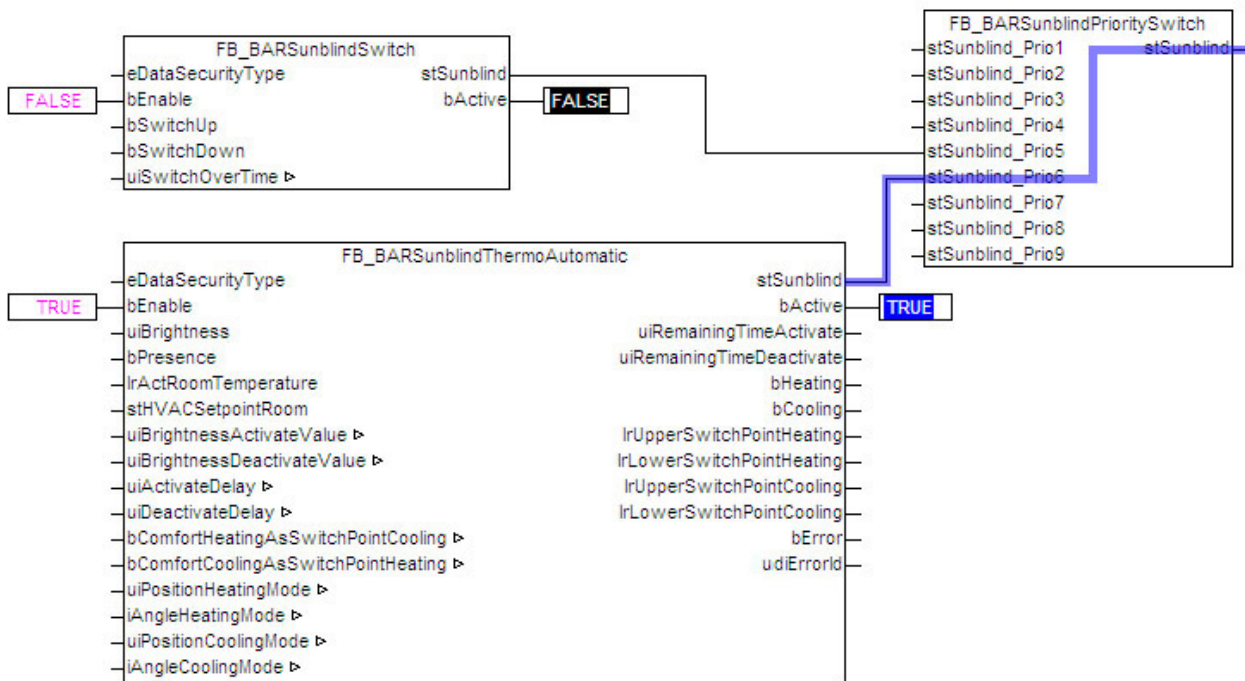
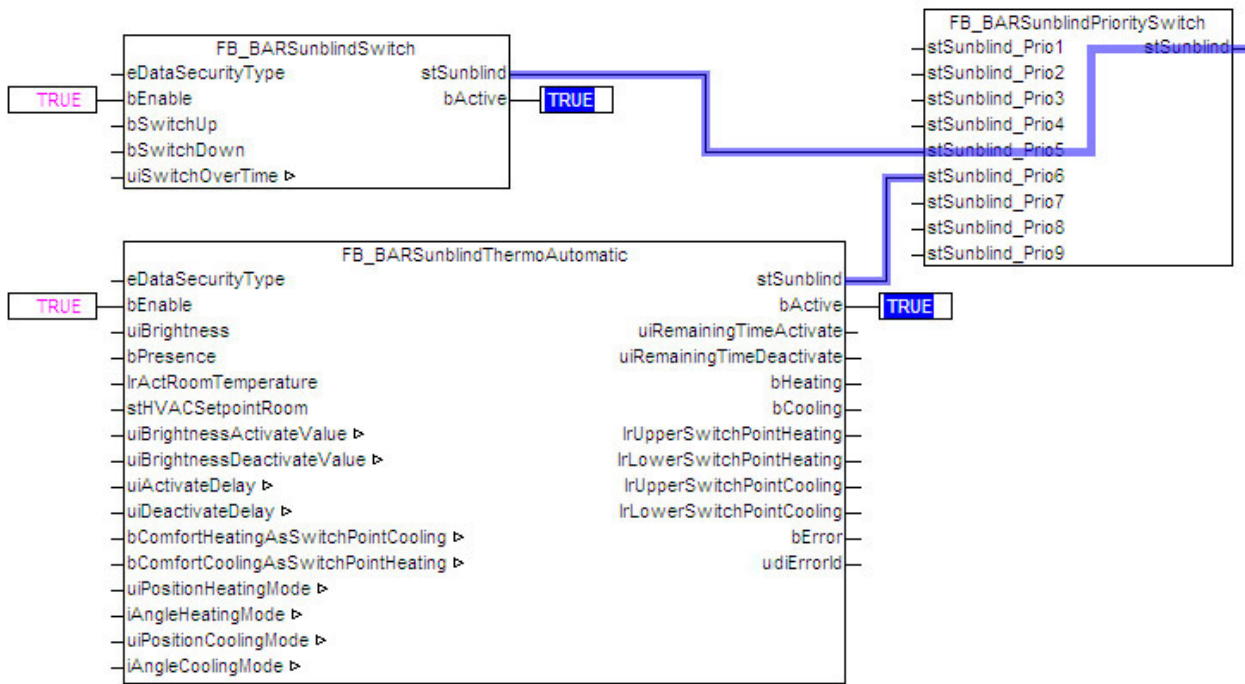
Fig. 11: SwitchWithResetEnable

Functioning

If one of the blind buttons *bSwitchUp* or *bSwitchDown* is actuated, the output of the flip-flop will initially be set to TRUE, but only if there is currently no pause signal applied (*bPauseClock*) or if the room is currently occupied (*bPresenceDetection*). The *bEnable* input of the function block `fbBARSunblindSwitch` is thus TRUE, which activates it. The respective push button signal which has led to the activation can be passed on by the function block directly in the command telegram *stSunblind*.

At the same time, a TRUE signal is present at the input of the switch-off delay `tofAutoResetTimer` and its output *Q* is immediately set to TRUE. On releasing the blind button the switch-off delay is started, and after the expiry of *tAutoResetTime* the output *Q* is reset to FALSE. This falling edge is in turn recognized by `ftAutoResetTimer` and a trigger pulse is sent that resets the flip-flop. The activation input *bEnable* is thus FALSE once again and the function block `fbBARSunblindSwitch` is passive.

If this function block is linked via the command telegram to a priority selection [FB_BARSunblindPrioritySwitch \[► 198\]](#), then the telegram will be passed through there to the next priority:



3.3.6 Structures and enumeration

3.3.6.1 E_BARCtrlFct

```

TYPE E_BARCtrlFct:
(
    eBARCtrlFct_Off           := 0,
    eBARCtrlFct_Heating      := 1,

```

```
eBARCtrlFct_Cooling      := 2
);
END_TYPE
```

eBARCtrlFct_Off: function selection OFF active.

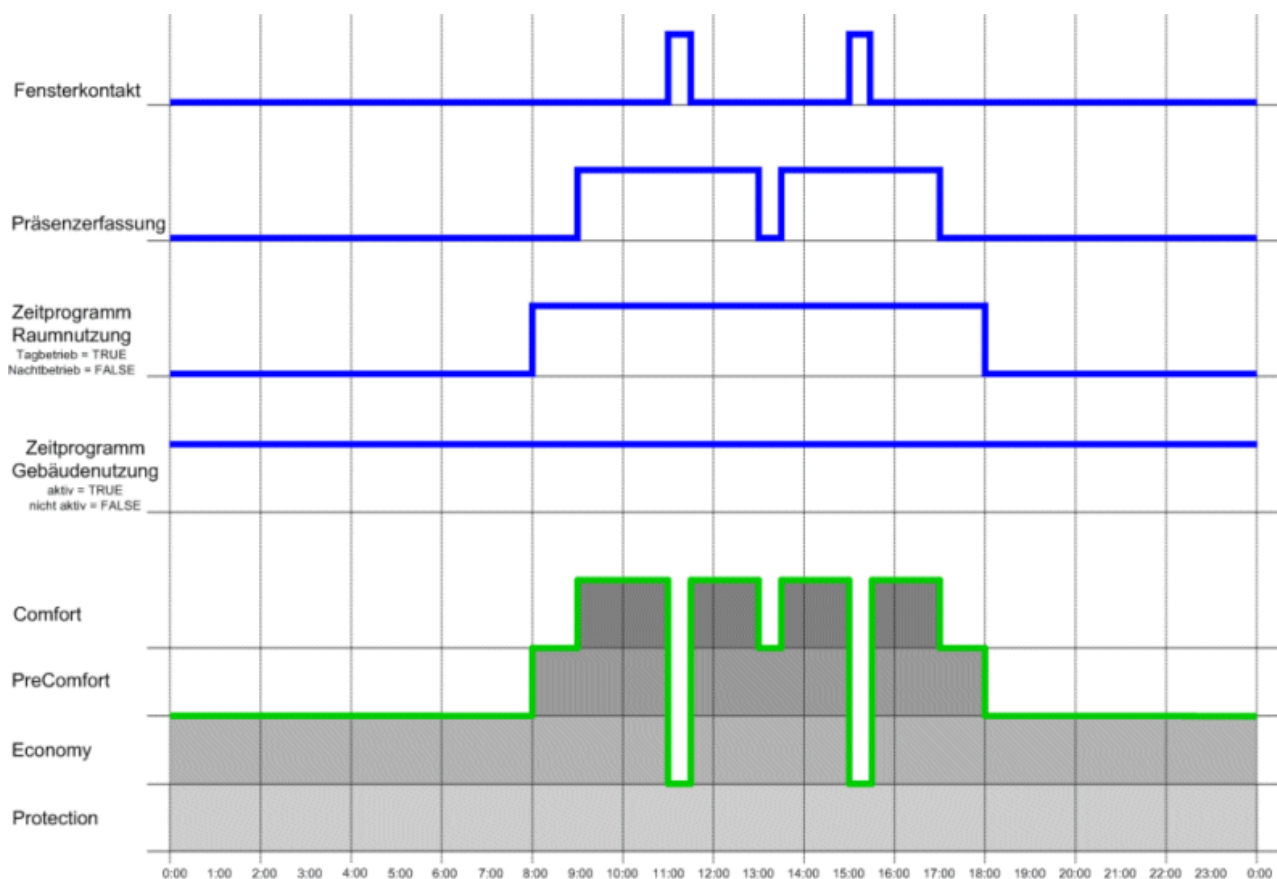
eBARCtrlFct_Heating: function selection for heating operation active.

eBARCtrlFct_Cooling: function selection for cooling operation active.

3.3.6.2 E_BAREnergyLevel

```
TYPE E_BAREnergyLevel:
(
  eBAREnergyLevel_AUTO_I      := 0,
  eBAREnergyLevel_Protection  := 1,
  eBAREnergyLevel_Economy     := 2,
  eBAREnergyLevel_PreComfort  := 3,
  eBAREnergyLevel_Comfort     := 4,
  eBAREnergyLevel_AUTO_II     := 5
);
END_TYPE
```

eBAREnergyLevel_AUTO_I: automatic mode I means that in this operation mode it is only possible to switch to the next higher energy level if the lower energy levels are already TRUE. In this operation mode the states of the inputs *bProtection*, *bEconomy*, *bPreComfort* and *bComfort* are evaluated for the selection of the energy level. Please see diagram.



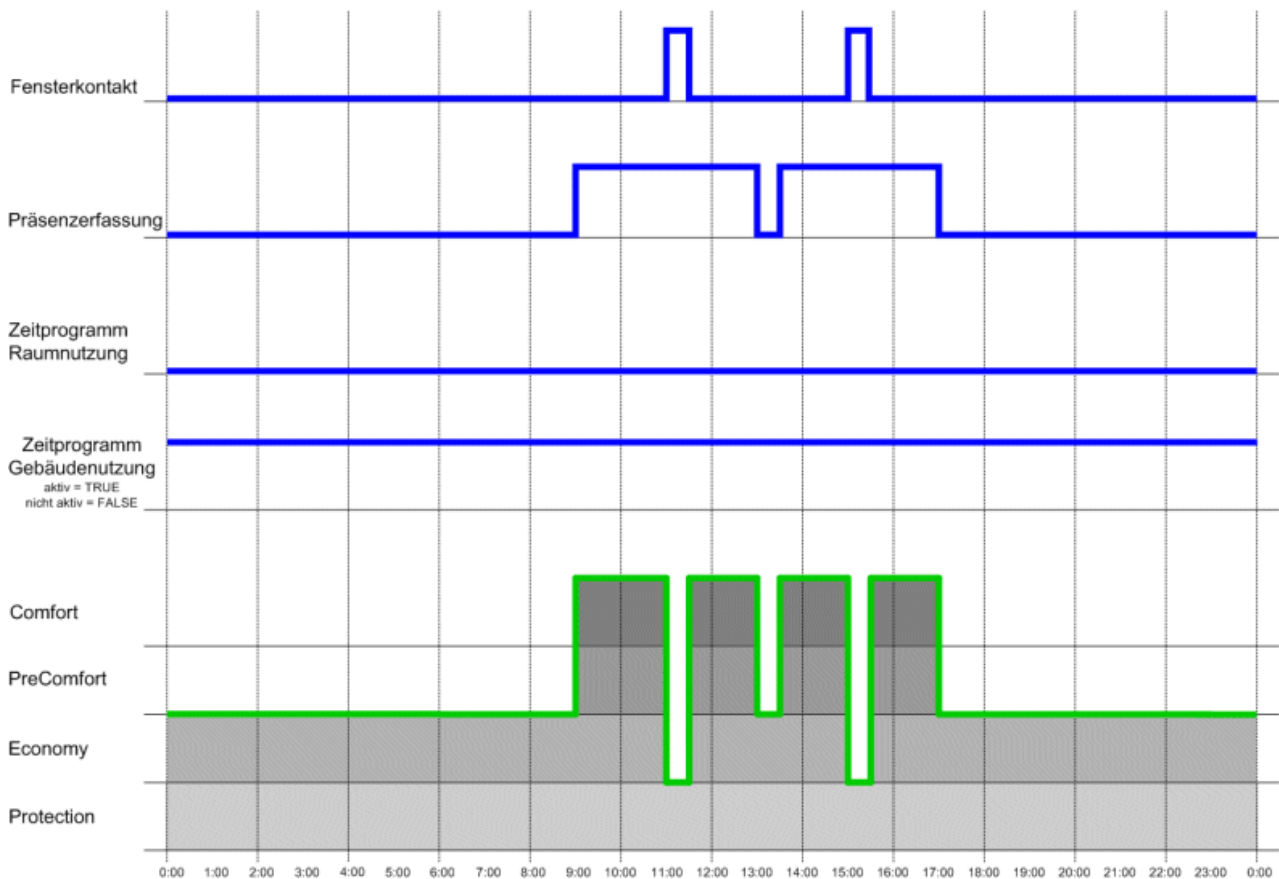
eBAREnergyLevel_Protection: in this operation mode, which can be manually set by the building management level, the room/zone/area is maintained in a minimum energy state. This operation mode is a pure building protection mode.

eBAREnergyLevel_Economy: in this operation mode, which can be set manually from the building management level, the room/zone/area is kept in economy mode. This operation mode must be set for longer periods of absence or at night.

eBAREnergyLevel_PreComfort: in this operation mode, which can be set manually from the building management level, the room/zone/area is kept in standby mode. This operation mode is to be set for short absence times.

eBAREnergyLevel_Comfort: in this operation mode, which can be set manually from the building management level, the room/zone/area is kept in comfort mode. This operation mode is to be set in the case of presence.

eBAREnergyLevel_AUTO_II: Automatic mode II means that in this operation mode the energy level switches directly from the Protection, Economy or PreComfort energy level to the Comfort energy level on recognition of presence. Please see diagram.



3.3.6.3 E_BARPosMode

Enumerator for the definition of the positioning mode.

```

TYPE E_BARPosMode :
(
    ePosModeFixed := 0,
    ePosModeTable,
    ePosModeMaxIncidence
);
END_TYPE
    
```

ePosModeFixed: the blind height adopts a fixed value, which is set (in %) via the *uiFixedShutterHeight* value on the function block [FB_BARSunProtectionEx](#) [▶ 221];

ePosModeTable: the height positioning takes place with the help of a table of 6 interpolation points, 4 of which are parameterizable. A blind position in relation to the position of the sun is then calculated from these points by linear interpolation. For a more detailed description please refer to [FB_BARBlindPositionEntry](#) [▶ 160].

ePosModeMaxIncidence: the positioning takes place with specification of the maximum desired incidence of light.

3.3.6.4 E_BARShadingObjectType

Enumerator for selection of shading object type.

```
TYPE E_BARShadingObjectType :
(
  eObjectTypeTetragon := 0,
  eObjectTypeGlobe    := 1
);
END_TYPE
```

eObjectTypeTetragon: Object type is a tetragon.

eObjectTypeGlobe: Object type is a globe.

3.3.6.5 E_HVACDataSecurityType

Enumeration for defining the saving of entered parameters.

```
TYPE E_HVACDataSecurityType :
(
  eDataSecurityType_Persistent := 0,
  eDataSecurityType_Idle      := 1
);
END_TYPE
```

eDataSecurityType_Persistent: the parameters entered are persistently saved after each change. The values are available on the input or output variables following a restart.

eDataSecurityType_Idle: the values are not saved. Following a restart only default values – if existent – are available.

3.3.6.6 ST_BARBlindPositionTable

Structure of the interpolation point entries for the height adjustment of the blind.

```
TYPE ST_BARBlindPositionTable:
STRUCT
  lrSunElevation : ARRAY[0..5] OF LREAL;
  uiBlindPosition : ARRAY[0..5] OF UINT;
  bValid          : BOOL;
END_STRUCT
END_TYPE
```

lrSunElevation / uiBlindPosition : the 6 interpolation points that are transferred, wherein the array elements 0 and 5 represent the automatically generated edge elements mentioned above.

bValid : validity flag for the function block [FB_BARSunProtectionEx \[▶ 221\]](#). It is set to TRUE by the function block [FB_BARBlindPositionEntry \[▶ 160\]](#) if the data entered correspond to the validity criteria described.

3.3.6.7 ST_BARCorner

Information about window corners

```
TYPE ST_BARCorner :
STRUCT
  lrX : LREAL;
  lrY : LREAL;
  bShaded : BOOL;
END_STRUCT
END_TYPE
```

lrX: X-coordinate of the window (on the facade).

lrY: Y-coordinate of the window (on the facade).

bShaded: information whether this corner point is shaded: *bShaded*=TRUE: corner point is shaded.

3.3.6.8 ST_BARFacadeElement

List entry for a facade element (window).

```

TYPE ST_BARFacadeElement :
STRUCT
  usiGroup      : USINT;
  lrWindowWidth : LREAL;
  lrWindowHeight : LREAL;
  stCorner      : ARRAY [1..4] OF ST_BARCorner;
  bValid        : BOOL;
END_STRUCT
END_TYPE
    
```

usiGroup: group membership of an element.

lrWindowWidth: width of the window.

lrWindowHeight: height of the window.

stCorner: coordinates of the window corners and information as to whether this corner point is in the shade; see [ST_BARCorner](#) [► 234].

bValid: plausibility of the entered data: *bValid*=TRUE: data are plausible.

3.3.6.9 ST_BARSetpointRoom

```

TYPE ST_BARSetpointRoom :
STRUCT
  stBARSetpointRoom_ComfortHeat      : REAL:= 21.0;
  stBARSetpointRoom_PreComfortHeat   : REAL:= 19.0;
  stBARSetpointRoom_EconomyHeat      : REAL:= 15.0;
  stBARSetpointRoom_ProtectionHeat    : REAL:= 12.0;
  stBARSetpointRoom_ComfortCool      : REAL:= 24.0;
  stBARSetpointRoom_PreComfortCool   : REAL:= 28.0;
  stBARSetpointRoom_EconomyCool      : REAL:= 35.0;
  stBARSetpointRoom_ProtectionCool    : REAL:= 40.0;
END_STRUCT
END_TYPE
    
```

The values in the structure are defined with the preset values.

3.3.6.10 ST_BARShadingObject

List entry for a shading object.

```

TYPE ST_BARShadingObject :
STRUCT
  lrP1x      : LREAL;
  lrP1y      : LREAL;
  lrP1z      : LREAL;
  lrP2x      : LREAL;
  lrP2y      : LREAL;
  lrP2z      : LREAL;
  lrP3x      : LREAL;
  lrP3y      : LREAL;
  lrP3z      : LREAL;
  lrP4x      : LREAL;
  lrP4y      : LREAL;
  lrP4z      : LREAL;
  lrMx       : LREAL;
  lrMy       : LREAL;
  lrMz       : LREAL;
  lrRadius   : LREAL;
  usiBeginMonth : USINT;
  usiEndMonth  : USINT;
  eType      : E_BARShadingObjectType;
  bValid     : BOOL;
END_STRUCT
END_TYPE
    
```

lrP1x .. lrP4z : corner coordinates. Of importance only if the element is a rectangle.

lrMx .. lrMz: center coordinates. Of importance only if the element is a sphere.

IrRadius: radius of the sphere. Of importance only if the element is a sphere.

usiBeginMonth: beginning of the shading period (month).

usiEndMonth: end of the shading period (month).

eType: object type, see [E_BARShadingObjectType](#) [[▶ 234](#)].

bValid: plausibility of the data: bValid=TRUE: data are plausible.

Remark about the shading period:

The month entries must not be 0 and not be greater than 12, all other combinations are possible.

Examples:

Start=1, End=1: shading in January.

Start=1, End=5: shading from the beginning of January to the end of May.

Start=11, End=5: shading from the beginning of November to the end of May (of the following year).

3.3.6.11 ST_BARSunblind

Structure of the blind positioning telegram.

```

TYPE ST_BARSunblind:
STRUCT
    uiPosition      : UINT;
    iAngle          : INT;
    bManUp          : BOOL;
    bManDown        : BOOL;
    bManualMode     : BOOL;
    bActive         : BOOL;
END_STRUCT
END_TYPE

```

uiPosition: transferred blind height in %.

iAngle: transferred slat position in degrees.

bManUp: manual command: blind up.

bManDown: manual command: blind down.

bManualMode: TRUE: manual operation mode is active.

FALSE: automatic mode is active.

bActive: the sender of the telegram is active. This bit is evaluated only by the priority controller [FB_BARSunblindPrioritySwitch](#) [[▶ 198](#)]. The sun protection actuators [FB_BARSunblindActuator](#) [[▶ 185](#)] and [FB_BARRollerblind](#) [[▶ 200](#)] ignore it.

3.3.6.12 ST_BARSunblindScene

Table entry for a blind scene.

```

TYPE ST_BARSunblindScene:
STRUCT
    uiPosition      : UINT;
    iAngle          : INT;
END_STRUCT
END_TYPE

```

uiPosition: blind height in %.

iAngle: louvre position in degrees.

3.3.7 List descriptions

3.3.7.1 List of facade elements

The data of all windows (facade elements) per facade are stored within the program in a field of structural elements of the type [ST_BARFacadeElement](#) [► 235].

The declaration is global, because the management function block [FB_BARFacadeElementEntry](#) [► 164] as well as the shading correction [FB_BARShadingCorrection](#) [► 176] / [FB_BARShadingCorrectionSouth](#) [► 179] access this field directly via input/output variable:

```
VAR_GLOBAL
    arrFacadeElement : ARRAY[1..iColumnsPerFacade, 1..iRowsPerFacade] OF ST_BARFacadeElement;
END_VAR
```

The variables *iColumnsPerFacade* and *iRowsPerFacade* thereby define the upper limit of the available elements and are to be globally declared as constants:

```
VAR_GLOBAL CONSTANT
    iRowsPerFacade : INT :=10;
    iColumnsPerFacade : INT :=20;
END_VAR
```

3.3.7.2 List of shading elements

The shading elements per facade are stored within the program in a field of structural elements of the type [ST_BARShadingObject](#) [► 235].

The declaration is global, because the management function block [FB_BARShadingObjectsEntry](#) [► 182] as well as the shading correction [FB_BARShadingCorrection](#) [► 176] / [FB_BARShadingCorrectionSouth](#) [► 179] access this field directly via input/output variable:

```
VAR_GLOBAL
    arrShadingObject : ARRAY[1..iShadingObjects] OF ST_BARShadingObject;
END_VAR
```

The variable *iShadingObjects* thereby represents the upper limit of the available elements and is to be globally defined as a constant:

```
VAR_GLOBAL CONSTANT
    iShadingObjects : INT := 20;
END_VAR
```

3.3.7.3 Error Codes

udiErrorId (hex)	udiErrorId (dec)	Error
0x0	0	No error
0x8001	32769	FB_BARBlindPositionEntry [► 160]: values for <i>rSunElevation</i> are not ascending or unequal
0x8002	32770	FB_BARBlindPositionEntry [► 160]: values for <i>rSunElevation</i> are not within the valid range of 0°..90°
0x8003	32771	FB_BARBlindPositionEntry [► 160]: values for <i>uiBlindPosition</i> are not within the valid range of 0%..100%
0x8004	32772	FB_BARSunProtectionEx [► 221]: the duration of the positioning interval is equal to zero, or it exceeds 7200 s.

udiErrorId (hex)	udiErrorId (dec)	Error
0x8005	32773	FB_BARSunProtectionEx [► 221]: the longitude entered is not within the valid range of -180°..180°.
0x8006	32774	FB_BARSunProtectionEx [► 221]: the latitude entered is not within the valid range of -90°..90°.
0x8007	32775	FB_BARSunProtectionEx [► 221]: the value for the slat spacing (<i>uiLouvreSpacing</i>) is greater than or equal to the value for the slat width (<i>uiLouvreWidth</i>).
0x8008	32776	FB_BARSunProtectionEx [► 221]: the "Values valid" bit (<i>bValid</i>) in the <i>stBlindPositionTable</i> positioning table is not set - invalid values, see FB_BARBlindPositionEntry [► 160].
0x8009	32777	FB_BARSunProtectionEx [► 221]: the value entered for the fixed shutter height (<i>uiFixedShutterHeight</i>) exceeds 100%
0x801A	32778	FB_BARSunProtectionEx [► 221]: the value entered for the window height is equal to zero.
0x801B	32779	FB_BARSunProtectionEx [► 221]: the value entered for the slat spacing is zero.
0x801C	32780	FB_BARSunProtectionEx [► 221]: the value entered for the slat width is zero.
0x801D	32781	FB_BARShadingObjectsEntry [► 182]: the sum of the angles of the rectangle is not 360°. This means that the corners are not in the order P1, P2, P3 and P4 but rather P1, P3, P2 and P4. This results in a crossed-over rectangle.
0x801E	32782	FB_BARShadingObjectsEntry [► 182]: the corners of the rectangle are not on the same level.
0x801F	32783	FB_BARShadingObjectsEntry [► 182]: the z-component of P1 is less than zero. This corner would thus lie behind the facade.
0x8010	32784	FB_BARShadingObjectsEntry [► 182]: the z-component of P3 is less than zero. This corner would thus lie behind the facade.
0x8011	32785	FB_BARShadingObjectsEntry [► 182]: P1 is equal to P2. The object entered is thus not a rectangle.

udiErrorId (hex)	udiErrorId (dec)	Error
0x8012	32786	FB_BARShadingObjectsEntry [▶ 182]: P1 is equal to P3. The object entered is thus not a rectangle.
0x8013	32787	FB_BARShadingObjectsEntry [▶ 182]: P1 is equal to P4. The object entered is thus not a rectangle.
0x8014	32788	FB_BARShadingObjectsEntry [▶ 182]: P2 is equal to P3. The object entered is thus not a rectangle.
0x8015	32789	FB_BARShadingObjectsEntry [▶ 182]: P2 is equal to P4. The object entered is thus not a rectangle.
0x8016	32790	FB_BARShadingObjectsEntry [▶ 182]: P3 is equal to P4. The object entered is thus not a rectangle.
0x8017	32791	FB_BARShadingObjectsEntry [▶ 182]: the entered radius is equal to zero
0x8018	32792	FB_BARShadingObjectsEntry [▶ 182]: the z-component of the sphere center is less than zero. This point would thus lie behind the facade.
0x8019	32793	FB_BARFacadeElementEntry [▶ 164]: the group index is 0, but at the same time another entry of the facade element is not zero. Only if all entries of a facade element are zero is it considered to be a valid, deliberately omitted facade component, otherwise it is interpreted as an incorrect entry.
0x801A	32794	FB_BARFacadeElementEntry [▶ 164]: the x-component of the first corner (Corner1) is less than zero.
0x801B	32795	FB_BARFacadeElementEntry [▶ 164]: the y-component of the first corner (Corner1) is less than zero.
0x801C	32796	FB_BARFacadeElementEntry [▶ 164]: the window width is less than or equal to zero.
0x801D	32797	FB_BARFacadeElementEntry [▶ 164]: the window height is less than or equal to zero.
0x801E	32798	FB_BARDelayedHysteresis [▶ 163]: the switch-on value <i>bOnValue</i> is smaller than the switch-off value <i>bOffValue</i> .

udiErrorId (hex)	udiErrorId (dec)	Error
0x801F	32799	FB_BARWithinRangeAzimuth [▶ 224]: the beginning of the range <i>rBeginRange</i> and the end of the range <i>rEndRange</i> are the same.
0x8020	32800	FB_BARWithinRangeAzimuth [▶ 224]: one of the range limits entered is greater than 360° or less than 0°
0x8021	32801	FB_BARWithinRangeAzimuth [▶ 224]: the entered range is greater than 180°.
0x8022	32802	FB_BARWithinRangeElevation [▶ 226]: the entered lower limit <i>rLowLimit</i> is greater than or equal to the upper limit <i>rHighLimit</i> .
0x8023	32803	FB_BARWithinRangeElevation [▶ 226]: the upper limit value <i>rHighLimit</i> is greater than 90° or the lower limit value <i>rLowLimit</i> is less than 0°.
0x8024	32804	FB_BARShadingObjectsEntry [▶ 182]: the number of the month when shading begins or when shading ends is zero or greater than 12.
0x8025	32805	FB_BARShadingObjectsEntry [▶ 182]: index error! <i>ild</i> lies outside the permissible limits <i>1..iShadingObjects</i> . See list of shading elements [▶ 237].
0x8026	32806	FB_BARSunblindActuator [▶ 185]/ FB_BARSunblindActuatorEx [▶ 190]: the entered blind height is 0.
0x8027	32807	FB_BARSunblindActuator [▶ 185]/ FB_BARSunblindActuatorEx [▶ 190]: the entire "Up" stroke time (<i>uiTotalTimeUp/udiTotalTimeUp</i>) or the total "Down" stroke time (<i>uiTotalTimeDown/udiTotalTimeDown</i>) is 0 ms.
0x8028	30808	FB_BARSunblindActuator [▶ 185]/ FB_BARSunblindActuatorEx [▶ 190]: the slat "Up" turning time (<i>uiTurningTimeUp</i>) or the slat "Down" turning time (<i>uiTurningTimeDown</i>) is 0 ms.
0x8029	32809	FB_BARSunblindActuator [▶ 185]/ FB_BARSunblindActuatorEx [▶ 190]: the upper slat angle limit value (<i>iAngleLimitUp</i>) is smaller than or equal to the lower slat angle limit value (<i>iAngleLimitDown</i>)

udiErrorId (hex)	udiErrorId (dec)	Error
0x802A	32810	FB_BARSunblindScene [▶ 203]: the selected scene number exceeds the maximum number of 20.
0x802B	32811	FB_BARSunblindThermoAutomatic [▶ 209]: the preset room temperature values are in the wrong order.
0x802C	32812	FB_BARSunblindThermoAutomatic [▶ 209]: the brightness activation limit value is smaller than or equal to the deactivation limit value.
0x802D	32813	FB_BARSunblindThermoAutomatic [▶ 209]: the blind position (height in %) of the heating mode is greater than or equal to that of the cooling mode.
0x802E	32814	FB_BARSunblindTwilightAutomatic [▶ 209]: the brightness limit value for the activation of the twilight automatic is greater than or equal to the deactivation value.
0x802F	32815	FB_BARSunblindWeatherProtection [▶ 218]: the frost limit is greater than 10 °C.
0x8030	32816	FB_BARSunblindWeatherProtection [▶ 218]: the limit value for the activation of the storm alarm is smaller than or equal to the deactivation value.
0x8031	32817	FB_BARSunblindWeatherProtection [▶ 218]: the input of wind force limit values of less than 0 is not permissible.
0x8032	32818	FB_BARShadingCorrection [▶ 176] / FB_BARShadingCorrectionSouth [▶ 179]: the group index is 0. This index is intended only for facade elements that are to be marked as non-existent, for example if windows are omitted from an even facade grid or if a door is installed instead of a window.
0x8033	32819	FB_BARShadingCorrection [▶ 176] / FB_BARShadingCorrectionSouth [▶ 179]: a window element of the selected group is marked as invalid (bValid=FALSE).
0x8034	32820	FB_BARFacadeElementEntry [▶ 164]: index error! iColumn and/or iRow are outside the permissible limits 1..iColumnsPerFacade or 1..iRowsPerFacade. See list of facade elements [▶ 237].

udiErrorId (hex)	udiErrorId (dec)	Error
0x8035	32821	FB_BARSunProtectionEx [▶ 221]: the facade inclination entered (<i>IrFacadeAngle</i>) is not within the valid range of -90°..90°.
0x8036	32822	FB_BARSunblindActuator [▶ 185]/ FB_BARSunblindActuatorEx [▶ 190]: the time to traverse the backlash (<i>uiBacklashTimeUp</i>) or (<i>uiBacklashTimeDown</i>) is 0 ms.
0x8037	32823	FB_BARRollerBlind [▶ 200]: the total "up" stroke time (<i>udiTotalTimeUp</i>) or the total "down" stroke time (<i>udiTotalTimeDown</i>) is 0 ms.
0x8038	32824	FB_BARSMSISunblindActuator [▶ 196]: the value for <i>wLouvreRange</i> must not be 0.
0x8039	32825	FB_BARSMSISunblindActuator [▶ 196]: the ratio <i>wLouvreRange / (iAngleLimitUp + iAngleLimitDown)</i> must be greater than 0.
0x803A.. 0x80FF	32826.. 33023	reserved for shading function blocks
0x8100	33024	FB_BARLightCircuit [▶ 133]: the operation mode, <i>uiLightCtrlMode</i> , is greater than 2. However, only 0, 1, 2 are permitted.
0x8101	33025	FB_BARLightCircuitDim [▶ 134]: The operation mode, <i>uiLightCtrlMode</i> , is greater than 2. However, only 0, 1, 2 are permitted.
0x8102	33026	FB_BARLightCircuitDim [▶ 134]: the lower limit value, <i>IrMinDimValue</i> , is less than zero or greater than 100%.
0x8103	33027	FB_BARLightCircuitDim [▶ 134]: the upper limit value, <i>IrMaxDimValue</i> , is less than zero or greater than 100%.
0x8104	33028	FB_BARLightCircuitDim [▶ 134]: the lower limit value, <i>IrMinDimValue</i> , is greater than the upper limit value <i>IrMaxDimValue</i> .
0x8105	33029	FB_BARLightCircuitDim [▶ 134]: the value for "Manual On", <i>IrManualDimValue</i> , is less than zero or greater than 100%.
0x8106	33030	FB_BARAAutomaticLight [▶ 117]: the operation mode, <i>uiLightCtrlMode</i> , is greater than 2. However, only 0, 1, 2 are permitted.

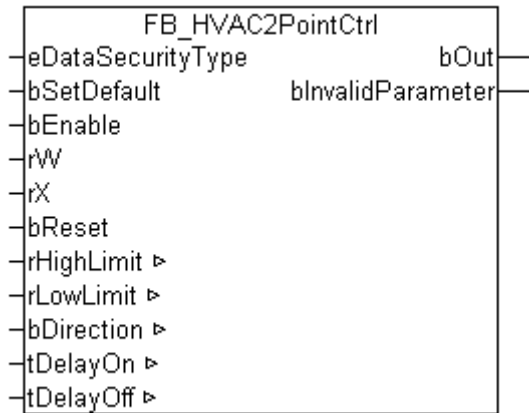
udiErrorId (hex)	udiErrorId (dec)	Error
0x8107	33031	FB_BARAutomaticLight [▶ 117]: the value for "Manual On", <i>IrManualDimValue</i> , is less than zero or greater than 100%.
0x8108	33032	FB_BARAutomaticLight [▶ 117]: the light switch-on value, <i>IrOnDimValue</i> , is less than zero or greater than 100%.
0x8109	33033	FB_BARStairwellAutomatic [▶ 138]: the operation mode, <i>uiLightCtrlMode</i> , is greater than 2. However, only 0,1, 2 are permitted.
0x810A	33034	FB_BARTwilightAutomatic [▶ 140]: the operation mode, <i>uiLightCtrlMode</i> , is greater than 2. However, only 0,1, 2 are permitted.
0x810B	33035	FB_BARTwilightAutomatic [▶ 140]: the value for "Manual On", <i>IrManualDimValue</i> , is less than zero or greater than 100%.
0x810C	33036	FB_BARTwilightAutomatic [▶ 140]: the ON value for the automatic function, <i>IrDimOnValue</i> , is less than zero or greater than 100%.
0x810D	33037	FB_BARTwilightAutomatic [▶ 140]: the OFF value for the automatic function, <i>IrDimOffValue</i> , is less than zero or greater than 100%.
0x810E	33038	FB_BARTwilightAutomatic [▶ 140]: the switch-on threshold value, <i>uiSwitchOnValue</i> , is greater than or equal to the switch-off threshold value, <i>uiSwitchOffValue</i> .
0x810F	33039	FB_BARDaylightControl [▶ 126]: the operation mode, <i>uiLightCtrlMode</i> , is greater than 2. However, only 0,1, 2 are permitted.
0x8110	33040	FB_BARConstantLightControl [▶ 119]: the operation mode, <i>uiLightCtrlMode</i> , is greater than 2. However, only 0,1, 2 are permitted.
0x8111	33041	FB_BARConstantLightControl [▶ 119]: the lower limit value, <i>IrMinDimValue</i> , is less than zero or greater than 100%.
0x8112	33042	FB_BARConstantLightControl [▶ 119]: the upper limit value, <i>IrMaxDimValue</i> , is less than zero or greater than 100%.

udiErrorId (hex)	udiErrorId (dec)	Error
0x8113	33043	FB_BARConstantLightControl [▶ 119]: the lower limit value, <i>lrMinDimValue</i> , is greater than the upper limit value <i>lrMaxDimValue</i> .
0x8114	33044	FB_BARConstantLightControl [▶ 119]: the starting value of the controller, <i>lrStartDimValue</i> , is less than zero or greater than 100%.
0x8115	33045	FB_BARConstantLightControl [▶ 119]: the value for "Manual On", <i>lrManualDimValue</i> , is less than zero or greater than 100%.
0x8116.. 0x81FF	33046.. 33279	reserved for light function blocks
0x8200	33280	FB_BARPICtrl [▶ 112]: the execution cycle number <i>uiCtrlCycleCall</i> is zero
0x8201	33281	FB_BARPICtrl [▶ 112]: the proportional band <i>rXp</i> is zero.
0x8202	33282	FB_BARPICtrl [▶ 112]: the lower control value limit value <i>rYMin</i> is greater than or equal to the upper limit value <i>rYMax</i> .
0x8203	33283	FB_BARPICtrl [▶ 112]: the task cycle time <i>tTaskCycleTime</i> is set to 0.
0x8204	33284	FB_HVACOptimizedOn [▶ 431]: the outside temperature values <i>rOutsideTemp[i]</i> of the pre-start function <i>stTempChangeFunction</i> are not in ascending order or 2 values are the same.
0x8205	33285	FB_HVACOptimizedOn [▶ 431]: one or more values of the room temperature change <i>rRoomTempChange[i]</i> in the pre-start function <i>stTempChangeFunction</i> are higher than the upper limit value <i>rMaxTempChange</i> .
0x8206	33286	FB_HVACOptimizedOn [▶ 431]: one or more values of the room temperature change <i>rRoomTempChange[i]</i> in the pre-start function <i>stTempChangeFunction</i> are lower than the lower limit value <i>rMinTempChange</i> .
0x8207	33287	FB_HVACOptimizedOn [▶ 431]: the control input <i>eCtrlFct</i> is set neither to heating nor to cooling mode (<i>eBARCtrlFct_Heating</i> or <i>eBARCtrlFct_Cooling</i>).

udiErrorId (hex)	udiErrorId (dec)	Error
0x8208	33288	FB_HVACOptimizedOff [▶ 441]: the outside temperature values <i>rOutsideTemp[i]</i> of the pre-start function <i>stTempChangeFunction</i> are not in ascending order or 2 values are the same.
0x8209	33289	FB_HVACOptimizedOff [▶ 441]: one or more values of the room temperature change <i>rRoomTempChange[i]</i> in the pre-start function <i>stTempChangeFunction</i> are higher than the upper limit value <i>rMaxTempChange</i> .
0x820A	33290	FB_HVACOptimizedOff [▶ 441]: one or more values of the room temperature change <i>rRoomTempChange[i]</i> in the pre-start function <i>stTempChangeFunction</i> are lower than the lower limit value <i>rMinTempChange</i> .
0x820B	33291	FB_HVACOptimizedOff [▶ 441]: the control input <i>eCtrlFct</i> is set neither to heating nor to cooling mode (<i>eBARCtrlFct_Heating</i> or <i>eBARCtrlFct_Cooling</i>).
0x820C	33292	FB_HVACTempChangeFunctionEntry [▶ 452]: the outside temperature values <i>rOutsideTemp[i]</i> of the pre-start function <i>stTempChangeFunction</i> are not in ascending order or 2 values are the same.
0x8C00.. 0x8CFF	35840.. 36095	See SMI error codes 0x0C00 to get the SMI specific error code.

3.4 HVAC Controller

3.4.1 FB_HVAC2PointCtrl



Application


This function block represents a 2-point controller. The controller is enabled via *bEnable* = TRUE and is then active.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
rW                : REAL;
rX                : REAL;
bReset            : BOOL;
```

eDataSecurityType: if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If *eDataSecurityType* := *eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the controller is enabled via a TRUE and is then active.

rW: The setpoint is transferred with the variable *rW*.

rX: the actual value of the control loop is transferred with the *rX* variable.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
bOut          : BOOL;
bInvalidParameter : BOOL;
```

bOut: switching output of the on-off controller.

bInvalidParameter: TRUE if an error occurs during the plausibility check. The message must be acknowledged with *bReset*.

VAR_IN_OUT

```
rHighLimit    : REAL;
rLowLimit     : REAL;
bDirection    : BOOL;
tDelayOn      : TIME;
tDelayOff     : TIME;
```

rHighLimit: upper limit of the control deviation (0..32767). The variable is saved persistently. Preset to 0.

rLowLimit: lower limit of the control deviation (0..32767). The variable is saved persistently. Preset to 0.

bDirection: the control direction of the controller is determined by *bDirection*. FALSE = heating mode; TRUE = cooling mode. The variable is saved persistently. Preset to 0.

tDelayOn: switch-on delay [s]. The variable is saved persistently. Preset to 0 s.

tDelayOff: switch-off delay [s]. The variable is saved persistently. Preset to 0 s.

Timing characteristics

Figure 1:

Curve of switching output *bOut* for a on-off controller in relation to the upper/lower limits of the control deviation, with no switch-on or switch-off delay.

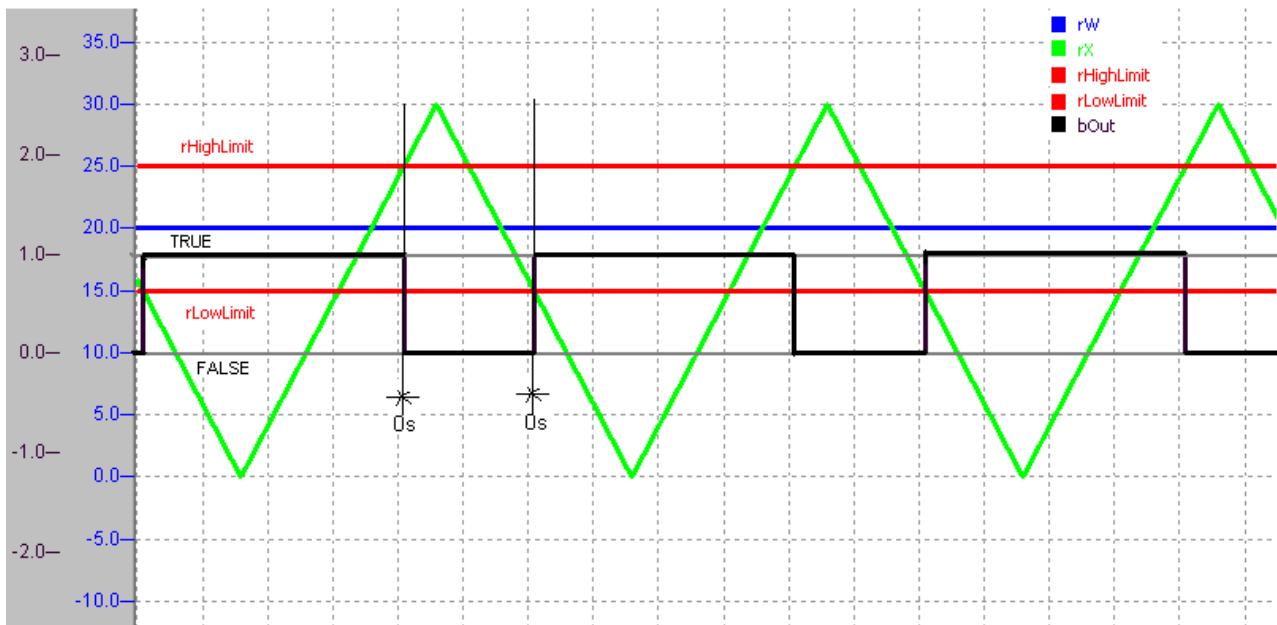


Figure: FB_HVAC2PointCtrl timing diagram

```
rHighLimit = 5
rLowLimit = 5
bDirection = FALSE
tDelayOn = 0s
tDelayOff = 0s
```

Figure 2:

Curve of switching output *bOut* for a on-off controller in relation to the upper limit *rHighLimit* = 3 and the lower limit *rLowLimit* = 8 with a switch-on delay of *tDelayOn* = 4 s and a switch-off delay of *tDelayOff* = 6 s.

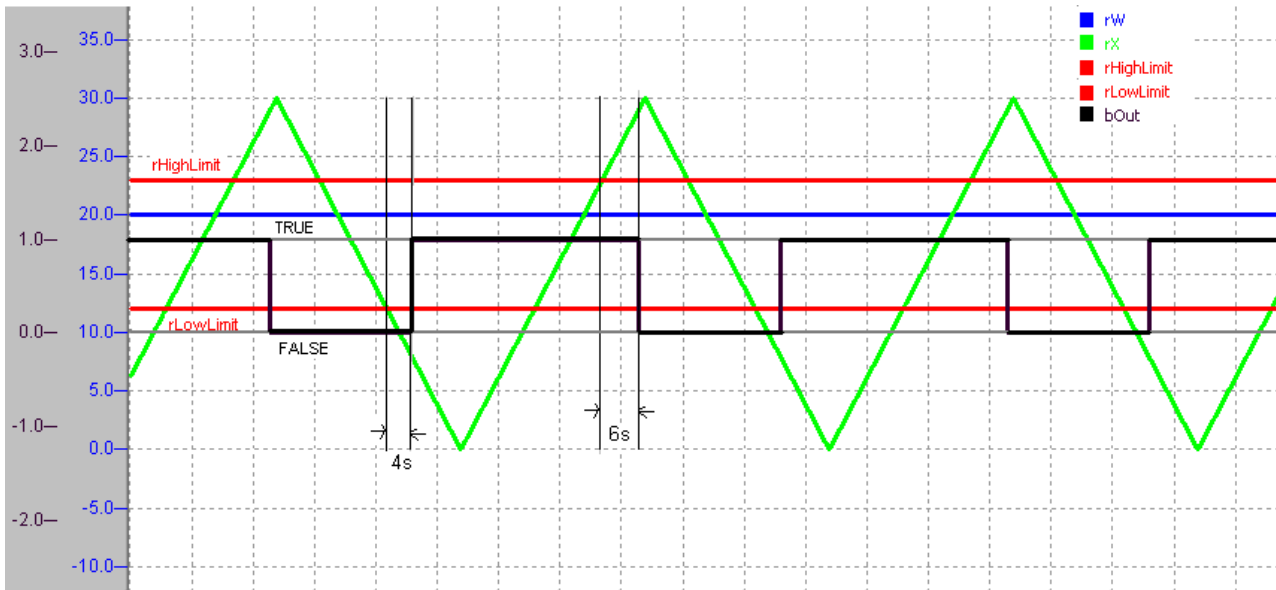


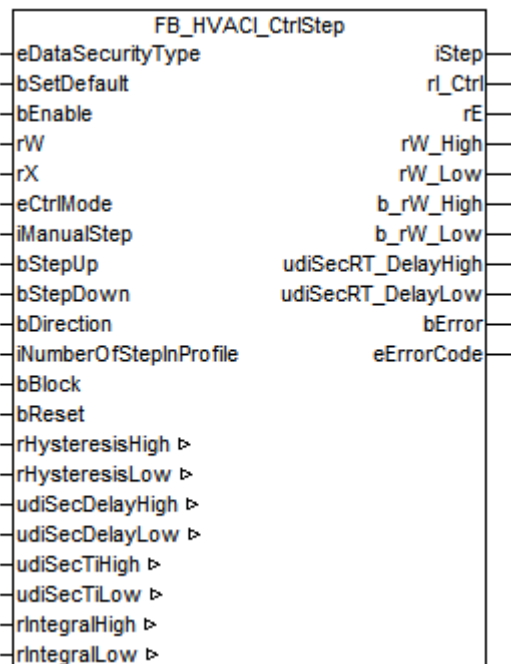
Figure: FB_HVAC2PointCtrl timing diagram

rHighLimit = 3
 rLowLimit = 8
 bDirection = FALSE
 tDelayOn = 4s
 tDelayOff = 6s

Documents about this

- example_persistent_e.zip (Resources/zip/11659714827.zip)

3.4.2 FB_HVACI_CtrlStep



Application

Application

The function block serves the sequential control of power generators.

In conjunction with the power range table [FB HVACPowerRangeTable \[► 275\]](#), the power step controller can be used for the stepped control of several heating boilers, refrigerating machines or recooling plants.

Switching up or down to the next higher or next lower power step respectively takes place via an integral ($rW - rX$) and a time delay. First of all, the actual temperature value rX must have exceeded or fallen below a threshold value rW_High / rW_Low . Subsequently, the integral starts. If the upper or lower limit value ($rIntegralHigh / rIntegralLow$) is reached at the output rI_Ctrl , then a timing element is started upon whose expiry ($udiSecRT_DelayHigh / udiSecRT_DelayLow$) the output $iStep$ is incremented or decremented; see [Program flowchart \[► 252\]](#)

i An integration time setting of $udiSecTiHigh/udiSecTiLow = 60$ seconds means that the I transfer element changes by one Kelvin per minute. With a control deviation $rE = 5$, the output of the I transfer element rI_Ctrl has the value 5 after one minute. Example: $bDirection = FALSE$; $udiSecTiLow = 60$; $rE = 2$; $rIntegralLow = 10$ With the remaining control deviation of 2 and the lower integral limit of 10, after 5 minutes $rI_Ctrl = rIntegralLow$. The consequence of this is that a timing element with the delay time $udiSecDelayLow$ is started. After its expiry ($udiSecRT_DelayHigh / udiSecRT_DelayLow$), the output $iStep$ is incremented by 1 and the I transfer element and the time delays are reset.

i $iStep$ is set to 1 for one PLC cycle if $bEnable = TRUE$, $bError = FALSE$, $eCtrlMode = eHVACCtrlMode_Auto$ AND $iNumberOfStepInProfile > 0$.

Conditions

The I transfer element is released, if

```
bEnable = TRUE AND NOT bError AND eCtrlMode = eHVACCtrlMode_Auto AND NOT bBlock
(
(rX >= rW_High AND ((iStep > 0 AND NOT bDirection) OR (iStep < iNumberOfStepInProfile AND bDirection))
OR
(rX < rW_Low AND ((iStep < iNumberOfStepInProfile AND NOT bDirection) OR (iStep > 0 AND bDirection))
)
```

. If $rI_Ctrl = rIntegralHigh / rIntegralLow$, then the timing elements of the delay times $udiSecDelayHigh / udiSecDelayLow$ are activated. After the expiry of the delay times $udiSecRT_DelayHigh / udiSecRT_DelayLow$, the steps of the power step sequence are controlled as follows:

$iStep = iStep + 1$ if

```
(rX < rW_Low AND iStep < iNumberOfStepInProfile AND NOT bDirection)
OR
(rX >= rW_High AND iStep < iNumberOfStepInProfile AND bDirection)
```

$iStep = iStep - 1$ if

```
(rX < rW_Low AND iStep > 0 AND bDirection)
OR
(rX >= rW_High AND iStep > 0 AND NOT bDirection)
```

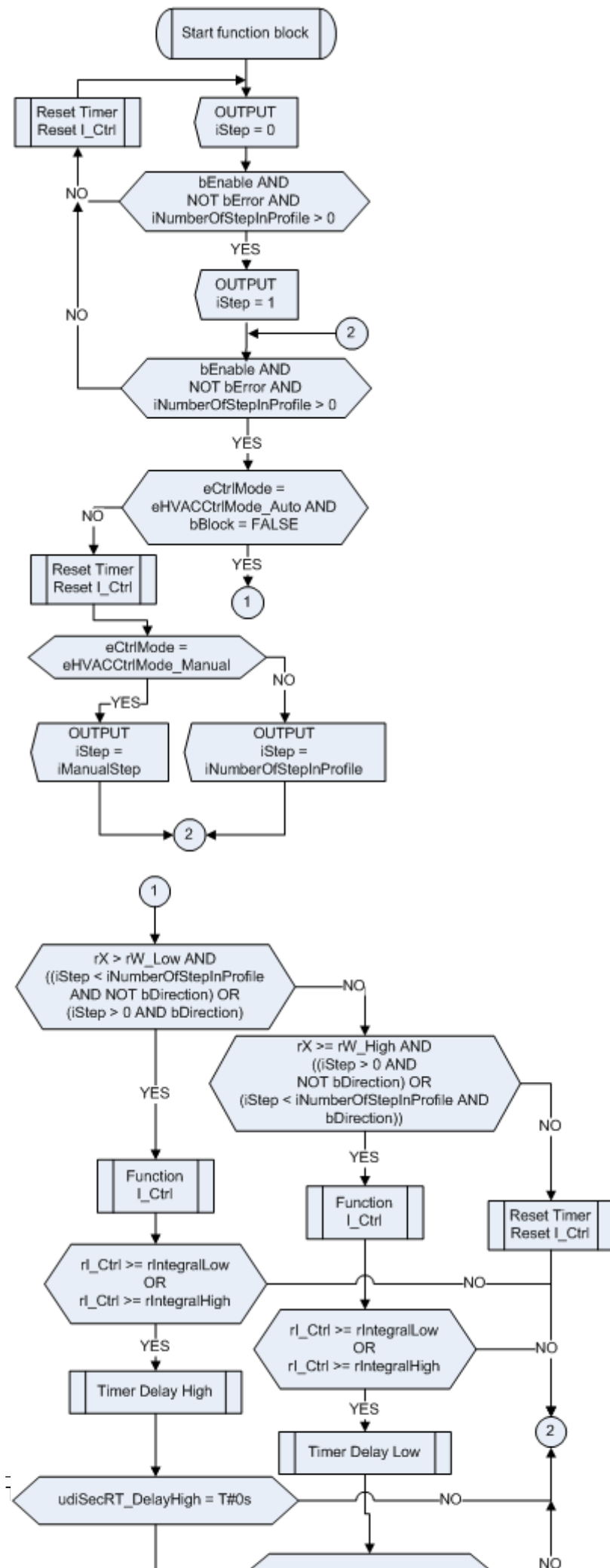
i The I transfer element and the internal time delays ($udiSecDelayLow$, $udiSecDelayHigh$) are reset after each change of $iStep$.

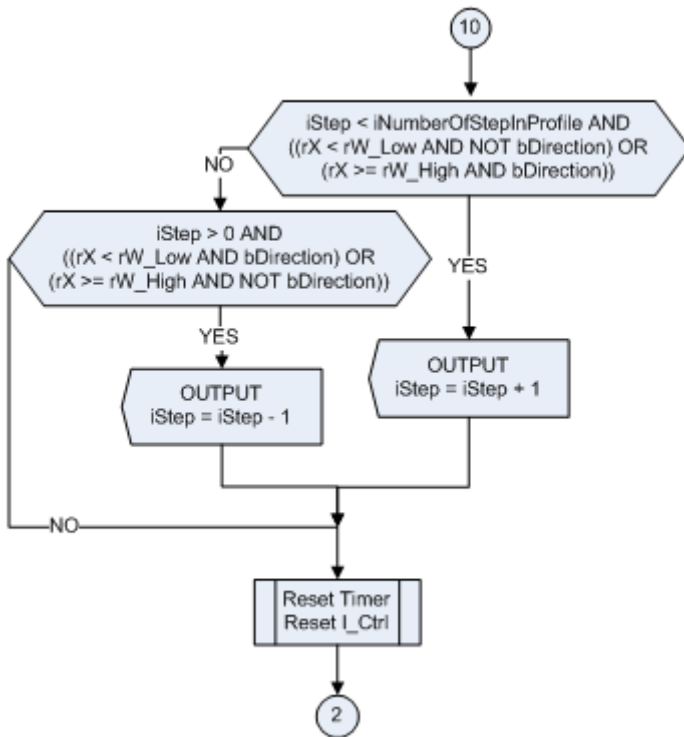
Transfer function of the I transfer element (I_Ctrl)

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

Program flowchart

Program flowchart





Program flowchart

Program flowchart

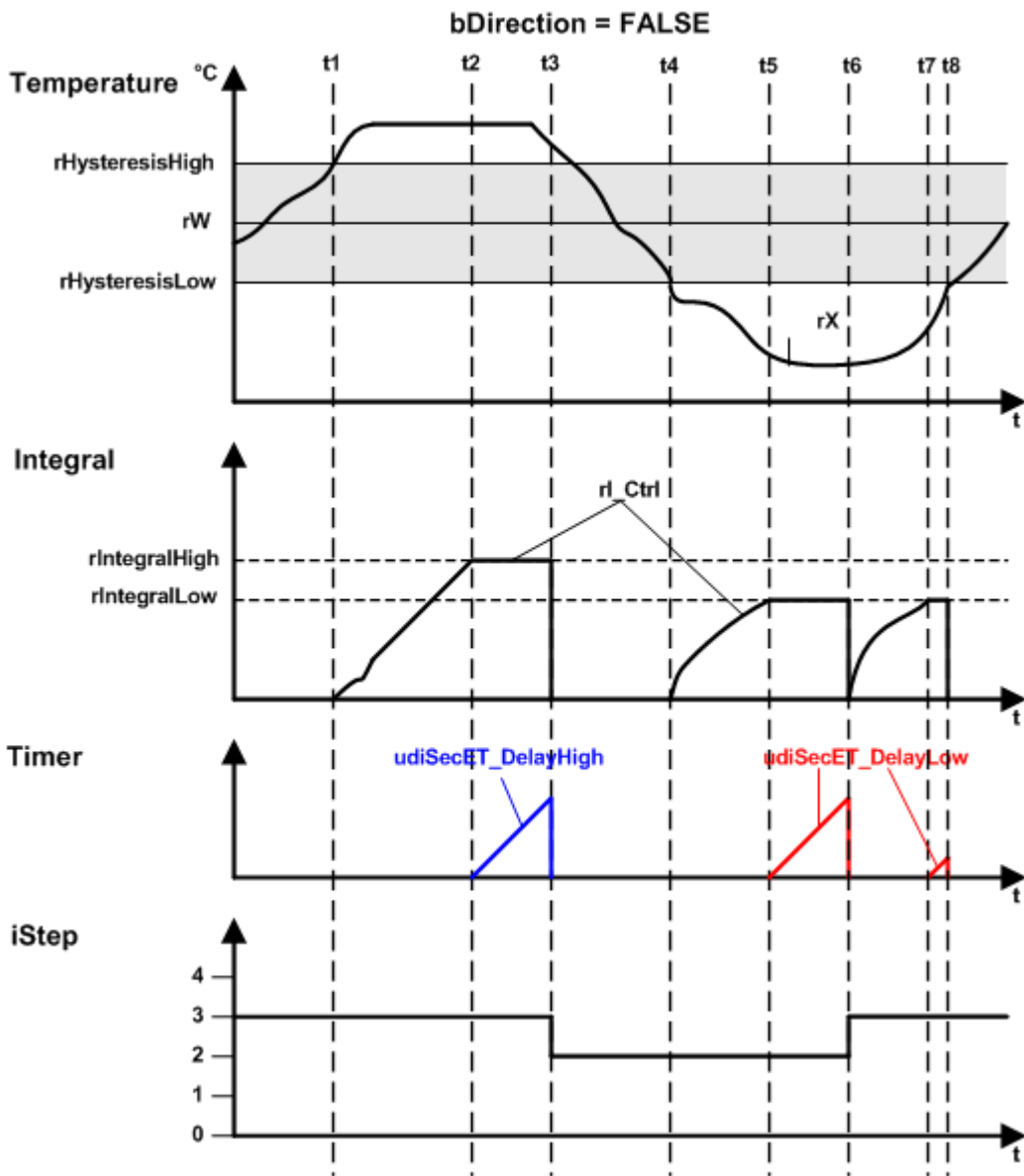


Fig. 12: FB_HVACI_CtrlStepDiagramm

Behavior of the different variables

Behavior of the different variables

The figure shows when each variable is used and how.

Application example

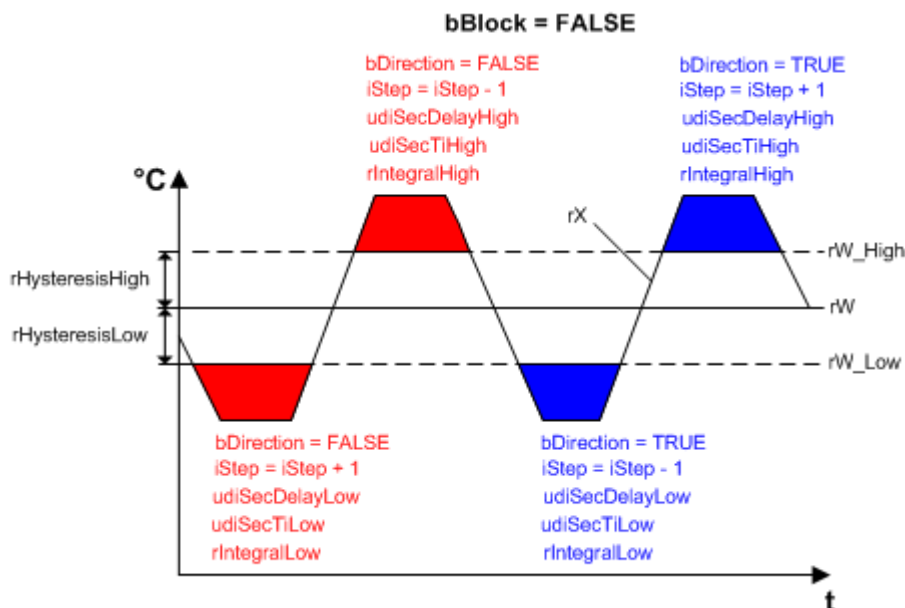


Fig. 13: FB_HVACI_CtrlStepParameter

The application example shows the function block FB_HVACI_CtrlStep in conjunction with the power range table [FB_HVACPowerRangeTable](#) [► 275]. The example is available in the programming languages ST and CFC. The program example **P_CFC_I_CtrlStep.PRG** in CFC can be found in the folder **Language CFC > Controller**, the program example **P_ST_I_CtrlStep.PRG** in ST in the folder **Language Structure Text > Controller**.

Download	Required library
TcHVAC.pro [► 531]	TcHVAC.lib

VAR_INPUT

```

eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
rW                : REAL;
rX                : REAL;
eCtrlMode         : E_HVACCtrlMode;
iManualStep       : INT;           0..iNumberOfStepInProfile
bStepUp           : BOOL;
bStepDown         : BOOL;
bDirection        : BOOL;
iNumberOfStepInProfile: INT;       0..g_iMaxNumberOfSteps
bBlock            : BOOL;
bReset            : BOOL;

```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instantiated once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example: <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled via TRUE. If `bEnable = FALSE`, then `iStep` is set constantly to 0. The check of the variable `iNumberOfStepInProfile` remains active. If an error occurs, it is displayed with `bError = TRUE` and can be acknowledged with `bReset` once the fault has been corrected.

rW: The setpoint is transferred with the variable `rW`.

rX: the actual value is transferred with the variable `rX`.

eCtrlMode: Enum that specifies the operation mode of the function block. If `bEnable = TRUE AND bError = FALSE AND eCtrlMode = eHVACCtrlMode_Manual AND iNumberOfStepInProfile > 0`, then the value of the output `iStep` can be specified via `iManualStep`. On starting the PLC, `eCtrlMode = eHVACCtrlMode_Auto`.

iManualStep: if the operation mode `eCtrlMode = eHVACCtrlMode_Manual AND bEnable = TRUE AND bError = FALSE AND iNumberOfStepInProfile > 0`, then the value of the output `iStep` can be specified via `iManualStep`. The input range of `iManualStep` can be 0 at the least and the value of `iNumberOfStepInProfile` at the most.

bStepUp: if `bEnable = TRUE AND bError = FALSE AND eCtrlMode = eHVACCtrlMode_Auto AND bBlock = FALSE`, then the value of the output `iStep` can be incremented by 1 by a rising edge at the input `bStepUp`. This can be repeated until `iStep = iNumberOfStepInProfile`.

bStepDown: if `bEnable = TRUE AND bError = FALSE AND eCtrlMode = eHVACCtrlMode_Auto AND bBlock = FALSE`, then the value of the output `iStep` can be decremented by 1 by a rising edge at the input `bStepDown`. This can be repeated until `iStep = 0`.

bDirection: the control direction of the function block is determined by `bDirection`. FALSE = heating mode; TRUE = cooling mode

iNumberOfStepInProfile: number of steps in the power step sequence. `iNumberOfStepInProfile` can be minimum 0 and maximum `g_iMaxNumberOfSteps` [► 531]. If the limit values are not adhered to, an error is output and displayed by `bError = TRUE` and `iStep` becomes 0.

bBlock: using the input variable `bBlock`, the control of the function block can either be released or the output `iStep` is set to the value of the variable `iNumberOfStepInProfile`.

If `bEnable = TRUE AND bError = FALSE AND eCtrlMode = eHVACCtrlMode_Auto AND bBlock = FALSE`, then the I transfer element and the internal timing elements are enabled for the control of the output `iStep`.

If `bEnable = TRUE AND bError = FALSE AND eCtrlMode = eHVACCtrlMode_Auto AND bBlock = TRUE`, then the I transfer element and the internal timing elements are blocked and `iStep = iNumberOfStepInProfile`.

bReset: input for acknowledgement of the faults once they have been corrected. Internally the system responds to a rising edge.

VAR_OUTPUT

<code>iStep</code>	: INT;	0..iNumberOfStepInProfile
<code>rI_Ctrl</code>	: REAL;	
<code>rE</code>	: REAL;	
<code>rW_High</code>	: REAL;	
<code>rW_Low</code>	: REAL;	
<code>b_rW_High</code>	: BOOL;	
<code>b_rW_Low</code>	: BOOL;	
<code>udiSecRT_DelayHigh</code>	: UDINT;	Second Remaining Time Delay High
<code>udiSecRT_DelayLow</code>	: UDINT;	Second Remaining Time Delay Low
<code>bError</code>	: BOOL;	
<code>eErrorCode</code>	: E_HVACErrorCodes;	

iStep: the output variable *iStep* indicates the step in a power step sequence. When the function block is started, *iStep* = 1 if *bEnable* = TRUE, *bError* = FALSE, *eCtrlMode* = *eHVACCtrlMode_Auto* AND *iNumberOfStepInProfile* > 0. *iStep* can be at most the value of *iNumberOfStepInProfile* and at least 0.

The value of *iStep* can be manually specified via *iManualStep*, if *bEnable* = TRUE, *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Manual*.

The value of *iStep* can be incremented or decremented via the input variables *bStepUp* / *bStepDown*, if *bEnable* = TRUE, *bError* = FALSE, *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bBlock* = FALSE.

If *bEnable* = TRUE, *bError* = FALSE, *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bBlock* = FALSE and the value of *iStep* has changed, then the delay timing elements and the I transfer element are reset.

If *bEnable* = TRUE, *bError* = FALSE, *eCtrlMode* = *eHVACCtrlMode_Auto*, *bBlock* = FALSE AND *iNumberOfStepInProfile* > 0, the output of the I transfer element *rl_Ctrl* = *rlIntegralHigh* or *rlIntegralLow* and one of the delay times *tRemainingTimeDelayHigh* / *tRemainingTimeDelayLow* = T#0s, then the output *iStep* is controlled as follows:

$iStep = iStep + 1$ if

$(rX < rW_Low \text{ AND } iStep < iNumberOfStepInProfile \text{ AND NOT } bDirection)$

OR

$(rX \geq rW_High \text{ AND } iStep < iNumberOfStepInProfile \text{ AND } bDirection)$.

$iStep = iStep - 1$ if

$(rX < rW_Low \text{ AND } iStep > 0 \text{ AND } bDirection)$

OR

$(rX \geq rW_High \text{ AND } iStep > 0 \text{ AND NOT } bDirection)$, see [Program flowchart \[► 250\]](#)

If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bBlock* = TRUE, then the I transfer element and the internal timing elements are blocked and *iStep* = *iNumberOfStepInProfile*.

rl_Ctrl: output of the I transfer element.

If *rl_Ctrl* ≥ *rlIntegralHigh* or *rlIntegralLow* and *tRemainingTimeDelayHigh* or *tRemainingTimeDelayLow* = T#0s, then the number of steps is incremented or decremented by 1 via *iStep*. Subsequently, the I transfer element and the timing elements for stepping up or down are reset, so that stepping up or down is restarted, see [Program flowchart \[► 250\]](#).

rE: control deviation with which the internal I transfer element works: $rE = rW - rX$

rW_High: $rW_High := rW + rHysteresisHigh$ - upper setpoint limit which, when exceeded by *rX*, causes the I transfer element to be activated first, followed by the timing element of the delay time *udiSecDelayHigh*; see [Application \[► 249\]](#) or [Behavior of different variables \[► 253\]](#).

rW_Low: $rW_Low := rW - rHysteresisLow$ - lower setpoint limit which, when undershot by *rX*, causes the I transfer element to be activated first, followed by the timing element of the delay time *udiSecDelayLow*; see [Application \[► 249\]](#) or [Behavior of different variables \[► 253\]](#).

b_rW_High: *b_rW_High* becomes TRUE, if $rX > rW_High$.

b_rW_Low: *b_rW_Low* becomes TRUE, if $rX < rW_Low$.

udiSecRT_DelayHigh: remaining time of delay time *udiSecDelayHigh*.

udiSecRT_DelayLow: remaining time of delay time *udiSecDelayLow*.

bError: the output signals with a TRUE that an error is present and an incorrect parameter is present at the variable *iNumberOfStepInProfile*. *iStep* is constantly set to 0 and the Enum *eErrorCode* indicates the error number. Once the fault has been corrected the *bError* message must be acknowledged with *bReset*.

eErrorCode: returns the error number [► 520] when the *bError* output is set. The following error can occur in this function block: *eHVACErrorCodes_InvalidParam_NumberOfStepInProfile*



To access the enum error numbers in the PLC, `eErrorCode` can be assigned to a variable of the data type `WORD.eHVACErrorCodes_Error_iNumberOfStepInProfil= 32`

VAR_IN_OUT

```
rHysteresisHigh      : REAL;
rHysteresisLow       : REAL;
udiSecDelayHigh      : UDINT;
udiSecDelayLow       : UDINT;
udiSecTiHigh         : UDINT;
udiSecTiLow          : UDINT;
rIntegralHigh        : REAL;
rIntegralLow         : REAL;
```

rHysteresisHigh: positive value of the upper limit of the control deviation, see [Application \[▶ 249\]](#) or [Behavior of different variables \[▶ 253\]](#). $rW_High := rW + rHysteresisHigh$. The variable is saved persistently. Preset to 5.

rHysteresisLow: positive value of the lower limit of the control deviation, see [Application \[▶ 249\]](#) or [Behavior of different variables \[▶ 253\]](#). $rW_Low := rW - rHysteresisLow$. The variable is saved persistently. Preset to 5.

udiSecDelayHigh: delay time after whose expiry *iStep* is incremented or decremented; see [Application \[▶ 249\]](#) or [Behavior of different variables \[▶ 253\]](#). The variable is saved persistently. Preset to 300 s.

udiSecDelayLow: delay time after whose expiry *iStep* is incremented or decremented; see [Application \[▶ 249\]](#) or [Behavior of different variables \[▶ 253\]](#). The variable is saved persistently. Preset to 300 s.

udiSecTiHigh: integration time for the upper limit of the I transfer element in seconds, see [Application \[▶ 249\]](#) or [Behavior of different variables \[▶ 253\]](#). *udiSecTiHigh* must be > 0. Otherwise the default value or the last valid variable value is used internally. The variable is saved persistently. Preset to 60 s.

udiSecTiLow: integration time for the lower limit of the I transfer element, see [Application \[▶ 249\]](#) or [Behavior of different variables \[▶ 253\]](#). *udiSecTiLow* must be > 0. Otherwise the default value or the last valid variable value is used internally. The variable is saved persistently. Preset to 60 s.

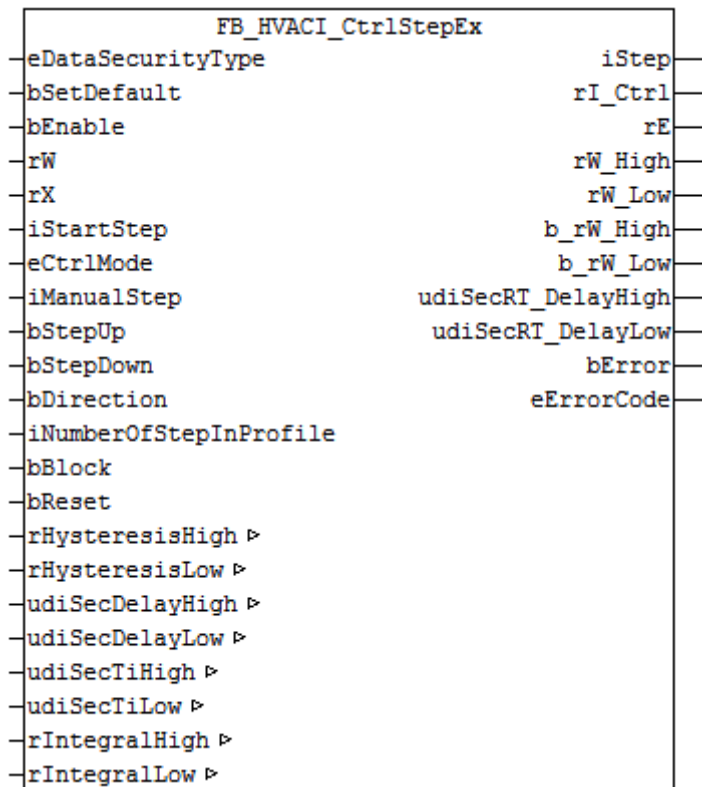
rIntegralHigh: positive value for the upper limit at which the integration of the I transfer element is stopped (ARW measure, anti-reset-windup) and one of the delay times starts; see [Application \[▶ 249\]](#) or [Behavior of different variables \[▶ 253\]](#). The variable is saved persistently. Preset to 15.

rIntegralLow: positive value for the lower limit at which the integration of the I transfer element is stopped (ARW measure, anti-reset-windup) and one of the delay times starts; see [Application \[▶ 249\]](#) or [Behavior of different variables \[▶ 253\]](#). The variable is saved persistently. Preset to 15.

Documents about this

 [example_persistent_e.zip \(Resources/zip/11659714827.zip\)](#)

3.4.3 FB_HVACI_CtrlStepEx



Application

Application

The function block serves the sequential control of power generators. It differs from [FB_HVACI_CtrlStep](#) [▶ 248] in that the value of *iStep* is specified by the variable *iStartStep* when starting the function block.

In conjunction with the power range table [FB_HVACPowerRangeTable](#) [▶ 275], the power step controller can be used for the stepped control of several heating boilers, refrigerating machines or recooling plants.

Switching up or down to the next higher or next lower power step respectively takes place via an integral (rW - rX) and a time delay. First of all, the actual temperature value rX must have exceeded or fallen below a threshold value rW_High / rW_Low. Subsequently, the integral starts. If the upper or lower limit value (rIntegralHigh / rIntegralLow) is reached at the output rI_Ctrl, then a timing element is started upon whose expiry (udiSecRT_DelayHigh / udiSecRT_DelayLow) the output iStep is incremented or decremented; see [Program flowchart](#) [▶ 261]

i An integration time setting of udiSecTiHigh/udiSecTiLow = 60 seconds means that the I transfer element changes by one Kelvin per minute. With a control deviation rE = 5, the output of the I transfer element rI_Ctrl has the value 5 after one minute. Example: bDirection = FALSE; udiSecTiLow = 60; rE = 2; rIntegralLow = 10 With the remaining control deviation of 2 and the lower integral limit of 10, after 5 minutes rI_Ctrl = rIntegralLow. The consequence of this is that a timing element with the delay time udiSecDelayLow is started. After its expiry (udiSecRT_DelayHigh / udiSecRT_DelayLow), the output iStep is incremented by 1 and the I transfer element and the time delays are reset.

i iStep is set to iStartStep for one PLC cycle if bEnable = TRUE AND bError = FALSE.

Conditions

The I transfer element is released, if

$bEnable = TRUE \text{ AND NOT } bError \text{ AND } eCtrlMode = eHVACCtrlMode_Auto \text{ AND NOT } bBlock$
 (
 $(rX \geq rW_High \text{ AND } ((iStep > 0 \text{ AND NOT } bDirection) \text{ OR } (iStep < iNumberOfStepInProfile \text{ AND } bDirection)))$
 OR
 $(rX < rW_Low \text{ AND } ((iStep < iNumberOfStepInProfile \text{ AND NOT } bDirection) \text{ OR } (iStep > 0 \text{ AND } bDirection)))$
)

. If $rI_Ctrl = rIntegralHigh / rIntegralLow$, then the timing elements of the delay times $udiSecDelayHigh / udiSecDelayLow$ are activated. After the expiry of the delay times $udiSecRT_DelayHigh / udiSecRT_DelayLow$, the steps of the power step sequence are controlled as follows:

$iStep = iStep + 1$ if

$(rX < rW_Low \text{ AND } iStep < iNumberOfStepInProfile \text{ AND NOT } bDirection)$
 OR
 $(rX \geq rW_High \text{ AND } iStep < iNumberOfStepInProfile \text{ AND } bDirection)$

$iStep = iStep - 1$ if

$(rX < rW_Low \text{ AND } iStep > 0 \text{ AND } bDirection)$
 OR
 $(rX \geq rW_High \text{ AND } iStep > 0 \text{ AND NOT } bDirection)$



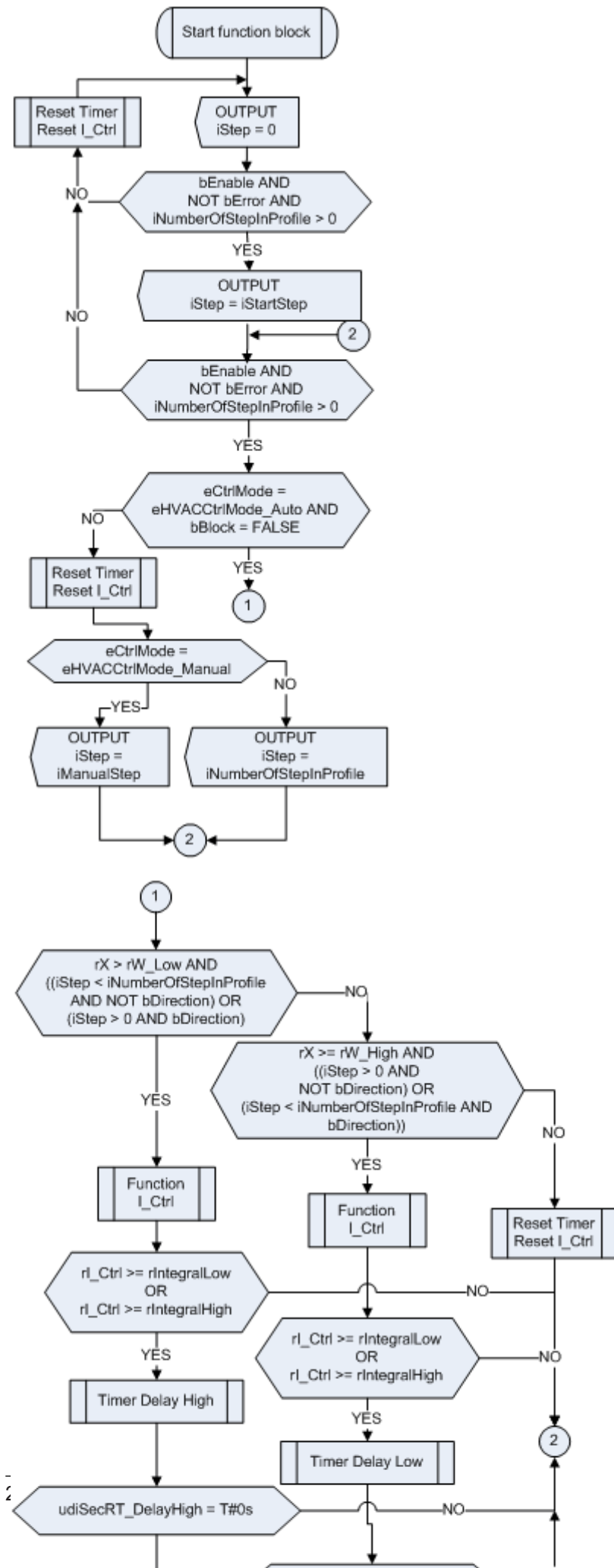
The I transfer element and the internal time delays ($udiSecDelayLow$, $udiSecDelayHigh$) are reset after each change of $iStep$.

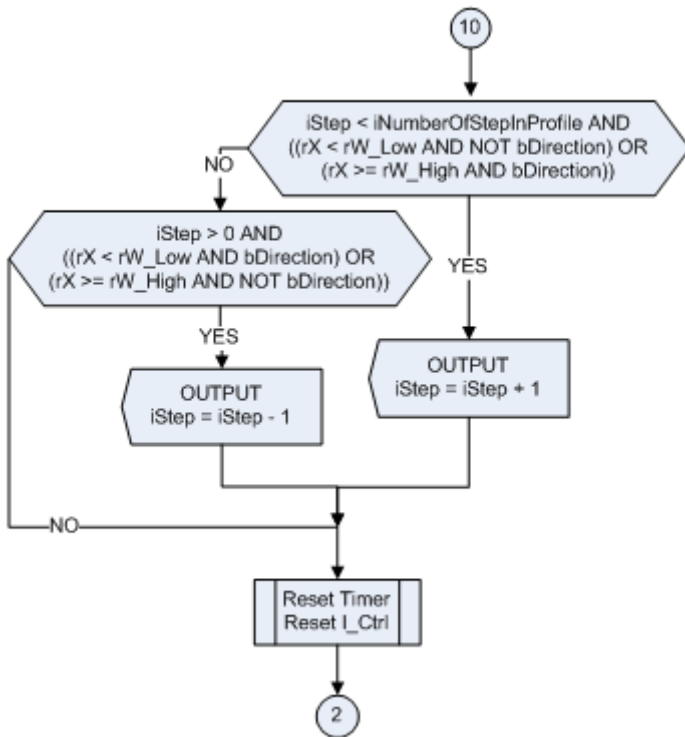
Transfer function of the I transfer element (I_Ctrl)

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

Program flowchart

Program flowchart





Program flowchart

Program flowchart

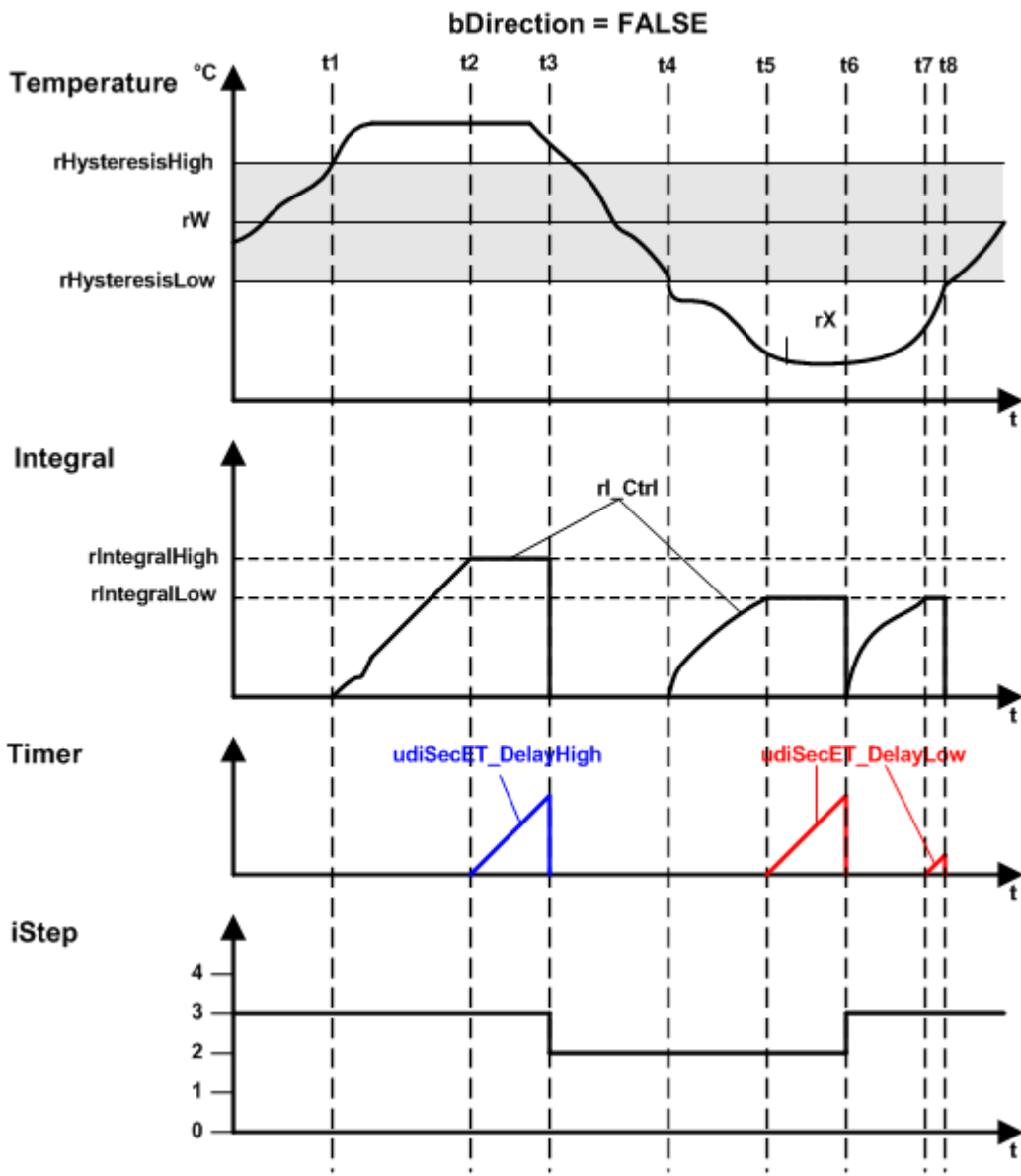


Fig. 14: FB_HVACI_CtrlStepDiagramm

Behavior of the different variables

Behavior of the different variables

The figure shows when each variable is used and how.

Application example

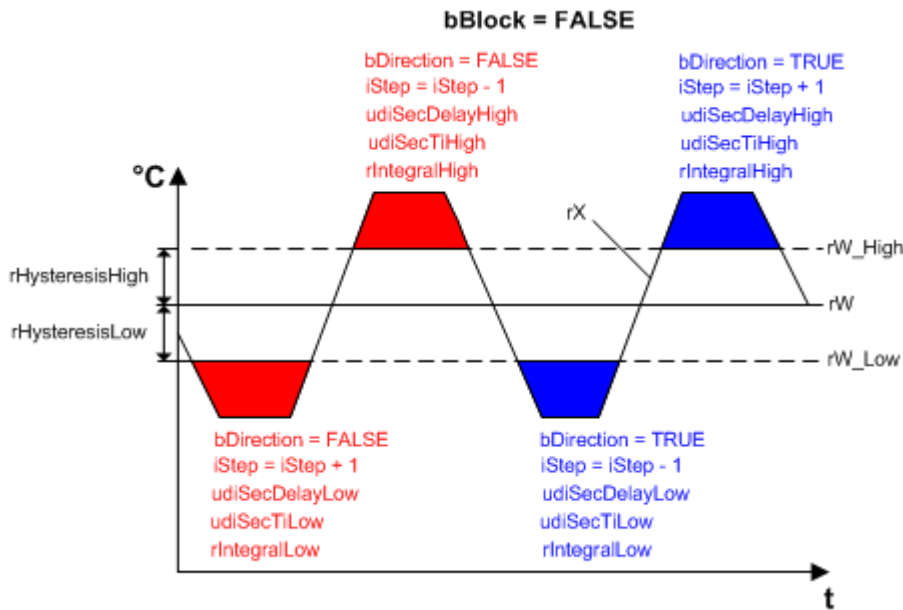


Fig. 15: FB_HVACI_CtrlStepParameter

The application example shows the function block FB_HVACI_CtrlStep in conjunction with the power range table [FB_HVACPowerRangeTable](#) [▶ 275]. The example is available in the programming languages ST and CFC. The program example **P_CFC_I_CtrlStep.PRG** in CFC can be found in the folder **Language CFC > Controller**, the program example **P_ST_I_CtrlStep.PRG** in ST in the folder **Language Structure Text > Controller**.

Download	Required library
TcHVAC.pro [▶ 531]	TcHVAC.lib

VAR_INPUT

```

eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
rW                : REAL;
rX                : REAL;
iStartStep        : INT;           0..iNumberOfStepInProfile
eCtrlMode         : E_HVACCtrlMode;
iManualStep       : INT;           0..iNumberOfStepInProfile
bStepUp           : BOOL;
bStepDown         : BOOL;
bDirection        : BOOL;
iNumberOfStepInProfile : INT;      0..g_iMaxNumberOfSteps
bBlock            : BOOL;
bReset            : BOOL;
    
```

eDataSecurityType: if `eDataSecurityType:= eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instantiated once in the main program, which is called cyclically. Otherwise the instantiated FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted the saved data are automatically read back from the flash into the RAM.

Application example: <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled via TRUE. If `bEnable = FALSE`, then `iStep` is set constantly to 0. The check of the variable `iNumberOfStepInProfile` remains active. If an error occurs, it is displayed with `bError = TRUE` and can be acknowledged with `bReset` once the fault has been corrected.

rW: The setpoint is transferred with the variable `rW`.

rX: the actual value is transferred with the variable `rX`.

iStartStep: if the function block is enabled (`bEnable = TRUE`) and there is no error (`bError = FALSE`), then `iStep = iStartStep` for one PLC cycle.

eCtrlMode: Enum that specifies the operation mode of the function block. If `bEnable = TRUE AND bError = FALSE AND eCtrlMode = eHVACCtrlMode_Manual AND iNumberOfStepInProfile > 0`, then the value of the output `iStep` can be specified via `iManualStep`. On starting the PLC, `eCtrlMode = eHVACCtrlMode_Auto`.

iManualStep: if the operation mode `eCtrlMode = eHVACCtrlMode_Manual AND bEnable = TRUE AND bError = FALSE AND iNumberOfStepInProfile > 0`, then the value of the output `iStep` can be specified via `iManualStep`. The input range of `iManualStep` can be 0 at the least and the value of `iNumberOfStepInProfile` at the most.

bStepUp: if `bEnable = TRUE AND bError = FALSE AND eCtrlMode = eHVACCtrlMode_Auto AND bBlock = FALSE`, then the value of the output `iStep` can be incremented by 1 by a rising edge at the input `bStepUp`. This can be repeated until `iStep = iNumberOfStepInProfile`.

bStepDown: if `bEnable = TRUE AND bError = FALSE AND eCtrlMode = eHVACCtrlMode_Auto AND bBlock = FALSE`, then the value of the output `iStep` can be decremented by 1 by a rising edge at the input `bStepDown`. This can be repeated until `iStep = 0`.

bDirection: the control direction of the function block is determined by `bDirection`. FALSE = heating mode; TRUE = cooling mode

iNumberOfStepInProfile: number of steps in the power step sequence. `iNumberOfStepInProfile` can be minimum 0 and maximum `g_iMaxNumberOfSteps` [► 531]. If the limit values are not adhered to, an error is output and displayed by `bError = TRUE` and `iStep` becomes 0.

bBlock: using the input variable `bBlock`, the control of the function block can either be released or the output `iStep` is set to the value of the variable `iNumberOfStepInProfile`.

If `bEnable = TRUE AND bError = FALSE AND eCtrlMode = eHVACCtrlMode_Auto AND bBlock = FALSE`, then the I transfer element and the internal timing elements are enabled for the control of the output `iStep`.

If `bEnable = TRUE AND bError = FALSE AND eCtrlMode = eHVACCtrlMode_Auto AND bBlock = TRUE`, then the I transfer element and the internal timing elements are blocked and `iStep = iNumberOfStepInProfile`.

bReset: input for acknowledgement of the faults once they have been corrected. Internally the system responds to a rising edge.

VAR_OUTPUT

```

iStep          : INT;          0..iNumberOfStepInProfile
rI_Ctrl        : REAL;
rE             : REAL;
rW_High        : REAL;
rW_Low         : REAL;
b_rW_High      : BOOL;
b_rW_Low       : BOOL;
udiSecRT_DelayHigh : UDINT;    Second Remaining Time Delay High
udiSecRT_DelayLow  : UDINT;    Second Remaining Time Delay Low
bError         : BOOL;
eErrorCode     : E_HVACErrorCodes;

```


iStep: the output variable *iStep* indicates the step in a power step sequence. When starting the function block, $iStep = iStartStep$, if $bEnable = TRUE$ AND $bError = FALSE$. *iStep* can reach at most the value of *iNumberOfStepInProfile* and be at least 0.

The value of *iStep* can be manually specified via *iManualStep*, if $bEnable = TRUE$, $bError = FALSE$ AND $eCtrlMode = eHVACCtrlMode_Manual$.

The value of *iStep* can be incremented or decremented via the input variables *bStepUp* / *bStepDown*, if $bEnable = TRUE$, $bError = FALSE$, $eCtrlMode = eHVACCtrlMode_Auto$ AND $bBlock = FALSE$.

If $bEnable = TRUE$, $bError = FALSE$, $eCtrlMode = eHVACCtrlMode_Auto$ AND $bBlock = FALSE$ and the value of *iStep* has changed, then the delay timing elements and the I transfer element are reset.

If $bEnable = TRUE$, $bError = FALSE$, $eCtrlMode = eHVACCtrlMode_Auto$, $bBlock = FALSE$ AND $iNumberOfStepInProfile > 0$, the output of the I transfer element $rI_Ctrl = rIntegralHigh$ or $rIntegralLow$ and one of the delay times $tRemainingTimeDelayHigh$ / $tRemainingTimeDelayLow = T\#0s$, then the output *iStep* is controlled as follows:

$iStep = iStep + 1$ if

$(rX < rW_Low$ AND $iStep < iNumberOfStepInProfile$ AND NOT $bDirection$)

OR

$(rX \geq rW_High$ AND $iStep < iNumberOfStepInProfile$ AND $bDirection$).

$iStep = iStep - 1$ if

$(rX < rW_Low$ AND $iStep > 0$ AND $bDirection$)

OR

$(rX \geq rW_High$ AND $iStep > 0$ AND NOT $bDirection$), see [Program flowchart \[▶ 259\]](#)

If $bEnable = TRUE$ AND $bError = FALSE$ AND $eCtrlMode = eHVACCtrlMode_Auto$ AND $bBlock = TRUE$, then the I transfer element and the internal timing elements are blocked and $iStep = iNumberOfStepInProfile$.

rI_Ctrl: output of the I transfer element.

If $rI_Ctrl \geq rIntegralHigh$ or $rIntegralLow$ and $tRemainingTimeDelayHigh$ or $tRemainingTimeDelayLow = T\#0s$, then the number of steps is incremented or decremented by 1 via *iStep*. Subsequently, the I transfer element and the timing elements for stepping up or down are reset, so that stepping up or down is restarted, see [Program flowchart \[▶ 259\]](#).

rE: control deviation with which the internal I transfer element works: $rE = rW - rX$

rW_High: $rW_High := rW + rHysteresisHigh$ - upper setpoint limit which, when exceeded by *rX*, causes the I transfer element to be activated first, followed by the timing element of the delay time *udiSecDelayHigh*; see [Application \[▶ 258\]](#) or [Behavior of different variables \[▶ 262\]](#).

rW_Low: $rW_Low := rW - rHysteresisLow$ - lower setpoint limit which, when undershot by *rX*, causes the I transfer element to be activated first, followed by the timing element of the delay time *udiSecDelayLow*; see [Application \[▶ 258\]](#) or [Behavior of different variables \[▶ 262\]](#).

b_rW_High: *b_rW_High* becomes TRUE, if $rX > rW_High$.

b_rW_Low: *b_rW_Low* becomes TRUE, if $rX < rW_Low$.

udiSecRT_DelayHigh: remaining time of delay time *udiSecDelayHigh*.

udiSecRT_DelayLow: remaining time of delay time *udiSecDelayLow*.

bError: the output signals with a TRUE that an error is present and an incorrect parameter is present at the variable *iNumberOfStepInProfile*. *iStep* is constantly set to 0 and the Enum *eErrorCode* indicates the error number. Once the fault has been corrected the *bError* message must be acknowledged with *bReset*.

eErrorCode: returns the [error number \[▶ 520\]](#) when the *bError* output is set. The following error can occur in this function block: *eHVACErrorCodes_InvalidParam_NumberOfStepInProfile*



To access the enum error numbers in the PLC, *eErrorCode* can be assigned to a variable of the data type WORD. *eHVACErrorCodes_Error_iNumberOfStepInProfile* = 32

VAR_IN_OUT

```

rHysteresisHigh      : REAL;
rHysteresisLow       : REAL;
udiSecDelayHigh      : UDINT;
udiSecDelayLow       : UDINT;
udiSecTiHigh         : UDINT;
udiSecTiLow          : UDINT;
rIntegralHigh        : REAL;
rIntegralLow         : REAL;

```

rHysteresisHigh: positive value of the upper limit of the control deviation, see [Application \[▶ 258\]](#) or [Behavior of different variables \[▶ 262\]](#). $rW_High := rW + rHysteresisHigh$ The variable is saved persistently. Preset to 5.

rHysteresisLow: positive value of the lower limit of the control deviation, see [Application \[▶ 258\]](#) or [Behavior of different variables \[▶ 262\]](#). $rW_Low := rW - rHysteresisLow$ The variable is saved persistently. Preset to 5.

udiSecDelayHigh: delay time after whose expiry $iStep$ is incremented or decremented; see [Application \[▶ 258\]](#) or [Behavior of different variables \[▶ 262\]](#). The variable is saved persistently. Preset to 300 s.

udiSecDelayLow: delay time after whose expiry $iStep$ is incremented or decremented; see [Application \[▶ 258\]](#) or [Behavior of different variables \[▶ 262\]](#). The variable is saved persistently. Preset to 300 s.

udiSecTiHigh: integration time for the upper limit of the I transfer element in seconds, see [Application \[▶ 258\]](#) or [Behavior of different variables \[▶ 262\]](#). $udiSecTiHigh$ must be > 0 . Otherwise the default value or the last valid variable value is used internally. The variable is saved persistently. Preset to 60 s.

udiSecTiLow: integration time for the lower limit of the I transfer element, see [Application \[▶ 258\]](#) or [Behavior of different variables \[▶ 262\]](#). $udiSecTiLow$ must be > 0 . Otherwise the default value or the last valid variable value is used internally. The variable is saved persistently. Preset to 60 s.

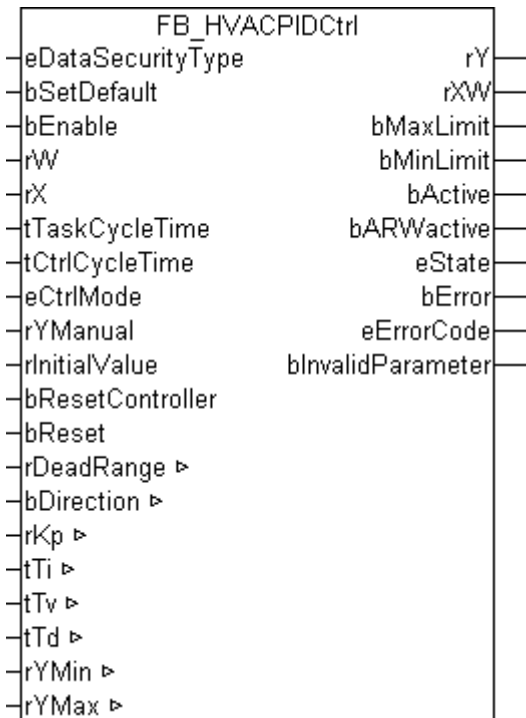
rIntegralHigh: positive value for the upper limit at which the integration of the I transfer element is stopped (ARW measure, anti-reset-windup) and one of the delay times starts; see [Application \[▶ 258\]](#) or [Behavior of different variables \[▶ 262\]](#). The variable is saved persistently. Preset to 15.

rIntegralLow: positive value for the lower limit at which the integration of the I transfer element is stopped (ARW measure, anti-reset-windup) and one of the delay times starts; see [Application \[▶ 258\]](#) or [Behavior of different variables \[▶ 262\]](#). The variable is saved persistently. Preset to 15.

Documents about this

 example_persistent_e.zip (Resources/zip/11659714827.zip)

3.4.4 FB_HVACPIDCtrl



Application


The PID controller is a standard controller for heating and air conditioning applications. It can be used to control temperatures, pressures, volumetric flows or humidity. The values of Ti and Tv can be set to zero. This allows a P, PI, or PD controller characteristic to be set. The controller has an ARW function (Anti-Reset-Windup). This prevents continuous integration in the event of a perpetual pending control deviation.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
bEnable          : BOOL;
rW               : REAL;
rX               : REAL;
tTaskCycleTime  : TIME;
tCtrlCycleTime  : TIME;
eCtrlMode        : E_HVACCtrlMode;
rYManual         : REAL;
rInitialValue   : REAL;
bResetController: BOOL;
bReset           : BOOL;
```

eDataSecurityType: if `eDataSecurityType:= eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: input variable for enabling the controller. The controller is active if `bEnable = TRUE`.

rW: the setpoint is transferred to the controller with the variable `rW`.

rX: `rX` acquires the actual value of the control loop..

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: the variable `tCtrlCycleTime` specifies the cycle time with which the PID controller is processed. The shortest possible cycle time is that of the controller. Since the controlled systems in building automation are predominantly slow, the cycle time of the controller can be several times that of the control cycle time.

eCtrlMode: the operation mode is selected via this enum.

rYManual: the controller output `rY` can be overridden for test purposes. The set value of `rYManual` will be found at the controller output in manual operation mode.

rInitialValue: the restart behavior of the controller is influenced by `rInitialValue`. The initial values are adopted at a rising edge of the controller enable `bEnable`.

0 = start with the value zero at the output `rY`

<>0 = start with the value of `rInitialValue`

bResetController: a positive edge on the input `bResetController` resets the PID controller.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
rY           : REAL;
rXW          : REAL;
bMaxLimit    : BOOL;
bMinLimit    : BOOL;
bActive      : BOOL;
bARWactive   : BOOL;
eState       : E_HVACState;
bError       : BOOL;
eErrorCode   : E_HVACErrorCodes;
bInvalidParameter: BOOL;
```

rY: control signal output of the PID controller.

rXW: control deviation.

bMaxLimit: the output `bMaxLimit` is TRUE, if the output `rY` has reached the value `rYMax`.

bMinLimit: the output `bMinLimit` is TRUE, if the output `rY` has reached the value `rYMin`.

bActive: `bActive` is TRUE, if the controller is active and enabled.

bARWactive: `bARWactive` is TRUE, if the integral component of the controller has reached the lower or upper control value limit.

eState: state of the controller. See [E_HVACState](#). [[▶ 523](#)]

bError: this output indicates with a TRUE that there is an error.

eErrorCode: contains the command-specific error code. See [E_HVACErrorCodes](#) [[▶ 520](#)].

bInvalidParameter: TRUE if an error occurs during the plausibility check. The message must be acknowledged with `bReset`.

VAR_IN_OUT

```

rDeadRange      : REAL;
bDirection      : BOOL;
rKp             : REAL;
tTi            : TIME;
tTv            : TIME;
tTd            : TIME;
rYMin          : REAL;
rYMax          : REAL;

```

rDeadRange: in order to avoid unnecessary driving and hence premature wear of the valves or damper drives, a dead range (0..32767) can be set for the controller output signal *rY*. This means that a control signal change is only active if the change of value is greater than the dead range. A constant change of the control signal *rY* is converted to a pulsating drive of the actuator if a dead range is specified. The larger the dead range the larger the pauses and the control signal jumps will be. The variable is saved persistently. Preset to 0.

bDirection: the control direction of the controller can be changed with the parameter *bDirection*. If *bDirection* is TRUE, then the direct control direction is active for cooling operation of the controller. If *bDirection* is FALSE, then the indirect control direction of the controller is activated for heating operation. The variable is saved persistently. Preset to 0.

rKp: proportional factor (0.01..100) gain. The variable is saved persistently. Preset to 1.0.

tTi: integral action time [s]. The I-part corrects the residual control deviation following correction of the P-part. The smaller the *tTi* time is set, the faster the controller corrects. The control loop becomes unstable if the time is too short. Larger *tTi*-times must be entered in order to reduce the integration component. The integral action time should be selected to be longer than the stroke time of the valve or damper drive. The variable is saved persistently. Preset to 30 s.

tTv: rate time [s]. The larger *tTv* is, the stronger the controller corrects. The control loop becomes unstable if the time is too long. Often in normal building automation applications only a PI controller is used. In this case zero must be entered for *tTv*. The variable is saved persistently. Preset to 0 s.

tTd : damping time [s]. The variable is saved persistently. Preset to 0 s.

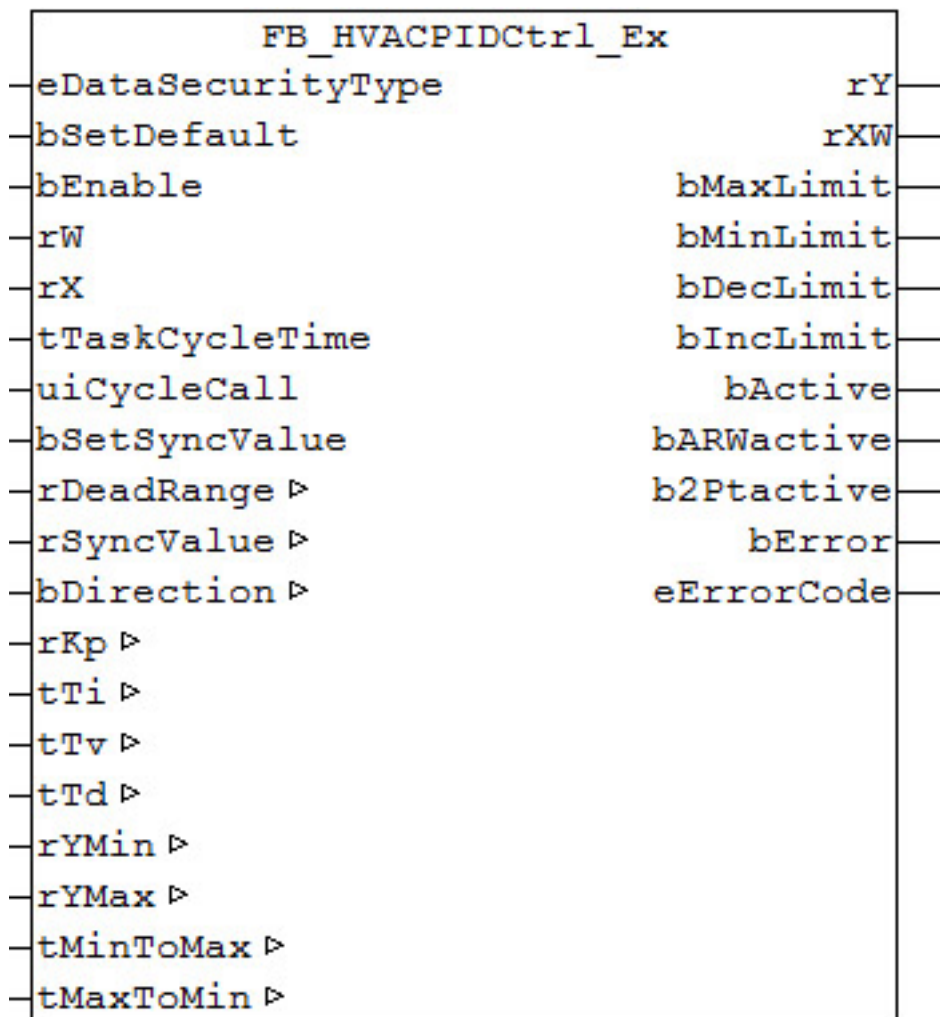
rYMin / rYMax: limiting the working range of the controller (0..32767). Several other function blocks, e.g. sequencers, require a symmetrical control range (-100 to +100). In the case of a cascade structure, the working range of the master controller determines the setpoint of the slave controller. For example, 15° to 25° as the limitation of the supply air set value for an exhaust/supply air cascade control. The variables are saved persistently. *rYMin* preset to 0. *rYMax* preset to 100.

Documents about this

 example_persistent_e.zip (Resources/zip/11659714827.zip)

3.4.5 FB_HVACPIDCtrl_Ex

Universal PID controller.




Inputs/outputs

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
rW                : REAL;
rX                : REAL;
tTaskCycleTime   : TIME;
uiCycleCall       : UINT;
bSetSyncValue     : BOOL;
```

eDataSecurityType: if `eDataSecurityType:= eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instantiated once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: controller activation. **At the moment of activation the controller reacts directly to the control deviation without internal synchronization to a value.**

rW : setpoint

rX : actual value

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.

uiCycleCall : call cycle of the function block as a multiple of the cycle time. A zero entry is automatically evaluated as `uiCycleCall =1`.

Example: `tTaskCycleTime = 20 ms, uiCycleCall =10` -> The control algorithm is called every 200 ms. Thus the outputs are also updated only every 200 ms.

bSetSyncValue: a rising edge at this input `bSetSyncValue` sets the control value `rY` to the value `rSyncValue` (VAR_IN_OUT). In addition, the I-component is changed internally. If the I-component doesn't exist (PD controller), the D-component is changed.

VAR_OUTPUT

```
rY           : REAL;
rXW          : REAL;
bMaxLimit    : BOOL;
bMinLimit    : BOOL;
bDecLimit    : BOOL;
bIncLimit    : BOOL;
bActive      : BOOL;
bARWactive   : BOOL;
b2Ptactive   : BOOL;
bError       : BOOL;
eErrorCode   : E_HVACErrorCodes;
```

rY : control value. Range limited by `rYMin` and `rYMax`.

rXW : control deviation (calculation dependent on [control direction](#) [[▶ 273](#)])

bMaxLimit : the control value has reached its upper limit value.

bMinLimit : the control value has reached its lower limit value.

bDecLimit : the control value slope has reached its limit value for the maximum decrease, see `tMaxToMin` (VAR_IN_OUT).

bIncLimit : the control value slope has reached its limit value for the maximum increase, see `tMinToMax` (VAR_IN_OUT).

bActive : the controller is active, i.e. enabled (`bEnable = TRUE`) and not in the error state (`bError = FALSE`).

bARWactive : anti-Reset-Windup function is active.

b2Ptactive : the 2-point behavior of the controller is active.

bError: this output is switched to TRUE if the parameters entered are erroneous.

eErrorCode: contains the command-specific error code. See [E_HVACErrorCodes](#) [[▶ 520](#)].

VAR_IN_OUT

```
rDeadRange   : REAL;
rSyncValue   : REAL;
bDirection   : BOOL;
rKp          : REAL;
```

```

tTi      : TIME;
tTv      : TIME;
tTd      : TIME;
rYMin    : REAL;
rYMax    : REAL;
tMinToMax : TIME;
tMaxToMin : TIME;

```

rDeadRange: a dead range can be set for the control deviation in order to avoid unnecessary movement and thus premature wear in the valves or damper drives. The P-I-D calculation and thus the control output *rY* are "frozen" if the value of the control deviation is smaller than half of the dead range *rDeadRange*. The variable is saved persistently. Preset to 0.

rSyncValue: a rising edge at this input *bSync* sets the control value *rY* to this value. In addition, the I-component is changed internally. If the I-component doesn't exist (PD controller), the D-component is changed. The variable is saved persistently. Preset to 0.

bDirection: the [control direction](#) [► 273] of the controller can be changed with the parameter *bDirection*. If *bDirection* is TRUE, then the direct control direction is active for cooling operation of the controller. If *bDirection* is FALSE, then the indirect control direction of the controller is activated for heating operation. The variable is saved persistently. Preset to 0.

rKp: proportional factor gain. Only affects the P-part. The variable is saved persistently. Preset to 1.0.

tTi: integral action time [s]. The I-part corrects the residual control deviation following correction of the P-part. The smaller *tTi* is set, the faster the controller corrects. The control loop becomes unstable if the time is too short. Larger *tTi*-times must be entered in order to reduce the integration component. The integral action time should be selected to be longer than the stroke time of the valve or damper drive. A zero value at this parameter disables the I-part. The variable is saved persistently. Preset to 30 s.

tTv: rate time [s]. The larger *tTv* is, the stronger the controller corrects. The control loop becomes unstable if the time is too long. Often in normal building automation applications only a PI controller is used. A zero value at this parameter disables the D-part. The variable is saved persistently. Preset to 0 s.

tTd : damping time [s]. The variable is saved persistently. Preset to 0 s.

rYMin / rYMax: limiting the working range of the controller. Several other function blocks, e.g. sequencers, require a symmetrical control range (-100 to +100). In the case of a cascade structure, the working range of the master controller determines the setpoint of the slave controller. For example, 15° to 25° as the limitation of the supply air set value for an exhaust/supply air cascade control. The variables are saved persistently. *rYMin* preset to 0. *rYMax* preset to 100.

tMinToMax: slope limit [s] of the controller output for increase: *tMinToMax* in seconds related to a change from *rYMin* to *rYMax*. The variable is saved persistently. Preset to 0 s.

tMaxToMin: slope limit [s] of the controller output for decrease: *tMaxToMin* in seconds related to a change from *rYMax* to *rYMin*. The variable is saved persistently. Preset to 0 s

Functional description

Passive behavior (bEnable = FALSE or bError = TRUE)

The outputs are set as follows:

rY	0.0
rXW	0.0
bMaxLimit	FALSE
bMinLimit	FALSE
bDeclimit	FALSE
bInclimit	FALSE
bActive	FALSE
bARWactive	FALSE
b2Ptactive	FALSE

In case of error *bError* is TRUE - *eErrorCode* indicates the current error code. The internal values for the P, I, and D components are set to 0, also the values for the I and D components of the preceding cycle. This ensures that the control value is "clean" in the first cycle after a restart, i.e. it is calculated without historical values.

Active behavior (bEnable = TRUE and bError = FALSE)

In the first cycle, the I and D components are calculated "clean", i.e. without historical values, as already mentioned. A positive signal on *bSetSyncValue* sets the I component such that the control value assumes the value *rSyncValue*. If *bEnable* and *bSetSyncValue* are set at the same time, this method can be used to set an initial value from which the controller "sets off". If the I component is not active, the D component is set accordingly. Note that only the rising edge of *bSetSyncValue* is evaluated internally as this is a setting action. A TRUE signal must be applied again to the input *bSetSyncValue* for renewed synchronization, for instance with a transfer value. If the I component is active, the controller ensures that it is retained, if the controller output *rY* is at the limits *rYMin* or *rYMax* and about to fall or increase further. This procedure is referred to as anti-wind-up. It ensures that the I-component is always only just sufficiently large to enable the control value to assume values within the limit immediately after a control deviation, without having to deal with an integral component that has become too large.

Control direction

Control direction

If *bDirection* = FALSE, the control direction of the controller is reversed so that a control deviation of less than 0 causes a change in the control value in the positive direction. This is achieved by a negative calculation of the control deviation:

bDirection	rXW (control deviation)	Control direction
TRUE	$IrX - IrW$ (actual value-set value)	direct (cooling)
FALSE	$IrW - IrX$ (set value-actual value)	indirect (heating)

Anti-Reset-Windup when the maximum or minimum value is reached

If the controller reaches its upper limit at the output and the control deviation is still positive, the integral component will continue to increase, until the control deviation is less than or equal to zero again. This may lead to an unnecessarily large integral component, which would have to be reduced again, if the sign of the control deviation changes and would make the control behavior sluggish. The same applies when the minimum value is reached at the output while the control deviation remains negative.

In order to prevent this, the I part is set in such a way that it reaches the respective limit value *IrYmin* or *IrYMax* at the control output in addition with the P and D part.

The further calculation of the P, I and D values is suspended until the sign of the control deviation allows the control range to be entered again; i.e. a control deviation of less than 0.0 when persisting at the maximum limit and or a control deviation of greater than 0.0 when persisting at the minimum limit.

In the PLC cycle of the re-entry also, the output *IrY* is set by manipulating the I component so that it doesn't move erratically in the control range, but starts to change from the limit of the preceding persistence.

Slope limitation

If the controller is set faster than the actuator, it is unable to follow the controller, which can lead to jitter. It is therefore possible to limit the slope of the control value.

It is based on the following parameters:

tMinToMax : slope limit of controller output for increase: *tMinToMax* in seconds related to a change from *rYMin* to *rYMax*.

tMaxToMin : slope limit of controller output for decrease: *tMaxToMin* in seconds related to a change from *rYMax* to *rYMin*.

This can be used to calculate the maximum change per PLC cycle (maximum increment or decrement).

If the calculated change of the control signal over a PLC cycle is now higher than that set under $tMinToMax$ or $tMaxToMin$, then the control signal is merely increased or decreased respectively by the maximum increment or decrement.

Internally, the I component is automatically adjusted in the same way (I component of last PLC cycle + maximum increment or I component of the last PLC cycle - maximum decrement).

Dead band

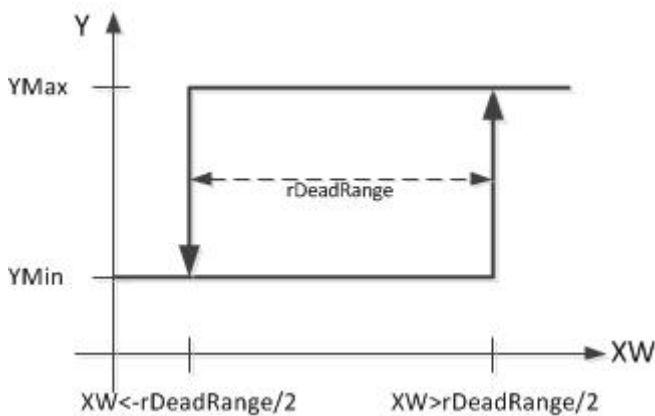
A value of $rDeadRange > 0.0$ enables the dead range function. If the absolute value of the control deviation is then less than $1/2 rDeadRange$, the neutral zone is active, and the P, I and D parts are no longer calculated, resulting in a constant control signal. The components are calculated again and the output signal of the controller changes again when the control deviation is larger again.

This function is intended to avoid an unnecessarily large number of actuating pulses.

2-point control behavior

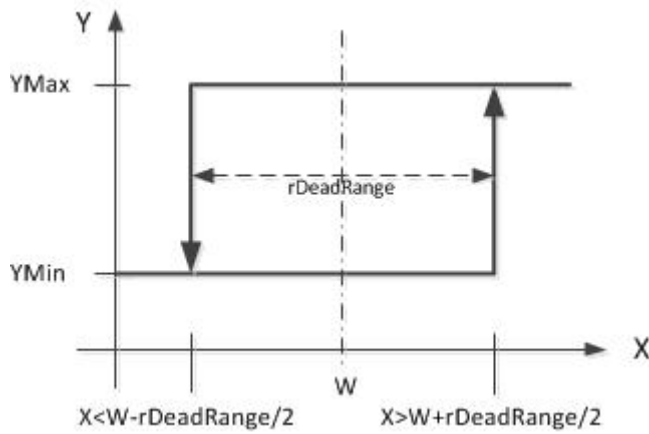
If the control parameters I_r , rKp , tTn , tTv and tTd are set to 0.0 or $\#0s$, then the controller has by definition a 2-point behavior.

The dead range $rDeadRange$ defines the hysteresis. Switching to the maximum value at the output rY fundamentally takes place when the control deviation rXW is larger than half of the hysteresis value $rDeadRange$, while switching to the minimum value always takes place when the control deviation rXW is smaller than the negative half of the hysteresis value $rDeadRange$:

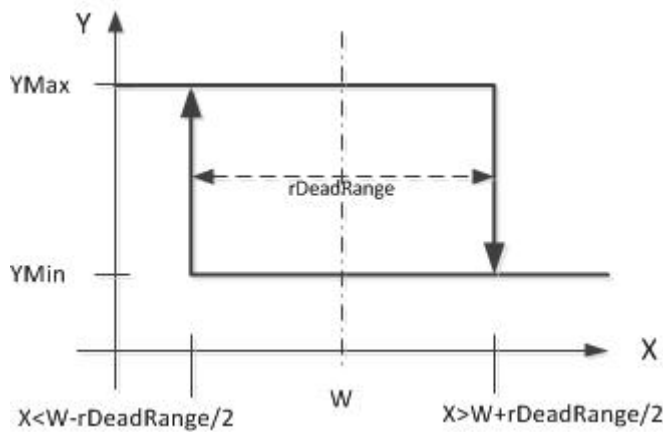


The different calculation methods for the control deviation for direct and indirect control direction result in the following switching behavior in relation to the actual value rX :

- direct control direction (cooling): control deviation = actual value - setpoint $rXW = rX - rW$



- indirect control direction (heating): control deviation = actual value-setpoint $rXW = rW - rX$
-



Documents about this

example_persistent_e.zip (Resources/zip/11659714827.zip)

3.4.6 FB_HVACPowerRangeTable

FB_HVACPowerRangeTable	
eDataSecurityType	bEnablePowerRangeTable
bEnable	iNumberOfStepInProfile
iNumberOfProfiles	iCurrentProfile
iProfile	iCurrentStep
iStep	stl_Ctrl
eCtrlModeProfile	stAggregate1
iManualProfile	stAggregate2
eCtrlModeStep	stAggregate3
iManualStep	stAggregate4
iNumberOfAggregates	stAggregate5
bReset	stAggregate6
arrPowerRangeTable >	bError
	eErrorCode
	iErrorPosArrayProfile
	iErrorPosArrayStep

Application

Application

This function block represents a power range table and serves the sequential control of power generators such as boilers or refrigerating machines. The power steps are determined by the upstream controller [FB HVAC Ctrl Step \[► 248\]](#) and transferred to the power range table via the input *iStep*.

All information or parameters relevant to system control are summarized in the power range table `ARRAY arrPowerRangeTable`. The power range table is a two-dimensional array with the structure [ST HVACPowerRange \[► 527\]](#). The power range table can be compared to a normal table. An individual element is clearly designated by naming line and column. The columns are marked by the field range 1..g [iMaxNumberOfProfiles \[► 531\]](#). This range is called the profile and is addressed via the input variable *iProfile*. The field range 0..g [iMaxNumberOfSteps \[► 531\]](#) represents the lines. This range is called the step and is addressed via the input variable *iStep*.

The power range table consists of 16 profiles. Each profile can have up to 33 steps, from 0 to 32. Each individual step contains the parameter structure [ST HVACPowerRange \[► 527\]](#) needed for system control. The parameters indicated here are transferred via the output structures *stl_Ctrl* and *stAggregate1-6* to the function block [FB HVAC CtrlStep \[► 248\]](#) for controlling the steps and to the function blocks for power generation.

How many steps are in a profile is specified via the output variable *iNumberOfStepInProfile*. The number depends on the entries in the power range table and on the specification of the selected profile *iCurrentProfile*. Internally, the selected profile is checked from step 1 to the step in which all variables of the structure [ST HVACPowerRange \[► 527\]](#) have the value 0. *iNumberOfStepInProfile* is always determined for the given profile only. *iNumberOfStepInProfile* can thus be used as a limitation of the steps in a profile for other function blocks such as for the input *iNumberOfStepInProfile* of the [FB HVAC CtrlStep \[► 248\]](#), which could control the individual steps of the `FB_HVACPowerRangeTable` via its output *iStep*. The step 0 is not taken into account when evaluating the number of steps via *iNumberOfStepInProfile*, because the power generators are switched off in this step; it can be considered to be a standby step. The values of the variables of the output structures *stAggregate1-6* then output the value 0. In the standby step, the parameters required for starting the power step sequence of the energy generators are transferred to the upstream controller [FB HVAC Ctrl Step \[► 248\]](#) via the output structure *stl_Ctrl*.

Table example for the power range table `arrPowerRangeTable`

Within a profile (table columns), an aggregate is assigned to each power step.

This determines the order in which the power generators are switched on or off within a sequence.

The power of the aggregates is specified from 0 to 6 for stepped power generators or from 0 to 100% for constant power generators.

After switching to a higher or lower power step, the integral component of the upstream function block [FB HVAC Ctrl Step \[► 248\]](#) is reinitialized. The current values for the step-up and step-down integrals are then transferred to the controller by means of the structure *stl_Ctrl*.

It is possible to switch to a different profile for a change of order of the aggregates within the sequence. By skilfully entering different orders of the aggregates in the profiles, it is possible to react with the utmost flexibility to all requirements with regard to the control of power generators.

The switching of the power profiles (*iCurrentProfile*) usually takes place on the basis of time or operating hours so that all power generators in a plant are utilized evenly.

For power generators with different nominal power, the power profile is changed depending on the load.



If an aggregate within a generator group is faulty, it is possible to switch to a profile in which the faulty generator is placed at the end of the power order.

	ST_HVACPowerRange	iCurrentProfile = 1	iCurrentProfile = 2	iCurrentProfile = 3
iCurrentStep = 0	iAggregate	0	0	0
	iAggregateStep	0	0	0
	rY_Max	0	0	0
	rY_Min	0	0	0
	bBlock	FALSE	FALSE	FALSE
	rIntegralHigh	2	3	4
	rIntegralLow	2	3	4
	udiSecDelayHigh	300	305	295
udiSecDelayLow	30	35	25	
iCurrentStep = 1	iAggregate	1	3	5
	iAggregateStep	1	1	1
	rY_Max			
	rY_Min			
	bBlock	FALSE	FALSE	FALSE
	rIntegralHigh	15	16	17
	rIntegralLow	5	6	7
	udiSecDelayHigh	300	305	295
udiSecDelayLow	30	35	25	
iCurrentStep = 2	iAggregate	2	3	4
	iAggregateStep	1	2	1
	rY_Max		60	
	rY_Min		30	
	bBlock	TRUE	FALSE	TRUE
	rIntegralHigh	10	11	7
	rIntegralLow	10	10	7
	udiSecDelayHigh	300	88	400
udiSecDelayLow	300	150	400	
iCurrentStep = 3	iAggregate	0	1	3
	iAggregateStep	0	1	1
	rY_Max	0	0	
	rY_Min	0	0	
	bBlock	FALSE	FALSE	FALSE
	rIntegralHigh	0	8	7
	rIntegralLow	0	8	7
	udiSecDelayHigh	0	23	400
udiSecDelayLow	0	123	200	
iCurrentStep = 4	iAggregate	2	0	0
	iAggregateStep	2	0	0
	rY_Max	70	0	0
	rY_Min	40	0	0
	bBlock	FALSE	FALSE	FALSE
	rIntegralHigh	10	0	0
	rIntegralLow	10	0	0
	udiSecDelayHigh	320	0	0
udiSecDelayLow	233	0	0	

The power range table shows three power profiles. The order of the power generators is entered in each power profile.

Profile 1:

in profile 1, aggregate 1 is switched to the first power step in power step 1. In the second power step, aggregate 2 is added with its first power step.

Profile 2:

in profile 2, aggregate 3 is switched to the first power step in power step 1. In the second power step, aggregate 3 is switched in its second power step. In the third power step, aggregate 1 is added with its first power step.

Profile 3:

in profile 3, aggregate 5 is switched to the first power step in power step 1. In the second power step, aggregate 4 is added with its first power step. In the third power step, aggregate 3 is added with its first power step.

By skillful arrangement of the power profiles, it is possible to react to faults in individual aggregates, the operating-hour dependent change of order or the load-optimized change of power orders.

The relationship between the power range table *arrPowerRange* and the output structures *stAggregate1-6* and *stl_Ctrl* is to be described in greater detail on the basis of the table example:

1. The column **iCurrentProfile= 1** is intended to represent the first profile of the power range table *arrPowerRange*. In this profile the output *iNumberOfStepsInProfile* has the value 2, because in the line **iCurrentStep= 3** no variable value of the structure ST_HVACPowerRange ▶ 527 (marked red) is greater than 0.

If we assume that the current power step *iCurrentStep* has the value 2, the variable values are output at the output structures as follows:

```
arrPowerRangeTable[] iCurrentProfile,iCurrentStep;
```

```
stAggregate1.iAggregateStep :=arrPowerRangeTable[,].iAggregateStep11;  
stAggregate1.rY_Max := arrPowerRangeTable[,].rY_Max11;  
stAggregate1.rY_Min :=arrPowerRangeTable[,].rY_Min11;  
stAggregate1.bBlock := arrPowerRangeTable[,].bBlock11;
```

```
stAggregate2.iAggregateStep :=arrPowerRangeTable[,].iAggregateStep12;  
stAggregate2.rY_Max := arrPowerRangeTable[,].rY_Max12;  
stAggregate2.rY_Min :=arrPowerRangeTable[,].rY_Min12;  
stAggregate2.bBlock := arrPowerRangeTable[,].bBlock12;
```

```
stl_Ctrl.rIntegralHigh := arrPowerRangeTable[,].rIntegralHigh12;  
stl_Ctrl.rIntegralLow := arrPowerRangeTable[,].rIntegralLow12;  
stl_Ctrl.udiSecDelayHigh :=arrPowerRangeTable[,].udiSecDelayHigh12;  
stl_Ctrl.udiSecDelayLow :=arrPowerRangeTable[,].udiSecDelayLow12;
```

Contents of the output structures *stAggregate1*, *stAggregate2* and *stl_Ctrl*. The variables of the output structures *stAggregate3-6* have the value 0.

```
stAggregate1.iAggregateStep :=1;  
stAggregate1.rY_Max := 0;  
stAggregate1.rY_Min :=0;  
stAggregate1.bBlock := FALSE;
```

```
stAggregate2.iAggregateStep :=1;  
stAggregate2.rY_Max := 0;  
stAggregate2.rY_Min :=0;  
stAggregate2.bBlock := TRUE;
```

```
stl_Ctrl.rIntegralHigh := 10;  
stl_Ctrl.rIntegralLow := 10;  
stl_Ctrl.udiSecDelayHigh :=300;  
stl_Ctrl.udiSecDelayLow :=300;
```

2. The column **iCurrentProfile= 2** is intended to represent the second profile of the power range table *arrPowerRange*. In this profile the output *iNumberOfStepsInProfile* has the value 3, because in the line **iCurrentStep= 4** no variable value of the structure ST_HVACPowerRange ▶ 527 (marked red) is greater than 0.

If we assume that the current power step *iCurrentStep* has the value 3, the variable values are output at the output structures as follows:

```
arrPowerRangeTable[] iCurrentProfile,iCurrentStep;
```

```
stAggregate1.iAggregateStep :=arrPowerRangeTable[,].iAggregateStep23;  
stAggregate1.rY_Max := arrPowerRangeTable[,].rY_Max23;
```

```
stAggregate1.rY_Min := arrPowerRangeTable[.].rY_Min23;
stAggregate1.bBlock := arrPowerRangeTable[.].bBlock23;
```

```
stAggregate3.iAggregateStep := arrPowerRangeTable[.iAggregateStep22];
stAggregate3.rY_Max := arrPowerRangeTable[.].rY_Max22;
stAggregate3.rY_Min := arrPowerRangeTable[.].rY_Min22;
stAggregate3.bBlock := arrPowerRangeTable[.].bBlock22;
```

```
stl_Ctrl.rIntegralHigh := arrPowerRangeTable[.].rIntegralHigh23;
stl_Ctrl.rIntegralLow := arrPowerRangeTable[.].rIntegralLow23;
stl_Ctrl.udiSecDelayHigh := arrPowerRangeTable[.].udiSecDelayHigh23;
stl_Ctrl.udiSecDelayLow := arrPowerRangeTable[.].udiSecDelayLow23;
```

Contents of the output structures *stAggregate1*, *stAggregate3* and *stl_Ctrl*. The variables of the output structures *stAggregate2* and *stAggregate4-6* have the value 0.

```
stAggregate1.iAggregateStep := 1;
stAggregate1.rY_Max := 0;
stAggregate1.rY_Min := 0;
stAggregate1.bBlock := FALSE;
```

```
stAggregate3.iAggregateStep := 2;
stAggregate3.rY_Max := 60;
stAggregate3.rY_Min := 30;
stAggregate3.bBlock := FALSE;
```

```
stl_Ctrl.rIntegralHigh := 8;
stl_Ctrl.rIntegralLow := 8;
stl_Ctrl.udiSecDelayHigh := 23;
stl_Ctrl.udiSecDelayLow := 123;
```

- The column **iCurrentProfile= 3** is intended to represent the third profile of the power range table *arrPowerRange*. In this profile the output *iNumberOfStepsInProfile* has the value 3, because in the line **iCurrentStep= 4** no variable value of the structure *ST_HVACPowerRange* [▶ 527](#) (marked red) is greater than 0.

If we assume that the current power step *iCurrentStep* has the value 3, the variable values are output at the output structures as follows:

```
arrPowerRangeTable[] iCurrentProfile, iCurrentStep;
```

```
stAggregate3.iAggregateStep := arrPowerRangeTable[.].iAggregateStep33;
stAggregate3.rY_Max := arrPowerRangeTable[.].rY_Max33;
stAggregate3.rY_Min := arrPowerRangeTable[.].rY_Min33;
stAggregate3.bBlock := arrPowerRangeTable[.].bBlock33;
```

```
stAggregate4.iAggregateStep := arrPowerRangeTable[.iAggregateStep32];
stAggregate4.rY_Max := arrPowerRangeTable[.].rY_Max32;
stAggregate4.rY_Min := arrPowerRangeTable[.].rY_Min32;
stAggregate4.bBlock := arrPowerRangeTable[.].bBlock32;
```

```
stAggregate5.iAggregateStep := arrPowerRangeTable[.iAggregateStep31];
stAggregate5.rY_Max := arrPowerRangeTable[.].rY_Max31;
stAggregate5.rY_Min := arrPowerRangeTable[.].rY_Min31;
stAggregate5.bBlock := arrPowerRangeTable[.].bBlock31;
```

```
stl_Ctrl.rIntegralHigh := arrPowerRangeTable[.].rIntegralHigh33;
stl_Ctrl.rIntegralLow := arrPowerRangeTable[.].rIntegralLow33;
stl_Ctrl.udiSecDelayHigh := arrPowerRangeTable[.].udiSecDelayHigh33;
stl_Ctrl.udiSecDelayLow := arrPowerRangeTable[.].udiSecDelayLow33;
```

Contents of the output structures *stAggregate3*, *stAggregate4*, *stAggregate5* and *stl_Ctrl*. The variables of the output structures *stAggregate1*, *stAggregate2* and *stAggregate6* have the value 0.

```
stAggregate3.iAggregateStep := 1;
stAggregate3.rY_Max := 0;
stAggregate3.rY_Min := 0;
stAggregate3.bBlock := FALSE;
```

```

stAggregate4.iAggregateStep :=1;
stAggregate4.rY_Max := 0;
stAggregate4.rY_Min :=0;
stAggregate4.bBlock := TRUE;

stAggregate5.iAggregateStep :=1;
stAggregate5.rY_Max := 0;
stAggregate5.rY_Min :=0;
stAggregate5.bBlock := FALSE;

stl_Ctrl.rIntegralHigh := 7;
stl_Ctrl.rIntegralLow := 7;
stl_Ctrl.udlSecDelayHigh :=400;
stl_Ctrl.udlSecDelayLow :=200;

```

Output structure stl_Ctrl adjusted to the example

		iCurrentProfile = 1	iCurrentProfile = 2	iCurrentProfile = 3
		stl_Ctrl	stl_Ctrl	stl_Ctrl
iCurrentStep = 0	rIntegralHigh	2	3	4
	rIntegralLow	2	3	4
	udiSecDelayHigh	300	305	295
	udiSecDelayLow	30	35	25
iCurrentStep = 1	rIntegralHigh	15	16	17
	rIntegralLow	5	6	7
	udiSecDelayHigh	300	305	295
	udiSecDelayLow	30	35	25
iCurrentStep = 2	rIntegralHigh	10	11	7
	rIntegralLow	10	10	7
	udiSecDelayHigh	300	88	400
	udiSecDelayLow	300	150	400
iCurrentStep = 3	rIntegralHigh	0	8	7
	rIntegralLow	0	8	7
	udiSecDelayHigh	0	23	400
	udiSecDelayLow	0	123	200
iCurrentStep = 4	rIntegralHigh	0	0	0
	rIntegralLow	0	0	0
	udiSecDelayHigh	0	0	0
	udiSecDelayLow	0	0	0

Output structures stAggregate1-6 adjusted to the example

		iCurrentProfile = 1						iCurrentProfile = 2						iCurrentProfile = 3					
		stAggregate						stAggregate						stAggregate					
		1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6
iCurrentStep = 0	iAggregateStep	0	0					0	0					0	0	0			
	rY_Max	0	0					0	0					0	0	0			
	rY_Min	0	0					0	0					0	0	0			
	bBlock	FALSE	FALSE					FALSE	FALSE					FALSE	FALSE	FALSE			
iCurrentStep = 1	iAggregateStep	1	0					0	1					0	0	1			
	rY_Max	0	0					0	0					0	0	0			
	rY_Min	0	0					0	0					0	0	0			
	bBlock	FALSE	FALSE					FALSE	FALSE					FALSE	FALSE	FALSE			
iCurrentStep = 2	iAggregateStep	1	1					0	2					0	1	1			
	rY_Max	0	0					0	60					0	0	0			
	rY_Min	0	0					0	30					0	0	0			
	bBlock	FALSE	TRUE					FALSE	FALSE					FALSE	TRUE	FALSE			
iCurrentStep = 3	iAggregateStep	0	0					1	2					1	1	1			
	rY_Max	0	0					0	60					0	0	0			
	rY_Min	0	0					0	30					0	0	0			
	bBlock	FALSE	FALSE					FALSE	FALSE					FALSE	TRUE	FALSE			
iCurrentStep = 4	iAggregateStep	0	0					0	0					0	0	0			
	rY_Max	0	0					0	0					0	0	0			
	rY_Min	0	0					0	0					0	0	0			
	bBlock	FALSE	FALSE					FALSE	FALSE					FALSE	FALSE	FALSE			

Application example

The application example shows the function block FB_HVACPowerRangeTable in conjunction with the I transfer element FB_HVACI_CtrlStep [▶ 248]. The example is illustrated in the programming languages ST and CFC. The program example **P_CFC_I_CtrlStep.PRG** for the CFC programming languages can be found in the folder **Language CFC > Controller**, the program example **P_ST_I_CtrlStep.PRG** for the ST programming languages in the folder **Language Structured Text > Controller**.


Download	Required library
TcHVAC.pro [▶ 531]	TcHVAC.lib

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
iNumberOfProfiles : INT;
iProfile          : INT;
iStep             : INT;
eCtrlModeProfile  : E_HVACCtrlMode;
iManualProfile    : INT;
eCtrlModeStep     : E_HVACCtrlMode;
iManualStep       : INT;
iNumberOfAggregates : INT;
bReset            : BOOL;
```

eDataSecurityType: if eDataSecurityType:= *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If eDataSecurityType:= *eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bEnable: the function block is enabled via TRUE. If `bEnable = FALSE`, then all output variables and output structures are set constantly to 0. The checking of the variables `iNumberOfProfiles`, `iProfile`, `iStep`, `iNumberOfStepInProfile`, `iNumberOfAggregates`, `arrPowerRangeTable[X,X].iAggregate`, `arrPowerRangeTable[X,X].iAggregateSteps`, `arrPowerRangeTable[iP,iS].udiSecDelayHigh`, and `arrPowerRangeTable[iP,iS].udiSecDelayLow` remains active. If an error occurs, it is displayed with `bError = TRUE` and can be acknowledged with `bReset` once the fault has been corrected.

iNumberOfProfiles: number of parameterized profiles in the power range table `arrPowerRangeTable`. The indication of the number of profiles must not be lower than `g_iMinNumberOfProfiles` [► 531] and must not be greater than `g_iMaxNumberOfProfiles` [► 531]. Otherwise an error is indicated by `bError = TRUE` and the execution of the function block is stopped.

iProfile: indicates the current profile with which the function block from the power range table `arrPowerRangeTable[iCurrentProfile,iCurrentStep]` is working. If `eCtrlModeProfile = eHVACCtrlMode_Auto`, then `iCurrentProfile = iProfile`. The indication of the profile must not be lower than `g_iMinNumberOfProfiles` [► 531] and must not be greater than `iNumberOfProfiles`. Otherwise an error is indicated by `bError = TRUE` and the execution of the function block is stopped.

iStep: indicates the current step with which the function block from the power range table `arrPowerRangeTable[iCurrentProfile,iCurrentStep]` is working. If `eCtrlModeStep = eHVACCtrlMode_Auto`, then `iCurrentStep = iStep`. The indication of the steps must not be lower than `g_iMinNumberOfStep` [► 531] and must not be greater than `iNumberOfStepInProfile`. Otherwise an error is indicated by `bError = TRUE` and the execution of the function block is stopped.

eCtrlModeProfile: the input variable via which the profile from the power range table `arrPowerRangeTable[iCurrentProfile,iCurrentStep]` is specified is decided via this enum. If `eCtrlModeProfile = eHVACCtrlMode_Auto`, then `iCurrentProfile = iProfile`. If `eCtrlModeProfile = eHVACCtrlMode_Manual`, then `iCurrentProfile = iManualProfile`.

iManualProfile: indicates which profile from the power range table `arrPowerRangeTable[iCurrentProfile,iCurrentStep]` is in use. If `eCtrlModeProfile = eHVACCtrlMode_Manual`, then `iCurrentProfile = iManualProfile`. The indication of the profile must not be lower than `g_iMinNumberOfProfiles` [► 531] and must not be greater than `iNumberOfProfiles`. Otherwise `iManualProfile` is set internally to `g_iMinNumberOfProfiles` [► 531] if it is lower than `g_iMinNumberOfProfiles` [► 531] or to `iNumberOfProfiles` if it is greater than `iNumberOfProfiles`.

eCtrlModeStep: the input variable via which the steps from the power range table `arrPowerRangeTable[iCurrentProfile,iCurrentStep]` are specified is decided via this enum. If `eCtrlModeStep = eHVACCtrlMode_Auto`, then `iCurrentStep = iStep`. If `eCtrlModeStep = eHVACCtrlMode_Manual`, then `iCurrentStep = iManualStep`.

iManualStep: indicates which step from the power range table `arrPowerRangeTable[iCurrentProfile,iCurrentStep]` is in use. If `eCtrlModeStep = eHVACCtrlMode_Manual`, then `iCurrentStep = iManualStep`. The indication of the steps must not be lower than `g_iMinNumberOfStep` [► 531] and must not be greater than `iNumberOfStepInProfile`. Otherwise `iManualStep` is set internally to `g_iMinNumberOfSteps` [► 531] if it is lower than `g_iMinNumberOfProfiles` [► 531] or to `iNumberOfStepInProfile` if it is greater than `iNumberOfStepInProfile`.

iNumberOfAggregates: `iNumberOfAggregates` specifies the number of aggregates in the power generator sequence. For example, if `iNumberOfAggregates = 4`, then the parameters from the array `arrPowerRangeTable` are written to the output structures `stAggregate1-4`, depending on the specification of `iCurrentStep/iCurrentProfile`. The indication must not be lower than `g_iMinNumberOfAggregates` [► 531] and must not be greater than `g_iMaxNumberOfAggregates` [► 531]. Otherwise an error is indicated by `bError = TRUE` and the execution of the function block is stopped.

bReset: input for acknowledgement of the faults once they have been corrected. Internally the system responds to a rising edge.

VAR_OUTPUT

```

bEnablePowerRangeTable: BOOL;
iNumberOfStepInProfile: INT;
iCurrentProfile       : INT;
iCurrentStep         : INT;
stI_Ctrl              : ST_HVACI_Ctrl;
stAggregate1         : ST_HVACAggregate;
stAggregate2         : ST_HVACAggregate;
stAggregate3         : ST_HVACAggregate;
stAggregate4         : ST_HVACAggregate;
stAggregate5         : ST_HVACAggregate;
stAggregate6         : ST_HVACAggregate;
bError               : BOOL;
eErrorCode            : E_HVACErrorCodes;
iErrorPosArrayProfile : INT;
iErrorPosArrayStep   : INT;

```

bEnablePowerRangeTable: indicates that the function block is enabled. *bEnablePowerRangeTable* is TRUE if *bEnable* = TRUE AND *bError* = FALSE.

iNumberOfStepInProfile: indicates the number of parameterized steps from the specified profile *iCurrentProfile* of the power range table *arrPowerRangeTable*. Each individual step contains the parameter structure *ST_HVACPowerRange* [► 527] needed for system control. Each profile can have up to 33 steps. The field for the steps in the power range table *arrPowerRangeTable* begins with 0 and ends with 32. The number of parameterized steps in a profile *iNumberOfStepInProfile* is determined from step 1 upwards to step 32. The counting of the parameterized steps ends in the step in which all variables of the parameter structure *ST_HVACPowerRange* [► 527] have the value 0. The minimum value that *iNumberOfStepInProfile* can have is 1 and the maximum is 32. Step 0 is not taken into account, because the power generators are switched off in this step. The step can be considered to be a standby step. The values of the variables of the output structures *stAggregate1-6* then output the value 0. In the standby step, the parameters required for starting the power step sequence of the energy generators are transferred to the upstream controller *FB_HVACI_Ctrl_Step* [► 248] via the output structure *stI_Ctrl*.



iNumberOfStepInProfile can be used to limit the steps for other function blocks, such as for the input *iNumberOfStepInProfile* of the function block *FB_HVACI_CtrlStep*. In turn, the output variable *iStep* of *FB_HVACI_CtrlStep* can be used to specify the step of the power range table *FB_HVACPowerRangeTable* via the input *iStep*.

iCurrentProfile: indicates the current profile with which the function block from the power range table *arrPowerRangeTable* [*iCurrentProfile*, *iCurrentStep*] is working. The indication depends on the input variables *iProfile*, *iManualProfile* and *eCtrlModeProfile*. If *eCtrlModeProfile* = *eHVACCtrlMode_Auto*, then *iCurrentProfile* = *iProfile*. If *eCtrlModeProfile* = *eHVACCtrlMode_Manual*, then *iCurrentProfile* = *iManualProfile*.

iCurrentStep: indicates the current power step with which the function block from the power range table *arrPowerRangeTable* [*iCurrentProfile*, *iCurrentStep*] is working. The most current step is always the highest step viewed from 1 upward. The indication depends on the input variables *iStep*, *iManualStep* and *eCtrlModeStep*. If *eCtrlModeStep* = *eHVACCtrlMode_Auto*, then *iCurrentStep* = *iStep*. If *eCtrlModeStep* = *eHVACCtrlMode_Manual*, then *iCurrentStep* = *iManualStep*.

stI_Ctrl: output structure of the parameters *ST_HVACI_Ctrl* [► 526] for the function block *FB_HVACI_CtrlStep* [► 248].

Which values the variables of the output structure *stI_Ctrl* take from the power range table *arrPowerRangeTable* depends on the selected profile and the current step. If the selected profile = **2***iCurrentProfile* and the step = **3***iCurrentStep*, then the contents of the following variables from the power range table are written to the structure *stI_Ctrl*:

```

stI_Ctrl.rIntegralHigh := arrPowerRangeTable[.].rIntegralHighiCurrentProfileiCurrentStep;
stI_Ctrl.rIntegralLow  := arrPowerRangeTable[.].rIntegralLowiCurrentProfileiCurrentStep;
stI_Ctrl.udiSecDelayHigh := arrPowerRangeTable[.].udiSecDelayHighiCurrentProfileiCurrentStep;
stI_Ctrl.udiSecDelayLow := arrPowerRangeTable[.].udiSecDelayLowiCurrentProfileiCurrentStep;

```

```

stI_Ctrl.rIntegralHigh := arrPowerRangeTable[.].rIntegralHigh23;
stI_Ctrl.rIntegralLow  := arrPowerRangeTable[.].rIntegralLow23;
stI_Ctrl.udiSecDelayHigh := arrPowerRangeTable[.].udiSecDelayHigh23;
stI_Ctrl.udiSecDelayLow := arrPowerRangeTable[.].udiSecDelayLow23;

```

stl_Ctrl.rIntegralHigh: positive value for the upper limit at which the integration of the I transfer element is stopped.

stl_Ctrl.rIntegralLow: positive value for the lower limit at which the integration of the I transfer element is stopped.

stl_Ctrl.udiSecDelayHigh: delay time after whose expiry the I transfer element is activated.

stl_Ctrl.udiSecDelayLow: delay time after whose expiry the I transfer element is activated.

stAggregate1-6: output structures of the parameters *ST_HVACAggregate* [► 524] for controlling the aggregates 1 to 6. *iNumberOfAggregates* determines the number of aggregates in the power generator sequence. The values adopted by the variables of the output structures *stAggregate1-6* from the power range table *arrPowerRangeTable* depend on the selected profile and the current step. If the selected profile = **8iCurrentProfile** and the step = **5iCurrentStep**, then it is crucial that the variable *arrPowerRangeTable[,].iAggregate85* has the value 1. The values parameterized in the power range table are then output via the output structure *stAggregate1*.

```
stAggregate1.rY_Max := arrPowerRangeTable[,].rY_MaxiCurrentProfileiCurrentStep;
stAggregate1.rY_Min := arrPowerRangeTable[,].rY_MiniCurrentProfileiCurrentStep;
stAggregate1.iAggregateStep := arrPowerRangeTable[,].iAggregateStepiCurrentProfileiCurrentStep;
stAggregate1.bBlock := arrPowerRangeTable[,].bBlockiCurrentProfileiCurrentStep;
```

```
stAggregate1.rY_Max := arrPowerRangeTable[,].rY_Max85;
stAggregate1.rY_Min := arrPowerRangeTable[,].rY_Min85;
stAggregate1.iAggregateStep := arrPowerRangeTable[,].iAggregateStep83;
stAggregate1.bBlock := arrPowerRangeTable[,].bBlock85;
```

If the value of the variable *arrPowerRangeTable[,].iAggregate83* is also 1, then the values of the highest power step *iCurrentStep* = 5 are written to the corresponding output structure *stAggregate1*, because *arrPowerRangeTable[,].iAggregate85* is also 1 here. Only if *iCurrentStep* = 3 does this mean that following values are output via the output structure *stAggregate1*:

```
stAggregate1.rY_Max := arrPowerRangeTable[,].rY_Max83;
stAggregate1.rY_Min := arrPowerRangeTable[,].rY_Min83;
stAggregate1.iAggregateStep := arrPowerRangeTable[,].iAggregateStep83;
stAggregate1.bBlock := arrPowerRangeTable[,].bBlock83;
```

If *iCurrentStep* = 4 and *arrPowerRangeTable[,].iAggregate84* = 2, then this means that the following values are output via the output structures *stAggregate1* and *stAggregate2*:

```
stAggregate2.rY_Max := arrPowerRangeTable[,].rY_Max84;
stAggregate2.rY_Min := arrPowerRangeTable[,].rY_Min84;
stAggregate2.iAggregateStep := arrPowerRangeTable[,].iAggregateStep84;
stAggregate2.bBlock := arrPowerRangeTable[,].bBlock84;
```

```
stAggregate1.rY_Max := arrPowerRangeTable[,].rY_Max83;
stAggregate1.rY_Min := arrPowerRangeTable[,].rY_Min83;
stAggregate1.iAggregateStep := arrPowerRangeTable[,].iAggregateStep83;
stAggregate1.bBlock := arrPowerRangeTable[,].bBlock83;
```

bError: this output indicates with a TRUE that there is an error. The execution of the function block is stopped. The enum *eErrorCode* indicates the error number. After the error has been corrected, the message *bError* must be acknowledged with *bReset*.

eErrorCode: returns the error number [► 520] when the *bError* output is set.

The following errors can occur in this function block:

eHVACErrorCodes_InvalidParam_iStep: error when checking the specified steps. The value of *iStep* must be greater than or equal to *g_iMinNumberOfSteps* [► 531] or smaller than or equal to *iNumberOfStepsInProfile*.

eHVACErrorCodes_InvalidParam_iNumberOfStepInProfile: error while checking the number of profiles whether each of the specified profiles has parameterized steps. The profiles are checked in step 0. If no variable value of the structure *ST_HVACPowerRange* [► 527] is greater than 0, an error has occurred. The variable *iErrorPosArrayProfile* is used to specify the incorrect profile in the power range table *arrPowerRangeTable[x1,x2]*: $x1 = iErrorPosArrayProfile$;

eHVACErrorCodes_InvalidParam_iNumberOfAggregates: error when checking the specified number of aggregates. The value of *iNumberOfAggregates* must be smaller than *g_iMinNumberOfAggregates* [► 531] and larger than *g_iMaxNumberOfAggregates* [► 531].

eHVACErrorCodes_InvalidParam_iAggregateSteps: error when checking the specified steps for the aggregate. The value of *arrPowerRangeTable[x1,x2].iAggregateStep* must be greater than or equal to *g_iAggregateMinNumberOfSteps* [► 531] and smaller than or equal to *g_iAggregateMaxNumberOfSteps* [► 531].

The variables *iErrorPosArrayProfile* and *iErrorPosArrayStep* are used to specify the incorrect point in the power range table *arrPowerRangeTable[x1,x2]*: $x1 = x2 = iErrorPosArrayProfile; iErrorPosArrayStep$

eHVACErrorCodes_InvalidParam_iNumberOfProfiles: error when checking the number of profiles. The value of *iNumberOfProfiles* must be greater than *g_iMinNumberOfProfiles* [► 531] and smaller than or equal to *g_iMaxNumberOfProfiles* [► 531].

eHVACErrorCodes_InvalidParam_iProfile: error when checking the specified profiles. The value of *iProfile* must be greater than *g_iMinNumberOfProfiles* [► 531] and smaller than or equal to *iNumberOfProfiles*.

eHVACErrorCodes_InvalidParam_iAggregate: error when checking the specified aggregates. The value of *arrPowerRangeTable[x1,x2].iAggregate* must be greater than or equal to *g_iMinNumberOfAggregates* [► 531] and less than or equal to *g_iMaxNumberOfAggregates* [► 531].

The variables *iErrorPosArrayProfile* and *iErrorPosArrayStep* are used to specify the incorrect point in the power range table *arrPowerRangeTable[x1,x2]*: $x1 = x2 = iErrorPosArrayProfile; iErrorPosArrayStep$

eHVACErrorCodes_InvalidParam_udiSecDelayUp: error when checking *arrPowerRangeTable[x1,x2].udiSecDelayUp*. The value of *arrPowerRangeTable[x1,x2].udiSecDelayUp* must not be greater than *g_udiMaxSec* [► 531].

The variables *iErrorPosArrayProfile* and *iErrorPosArrayStep* are used to specify the incorrect point in the power range table *arrPowerRangeTable[x1,x2]*: $x1 = x2 = iErrorPosArrayProfile; iErrorPosArrayStep$

eHVACErrorCodes_InvalidParam_udiSecDelayDown: error when checking *arrPowerRangeTable[x1,x2].udiSecDelayDown*. The value of *arrPowerRangeTable[x1,x2].udiSecDelayDown* must not be greater than *g_udiMaxSec* [► 531].

The variables *iErrorPosArrayProfile* and *iErrorPosArrayStep* are used to specify the incorrect point in the power range table *arrPowerRangeTable[x1,x2]*: $x1 = x2 = iErrorPosArrayProfile; iErrorPosArrayStep$

i

To get the error numbers of the enum in the PLC, *eErrorCode* can be assigned to a variable of data type WORD.

eHVACErrorCodes_InvalidParam_iStep = 31
eHVACErrorCodes_InvalidParam_iNumberOfStepInProfile = 32
eHVACErrorCodes_InvalidParam_iNumberOfAggregates = 33
eHVACErrorCodes_InvalidParam_iAggregateSteps = 34
eHVACErrorCodes_InvalidParam_iNumberOfProfiles = 35
eHVACErrorCodes_InvalidParam_iProfile = 36
eHVACErrorCodes_InvalidParam_iAggregate = 39
eHVACErrorCodes_InvalidParam_udiSecDelayUp = 40
eHVACErrorCodes_InvalidParam_udiSecDelayDown = 41

iErrorPosArrayProfile: the variable *iErrorPosArrayProfile* specifies the profile in which the incorrect position in the power range table *arrPowerRangeTable[iErrorPosArrayProfile,iErrorPosArrayStep]* can be found. If *iErrorPosArrayProfile* > 0, then one of the following errors is present at *eErrorCode*:

- *eHVACErrorCodes_InvalidParam_iNumberOfStepInProfile*
- *eHVACErrorCodes_InvalidParam_iAggregateSteps*
- *eHVACErrorCodes_InvalidParam_iAggregate*
- *eHVACErrorCodes_InvalidParam_udiSecDelayUp*
- *eHVACErrorCodes_InvalidParam_udiSecDelayDown*

The exact position of the step can be determined with the aid of the variable *iErrorPosArrayStep*.

Sample 1 for locating an error:

The following values are present at the output variables:

```
bError =TRUE;
eErrorCode= eHVACErrorCodes_InvalidParam_iNumberOfStepInProfile;
iErrorPosArrayProfile = 2;
```

In the power range table *arrPowerRangeTable*[2,x] in the profile 2 in step 0 no variable value of the structure *ST_HVACPowerRange* [► 527] is greater than 0. *iErrorPosArrayStep* is not taken into account with this error.

Sample 2 for locating an error:

The following values are present at the output variables:

```
bError =TRUE;
eErrorCode= eHVACErrorCodes_InvalidParam_iAggregateSteps;
iErrorPosArrayProfile = 2;
iErrorPosArrayStep = 3;
```

There is an incorrect value on the following variable in the power range table *arrPowerRangeTable*:

```
arrPowerRangeTable[2,3].iAggregateStep
```

iErrorPosArrayStep: if *iErrorPosArrayProfile* > 0, then the variable *iErrorPosArrayStep* indicates the step in the profile in which the incorrect place in the power range table is to be found.

Sample for locating an error:

The following values are present at the output variables:

```
bError =TRUE;
eErrorCode= eHVACErrorCodes_InvalidParam_iAggregate;
iErrorPosArrayProfile = 5;
iErrorPosArrayStep = 15;
```

There is an incorrect value on the following variable in the power range table *arrPowerRangeTable*:

```
arrPowerRangeTable[5,15].iAggregate
```

VAR_IN_OUT

```
arrPowerRangeTable : ARRAY [1..g_iMaxNumberOfProfiles,0..g_iMaxNumberOfSteps] OF ST_HVACPowerRange;
                    X
```

arrPowerRangeTable: all information or parameters relevant to system control are summarized in the power range table *arrPowerRangeTable*. The power range table is a two-dimensional field (array) with the structure *ST_HVACPowerRange* [► 527]. The power range table can be compared to a normal table in that the horizontal entries are called lines and the vertical entries columns. An individual element is thus clearly designated by naming line and column. The field range 1..g_iMaxNumberOfProfiles [► 531] of the power range table would be the vertical part, i.e. the columns. This range is called the profile and *iCurrentProfile* indicates which profile is being addressed. The field range 0..g_iMaxNumberOfSteps [► 531] is regarded as the horizontal part, i.e. the lines. This range is called the step and *iCurrentStep* indicates which step is being addressed.

The power range table consists of 16 profiles. Each profile can have up to 33 steps. Each individual step contains the parameter structure *ST_HVACPowerRange* [► 527] needed for system control. The parameters indicated here are transferred via the output structures *stI_Ctrl* to the function block *FB_HVACI_CtrlStep* [► 248] for controlling the steps and via *stAggregate1-6* to the function blocks for power generation.

The number of steps in a profile is indicated by the output variable *iNumberOfStepInProfile*. The number depends on the entries in the power range table and on the selected profile *iCurrentProfile*. The selected profile is checked internally in the function block from step 1 to the step in which all variables of the structure *ST_HVACPowerRange* [► 527] have the value 0. *iNumberOfStepInProfile* is always determined for the specified profile only. Each profile can have up to 33 steps from 0 to 32; step 0 is not taken into account when evaluating the number of steps via *iNumberOfStepInProfile*.

Structure *ST_HVACPowerRange* [► 527]

arrPowerRangeTable[x,x].iAggregate: parameter that specifies to which output structure *stAggregate1-6* of the function block [FB_HVACPowerRangeTable](#) [► 275] the variables *rY_Min*, *rY_Max*, *iAggregateStep* and *bBlock* are written.

arrPowerRangeTable[x,x].iAggregateStep: parameter that specifies the step in which the addressed aggregate should be fixed or should regulate; see *bBlock*.
iAggregateStep is output via the structure *stAggregateX*.

arrPowerRangeTable[x,x].rY_Max: parameter specification for continuous aggregates. *rY_Max* is output via the structure *stAggregateX*.

arrPowerRangeTable[x,x].rY_Min: parameter specification for continuous aggregates. *rY_Min* is output via the structure *stAggregateX*.

arrPowerRangeTable[x,x].rIntegralHigh: positive value for the upper limit at which the integration of the I transfer element is stopped, see the VAR_IN_OUT variable *rIntegralHigh* in [FB_HVACI_CtrlStep](#) [► 248].
rIntegralHigh is output via the structure *stI_Ctrl*.

arrPowerRangeTable[x,x].rIntegralLow : positive value for the lower limit at which the integration of the I transfer element is stopped, see the VAR_IN_OUT variable *rIntegralHigh* in [FB_HVACI_CtrlStep](#) [► 248].
rIntegralLow is output via the structure *stI_Ctrl*.

arrPowerRangeTable[x,x].udiSecDelayHigh: delay time after which the I transfer element is activated, see the VAR_IN_OUT variable *udiSecDelayHigh* in [FB_HVACI_CtrlStep](#) [► 248].
udiSecDelayHigh is output via the structure *stI_Ctrl*.

arrPowerRangeTable[x,x].udiSecDelayLow: delay time after which the I transfer element is activated, see the VAR_IN_OUT variable *udiSecDelayHigh* in [FB_HVACI_CtrlStep](#) [► 248].
udiSecDelayLow is output via the structure *stI_Ctrl*.

arrPowerRangeTable[x,x].bBlock: if *bBlock* = FALSE, then the addressed aggregate is fixed in the specified step via *iAggregateStep*. If *bBlock* = TRUE, then the control of the addressed aggregate is released from the off step (0) to the specified step via *iAggregateStep*.
bBlock is output via the structure *stAggregateX*.

The variable is saved persistently.

Documents about this

 [example_persistent_e.zip](#) (Resources/zip/11659714827.zip)

3.4.7 Sequence-Controller

Introduction – sequence controller

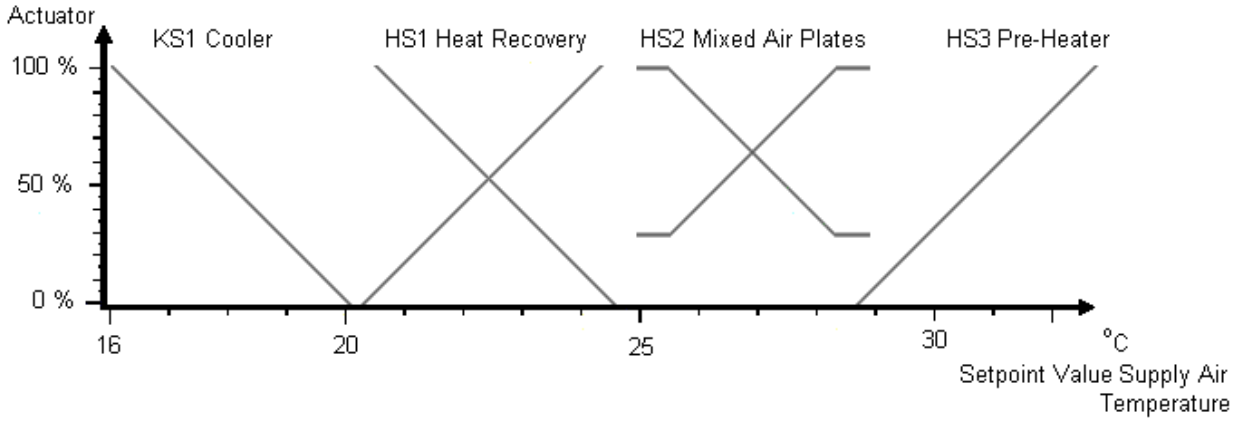
In heating, ventilation and air conditioning systems, it is often the case that several actuators, working in a so-called controller sequence, are used in order to achieve a control variable. In the air conditioning system shown below, four actuators are involved in the regulation of the supply air temperature. A dedicated sequence controller is instanced for each of these actuators in the HVAC library. During active control only one of these sequence controllers is active. The other, non-active controllers fix their control signal so that it is energetically optimal for the tempering of the supply air temperature. Depending on the control direction of the individual controller, this means either the maximum or the minimum for the control value *rY*.

If the effect of the active actuator (controller) is not sufficient when reaching a end position, the active controller switches to the adjacent controller to the left or right in the controller sequence. This then takes over control. The previously active controller remains in the end position of *rYmax* or *rYmin* depending on the control direction.

The same procedure is followed with the other actuators until the setpoint or the right or left end of the sequence is reached.

In the sequence of the illustrated air conditioning system, all actuators that influence the control variable are shown from left to right. At the far left is the actuator that enables the greatest possible reduction of the control variable; at the far right is the actuator that effects the greatest possible increase of the control variable.

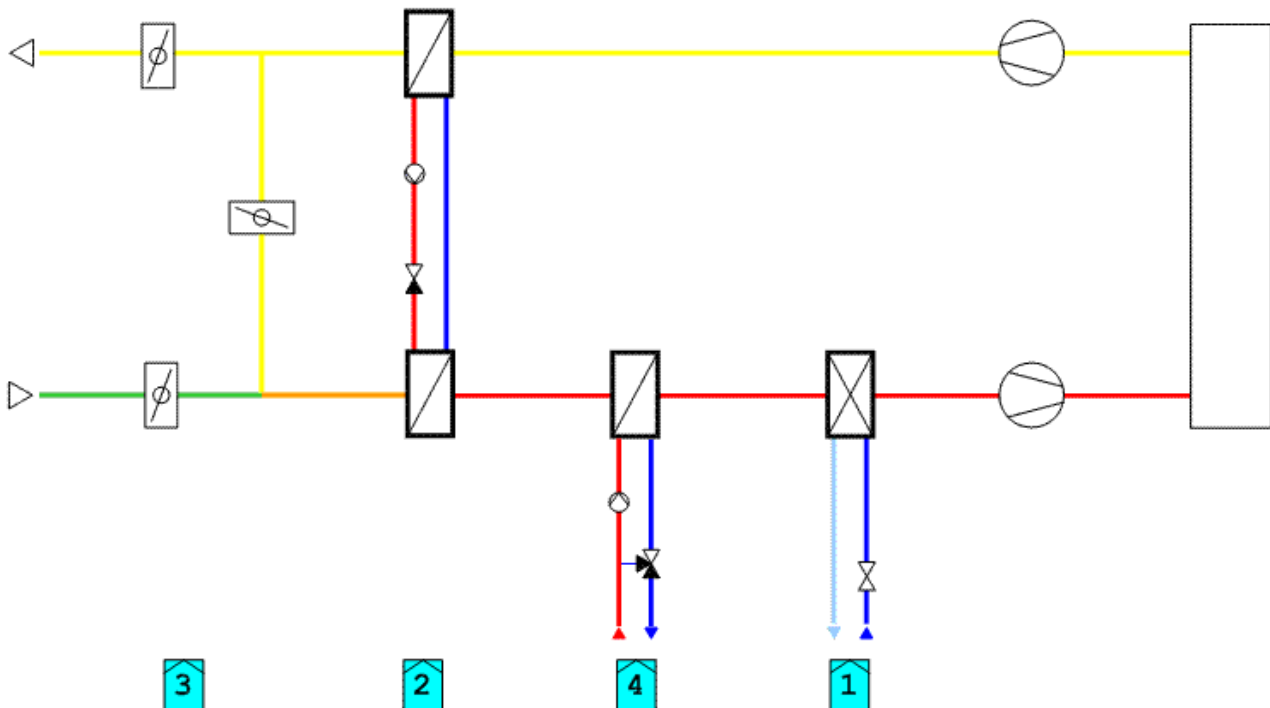
Actuators that should drive in parallel or in opposite directions, e.g. outside air and circulating air dampers (HS2) are implemented once only. The individual actuators can have a positive control direction (more actuation results in a larger control variable, e.g. a heater) or a negative control direction (more actuation results in a smaller control variable, e.g. a cooler). Some actuators, e.g. a circulating air damper (HS1 & HS2), change their control direction during operation.



Example: Supply air temperature regulation in an air conditioning system with an air cooler, a heat recovery unit, a mixed air chamber and an air heater, as shown in the figure above.

iMyNumberInSequence

1. KS1 cooler 1
2. HS1 heat recovery 2
3. HS2 mixed air dampers 3
4. HS3 preheater 4



Each sequence controller gets the number of the total controllers in the sequence. In this example 4. The PLC variable is called *iNumberOfSequences* in the function blocks. In addition each function block is given its own number. In the sequence *iMyNumberInSequence*. The controllers are

numbered from left to right in ascending order. Hence each controller knows which position it occupies within the sequence. The outer sequence controllers know that switching to a further sequence controller should not take place in the event of a residual control deviation.

In the example, the controller [FB_HVACPIDEnergyRecovery \[▶ 309\]](#) would start controlling with the heat recovery unit in the event of the control variable being too small. All other actuators are closed. If the maximum or minimum position of the heat recovery unit has been reached, the controller switches to the damper controller [FB_HVACPIDMixedAir \[▶ 317\]](#). This increases the parameter *eHVACSequenceCtrlMode* to 3. If the control effect has still not been achieved and the mixed air damper has reached its limit (e.g. minimum outside air rate), the mixed air controller raises the value of *eHVACSequenceCtrlMode* to 4. As a result the preheater or rather the sequence controller [FB_HVACPIDPreHeating \[▶ 320\]](#) is activated. If the control variable is too high, the sequence is reversed until the cooler is activated by means of [FB_HVACPIDCooling \[▶ 303\]](#) being enabled.

If the control value is at the lower or upper limit of a controller, the actual value of the controlled system oscillates around the setpoint with a small amplitude, frequent switching back and forth between two sequence controllers can be damped by an additional parameter for the switchover. To this end, the difference between the actual values and the setpoints of the controlled system is integrated after the sequence controller has reached its lower or upper limit. Switching to the next sequence only takes place if the sum of this integration is greater than the value of *rDeadRange*.

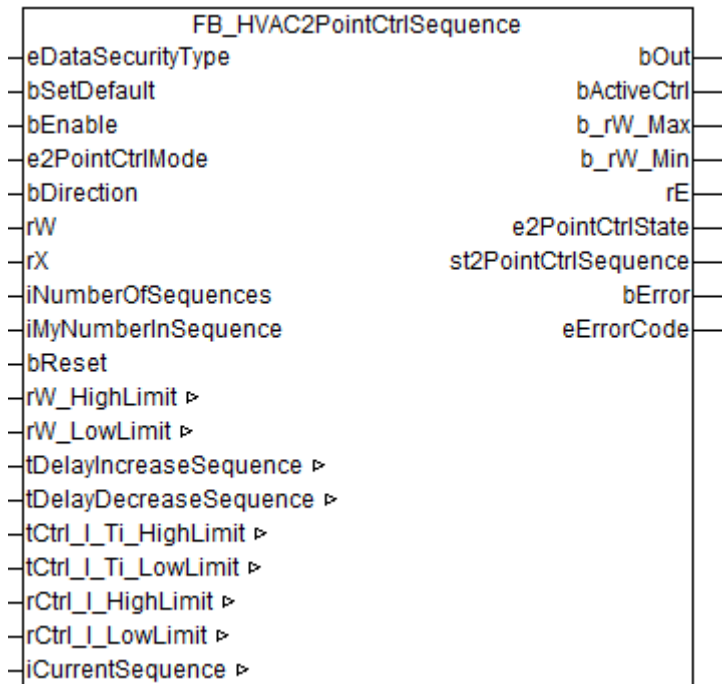
Table of operation modes

Another special feature of the sequence controllers is their control with the enum [E_HVACSequenceCtrlMode \[▶ 522\]](#)

By means of the enum [E_HVACSequenceCtrlMode \[▶ 522\]](#) is used not only to enable control, but also to transmit the operation mode of the air conditioning system to the control function blocks in the sequence. Depending on the operation modes, each sequence controller hence reacts specially to the Enum [E_HVACSequenceCtrlMode \[▶ 522\]](#) as illustrated in the table.

Value of: E_HVACSequenceCtrlMode	0 (Stop)	1 (On)	2 (night cooling)	3 (support operation)	4 (overheating protection)	5 (night cooling and overheating protection)
FB_HVACMasterSequenceCtrl master controller	disabled 0 %	Enable	disabled 0 %	Enable rY=Supply air max. temp.	Enable rY=Supply air min. temp.	Enable rY=Supply air min. temp.
FB_HVACPIDPreHeating preheater	disabled 0 %	Enable	disabled 0 %	Enable 0 % - 100 %	disabled 0 %	disabled 0 %
FB_HVACPIDReHeating reheater	disabled 0 %	Enable	disabled 0 %	disabled 0 %	disabled 0 %	disabled 0 %
FB_HVACPIDCooling cooler	disabled 0 %	Enable	disabled 0 %	disabled 0 %	Enable 0 % - 100 %	Enable 0 % - 100 %
FB_HVACPIDEnergyRecovery heat recovery	disabled 0 %	Enable	disabled 0 %	disabled 0 %	disabled 0 %	disabled 0 %
FB_HVACPIDMixedAir mixed air dampers	disabled 0 %	Enable	max. outside air rate 100 %	0 % air circulation only	0 % air circulation only	max. outside air rate 100 %

3.4.7.1 FB_HVAC2PointCtrlSequence



Application

This function block represents a 2-point sequence controller. It is used for sequential staged control of discontinuous aggregates. It can be used for cascades of vessels and in sequences of refrigerating machines or recooling plants for controlling the power stages. For each power stage an instance of the function block is used.



FB_HVAC2PointCtrlSequence can be used in a control sequence with other sequence controllers from TcHVAC.lib.

Depending on the application several actuators (stages) are used for reaching a controlled variable, which operate in a so-called control sequence. In the control sequence shown below four stages are enabled via the respective output $bOut = TRUE$ of *FB_HVAC2PointCtrlSequence*. With active control only one of the 2-point sequence controllers is active at a time. The outputs of the non-active 2-point sequence controller are fixed. This means that depending on the control direction $bDirection$ of the individual controller; $bOut$ is either TRUE or FALSE.

If the action of the active step is insufficient in the switched-on state, the active 2-point controller switches over to the neighboring 2-point controller on the left or right in the controller sequence via $iCurrentSequence$. This then takes over control of the steps. The previously active controller remains in the end position $bOut = TRUE$ or FALSE, depending on the control direction.

Transfer function of the internal I transfer element

Transfer function of the internal I transfer element

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

Through an internal AND link of the I transfer element consisting of ($tCtrl_I_Ti_HighLimit$, $tCtrl_I_Ti_LowLimit$, $rCtrl_I_HighLimit$, $rCtrl_I_LowLimit$, $st2PointCtrlSequence.rCtrl_I_Out$) and the delay times $tDelayIncreaseSequence/tDelayDecreaseSequence$ switching in the control sequence to the right or left is controlled via $iCurrentSequence$. The active sequence controller is indicated via $bActiveCtrl = TRUE$ ($iCurrentSequence = iMyNumberInSequence$).

The I transfer element and the timing elements for the delay times $tDelayIncreaseSequence$ / $tDelayDecreaseSequence$ are activated if

```

bActiveCtrl = TRUE AND
( (*Decrease*)
(
rX > st2PointCtrlSequence.rW_Max AND ((iMyNumberInSequence <= 1) = FALSE) AND
(
(bDirection = TRUE AND bOut = TRUE) OR
(bDirection = FALSE AND bOut = FALSE)
)
)
OR
( (*Increase*)
rX < st2PointCtrlSequence.rW_Min AND ((iMyNumberInSequence >= iNumberOfSequences) = FALSE) AND
(
(bDirection = FALSE AND bOut = TRUE) OR
(bDirection = TRUE AND bOut = FALSE)
)
)
)

```

Once the I transfer element ($st2PointCtrlSequence.rCtrl_I_Out$ output of the internal I transfer element) AND the timing elements for the delay times $tDelayIncreaseSequence$ / $tDelayDecreaseSequence$ have been activated, switching in the control sequence to the right or left via $iCurrentSequence$ is controlled as follows:

$iCurrentSequence = iCurrentSequence - 1$ if $st2PointCtrlSequence.rCtrl_I_Out \leq st2PointCtrlSequence.rCtrl_I_LowLimit$ AND $st2PointCtrlSequence.tRemainingTimeDecreaseSequence = T\#0s$.

$iCurrentSequence = iCurrentSequence + 1$ if $st2PointCtrlSequence.rCtrl_I_Out \geq st2PointCtrlSequence.rCtrl_I_HighLimit$ AND $st2PointCtrlSequence.tRemainingTimeIncreaseSequence = T\#0s$.

Behavior of the outputs of four FB_HVAC2PointCtrlSequence in a control sequence

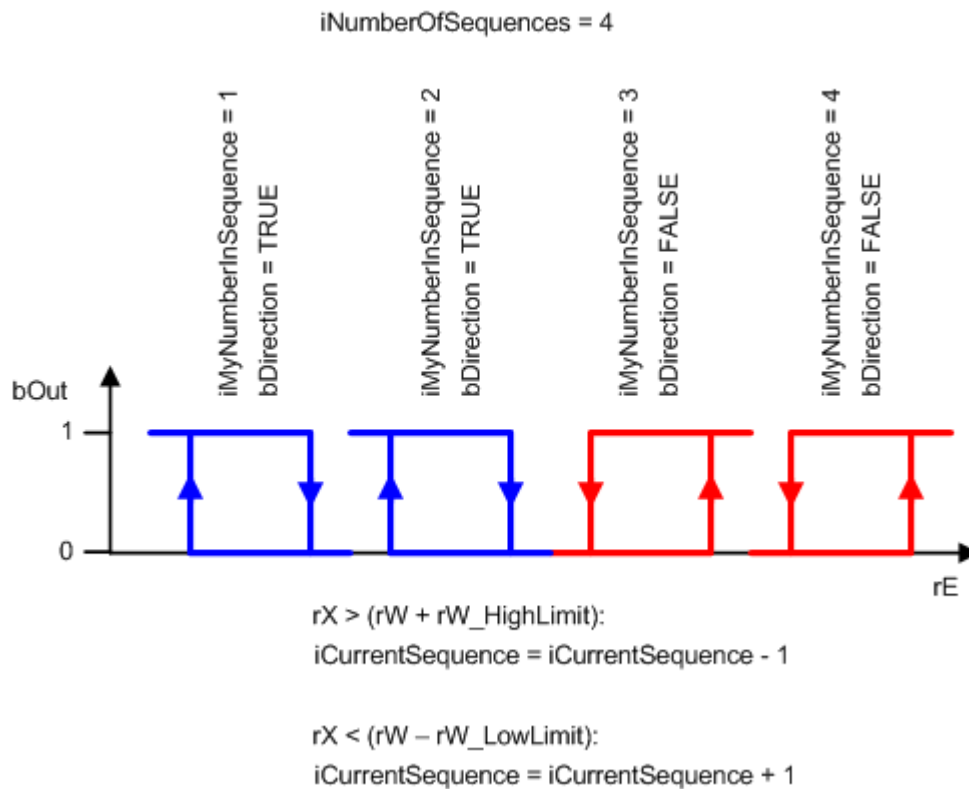


Fig. 16: FB_HVAC2PointCtrlSequence_1

i In the control sequence of four FB_HVAC2PointCtrlSequence controllers shown above there may be no gap in the allocation of iMyNumberInSequence (1,2,3,4) and iNumberOfSequence (4), since otherwise automatic switching of the controllers from the active controller to the adjacent controller to the right or left in the control sequence would not work.

i If bEnable = FALSE and bError = TRUE (e2PointCtrlMode = eHVAC2PointCtrlMode_On_BMS and eHVAC2PointCtrlMode_On_OPODEReHVAC2PointCtrlMode_Off_BMS and eHVAC2PointCtrlMode_Off_OP), automatic switching of the controllers from the active controller to the adjacent controller to the right or left in the control sequence is still operational. The switching mode is therefore always active. The active function block is displayed with bActiveCtrl = TRUE. Depending on the control deviation rE up- or down-switching of the sequence via iCurrentSequence is directly executed. If $rX \geq rW$, then $iCurrentSequence = iCurrentSequence - 1$. If $rX < rW$, then $iCurrentSequence = iCurrentSequence + 1$.

Application example

Download	Required library
TcHVAC.pro [► 531]	TcHVAC.lib

VAR_INPUT


```

eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
e2PointCtrlMode  : E_HVAC2PointCtrlMode;
bDirection        : BOOL;
rW                : REAL;
rX                : REAL;
iNumberOfSequences : INT;           1..32
iMyNumberInSequence: INT;         1..32
bReset            : BOOL;

```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled via TRUE. If `bEnable = FALSE`, the 2-point sequence controller is disabled. Validation of the variables `iNumberOfSequences`, `iMyNumberInSequence`, `iCurrentSequence` and `tTi_Ctrl_I` is still active. If an error occurs, it is displayed with `bError = TRUE` and can be acknowledged with `bReset` once the fault has been corrected.

i If `bEnable = FALSE` and `bError = TRUE` (e2PointCtrlMode = `eHVAC2PointCtrlMode_On_BMS` or `eHVAC2PointCtrlMode_Off_OP`), automatic switching of the controllers from the active controller to the adjacent controller to the right or left in the control sequence is still operational. The switching mode is therefore always active. The active function block is displayed with `bActiveCtrl = TRUE`. Depending on the control deviation `rE` up- or down-switching of the sequence via `iCurrentSequence` is directly executed. If `rX >= rW`, then `iCurrentSequence = iCurrentSequence - 1`. If `rX < rW`, then `iCurrentSequence = iCurrentSequence + 1`.

e2PointCtrlMode: Enum that specifies the operation mode of the 2-point sequence controller. If `bEnable = TRUE` and `bError = FALSE`, the output `bOut` can be switched on or off directly via the Enum. When starting the PLC, `e2PointCtrlMode = eHVAC2PointCtrlMode_Auto_BMS`.

Switching the sequence up or down via `iCurrentSequence` is not dependent on the operating mode `e2PointCtrlMode` of the 2-point sequence controller.

i If `bEnable = FALSE` and `bError = TRUE` (e2PointCtrlMode = `eHVAC2PointCtrlMode_On_BMS` or `eHVAC2PointCtrlMode_Off_OP`), automatic switching of the controllers from the active controller to the adjacent controller to the right or left in the control sequence is still operational. The switching mode is therefore always active. The active function block is displayed with `bActiveCtrl = TRUE`. Depending on the control deviation `rE` up- or down-switching of the sequence via `iCurrentSequence` is directly executed. If `rX >= rW`, then `iCurrentSequence = iCurrentSequence - 1`. If `rX < rW`, then `iCurrentSequence = iCurrentSequence + 1`.

bDirection: `bDirection` determines the control direction of the internal 2-point controller. FALSE = heating mode; TRUE = cooling mode.

rW: The setpoint is transferred with the variable `rW`.

rX: the actual value is transferred with the variable `rX`.

iNumberOfSequences: number of controllers in the sequence. If `iNumberOfSequences <= 0` an error is issued and indicated with `bError = TRUE`. The number of devices in a sequence is not exceeded by the active sequence controller via `iCurrentSequence`.

iMyNumberInSequence: the 2-point controller's own number in the sequence. If `iMyNumberInSequence > iNumberOfSequences` or `iMyNumberInSequence <= 0`, an error is issued and indicated with `bError = TRUE`.

bReset: input for acknowledgement of the faults once they have been corrected. Internally the system responds to a rising edge.

VAR_OUTPUT

```

bOut          : BOOL;
bActiveCtrl   : BOOL;
b_rW_Max      : BOOL;
b_rW_Min      : BOOL;
rE            : REAL;
e2PointCtrlState : E_HVAC2PointCtrlMode;
st2PointCtrlSequence: ST_HVAC2PointCtrlSequence;
bError        : BOOL;
eErrorCode    : E_HVACErrorCodes;

```

bOut: output of the 2-point sequence controller.

bOut becomes TRUE, if

1.
bEnable = TRUE AND *bError* = FALSE AND
 (*e2PointCtrlMode* = *eHVAC2PointCtrlMode_Auto_BMSORe2PointCtrlMode* =
eHVAC2PointCtrlMode_Auto_OP) AND
bActiveCtrl = TRUE AND
 (*bDirection* = TRUE AND (*rX* > *st2PointCtrlSequence.rW_Max*)) OR (*bDirection* = FALSE AND (*rX* <
st2PointCtrlSequence.rW_Min))
)
2.
bEnable = TRUE AND *bError* = FALSE AND
 (*e2PointCtrlMode* = *eHVAC2PointCtrlMode_Auto_BMSORe2PointCtrlMode* =
eHVAC2PointCtrlMode_Auto_OP) AND
 (*bDirection* = TRUE AND (*iCurrentSequence* < *iMyNumberInSequence*)) OR (*bDirection* = FALSE AND
 (*iCurrentSequence* > *iMyNumberInSequence*))
)
3.
bEnable = TRUE AND *bError* = FALSE AND
 (*e2PointCtrlMode* = *eHVAC2PointCtrlMode_On_BMSORe2PointCtrlMode* =
eHVAC2PointCtrlMode_On_OP)

bOut becomes FALSE, if

1.
bEnable = FALSE OR *bError* = TRUE
2.
bEnable = TRUE AND *bError* = FALSE AND
 (*e2PointCtrlMode* = *eHVAC2PointCtrlMode_Auto_BMSORe2PointCtrlMode* =
eHVAC2PointCtrlMode_Auto_OP) AND
bActiveCtrl = TRUE AND
 (*bDirection* = TRUE AND (*rX* < *st2PointCtrlSequence.rW_Min*)) OR (*bDirection* = FALSE AND (*rX* >
st2PointCtrlSequence.rW_Max))
)
3.
bEnable = TRUE AND *bError* = FALSE AND
 (*e2PointCtrlMode* = *eHVAC2PointCtrlMode_Auto_BMSORe2PointCtrlMode* =
eHVAC2PointCtrlMode_Auto_OP) AND
 (*bDirection* = TRUE AND (*iCurrentSequence* > *iMyNumberInSequence*)) OR (*bDirection* = FALSE AND
 (*iCurrentSequence* < *iMyNumberInSequence*))
)

4.

$bEnable = TRUE \wedge bError = FALSE \wedge$
 $(e2PointCtrlMode = eHVAC2PointCtrlMode_Off_BMSORe2PointCtrlMode =$
 $eHVAC2PointCtrlMode_Off_OP)$

bActiveCtrl: $bActiveCtrl$ indicates with TRUE that the function block is the active one in the sequence. $bActiveCtrl$ becomes TRUE, if $bEnable = TRUE$, $bError = FALSE \wedge iCurrentSequence = iMyNumberInSequence$.

b_rW_Max: b_rW_Max becomes TRUE, if $rX > st2PointCtrlSequence.rW_Max$.

b_rW_Min: b_rW_Min becomes TRUE, if $rX < st2PointCtrlSequence.rW_Min$.

rE: control deviation: $rE = rW - rX$

e2PointCtrlState: Enum indicating the state of the operation mode $e2PointCtrlMode$ of the 2-point sequence controller

st2PointCtrlSequence: the structure indicates various states, inputs and outputs of the function block. Furthermore the remaining times of the VAR_IN_OUT variables $tDelayIncreaseSequence$ and $tDelayDecreaseSequence$ are output if their function is active. Via $st2PointCtrlSequence.rCtrl_I_Out$ the output signal of the internal I transfer element is displayed.

st2PointCtrlSequence.tRemainingTimeIncreaseSequence: remaining delay time $tDelayIncreaseSequence$.

st2PointCtrlSequence.tRemainingTimeDecreaseSequence: remaining delay time $tDelayDecreaseSequence$.

st2PointCtrlSequence.rX: state of rX

st2PointCtrlSequence.rW_Max: $st2PointCtrlSequence.rW_Max := rW + rW_HighLimit$ – upper setpoint limit; if rX exceeds it, the internal I transfer element and the timing elements of the delay times $tDelayIncreaseSequence/tDelayDecreaseSequence$ can be activated or deactivated respectively, see [Transfer function of the internal I transfer element \[► 290\]](#) in this document

st2PointCtrlSequence.rW_Min: $st2PointCtrlSequence.rW_Min := rW - rW_LowLimit$ - lower setpoint limit; if rX falls below it, the internal I transfer element and the timing elements of the delay times $tDelayIncreaseSequence/tDelayDecreaseSequence$ can be activated or deactivated respectively, see [Transfer function of the internal I transfer element \[► 290\]](#) in this document

st2PointCtrlSequence.rE: control deviation: $rE = rW - rX$

st2PointCtrlSequence.rCtrl_I_HighLimit: upper limit at which the integration of the internal I transfer element is stopped. $st2PointCtrlSequence.rCtrl_I_HighLimit = rCtrl_I_HighLimit$

st2PointCtrlSequence.rCtrl_I_LowLimit: lower limit at which the integration of the internal I transfer element is stopped. $st2PointCtrlSequence.rCtrl_I_Low = rCtrl_I_LowLimit * (-1)$

st2PointCtrlSequence.rCtrl_I_Out: output of the internal I transfer element.
 If $st2PointCtrlSequence.rCtrl_I_Out = st2PointCtrlSequence.rCtrl_I_HighLimit$ OR
 $st2PointCtrlSequence.rCtrl_I_LowLimit$ AND
 either $st2PointCtrlSequence.tRemainingTimeIncreaseSequence$ OR
 $st2PointCtrlSequence.tRemainingTimeDecreaseSequence = T\#0s$ AND
 $bActive = TRUE$, then the number of the active controller $iCurrentSequence$ is incremented or decremented by 1, depending on $bDirection$.

st2PointCtrlSequence.e2PointCtrlState: see $e2PointCtrlState$

st2PointCtrlSequence.iNumberOfSequences: see $iNumberOfSequences$

st2PointCtrlSequence.iMyNumberInSequence: see $iMyNumberInSequence$

st2PointCtrlSequence.iCurrentSequence: see $iCurrentSequence$

st2PointCtrlSequence.bEnable: see $bEnable$

st2PointCtrlSequence.bError: see $bError$

st2PointCtrlSequence.bOut: see *bOut*

st2PointCtrlSequence.bActiveCtrl: see *bActiveCtrl*

st2PointCtrlSequence.b_rW_Max: see *b_rW_Max*

st2PointCtrlSequence.b_rW_Min: see *b_rW_Min*

bError: the output is TRUE if an error is active and one of the variables *iNumberOfSequences*, *iMyNumberInSequence* or *iCurrentSequence* has an incorrect parameter. Once the fault has been corrected the *bError* message must be acknowledged with *bReset*. The *eErrorCode* enum indicates the error number. If *bError* = TRUE, then the output *bOut* = FALSE.

i If *bEnable* = FALSE and *bError* = TRUE (e2PointCtrlMode = eHVAC2PointCtrlMode_On_BMSODEReHVAC2PointCtrlMode_On_OPODEReHVAC2PointCtrlMode_Off_BMSODEReHVAC2PointCtrlMode_Off_OP), automatic switching of the controllers from the active controller to the adjacent controller to the right or left in the control sequence is still operational. The switching mode is therefore always active. The active function block is displayed with *bActiveCtrl* = TRUE. Depending on the control deviation *rE* up- or down-switching of the sequence via *iCurrentSequence* is directly executed. If $rX \geq rW$, then $iCurrentSequence = iCurrentSequence - 1$. If $rX < rW$, then $iCurrentSequence = iCurrentSequence + 1$.

eErrorCode: returns the error number [► 520] when the *bError* output is set. The following errors may occur in this function block: *eHVACErrorCodes_Error_iMyNumberInSequence*, *eHVACErrorCodes_Error_iNumberOfSequences*, *eHVACErrorCodes_Error_iCurrentSequences*

i To get the error numbers of the enum in the PLC, *eErrorCode* can be assigned to a variable of data type WORD. *eHVACErrorCodes_Error_iNumberOfSequences* = 27
eHVACErrorCodes_Error_iMyNumberInSequence = 28
eHVACErrorCodes_Error_iCurrentSequences = 29

VAR_IN_OUT

```
rW_HighLimit      : REAL;
rW_LowLimit       : REAL;
tDelayIncreaseSequence : TIME;
tDelayDecreaseSequence : TIME;
tCtrl_I_Ti_HighLimit  : TIME;
tCtrl_I_Ti_LowLimit  : TIME;
rCtrl_I_HighLimit    : REAL;
rCtrl_I_LowLimit     : REAL;
iCurrentSequence    : INT;
```

rW_HighLimit: positive value of the upper limit of the control deviation. $st2PointCtrlSequence.rW_Max = rW + rW_HighLimit$. The variable is saved persistently. Preset to 5.

rW_LowLimit: positive value of the lower limit of the control deviation. $st2PointCtrlSequence.rW_Min = rW - rW_LowLimit$. The variable is saved persistently. Preset to 5.

tDelayIncreaseSequence: delay time after which *iCurrentSequence* is increased by 1, see [Transfer function of the internal I transfer element \[► 290\]](#) in this document. The variable is saved persistently. Preset to 5 min.

tDelayDecreaseSequence: delay time after which *iCurrentSequence* is decreased by 1, see [Transfer function of the internal I transfer element \[► 290\]](#) in this document. The variable is saved persistently. Preset to 5 min.

tCtrl_I_Ti_HighLimit: integration time for the upper limit of the internal I transfer element, see [Transfer function of the internal I transfer element \[► 290\]](#) in this document. *tCtrl_I_Ti_HighLimit* must be > T#0s. The variable is saved persistently. Preset to 10 min.

tCtrl_I_Ti_LowLimit: integration time for the lower limit of the internal I transfer element, see [Transfer function of the internal I transfer element \[► 290\]](#) in this document. *tCtrl_I_Ti_LowLimit* must be > T#0s. The variable is saved persistently. Preset to 10 min.

rCtrl_I_HighLimit: positive value for the upper limit at which the integration of the internal I transfer element is stopped (ARW measure, anti-reset windup), see [Transfer function of the internal I transfer element \[► 290\]](#) in this document. The variable is saved persistently. Preset to 10.

rCtrl_I_LowLimit: negative value for the lower limit at which the integration of the internal I transfer element is stopped (ARW measure, anti-reset windup), see [Transfer function of the internal I transfer element \[► 290\]](#) in this document. The variable is saved persistently. Preset to -10.

iCurrentSequence: number of the active controller in the sequence (0..32). The number of devices in a sequence *iNumberOfSequences* is not exceeded by the active sequence controller via *iCurrentSequence*. If *iCurrentSequence* > *iNumberOfSequences* or *iCurrentSequence* < 0, an error is indicated with *bError* = TRUE.

The sequence is switched up or down via *iCurrentSequence* depending on the control deviation *rE* if the function block is active in the sequence *bActiveCtrl* = TRUE.

1. $iCurrentSequence = iCurrentSequence - 1$ if $st2PointCtrlSequence.rCtrl_I_Out \geq st2PointCtrlSequence.rLimit_Ctrl_I_Min$ AND $st2PointCtrlSequence.tRemainingTimeDecreaseSequence = T\#0s$ AND $bActiveCtrl = TRUE$

2. $iCurrentSequence = iCurrentSequence + 1$ if $st2PointCtrlSequence.rCtrl_I_Out \geq st2PointCtrlSequence.rLimit_Ctrl_I_Max$ AND $st2PointCtrlSequence.tRemainingTimeIncreaseSequence = T\#0s$ AND $bActiveCtrl = TRUE$

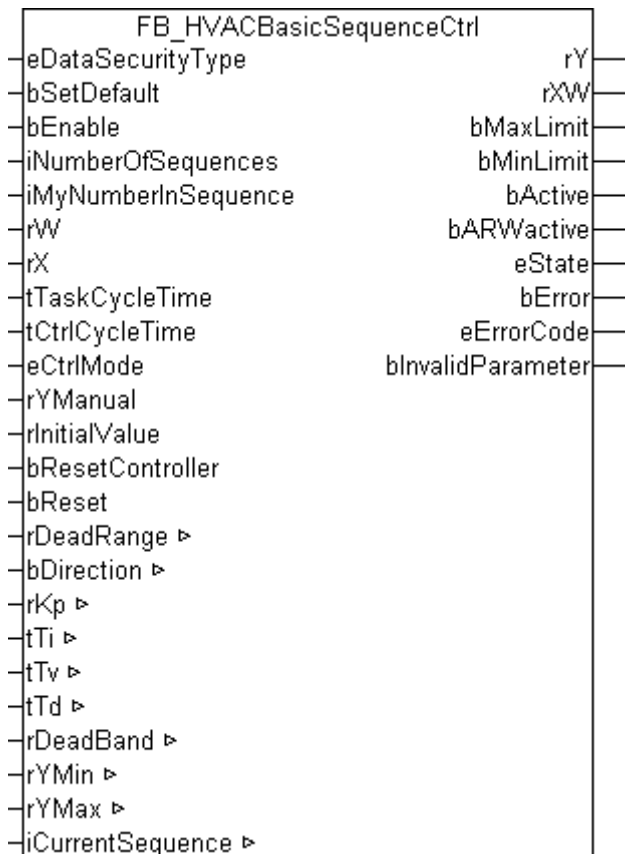
i To ensure correct functioning of the automatic switchover of the controller from the active controller to the neighboring controller on the left or right in the control sequence, *iCurrentSequence* may not be written to continuously from the outside in a control sequence. When starting a control sequence it must be defined which sequence controller is active. *iCurrentSequence* must be written for a PLC cycle and be > 0 and <= *iNumberOfSequences*.

i If *bEnable* = FALSE OR *bError* = TRUE OR *e2PointCtrlMode* = *eHVAC2PointCtrlMode_On_BMS* OR *e2PointCtrlMode* = *eHVAC2PointCtrlMode_On_OPO* OR *e2PointCtrlMode* = *eHVAC2PointCtrlMode_Off_BMS* OR *e2PointCtrlMode* = *eHVAC2PointCtrlMode_Off_OP*, automatic switching of the controllers from the active controller to the adjacent controller to the right or left in the control sequence is still operational. The switching mode is therefore always active. The active function block is displayed with *bActiveCtrl* = TRUE. Depending on the control deviation *rE* up- or down-switching of the sequence via *iCurrentSequence* is directly executed. If $rX \geq rW$, then $iCurrentSequence = iCurrentSequence - 1$. If $rX < rW$, then $iCurrentSequence = iCurrentSequence + 1$.

Documents about this

 [example_persistent_e.zip \(Resources/zip/11659714827.zip\)](#)

3.4.7.2 FB_HVACBasicSequenceCtrl



Application

Unlike the sequence controllers FB_HVACPIDCooling, FB_HVACDehumidify, FB_HVACEnergyRecovery, FB_HVACPIDHumidify, FB_HVACPIDMixedAir, FB_HVACPIDPreHeating and FB_HVACPIDReHeating, the function block FB_HVACBasicSequenceCtrl has no special system-specific extensions or application. It is more general. The control direction of the controller is not automatically determined depending on the room air and outside air as is the case with the function blocks for heat recovery or the mixed air chamber.


A sequence of static heating and cooling ceiling can be implemented in the field of room automation with this function block.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
iNumberOfSequences : INT;
iMyNumberInSequence : INT;
rW                : REAL;
rX                : REAL;
tTaskCycleTime    : TIME;
tCtrlCycleTime    : TIME;
eCtrlMode         : E_HVACCtrlMode;
rYManual          : REAL;
rInitialValue     : REAL;
bResetController  : BOOL;
bReset            : BOOL;
```

eDataSecurityType: if eDataSecurityType:= *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: input variable for enabling the controller. The controller is active if `bEnable = TRUE`.

iNumberOfSequences: number of sequence controllers in the system.

iMyNumberInSequence: own number in the controller in the sequence.

rW: the setpoint is transferred to the controller with the variable `rW`.

rX: actual value of the control loop.

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.

eCtrlMode: the operation mode is selected via this enum.

rYManual: manual value.

rInitialValue: the restart behavior of the controller is influenced by `rInitialValue`.

bResetController: a positive edge on the input `bResetController` resets the PID controller.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
rY           : REAL;
rXW          : REAL;
bMaxLimit    : BOOL;
bMinLimit    : BOOL;
bActive      : BOOL;
bARWactive   : BOOL;
eState       : E_HVACState;
bError       : BOOL;
eErrorCode   : E_HVACErrorCodes;
bInvalidParameter: BOOL;
```

rY: control signal output of the PID controller.

rXW: control deviation

bMaxLimit: the output `bMaxLimit` is TRUE, if the output `rY` has reached the value `rYMax`.

bMinLimit: the output `bMinLimit` is TRUE, if the output `rY` has reached the value `rYMin`.

bActive: `bActive` is TRUE, if the controller is active and enabled.

bARWactive: `bARWactive` is TRUE, if the integral component of the controller has reached the lower or upper control value limit.

eState: state of the controller. See [E_HVACState](#). [► 523]

bError: this output indicates with a TRUE that there is an error.

eErrorCode: contains the command-specific error code. See [E_HVACErrorCodes](#). [► 520]

bInvalidParameter: TRUE if an error occurs during the plausibility check. The message must be acknowledged with *bReset*.

VAR_IN_OUT

```
rDeadRange      : REAL;
bDirection      : BOOL;
rKp             : REAL;
tTi            : TIME;
tTv            : TIME;
tTd            : TIME;
rDeadBand      : REAL;
rYMin          : REAL;
rYMax          : REAL;
iCurrentSequence: INT;
```

rDeadRange: in order to avoid unnecessary driving and hence premature wear of the valves or damper drives, a dead range (0..32767) can be set for the controller output signal *rY*. This means that a control signal change is only active if the change of value is greater than the dead range. A constant change of the control signal *rY* is converted to a pulsating drive of the actuator if a dead range is specified. The larger the dead range the larger the pauses and the control signal jumps will be. The variable is saved persistently. Preset to 0.

bDirection: the control direction of the controller can be changed with the parameter *bDirection*. If *bDirection* is TRUE, the direct control direction for cooling operation of the controller is active. If *bDirection* is FALSE, the indirect control direction of the controller is activated for heating operation. The variable is saved persistently. Preset to FALSE.

rKp: proportional factor gain. The variable is saved persistently. Preset to 1.

tTi: integral action time. The I-part corrects the residual control deviation following correction of the P-part. The smaller the *tTi* time is set, the faster the controller corrects. The control loop becomes unstable if the time is too short. Larger *tTi*-times must be entered in order to reduce the integration component. The integral action time should be selected to be longer than the stroke time of the valve or damper drive. The variable is saved persistently. Preset to 30 s.

tTv: rate time. The larger *tTv* is, the stronger the controller corrects. The control loop becomes unstable if the time is too long. Often in normal building automation applications only a PI controller is used. In this case zero must be entered for *tTv*. The variable is saved persistently. Preset to 0 s.

tTd : damping time. The variable is saved persistently. Preset to 0 s.

rDeadBand: if the control value is at the lower or upper limit of a controller and the actual value of the controlled system oscillates around the setpoint with a small amplitude, frequent switching back and forth between two sequence controllers can be damped by an additional parameter for the switchover. To this end, the difference between the actual values and the setpoints of the controlled system is integrated after the sequence controller has reached its lower or upper limit. Switching to the next sequence only takes place if the sum of this integration is greater than the value of *rDeadband* ([see sample](#)). [► 530] The variable is saved persistently. Preset to 0.

rYMin: lower limit of the working range of the controller. The variable is saved persistently. Preset to 0.

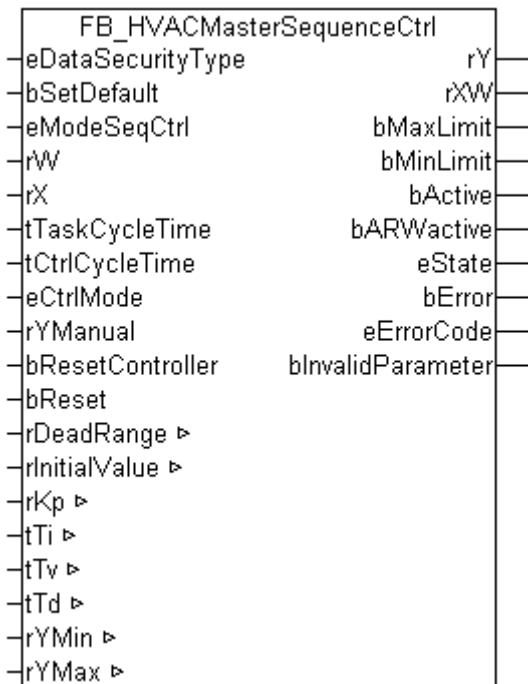
rYMax: upper limit of the working range of the controller. The variable is saved persistently. Preset to 100.

iCurrentSequence: currently active controller in the sequence.

Documents about this

 [example_persistent_e.zip](#) (Resources/zip/11659714827.zip)

3.4.7.3 FB_HVACMasterSequenceCtrl



Application

Improved controllability can be achieved with the aid of the exhaust air or room temperature cascade control. The master controller measures the room or exhaust air temperature and adapts the setpoint for the supply air temperature to the room conditions. The setpoint for the supply air is limited by a minimum and a maximum value.


In backup mode of the air conditioning system (*eModeSeqCtrl = eHVACSequenceCtrlMode_FreezeProtection*) the value of *rYMaxis* switched directly through to the output *rY*. If overheating protection is active (*eModeSeqCtrl = eHVACSequenceCtrlMode_OverheatingProtection*) the value of *rYMinis* switched directly through to the output *rY*.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
eModeSeqCtrl     : E_HVACSequenceCtrlMode;
rW               : REAL;
rX               : REAL;
tTaskCycleTime  : TIME;
tCtrlCycleTime  : TIME;
eCtrlMode       : E_HVACCtrlMode;
rYManual        : REAL;
bResetController: BOOL;
bReset          : BOOL;
```

eDataSecurityType: if *eDataSecurityType = eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

eModeSeqCtrl: the plant operation mode is transferred to the controller with the enum `eModeSeqCtrl`.

rX: acquires the actual value of the control loop.

rW: the setpoint is transferred to the controller with the variable `rW`.

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 `tTaskCycleTime`. 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.

eCtrlMode: the operation mode is selected via this enum. Manual or automatic operation mode.

rYManual: manual value for the manual operation mode.

bResetController: the internal variables of the PID controller are reset.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
rY           : REAL;
rXW          : REAL;
bMaxLimit    : BOOL;
bMinLimit    : BOOL;
bActive      : BOOL;
bARWactive   : BOOL;
eState       : E_HVACState;
bError       : BOOL;
eErrorCode   : E_HVACErrorCodes;
bInvalidParameter: BOOL;
```

rY: temperature setpoint for the supply air temperature controller.

rXW: control deviation.

bMaxLimit: the output `bMaxLimit` is TRUE, if the output `rY` has reached the value `rYMax`.

bMinLimit: the output `bMinLimit` is TRUE, if the output `rY` has reached the value `rYMin`.

bActive: `bActive` is TRUE, if the controller is active and enabled.

bARWactive: `bARWactive` is TRUE, if the integral component of the controller has reached the lower or upper control value limit.

eState: state of the controller. See [E_HVACState](#). [[▶ 523](#)]

bError: this output indicates with a TRUE that there is an error.

eErrorCode: contains the command-specific error code. See [E_HVACErrorCodes](#) [[▶ 520](#)].

bInvalidParameter: TRUE if an error occurs during the plausibility check. The message must be acknowledged with `bReset`.

VAR_IN_OUT

```
rDeadRange   : REAL;
rInitialValue : REAL;
rKp           : REAL;
```

```
tTi      : TIME;  
tTv      : TIME;  
tTd      : TIME;  
rYMin    : REAL;  
rYMax    : REAL;
```

rDeadRange: in order to avoid unnecessary driving and hence premature wear of the valves or damper drives, a dead range (0..32767) can be set for the controller output signal *rY*. This means that a control signal change is only active if the change of value is greater than the dead range. A constant change of the control signal *rY* is converted to a pulsating drive of the actuator if a dead range is specified. The larger the dead range the larger the pauses and the control signal jumps will be. The variable is saved persistently. Preset to 0.

rInitialValue: the restart behavior of the controller is influenced by *rInitialValue* (0..32767). The variable is saved persistently. Preset to 0.

rKp: proportional factor gain. The variable is saved persistently. Preset to 1.

tTi: integral action time. The I-part corrects the residual control deviation following correction of the P-part. The smaller the *tTi* time is set, the faster the controller corrects. The control loop becomes unstable if the time is too short. Larger *tTi*-times must be entered in order to reduce the integration component. The integral action time should be selected to be longer than the stroke time of the valve or damper drive. The variable is saved persistently. Preset to 30 s.

tTv: rate time. The larger *tTv* is, the stronger the controller corrects. The control loop becomes unstable if the time is too long. Often in normal building automation applications only a PI controller is used. In this case zero must be entered for *tTv*. The variable is saved persistently. Preset to 0 s.

tTd : damping time. The variable is saved persistently. Preset to 0 s.

rYMin: lower limit of the working range of the controller. The variable is saved persistently. Preset to 16.

rYMax: upper limit of the working range of the controller. The variable is saved persistently. Preset to 25.

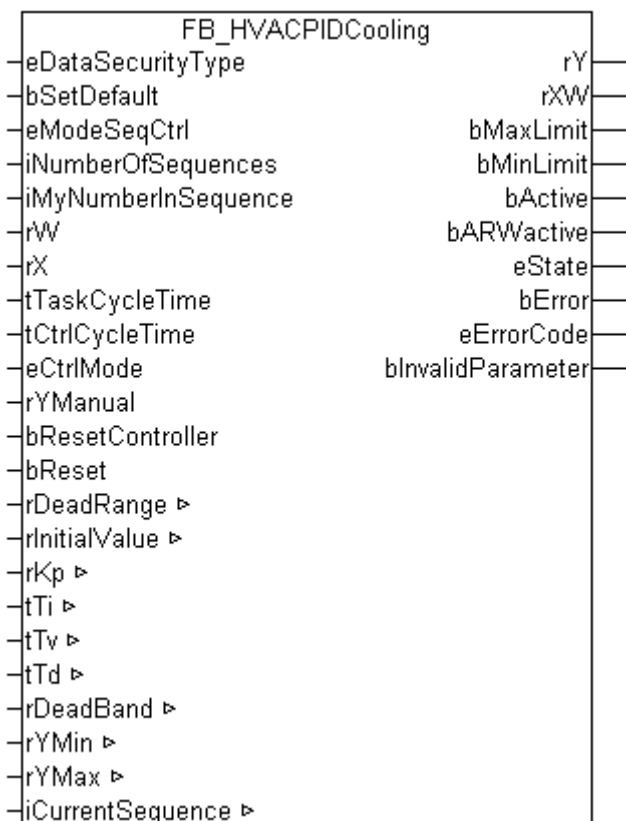
Documents about this

 [example_persistent_e.zip \(Resources/zip/11659714827.zip\)](#)

3.4.7.4 FB_HVACPIDCooling

PID controller cooler

The control direction of the cooling controller is inverted compared to the heating controller. If the inlet air temperature set value is reduced, the sequence controller raises its control signal. In the operation mode Overheating Protection, the cooler regulates the supply air temperature to the minimum. Outside the normal sequence control operation the controller is also activated if overheating protection is active.



VAR_INPUT

```

eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
eModeSeqCtrl      : E_HVACSequenceCtrlMode;
iNumberOfSequences : INT;
iMyNumberInSequence : INT;
rW                : REAL;
rX                : REAL;
tTaskCycleTime    : TIME;
tCtrlCycleTime    : TIME;
eCtrlMode         : E_HVACCtrlMode;
rYManual          : REAL;
bResetController  : BOOL;
bReset            : BOOL;

```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instantiated once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

eModeSeqCtrl: among other things, control is enabled via this enum. In addition, the operation mode of the VAC system is transmitted to the controller blocks in the sequence. See also [Table operation modes](#). [► 531]

iNumberOfSequences: number of sequence controllers in the system.

iMyNumberInSequence: the controller's own number in the sequence.

rW: the setpoint is transferred to the controller with the variable rW.

rX: actual value of the control loop.

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: the variable *tCtrlCycleTime* specifies the cycle time with which the PID controller is processed. The shortest possible cycle time is that of the controller. Since the controlled systems in building automation are predominantly slow, the cycle time of the controller can be several times that of the control cycle time.

eCtrlMode: the operation mode is selected via this enum. Manual or automatic operation mode.

rYManual: manual value that is set at the output rY if *eCtrlMode* = *eHVACCtrlMode_Manual*.

bResetController: a positive edge on the input *bResetController* resets the PID controller.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
rY           : REAL;
rXW          : REAL;
bMaxLimit    : BOOL;
bMinLimit    : BOOL;
bActive      : BOOL;
bARWactive   : BOOL;
eState       : E_HVACState;
bError       : BOOL;
eErrorCode   : E_HVACErrorCodes;
bInvalidParameter: BOOL;
```

rY: control signal output of the PID controller.

rXW: control deviation.

bMaxLimit: the output *bMaxLimit* is TRUE, if the output rY has reached the value *rYMax*.

bMinLimit: the output *bMinLimit* is TRUE, if the output rY has reached the value *rYMin*.

bActive: *bActive* is TRUE, if the controller is active and enabled.

bARWactive: *bARWactive* is TRUE, if the integral component of the controller has reached the lower or upper control value limit.

eState: state of the controller. See [E_HVACState](#). [► 523]

bError: this output indicates with a TRUE that there is an error.

eErrorCode: *E_HVACErrorCodes*. See [error codes](#) [► 520].

bInvalidParameter: TRUE if an error occurs during the plausibility check. The message must be acknowledged with *bReset*.

VAR_IN_OUT

```
rDeadRange   : REAL;
rInitialValue : REAL;
rKp           : REAL;
tTi          : TIME;
tTv          : TIME;
tTd          : TIME;
```

```

rDeadBand      : REAL;
rYMin          : REAL;
rYMax          : REAL;
iCurrentSequence: INT;

```

rDeadRange: in order to avoid unnecessary driving and hence premature wear of the valves or damper drives, a dead range can be set for the controller output signal rY . This means that a control signal change is only active if the change of value is greater than the dead range. A constant change of the control signal rY is converted to a pulsating drive of the actuator if a dead range is specified. The larger the dead range the larger the pauses and the control signal jumps will be. The variable is saved persistently. Preset to 0.

rInitialValue: the restart behavior of the controller is influenced by *rInitialValue*. The variable is saved persistently. Preset to 0.

rKp: proportional factor gain. The variable is saved persistently. Preset to 0.

tTi: integral action time. The I-part corrects the residual control deviation following correction of the P-part. The smaller the tTi time is set, the faster the controller corrects. The control loop becomes unstable if the time is too short. Larger tTi -times must be entered in order to reduce the integration component. The integral action time should be selected to be longer than the stroke time of the valve or damper drive. The variable is saved persistently. Preset to 30 s.

tTv: rate time. The larger tTv is, the stronger the controller corrects. The control loop becomes unstable if the time is too long. Often in normal building automation applications only a PI controller is used. In this case zero must be entered for tTv . The variable is saved persistently. Preset to 0 s.

tTd: damping time. The variable is saved persistently. Preset to 0 s.

rDeadBand: if the control value is at the lower or upper limit of a controller and the actual value of the controlled system oscillates around the setpoint with a small amplitude, frequent switching back and forth between two sequence controllers can be damped by an additional parameter for the switchover. To this end, the difference between the actual values and the setpoints of the controlled system is integrated after the sequence controller has reached its lower or upper limit. Switching to the next sequence only takes place if the sum of this integration is greater than the value of *rDeadband* (see sample). [▶ 530] The variable is saved persistently. Preset to 0.

rYMin: lower limit of the working range of the controller. The variable is saved persistently. Preset to 0.

rYMax: upper limit of the working range of the controller. The variable is saved persistently. Preset to 100.

iCurrentSequence: number of the active controller in the sequence.

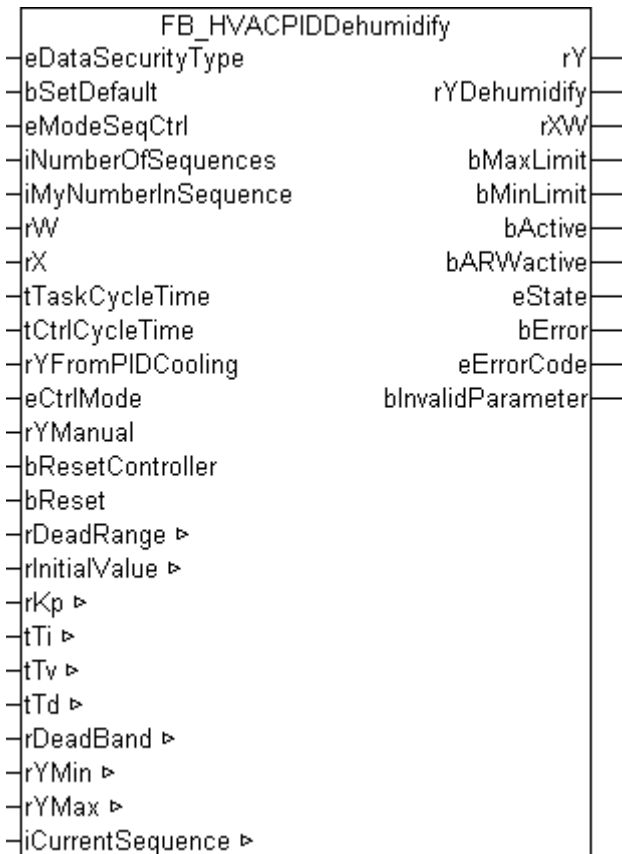
Documents about this

 example_persistent_e.zip (Resources/zip/11659714827.zip)

3.4.7.5 FB_HVACPIDDehumidify

PID dehumidification controller

In order to dehumidify the supply air, it is cooled down by the cooler. The dehumidification is constantly controlled by dosing the coolant at the cooling register with the cooler valve. Two controllers, [FB_HVACPIDCooling \[▶ 303\]](#) and [FB_HVACPIDDehumidify](#), act on the cooler valve. First of all the control signal rY is relayed by [FB_HVACPIDCooling \[▶ 303\]](#) to the dehumidification sequence controller [FB_HVACPIDDehumidify](#). Inside the [FB_HVACPIDDehumidify](#) function block, the larger of the two control signals is routed through to the controller output. If the air humidity is too high, the dehumidification controller takes precedence over the cooling controller. However, in order that the correct supply air temperature can still be achieved, the preheater is disabled if dehumidification is in operation. The reheater is put into operation in order to reheat the air.




VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
eModeSeqCtrl     : E_HVACSequenceCtrlMode;
iNumberOfSequences : INT;
iMyNumberInSequence : INT;
rW               : REAL;
rX               : REAL;
tTaskCycleTime  : TIME;
tCtrlCycleTime  : TIME;
rYFromPIDCooling : REAL;
eCtrlMode       : E_HVACCtrlMode;
rYManual        : REAL;
bResetcontroller : BOOL;
bReset          : BOOL;
```

eDataSecurityType: if eDataSecurityType:= *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If eDataSecurityType:= *eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType:= *eHVACDataSecurityType_Persistent*. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

eModeSeqCtrl: among other things, control is enabled via this enum. In addition, the operation mode of the VAC system is transmitted to the controller function block in the sequence. See also [Table operation modes. \[► 531\]](#)

iNumberOfSequences: number of sequence controllers in the system.

iMyNumberInSequence: the controller's own number in the sequence.

rW: the setpoint is transferred to the controller with the variable rW.

rX: actual value of the control loop.

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: the variable *tCtrlCycleTime* specifies the cycle time with which the PID controller is processed. The shortest possible cycle time is that of the controller. Since the controlled systems in building automation are predominantly slow, the cycle time of the controller can be several times that of the control cycle time.

rYFromPIDCooling: in the case of a VAC system with dehumidification, the output of the cooling controller is connected to this input. Inside the function block FB_HVACPIDDehumidify there is a MAX selection that relays the larger of the control signals from the two controllers FB_HVACPIDDehumidify and FB_HVACPIDCooling to the cooler actuator valve.

eCtrlMode: the operation mode is selected via this enum. Manual or automatic operation mode.

rYManual: manual value that is set at the output rY if *eCtrlMode = eHVACCtrlMode_Manual*.

bResetController: a positive edge on the input *bResetController* resets the PID controller.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
rY                : REAL;
rYDehumidify     : REAL;
rXW              : REAL;
bMaxLimit        : BOOL;
bMinLimit        : BOOL;
bActive          : BOOL;
bARWactive       : BOOL;
eState           : E_HVACState;
bError           : BOOL;
eErrorCode       : E_HVACErrorCodes;
bInvalidParameter: BOOL;
```

rY: control signal output of the PID controller.

rYDehumidify: control signal output of the dehumidification controller.

rXW: control deviation.

bMaxLimit: the output *bMaxLimit* is TRUE, if the output rY has reached the value *rYMax*.

bMinLimit: the output *bMinLimit* is TRUE, if the output rY has reached the value *rYMin*.

bActive: *bActive* is TRUE, if the controller is active and enabled.

bARWactive: *bARWactive* is TRUE, if the integral component of the controller has reached the lower or upper control value limit.

eState: state of the controller. See [E_HVACState. \[► 523\]](#)

bError: this output indicates with a TRUE that there is an error.

eErrorCode: contains the command-specific error code. See [E_HVACErrorCodes. \[► 520\]](#)

InvalidParameter: TRUE if an error occurs during the plausibility check. The message must be acknowledged with *bReset*.

VAR_IN_OUT

```
rDeadRange      : REAL;
rInitialValue    : REAL;
rKp              : REAL;
tTi              : TIME;
tTv              : TIME;
tTd              : TIME;
rDeadBand        : REAL;
rYMin            : REAL;
rYMax            : REAL;
iCurrentSequence: INT;
```

rDeadRange: in order to avoid unnecessary driving and hence premature wear of the valves or damper drives, a dead range can be set for the controller output signal *rY*. This means that a control signal change is only active if the change of value is greater than the dead range. A constant change of the control signal *rY* is converted to a pulsating drive of the actuator if a dead range is specified. The larger the dead range the larger the pauses and the control signal jumps will be. The variable is saved persistently. Preset to 0.

rInitialValue: the restart behavior of the controller is influenced by *rInitialValue*. The variable is saved persistently. Preset to 0.

rKp: proportional factor gain. The variable is saved persistently. Preset to 0.

tTi: integral action time. The I-part corrects the residual control deviation following correction of the P-part. The smaller the *tTi* time is set, the faster the controller corrects. The control loop becomes unstable if the time is too short. Larger *tTi*-times must be entered in order to reduce the integration component. The integral action time should be selected to be longer than the stroke time of the valve or damper drive. The variable is saved persistently. Preset to 30 s.

tTv: rate time. The larger *tTv* is, the stronger the controller corrects. The control loop becomes unstable if the time is too long. Often in normal building automation applications only a PI controller is used. In this case zero must be entered for *tTv*. The variable is saved persistently. Preset to 0 s.

tTd : damping time. The variable is saved persistently. Preset to 0 s.

rDeadBand: if the control value is at the lower or upper limit of a controller and the actual value of the controlled system oscillates around the setpoint with a small amplitude, frequent switching back and forth between two sequence controllers can be damped by an additional parameter for the switchover. To this end, the difference between the actual values and the setpoints of the controlled system is integrated after the sequence controller has reached its lower or upper limit. Switching to the next sequence only takes place if the sum of this integration is greater than the value of *rDeadband* (see sample). [► 530] The variable is saved persistently. Preset to 0.

rYMin: lower limit of the working range of the controller. The variable is saved persistently. Preset to 0.

rYMax: upper limit of the working range of the controller. The variable is saved persistently. Preset to 100.

iCurrentSequence: number of the active controller in the sequence.

Documents about this

 example_persistent_e.zip (Resources/zip/11659714827.zip)

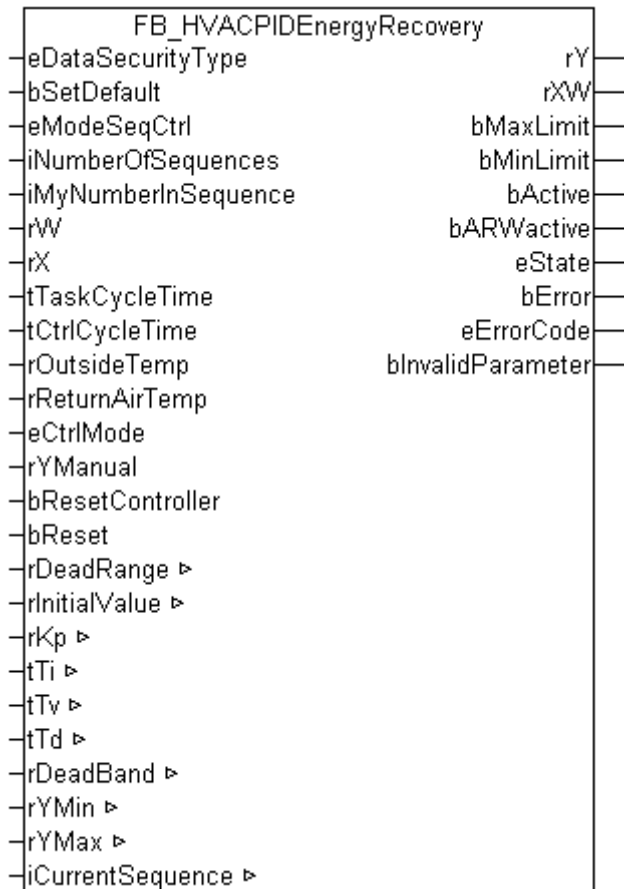
3.4.7.6 FB_HVACPIDEnergyRecovery

PID heat recovery controller

One of the special features of the heat recovery controller is the reversal of its control direction depending on the outside temperature and the exhaust air temperature. The enthalpies of the outside air and the exhaust air can be used instead of their temperatures.

If the outside temperature is higher than the exhaust air temperature, the heat recovery system can use the exhaust air or room air, which was cooled with a high expenditure of energy, to cool the outside air. A reduction of the supply air temperature, leads in cooling mode to an increase in the control value for the heat

recovery (curve falls from left to right). If the outside temperature is lower than the exhaust air temperature, the heat recovery system is used to feed the exhaust air heat back to the supply air. The heat recovery system is then in heating mode (curve rises from left to right).




VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
eModeSeqCtrl      : E_HVACSequenceCtrlMode;
iNumberOfSequences : INT;
iMyNumberInSequence : INT;
rW                : REAL;
rX                : REAL;
tTaskCycleTime    : TIME;
tCtrlCycleTime    : TIME;
rOutsideTemp      : REAL;
rReturnAirTemp    : REAL;
eCtrlMode         : E_HVACCtrlMode;
rYManual          : REAL;
bResetController  : BOOL;
bReset            : BOOL;
```

eDataSecurityType: if eDataSecurityType:= *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

eModeSeqCtrl: among other things, control is enabled via this enum. In addition, the operation mode of the VAC system is transmitted to the controller function block in the sequence. See also [Table operation modes](#). [[▶ 531](#)]

iNumberOfSequences: number of sequence controllers in the system.

iMyNumberInSequence: the controller's own number in the sequence.

rW: the setpoint is transferred to the controller with the variable rW.

rX: actual value of the control loop.

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: the variable `tCtrlCycleTime` specifies the cycle time with which the PID controller is processed. The shortest possible cycle time is that of the controller. Since the controlled systems in building automation are predominantly slow, the cycle time of the controller can be several times that of the control cycle time.

rOutsideTemp: outside temperature.

rReturnAirTemp: exhaust air temperature.

eCtrlMode: the operation mode is selected via this enum. Manual or automatic operation mode.

rYManual: manual value that is set at the output rY if `eCtrlMode = eHVACCtrlMode_Manual`.

bResetController: a positive edge on the input `bResetController` resets the PID controller.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
rY                : REAL;
rXW               : REAL;
bMaxLimit         : BOOL;
bMinLimit         : BOOL;
bActive           : BOOL;
bARWactive        : BOOL;
eState            : E_HVACState;
bError            : BOOL;
eErrorCode        : E_HVACErrorCodes;
bInvalidParameter: BOOL;
```

rY: control signal output of the PID controller.

rXW: control deviation

bMaxLimit: the output `bMaxLimit` is TRUE, if the output rY has reached the value `rYMax`.

bMinLimit: the output `bMinLimit` is TRUE, if the output rY has reached the value `rYMin`.

bActive: `bActive` is TRUE, if the controller is active and enabled.

bARWactive: `bARWactive` is TRUE, if the integral component of the controller has reached the lower or upper control value limit.

eState: state of the controller. See [E_HVACState](#). [[▶ 523](#)]

bError: this output indicates with a TRUE that there is an error.

eErrorCode: contains the command-specific error code. See [E_HVACErrorCodes](#) [▶ 520].

blInvalidParameter: TRUE if an error occurs during the plausibility check. The message must be acknowledged with *bReset*.

VAR_IN_OUT

```
rDeadRange      : REAL;
rInitialValue   : REAL;
rKp             : REAL;
tTi            : TIME;
tTv            : TIME;
tTd            : TIME;
rDeadBand      : REAL;
rYMin          : REAL;
rYMax          : REAL;
iCurrentSequence: INT;
```

rDeadRange: in order to avoid unnecessary driving and hence premature wear of the valves or damper drives, a dead range can be set for the controller output signal *rY*. This means that a control signal change is only active if the change of value is greater than the dead range. A constant change of the control signal *rY* is converted to a pulsating drive of the actuator if a dead range is specified. The larger the dead range the larger the pauses and the control signal jumps will be. The variable is saved persistently. Preset to 0.

rInitialValue: the restart behavior of the controller is influenced by *rInitialValue*. The variable is saved persistently. Preset to 0.

rKp: proportional factor gain. The variable is saved persistently. Preset to 0.

tTi: integral action time. The I-part corrects the residual control deviation following correction of the P-part. The smaller the *tTi* time is set, the faster the controller corrects. The control loop becomes unstable if the time is too short. Larger *tTi*-times must be entered in order to reduce the integration component. The integral action time should be selected to be longer than the stroke time of the valve or damper drive. The variable is saved persistently. Preset to 30 s.

tTv: rate time. The larger *tTv* is, the stronger the controller corrects. The control loop becomes unstable if the time is too long. Often in normal building automation applications only a PI controller is used. In this case zero must be entered for *tTv*. The variable is saved persistently. Preset to 0 s.

tTd : damping time. The variable is saved persistently. Preset to 0 s.

rDeadBand: if the control value is at the lower or upper limit of a controller and the actual value of the controlled system oscillates around the setpoint with a small amplitude, frequent switching back and forth between two sequence controllers can be damped by an additional parameter for the switchover. To this end, the difference between the actual values and the setpoints of the controlled system is integrated after the sequence controller has reached its lower or upper limit. Switching to the next sequence only takes place if the sum of this integration is greater than the value of *rDeadband* (see sample). [▶ 530] The variable is saved persistently. Preset to 0.

rYMin: lower limit of the working range of the controller. The variable is saved persistently. Preset to 0.

rYMax: upper limit of the working range of the controller. The variable is saved persistently. Preset to 100.

iCurrentSequence: number of the active controller in the sequence.

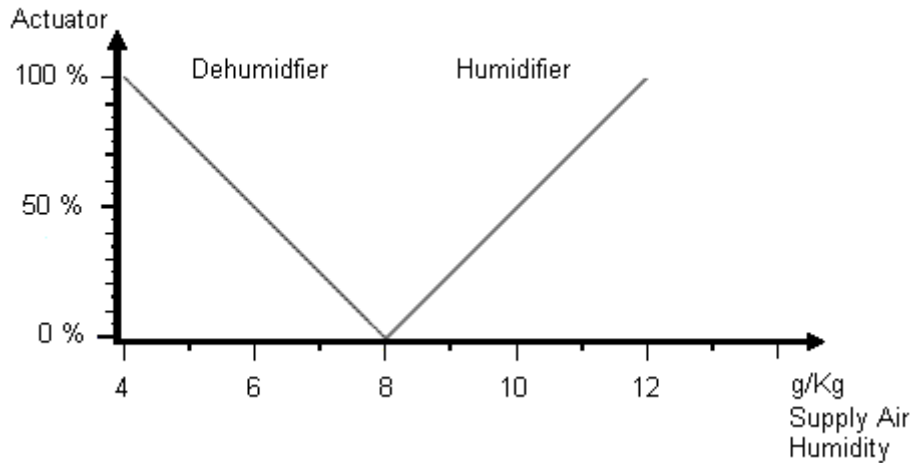
Documents about this

📄 [example_persistent_e.zip](#) (Resources/zip/11659714827.zip)

3.4.7.7 FB_HVACPIDHumidify

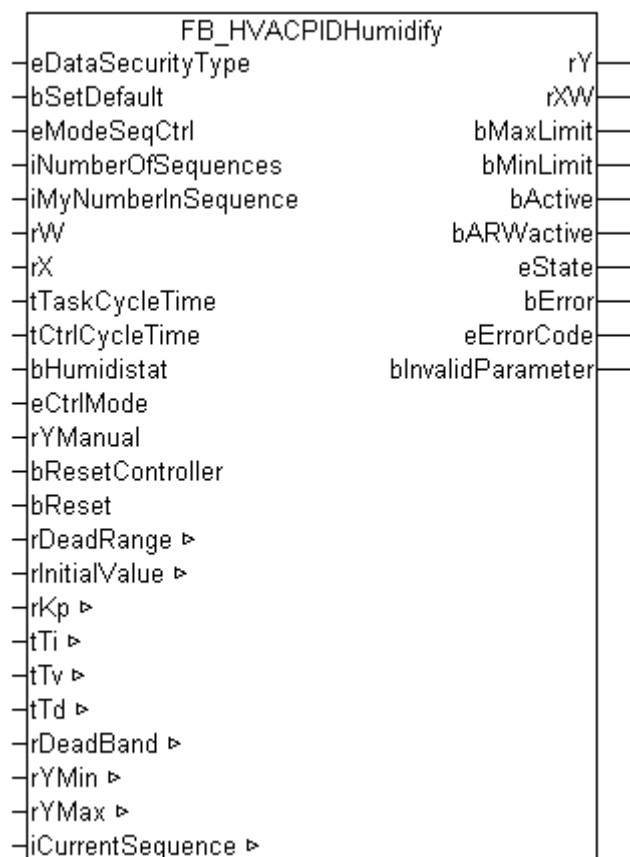
Humidity control

Besides the temperature, the humidity of the air is also controlled in some air conditioning systems. Steam humidifiers or air washers are used for humidification. For dehumidification the supply air can be cooled down to the dew point by a cooler. As a result, the moisture in the air condenses on the air cooler. The control of humidification and dehumidification is realized with a sequence comprising a humidification controller and a dehumidification controller.



PID humidification controller

This function block is a special application of a PID controller. The humidity controller is only enabled if the function block input *bHumidistat* is TRUE.



VAR_INPUT


```

eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
eModeSeqCtrl     : E_HVACSequenceCtrlMode;
iNumberOfSequences : INT;
iMyNumberInSequence : INT;
rW               : REAL;
rX               : REAL;
tTaskCycleTime  : TIME;
tCtrlCycleTime  : TIME;
bHumidistat     : BOOL;
eCtrlMode       : E_HVACCtrlMode;
rYManual        : REAL;
bResetController : BOOL;
bReset          : BOOL;

```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

eModeSeqCtrl: among other things, this Enum enables the controller.°In addition, the operation mode of the VAC system is transmitted to the controller blocks in the sequence. See also [Table operation modes. \[► 531\]](#)

iNumberOfSequences: number of sequence controllers in the system.

iMyNumberInSequence: the own number from the controller in the sequence.

rW: the setpoint is transferred to the controller with the variable rW.

rX: actual value of the control loop.

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: the variable `tCtrlCycleTime` specifies the cycle time with which the PID controller is processed. The shortest possible cycle time is that of the controller. Since the controlled systems in building automation are predominantly slow, the cycle time of the controller can be several times that of the control cycle time.

bHumidistat: a TRUE on this input enables the humidity control. The hygostat is connected to this input so that over-humidification cannot occur.

eCtrlMode: the operation mode is selected via this enum. Manual or automatic operation mode.

rYManual: manual value that is set at the output rY if `eCtrlMode = eHVACCtrlMode_Manual`.

bResetController: a positive edge on the input `bResetController` resets the PID controller.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
rY                : REAL;
rXW               : REAL;
bMaxLimit        : BOOL;
bMinLimit        : BOOL;
bActive          : BOOL;
bARWactive       : BOOL;
eState           : E_HVACState;
bError           : BOOL;
eErrorCode       : E_HVACErrorCodes;
bInvalidParameter: BOOL;
```

rY: control signal output of the PID controller.

rXW: control deviation.

bMaxLimit: the output `bMaxLimit` is TRUE, if the output rY has reached the value `rYMax`.

bMinLimit: the output `bMinLimit` is TRUE, if the output rY has reached the value `rYMin`.

bActive: `bActive` is TRUE, if the controller is active and enabled.

bARWactive: `bARWactive` is TRUE, if the integral component of the controller has reached the lower or upper control value limit.

eState: state of the controller. See [E_HVACState. \[► 523\]](#)

bError: this output indicates with a TRUE that there is an error.

eErrorCode: contains the command-specific error code. See [E_HVACErrorCodes \[► 520\]](#).

blInvalidParameter: TRUE if an error occurs during the plausibility check. The message must be acknowledged with *bReset*.

VAR_IN_OUT

```
rDeadRange      : REAL;
rInitialValue   : REAL;
rKp             : REAL;
tTi            : TIME;
tTv            : TIME;
tTd            : TIME;
rDeadBand      : REAL;
rYMin          : REAL;
rYMax          : REAL;
iCurrentSequence : INT;
```

rDeadRange: in order to avoid unnecessary driving and hence premature wear of the valves or damper drives, a dead range can be set for the controller output signal *rY*. This means that a control signal change is only active if the change of value is greater than the dead range. A constant change of the control signal *rY* is converted to a pulsating drive of the actuator if a dead range is specified. The larger the dead range the larger the pauses and the control signal jumps will be. The variable is saved persistently. Preset to 0.

rInitialValue: the restart behavior of the controller is influenced by *rInitialValue*. The variable is saved persistently. Preset to 0.

rKp: proportional factor gain. The variable is saved persistently. Preset to 0.

tTi: integral action time. The I-part corrects the residual control deviation following correction of the P-part. The smaller the *tTi* time is set, the faster the controller corrects. The control loop becomes unstable if the time is too short. Larger *tTi*-times must be entered in order to reduce the integration component. The integral action time should be selected to be longer than the stroke time of the valve or damper drive. The variable is saved persistently. Preset to 30 s.

tTv: rate time. The larger *tTv* is, the stronger the controller corrects. The control loop becomes unstable if the time is too long. Often in normal building automation applications only a PI controller is used. In this case zero must be entered for *tTv*. The variable is saved persistently. Preset to 0 s.

tTd: damping time. The variable is saved persistently. Preset to 0 s.

rDeadBand: if the control value is at the lower or upper limit of a controller and the actual value of the controlled system oscillates around the setpoint with a small amplitude, frequent switching back and forth between two sequence controllers can be damped by an additional parameter for the switchover. To this end, the difference between the actual values and the setpoints of the controlled system is integrated after the sequence controller has reached its lower or upper limit. Switching to the next sequence only takes place if the sum of this integration is greater than the value of *rDeadband* (see sample). [► 530] The variable is saved persistently. Preset to 0.

rYMin: lower limit of the working range of the controller. The variable is saved persistently. Preset to 0.

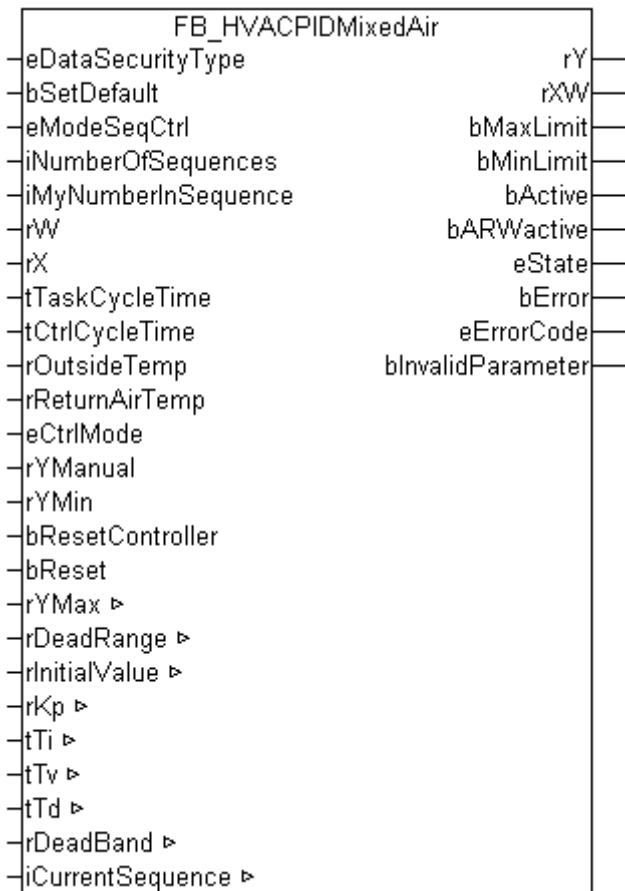
rYMax: upper limit of the working range of the controller. The variable is saved persistently. Preset to 100.

iCurrentSequence: number of the active controller in the sequence.

Documents about this

 example_persistent_e.zip (Resources/zip/11659714827.zip)

3.4.7.8 FB_HVACPIDMixedAir



PID mixed air chamber controller

One of the special features of the mixed air controller is the reversal of its control direction depending on the outside temperature and the exhaust air temperature. The enthalpies of the outside air and the exhaust air can be used instead of their temperatures.

If the outside temperature is lower than the exhaust air temperature, the air can be used for cooling in the case of high thermal loads inside the building. Hence, the outside air rate increases as the control variable or the supply air temperature set value decreases (curve falls from left to right).

If the outside temperature is higher than the exhaust air temperature, the outside air proportion can be increased in the event of the supply air temperature set value being increased (curve rises from left to right). The outside air temperature is applied to the input variable *rOutsideTemp*. The variable *rReturnAirTemp* is for the exhaust air or room temperature. The Night Cooling operating mode (*eHVACSequenceCtrlMode_NightCooling*) is transmitted to the function block by means of the Enum *eModeSeqCtrl*. During night cooling the control is deactivated and 100% control value is output at *rY*. Further information on the Summer Night Cooling function can be found under [FB_HVACSummerNightCooling](#) [► 461].

The maximum and minimum outside air rates can be set with the parameters *rYMax* and *rYMin*.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
eModeSeqCtrl      : E_HVACSequenceCtrlMode;
iNumberOfSequences : INT;
iMyNumberInSequence : INT;
rW                : REAL;
rX                : REAL;
tTaskCycleTime    : TIME;
tCtrlCycleTime    : TIME;
```


```

rOutsideTemp      : REAL;
rReturnAirTemp    : REAL;
eCtrlMode         : E_HVACCtrlMode;
rYManual          : REAL;
rYMin             : REAL;
bResetController  : BOOL;
bReset            : BOOL;

```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

eModeSeqCtrl: among other things, control is enabled via this enum. In addition, the operation mode of the VAC system is transmitted to the controller blocks in the sequence. See also [Table operation modes. \[► 531\]](#)

iNumberOfSequences: number of sequence controllers in the system.

iMyNumberInSequence: the own number from the controller from the sequence.

rW: the setpoint is transferred to the controller with the variable `rW`.

rX: actual value of the control loop.

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: the variable `tCtrlCycleTime` specifies the cycle time with which the PID controller is processed. The shortest possible cycle time is that of the controller. Since the controlled systems in building automation are predominantly slow, the cycle time of the controller can be several times that of the control cycle time.

rOutsideTemp: outside temperature

rReturnAirTemp: exhaust air temperature

eCtrlMode: the operation mode is selected via this enum. Manual or automatic operation mode.

rYManual: manual value that is set at the output `rY` if `eCtrlMode = eHVACCtrlMode_Manual`.

rYMin: lower limit of the working range of the controller.

bResetController: a positive edge on the input `bResetController` resets the PID controller.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```

rY                : REAL;
rXW               : REAL;
bMaxLimit         : BOOL;
bMinLimit         : BOOL;

```

```

bActive       : BOOL;
bARWactive    : BOOL;
eState        : E_HVACState;
bError        : BOOL;
eErrorCode    : E_HVACErrorCodes;
bInvalidParameter: BOOL;

```

rY: control signal output of the PID controller.

rXW: control deviation

bMaxLimit: the output *bMaxLimit* is TRUE, if the output *rY* has reached the value *rYMax*.

bMinLimit: the output *bMinLimit* is TRUE, if the output *rY* has reached the value *rYMin*.

bActive: *bActive* is TRUE, if the controller is active and enabled.

bARWactive: *bARWactive* is TRUE, if the integral component of the controller has reached the lower or upper control value limit.

eState: state of the controller. See [E_HVACState](#). [► 523]

bError: this output indicates with a TRUE that there is an error.

eErrorCode: contains the command-specific error code. See [E_HVACErrorCodes](#) [► 520].

bInvalidParameter: TRUE if an error occurs during the plausibility check. The message must be acknowledged with *bReset*.

VAR_IN_OUT

```

rDeadRange    : REAL;
rInitialValue : REAL;
rKp           : REAL;
tTi           : TIME;
tTv           : TIME;
tTd           : TIME;
rDeadBand     : REAL;
rYMax         : REAL;
iCurrentSequence: INT;

```

rDeadRange: in order to avoid unnecessary driving and hence premature wear of the valves or damper drives, a dead range can be set for the controller output signal *rY*. This means that a control signal change is only active if the change of value is greater than the dead range. A constant change of the control signal *rY* is converted to a pulsating drive of the actuator if a dead range is specified. The larger the dead range the larger the pauses and the control signal jumps will be. The variable is saved persistently. Preset to 0.

rInitialValue: the restart behavior of the controller is influenced by *rInitialValue*. The variable is saved persistently. Preset to 0.

rKp: proportional factor gain. The variable is saved persistently. Preset to 0.

tTi: integral action time. The I-part corrects the residual control deviation following correction of the P-part. The smaller the *tTi* time is set, the faster the controller corrects. The control loop becomes unstable if the time is too short. Larger *tTi*-times must be entered in order to reduce the integration component. The integral action time should be selected to be longer than the stroke time of the valve or damper drive. The variable is saved persistently. Preset to 30 s.

tTv: rate time. The larger *tTv* is, the stronger the controller corrects. The control loop becomes unstable if the time is too long. Often in normal building automation applications only a PI controller is used. In this case zero must be entered for *tTv*. The variable is saved persistently. Preset to 0 s.

tTd : damping time. The variable is saved persistently. Preset to 0 s.

rDeadBand: if the control value is at the lower or upper limit of a controller and the actual value of the controlled system oscillates around the setpoint with a small amplitude, frequent switching back and forth between two sequence controllers can be damped by an additional parameter for the switchover. To this end, the difference between the actual values and the setpoints of the controlled system is integrated after

the sequence controller has reached its lower or upper limit. Switching to the next sequence only takes place if the sum of this integration is greater than the value of *rDeadband* (see sample). [► 530] The variable is saved persistently. Preset to 0.

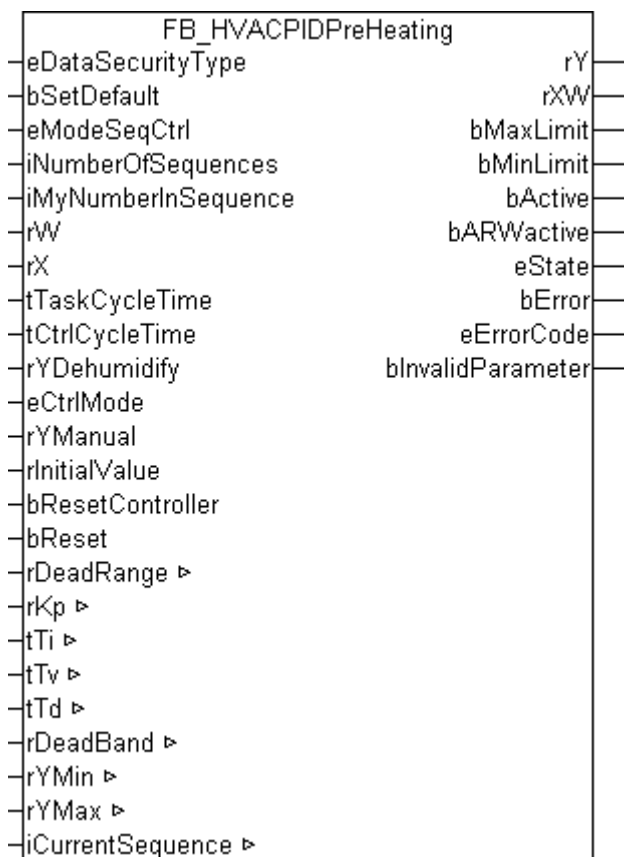
rYMax: upper limit of the working range of the controller. The variable is saved persistently. Preset to 100.

iCurrentSequence: number of the active controller in the sequence.

Documents about this

📄 example_persistent_e.zip (Resources/zip/11659714827.zip)

3.4.7.9 FB_HVACPIDPreHeating



PID controller preheater

The function block is a special application of a PID controller.

In addition to the PID control algorithm, the conditions of switching to the sequence of the upstream or downstream sequence controller are included. A special feature of the pre-heater controller is the input *rYDehumidify*. As soon as the signal of *rYDehumidify* is greater than zero, the preheater is disabled and the value of *iCurrentSequence* is incremented by one. This activates the reheater [FB_HVACPIDReHeating](#) [► 323] [► 323] instead of the preheater. If the value of *rYDehumidify* returns to zero, the preheater is automatically reactivated. Outside the normal sequence control operation the controller is also activated in the case of backup operation (freeze protection).

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
eModeSeqCtrl      : E_HVACSequenceCtrlMode;
iNumberOfSequences : INT;
iMyNumberInSequence : INT;
rW                : REAL;
rX                : REAL;
tTaskCycleTime    : TIME;
tCtrlCycleTime    : TIME;
```




```

rYDehumidify      : REAL;
eCtrlMode         : E_HVACCtrlMode;
rYManual          : REAL;
rInitialValue     : REAL;
bResetController  : BOOL;
bReset            : BOOL;

```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

eModeSeqCtrl: among other things, control is enabled via this enum. In addition, the operation mode of the VAC system is transmitted to the controller function block in the sequence. See also [Table operation modes](#). [► 531]

iNumberOfSequences: number of sequence controllers in the system.

iMyNumberInSequence: the controller's own number in the sequence.

rW: the setpoint is transferred to the controller with the variable `rW`.

rX: actual value of the control loop.

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: the variable `rCtrlCycleTime` specifies the cycle time with which the PID controller is processed. The shortest possible cycle time is that of the controller. Since the controlled systems in building automation are predominantly slow, the cycle time of the controller can be several times that of the control cycle time.

rYDehumidify: in the case of an VAC system with dehumidification, the control value of the dehumidification controller `FB_HVACPIDDehumidify` [► 306] is connected to this input. If the value of `rYDehumidify > 0`, then the VAR_IN_OUT variable `iCurrentSequence` will be increased automatically by one. In dehumidification mode the system will thus be switched from the preheater `FB_HVACPIDPreHeating` to the reheater `FB_HVACPIDReHeating`.

eCtrlMode: the operation mode is selected via this enum. Manual or automatic operation mode.

rYManual: manual value that is set at the output `rY` if `eCtrlMode = eHVACCtrlMode_Manual`.

rInitialValue: the restart behavior of the controller is influenced by `rInitialValue`.

bResetController: a positive edge on the input `bResetController` resets the PID controller.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```

rY           : REAL;
rXW         : REAL;
bMaxLimit   : BOOL;
bMinLimit   : BOOL;
bActive     : BOOL;
bARWactive  : BOOL;
eState      : E_HVACState;
bError      : BOOL;
eErrorCode  : E_HVACErrorCodes;
bInvalidParameter: BOOL;

```

rY: control signal output of the PID controller.

rXW: control deviation

bMaxLimit: the output *bMaxLimit* is TRUE, if the output *rY* has reached the value *rYMax*.

bMinLimit: the output *bMinLimit* is TRUE, if the output *rY* has reached the value *rYMin*.

bActive: *bActive* is TRUE, if the controller is active and enabled.

bARWactive: *bARWactive* is TRUE, if the integral component of the controller has reached the lower or upper control value limit.

eState: state of the controller. See [E_HVACState](#). [[▶ 523](#)]

bError: this output indicates with a TRUE that there is an error.

eErrorCode: contains the command-specific error code. See [E_HVACErrorCodes](#) [[▶ 520](#)].

bInvalidParameter: TRUE if an error occurs during the plausibility check. The message must be acknowledged with *bReset*.

VAR_IN_OUT

```

rDeadRange  : REAL;
rKp         : REAL;
tTi         : TIME;
tTv         : TIME;
tTd         : TIME;
rDeadBand   : REAL;
rYMin       : REAL;
rYMax       : REAL;
iCurrentSequence: INT;

```

rDeadRange: in order to avoid unnecessary driving and hence premature wear of the valves or damper drives, a dead range can be set for the controller output signal *rY*. This means that a control signal change is only active if the change of value is greater than the dead range. A constant change of the control signal *rY* is converted to a pulsating drive of the actuator if a dead range is specified. The larger the dead range the larger the pauses and the control signal jumps will be. The variable is saved persistently. Preset to 0.

rKp: proportional factor gain. The variable is saved persistently. Preset to 0.

tTi: integral action time. The I-part corrects the residual control deviation following correction of the P-part. The smaller the *tTi* time is set, the faster the controller corrects. The control loop becomes unstable if the time is too short. Larger *tTi*-times must be entered in order to reduce the integration component. The integral action time should be selected to be longer than the stroke time of the valve or damper drive. The variable is saved persistently. Preset to 30 s.

tTv: rate time. The larger *tTv* is, the stronger the controller corrects. The control loop becomes unstable if the time is too long. Often in normal building automation applications only a PI controller is used. In this case zero must be entered for *tTv*. The variable is saved persistently. Preset to 0 s.

tTd : damping time. The variable is saved persistently. Preset to 0 s.

rDeadBand: if the control value is at the lower or upper limit of a controller and the actual value of the controlled system oscillates around the setpoint with a small amplitude, frequent switching back and forth between two sequence controllers can be damped by an additional parameter for the switchover. To this end, the difference between the actual values and the setpoints of the controlled system is integrated after

the sequence controller has reached its lower or upper limit. Switching to the next sequence only takes place if the sum of this integration is greater than the value of *rDeadband* (see sample). [▶ 530] The variable is saved persistently. Preset to 0.

rYMin: lower limit of the working range of the controller. The variable is saved persistently. Preset to 0.

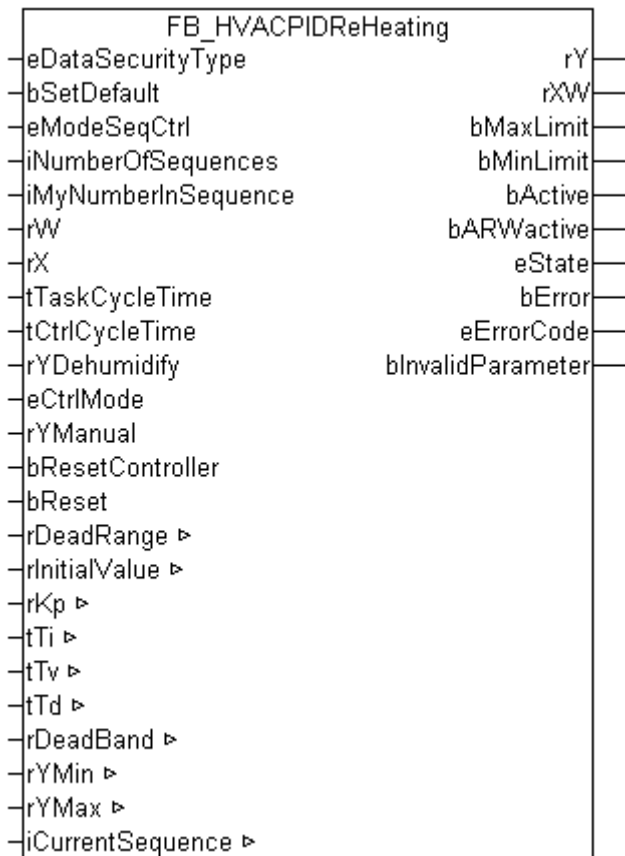
rYMax: upper limit of the working range of the controller. The variable is saved persistently. Preset to 100.

iCurrentSequence: number of the active controller in the sequence.

Documents about this

- example_persistent_e.zip (Resources/zip/11659714827.zip)

3.4.7.10 FB_HVACPIDReHeating



PID controller reheater

The reheater is used to reheat the supply air that was cooled down by the cooler for dehumidification. The controller can only be enabled within the normal **sequence control operation**.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
eModeSeqCtrl      : E_HVACSequenceCtrlMode;
iNumberOfSequences : INT;
iMyNumberInSequence : INT;
rW                : REAL;
rX                : REAL;
tTaskCycleTime    : TIME;
tCtrlCycleTime    : TIME;
rYDehumidify      : REAL;
eCtrlMode         : E_HVACCtrlMode;
```


```

rYManual      : REAL;
bResetController : BOOL;
bReset       : BOOL;

```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

eModeSeqCtrl: among other things, control is enabled via this enum. In addition, the operation mode of the VAC system is transmitted to the controller blocks in the sequence. See also [Table operation modes](#). ▶ 531

iNumberOfSequences: number of sequence controllers in the system.

iMyNumberInSequence: the controller's own number in the sequence.

rW: the setpoint is transferred to the controller with the variable rW.

rX: actual value of the control loop.

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: the variable `tCtrlCycleTime` specifies the cycle time with which the PID controller is processed. The shortest possible cycle time is that of the controller. Since the controlled systems in building automation are predominantly slow, the cycle time of the controller can be several times that of the control cycle time.

rYDehumidify: constant value.

eCtrlMode: the operation mode is selected via this enum. Manual or automatic operation mode.

rYManual: manual value that is set at the output rY if `eCtrlMode = eHVACCtrlMode_Manual`.

bResetController: a positive edge on the input `bResetController` resets the PID controller.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```

rY      : REAL;
rXW     : REAL;
bMaxLimit : BOOL;
bMinLimit : BOOL;
bActive  : BOOL;
bARWactive : BOOL;
eState   : E_HVACState;
bError   : BOOL;
eErrorCode : E_HVACErrorCodes;
bInvalidParameter : BOOL;

```

rY: control signal output of the PID controller.

rXW: control deviation.

bMaxLimit: the output *bMaxLimit* is TRUE, if the output *rY* has reached the value *rYMax*.

bMinLimit: the output *bMinLimit* is TRUE, if the output *rY* has reached the value *rYMin*.

bActive: *bActive* is TRUE, if the controller is active and enabled.

bARWactive: *bARWactive* is TRUE, if the integral component of the controller has reached the lower or upper control value limit.

eState: state of the controller. See [E_HVACState](#). [[▶ 523](#)]

bError: this output indicates with a TRUE that there is an error.

eErrorCode: contains the command-specific error code. See [E_HVACErrorCodes](#) [[▶ 520](#)].

InvalidParameter: TRUE if an error occurs during the plausibility check. The message must be acknowledged with *bReset*.

VAR_IN_OUT

```
rDeadRange      : REAL;
rInitialValue   : REAL;
rKp             : REAL;
tTi            : TIME;
tTv            : TIME;
tTd            : TIME;
rDeadBand      : REAL;
rYMin          : REAL;
rYMax          : REAL;
iCurrentSequence: INT;
```

rDeadRange: in order to avoid unnecessary driving and hence premature wear of the valves or damper drives, a dead range can be set for the controller output signal *rY*. This means that a control signal change is only active if the change of value is greater than the dead range. A constant change of the control signal *rY* is converted to a pulsating drive of the actuator if a dead range is specified. The larger the dead range the larger the pauses and the control signal jumps will be. The variable is saved persistently. Preset to 0.

rInitialValue: the restart behavior of the controller is influenced by *rInitialValue*. The variable is saved persistently. Preset to 0.

rKp: proportional factor gain. The variable is saved persistently. Preset to 0.

tTi: integral action time. The I-part corrects the residual control deviation following correction of the P-part. The smaller the *tTi* time is set, the faster the controller corrects. The control loop becomes unstable if the time is too short. Larger *tTi*-times must be entered in order to reduce the integration component. The integral action time should be selected to be longer than the stroke time of the valve or damper drive. The variable is saved persistently. Preset to 30 s.

tTv: rate time. The larger *tTv* is, the stronger the controller corrects. The control loop becomes unstable if the time is too long. Often in normal building automation applications only a PI controller is used. In this case zero must be entered for *tTv*. The variable is saved persistently. Preset to 0 s.

tTd : damping time. The variable is saved persistently. Preset to 0 s.

rDeadBand: if the control value is at the lower or upper limit of a controller and the actual value of the controlled system oscillates around the setpoint with a small amplitude, frequent switching back and forth between two sequence controllers can be damped by an additional parameter for the switchover. To this end, the difference between the actual values and the setpoints of the controlled system is integrated after the sequence controller has reached its lower or upper limit. Switching to the next sequence only takes place if the sum of this integration is greater than the value of *rDeadband* ([see sample](#)). [[▶ 530](#)] The variable is saved persistently. Preset to 0.

rYMin: lower limit of the working range of the controller. The variable is saved persistently. Preset to 0.

rYMax: upper limit of the working range of the controller. The variable is saved persistently. Preset to 100.

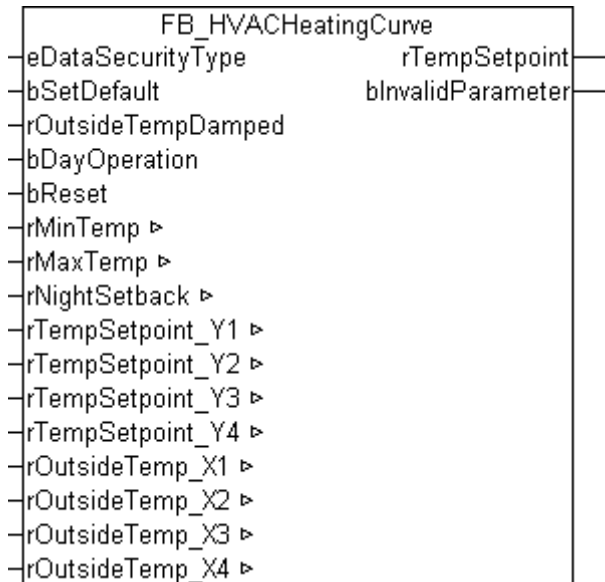
iCurrentSequence: number of the active controller in the sequence.

Documents about this

 example_persistent_e.zip (Resources/zip/11659714827.zip)

3.5 HVAC Setpoint modules

3.5.1 FB_HVACHeatingCurve



Application

The four-point heating curve serves to determine the setpoint for the flow temperature of a heating circuit, depending on the outside temperature.


For functional reasons, this function block must be used together with **FB_HVACSetpointHeating**. The reason for this is that the value for the night setback is taken into account in **FB_HVACHeatingCurve**.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
rOutsideTempDamped: REAL;
bDayOperation     : BOOL;
bReset           : BOOL;
```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block **FB_HVACPersistentDataHandling** must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

rOutsideTempDamped: this input variable transfers the current damped outside temperature to the function block.

bDayOperation: TRUE = day operation, FALSE = night operation.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
rTempSetpoint      : REAL;      0 .. 500      ° C
bInvalidParameter : BOOL;
```

rTempSetpoint: the calculated setpoint for the flow temperature.

bInvalidParameter: Indicates that an incorrect input parameter is present. *bInvalidParameter* must be acknowledged with *bReset*.

VAR_IN_OUT

```
rMinTemp           : REAL;
rMaxTemp           : REAL;
rNightSetback      : REAL;
rTempSetpoint_Y1   : REAL;
rTempSetpoint_Y2   : REAL;
rTempSetpoint_Y3   : REAL;
rTempSetpoint_Y4   : REAL;
rOutsideTemp_X1    : REAL;
rOutsideTemp_X2    : REAL;
rOutsideTemp_X3    : REAL;
rOutsideTemp_X4    : REAL;
```

rMinTemp: the minimum value for the setpoint of the flow temperature is defined by this variable. The variable is saved persistently. Preset to 0.

rMaxTemp: the maximum value for the setpoint of the flow temperature is defined by this variable. The variable is saved persistently. Preset to 500.

rNightSetback: the value for the night setback is specified by this variable. The variable is saved persistently. Preset to 20.

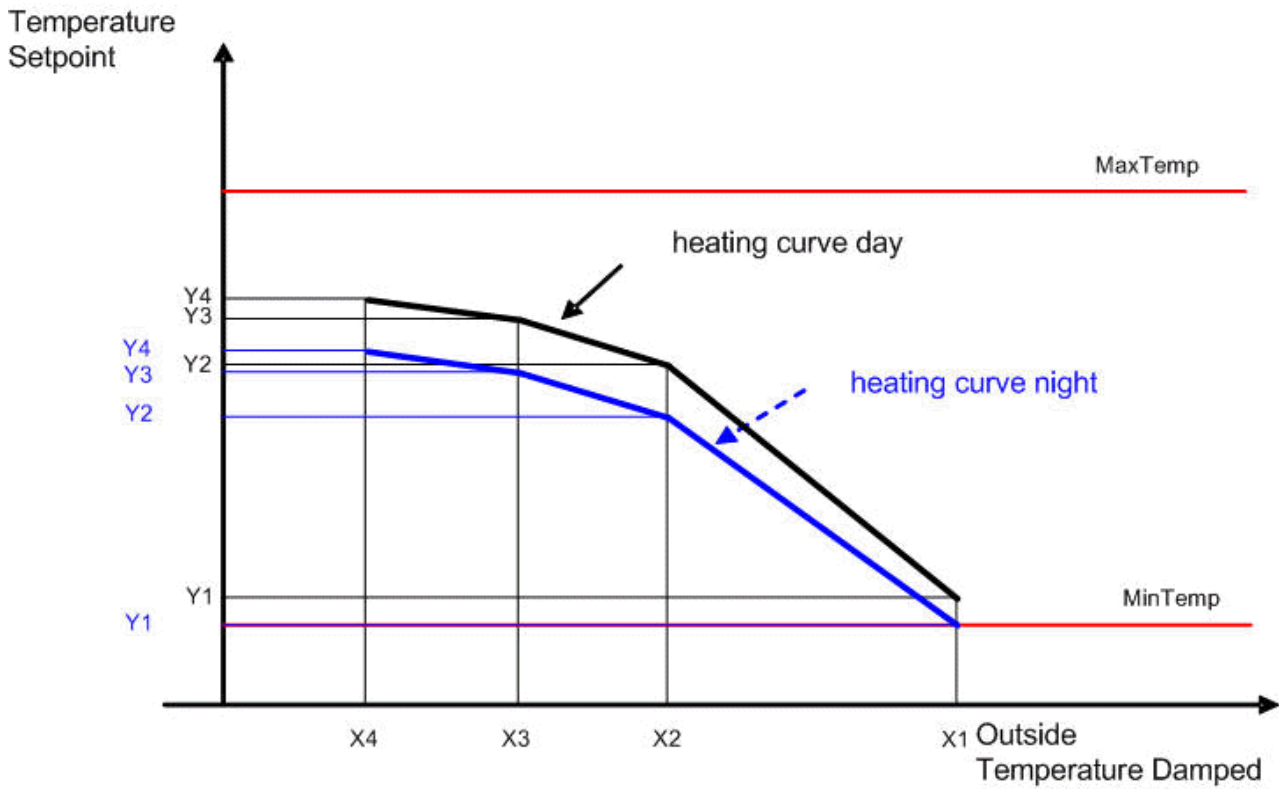
rTempSetpoint_Y1 / rOutsideTemp_X1: the course of point 1 of the heating curve is parameterized by this value pair. The variable is saved persistently. Preset to 20.

rTempSetpoint_Y2 / rOutsideTemp_X2: the course of point 2 of the heating curve is parameterized by this value pair. The variable is saved persistently. Preset to 65 and 0.

rTempSetpoint_Y3 / rOutsideTemp_X3: the course of point 3 of the heating curve is parameterized by this value pair. The variable is saved persistently. Preset to 74 and -10.

rTempSetpoint_Y4 / rOutsideTemp_X4: the course of point 4 of the heating curve is parameterized by this value pair. The variable is saved persistently. Preset to 80 and -20.

Course of the heating characteristic curves



Conditions

The following applies to the input of the values: $X1 > X2 > X3 > X4$ and $Y1 < Y2 < Y3 < Y4$.

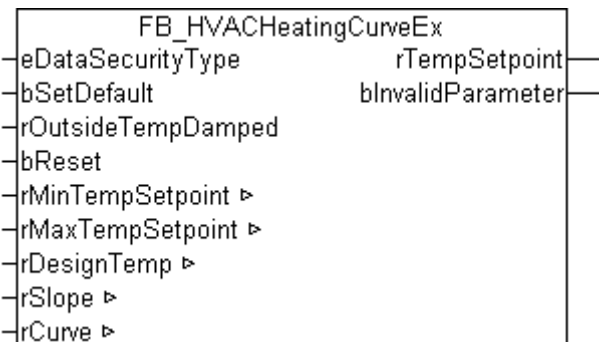
Furthermore the minimum value for the setpoint of the flow temperature must be $\leq rTempSetpoint_Y1$ and ≥ 0 . The maximum value for the setpoint of the flow temperature must be $\geq rTempSetpoint_Y4$.

If one of these conditions is not met, the variable *blInvalidParameter* will be set to TRUE and the default values of the VAR_IN_OUT variables will be adopted.

Documents about this

- example_persistent_e.zip (Resources/zip/11659714827.zip)

3.5.2 FB_HVACHeatingCurveEx



Application

The heating curve serves to determine the setpoint for the flow temperature of a heating circuit, depending on the outside temperature. Compared with the FB_HVACHeatingCurve, the heating curve is shown as a function.

$$a = rSlope * e^{(b * \ln(z))}$$

$$b = 1/\ln(4) * \ln(1+rCurve) \quad z, u < 0 \implies z, u = 0.0001$$

$$rTempSetpoint = 20 + a * e^{((1-b) * \ln(u))}$$

$$z = 20 - rDesignTemp;$$


$$u = 20 - rOutsideTemp;$$

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
rOutsideTempDamped : REAL;
bDayOperation     : BOOL;
bReset           : BOOL;
```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

rOutsideTempDamped: this input variable transfers the current damped outside temperature to the function block.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
rTempSetpoint : REAL;           ° C
bInvalidParameter : BOOL;
```

rTempSetpoint: the calculated setpoint for the flow temperature.

bInvalidParameter: indicates that an incorrect input parameter is present. *bInvalidParameter* must be acknowledged with *bReset*.

VAR_IN_OUT

```

rMinTempSetpoint : REAL;
rMaxTempSetpoint : REAL;
rDesignTemp      : REAL;
rSlope           : REAL;
rCurve           : REAL;

```

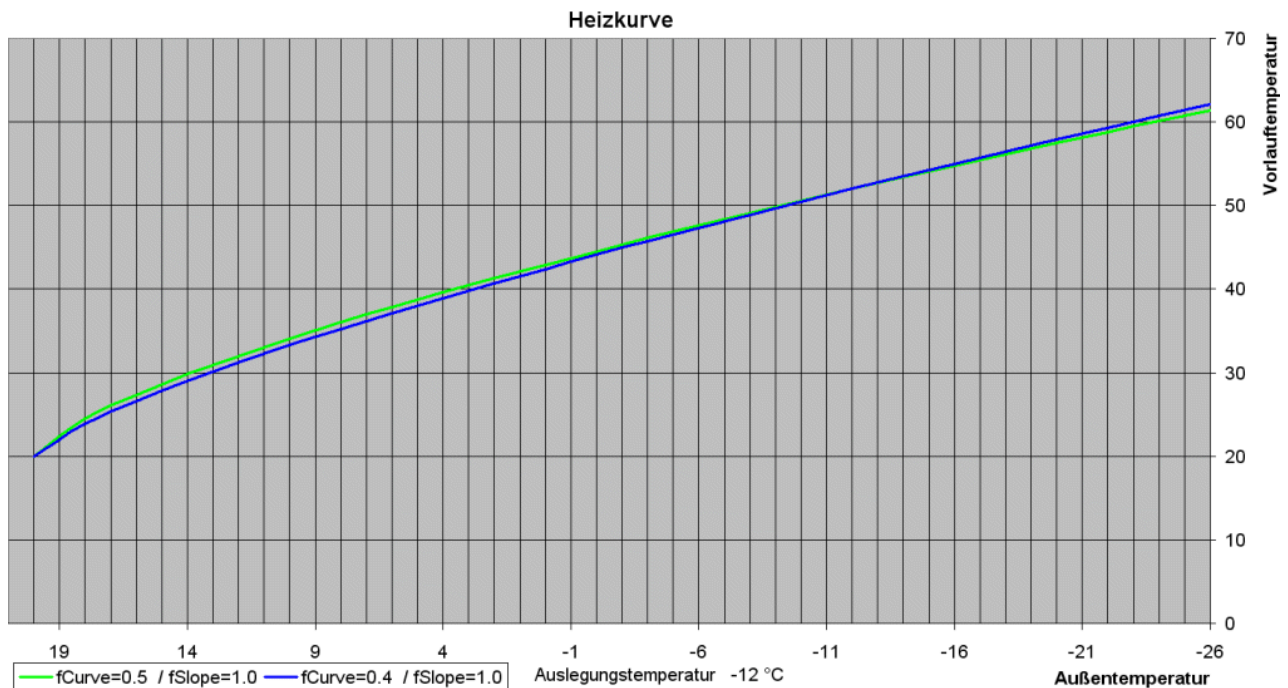
rMinTempSetpoint: the minimum value for the setpoint of the flow temperature [°C] is defined by this variable. The variable is saved persistently. Preset to 20.

rMaxTempSetpoint: the maximum value for the setpoint of the flow temperature [°C] is defined by this variable. The variable is saved persistently. Preset to 90.

rDesignTemp: design temperature [°C] for the dimensioning of a heating system. Typical values for Germany lie between -12 °C and -16 °C. The variable is saved persistently. Preset to -16.

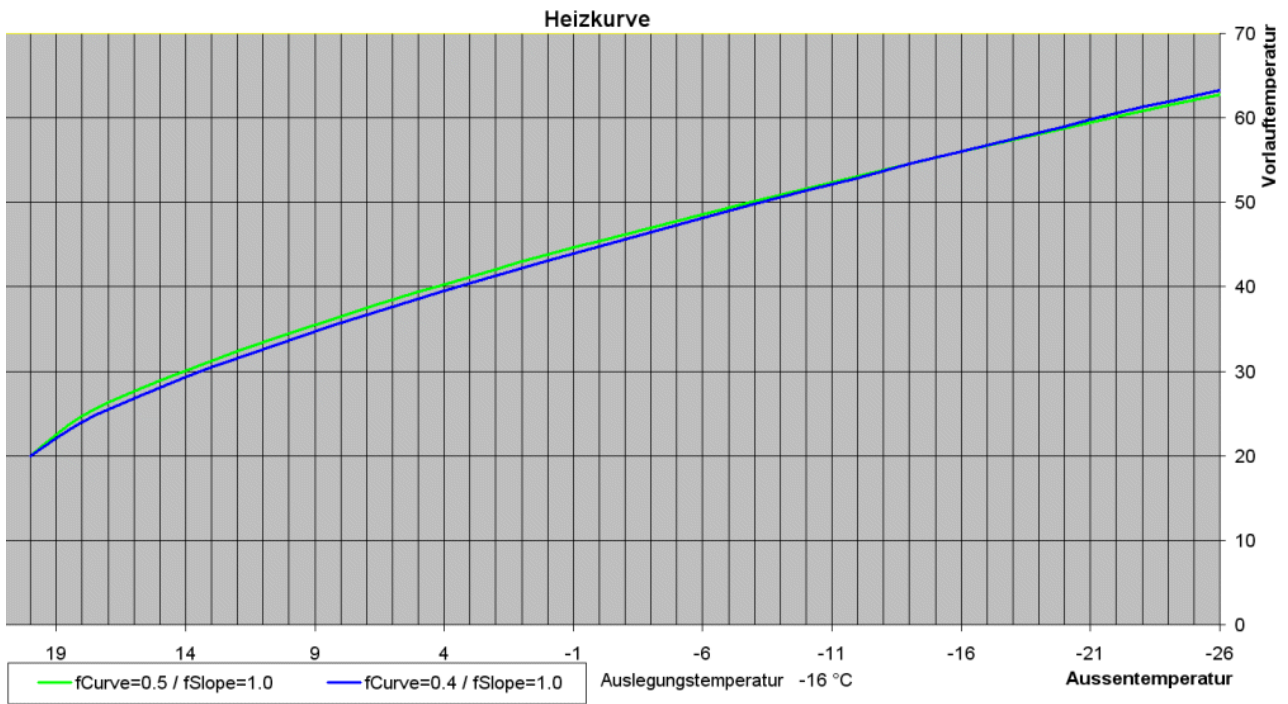
rSlope: factor for the slope. The variable is saved persistently. Preset to 1.

rCurve: factor for the curve. The variable is saved persistently. Preset to 0.5.

Course of the heating characteristic curves

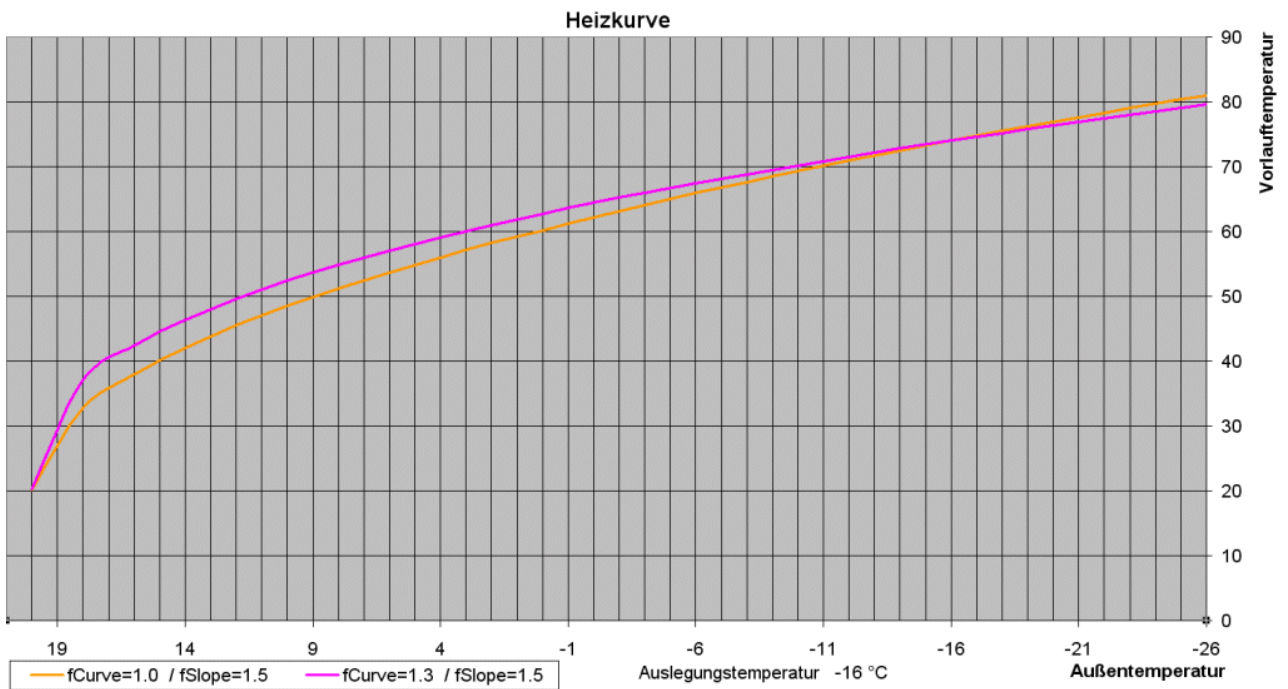
Flow temperature of 50 °C at approx. -9.2 °C

Flow temperature of 50 °C at approx. -9.4 °C



Flow temperature of 50 °C at approx. -7.8 °C

Flow temperature of 50 °C at approx. -8.3 °C



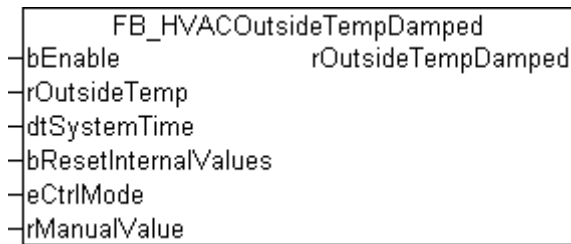
Flow temperature of 60 °C at approx. 0.2 °C

Flow temperature of 60 °C at approx. 3.0 °C

Documents about this

- example_persistent_e.zip (Resources/zip/11659714827.zip)

3.5.3 FB_HVACOutsideTempDamped



Application

This function block serves to determine the average or damped outside temperature. In automatic mode the mean outside temperature is calculated from the outside temperature values at 7:00, 14:00 and 21:00. The measurement taken at 21:00 receives double the weighting.

VAR_INPUT

```
bEnable           : BOOL;
rOutsideTemp      : REAL;
dtSystemTime      : DT;
bResetInternalValue : BOOL;
eCtrlMode         : E_HVACCtrlMode;
rManualValue      : REAL;
```

bEnable: the function block is enabled by the PLC program with the input variable *bEnable*. If *bEnable* = FALSE, the last valid value of the damped outside temperature is output.

rOutsideTemp: this input variable transfers the current outside temperature to the function block.

dtSystemTime: this input variable transfers the date and time to the function block.

bResetInternalValues: using this input variable, the internally stored outside temperatures are reset and the current outside temperature is then adopted.

eCtrlMode: the operation mode is selected via this enum. Manual or automatic operation mode.

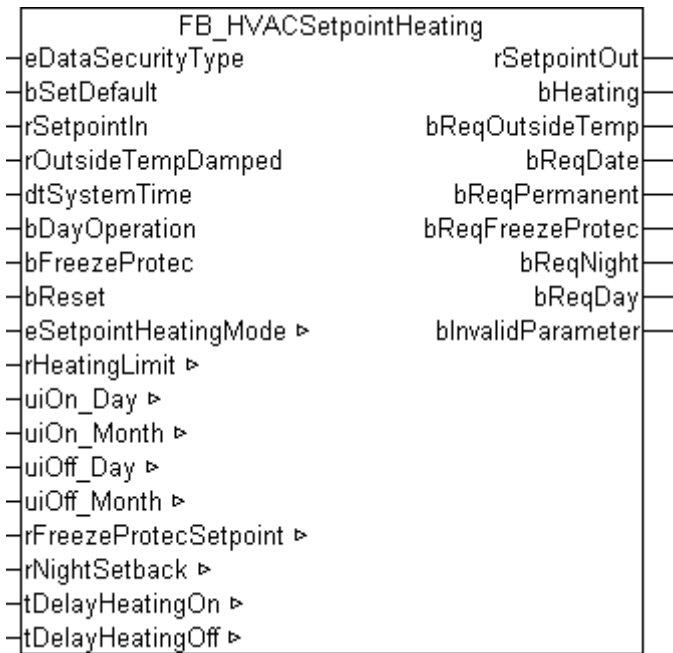
rManualValue: manual value that is set at the output *rOutsideTempDamped* if *eCtrlMode* = *eHVACCtrlMode_Manual*.

VAR_OUTPUT

```
rOutsideTempDamped : REAL;
```

rOutsideTempDamped: damped outside temperature.

3.5.4 FB_HVACSetpointHeating



Application

Using this function block a heating circuit can be switched through different operation modes. Depending on the operation mode a corresponding setpoint for controlling the inlet temperature of a static heating circuit is output from the function block. The output *bHeating* is set or reset depending on the operation mode and the state of the parameters described below.


For functional reasons, this function block must be used together with **FB_HVACHeatingCurve**. The reason for this is that the value for the night setback is taken into account in **FB_HVACHeatingCurve**.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
rSetpointIn       : REAL;
rOutsideTempDamped : REAL;
dtSystemTime      : DT;
bDayOperation     : BOOL;
bFreeze Protec    : BOOL;
bReset            : BOOL;
```

eDataSecurityType: if eDataSecurityType:= *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If eDataSecurityType:= *eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is activated if the variable `bEnable` is TRUE. If it is FALSE, zero will be output at `rSetpointOut` and the pump output remains FALSE.

rSetpointIn: in the case of a heating circuit controlled by the room temperature, the value of a room setpoint module is applied at `rSetpointIn`. If the heating circuit is controlled by the outside temperature, the setpoint comes from the heating characteristic curve. See [FB_HVACHeatingCurve](#) [► 326].

rOutsideTempDamped: short-term fluctuations in the outside temperature must not be allowed to have an unfiltered effect on the setpoint of the flow temperature. For this reason the outside temperature must not be connected directly, but rather via a damping function block. See also [FB_HVACOutsideTempDamped](#) [► 332] regarding this point.

dtSystemtime: this variable transfers the computer system time to the function block.

bDayOperation: the output variable of the timer program is transferred at this input variable. If `bDayOperation` is TRUE, then the heating circuit is in day operation mode. The night setback of the flow temperature is hence deactivated.

bFreezeProtec: input to which the frost protection signal is applied.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
rSetpointOut      : REAL;
bHeating          : BOOL;
bReqOutsideTemp  : BOOL;
bReqDate         : BOOL;
bReqPermanent    : BOOL;
bReqFreezeProtec : BOOL;
bReqNight        : BOOL;
bReqDay          : BOOL;
bInvalidParameter : BOOL;
```

rTempSetpoint: setpoint for the flow temperature control.

bHeating: the output `bHeating` is set (TRUE) or reset (FALSE) immediately or after a delay, depending on the selected operating mode as well as the switch-on and switch-off delays. The output can be used to enable the controller.

bReqOutsideTemp: the heating circuit is in the operation mode 'heating period according to outside temperature'.

bReqDate: the heating circuit is in the operation mode 'heating period according to date'.

bReqPermanent: the heating circuit is in the operation mode 'heating circuit always on'.

bReqFreezeProtec: the heating circuit is in frost protection mode.

bReqNight: the heating circuit is in the night operation mode.

bReqDay: the heating circuit is in the day operation mode.

bInvalidParameter: Indicates that an incorrect input parameter is present. `bInvalidParameter` must be acknowledged with `bReset`.

VAR_IN_OUT

```
eSetpointHeatingMode : E_HVACSetpointHeatingMode;
rHeatingLimit        : REAL;
uiOn_Day             : UINT;
uiOn_Month           : UINT;
uiOff_Day            : UINT;
```

```

uiOff_Month      : UINT;
rFreeze ProtecSetpoint : REAL;
rNightSetback   : REAL;
tDelayHeatingOn  : TIME;
tDelayHeatingOff : TIME;
    
```

eSetpointHeatingMode: Enum that specifies the operation mode of the heating circuit. The variable is saved persistently. Preset to 1.

rHeatingLimit: heating limit (-60°C..100°C). Necessary for the operation modes of the heating circuit; heating period according to the outside temperature and heating period according to the date. The variable is saved persistently. Preset to 19.

uiOn_Day: switch-on day of the month. Necessary for the operation mode 'heating period according to date'. The variable is saved persistently. Preset to 1.

uiOn_Month: switch-on month of the year. Necessary for the operation mode 'heating period according to date'. The variable is saved persistently. Preset to 9.

uiOff_Day: switch-off day of the month. Necessary for the operation mode 'heating period according to date'. The variable is saved persistently. Preset to 1.

uiOff_Month: switch-off month of the year. Necessary for the operation mode 'heating period according to date'. The variable is saved persistently. Preset to 5.

rFreeze ProtecSetpoint: setpoint for the heating circuit during frost protection (0°C..100°C). The variable is saved persistently. Preset to 8.

rNightSetback: with *rNightSetback* the amount of night setback in °C is specified (0°C..100°C). *rNightSetback* is considered in the operation mode *eHVACSetpointHeatingMode_OnNight*. The variable is saved persistently. Preset to 10.

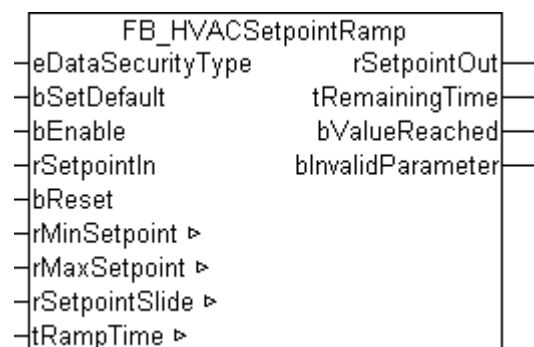
tDelayHeatingOn: time delay for the setting of the output *bHeating* to TRUE after the damped outside temperature has undershot the heating limit or the date has been reached (0h..100h). The variable is saved persistently. Preset to 0.

tDelayHeatingOff: time delay for the resetting of the output *bHeating* to FALSE after the damped outside temperature has exceeded the heating limit or the date has been reached (0h..100h). The variable is saved persistently. Preset to 0.

Documents about this

 [example_persistent_e.zip \(Resources/zip/11659714827.zip\)](#)

3.5.5 FB_HVACSetpointRamp



Application


The function block generates a sliding setpoint ramp. This function block must be called in every PLC cycle, because the calculation for reaching the final setpoint depends on the TaskTime.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
rSetpointIn       : REAL;
bReset            : BOOL;
```

eDataSecurityType: if `eDataSecurityType:= eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: enable of the function block.

rSetpointIn: setpoint to which the output `rSetpointOut` is regulated, depending on the absolute temperature change `rSetpointSlide` per time change `tRampTime`. Each change of the setpoint starts a new calculation of the setpoint to the target value of the ramp. The target value is the current value of `rSetpointIn`. The start value of the ramp is the last value before changing `rSetpointIn` to the new target value, see Figure 1.1. The parameters `rMaxSetpoint` and `rMinSetpoint` define the value range of `rSetpointIn`. If an incorrect variable value is present at `rSetpointIn`, then the last valid variable value is taken, if available. If there is no valid, last value, then work continues internally with `rMinSetpoint`. `blInvalidParameter` is set if the variable value is incorrect, the function block continues to operate normally.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
rSetpointOut      : REAL;
tRemainingTime    : TIME;
bValueReached     : BOOL;
blInvalidParameter : BOOL;
```

rSetpointOut: output of the ramp generator

tRemainingTime: time duration in which the output `rSetpointOut` has reached the target value `rSetpointIn`.

bValueReached: a TRUE at this output indicates that the output `rSetpointOut` has reached the target value `rSetpointIn`.

blInvalidParameter: indicates that an incorrect parameter is present at one of the variables `rSetpointSlide` or `tRampTime`. An incorrect parameter entry does not lead to a standstill of the function block; see description of variables. After rectifying the incorrect parameter entry, the message `blInvalidParameter` must be acknowledged via `bReset`.

VAR_IN_OUT

```
rMinSetpoint      : REAL;
rMaxSetpoint      : REAL;
rSetpointSlide    : REAL;
tRampTime         : TIME;
```


rMinSetpoint/rMaxSetpoint: the parameters *rMaxSetpoint* and *rMinSetpoint* define the value range (-10000..10000) of *rSetpointIn*. *rMaxSetpoint* must be greater than *rMinSetpoint*.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *blInvalidParameter* will be set in the event of an incorrect parameter entry. The function block continues to operate normally. The variable is saved persistently. Preset to 10 or 40 respectively.

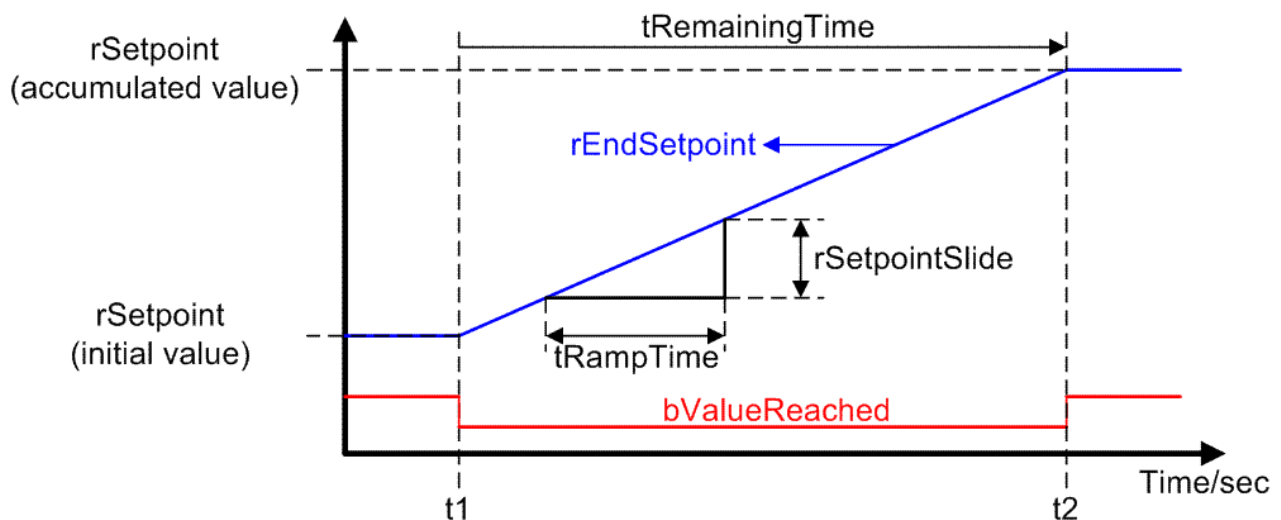
rSetpointSlide: absolute temperature change in [Kelvin / *tRampTime*] with which the output is converted in sliding steps from a lower to a higher value or from a higher to a lower value, see fig. 1.1

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *blInvalidParameter* will be set in the event of an incorrect parameter entry. The function block continues to operate normally. The variable is saved persistently. Preset to 1.

tRampTime: ramp time (1s..24h) during which the final setpoint is reached in sliding steps, depending on the absolute temperature change *rSetpointSlide*, see fig. 1.1

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *blInvalidParameter* will be set in the event of an incorrect parameter entry. The function block continues to operate normally. The variable is saved persistently. Preset to 3600 s.

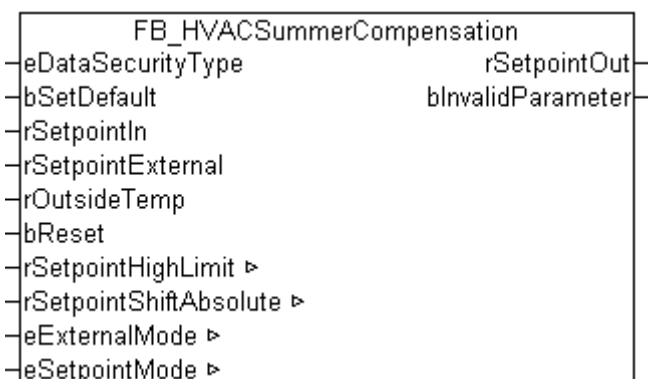
Fig. 1.1:



Documents about this

- example_persistent_e.zip (Resources/zip/11659714827.zip)

3.5.6 FB_HVACSummerCompensation



Application

This function block is a set value generator for the exhaust air temperature or room temperature of an air conditioning system. With the aid of this summer compensation, overly large temperature differences between the outside temperature and the exhaust air/room temperature are avoided. This helps to save energy whilst at the same time increasing comfort.

If the outside temperature increases to the point where the difference between it and the room temperature is greater than the set difference of *rSetpointShiftAbsolute*, then the room temperature setpoint (*rSetpointOut*) is raised. The permissible difference can be set from 0 to 10 Kelvin.

A check is performed in the function block that the maximum room temperature setpoint is not exceeded if *eSetpointMode:= eHVACSetpointMode_DINLimited*.

Example I:

Room setpoint (*rSetpointIn*):= 21 °C

external setpoint (*rSetpointExternal*):= 2K (e.g. +/- 3K can be set via a potentiometer, but are not taken into account because *eExternalMode:= eHVACExternalMode_Off*)

set differential value between outside temperature and room temperature (*rSetpointShiftAbsolute*):= 6 K

Outside temperature (*rOutsideTemp*):= 28°C

eExternalMode:= eHVACExternalMode_Off

eSetpointMode:= eHVACSetpointMode_DIN

calculated room temperature setpoint (*rSetpointOut*):= 22

Example II:

Room setpoint (*rSetpointIn*):= 20 °C

external setpoint (*rSetpointExternal*):= -2K (e.g. +/- 3K can be set via a potentiometer)

set differential value between outside temperature and room temperature (*rSetpointShiftAbsolute*):= 6 K

Outside temperature (*rOutsideTemp*):= 28°C

eExternalMode:= eHVACExternalMode_On

eSetpointMode:= eHVACSetpointMode_DIN

calculated room temperature setpoint (*rSetpointOut*):= 20

Example III:

Room setpoint (*rSetpointIn*):= 23 °C

external setpoint (*rSetpointExternal*):= 3K (e.g. +/- 3K can be set via a potentiometer)

set differential value between outside temperature and room temperature (*rSetpointShiftAbsolute*):= 6 K

Outside temperature (*rOutsideTemp*):= 34°C

rSetpointHighLimit:= 35°C

eExternalMode:= eHVACExternalMode_On

eSetpointMode:= eHVACSetpointMode_DINLimited

calculated room temperature setpoint (*rSetpointOut*):= 31

Example IV:

Room setpoint (*rSetpointIn*):= 24 °C

external setpoint (*rSetpointExternal*):= 3K (e.g. +/- 3K can be set via a potentiometer)

set differential value between outside temperature and room temperature (*rSetpointShiftAbsolute*):= 6 K

Outside temperature (*rOutsideTemp*):= 36°C

rSetpointHighLimit:= 30°C

eExternalMode:= *eHVACExternalMode_On*

eSetpointMode:= *eHVACSetpointMode_DINLimited*


calculated room temperature setpoint := 31 °C, but is limited to 30 °C on account of *rSetpointHighLimit* being set to 30 °C ==> *rSetpointOut*:=30 °C

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
rSetpointIn       : REAL;
rSetpointExternal : REAL;
rOutsideTemp      : REAL;
bReset            : BOOL;
```

eDataSecurityType: if *eDataSecurityType*:= *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If *eDataSecurityType*:= *eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if *eDataSecurityType*:= *eHVACDataSecurityType_Persistent*. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

rSetpointIn: the setpoint for the room is applied to this input.

rSetpointExternal: the setpoint adjustment or correction, e.g. from a potentiometer, is applied to this input.

rOutsideTemp: input for the outside temperature.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
rSetpointOut : REAL;
bInvalidParameter : BOOL;
```

rSetpointOut: calculated room temperature setpoint.

bInvalidParameter: Indicates that an incorrect input parameter is present. *bInvalidParameter* must be acknowledged with *bReset*.

VAR_IN_OUT

```
rSetpointHighLimit : REAL;
rSetpointShiftAbsolute : REAL;
eExternalMode : E_HVACExternalMode;
eSetpointMode : E_HVACSetpointMode;
```

rSetpointHighLimit: upper limit for the room temperature setpoint (0°C..100°C). The variable is saved persistently. Preset to 35.

rSetpointShiftAbsolute: parameter value that defines the permissible difference between the outside temperature and the room temperature. The permissible difference can be set from 0 to 10 Kelvin. If the difference between the outside temperature and the room temperature is greater than the set parameter value, the room temperature setpoint is raised. The variable is saved persistently. Preset to 6.

eExternalMode: the ENUM E_HVACExternalMode is used to activate/deactivate the external setpoint specification.

eHVACExternalMode_Off, corresponds to the external setpoint specification via potentiometer in the panel being deactivated.

eHVACExternalMode_On, corresponds to the external setpoint specification being activated, e.g. +/- 3 °C

eHVACExternalMode_ShiftAbsolut, corresponds to external setpoint specification absolute in °C

eSetpointMode: Enum that specifies the type of setpoint determination.

This increase of the setpoint in the case of high outside temperatures can be unlimited or limited according to DIN. Selection takes place with the ENUM E_HVACSetpointMode. Besides these two modes for summer compensation it is also possible to specify a fixed setpoint.

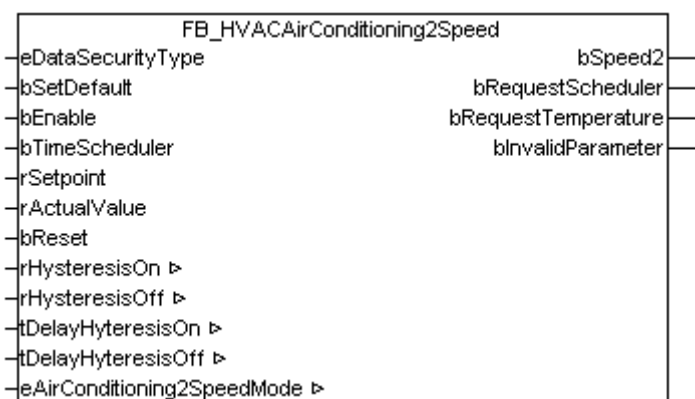
1. According to DIN (*eHVACSetpointMode_DIN*). The room temperature setpoint follows the outside temperature, guided by a difference.
2. Limited (*eHVACSetpointMode_DINLimited*). The room temperature setpoint follows the outside temperature, guided by a difference. However, it is limited. The limit value is *rSetpointHighLimit*.
3. Constant (*eHVACSetpointMode_ConstantValueBase*). The room temperature setpoint is specified as a constant value; the outside temperature does not exert any influence.

Documents about this

- example_persistent_e.zip (Resources/zip/11659714827.zip)

3.6 HVAC Special functions

3.6.1 FB_HVACAirConditioning2Speed



Application


The function block *FB_HVACAirConditioning2Speed* controls switching to the second stage of air conditioning systems with two-speed fans. Switching to the second stage via the output variable *bSpeed2* can be performed either by a schedule via the input variable *bTimeScheduler* or load-dependent via a difference between the setpoint of the room or exhaust temperature, *rSetpoint*, and the actual value of the room or exhaust temperature, *rActualValue*, depending on the time delays *tDelayHysteresisOn*/*tDelayHysteresisOff*, the switch-on and switch-off hystereses *rHysteresisOn*/*rHysteresisOff* and the setting of the operation mode via the Enum *eAirConditioning2SpeedMode*.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
bTimeScheduler    : BOOL;
rSetpoint         : REAL;
rActualValue      : REAL;
bReset            : BOOL;
```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled by the PLC program with the input variable `bEnable = TRUE`. The output variable `bSpeed2` is FALSE if the function block is not enabled.

bTimeScheduler: request by a timer program for the second fan stage. If `bEnable` and `bTimeScheduler` are TRUE, then `bSpeed2` and `bRequestScheduler` are also TRUE.

rSetpoint: setpoint of room or exhaust air temperature

rActualValue: actual value of room or exhaust air temperature

bReset: input for acknowledging an incorrect parameter entry or an error via a rising edge.

VAR_OUTPUT

```
bSpeed2           : BOOL;
bRequestScheduler : BOOL;
bRequestTemperature : BOOL;
bInvalidParameter : BOOL;
```

bSpeed2: output for controlling the second fan stage.

bRequestScheduler: request for the second fan stage via the input variable `bTimeScheduler`.

bRequestTemperature: request for the second stage, depending on the load, via a difference between the setpoint of the room or exhaust air temperature `rSetpoint` and the actual value of the room or exhaust air temperature `rActualValue` depending on the time delays `tDelayHysteresisOn/tDelayHysteresisOff`, the switch-on and switch-off hystereses `rHysteresisOn/rHysteresisOff` and the setting of the mode via the Enum `eAirConditioning2SpeedMode`.

bInvalidParameter: indicates that an incorrect parameter is present at one of the variables `rHysteresisOn`, `rHysteresisOff` and `eAirConditioning2SpeedMode`. An incorrect parameter entry does not lead to a standstill of the function block; see description of variables. After rectifying the incorrect parameter entry, the message `bInvalidParameter` must be acknowledged via `bReset`.

VAR_IN_OUT

```

rHysteresisOn      : REAL;
rHysteresisOff    : REAL;
tDelayHysteresisOn : TIME;
tDelayHysteresisOff : TIME;
eAirConditioning2SpeedMode : E_HVACAirConditioning2SpeedMode;

```

rHysteresisOn: to switch on the second fan stage via a hysteresis loop, load-dependent, via the output variable *bSpeed2* depending on the enum *eAirConditioning2SpeedMode* (0..1000). *rHysteresisOn* must be greater than *rHysteresisOff*, if the operation mode is *eAirConditioning2SpeedMode=eHVACAirConditioning2SpeedModeHeatingAndCooling*.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *blInvalidParameter* will be set in the event of an incorrect parameter entry. The function block continues to operate normally. The variable is saved persistently. Preset to 5.

rHysteresisOff: parameter to switch off the second fan stage via a hysteresis loop, load-dependent, via the output variable *bSpeed2* depending on the enum *eAirConditioning2SpeedMode* (0..1000).

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *blInvalidParameter* will be set in the event of an incorrect parameter entry. The function block continues to operate normally. The variable is saved persistently. Preset to 1.

tDelayHysteresisOn: switch-on delay of the output variable *bSpeed2* in the event of a load-dependent request for the second fan stage. The variable is saved persistently. Preset to 300 s.

tDelayHysteresisOff: switch-off delay of the output variable *bSpeed2* in the event of a load-dependent request for the second fan stage. The variable is saved persistently. Preset to 300 s.

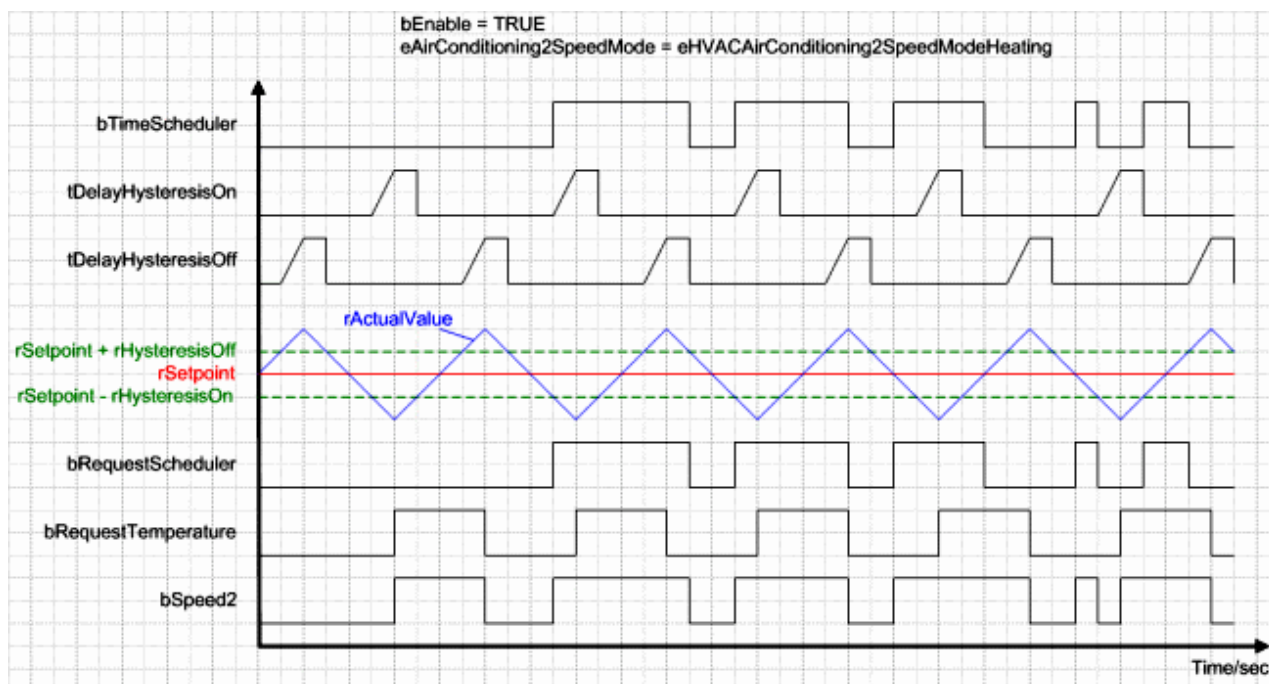
eAirConditioning2SpeedMode: Enum used to specify the load-dependent operation mode for switching on the second fan stage via the output variable *bSpeed2*.

eAirConditioning2SpeedMode = eHVACAirConditioning2SpeedMode_Off: operation mode off

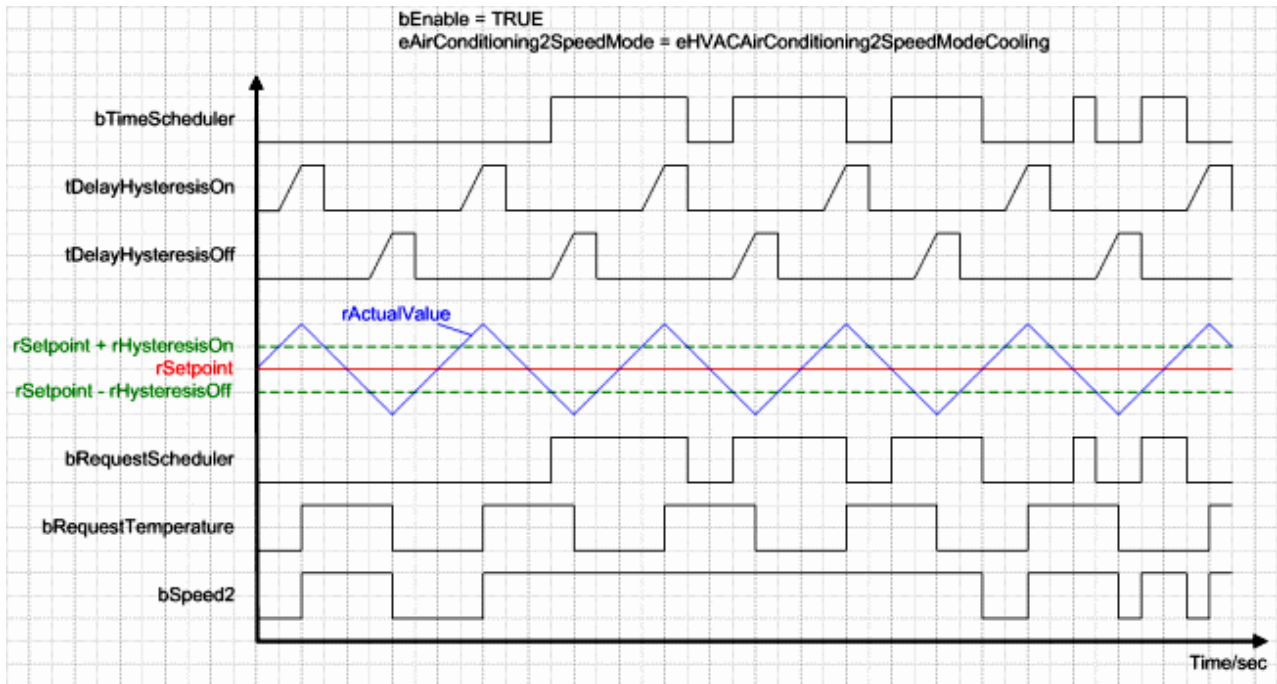
eAirConditioning2SpeedMode = eHVACAirConditioning2SpeedMode_Heating: heating operation mode. This is an air conditioning system with heating operation only.

eAirConditioning2SpeedMode = eHVACAirConditioning2SpeedMode_Cooling: cooling operation mode. This is an air conditioning system with cooling operation only.

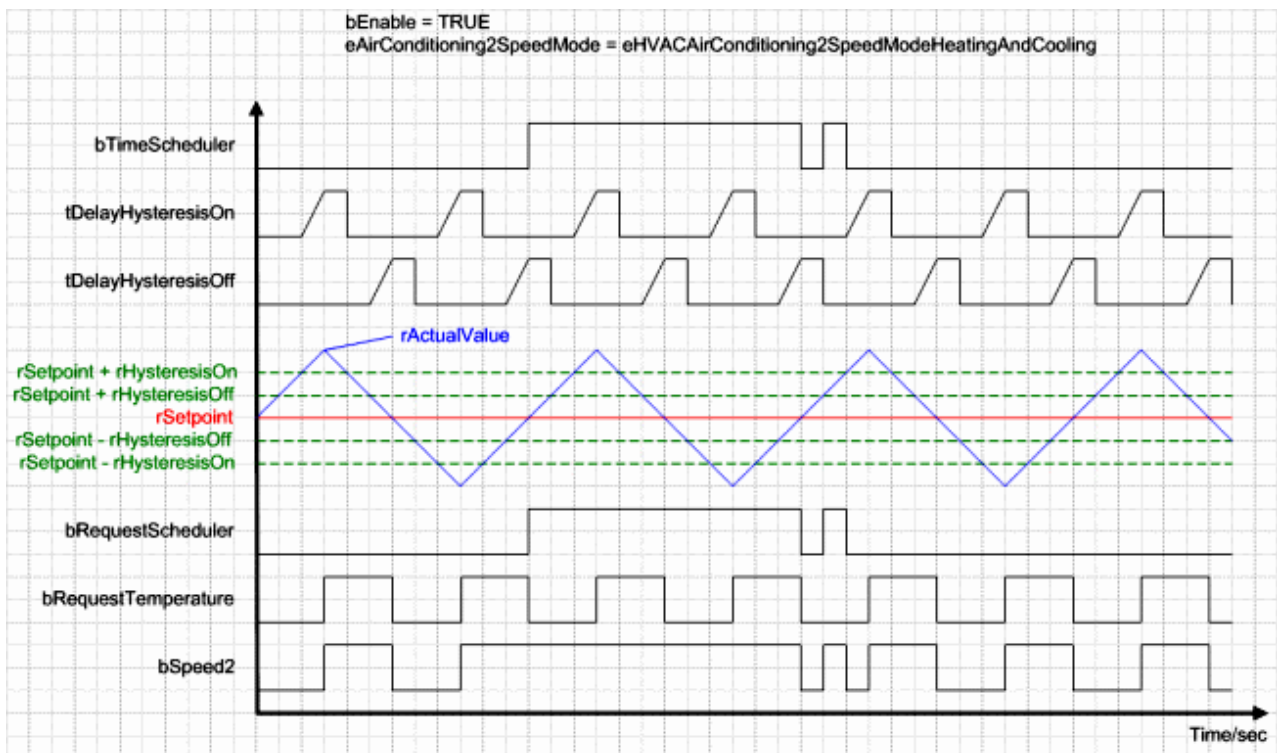
eAirConditioning2SpeedMode = eHVACAirConditioning2SpeedMode_HeatingAndCooling: heating and cooling operation mode. This is an air conditioning system with heating and cooling operation. The variable is saved persistently. Preset to 1.

Operation mode *eAirConditioning2SpeedMode = eHVACAirConditioning2SpeedMode_Heating*

Operation mode eAirConditioning2SpeedMode = eHVACAirConditioning2SpeedMode_Cooling



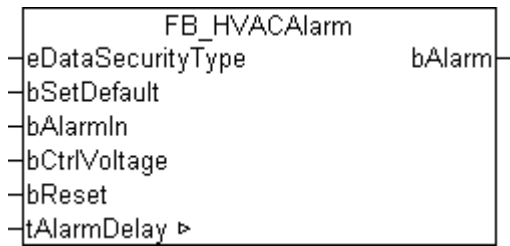
Operation mode eAirConditioning2SpeedMode = eHVACAirConditioning2SpeedMode_HeatingAndCooling



Documents about this

- example_persistent_e.zip (Resources/zip/11659714827.zip)
- fb_hvacairconditioning2speeddrawingheating.gif (Resources/gif/11659721995.gif)
- fb_hvacairconditioning2speeddrawingcooling.gif (Resources/gif/11659723403.gif)
- fb_hvacairconditioning2speeddrawingheatingandcooling.gif (Resources/gif/11659724811.gif)

3.6.2 FB_HVACAlarm



Application


A TRUE on the input *bAlarmIn* indicates that an alarm signal is present. The alarm is passed on to the output *bAlarm* only if *bAlarmIn* is present for longer than the preset time *tAlarmDelay*. Furthermore, the input *bCtrlVoltage* must be TRUE in order for an alarm to be announced. Output *bAlarm* remains set to TRUE until *bAlarmIn* = FALSE and the alarm is acknowledged by a positive edge on *bReset*.

VAR_INPUT

```
eDataSecurityType      : E_HVACDataSecurityType;
bSetDefault            : BOOL;
bAlarmIn               : BOOL;
bCtrlVoltage           : BOOL;
bReset                 : BOOL;
```

eDataSecurityType: if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If *eDataSecurityType* := *eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bAlarmIn: a TRUE on the input *bAlarmIn* indicates that an alarm signal is present.

bCtrlVoltage: a check via the input *bCtrlVoltage* is made of whether control voltage is present.

bReset: acknowledge input.

VAR_OUTPUT

```
bAlarm                : BOOL;
```

bAlarm: TRUE if an alarm is announced.

VAR_IN_OUT

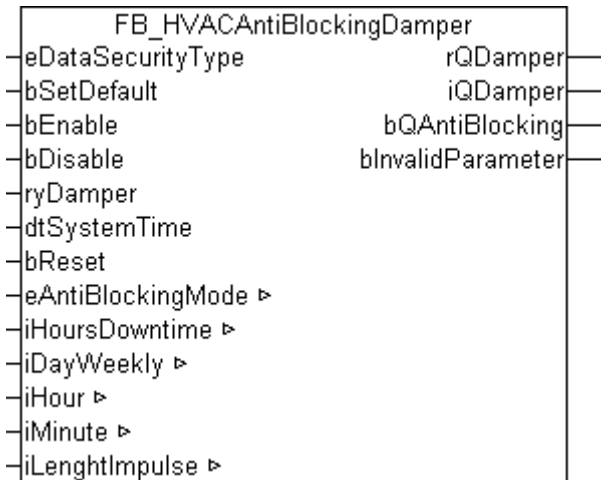
```
tAlarmDelay           : TIME;
```

tAlarmDelay: delay after the alarm is first signaled if, during the preset time, *bAlarmIn* was constantly present (0s..500s). The variable is saved persistently. Preset to 500s..

Documents about this

 example_persistent_e.zip (Resources/zip/11659714827.zip)

3.6.3 FB_HVACAntiBlockingDamper



Application


This function block prevents the blocking of an actuator over long time intervals with an unchanged control value. When activated, the blocking protection forces the actuator to drive from fully closed to fully open.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable          : BOOL;
bDisable         : BOOL;
ryDamper         : REAL;           0 .. 100   %
dtSystemTime     : DT;
bReset           : BOOL;
```

eDataSecurityType: if eDataSecurityType:= *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235.zip>

If eDataSecurityType:= *eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType:= *eHVACDataSecurityType_Persistent*. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function *AntiBlocking* is enabled with the input variable *bEnable*. If *bEnable* = FALSE, only the input value will be applied at the output; the *AntiBlocking* function is not active.

bDisable: resets the outputs *rQDamper*, *iQDamper* and *bQAntiBlocking* to 0.

ryDamper: control value from the controller to the valve, which is passed on to the outputs *rQDamper* and *iQDamper*. The value must be smaller than 1 % in order for the function block to recognize this as standstill time. No standstill time will be determined for a valve with a control value ≥ 1 %.

dtSystemTime: system time

bReset: acknowledge input in the event of a fault. Resets the flag *blInvalidParameter*.

VAR_IN_OUT

```
eAntiBlockingMode : E_HVACAntiBlockingMode;
iHoursDowntime    : INT;
iDayWeekly        : INT;
iHour             : INT;
iMinute          : INT;
iLengthImpulse   : INT;
```

eHVACAntiBlockingMode: enumeration value that specifies the type of *Antiblocking* method.

eHVACAntiBlockingMode_Off:= 0 : off

eHVACAntiBlockingMode_Downtime:= 1 : after the expiry of a standstill time in hours, the *Antiblocking* pulse will be passed on to the *bOut* output.

eHVACAntiBlockingMode_Weekly:= 2 : the antiblocking pulse is only generated on a certain day of the week and at a certain time, independent of how long the standstill has lasted.

The variable is saved persistently.

iHoursDowntime: for *eHVACAntiBlockingMode_Downtime*, the time in hours for which the input *bln* may not be active until the *Antiblocking* pulse is formed (0h..6000h). The variable is saved persistently. Preset to 24.

iDayWeekly: for *eHVACAntiBlockingMode_Weekly*, *Antiblocking* pulse is formed on this day. Sun=0, Mon=1, Tue=2, Wed=3, Thu=4, Fri=5, Sat=6

The variable is saved persistently. Preset to 6.

iHour: switch-on time in hours (0h..23 h). The variable is saved persistently. Preset to 12.

iMinute: switch-on time in minutes (0min..59min). The variable is saved persistently. Preset to 0.

iLengthImpulse: switch-on time in seconds (0s..600s). The variable is saved persistently. Preset to 150.

VAR_OUTPUT

```
rQDamper          : REAL;          0 .. 100    %
iQDamper          : INT;           0 .. 32767
bQAntiBlocking    : BOOL;
bInvalidParameter: BOOL;
```

rQDamper: this output corresponds to the input value of the control value. As soon as the input *bEnable* = is TRUE, the *Antiblocking* pulse is additionally applied to the output. 100% is present at the output when the *Antiblocking* state is reached.

iQDamper: this output behaves like *rQDamper*, but in order to suit an analog output it is output as an integer value from 0 - 32767, corresponding to 0 - 100%.

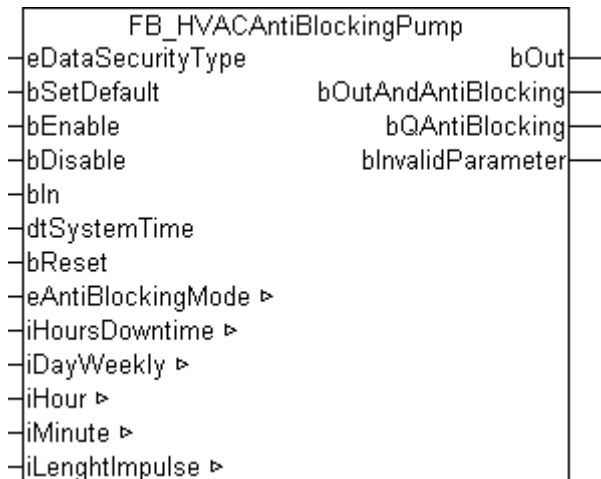
bQAntiBlocking: this output is independent of the switch-on signal and shows only the *Antiblocking pulse*.

blInvalidParameter: an error occurred during the plausibility check. It is deleted again by *bReset*.

Documents about this

📄 example_persistent_e.zip (Resources/zip/11659714827.zip)

3.6.4 FB_HVACAntiBlockingPump



Application


This function block relays the switch-on condition for a pump drive and can generate an additional switch-on pulse, depending on the mode and the switch-off time, in order e.g. to prevent the blockage of a pump.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable          : BOOL;
bDisable         : BOOL;
bIn              : BOOL;
dtSystemTime     : DT;
bReset           : BOOL;
```

eDataSecurityType: if `eDataSecurityType:= eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function *AntiBlocking* is enabled with the input variable *bEnable*. If *bEnable* = FALSE, only the switch-on condition will be applied at the output; the *AntiBlocking* function is not active.

bDisable: resets the outputs *bOut*, *bOutAndAntiBlocking* and *bQAntiBlocking* to 0. (See example below)

bIn: switch-on condition that is passed on to the output *bOut*.

dtSystemTime: system time

bReset: acknowledge input in the event of a fault. Resets the flag *blInvalidParameter*.

VAR_IN_OUT

```
eAntiBlockingMode: E_HVACAntiBlockingMode;
iHoursDowntime : INT;
iDayWeekly : INT;
iHour : INT;
iMinute : INT;
iLengthImpulse : INT;
```

eHVACAntiBlockingMode: enumeration value that specifies the type of *Antiblocking* method.

eHVACAntiBlockingMode_Off:= 0 : off

eHVACAntiBlockingMode_Downtime:= 1 : after the expiry of a standstill time in hours, the *Antiblocking* pulse will be passed on to the *bOut* output.

eHVACAntiBlockingMode_Weekly:= 2 : the antiblocking pulse is only generated on a certain day of the week and at a certain time, independent of how long the standstill has lasted.

The variable is saved persistently.

iHoursDowntime: for *eHVACAntiBlockingMode_Downtime*, the time in hours for which the input *bIn* may not be active until the *Antiblocking* pulse is formed (0h..6000h). The variable is saved persistently. Preset to 24.

iDayWeekly: for *eHVACAntiBlockingMode_Weekly*, *Antiblocking* pulse is formed on this day. Sun=0, Mon=1, Tue=2, Wed=3, Thu=4, Fri=5, Sat=6

The variable is saved persistently. Preset to 6.

iHour: switch-on time in hours (0h..23 h). The variable is saved persistently. Preset to 12.

iMinute: switch-on time in minutes (0min..23min). The variable is saved persistently. Preset to 0.

iLengthImpulse: switch-on time in seconds (0s..600s). The variable is saved persistently. Preset to 150.

VAR_OUTPUT

```
bOut : BOOL;
bOutAndAntiBlocking: BOOL;
bQAntiBlocking : BOOL;
bInvalidParameter : BOOL;
```

bOut: if TRUE, then the request comes from the pump function block. This output corresponds to the switch-on condition independent of the state of the *Antiblocking* pulse and the input *bEnable*.

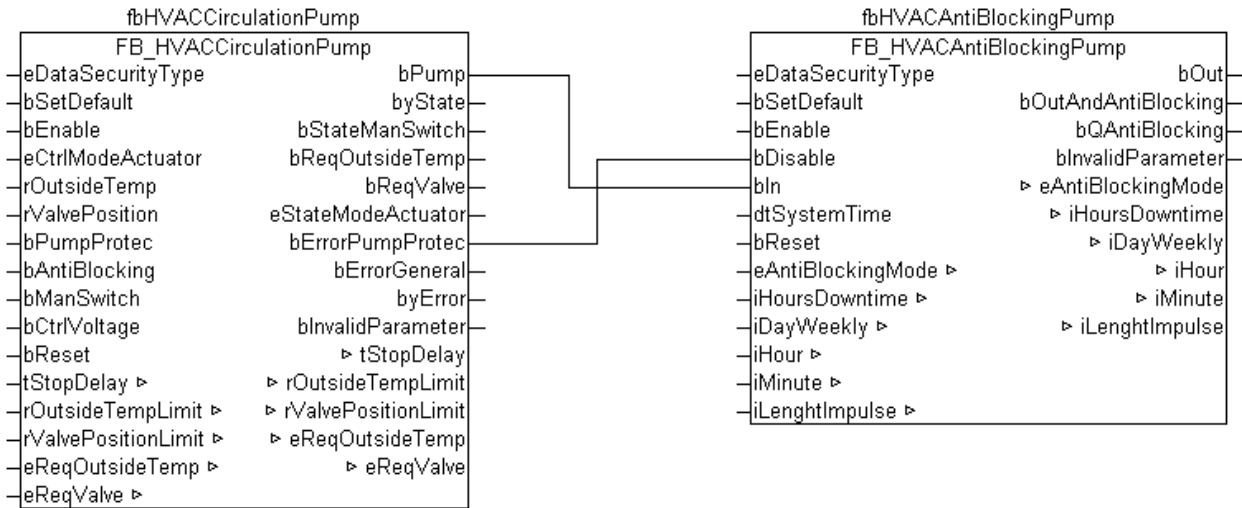
bOutAndAntiBlocking: the output can go TRUE if the request comes from the pump function block OR from the antiblocking function. The pump is connected to this output. This output corresponds to the switch-on condition independent of the state of the *Antiblocking* pulse and the inputs *bEnable* and *bDisable*. If *bEnable* = FALSE the output corresponds to the output *bOut*. If *bEnable* = TRUE, then the output corresponds to the output *bOut* and is additionally Ored° with the *Antiblocking* pulse.

bQAntiBlocking: if TRUE, then the request comes from the antiblocking function. This output is independent of the switch-on signal and shows only the *Antiblocking* pulse.

bInvalidParameter: an error occurred during the plausibility check. It is deleted again by *bReset*.

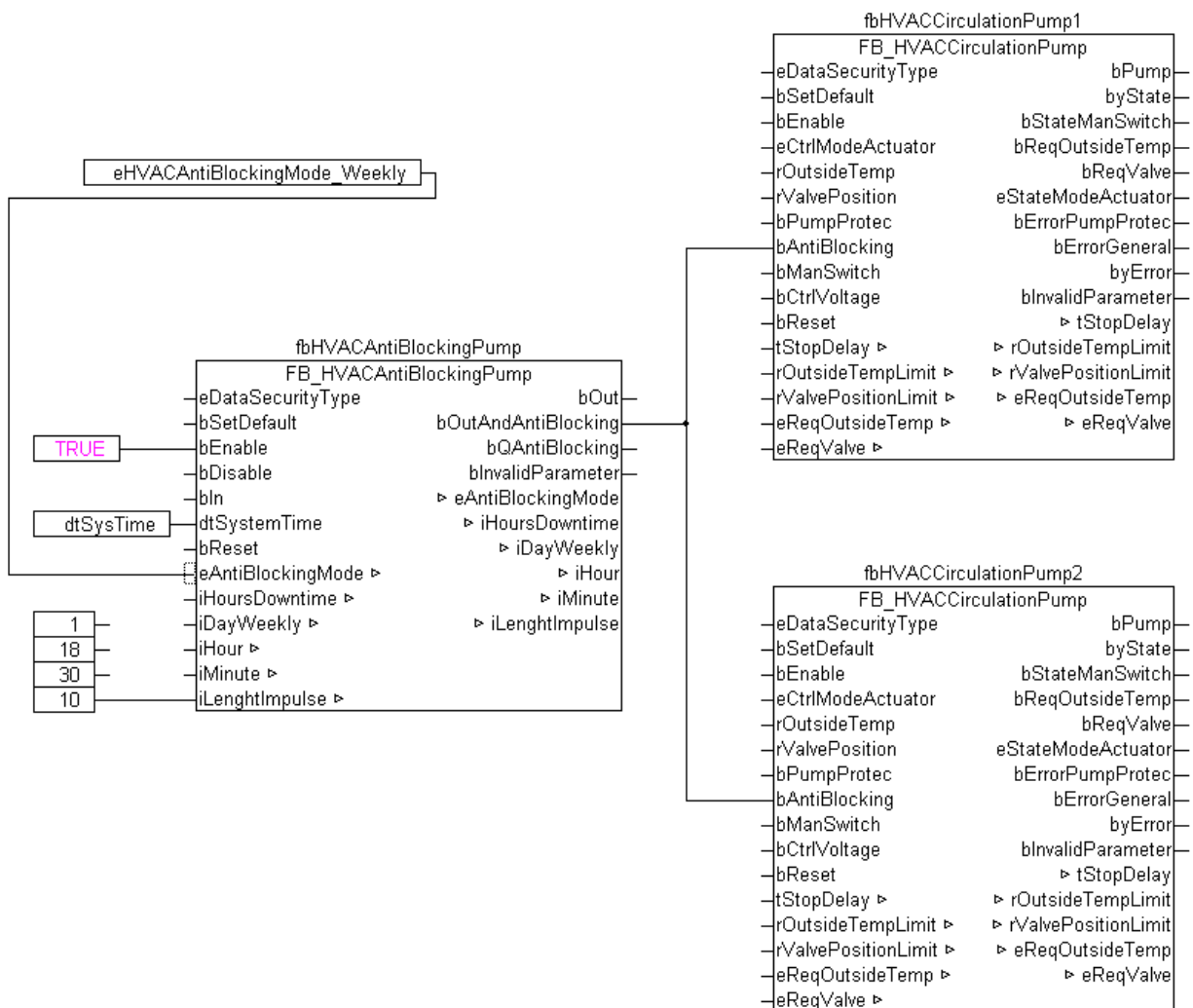
Example of a combination of a pump block and the FB_HVACAntiBlocking:

- The pump output *bPump* is applied to *bIn* of the **FB_HVACAntiBlocking** also without *bEnable*.
- If the input *bPumpProtec* goes to FALSE, *bPump* is switched off and *bDisable* becomes active via the message *bErrorPumpProtec*, so that *bOut* is switched off.



Example of the control of several pumps via one AntiBlocking instance:

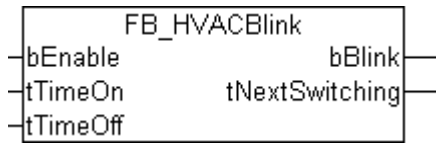
- Enable fbHVACAntiBlocking via *bEnable*.
- Select weekly mode (e.g. Monday at 18:30 for 10 secs)



Documents about this

example_persistent_e.zip (Resources/zip/11659714827.zip)

3.6.5 FB_HVACBlink



Application

The function block delivers a flash sequence at the output *bBlink* that is dependent on the two adjustable times *tTimeOn* and *tTimeOff*.

VAR_INPUT

```
bEnable      :REAL;
tTimeOn      :TIME;
tTimeOff     :TIME;
```

bEnable: enable of the function block.

tTimeOn: switch-on time of the flash pulse, *bBlink* = TRUE.

tTimeOff: switch-off time of the flash pulse, *bBlink* = FALSE.

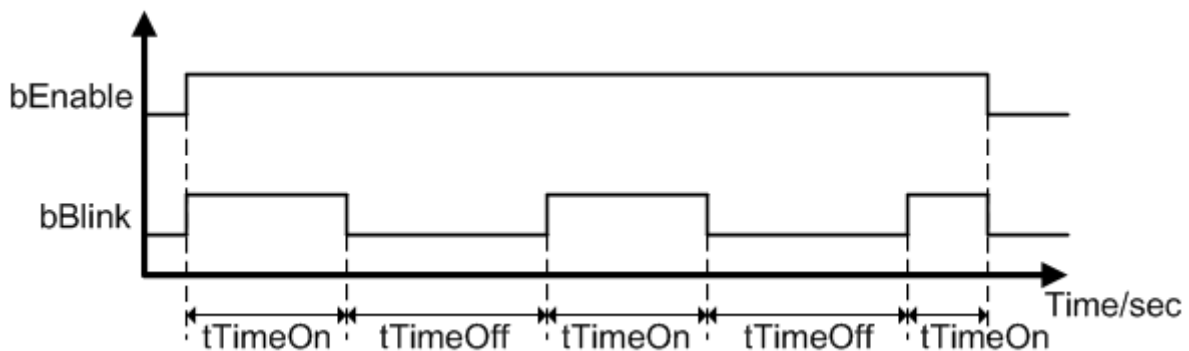
VAR_OUTPUT

```
bBlink       :BOOL;
tNextSwitching :TIME;
```

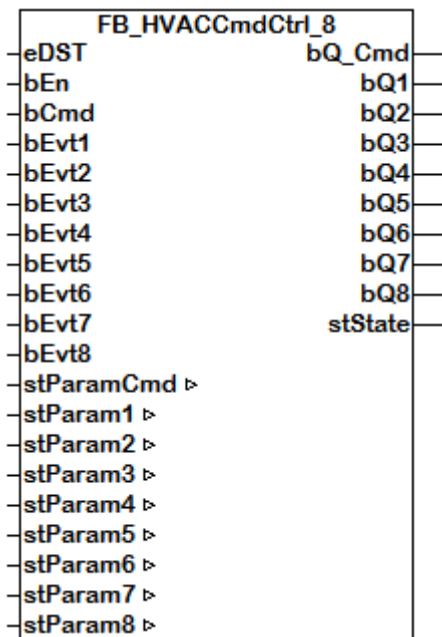
bBlink: flash output

tNextSwitching: time until the next change of state of the output *bBlink*.

Behavior of the output value



3.6.6 FB_HVACCmdCtrl_8



Application

With this function block, individual aggregates in a system can be sequentially switched on ($bQ_Cmd > bQ1 > bQ2 \dots bQ8$) or switched off ($bQ8 > bQ7 > bQ6 \dots bQ_Cmd$). *FB_HVACCmdCtrl_8* can be used as the starting function block of a ventilation system.

An event and a parameter structure with time variables belong to each output. These can be used to define a switch-on or switch-off delay as well as a minimum switch-on or minimum switch-off duration of the output.

Switch-on condition if and are TRUE: *bEn bCmd*

$$Q(n) = Q(n - 1) \text{ AND Event}(n) \text{ AND (DelayOn}(n) = 0 \text{ AND (MinOff}(n) = 0 \text{ AND } stState.udiStep = n$$

Example for the output *bQ_Cmd*

$$bQ_Cmd = bCmd \text{ AND } stState.udiSecRT_DelayOn = 0 \text{ AND } stState.udiSecRT_MinOff[0] = 0 \text{ AND } stState.udiStep = 0$$

Example for the output *bQ2*

$$bQ2 = bQ1 \text{ AND } bEvt2 \text{ AND } stState.udiSecRT_DelayOn = 0 \text{ AND } stState[2].udiSecRT_MinOff = 0 \text{ AND } stState.udiStep = 2$$

Switch-off condition if and are TRUE: *bEn bCmd*

$$Q(n) = \text{NOT } Q(n + 1) \text{ AND (DelayOff}(n) = 0 \text{ AND (MinOn}(n) = 0 \text{ AND } stState.udiStep = n$$

Example for the output *bQ8*

$$bQ8 = stState.udiSecRT_DelayOff = 0 \text{ AND } stState[7].udiSecRT_MinOn = 0 \text{ AND } stState.udiStep = 8$$

Example for the output *bQ_Cmd*

$$bQ_Cmd = \text{NOT } bQ1 \text{ AND } stState.udiSecRT_DelayOff = 0 \text{ AND } stState[0].udiSecRT_MinOn = 0 \text{ AND } stState.udiStep = 0$$

Current step

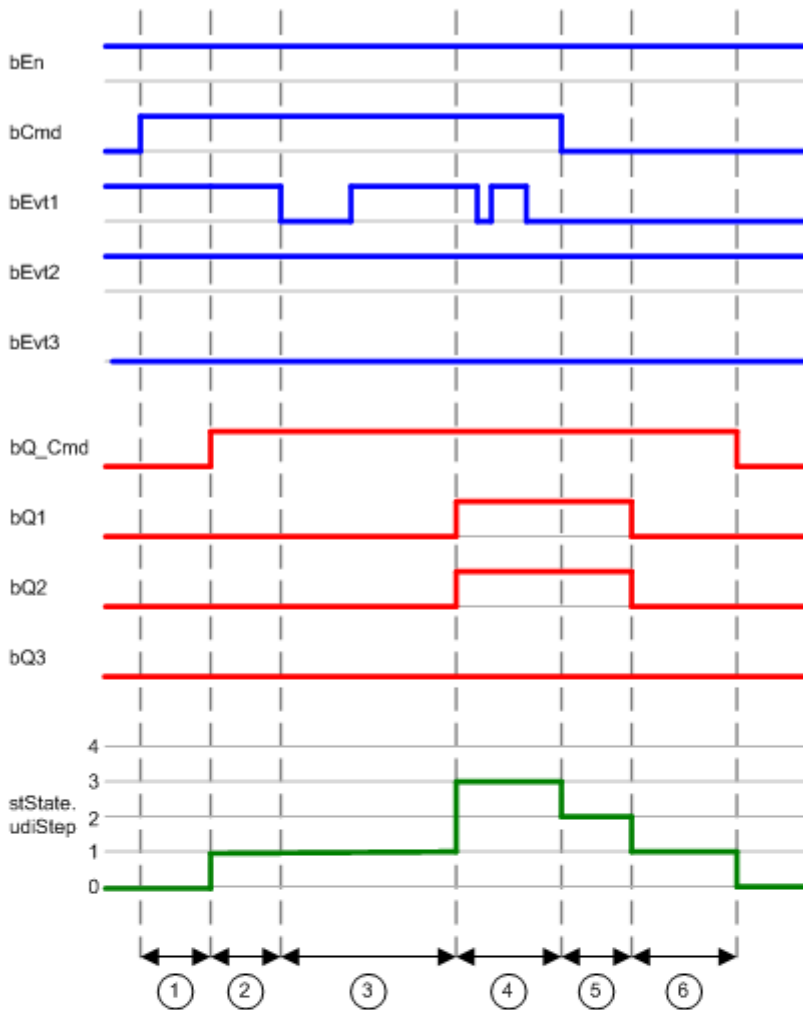
On the basis of the status variable *stState.udiStep* it is possible to see which step the function block is currently in.

Switch-on and switch-off sequence

If bEn and $bCmd$ are TRUE, the switch-on sequence is ($bQ_Cmd > bQ1 > bQ2 \dots bQ8$). The switch-off sequence ($bQ8 > bQ7 > bQ6 \dots bQ_Cmd$) is active if $bEn = \text{TRUE}$ and $bCmd = \text{FALSE}$.

If an output has been set in the switch-on sequence, then it remains latched and is only reset in the switch-off phase. In the switch-off sequence, no event ($bEvt1-8$) has any effect on the switch-off conditions of the outputs.

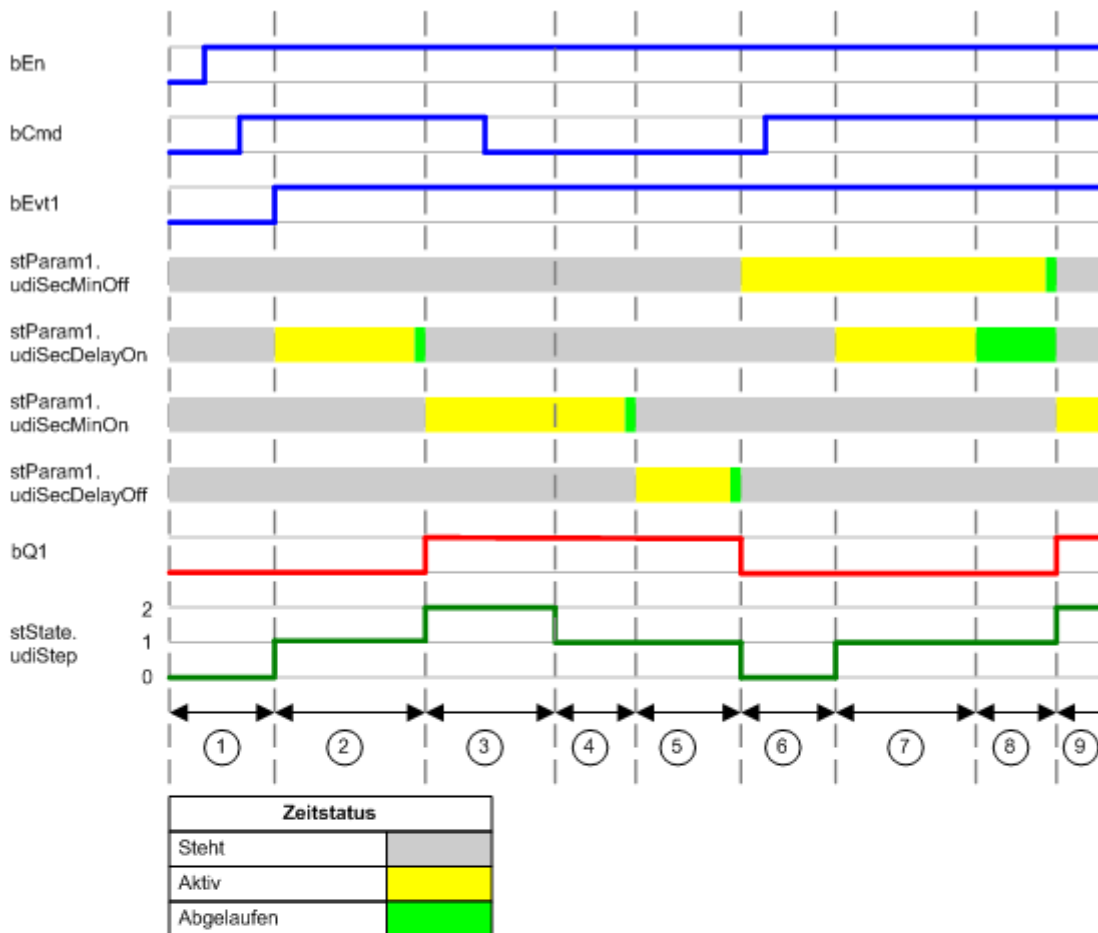
If the function block is in the switch-off sequence and the switch-on phase is activated by $bCmd = \text{TRUE}$, then the function block starts in step 0 with the switch-on conditions of bQ_Cmd . For the other outputs the minimum switch-on or switch-off times are active until their expiry.



- Start der Einschaltfolge $bCmd = TRUE$. Nach Ablauf der Einschaltverzögerungszeit $stState.udiSecRT_DelayOn$ wird bQ_Cmd TRUE. Der Status der Stufe ($stState.udiStep$) ändert seinen Wert von 0 nach 1.
- Das Ereignis $bEvt1$ ändert seinen Status von TRUE nach FALSE. An dem Status von $bQ1$ ändert sich nichts, weil die eingestellte Einschaltverzögerungszeit der Stufe 1 ($stParam1.udiSecDelayOn$) noch nicht abgelaufen ist.
- Das Ereignis $bEvt1$ ändert seinen Status von FALSE nach TRUE. Nach Ablauf der Einschaltverzögerungszeit $stState.udiSecRT_DelayOn$ wird $bQ1 = TRUE$. Der Status der Stufe ($stState.udiStep$) ändert seinen Wert von 1 nach 2.
- Da das Ereignis $bEvt2$ vor Freigabe der Stufe 2 TRUE war und für die Stufe 2 keine Einschaltverzögerungszeit eingestellt wurde, ändert sich sofort der Status von $bQ2$ auf TRUE. Im Zuge dessen ändert sich der Wert der Stufe ($stState.udiStep$) von 2 nach 3.
- Start der Ausschaltfolge $bCmd = FALSE$. Der Status der Stufe ($stState.udiStep$) ändert seinen Wert direkt von 3 nach 2, weil in Stufe 3 der Ausgang $bQ3$ nicht gesetzt wurde.
- Nach Ablauf der Ausschaltverzögerungszeit $stState.udiSecRT_DelayOff$ ändert sich der Status von $bQ2$ nach FALSE und der Wert der Stufe von 2 nach 1.
- Nach Ablauf der Ausschaltverzögerungszeit $stState.udiSecRT_DelayOff$ ändert sich der Wert der Stufe ($stState.udiStep$) von 1 nach 0.

Time responses of an individual step

The diagram shows the time responses of step 1 or the output $bQ1$.



- ① Freigabe des Funktionsbaustein mit $bEn = TRUE$. Start der Einschaltfolge $bCmd = TRUE$.
- ② Der Status der Stufe ($stState.udiStep$) ändert sich von 0 nach 1. Dadurch wird die Einschaltverzögerungszeit der Stufe 1 ($stParam1.udiSecDelayOn$) aktiviert.
- ③ Die Einschaltverzögerung der Stufe 1 ist abgelaufen. Da die Mindestausschaltzeit nicht aktiv ist und das Ereignis $bEvt1$ $TRUE$ ist, wird der Ausgang $bQ1$ $TRUE$. Die Mindesteinschaltzeit ($stParam1.udiSecDelayOn$) des Ausgangs $bQ1$ wird aktiviert. Der Status der Stufe ($stState.udiStep$) ändert sich von 1 nach 2.
- ④ Die Stufe 1 ($stState.udiStep$) ist wieder aktiv.
- ⑤ Nach Ablauf der Mindesteinschaltzeit ($stParam1.udiSecDelayOn$) wird die Ausschaltverzögerung ($stParam1.udiSecDelayOff$) der Stufe 1 aktiviert.
- ⑥ Nach Ablauf der Ausschaltverzögerungszeit ($stParam1.udiSecDelayOff$) ändert sich die Status von $bQ1$ auf $FALSE$ und der Stufe ($stSate.udiStep$) auf 0. Die Mindestausschaltzeit ($stParam1.udiSecDelayOff$) des Ausgangs $bQ1$ wird aktiviert.
- ⑦ Der Status der Stufe ($stSate.udiStep$) ändert sich von 0 nach 1. Die Einschaltverzögerungszeit der Stufe 1 ($stParam1.udiSecDelayOn$) wird aktiviert.
- ⑧ Die Einschaltverzögerungszeit der Stufe 1 ($stParam1.udiSecDelayOn$) ist abgelaufen. Der Ausgang $bQ1$ wird nicht eingeschaltet, weil die Mindestausschaltzeit ($stParam1.udiSecMinOff$) noch nicht abgelaufen ist.
- ⑨ Der Ausgang $bQ1$ wird eingeschaltet, weil die Mindestausschaltzeit ($stParam1.udiSecMinOff$) abgelaufen ist. Der Status der Stufe ($stState.udiStep$) ändert sich von 1 nach 2.

Application example

The application example shows the function block *FB_HVACCmdCtrl_8* in conjunction with *FB_HVACPRIORITY_INT_8* as the system start program for a ventilation system. The example **AC03_StartAC.PRG** can be found in the folder **Language CFC > SpecialFunctions > Start**.


Download	Required library
TcHVAC.pro [▶ 531]	TcHVAC.lib

VAR_INPUT

```
eDST      : E_HVACDataSecurityType;
bEn       : BOOL;
bCmd      : BOOL;
bEvt1     : BOOL;
bEvt2     : BOOL;
bEvt3     : BOOL;
bEvt4     : BOOL;
bEvt5     : BOOL;
bEvt6     : BOOL;
bEvt7     : BOOL;
bEvt8     : BOOL;
```

eDST: if *eDST := eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block *FB_HVACPersistentDataHandling* must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If *eDST:= eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE
A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if <i>eDST:= eHVACDataSecurityType_Persistent</i> . It would lead to early wear of the flash memory.

bEn: enable of the function block. The function block is disabled if *bEn* is FALSE. All outputs go to FALSE and the timers for the switch-on and switch-off delays as well as the minimum switch-on and switch-off times are reset. A collective error message or an emergency stop switch for immediately switching off a system can be connected to the input *bEn*.

bCmd: the switch-on or switch-off sequence of the function block is defined with *bCmd*. If *bCmd* is TRUE, then the function block is in the switch-on sequence of the outputs. The switching on of a system could come, for example, from a timer program. If *bCmd* is FALSE, then the function block is in the switch-off sequence of the outputs.

bCmd counts as one of the switch-on conditions of *bQ_Cmd*.

bEvt1: the event *bEvt1* counts as one of the switch-on conditions of *bQ1*. In the switch-on phase of *bQ1*, the switch-on delay of the output is active only if *bEvt1* is TRUE.

Switch-on condition:

```
bQ1 = bQ_Cmd AND bEvt1 AND stState.udiSecRT_DelayOn = 0 AND stState.udiSecRT_MinOff[1] = 0 AND stState.udiStep = 1
```

The event *bEvt1* has no effect on the switch-off condition of *bQ1*.

bEvt2: the event *bEvt2* counts as one of the switch-on conditions of *bQ2*. In the switch-on phase of *bQ2*, the switch-on delay of the output is active only if *bEvt2* is TRUE.

Switch-on condition:

$bQ2 = bQ1 \text{ AND } bEvt2 \text{ AND } stState.udiSecRT_DelayOn = 0 \text{ AND } stState.udiSecRT_MinOff[2] = 0$
 $\text{AND } stState.udiStep = 2$

The event *bEvt2* has no effect on the switch-off condition of *bQ2*.

bEvt3: the event *bEvt3* counts as one of the switch-on conditions of *bQ3*. In the switch-on phase of *bQ3*, the switch-on delay of the output is active only if *bEvt3* is TRUE.

Switch-on condition:

$bQ3 = bQ2 \text{ AND } bEvt3 \text{ AND } stState.udiSecRT_DelayOn = 0 \text{ AND } stState.udiSecRT_MinOff[3] = 0$
 $\text{AND } stState.udiStep = 3$

The event *bEvt3* has no effect on the switch-off condition of *bQ3*.

bEvt4: the event *bEvt4* counts as one of the switch-on conditions of *bQ4*. In the switch-on phase of *bQ4*, the switch-on delay of the output is active only if *bEvt4* is TRUE.

Switch-on condition:

$bQ4 = bQ3 \text{ AND } bEvt4 \text{ AND } stState.udiSecRT_DelayOn = 0 \text{ AND } stState.udiSecRT_MinOff[4] = 0$
 $\text{AND } stState.udiStep = 4$

The event *bEvt4* has no effect on the switch-off condition of *bQ4*.

bEvt5: the event *bEvt5* counts as one of the switch-on conditions of *bQ5*. In the switch-on phase of *bQ5*, the switch-on delay of the output is active only if *bEvt5* is TRUE.

Switch-on condition:

$bQ5 = bQ4 \text{ AND } bEvt5 \text{ AND } stState.udiSecRT_DelayOn = 0 \text{ AND } stState.udiSecRT_MinOff[5] = 0$
 $\text{AND } stState.udiStep = 5$

The event *bEvt5* has no effect on the switch-off condition of *bQ5*.

bEvt6: the event *bEvt6* counts as one of the switch-on conditions of *bQ6*. In the switch-on phase of *bQ6*, the switch-on delay of the output is active only if *bEvt6* is TRUE.

Switch-on condition:

$bQ6 = bQ5 \text{ AND } bEvt6 \text{ AND } stState.udiSecRT_DelayOn = 0 \text{ AND } stState.udiSecRT_MinOff[6] = 0$
 $\text{AND } stState.udiStep = 6$

The event *bEvt6* has no effect on the switch-off condition of *bQ6*.

bEvt7: the event *bEvt7* counts as one of the switch-on conditions of *bQ7*. In the switch-on phase of *bQ7*, the switch-on delay of the output is active only if *bEvt7* is TRUE.

Switch-on condition:

$bQ7 = bQ6 \text{ AND } bEvt7 \text{ AND } stState.udiSecRT_DelayOn = 0 \text{ AND } stState.udiSecRT_MinOff[7] = 0$
 $\text{AND } stState.udiStep = 7$

The event *bEvt7* has no effect on the switch-off condition of *bQ7*.

bEvt8: the event *bEvt8* counts as one of the switch-on conditions of *bQ8*. In the switch-on phase of *bQ8*, the switch-on delay of the output is active only if *bEvt8* is TRUE.

Switch-on condition:

$bQ8 = bQ7 \text{ AND } bEvt8 \text{ AND } stState.udiSecRT_DelayOn = 0 \text{ AND } stState.udiSecRT_MinOff[8] = 0$
 $\text{AND } stState.udiStep = 8$

The event *bEvt8* has no effect on the switch-off condition of *bQ8*.

VAR_OUTPUT

```

bQ_Cmd      : BOOL;
bQ1         : BOOL;
bQ2         : BOOL;
bQ3         : BOOL;
bQ4         : BOOL;
bQ5         : BOOL;
bQ6         : BOOL;
bQ7         : BOOL;
bQ8         : BOOL;
stState     : ST_HVACCmdCtrl_8State;

```

bQ_Cmd: output to enable a device. If the output *bQ_Cmd* has been set in the switch-on sequence, then it remains latched and is only reset in the switch-off phase.

To *bQ_Cmd* belongs the input variable *bCmd* and the parameter structure *stParamCmd*. *bCmd* is one of the switch-on conditions of *bQ_Cmd*. The time variables of the parameter structure can be used to define a switch-on or switch-off delay as well as a minimum switch-on or minimum switch-off duration for the output. The state of the time responses is indicated with the output structure *stState*.

Switch-on condition

$$bQ_Cmd = bCmd \text{ AND } stState.udiSecRT_DelayOn = 0 \text{ AND } stState.udiSecRT_MinOff[0] = 0 \\ \text{ AND } stState.udiStep = 0$$

Switch-off condition

$$bQ_Cmd = \text{ NOT } bQ1 \text{ AND } stState.udiSecRT_DelayOff = 0 \text{ AND } stState.udiSecRT_MinOn[0] = 0 \\ \text{ AND } stState.udiStep = 0$$

bQ1: output to enable a device. If the output *bQ1* has been set in the switch-on sequence, then it remains latched and is only reset in the switch-off phase.

To *bQ1* belongs the event 1 *bEvt1* and the parameter structure *stParam1*. *bEvt1* is one of the switch-on conditions of *bQ1*. The time variables of the parameter structure can be used to define a switch-on or switch-off delay as well as a minimum switch-on or minimum switch-off duration for the output. The state of the time responses is indicated with the output structure *stState*.

Switch-on condition

$$bQ1 = bEvt1 \text{ AND } stState.udiSecRT_DelayOn = 0 \text{ AND } stState.udiSecRT_MinOff[1] = 0 \text{ AND } stState.udiStep = 1$$

Switch-off condition

$$bQ1 = \text{ NOT } bQ1 \text{ AND } stState.udiSecRT_DelayOff = 0 \text{ AND } stState.udiSecRT_MinOn[1] = 0 \\ \text{ AND } stState.udiStep = 1$$

bQ2: output to enable a device. If the output *bQ2* has been set in the switch-on sequence, then it remains latched and is only reset in the switch-off phase.

To *bQ2* belongs the event 2 *bEvt2* and the parameter structure *stParam2*. *bEvt2* is one of the switch-on conditions of *bQ2*. The time variables of the parameter structure can be used to define a switch-on or switch-off delay as well as a minimum switch-on or minimum switch-off duration for the output. The state of the time responses is indicated with the output structure *stState*.

Switch-on condition

$$bQ2 = bEvt2 \text{ AND } stState.udiSecRT_DelayOn = 0 \text{ AND } stState.udiSecRT_MinOff[2] = 0 \text{ AND } stState.udiStep = 2$$

Switch-off condition

$$bQ2 = \text{ NOT } bQ2 \text{ AND } stState.udiSecRT_DelayOff = 0 \text{ AND } stState.udiSecRT_MinOn[2] = 0 \\ \text{ AND } stState.udiStep = 2$$

bQ3: output to enable a device. If the output *bQ3* has been set in the switch-on sequence, then it remains latched and is only reset in the switch-off phase.

To *bQ3* belongs the event 3 *bEvt3* and the parameter structure *stParam3.bEvt3* is one of the switch-on conditions of *bQ3*. The time variables of the parameter structure can be used to define a switch-on or switch-off delay as well as a minimum switch-on or minimum switch-off duration for the output. The state of the time responses is indicated with the output structure *stState*.

Switch-on condition

$$bQ3 = bEvt3 \text{ AND } stState.udiSecRT_DelayOn = 0 \text{ AND } stState.udiSecRT_MinOff[3] = 0 \text{ AND } stState.udiStep = 3$$

Switch-off condition

$$bQ3 = \text{NOT} bQ3 \text{ AND } stState.udiSecRT_DelayOff = 0 \text{ AND } stState.udiSecRT_MinOn[3] = 0 \text{ AND } stState.udiStep = 3$$

bQ4: output to enable a device. If the output *bQ4* has been set in the switch-on sequence, then it remains latched and is only reset in the switch-off phase.

To *bQ4* belongs the event 4 *bEvt4* and the parameter structure *stParam4.bEvt4* is one of the switch-on conditions of *bQ4*. The time variables of the parameter structure can be used to define a switch-on or switch-off delay as well as a minimum switch-on or minimum switch-off duration for the output. The state of the time responses is indicated with the output structure *stState*.

Switch-on condition

$$bQ4 = bEvt4 \text{ AND } stState.udiSecRT_DelayOn = 0 \text{ AND } stState.udiSecRT_MinOff[4] = 0 \text{ AND } stState.udiStep = 4$$

Switch-off condition

$$bQ4 = \text{NOT} bQ4 \text{ AND } stState.udiSecRT_DelayOff = 0 \text{ AND } stState.udiSecRT_MinOn[4] = 0 \text{ AND } stState.udiStep = 4$$

bQ5: output to enable a device. If the output *bQ5* has been set in the switch-on sequence, then it remains latched and is only reset in the switch-off phase.

To *bQ5* belongs the event 5 *bEvt5* and the parameter structure *stParam5.bEvt5* is one of the switch-on conditions of *bQ5*. The time variables of the parameter structure can be used to define a switch-on or switch-off delay as well as a minimum switch-on or minimum switch-off duration for the output. The state of the time responses is indicated with the output structure *stState*.

Switch-on condition

$$bQ5 = bEvt5 \text{ AND } stState.udiSecRT_DelayOn = 0 \text{ AND } stState.udiSecRT_MinOff[5] = 0 \text{ AND } stState.udiStep = 5$$

Switch-off condition

$$bQ5 = \text{NOT} bQ5 \text{ AND } stState.udiSecRT_DelayOff = 0 \text{ AND } stState.udiSecRT_MinOn[5] = 0 \text{ AND } stState.udiStep = 5$$

bQ6: output to enable a device. If the output *bQ6* has been set in the switch-on sequence, then it remains latched and is only reset in the switch-off phase.

To *bQ6* belongs the event 6 *bEvt6* and the parameter structure *stParam6.bEvt6* is one of the switch-on conditions of *bQ6*. The time variables of the parameter structure can be used to define a switch-on or switch-off delay as well as a minimum switch-on or minimum switch-off duration for the output. The state of the time responses is indicated with the output structure *stState*.

Switch-on condition

$$bQ6 = bEvt6 \text{ AND } stState.udiSecRT_DelayOn = 0 \text{ AND } stState.udiSecRT_MinOff[6] = 0 \text{ AND } stState.udiStep = 6$$

Switch-off condition

$$bQ6 = \text{NOT} bQ6 \text{ AND } stState.udiSecRT_DelayOff = 0 \text{ AND } stState.udiSecRT_MinOn[6] = 0 \text{ AND } stState.udiStep = 6$$

bQ7: output to enable a device. If the output *bQ7* has been set in the switch-on sequence, then it remains latched and is only reset in the switch-off phase.

To *bQ7* belongs the event 7 *bEvt7* and the parameter structure *stParam7*. *bEvt7* is one of the switch-on conditions of *bQ7*. The time variables of the parameter structure can be used to define a switch-on or switch-off delay as well as a minimum switch-on or minimum switch-off duration for the output. The state of the time responses is indicated with the output structure *stState*.

Switch-on condition

$$bQ7 = bEvt7 \text{ AND } stState.udiSecRT_DelayOn = 0 \text{ AND } stState.udiSecRT_MinOff[7] = 0 \text{ AND } stState.udiStep = 7$$

Switch-off condition

$$bQ7 = \text{NOT } bQ7 \text{ AND } stState.udiSecRT_DelayOff = 0 \text{ AND } stState.udiSecRT_MinOn[7] = 0 \text{ AND } stState.udiStep = 7$$

bQ8: output to enable a device. If the output *bQ8* has been set in the switch-on sequence, then it remains latched and is only reset in the switch-off phase.

To *bQ8* belongs the event 8 *bEvt8* and the parameter structure *stParam8*. *bEvt8* is one of the switch-on conditions of *bQ8*. The time variables of the parameter structure can be used to define a switch-on or switch-off delay as well as a minimum switch-on or minimum switch-off duration for the output. The state of the time responses is indicated with the output structure *stState*.

Switch-on condition

$$bQ8 = bEvt8 \text{ AND } stState.udiSecRT_DelayOn = 0 \text{ AND } stState.udiSecRT_MinOff[8] = 0 \text{ AND } stState.udiStep = 8$$

Switch-off condition

$$bQ8 = stState.udiSecRT_DelayOff = 0 \text{ AND } stState.udiSecRT_MinOn[8] = 0 \text{ AND } stState.udiStep = 8$$

stState: this structure indicates the remaining time of the switch-on or switch-off delay of the active step and the minimum switch-on and switch-off duration of the outputs.

stState.udiSecRT_DelayOn: in the switch-on phase of the outputs (*bCmd* = TRUE), the remaining time of the switch-on delay of the current step *stState.udiStep* is indicated by *stState.udiSecRT_DelayOn*.

Example: If *stState.udiStep* = 5, then the function block is in step 5. The time variable of the parameter structure *stParam5.udiSecDelayOn* is used for the switch-on delay of the output *bQ5* and the remaining time is indicated by *udiSecRT_DelayOn*.

stState.udiSecRT_DelayOff: in the switch-off phase of the outputs (*bCmd* = FALSE), the remaining time of the switch-off delay of the current step *stState.udiStep* is indicated by *udiSecRT_DelayOff*.

Example: If *stState.udiStep* = 0, then the function block is in step 0. The time variable *stParamCmd.udiSecDelayOff* is used for the switch-off delay of the output *bQ_Cmd* and the remaining time is indicated by *udiSecRT_DelayOff*.

stState.udiStep: state indicating which step the function block is in.

Examples:

- If *stState.udiStep* = 0, then the function block is in the switch-on or switch-off sequence of the output *bQ_Cmd*.

- If *stState.udiStep* = 1, then the function block is in the switch-on or switch-off sequence of the output *bQ1*.

- If *stState.udiStep* = 8, then the function block is in the switch-on or switch-off sequence of the output *bQ8*.

stState.udiSecRT_MinOn[0]: .. g_iNumOfCmdCtrl 8 [► 531] The remaining time of the minimum switch-on duration of the outputs is indicated in the one-dimensional field (table)

stState.udiSecRT_MinOn[0..g_iNumOfCmdCtrl 8 [► 531]]. Following the expiry of the minimum switch-on duration of an output, the output can be switched off in the current step *stState.udiStep*.

stState.udiSecRT_MinOff[0]: .. g *iNumOfCmdCtrl_8* [► 531] The remaining time of the minimum switch-off duration of the outputs is indicated in the one-dimensional field (table)

stState.udiSecRT_MinOff[0..g iNumOfCmdCtrl_8 [► 531]]. Following the expiry of the minimum switch-off duration of an output, the output can be switched on in the current step *stState.udiStep*.

VAR_IN_OUT

```
stParamCmd   : ST_HVACCmdCtrl_8Param;
stParam1     : ST_HVACCmdCtrl_8Param;
stParam2     : ST_HVACCmdCtrl_8Param;
stParam3     : ST_HVACCmdCtrl_8Param;
stParam4     : ST_HVACCmdCtrl_8Param;
stParam5     : ST_HVACCmdCtrl_8Param;
stParam6     : ST_HVACCmdCtrl_8Param;
stParam7     : ST_HVACCmdCtrl_8Param;
stParam8     : ST_HVACCmdCtrl_8Param;
```

stParamCmd: The structure contains time variables that can be used to define a switch-on or switch-off delay as well as a minimum switch-on or minimum switch-off duration of the output *bQ_CMD*. The variable is saved persistently.

stParam1: the structure contains time variables that can be used to define a switch-on or switch-off delay as well as a minimum switch-on or minimum switch-off duration of the output *bQ1*. The variable is saved persistently.

stParam2: the structure contains time variables that can be used to define a switch-on or switch-off delay as well as a minimum switch-on or minimum switch-off duration of the output *bQ2*. The variable is saved persistently.

stParam3: the structure contains time variables that can be used to define a switch-on or switch-off delay as well as a minimum switch-on or minimum switch-off duration of the output *bQ3*. The variable is saved persistently.

stParam4: the structure contains time variables that can be used to define a switch-on or switch-off delay as well as a minimum switch-on or minimum switch-off duration of the output *bQ4*. The variable is saved persistently.

stParam5: the structure contains time variables that can be used to define a switch-on or switch-off delay as well as a minimum switch-on or minimum switch-off duration of the output *bQ5*. The variable is saved persistently.

stParam6: the structure contains time variables that can be used to define a switch-on or switch-off delay as well as a minimum switch-on or minimum switch-off duration of the output *bQ6*. The variable is saved persistently.

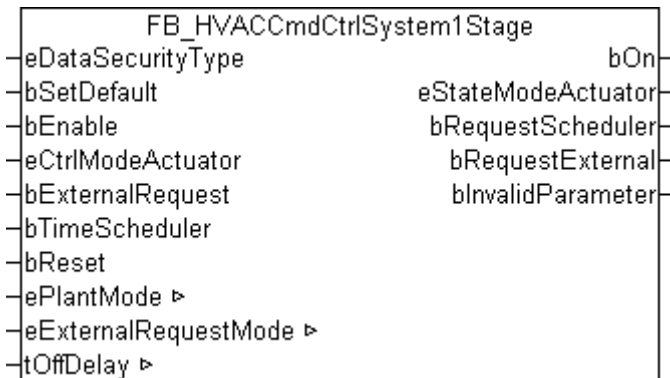
stParam7: the structure contains time variables that can be used to define a switch-on or switch-off delay as well as a minimum switch-on or minimum switch-off duration of the output *bQ7*. The variable is saved persistently.

stParam8: the structure contains time variables that can be used to define a switch-on or switch-off delay as well as a minimum switch-on or minimum switch-off duration of the output *bQ8*. The variable is saved persistently.

Documents about this

- 📄 [example_persistent_e.zip \(Resources/zip/11659714827.zip\)](#)

3.6.7 FB_HVACCmdCtrlSystem1Stage



Application


This function block *FB_HVACCmdCtrlSystem1Stage* is a system switch. It is used, for example, to switch a single-stage ventilation system to automatic or manual operation mode. In automatic operation mode the system can be controlled via a timer program or via the request from a control panel. The function block *FB_HVACCmdCtrlSystem1Stage* is active if the input variable *bEnable* is TRUE and *eCtrlModeActuator* = *eHVACActuatorMode_Auto_BMS* or *eHVACActuatorMode_Auto_OP*.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
eCtrlModeActuator : E_HVACActuatorMode;
bExternalRequest  : BOOL;
bTimeScheduler    : BOOL;
bReset            : BOOL;
```

eDataSecurityType: if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block *FB_HVACPersistentDataHandling* must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If *eDataSecurityType* := *eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled if *bEnable* = TRUE. If *bEnable* = FALSE, then *bOn* = FALSE.

eCtrlModeActuator: enum that specifies the system operation modes Manual, Auto and Off. In the event of an incorrect entry, operation continues internally with the last valid operating mode. This is *eHVACActuatorMode_Auto_BMS* in the case of initial commissioning. *bInvalidParameter* will be set in the event of an incorrect parameter entry.

bExternalRequest: external request for the system, e.g. from a control panel via a button or switch.

bTimeScheduler: request for the system via a timer program.

bReset: acknowledge input in the event of an error

VAR_OUTPUT

```
bOn          : BOOL;
eStateModeActuator: E_HVACActuatorMode;
bRequestScheduler : BOOL;
bRequestExternal  : BOOL;
bInvalidParameter : BOOL;
```

bOn: this output variable enables the system.

eStateModeActuator: Enum via which the state of the operation mode of the motor is fed back to the controller.

bRequestScheduler: this output indicates that the system is requested by the input variable *bTimeScheduler*.

bRequestExternal: this output indicates that the system is requested by the input variable *bExternalRequest*.

bInvalidParameter: indicates that an incorrect parameter is present at one of the variables *eCtrlModeActuator*, *ePlantMode* or *eExternalRequestMode*. An incorrect parameter entry does not lead to a standstill of the function block; see description of variables. After rectifying the incorrect parameter entry, the message *bInvalidParameter* must be acknowledged via *bReset*.

VAR_IN_OUT

```
ePlantMode      : E_HVACPlantMode;
eExternalRequestMode : E_HVACExternalRequestMode;
tOffDelay       : TIME;
```

ePlantMode: with this enumeration variable various functions of the system can be performed in automatic operation mode, depending on the input variables *bExternalRequest* and *bTimeScheduler*.

ePlantMode = eHVACPlantMode_TimeSchedulingOnly: the plant is switched on and off exclusively via the input variable *bTimeScheduler*.

ePlantMode = eHVACPlantMode_TimeScheduling_And_ExternalRequest: the plant is switched on if the input variables *bTimeScheduler* AND *bExternalRequest* = TRUE.

ePlantMode = eHVACPlantMode_TimeScheduling_Or_ExternalRequest: the plant is switched on by the input variables *bTimeScheduler* OR *bExternalRequest*.

ePlantMode = eHVACPlantMode_ExternalRequestOnly: the plant is switched on and off exclusively via the input variable *bExternalRequest*.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, operation continues with the default value. *bInvalidParameter* will be set in the event of an incorrect parameter entry. The variable is saved persistently. Preset to 0.

eExternalRequestMode: the enum *eExternalRequestMode* defines the mode of operation of the input variable *bExternalRequest* in automatic mode depending on the enum *ePlantMode*.

eExternalRequestMode = eHVACExternalRequestMode_ButtonOn_Off: the external request is set to TRUE after a rising edge of *bExternalRequest*. Another rising edge resets the request to FALSE again.

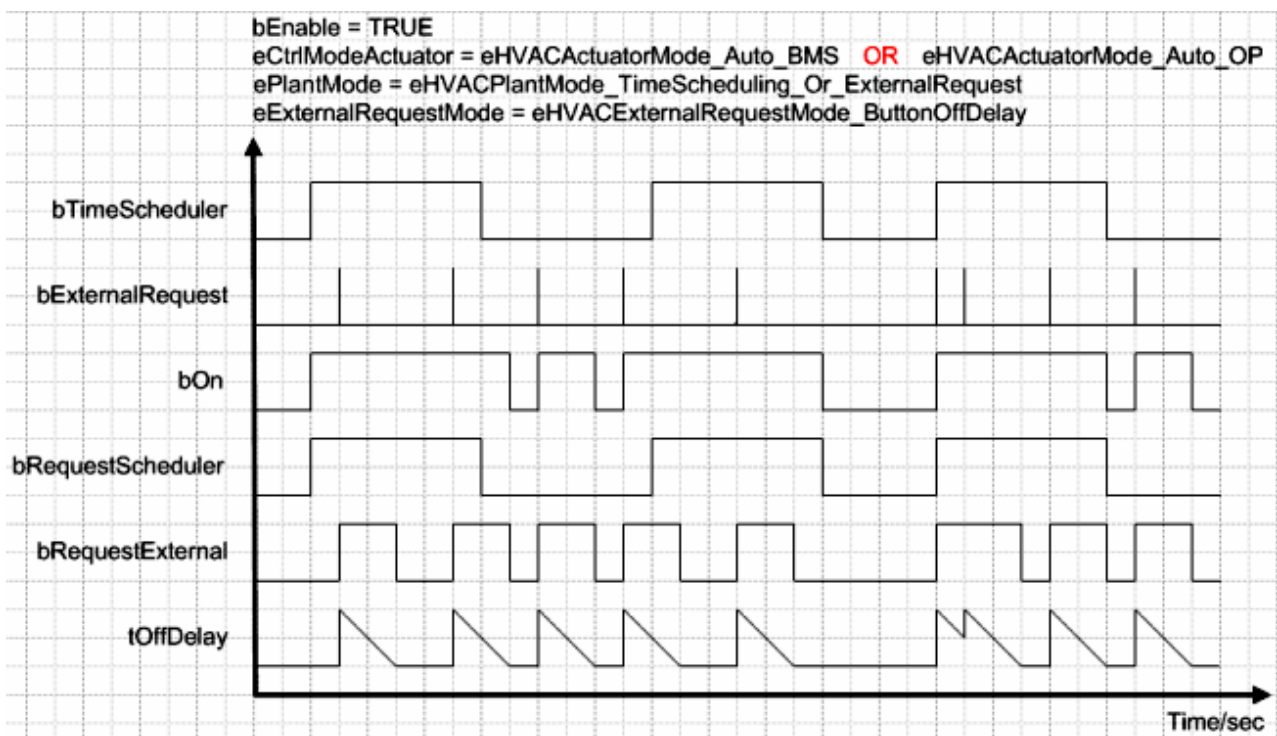
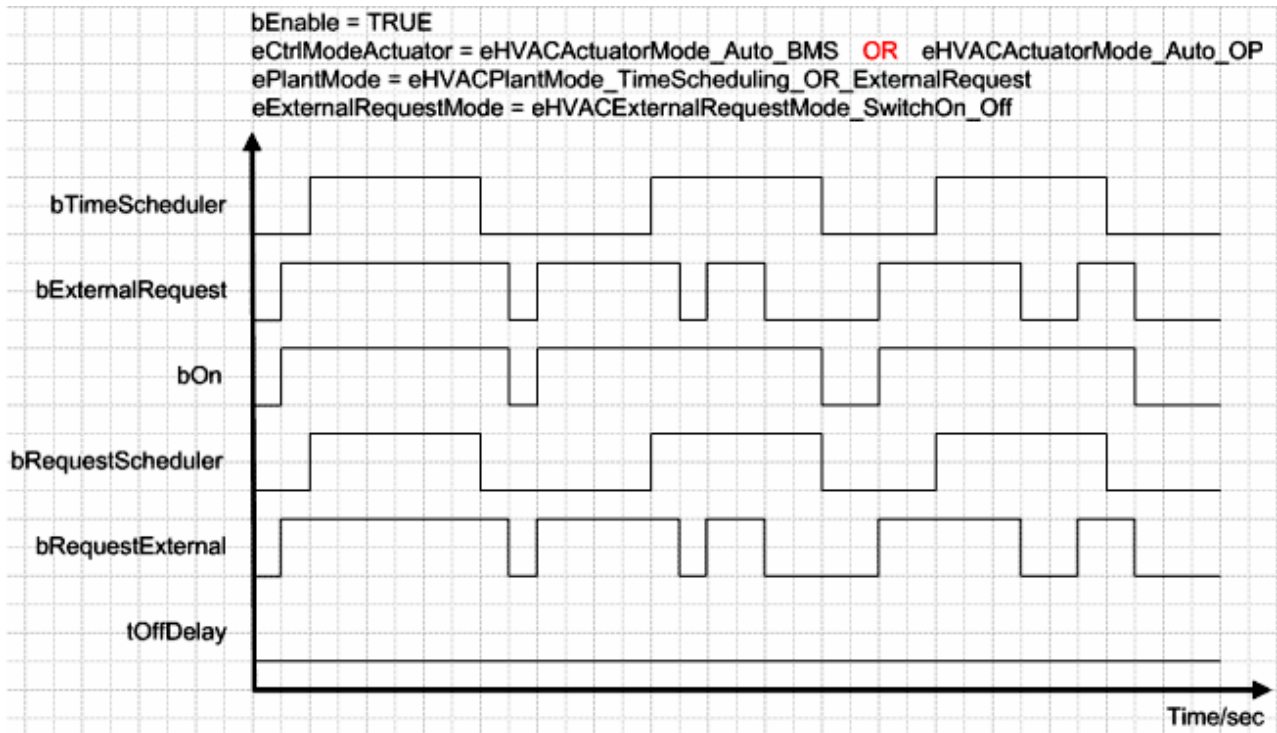
eExternalRequestMode = eHVACExternalRequestMode_ButtonOffDelay: the external request extends or sets the utilization time of the plant after a rising edge at the input variable *bExternalRequest* by the set time of *tOffDelay*.

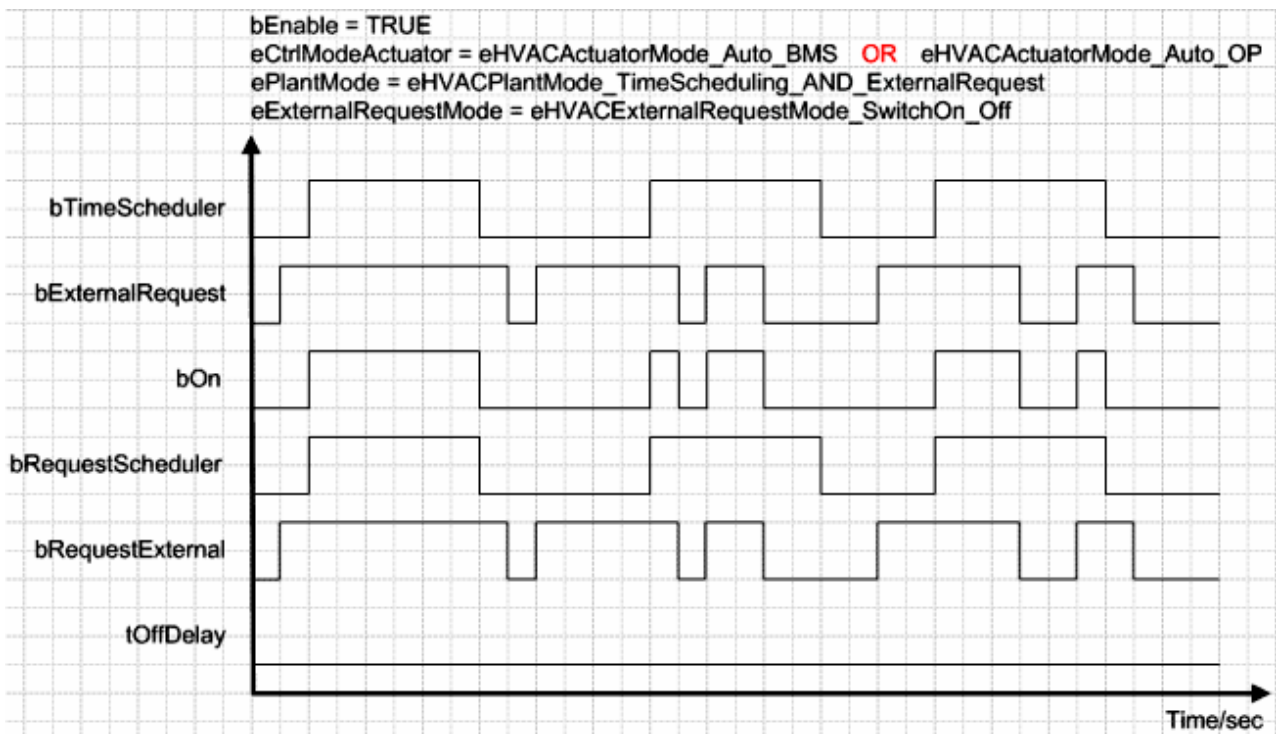
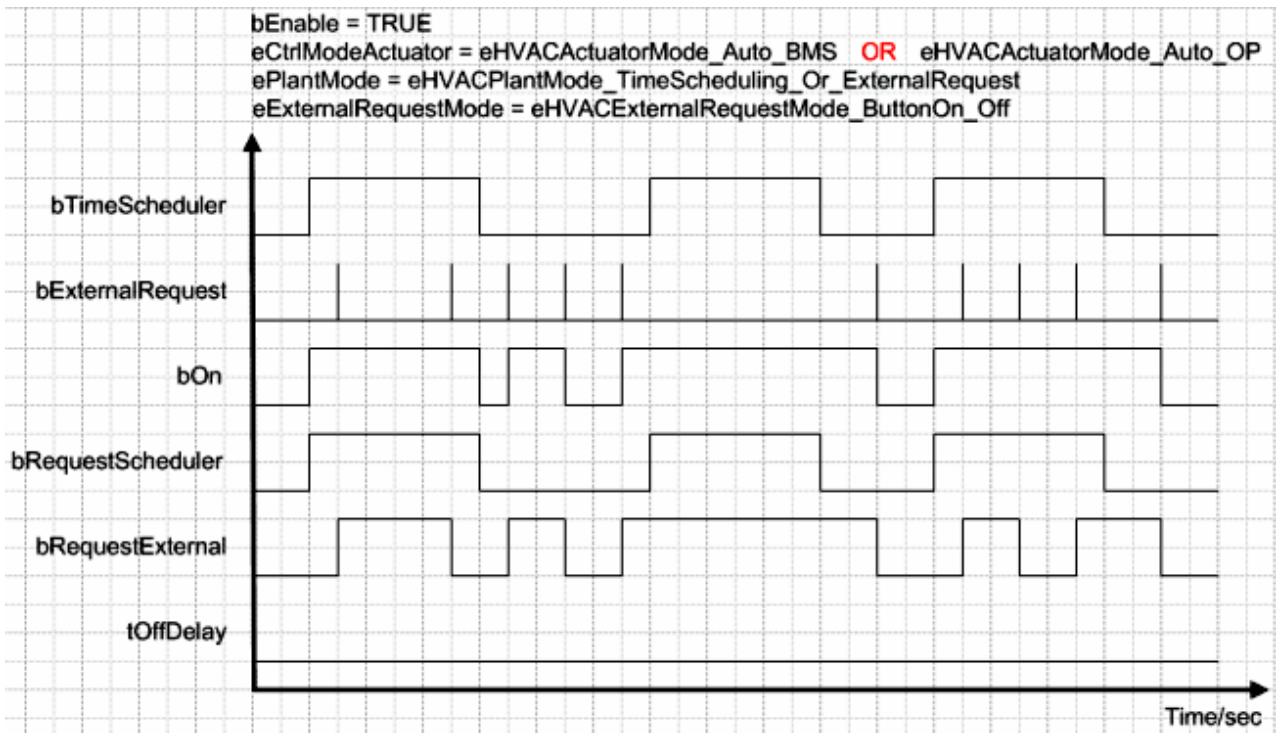
eExternalRequestMode = eHVACExternalRequestMode_SwitchOn_Off: the external request is active if *bExternalRequest* = TRUE. It is deactivated if *bExternalRequest* = FALSE.

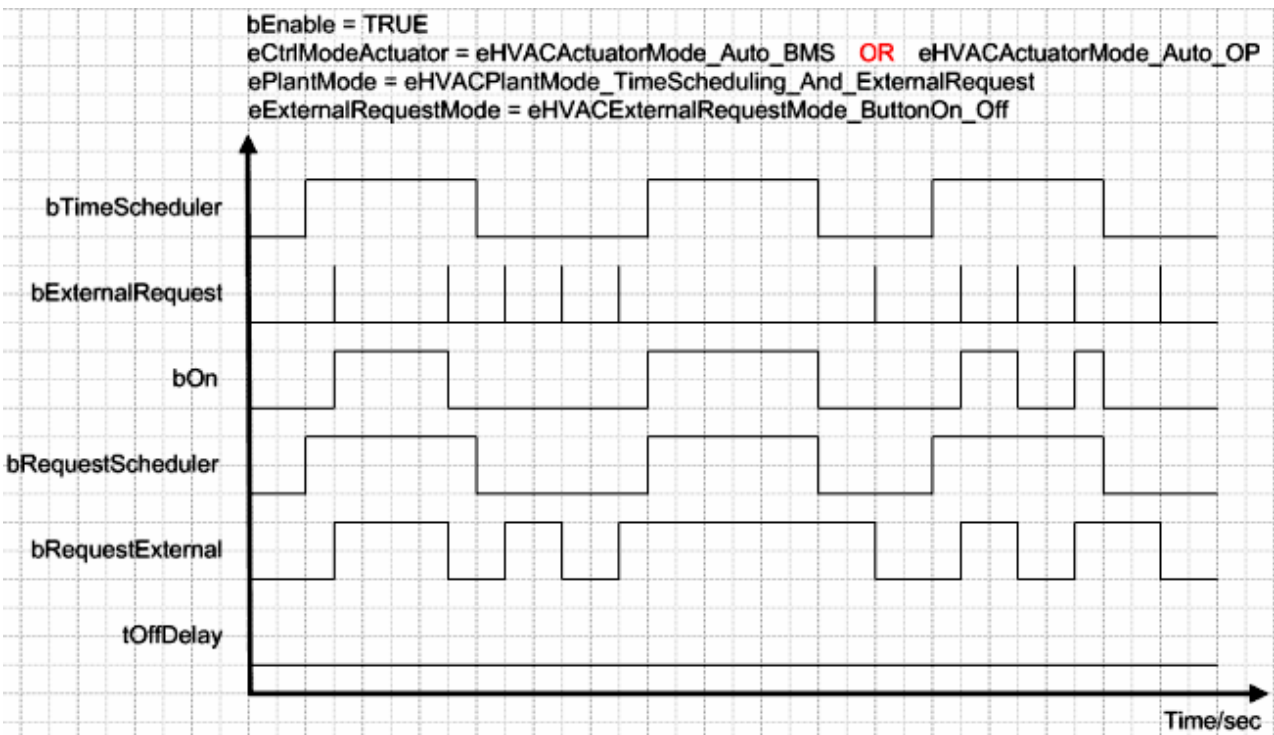
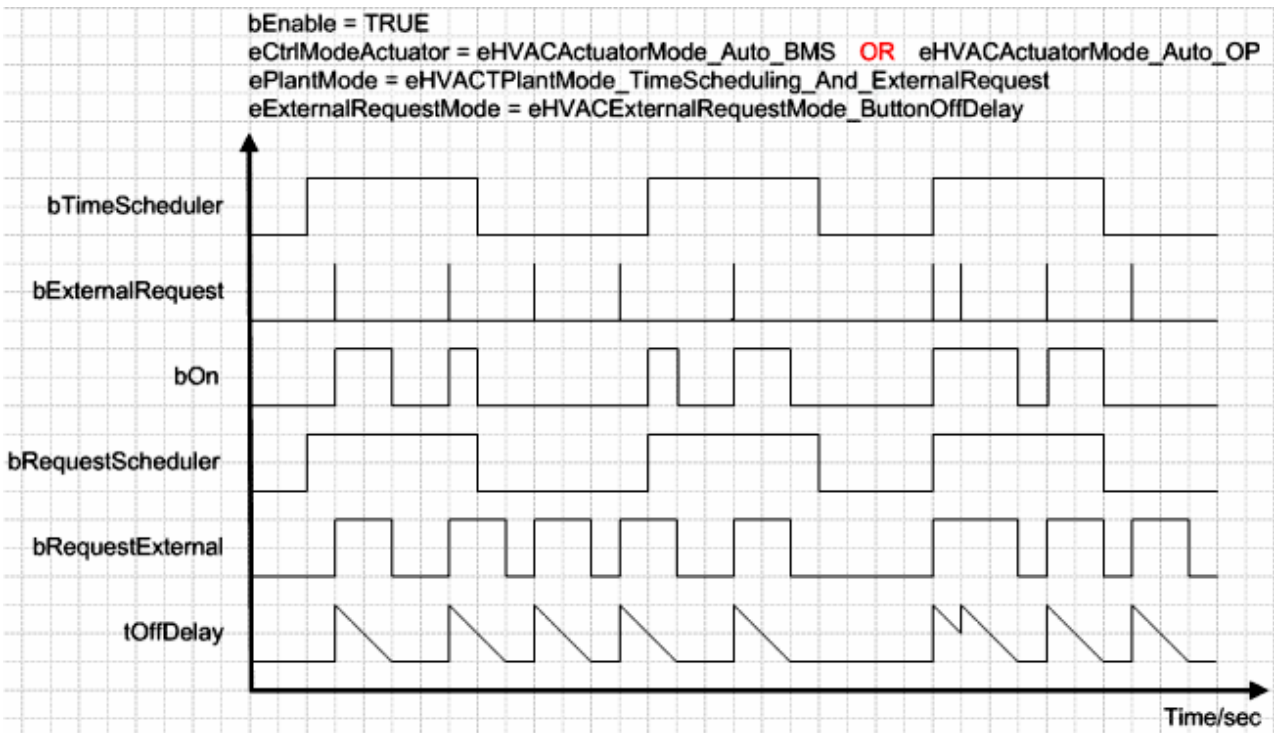
If an incorrect variable value is present, then the last valid variable value is used, if available. If there is no valid last value, operation continues with the default value. *bInvalidParameter* will be set in the event of an incorrect parameter entry. The variable is saved persistently. Preset to 0.

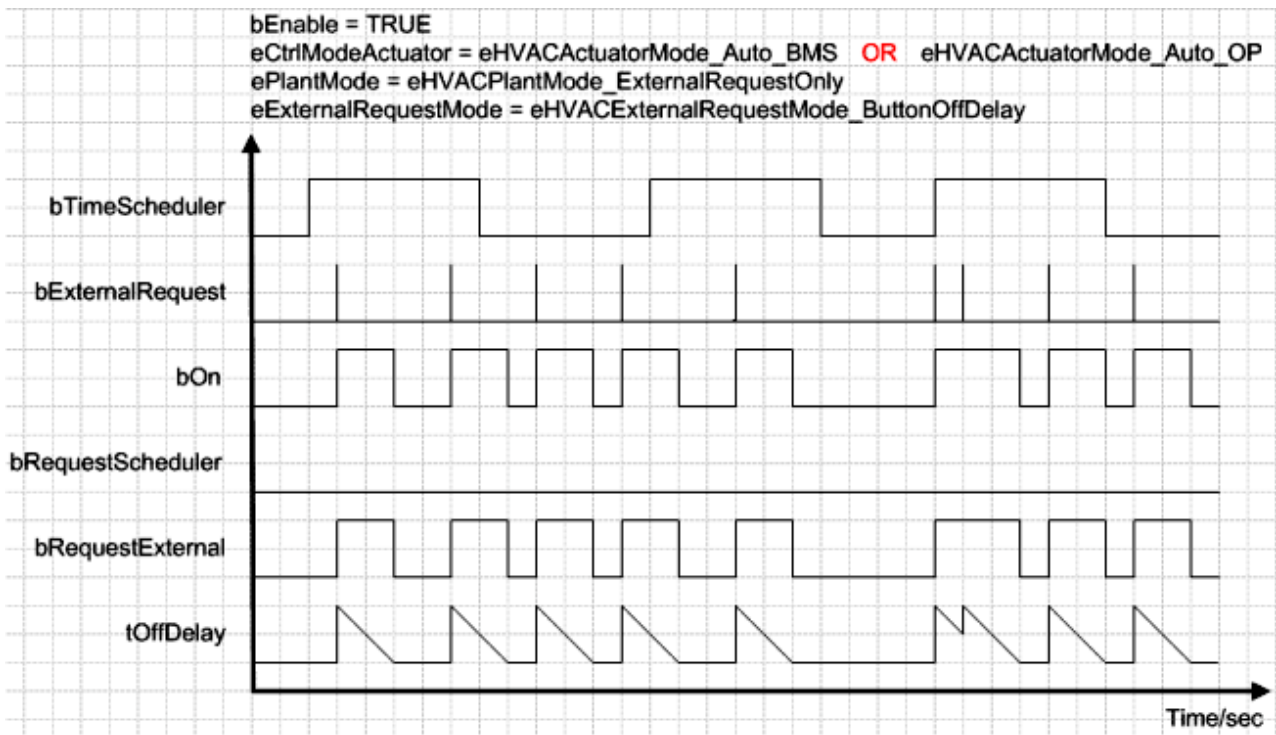
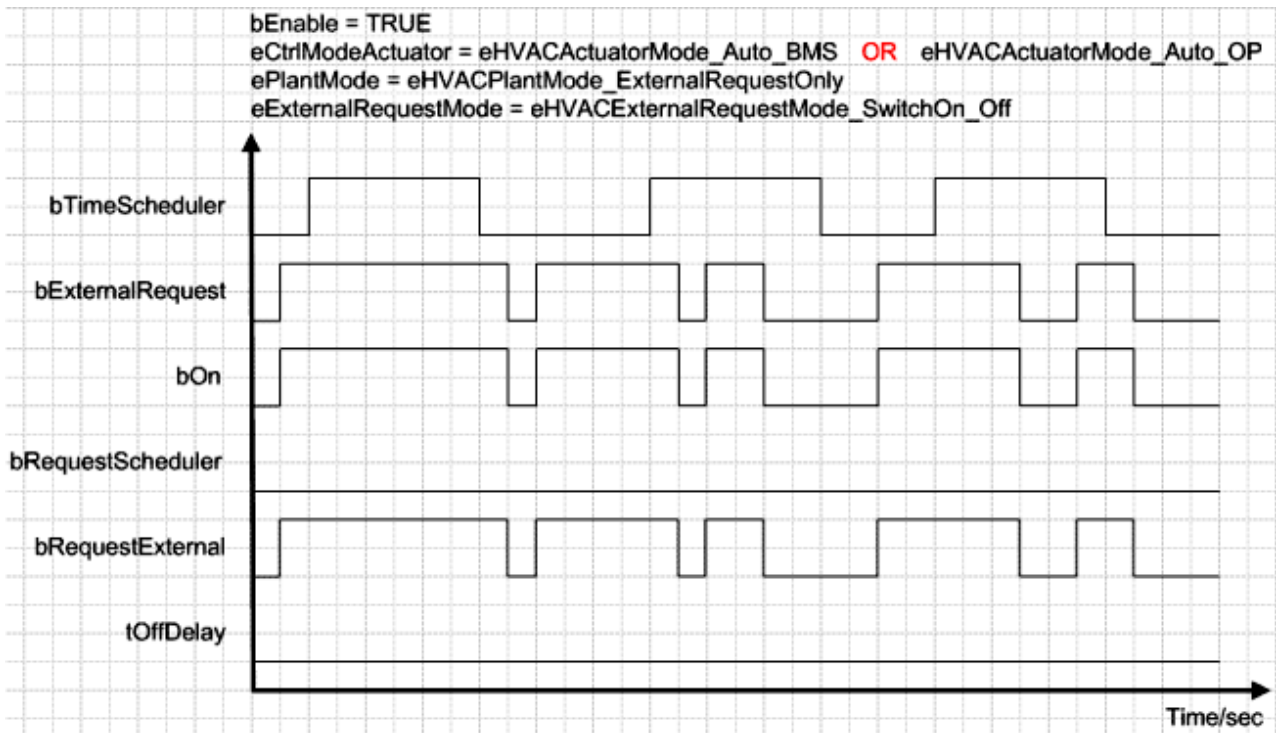
tOffDelay: time value for the extension of the system utilization time. The extension of the utilization time can only be activated if *eModeExternalRequest* = 2. The variable is saved persistently. Preset to 30 min.

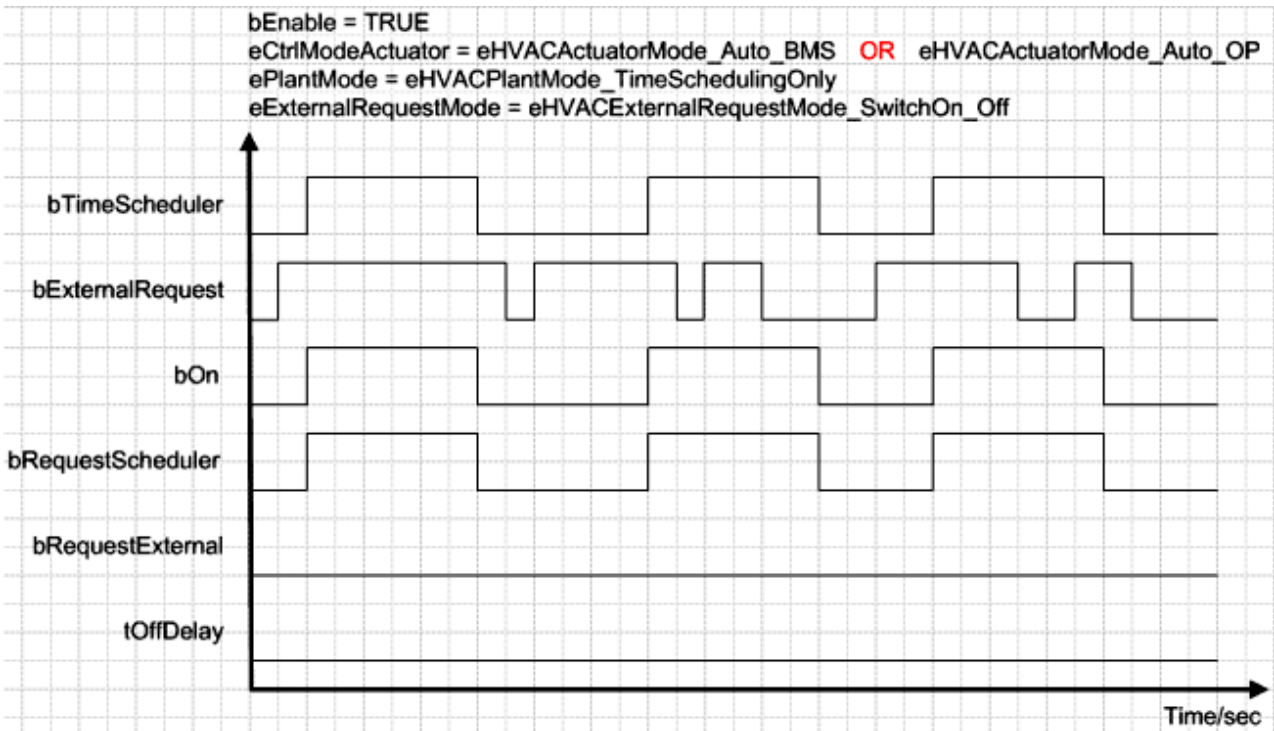
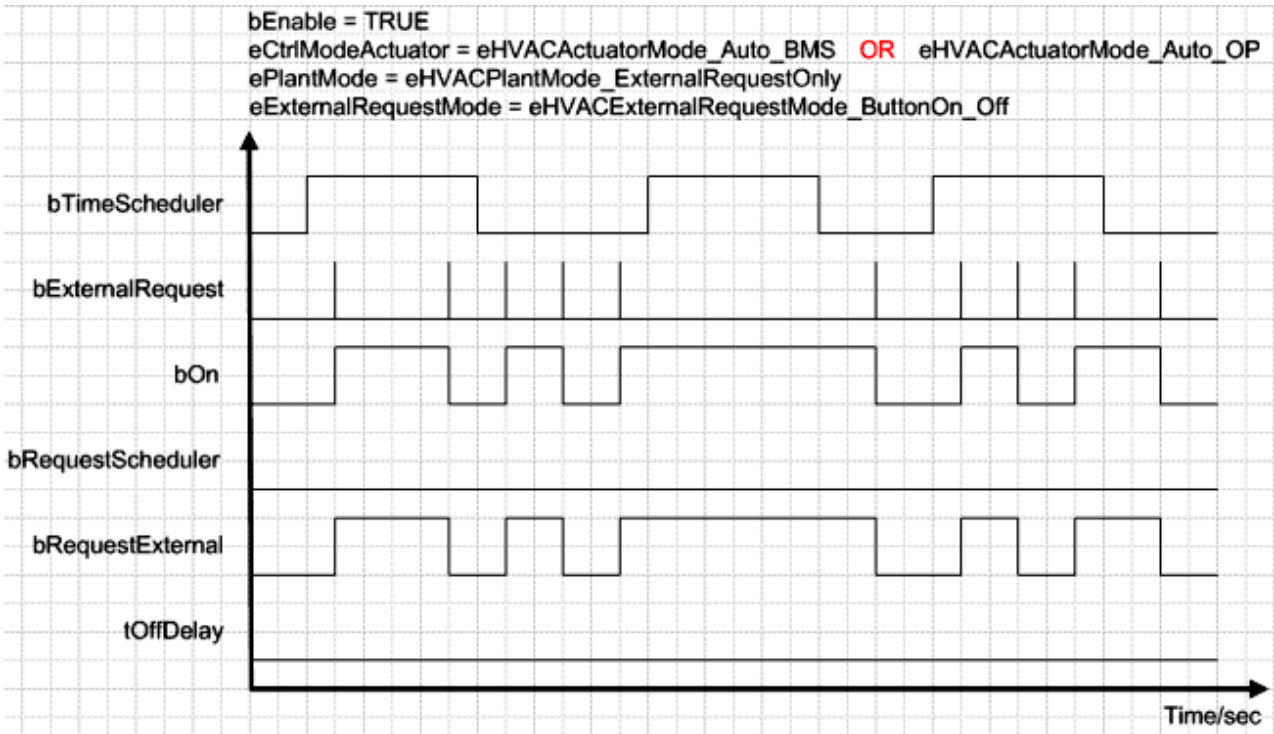
Behavior of the output value

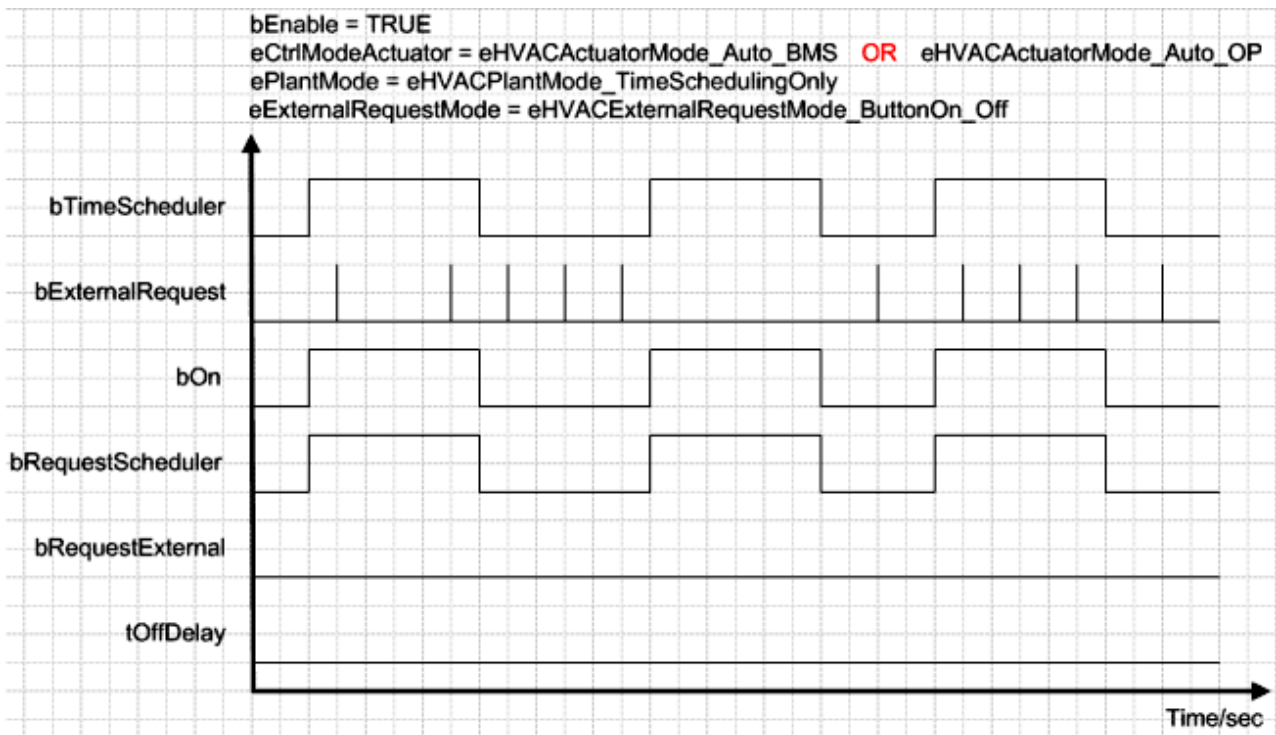
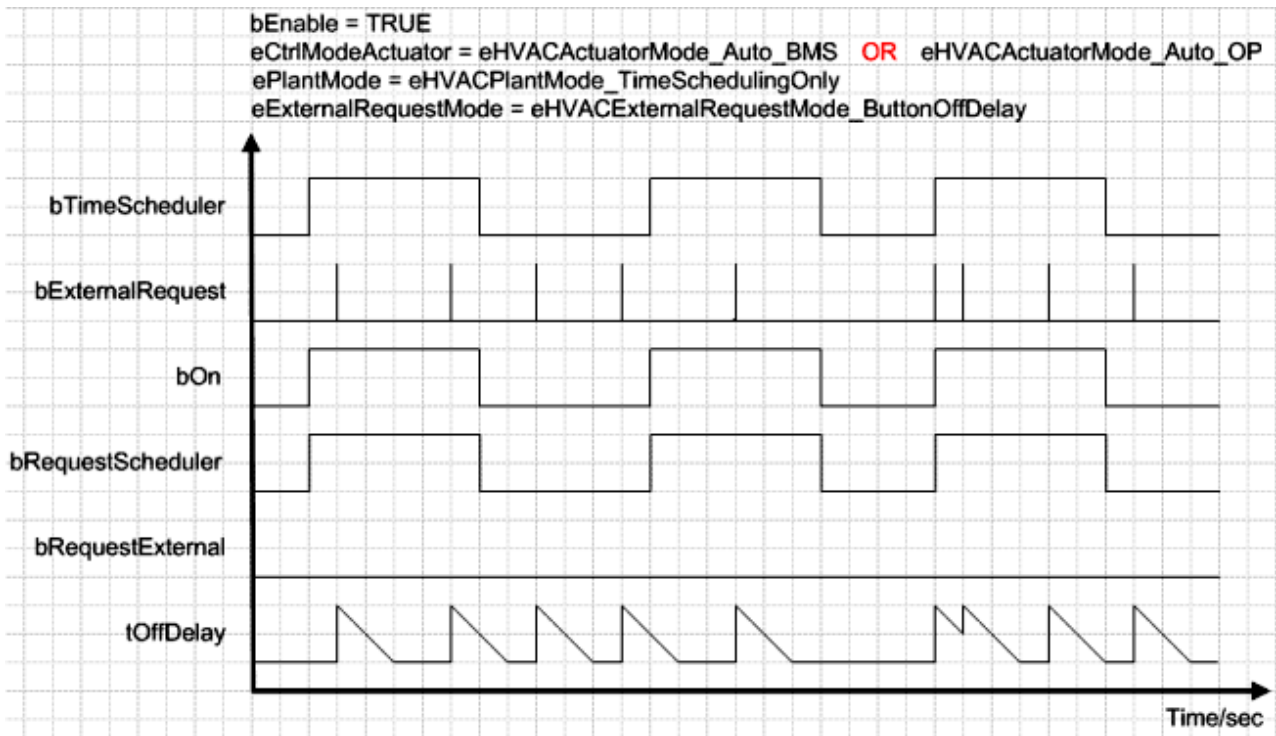








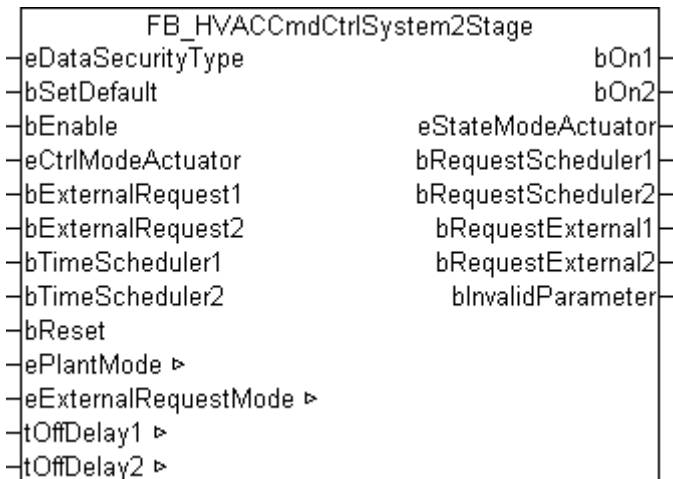




Documents about this

- example_persistent_e.zip (Resources/zip/11659714827.zip)

3.6.8 FB_HVACCmdCtrlSystem2Stage



Application

This function block *FB_HVACCmdCtrlSystem2Stage* is a system switch. It is used, for example, to switch a two-stage ventilation system to automatic or manual operation mode. In automatic operation mode the system can be controlled via a timer program or via the request from a control panel. The function block *FB_HVACCmdCtrlSystem2Stage* is active if the input variable *bEnable* is TRUE and *eCtrlModeActuator* = *eHVACActuatorMode_Auto_BMS* or *eHVACActuatorMode_Auto_OP*.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
eCtrlModeActuator : E_HVACActuatorMode;
bExternalRequest1 : BOOL;
bExternalRequest2 : BOOL;
bTimeScheduler1   : BOOL;
bTimeScheduler2   : BOOL;
bReset            : BOOL;
```

eDataSecurityType: if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block *FB_HVACPersistentDataHandling* must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If *eDataSecurityType* := *eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled if *bEnable* = TRUE. If *bEnable* = FALSE, then *bOn1* and *bOn2* = FALSE.

eCtrlModeActuator: enum that specifies the system operation modes Manual, Auto and Off. In the event of an incorrect entry, operation continues internally with the last valid operating mode. This is *eHVACActuatorMode_Auto_BMS* in the case of initial commissioning. *bInvalidParameter* will be set in the event of an incorrect parameter entry.

bExternalRequest1: external request for the system in stage 1, e.g. from a control panel via a button or switch.

bExternalRequest2: external request for the system in stage 2, e.g. from a control panel via a button or switch.

bTimeScheduler1: request for the system in stage 1 via a timer program.

bTimeScheduler2: request for the system in stage 2 via a timer program.

bReset: acknowledge input in the event of an error.

VAR_OUTPUT

```
bOn1           : BOOL;
bOn2           : BOOL;
eStateModeActuator : E_HVACActuatorMode;
bRequestScheduler1 : BOOL;
bRequestScheduler2 : BOOL;
bRequestExternal1  : BOOL;
bRequestExternal2  : BOOL;
bInvalidParameter : BOOL;
```

bOn1: this output variable enables the system for stage 1.

bOn2: this output variable enables the system for stage 2. If stage 2 is enabled, then stage 1 is automatically also enabled.

eStateModeActuator: Enum via which the state of the operation mode of the motor is fed back to the controller.

bRequestScheduler1: this output indicates that the system is requested by the input variable *bTimeScheduler1*.

bRequestScheduler2: this output indicates that the system is requested by the input variable *bTimeScheduler2*.

bRequestExternal1: this output indicates that the system is requested by the input variable *bExternalRequest1*.

bRequestExternal2: this output indicates that the system is requested by the input variable *bExternalRequest2*.

bInvalidParameter: indicates that an incorrect parameter is present at one of the variables *eCtrlModeActuator*, *ePlantMode* or *eExternalRequestMode*. An incorrect parameter entry does not lead to a standstill of the function block; see description of variables. After rectifying the incorrect parameter entry, the message *bInvalidParameter* must be acknowledged via *bReset*.

VAR_IN_OUT

```
ePlantMode           : E_HVACPlantMode;
eExternalRequestMode : E_HVACExternalRequestMode;
tOffDelay1           : TIME;
tOffDelay2           : TIME;
```

ePlantMode: this enumeration variable can be used to perform various functions of the plant in automatic mode depending on the input variables *bExternalRequest1*, *bExternalRequest2*, *bTimeScheduler1* and *bTimeScheduler2*.

ePlantMode = eHVACPlantMode_TimeSchedulingOnly: the plant is switched on exclusively via the input variables *bTimeScheduler1* or *bTimeScheduler2* into the respective stage.

ePlantMode = eHVACPlantMode_TimeScheduling_And_ExternalRequest: the plant is switched on in stage 1 if the input variables *bTimeScheduler1* AND *bExternalRequest1* = TRUE. If the input variables *bTimeScheduler2* AND *bExternalRequest2* = TRUE, the plant is switched on in stage 2.

ePlantMode = eHVACPlantMode_TimeScheduling_Or_ExternalRequest: the plant is switched on in the respective stage via the input variables *bExternalRequest1*, *bExternalRequest2*, OR *bTimeScheduler1*,

bTimeScheduler2.

ePlantMode = eHVACPlantMode_ExternalRequestOnly: the plant is switched on exclusively via the input variable *bExternalRequest1* or *bExternalRequest2* into the respective stage.

If an incorrect variable value is present, the last valid variable value is used, if available. If there is no valid last value, operation continues with the default value. *blInvalidParameter* will be set in the event of an incorrect parameter entry.

The variable is saved persistently. Preset to 0.

eExternalRequestMode: the enum *eExternalRequestMode* defines the mode of operation of the input variable *bExternalRequest1* and *bExternalRequest2* in automatic mode depending on the enum *ePlantMode*.
eExternalRequestMode = eHVACExternalRequestMode_ButtonOn_Off: the external request is set to TRUE after a rising edge of *bExternalRequest1* or *bExternalRequest2*. Another rising edge resets the request to FALSE again.

eExternalRequestMode = eHVACExternalRequestMode_ButtonOffDelay. The external request extends or sets the utilization time of the plant in the respective stage after a rising edge at the input variable *bExternalRequest1* or *bExternalRequest2* by the set time of *tOffDelay1* or *tOffDelay2*.

eExternalRequestMode = eHVACExternalRequestMode_SwitchOn_Off: the external request is active if *bExternalRequest1* or *bExternalRequest2* = TRUE. It is deactivated if *bExternalRequest1* or *bExternalRequest2* = FALSE.

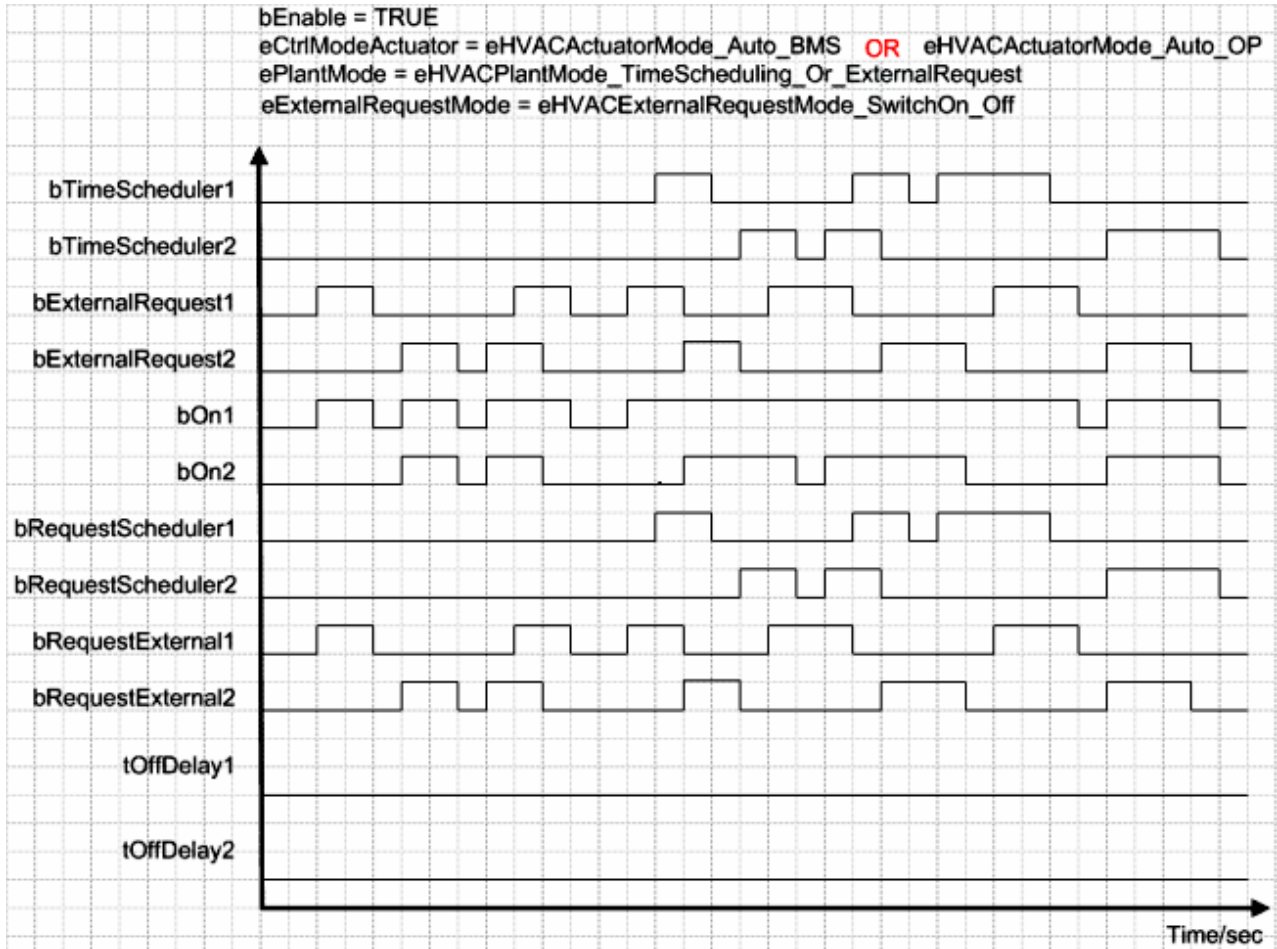
If an incorrect variable value is present, then the last valid variable value is used, if available. If there is no valid last value, operation continues with the default value. *blInvalidParameter* will be set in the event of an incorrect parameter entry.

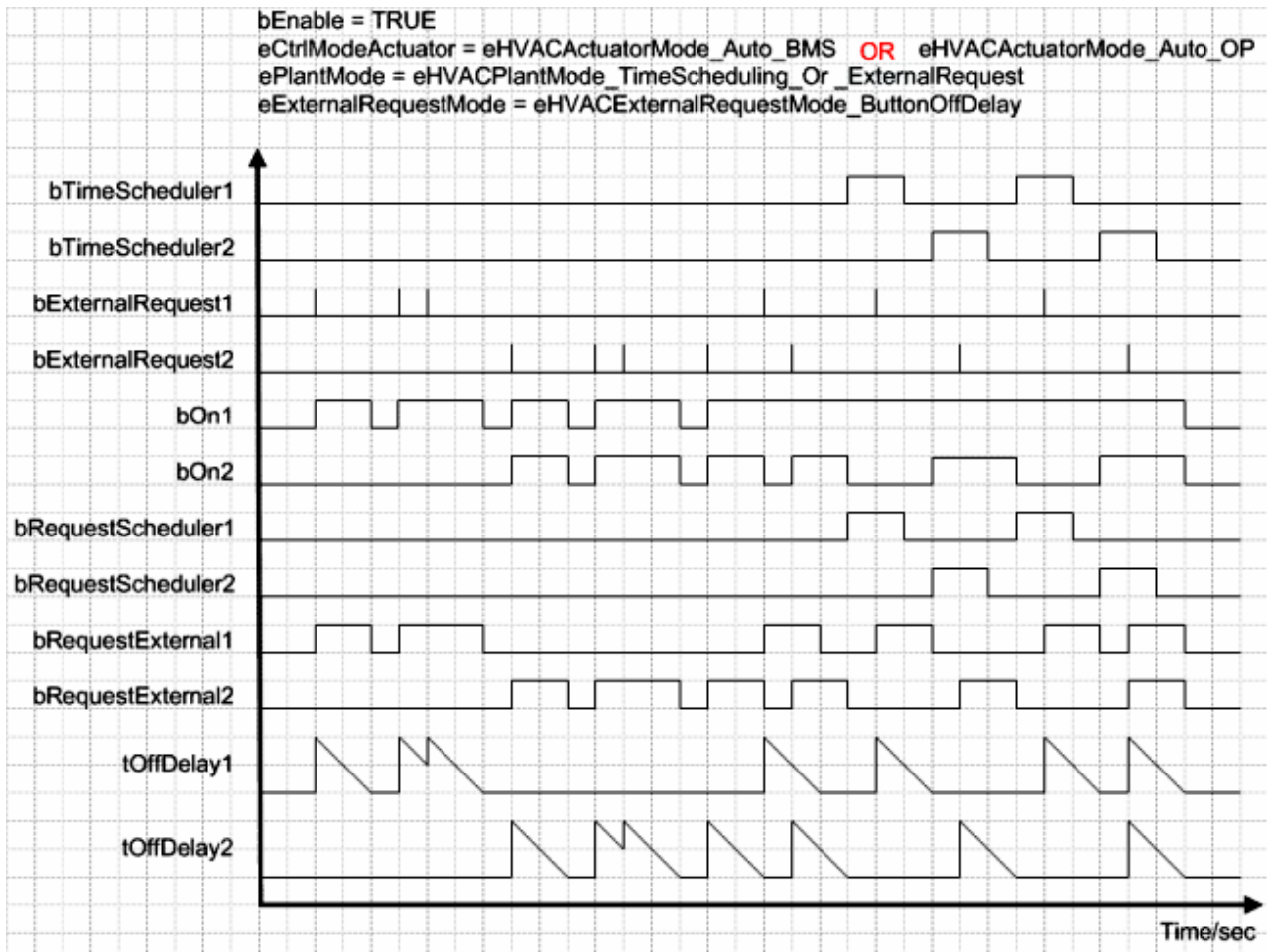
The variable is saved persistently. Preset to 0.

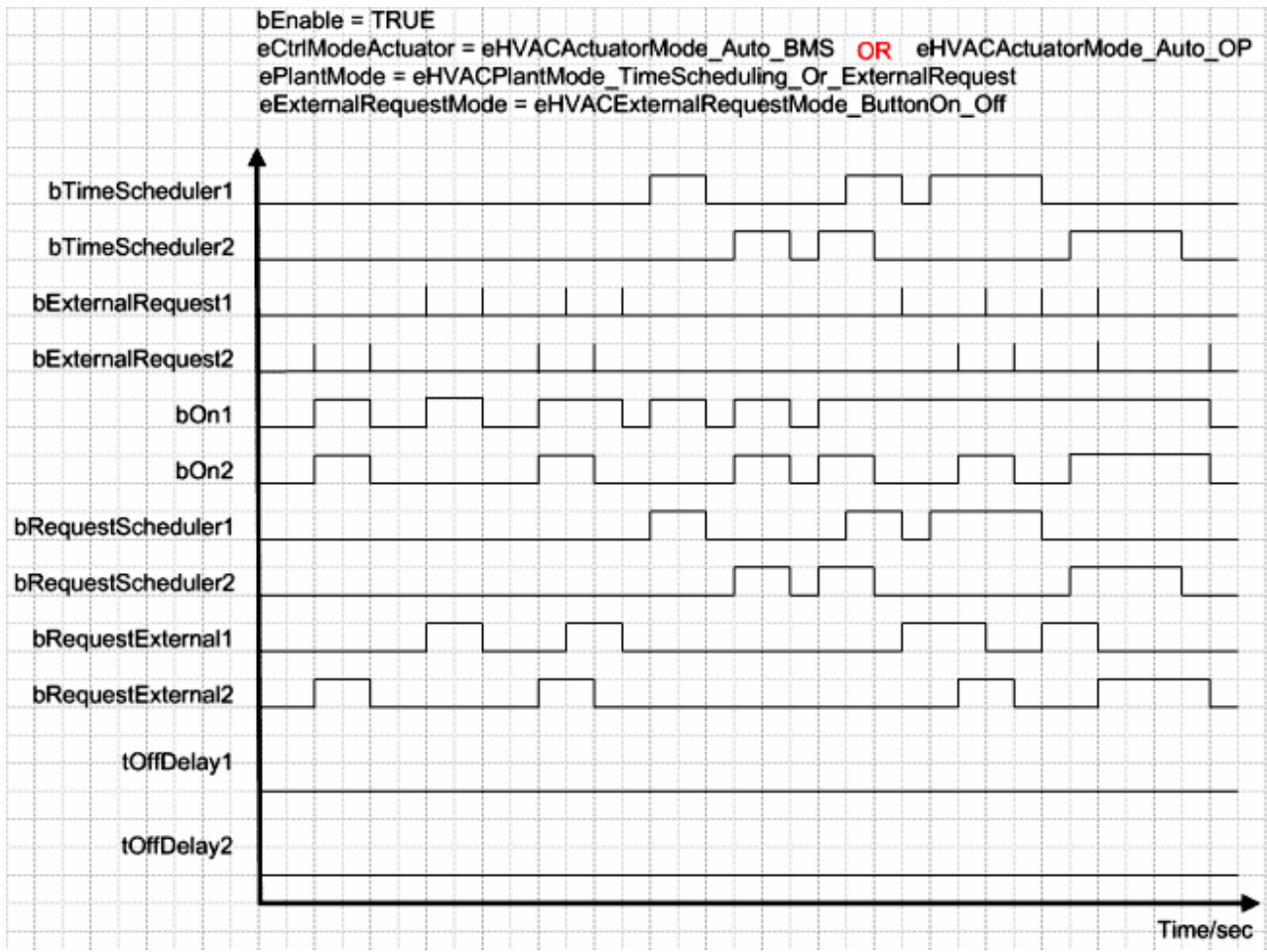
tOffDelay1: time value for the utilization time extension of the plant in stage 1. The utilization time extension can only be activated if *eModeExternalRequest* = 2. The variable is saved persistently. Preset to 30 min.

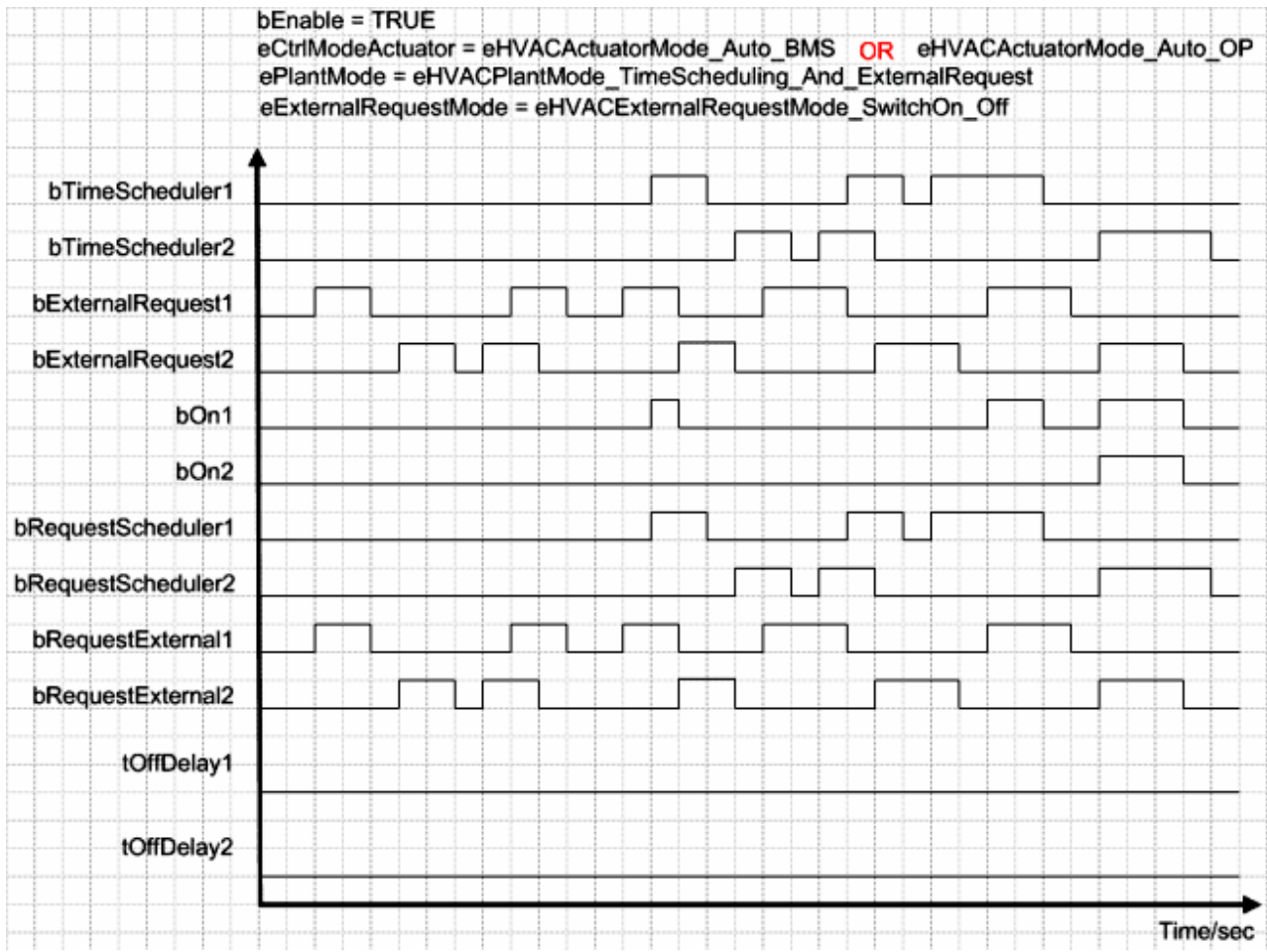
tOffDelay2: time value for the utilization time extension of the plant in stage 2. The utilization time extension can only be activated if *eModeExternalRequest* = 2. The variable is saved persistently. Preset to 30 min.

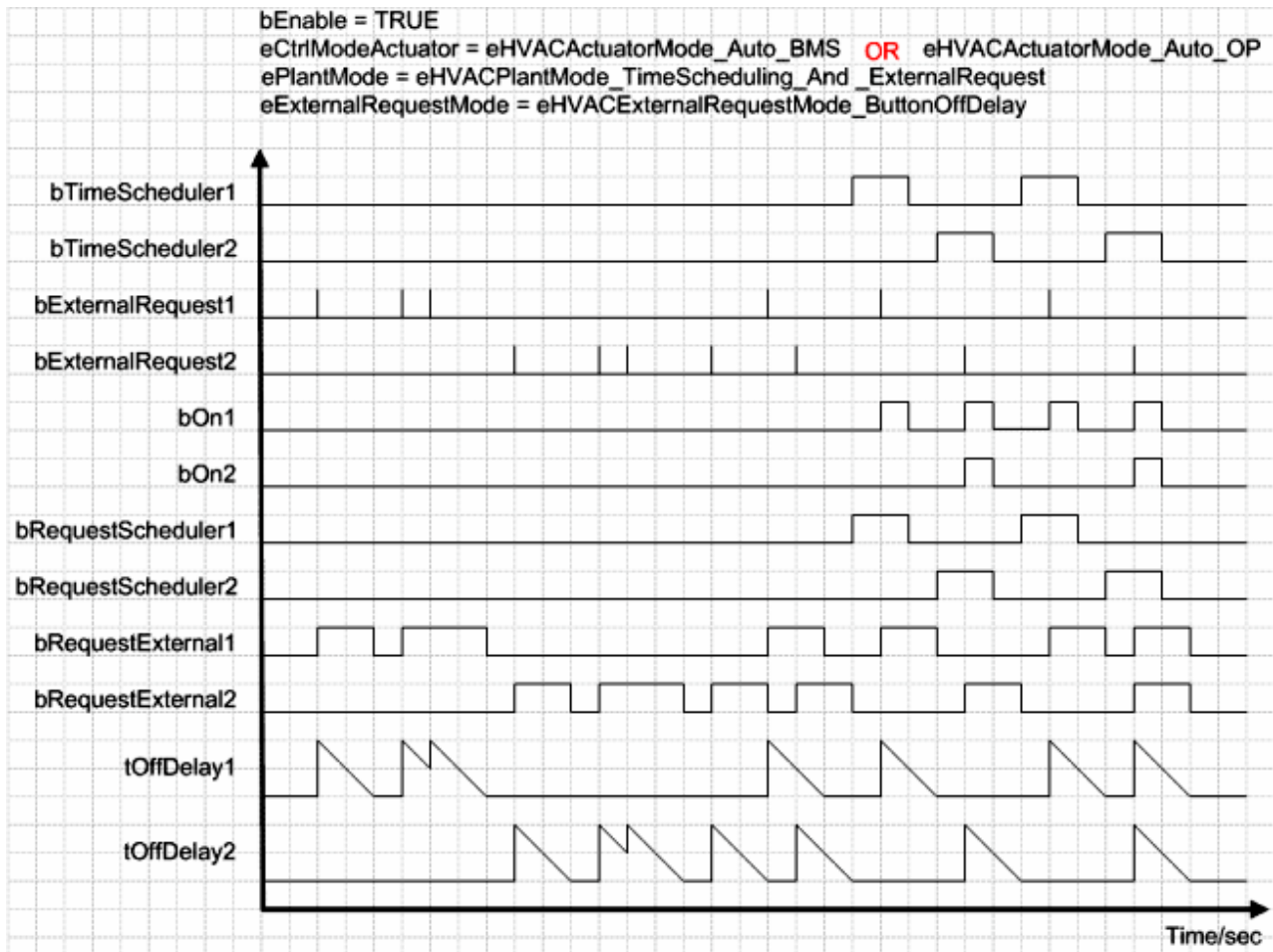
Behavior of the output value

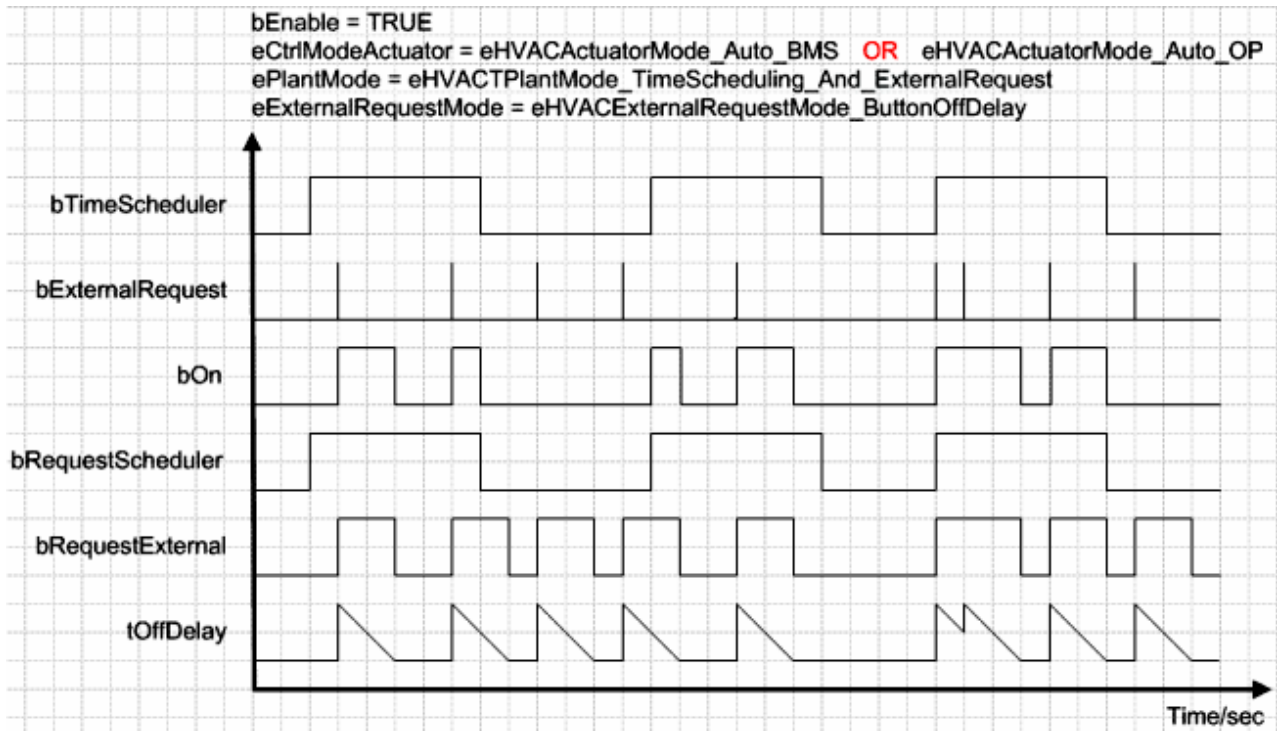
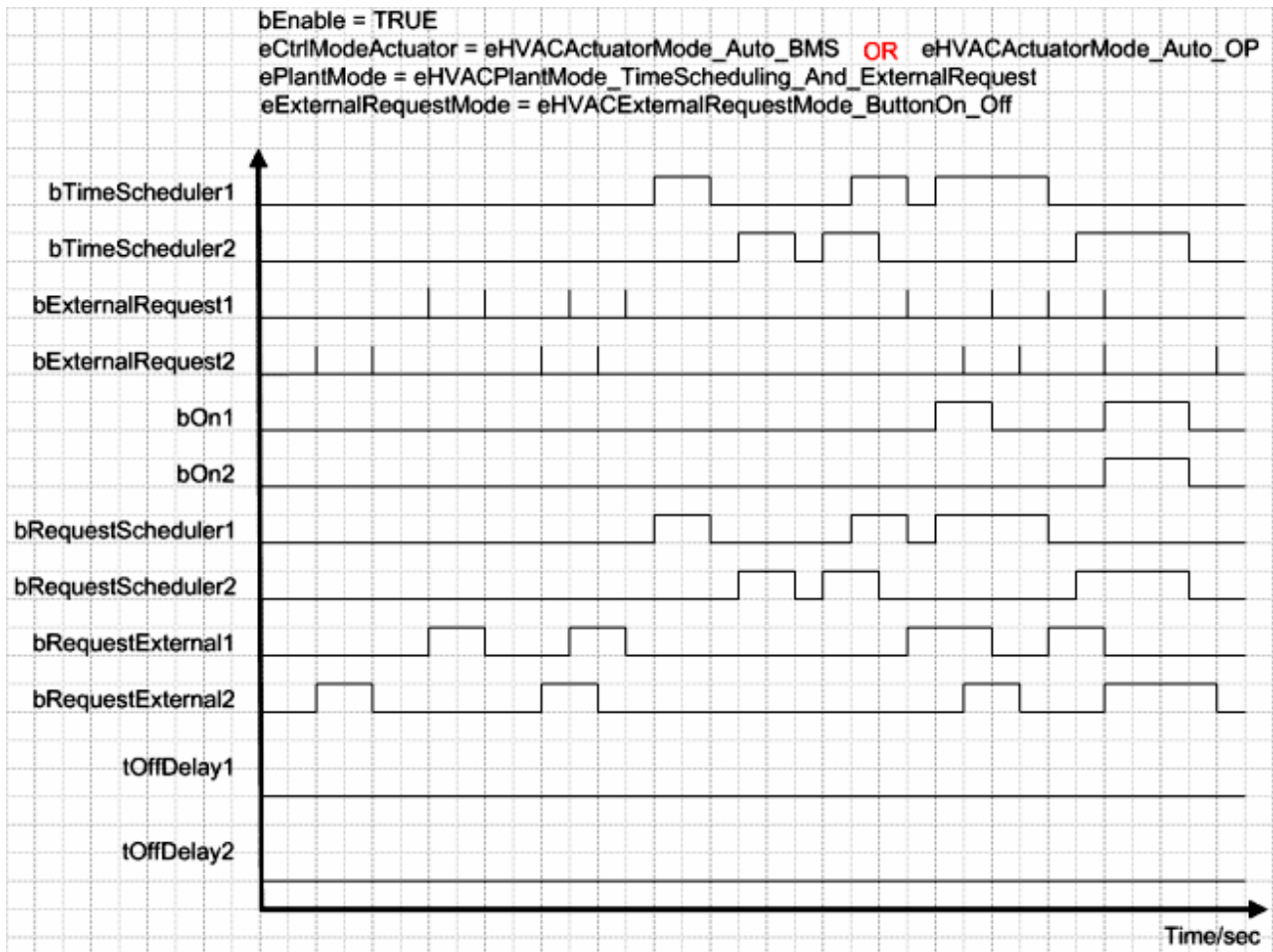


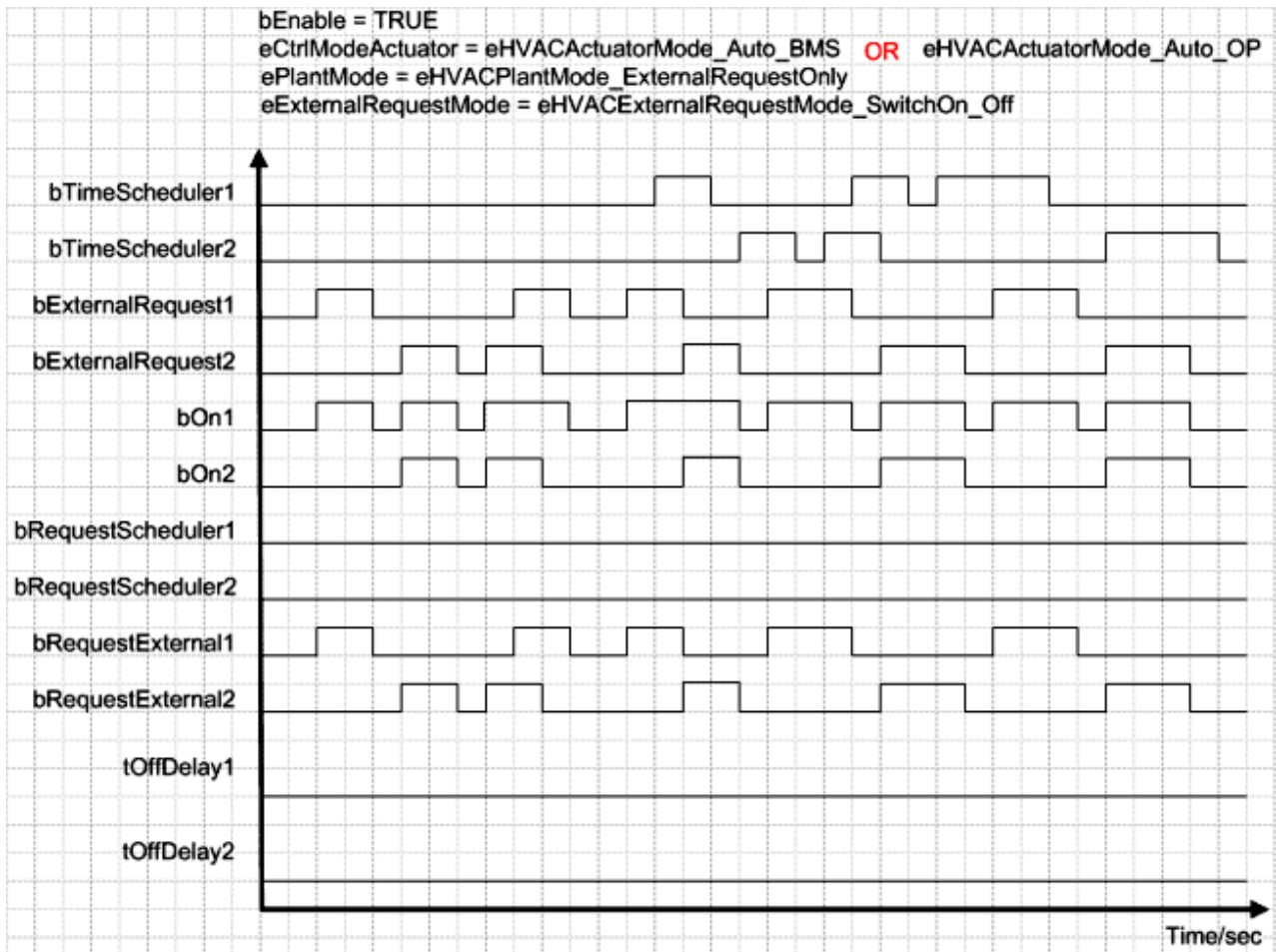


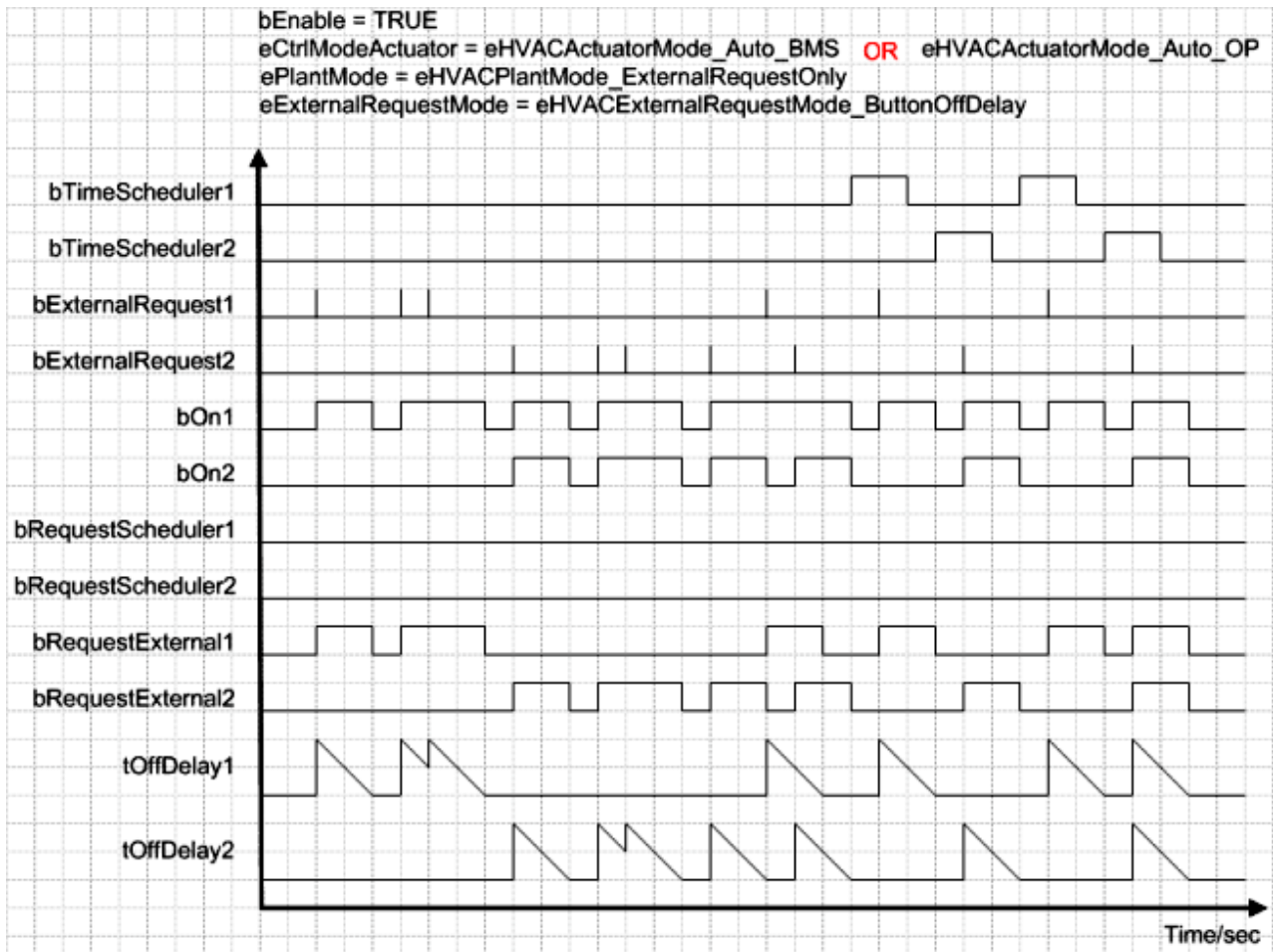


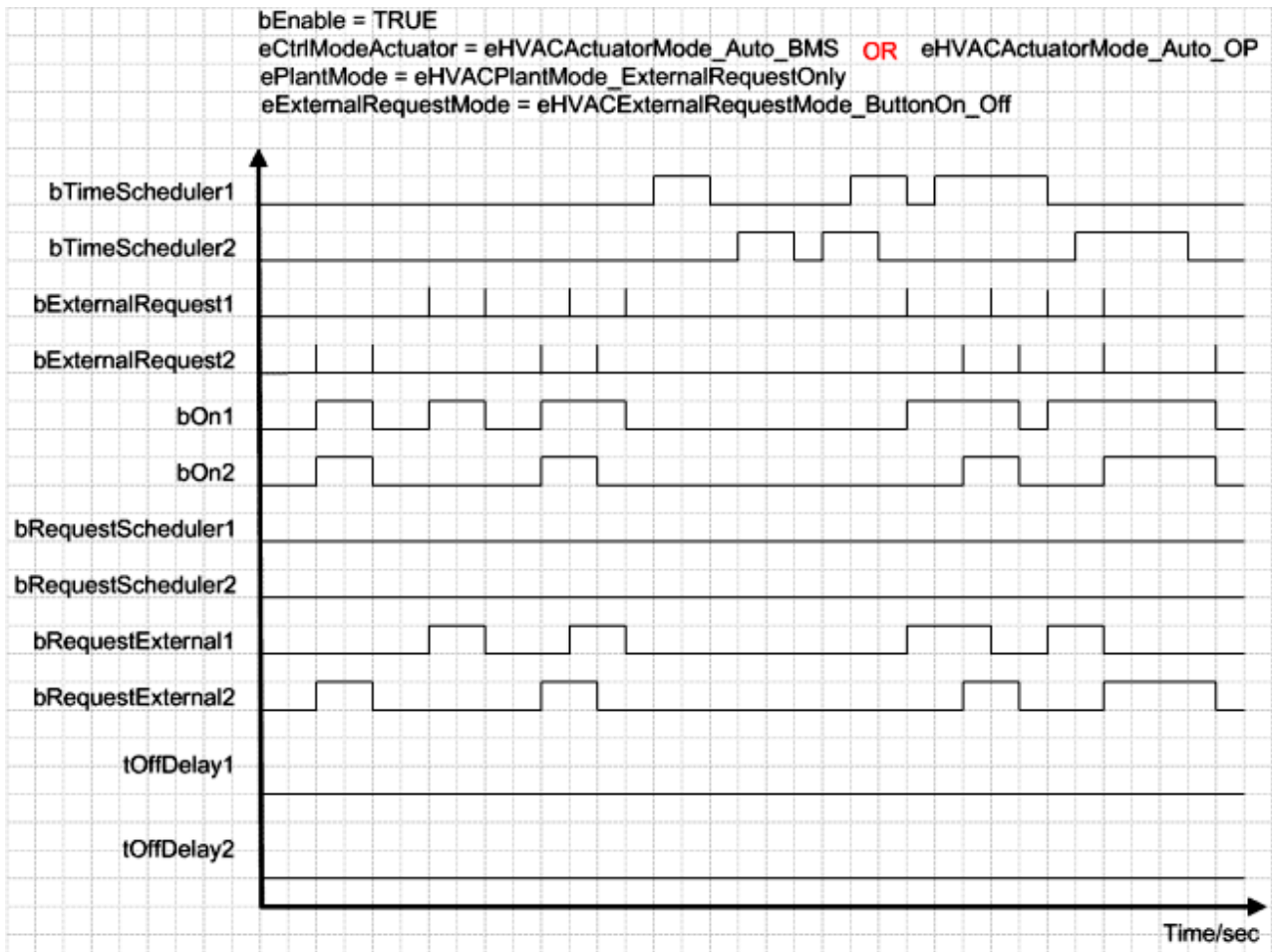


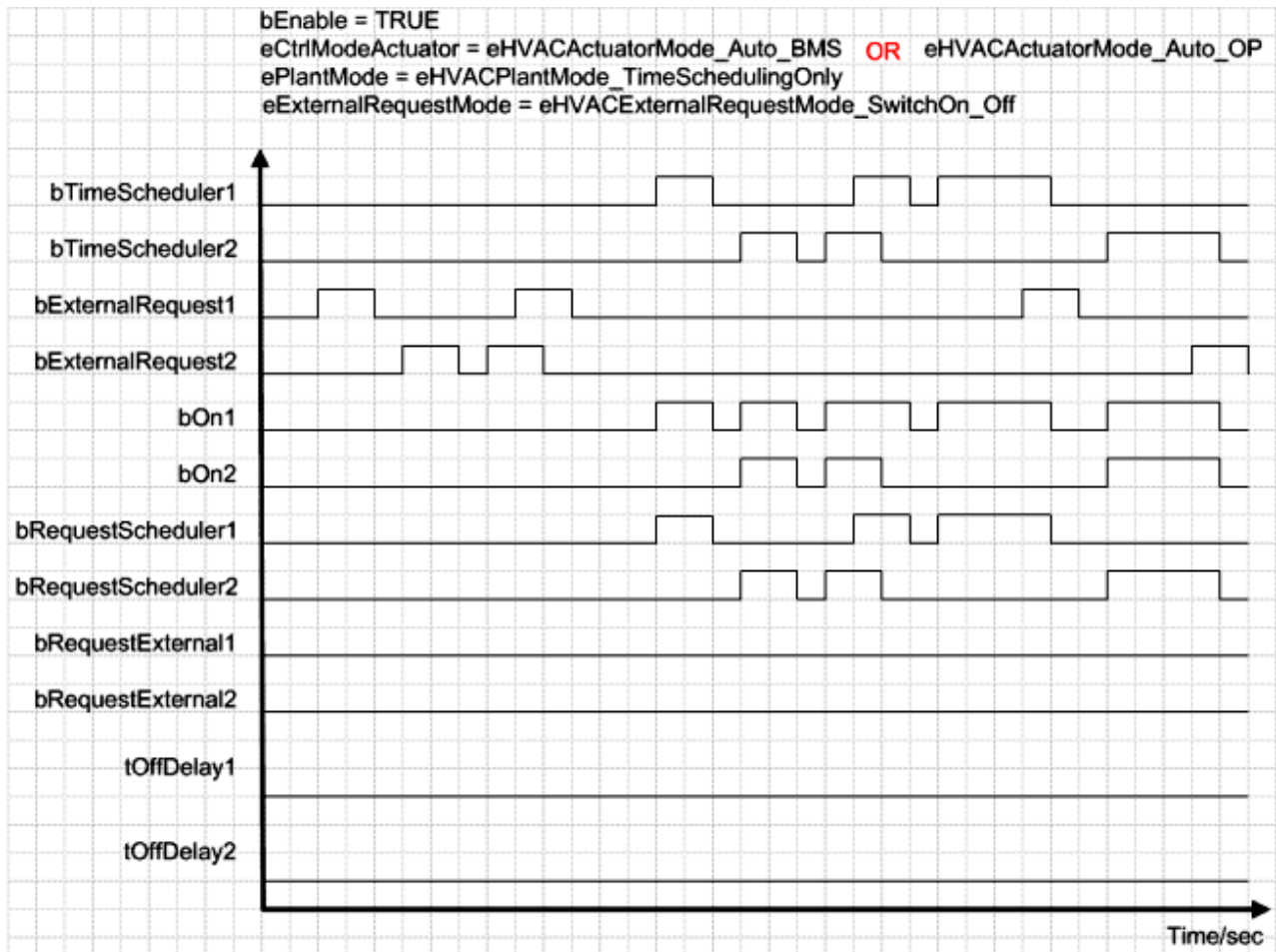


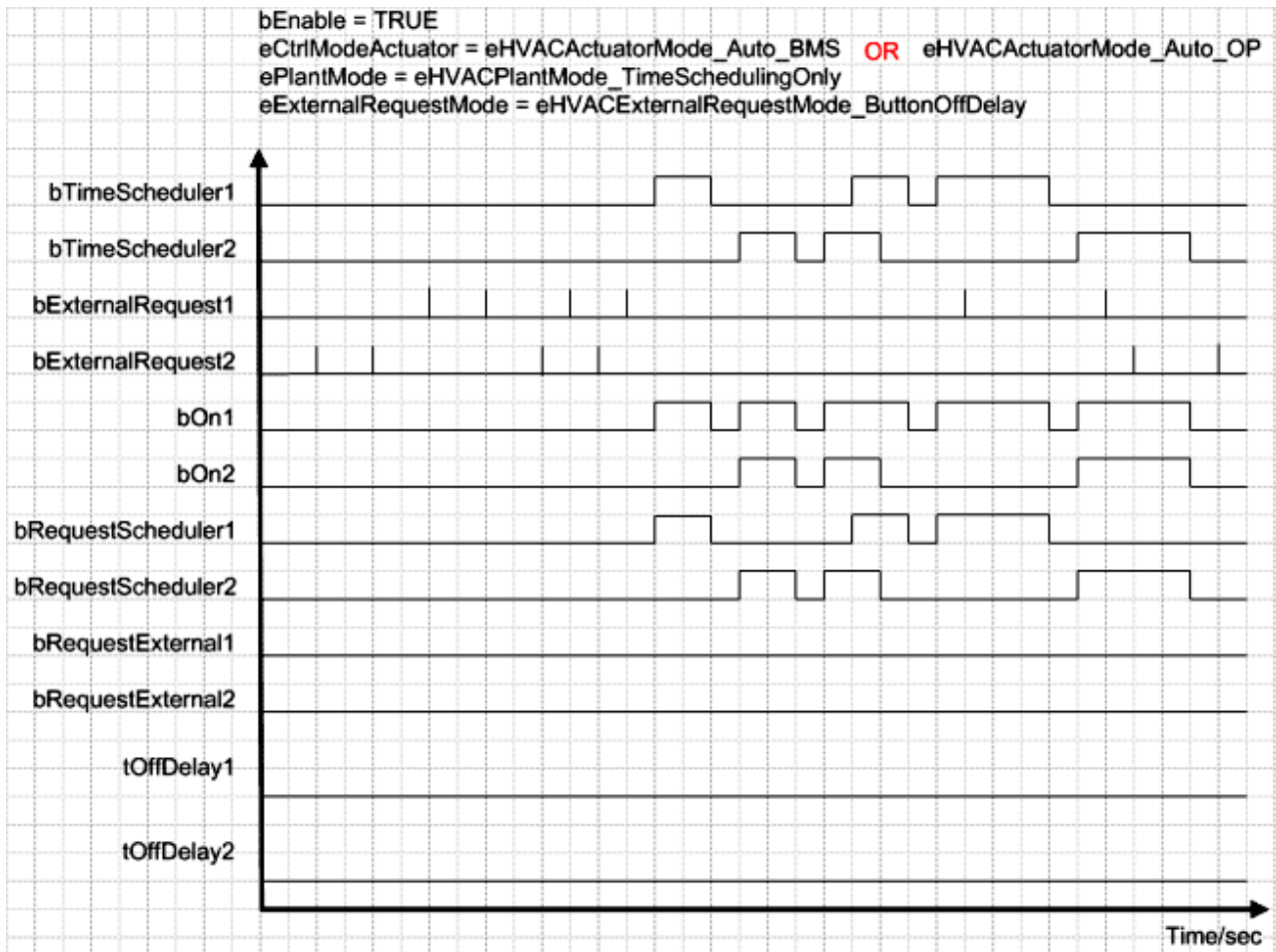


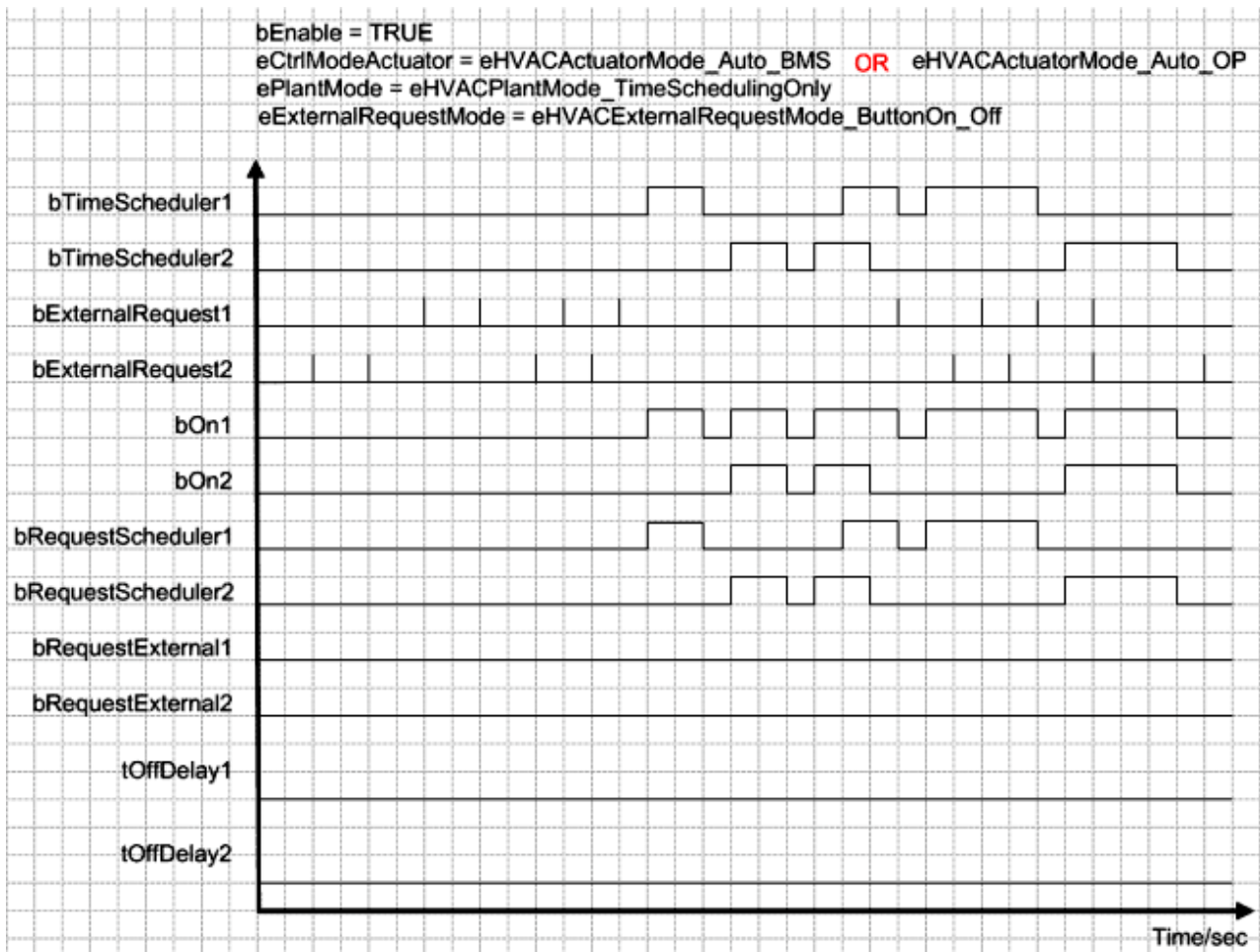








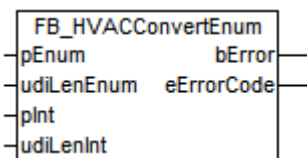




Documents about this

example_persistent_e.zip (Resources/zip/11659714827.zip)

3.6.9 FB_HVACConvertEnum



Application

This function block converts an Enum into an integer value and vice versa. This conversion is particularly suitable for Enums that are used as VAR_IN_OUT variables on function blocks.

VAR_INPUT

```

pEnum      : UDINT;
udiLenEnum : UDINT;
pInt       : UDINT;
udiLenInt  : UDINT;
  
```

pEnum: address of the Enum to be converted. The address is determined with the ADR operator.

udiLenEnum: number of bytes of which the data type Enum consists. The number is determined with the SIZEOF operator.

pInt: address of the integer variable to be converted. The address is determined with the ADR operator.

udiLenInt: number of bytes of which the data type Integer consists. The number is determined with the SIZEOF operator.

VAR_OUTPUT

```
bError      : BOOL;
eErrorCode  : E_HVACErrorCodes;
```

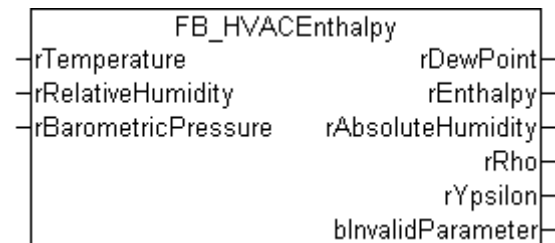
bError: this output indicates with a TRUE that there is an error. (The number of bytes of which the data types Integer or Enum consist is wrong.)

eErrorCode: returns the [error number](#) [► 520] when the *bError* output is set. The following errors can occur in this function block: *eHVACErrorCodes_Error_LEN_Int* (43), *eHVACErrorCodes_Error_LEN_Enum* (44)

Application example

Download	Required library
https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659726219/.zip	TcHVAC.lib

3.6.10 FB_HVACEnthalpy



Application

This function block is used to calculate the dew point, the specific enthalpy and the absolute humidity. The temperature, the relative humidity and the barometric air pressure are required for the calculation of these parameters.

Enthalpy is a measure of the energy of a thermodynamic system.

VAR_INPUT

```
rTemperature      : REAL;      [ °C ]
rRelativeHumidity : REAL;      [ % ]      1..100
rBarometricPressure : REAL;    [ hPa ]
```

rTemperature: input for the temperature value, in degrees Celsius.

rRelativeHumidity: input for relative humidity, in percent. The value must equal or greater 1.

rBarometricPressure: input for air pressure, in hectopascal.

VAR_OUTPUT

```
rDewPoint      : REAL;      [ °C ]
rEnthalpy      : REAL;      [ kJ/kg ]
rAbsoluteHumidity : REAL;    [ kg/kg ]
rRho           : REAL;      [ kg/m³ ]
rYpsilon       : REAL;      [ m³/kg ]
bInvalidParameter : BOOL;
```

rDewPoint: DEW point [°C].

rEnthalpy: enthalpy [kJ / kg].

rAbsoluteHumidity: absolute humidity [kg/kg]. The result must be multiplied by 1000 in order to obtain the absolute humidity in [g/kg].

rRho: density of humid air, showing the weight of the mass of the mixture in relation to volume. The unit is [kg mixture / m³].

rYpsion: specific volume per 1 kg of dry air. The unit is [m³ / kg dry air].

InvalidParameter: set to TRUE if the input variable *rRelativeHumidity* or *rBarometricPressure* is smaller than or equal to zero.

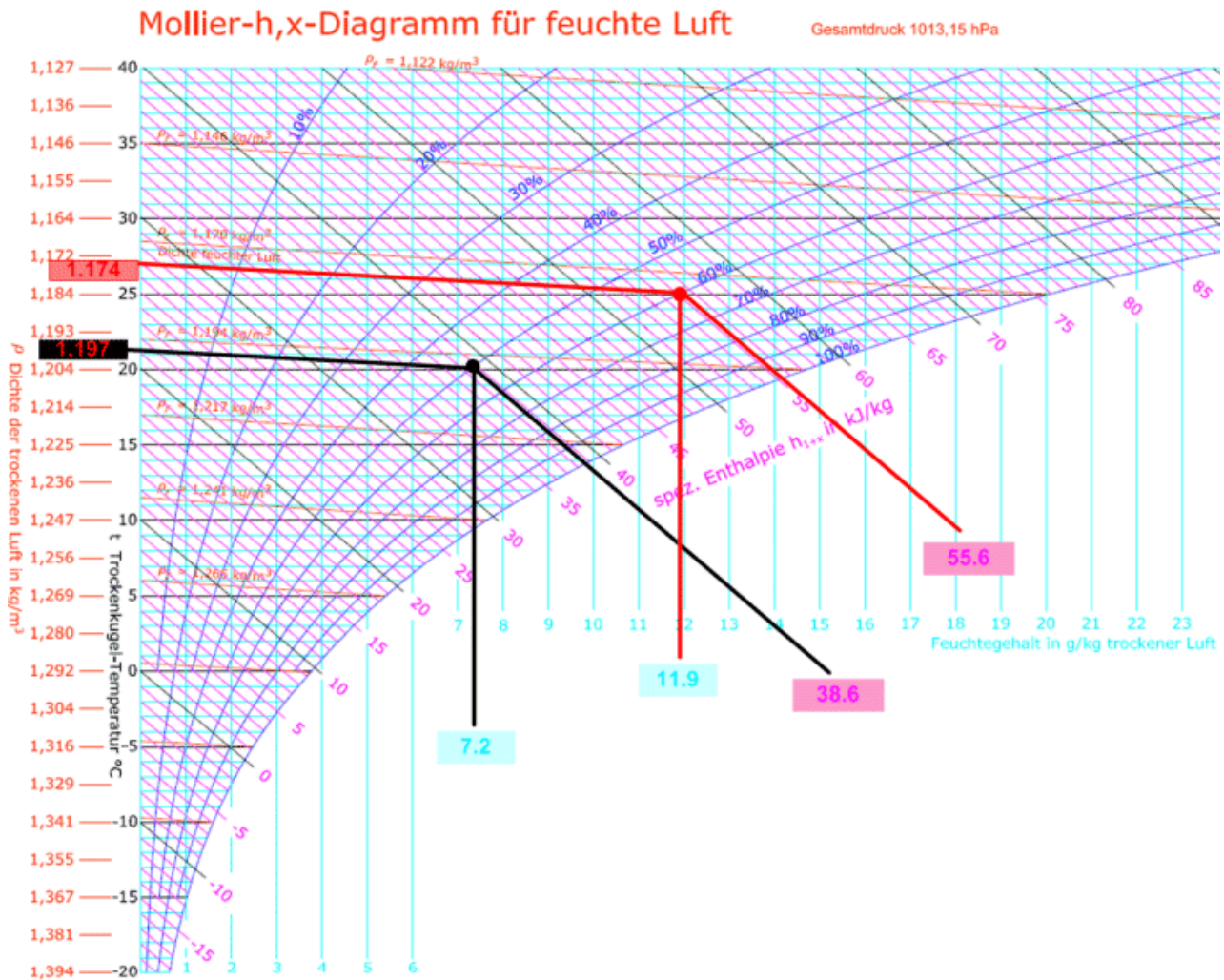


Fig. 1 h,x diagram [www.air2000.de]

Example 1:

rTemperature:= 20 °C
 rRelativeHumidity:= 50 %
 rBarometricPressure:= 1013.15 hPa

Results from function block: rDewPoint:= 9.4 °C
 rEnthalpy:= 38.6 kJ/kg
 rAbsoluteHumidity:= 0.0072 kg/kg , converted to g/kg ==> 7.2 g/kg

Example 2:

rTemperature:= 25 °C
 rRelativeHumidity:= 60 %
 rBarometricPressure:= 1013.15 hPa

Results from function block: rDewPoint:= 17 °C
 rEnthalpy:= 55.6 kJ/kg
 rAbsoluteHumidity:= 0.0119 kg/kg , converted to g/kg ==> 11.9 g/kg

3.6.11 FB_HVACFixedLimit



Application


This function block represents a limit switch, whose output signal is switched depending on the time delays `tDelayHighLimit` / `tDelayLowLimit`, the mode `bModeFixedLimit`, the limit values `rHighLimit` / `rLowLimit` and the input signal `rInputValue`.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable          : BOOL;
rInputValue      : REAL;
bReset           : BOOL;
```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: the default values of the VAR_IN_OUT variables are adopted with a rising edge at this input variable.

bEnable: the function block is enabled if `bEnable = TRUE`.

rInputValue: input signal

bReset: input for acknowledging an incorrect parameter entry via a rising edge.

VAR_OUTPUT

```
bEvent           : BOOL;
bEdgeHighLimit   : BOOL;
bEdgeLowLimit    : BOOL;
bInvalidParameter: BOOL;
```

bEvent: output signal which is switched depending on the time delays `tDelayHighLimit` / `tDelayLowLimit`, the mode `bModeFixedLimit`, the input signal `rInputValue` and the limit values `rHighLimit` / `rLowLimit` is switched.

bEdgeHighLimit: is TRUE for one PLC cycle after the input signal *rInputValue* has exceeded the variable *rHighLimit* for the duration of *tDelayHighLimit*.

bEdgeLowLimit: is TRUE for one PLC cycle after the input signal *rInputValue* has undershot the variable *rLowLimit* for the duration of *tDelayLowLimit*.

blInvalidParameter: indicates that an incorrect parameter is present at one of the variables *rHighLimit*, *rLowLimit*, *tDelayHighLimit* or *tDelayLowLimit*. An incorrect parameter entry does not lead to a standstill of the function block; see description of variables. After rectifying the incorrect parameter entry, the message *blInvalidParameter* must be acknowledged via *bReset*.

VAR_IN_OUT

```
bModeFixedLimit : BOOL;
rHighLimit      : REAL;
rLowLimit       : REAL;
tDelayHighLimit : TIME;
tDelayLowLimit  : TIME;
```

bModeFixedLimit: if *bModeFixedLimit* = TRUE, then the output variable *bEvent* becomes TRUE, if the input signal *rInputValue* has exceeded the variable *rHighLimit* for the duration of *tDelayHighLimit*. If *bEvent* = TRUE, then *bEdgeHighLimit* is set for one PLC cycle. If the input signal *rInputValue* falls below the variable *rLowLimit* for the duration of *tDelayLowLimit*, then *bEvent* becomes FALSE and *bEdgeLowLimit* is set for one PLC cycle.

If *bModeFixedLimit* = FALSE, then the output variable *bEvent* becomes TRUE if the input signal *rInputValue* has fallen below the variable *rLowLimit* for the duration of *tDelayLowLimit*. If *bEvent* = TRUE, then *bEdgeLowLimit* is set for one PLC cycle. If the input signal *rInputValue* exceeds the variable *rHighLimit* for the duration of *tDelayHighLimit*, then *bEvent* becomes FALSE and *bEdgeHighLimit* is set for one PLC cycle.

The variable is saved persistently. Preset to 1.

rHighLimit: upper limit value.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *blInvalidParameter* will be set in the event of an incorrect parameter entry. The function block continues to operate normally.

The variable is saved persistently. Preset to 22.

rLowLimit: lower limit value.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *blInvalidParameter* will be set in the event of an incorrect parameter entry. The function block continues to operate normally.

The variable is saved persistently. Preset to 18.

tDelayHighLimit: switching delay [s] when the upper limit value is exceeded.

If an incorrect variable value is present, the last valid variable value is used, if available. If there is no valid last value, then the default value is used. *blInvalidParameter* will be set in the event of an incorrect parameter entry. The function block continues to operate normally.

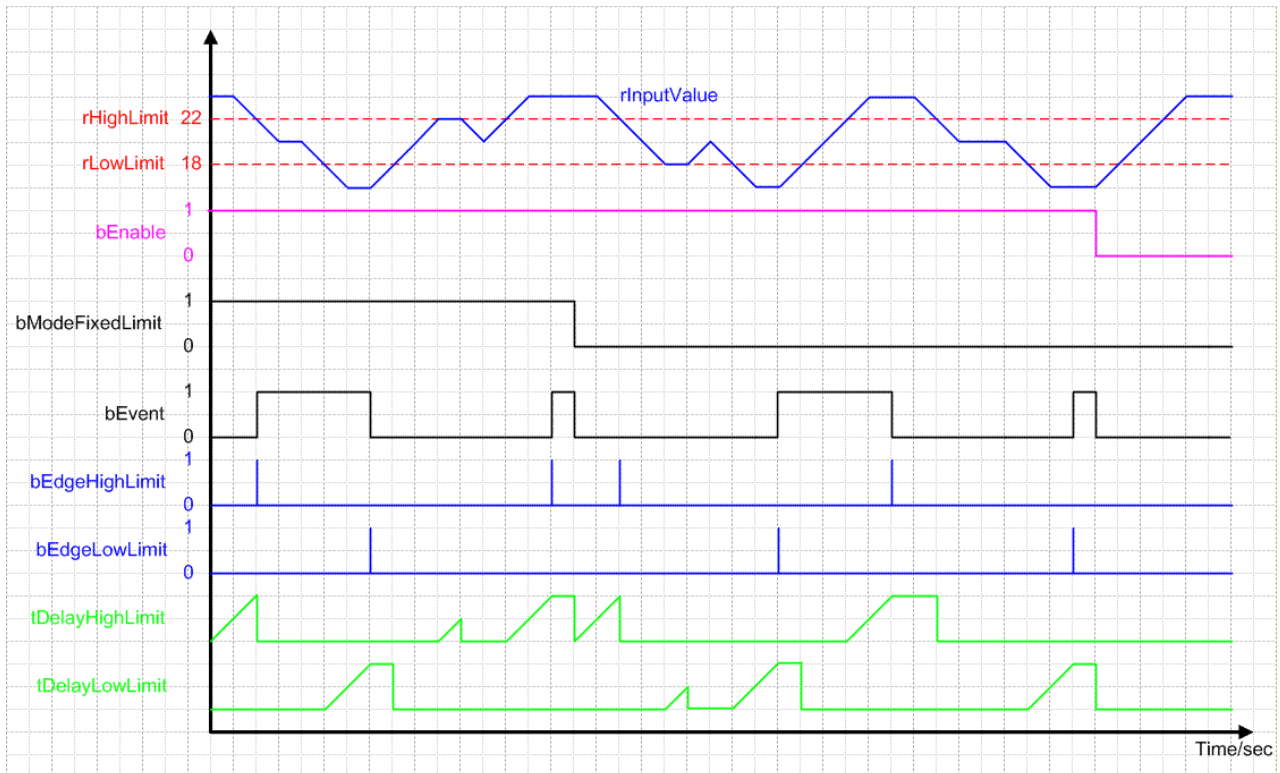
The variable is saved persistently. Preset to 1 s.

tDelayLowLimit: switching delay [s] when falling below the lower limit value.

If an incorrect variable value is present, the last valid variable value is used, if available. If there is no valid last value, then the default value is used. *blInvalidParameter* will be set in the event of an incorrect parameter entry. The function block continues to operate normally.

The variable is saved persistently. Preset to 1 s.

Behavior of the output value



Documents about this

example_persistent_e.zip (Resources/zip/11659714827.zip)

3.6.12 FB_HVACFreezeProtectionHeater

FB_HVACFreezeProtectionHeater	
eDataSecurityType	byState
bSetDefault	bFrost ProtecRunning
bThermostat	bHardwareReset
rReturnWaterTemp	bErrorGeneral
bCtrlVoltage	bAlarmFrost
bReset	bAlarmReturnWaterTemp
rReturnWaterAlarmLimit ▾	bAlarmThermostat
rReturnWaterNormalLimit ▾	byError
tDelayHardwareReset ▾	blInvalidParameter
tMonitoringAfterReset ▾	

Application

This function block is for the frost protection monitoring of an air heater in an air conditioning system. Frost protection monitoring takes place both on the air side via a frost protection thermostat and on the water side via a temperature sensor in the heater return flow.

The frost protection program becomes active when the return temperature of the air heater *rReturnWaterTemp* is lower than the limit value *rReturnWaterAlarmLimit* or the input *bThermostat* is FALSE = fault.

For each of these faults, the respective message *bAlarmReturnWaterTemp* and/or *bAlarmThermostat* is sent to the output of the function block TRUE. Similarly, the output *bAlarmFrost* becomes TRUE and the activation of the frost protection program is indicated by a TRUE at the output variable *bFrostProtectionRunning*.

In order to avoid the air heater freezing up when the frost protection is activated, the heater pump must be switched on and the heater valve compulsorily opened. In many systems this function is implemented not only on the software side, but for reasons of safety on the hardware side also by means of a relay circuit in the control cabinet. In the normal case the frost protection relay is self-latched, since the frost protection circuit is based on the quiescent current principle. Following an error the frost protection relay is normally brought back to its self-latched state by a reset button on the control cabinet. This function is implemented automatically by means of a short pulse at the output *bHardwareReset*. The condition for this is that the frost protection thermostat has returned to its normal state, the temperature in the return flow of the heater has reached the value of *rReturnWaterNormalLimit* and the time *tDelayFirstWarning* has elapsed.

After the system has restarted after an automatic acknowledgement, the timer *tMonitoringAfterReset* is started.


If another frost warning occurs within this time, the output *bAlarmFrost* becomes TRUE again. It is only possible to reset the frost alarm by means of a rising edge at the input *bReset*.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bThermostat      : BOOL;
rReturnWaterTemp  : REAL;
bCtrlVoltage     : BOOL;
bReset           : BOOL;
```

eDataSecurityType: if *eDataSecurityType:= eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If *eDataSecurityType:= eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if *eDataSecurityType:= eHVACDataSecurityType_Persistent*. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bThermostat: signal from the frost protection thermostat. In the normal case TRUE.

rReturnWaterTemp: water temperature in the return flow of the air heater.

bCtrlVoltage: control voltage monitoring. If control voltage is present *bCtrlVoltage* = TRUE.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
byState          : BYTE;
bFrost ProtecRunning : BOOL;
bHardwareReset   : BOOL;
bErrorGeneral    : BOOL;
bAlarmFrost      : BOOL;
bAlarmReturnWaterTemp: BOOL;
bAlarmThermostat : BOOL;
byError          : BYTE;
bInvalidParameter : BOOL;
```

byState: output of the state as a byte.

```

byState.0 := bFrost ProtecRunning;
byState.1 := bHardwareReset;
byState.2 := bThermostat;
byState.7 := bCtrlVoltage;

```

bFrost ProtecRunning: TRUE, if the frost protection program is active.

bHardwareReset: output that indicates with a 1s pulse that the frost protection relay is latched again following an error.

bErrorGeneral: there is a general error.

bAlarmFrost: TRUE if there is a frost warning.

bAlarmReturnWaterTemp: the water temperature limit in the return flow of the air heater was undershot.

bAlarmThermostat: TRUE if there is an error in the frost protection thermostat.

byError: output of the error as a byte.

```

byError.1 := bInvalidParameter;
byError.2 := bErrorGeneral;
byError.3 := bAlarmFrost;
byError.4 := bAlarmReturnWaterTemp;
byError.5 := bAlarmThermostat;

```

bInvalidParameter: Indicates that an incorrect input parameter is present. *bInvalidParameter* must be acknowledged with *bReset*.

VAR_IN_OUT

```

rReturnWaterAlarmLimit : REAL;
rReturnWaterNormalLimit : REAL;
tDelayHardwareReset : TIME;
tMonitoringAfterReset : TIME;

```

rReturnWaterAlarmLimit: the frost alarm is activated below the temperature value *rReturnWaterAlarmLimit* in the return flow of the air heater (0°C..22°C). The variable is saved persistently.

rReturnWaterNormalLimit: the air heater returns to its normal state when the temperature is once again above *bReturnWaterNormalLimit* following the temperature having been undershot (0 °C..70 °C). The variable is saved persistently.

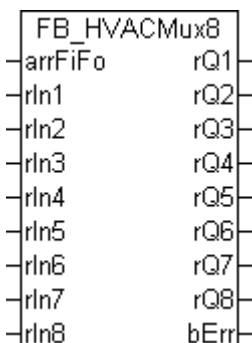
tDelayHardwareReset: time that elapses, after input, between the input of the frost error and output of an acknowledgement pulse in order to return the system to normal operation. The variable is saved persistently.

tMonitoringAfterReset: monitoring time after the first frost warning. The variable is saved persistently.

Documents about this

📄 example_persistent_e.zip (Resources/zip/11659714827.zip)

3.6.13 FB_HVACMux8

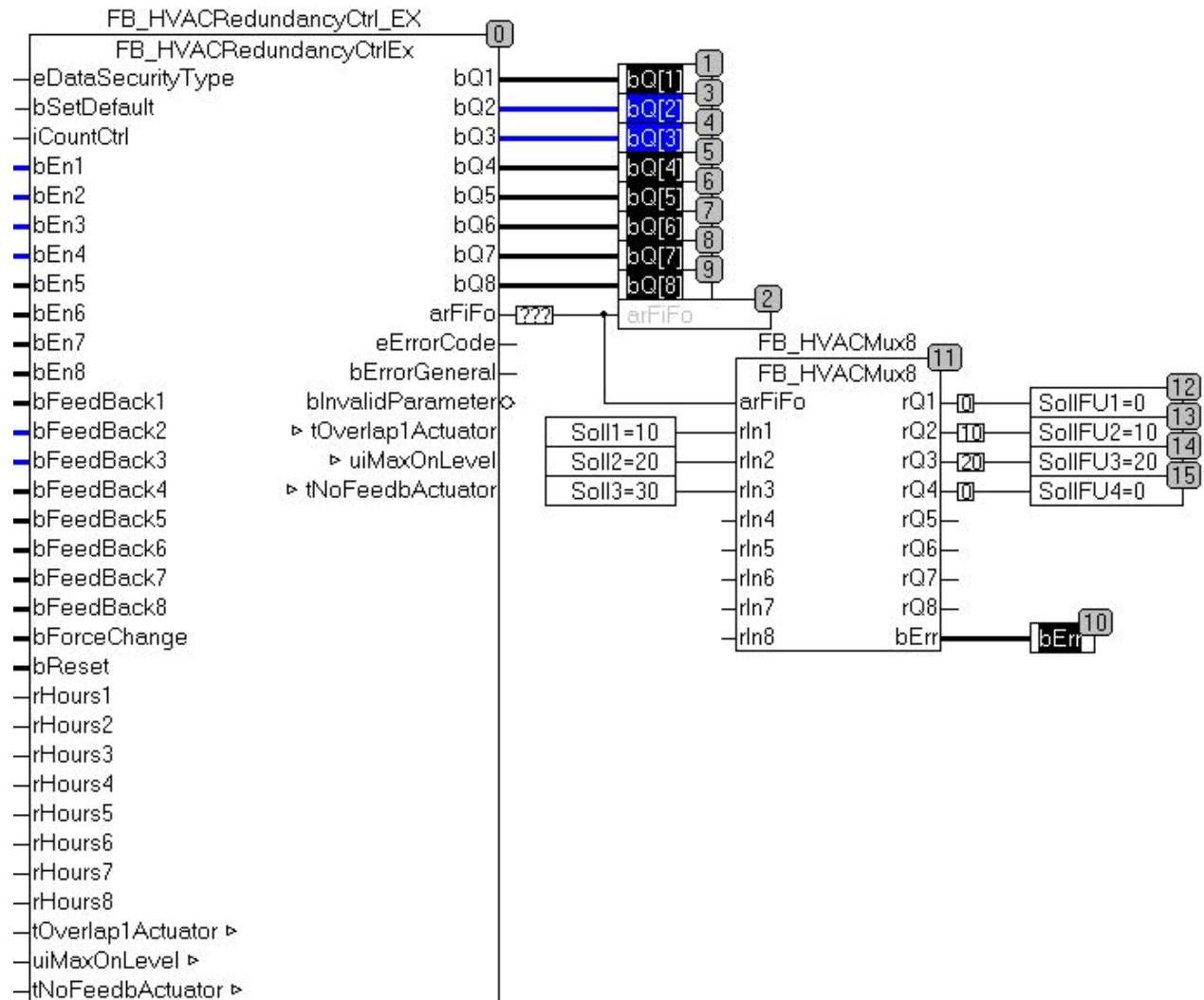


Application

This function block evaluates the FIFO memory from FB_HVACRedundancyCtrlEx. The inputs are mapped to the respective outputs according to a FIFO table.

If this function block is used, for example, to control speed-controlled fans, then the setpoints from a controller cascade can be assigned to the respective outputs (drives). To do this, the setpoints are applied to the inputs of FB_HVACMux8. The processing takes place with the aid of information from the FIFO memory (which drive is running and in which order did switch-on take place).

Example:



There are 4 drives with speed controllers (SetFC1 – SetFC4), of which a maximum of three should be running. The control range of 0 to 300% is divided into 3 setpoints (Set1 to Set3). The output *arrFiFo* of *FB_HVACRedundancyCtrlEx* supplies *FB_HVACMux8* with the assignment description. In the sample the array contains: 2,3,0,0,0,0,0,0. Therefore Q2 was switched on first and then Q3. As a result *rIn1* is now output on *rQ2* and *rIn2* on *rQ3* in *FB_HVACMux8*.

VAR_INPUT

```
arrFiFo      : Array[1..8] of INT;      0 - 8
rIn1 - rIn8  : REAL;
```

arrFiFo: contains the assignment table. The first value indicates where the first input will be copied to, the second value indicates where the second value will be copied to, etc. Identical values are permitted. No assignment is made for "0".

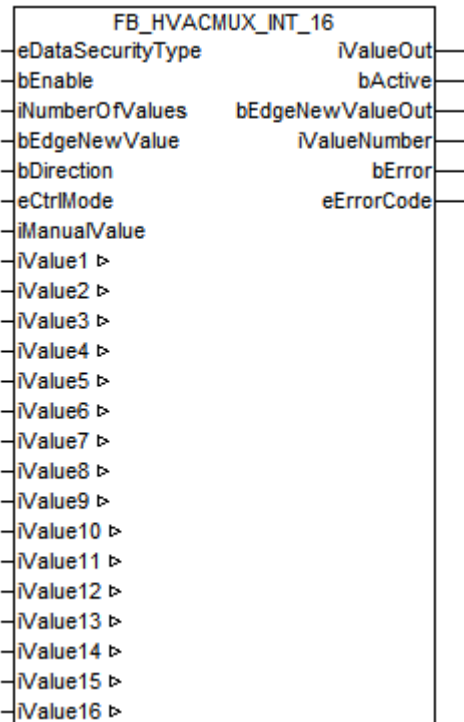
rIn1 - 8: setpoints to be mapped.

VAR_OUTPUT

```
rQ1 - rQ8      : REAL;
bErr           : BOOL;
```

rQ1 -rQ8: actuator setpoint, input value mapped according to the FIFO table.

bErr: value range in the FIFO table was exceeded; 0 to 8 is permissible.

3.6.14 FB_HVACMUX_INT_16**Application**

Application

This function block contains two different types of multiplexers, one each for the operation modes *eHVACCtrlMode_Auto* or *eHVACCtrlMode_Manual*. This selection is made with the help of the enum *eCtrlMode*.

The following conditions must be met in advance:

bEnable = TRUE AND *bError* = FALSE

If *eCtrlMode* = *eHVACCtrlMode_Auto*, the variable *bDirection* determines whether the output variable *iValueNumber* is incremented or decremented when using the input *bEdgeNewValue*. If *bDirection* = FALSE it is incremented, if *bDirection* = TRUE it is decremented.

If *eCtrlMode* = *eHVACCtrlMode_Manual*, the output *iValueNumber* is determined by the variable *iManualValue*. The variables *iNumberOfValues*, *bEdgeNewValue* and *bDirection* are not taken into account in this operation mode.

The output *iValueNumber* always indicates the number of the variable *iValueX*, whose content is output in the output variable *iValueOut*.

Examples of eCtrlMode = eHVACCtrlMode_Auto

- If *bDirection* = FALSE AND *iNumberOfValues* = 12 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* increments. If *iValueNumber* = 6, then *iValueOut* = *iValue6*. If *iValueNumber* = *iNumberOfValues*(12) and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = 1 and hence *iValueOut* = *iValue1*.

- If *bDirection* = TRUE AND *iNumberOfValues* = 8 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* decrements. If *iValueNumber* = 5, then *iValueOut* = *iValue5*. If *iValueNumber* = 1 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues*(8) and hence *iValueOut* = *iValue8*.

The start behavior of the outputs *iValueNumber* and *iValueOut* is as follows:

- If *bDirection* = FALSE AND *iNumberOfValues* > 0, then *iValueNumber* = 1 and hence *iValueOut* = *iValue1*.
- If *bDirection* = TRUE AND *iNumberOfValues* = 12, then *iNumberOfValues* = 12 and hence *iValueOut* = *iValue12*.

Examples of *eCtrlMode* = *eHVACCtrlMode_Manual*

- The value of *iValueNumber* is determined by *iManualValue*. If *iManualValue* = 7, then *iValueNumber* = 7, which then means that *iValueOut* = *iValue7*.

The start behavior of the outputs *iValueNumber* and *iValueOut* is as follows:

- If *iManualValue* = 13, then *iValueNumber* = 13 and hence *iValueOut* = *iValue13*.
- If *iManualValue* = 0, then *iValueNumber* = 0 and hence *iValueOut* = 0.

NOTICE
A frequently changing variable may not be applied to the VAR_IN_OUT variables iValue1-16 if <i>eDataSecurityType</i> = <i>eHVACDataSecurityType_Persistent</i> . This would lead to premature wear of the storage medium of the controller. If the VAR_IN_OUT variables iValue1-16 change frequently and are not to be stored persistently, then <i>eDataSecurityType</i> must be <i>eDataSecurityType_Idle</i> .

Application example

Download	Required library
TcHVAC.pro [▶ 531]	TcHVAC.lib

VAR_INPUT

```

eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
iNumberOfValues   : INT;                1..16
bEdgeNewValue     : BOOL;
bDirection        : BOOL;
eCtrlMode         : E_HVACCtrlMode;
iManualValue      : INT;                0..16
```

eDataSecurityType: if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example: <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If *eDataSecurityType* := *eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE
A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if <i>eDataSecurityType</i> := <i>eHVACDataSecurityType_Persistent</i> . It would lead to early wear of the flash memory.

bEnable: the function block is enabled via TRUE. If *bEnable* = FALSE, then *iValueOut* and *iValueNumber* are set constantly to 0.

iNumberOfValues: indicates the number of variables *iValue1-16* that can be output via *iValueOut*. If *iNumberOfValues* = 8, the output *iValueNumber* moves between 1 and 8. Then for the output *iValueOut* the variables *iValue1* to *iValue8* are taken into account, see [Application \[► 392\]](#)

iNumberOfValues determines the start behavior of the outputs *iValueOut* and *iValueNumber*, see [Application \[► 392\]](#)

iNumberOfValues is not taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual*. *iNumberOfValues* must not fall below 1 and must not exceed 16. Otherwise an error is indicated by *bError* = TRUE and the execution of the function block is stopped.

bEdgeNewValue: if there is a rising edge at *bEdgeNewValue*, then *iValueNumber* increments or decrements.

If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *iNumberOfValues* > 0 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* increments. If *iValueNumber* = 6, then *iValueOut* = *iValue6*. If *iValueNumber* = *iNumberOfValues* and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* = 1 and thus *iValueOut* = *iValue1*.

If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *iNumberOfValues* = 12 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* decrements. If *iValueNumber* = 5, then *iValueOut* = *iValue5*. If *iValueNumber* = 1 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues*(12) and thus *iValueOut* = *iValue12*.

bEdgeNewValue is not taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual*.

bDirection: *bDirection* determines the control direction of the function block.

bDirection = FALSE means that *iValueNumber* increments in ascending order from 1 to *iNumberOfValues* and thereby determines the output value *iValueOut* see [Application \[► 392\]](#)

bDirection = TRUE means, that *iValueNumber* decrements in decreasing order from *iNumberOfValues* to 1 and thereby determines the output value *iValueOut*, see [Application \[► 392\]](#)

bDirection is not taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual*.

eCtrlMode: the operation mode of the function block is specified by means of the enum, see [Application \[► 392\]](#)

If the operation mode is *eCtrlMode* = *eHVACCtrlMode_Auto*, the variable *iManualValue* is not taken into account.

If the operation mode is *eCtrlMode* = *eHVACCtrlMode_Manual*, then the variables *iNumberOfValues*, *bEdgeNewValue* and *bDirection* are not taken into account.

iManualValue: if the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual* AND *bEnable* = TRUE AND *bError* = FALSE AND *iManualValue* = 15, then *iValueNumber* = 15. The content of the variable *iValue15* is then output via *iValueOut*, see [Application \[► 392\]](#).

iManualValue is only taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual*. The value of *iValueNumber* is determined by *iManualValue*. *iManualValue* should be in the range of 0 and 16. If *iManualValue* falls below the value 0, then 0 is output via *iValueNumber*. If *iManualValue* exceeds the value 16, then 16 is output via *iValueNumber*. If *iManualValue* = 0, then *iValueNumber* = 0 and hence *iValueOut* = 0.

VAR_OUTPUT

```
iValueOut      : INT;
bActive        : BOOL;
bEdgeNewValueOut : BOOL;
iValueNumber   : INT;
bError         : BOOL;
eErrorCode     : E_HVACErrorCodes;
```

iValueOut: the value of the variable *iValue1-16* is output via *iValueOut*.

If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *iNumberOfValues* > 0 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* increments. If *iValueNumber* = 6, then *iValueOut* = *iValue6*. If *iValueNumber* = *iNumberOfValues* and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* = 1 and thus *iValueOut* = *iValue1*.

If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *iNumberOfValues* = 12 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* decrements. If *iValueNumber* = 5, then *iValueOut* = *iValue5*. If *iValueNumber* = 1 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues*(12) and thus *iValueOut* = *iValue12*.

If $bEnable = TRUE \wedge bError = FALSE \wedge eCtrlMode = eHVACCtrlMode_Manual$, then the value of $iValueNumber$ is determined by $iManualValue$. If $iManualValue = 7$, then $iValueNumber = 7$, which then means that $iValueOut = iValue7$.

The start behavior of the output $iValueOut$ looks like this, if $bEnable = TRUE \wedge bError = FALSE \wedge$

- $eCtrlMode = eHVACCtrlMode_Auto \wedge bDirection = FALSE \wedge iNumberOfValues > 0$, so $iValueNumber = 1$ and $iValueOut = iValue1$.

- $eCtrlMode = eHVACCtrlMode_Auto \wedge bDirection = TRUE \wedge iNumberOfValues = 12$, so $iNumberOfValues = 12$ and $iValueOut = iValue12$.

- $eCtrlMode = eHVACCtrlMode_Manual \wedge iManualValue = 13$, so $iValueNumber = 13$ and $iValueOut = iValue13$.

- $eCtrlMode = eHVACCtrlMode_Manual \wedge iManualValue = 0$, so $iValueNumber = 0$ and $iValueOut = 0$.

bActive: $bActive$ becomes TRUE, if

1. $bEnable = TRUE \wedge bError = FALSE \wedge eCtrlMode = eHVACCtrlMode_Auto$.

2. $bEnable = TRUE \wedge bError = FALSE \wedge eCtrlMode = eHVACCtrlMode_Manual \wedge iManualValue > 0$.

bEdgeNewValueOut: is TRUE for one PLC cycle if $bEnable = TRUE \wedge bError = FALSE \wedge iValueNumber$ changes its value.

iValueNumber: the output $iValueNumber$ always indicates the number of the variable $iValueX$, whose content is output in the output variable $iValueOut$.

If $eCtrlMode = eHVACCtrlMode_Auto$ the variable $bDirection$ determines whether the output variable $iValueNumber$ is incremented or decremented when using the input $bEdgeNewValue$. If $bDirection = FALSE$ it is incremented, if $bDirection = TRUE$ it is decremented.

If $eCtrlMode = eHVACCtrlMode_Manual$ the value of $iValueNumber$ is determined by $iManualValue$.

bError: the output signals with a TRUE that an error is present and an incorrect parameter is present at the variable $iNumberOfValues$. $iValueOut$ and $iValueNumber$ are constantly set to 0 and the enum $eErrorCode$ indicates the error number. The message $bError$ does not have to be acknowledged after rectification of the error.

eErrorCode: returns the [error number](#) [► 520] when the $bError$ output is set. The following error can occur in this function block:

$eHVACErrorCodes_InvalidParam_iNumberOfValues$: there is an incorrect value at $iNumberOfValues$. $iNumberOfValues$ must not fall below 1 and not exceed 16.



To access the enum error numbers in the PLC, $eErrorCode$ can be assigned to a variable of the data type WORD. $eHVACErrorCodes_InvalidParam_iNumberOfValues = 42$

VAR_IN_OUT

```
iValue1-16 : INT;
```

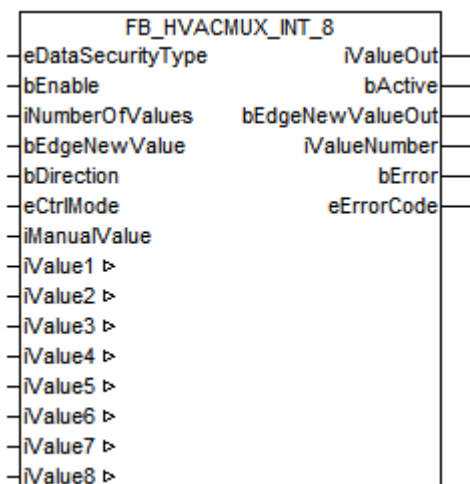
iValue1-16: the value of the output variable $iValueOut$ is determined by the variables $iValue1$ to $iValue16$. If $iValueNumber = 6$, then $iValueOut = iValue6$.

If $iNumberOfValues = 8$, the output $iValueNumber$ moves between 1 and 8. The variables $iValue1$ to $iValue8$ are then taken into account for the output $iValueOut$ if the operation mode is $eCtrlMode = eHVACCtrlMode_Auto$, see [Application](#) [► 392]

Documents about this

📄 [example_persistent_e.zip](#) (Resources/zip/11659714827.zip)

3.6.15 FB_HVACMUX_INT_8



Application

Application

This function block contains two different types of multiplexers, one each for the operation modes *eHVACCtrlMode_Auto* or *eHVACCtrlMode_Manual*. This selection is made with the help of the enum *eCtrlMode*.

The following conditions must be met in advance:

$bEnable = TRUE \text{ AND } bError = FALSE$

If *eCtrlMode* = *eHVACCtrlMode_Auto*, the variable *bDirection* determines whether the output variable *iValueNumber* is incremented or decremented when using the input *bEdgeNewValue*. If *bDirection* = FALSE it is incremented, if *bDirection* = TRUE it is decremented.

If *eCtrlMode* = *eHVACCtrlMode_Manual*, the output *iValueNumber* is determined by the variable *iManualValue*. The variables *iNumberOfValues*, *bEdgeNewValue* and *bDirection* are not taken into account in this operation mode.

The output *iValueNumber* always indicates the number of the variable *iValueX*, whose content is output in the output variable *iValueOut*.

Examples of *eCtrlMode* = *eHVACCtrlMode_Auto*

- If *bDirection* = FALSE AND *iNumberOfValues* = 7 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* increments. If *iValueNumber* = 4, then *iValueOut* = *iValue4*. If *iValueNumber* = *iNumberOfValues*(7) and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = 1 and hence *iValueOut* = *iValue1*.

- If *bDirection* = TRUE AND *iNumberOfValues* = 8 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* decrements. If *iValueNumber* = 3, then *iValueOut* = *iValue3*. If *iValueNumber* = 1 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues*(8) and hence *iValueOut* = *iValue8*.

The start behavior of the outputs *iValueNumber* and *iValueOut* is as follows:

- If *bDirection* = FALSE AND *iNumberOfValues* > 0, then *iValueNumber* = 1 and *iValueOut* = *iValue1*.

- If *bDirection* = TRUE AND *iNumberOfValues* = 6, then *iNumberOfValues* = 6 and hence *iValueOut* = *iValue6*.

Examples of *eCtrlMode* = *eHVACCtrlMode_Manual*

- The value of *iValueNumber* is determined by *iManualValue*. If *iManualValue* = 7, then *iValueNumber* = 7, which then means that *iValueOut* = *iValue7*.

The start behavior of the outputs *iValueNumber* and *iValueOut* is as follows:

- If *iManualValue* = 13, then *iValueNumber* = 13 and *iValueOut* = *iValue13*.
- If *iManualValue* = 0, then *iValueNumber* = 0 and *iValueOut* = 0.

NOTICE
A frequently changing variable may not be applied to the VAR_IN_OUT variables <i>iValue1-8</i> if <i>eDataSecurityType</i> = <i>eHVACDataSecurityType_Persistent</i> . This would lead to premature wear of the storage medium of the controller. If the VAR_IN_OUT variables <i>iValue1-8</i> change frequently and are not to be stored persistently, then <i>eDataSecurityType</i> must be <i>eDataSecurityType_Idle</i> .

Application example


Download	Required library
TcHVAC.pro [► 531]	TcHVAC.lib

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
iNumberOfValues   : INT;                1..8
bEdgeNewValue     : BOOL;
bDirection        : BOOL;
eCtrlMode         : E_HVACCtrlMode;
iManualValue      : INT;                0..8
```

eDataSecurityType: if *eDataSecurityType*:= *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If *eDataSecurityType*:= *eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE
A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if <i>eDataSecurityType</i> := <i>eHVACDataSecurityType_Persistent</i> . It would lead to early wear of the flash memory.

bEnable: the function block is enabled via TRUE. If *bEnable* = FALSE, then *iValueOut* and *iValueNumber* are set constantly to 0.

iNumberOfValues: indicates the number of variables *iValue1-8* that can be output via *iValueOut*. If *iNumberOfValues* = 8, the output *iValueNumber* moves between 1 and 8. Then for the output *iValueOut* the variables *iValue1* to *iValue8* are taken into account, see [Application \[► 396\]](#)
iNumberOfValues determines the start behavior of the outputs *iValueOut* and *iValueNumber*, see [Application \[► 396\]](#)
iNumberOfValues is not taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual*. *iNumberOfValues* must not fall below 1 and must not exceed 8. Otherwise an error is indicated by *bError* = TRUE and the execution of the function block is stopped.

bEdgeNewValue: if there is a rising edge at *bEdgeNewValue*, then *iValueNumber* increments or decrements.
 If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *iNumberOfValues* > 0 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* increments. If *iValueNumber* = 6, then *iValueOut* = *iValue6*. If *iValueNumber* = *iNumberOfValues* and there is

a rising edge at *bEdgeNewValue*, then *iValueNumber* = 1 and thus *iValueOut* = *iValue1*.
 If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *iNumberOfValues* = 6 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* decrements. If *iValueNumber* = 5, then *iValueOut* = *iValue5*. If *iValueNumber* = 1 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues*(6) and thus *iValueOut* = *iValue6*.
bEdgeNewValue is not taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual*.

bDirection: *bDirection* determines the control direction of the function block.

bDirection = FALSE means that *iValueNumber* increments in ascending order from 1 to *iNumberOfValues* and thereby determines the output value *iValueOut* see [Application \[► 396\]](#)

bDirection = TRUE means, that *iValueNumber* decrements in decreasing order from *iNumberOfValues* to 1 and thereby determines the output value *iValueOut*, see [Application \[► 396\]](#)

bDirection is not taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual*.

eCtrlMode: the operation mode of the function block is specified by means of the enum, see [Application \[► 396\]](#)

If the operation mode is *eCtrlMode* = *eHVACCtrlMode_Auto*, the variable *iManualValue* is not taken into account.

If the operation mode is *eCtrlMode* = *eHVACCtrlMode_Manual*, then the variables *iNumberOfValues*, *bEdgeNewValue* and *bDirection* are not taken into account.

iManualValue: if the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual* AND *bEnable* = TRUE AND *bError* = FALSE AND *iManualValue* = 15, then *iValueNumber* = 15. The content of the variable *iValue15* is then output via *iValueOut*, see [Application \[► 396\]](#).

iManualValue is only taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual*. The value of *iValueNumber* is determined by *iManualValue*. *iManualValue* should be in the range of 0 and 16. If *iManualValue* falls below the value 0, then 0 is output via *iValueNumber*. If *iManualValue* exceeds the value 16, then 16 is output via *iValueNumber*. If *iManualValue* = 0, then *iValueNumber* = 0 and *iValueOut* = 0.

VAR_OUTPUT

```
iValueOut      : INT;
bActive        : BOOL;
bEdgeNewValueOut : BOOL;
iValueNumber   : INT;
bError         : BOOL;
eErrorCode     : E_HVACErrorCodes;
```

iValueOut: the value of the variable *iValue1-8* is output via *iValueOut*.

If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *iNumberOfValues* > 0 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* increments. If *iValueNumber* = 6, then *iValueOut* = *iValue6*. If *iValueNumber* = *iNumberOfValues* and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* = 1 and thus *iValueOut* = *iValue1*.

If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *iNumberOfValues* = 7 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* decrements. If *iValueNumber* = 5, then *iValueOut* = *iValue5*. If *iValueNumber* = 1 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues*(7) and thus *iValueOut* = *iValue7*.

If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Manual*, then the value of *iValueNumber* is determined by *iManualValue*. If *iManualValue* = 7, then *iValueNumber* = 7, which then means that *iValueOut* = *iValue7*.

The start behavior of the output *iValueOut* looks like this, if *bEnable* = TRUE AND *bError* = FALSE AND

- *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *iNumberOfValues* > 0, so *iValueNumber* = 1 and *iValueOut* = *iValue1*.

- *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *iNumberOfValues* = 7, so *iNumberOfValues* = 7 and *iValueOut* = *iValue7*.

- *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 13, so *iValueNumber* = 13 and *iValueOut* = *iValue13*.

- *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 0, so *iValueNumber* = 0 and *iValueOut* = 0.

bActive: *bActive* becomes TRUE, if

1. *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto*.

2. $bEnable = TRUE \wedge bError = FALSE \wedge eCtrlMode = eHVACCtrlMode_Manual \wedge iManualValue > 0$.

bEdgeNewValueOut: is TRUE for one PLC cycle if $bEnable = TRUE \wedge bError = FALSE \wedge iValueNumber$ changes its value.

iValueNumber: the output $iValueNumber$ always indicates the number of the variable $iValueX$, whose content is output in the output variable $iValueOut$.

If $eCtrlMode = eHVACCtrlMode_Auto$ the variable $bDirection$ determines whether the output variable $iValueNumber$ is incremented or decremented when using the input $bEdgeNewValue$. If $bDirection = FALSE$ it is incremented, if $bDirection = TRUE$ it is decremented.

If $eCtrlMode = eHVACCtrlMode_Manual$ the value of $iValueNumber$ is determined by $iManualValue$.

bError: the output signals with a TRUE that an error is present and an incorrect parameter is present at the variable $iNumberOfValues$. $iValueOut$ and $iValueNumber$ are constantly set to 0 and the enum $eErrorCode$ indicates the error number. The message $bError$ does not have to be acknowledged after rectification of the error.

eErrorCode: returns the [error number](#) [► 520] when the $bError$ output is set. The following error can occur in this function block:

$eHVACErrorCodes_InvalidParam_iNumberOfValues$: there is an incorrect value at $iNumberOfValues$. $iNumberOfValues$ must not fall below 1 and must not exceed 8.



To access the enum error numbers in the PLC, $eErrorCode$ can be assigned to a variable of the data type WORD. $eHVACErrorCodes_InvalidParam_iNumberOfValues = 42$

VAR_IN_OUT

```
iValue1-8      : INT;
```

iValue1-8: the value of the output variable $iValueOut$ is determined by the variables $iValue1$ to $iValue8$. If $iValueNumber = 6$, then $iValueOut = iValue6$.

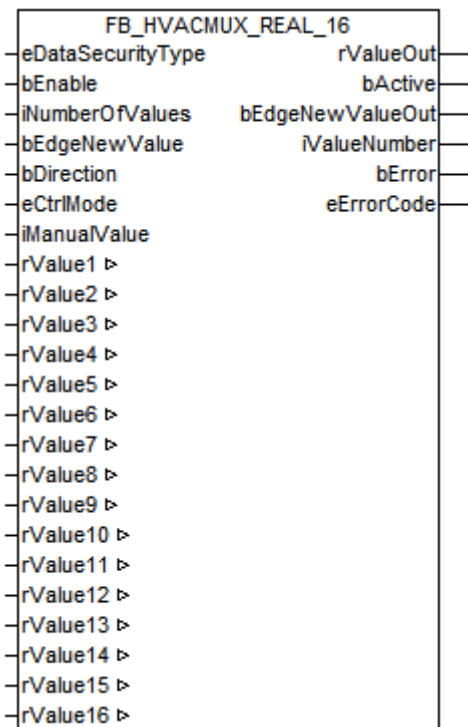
If $iNumberOfValues = 8$, the output $iValueNumber$ moves between 1 and 8. The variables $iValue1$ to $iValue8$ are then taken into account for the output $iValueOut$ if the operation mode is $eCtrlMode =$

$eHVACCtrlMode_Auto$, see [Application](#) [► 396]

Documents about this

- example_persistent_e.zip (Resources/zip/11659714827.zip)

3.6.16 FB_HVACMUX_REAL_16



Application

Application

This function block contains two different types of multiplexers, one each for the operation modes *eHVACCtrlMode_Auto* or *eHVACCtrlMode_Manual*. This selection is made with the help of the enum *eCtrlMode*.

The following conditions must be met in advance:

bEnable = TRUE AND *bError* = FALSE

If *eCtrlMode* = *eHVACCtrlMode_Auto*, the variable *bDirection* determines whether the output variable *iValueNumber* is incremented or decremented when using the input *bEdgeNewValue*. If *bDirection* = FALSE it is incremented, if *bDirection* = TRUE it is decremented.

If *eCtrlMode* = *eHVACCtrlMode_Manual*, the output *iValueNumber* is determined by the variable *iManualValue*. The variables *iNumberOfValues*, *bEdgeNewValue* and *bDirection* are not taken into account in this operation mode.

The output *iValueNumber* always indicates the number of the variable *rValueX*, whose content is output in the output variable *rValueOut*.

Examples of *eCtrlMode* = *eHVACCtrlMode_Auto*

- If *bDirection* = FALSE AND *iNumberOfValues* = **12** and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* increments. If *iValueNumber* = 6, then *rValueOut* = *rValue6*. If *iValueNumber* = *iNumberOfValues*(**12**) and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = 1 and thus *rValueOut* = *rValue1*.

- If *bDirection* = TRUE AND *iNumberOfValues* = **8** and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* decrements. If *iValueNumber* = 5, then *rValueOut* = *rValue5*. If *iValueNumber* = 1 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues*(**8**) and thus *rValueOut* = *rValue8*.

The start behavior of the outputs *iValueNumber* and *rValueOut* is as follows:

- If *bDirection* = FALSE AND *iNumberOfValues* > 0, then *iValueNumber* = 1 and hence *rValueOut* = *rValue1*.

- If *bDirection* = TRUE AND *iNumberOfValues* = 12, then *iNumberOfValues* = 12 and hence *rValueOut* = *rValue12*.

Examples of *eCtrlMode* = *eHVACCtrlMode_Manual*

- The value of *iValueNumber* is determined by *iManualValue*. If *iManualValue* = 7, then *iValueNumber* = 7, which then means that *rValueOut* = *rValue7*.

The start behavior of the outputs *iValueNumber* and *rValueOut* is as follows:

- If *iManualValue* = 13, then *iValueNumber* = 13 and *rValueOut* = *rValue13*.
- If *iManualValue* = 0, then *iValueNumber* = 0 and *rValueOut* = 0.

NOTICE
A frequently changing variable may not be applied to the VAR_IN_OUT variables <i>rValue1-16</i> if <i>eDataSecurityType</i> = <i>eHVACDataSecurityType_Persistent</i> . This would lead to premature wear of the storage medium of the controller. If the VAR_IN_OUT variables <i>rValue1-16</i> change frequently and are not to be stored persistently, then <i>eDataSecurityType</i> must be <i>eDataSecurityType_Idle</i> .

Application example


Download	Required library
TcHVAC.pro [▶ 531]	TcHVAC.lib

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
iNumberOfValues   : INT;                1..16
bEdgeNewValue     : BOOL;
bDirection        : BOOL;
eCtrlMode         : E_HVACCtrlMode;
iManualValue      : INT;                0..16
```

eDataSecurityType: if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block *FB_HVACPersistentDataHandling* must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If *eDataSecurityType* := *eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE
A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if <i>eDataSecurityType</i> := <i>eHVACDataSecurityType_Persistent</i> . It would lead to early wear of the flash memory.

bEnable: the function block is enabled via TRUE. If *bEnable* = FALSE, then *rValueOut* and *iValueNumber* are set constantly to 0.

iNumberOfValues: indicates the number of variables *rValue1-16* that can be output via *rValueOut*. If *iNumberOfValues* = 8, the output *iValueNumber* moves between 1 and 8. The variables *rValue1* to *rValue8* are then taken into account for the output *rValueOut*, see [Application \[▶ 400\]](#) *iNumberOfValues* determines the start behavior of the outputs *rValueOut* and *iValueNumber*, see [Application \[▶ 400\]](#)

iNumberOfValues is not taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual*. *iNumberOfValues* must not fall below 1 and must not exceed 16. Otherwise an error is indicated by *bError* = TRUE and the execution of the function block is stopped.

bEdgeNewValue: if there is a rising edge at *bEdgeNewValue*, then *iValueNumber* increments or decrements.

If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *iNumberOfValues* > 0 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* increments. If *iValueNumber* = 6, then *rValueOut* = *rValue6*. If *iValueNumber* = *iNumberOfValues* and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* = 1 and thus *rValueOut* = *rValue1*.

If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *iNumberOfValues* = 12 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* decrements. If *iValueNumber* = 5, then *rValueOut* = *rValue5*. If *iValueNumber* = 1 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues*(12) and thus *rValueOut* = *rValue12*.

bEdgeNewValue is not taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual*.

bDirection: *bDirection* determines the control direction of the function block.

bDirection = FALSE means that *iValueNumber* increments in ascending order from 1 to *iNumberOfValues* and thereby determines the output value *rValueOut* see [Application \[▶ 400\]](#)

bDirection = TRUE means, that *iValueNumber* decrements in decreasing order from *iNumberOfValues* to 1 and thereby determines the output value *rValueOut*, see [Application \[▶ 400\]](#)

bDirection is not taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual*.

eCtrlMode: the operation mode of the function block is specified by means of the enum, see [Application \[▶ 400\]](#)

If the operation mode is *eCtrlMode* = *eHVACCtrlMode_Auto*, the variable *iManualValue* is not taken into account.

If the operation mode is *eCtrlMode* = *eHVACCtrlMode_Manual*, then the variables *iNumberOfValues*, *bEdgeNewValue* and *bDirection* are not taken into account.

iManualValue: if the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual* AND *bEnable* = TRUE AND *bError* = FALSE AND *iManualValue* = 15, then *iValueNumber* = 15. The content of the variable *rValue15* is then output via *rValueOut*, see [Application \[▶ 400\]](#).

iManualValue is only taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual*. The value of *iValueNumber* is determined by *iManualValue*. *iManualValue* should be in the range of 0 and 16. If *iManualValue* falls below the value 0, then 0 is output via *iValueNumber*. If *iManualValue* exceeds the value 16, then 16 is output via *iValueNumber*. If *iManualValue* = 0, then *iValueNumber* = 0 and *rValueOut* = 0.

VAR_OUTPUT

```
rValueOut      : REAL;
bActive        : BOOL;
bEdgeNewValueOut : BOOL;
iValueNumber   : INT;
bError         : BOOL;
eErrorCode     : E_HVACErrorCodes;
```

rValueOut: the value of the variable *rValue1-16* is output via *rValueOut*.

If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *iNumberOfValues* > 0 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* increments. If *iValueNumber* = 6, then *rValueOut* = *rValue6*. If *iValueNumber* = *iNumberOfValues* and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* = 1 and thus *rValueOut* = *rValue1*.

If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *iNumberOfValues* = 12 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* decrements. If *iValueNumber* = 5, then *rValueOut* = *rValue5*. If *iValueNumber* = 1 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues*(12) and thus *rValueOut* = *rValue12*.

If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Manual*, then the value of *iValueNumber* is determined by *iManualValue*. If *iManualValue* = 7, then *iValueNumber* = 7, which then means that *rValueOut* = *rValue7*.

The start behavior of the output *rValueOut* looks like this, if *bEnable* = TRUE AND *bError* = FALSE AND

- *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *iNumberOfValues* > 0, so *iValueNumber* = 1 and *rValueOut* = *rValue1*.

- $eCtrlMode = eHVACCtrlMode_Auto$ AND $bDirection = TRUE$ AND $iNumberOfValues = 12$, so $iValueNumber = 12$ and $rValueOut = rValue12$.

- $eCtrlMode = eHVACCtrlMode_Manual$ AND $iManualValue = 13$, so $iValueNumber = 13$ and $rValueOut = rValue13$.

- $eCtrlMode = eHVACCtrlMode_Manual$ AND $iManualValue = 0$, so $iValueNumber = 0$ and $rValueOut = 0$.

bActive: $bActive$ becomes TRUE, if

1. $bEnable = TRUE$ AND $bError = FALSE$ AND $eCtrlMode = eHVACCtrlMode_Auto$.
2. $bEnable = TRUE$ AND $bError = FALSE$ AND $eCtrlMode = eHVACCtrlMode_Manual$ AND $iManualValue > 0$.

bEdgeNewValueOut: is TRUE for one PLC cycle if $bEnable = TRUE$ AND $bError = FALSE$ AND $iValueNumber$ changes its value.

iValueNumber: the output $iValueNumber$ always indicates the number of the variable $rValueX$, whose content is output in the output variable $rValueOut$.

If $eCtrlMode = eHVACCtrlMode_Auto$ the variable $bDirection$ determines whether the output variable $iValueNumber$ is incremented or decremented when using the input $bEdgeNewValue$. If $bDirection = FALSE$ it is incremented, if $bDirection = TRUE$ it is decremented.

If $eCtrlMode = eHVACCtrlMode_Manual$ the value of $iValueNumber$ is determined by $iManualValue$.

bError: the output signals with a TRUE that an error is present and an incorrect parameter is present at the variable $iNumberOfValues$. $rValueOut$ and $iValueNumber$ are constantly set to 0 and the enum $eErrorCode$ indicates the error number. The message $bError$ does not have to be acknowledged after rectification of the error.

eErrorCode: returns the [error number \[► 520\]](#) when the $bError$ output is set. The following error can occur in this function block:

$eHVACErrorCodes_InvalidParam_iNumberOfValues$: there is an incorrect value at $iNumberOfValues$. $iNumberOfValues$ must not fall below 1 and not exceed 16.



To access the enum error numbers in the PLC, $eErrorCode$ can be assigned to a variable of the data type WORD. $eHVACErrorCodes_InvalidParam_iNumberOfValues = 42$

VAR_IN_OUT

```
rValue1-16 : REAL;
```

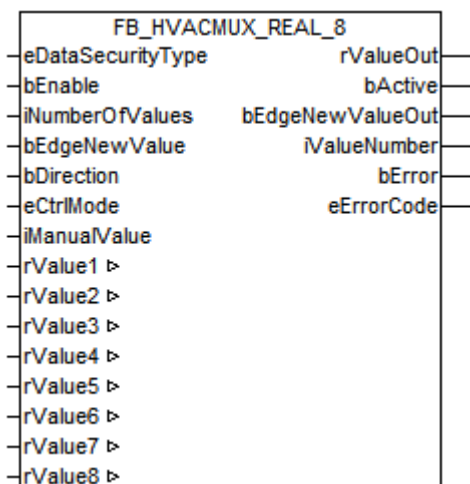
rValue1-16: the value of the output variable $rValueOut$ is determined by the variables $rValue1$ to $rValue16$. If $iValueNumber = 6$, then $rValueOut = rValue6$.

If $iNumberOfValues = 8$, the output $iValueNumber$ moves between 1 and 8. The variables $rValue1$ to $rValue8$ are taken into account for the output $rValueOut$ if the operation mode is $eCtrlMode = eHVACCtrlMode_Auto$, see [Application \[► 400\]](#): the variable is saved persistently.

Documents about this

- example_persistent_e.zip (Resources/zip/11659714827.zip)

3.6.17 FB_HVACMUX_REAL_8



Application

Application

This function block contains two different types of multiplexers, one each for the operation modes *eHVACCtrlMode_Auto* or *eHVACCtrlMode_Manual*. This selection is made with the help of the enum *eCtrlMode*.

The following conditions must be met in advance:

bEnable = TRUE AND *bError* = FALSE

If *eCtrlMode* = *eHVACCtrlMode_Auto*, the variable *bDirection* determines whether the output variable *iValueNumber* is incremented or decremented when using the input *bEdgeNewValue*. If *bDirection* = FALSE it is incremented, if *bDirection* = TRUE it is decremented.

If *eCtrlMode* = *eHVACCtrlMode_Manual*, the output *iValueNumber* is determined by the variable *iManualValue*. The variables *iNumberOfValues*, *bEdgeNewValue* and *bDirection* are not taken into account in this operation mode.

The output *iValueNumber* always indicates the number of the variable *rValueX*, whose content is output in the output variable *rValueOut*.

Examples of *eCtrlMode* = *eHVACCtrlMode_Auto*

- If *bDirection* = FALSE AND *iNumberOfValues* = 7 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* increments. If *iValueNumber* = 4, then *rValueOut* = *rValue4*. If *iValueNumber* = *iNumberOfValues*(7) and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = 1 and thus *rValueOut* = *rValue1*.

- If *bDirection* = TRUE AND *iNumberOfValues* = 8 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* decrements. If *iValueNumber* = 3, then *rValueOut* = *rValue3*. If *iValueNumber* = 1 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues*(8) and thus *rValueOut* = *rValue8*.

The start behavior of the outputs *iValueNumber* and *rValueOut* is as follows:

- If *bDirection* = FALSE AND *iNumberOfValues* > 0, then *iValueNumber* = 1 and hence *rValueOut* = *rValue1*.
- If *bDirection* = TRUE AND *iNumberOfValues* = 6, then *iValueNumber* = 6 and hence *rValueOut* = *rValue6*.

Examples of *eCtrlMode* = *eHVACCtrlMode_Manual*

- The value of *iValueNumber* is determined by *iManualValue*. If *iManualValue* = 7, then *iValueNumber* = 7, which then means that *rValueOut* = *rValue7*.

The start behavior of the outputs *iValueNumber* and *rValueOut* is as follows:

- If *iManualValue* = 13, then *iValueNumber* = 13 and *rValueOut* = *rValue13*.
- If *iManualValue* = 0, then *iValueNumber* = 0 and *rValueOut* = 0.

NOTICE
A frequently changing variable may not be applied to the VAR_IN_OUT variables <i>rValue1-8</i> if <i>eDataSecurityType</i> = <i>eHVACDataSecurityType_Persistent</i> . This would lead to premature wear of the storage medium of the controller. If the VAR_IN_OUT variables <i>rValue1-8</i> change frequently and are not to be stored persistently, then <i>eDataSecurityType</i> must be <i>eDataSecurityType_Idle</i> .

Application example


Download	Required library
TcHVAC.pro [▶ 531]	TcHVAC.lib

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
iNumberOfValues   : INT;                1..8
bEdgeNewValue     : BOOL;
bDirection        : BOOL;
eCtrlMode         : E_HVACCtrlMode;
iManualValue      : INT;                0..8
```

eDataSecurityType: if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If *eDataSecurityType* := *eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE
A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if <i>eDataSecurityType</i> := <i>eHVACDataSecurityType_Persistent</i> . It would lead to early wear of the flash memory.

bEnable: the function block is enabled via TRUE. If *bEnable* = FALSE, then *rValueOut* and *iValueNumber* are set constantly to 0.

iNumberOfValues: indicates the number of variables *rValue1-8* that can be output via *rValueOut*. If *iNumberOfValues* = 8, the output *iValueNumber* moves between 1 and 8. The variables *rValue1* to *rValue8* are then taken into account for the output *rValueOut*, see [Application \[\[▶ 404\]\(#\)\]](#) *iNumberOfValues* determines the start behavior of the outputs *rValueOut* and *iValueNumber*, see [Application \[\[▶ 404\]\(#\)\]](#) *iNumberOfValues* is not taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual*. *iNumberOfValues* must not fall below 1 and must not exceed 8. Otherwise an error is indicated by *bError* = TRUE and the execution of the function block is stopped.

bEdgeNewValue: if there is a rising edge at *bEdgeNewValue*, then *iValueNumber* increments or decrements. If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *iNumberOfValues* > 0 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* increments. If *iValueNumber* = 6, then *rValueOut* = *rValue6*. If *iValueNumber* = *iNumberOfValues* and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* = 1 and *rValueOut* = *rValue1*. If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* =

TRUEANDiNumberOfValues = 6 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* decrements. If *iValueNumber* = 5, then *rValueOut* = *rValue5*. If *iValueNumber* = 1 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues*(6) and *rValueOut* = *rValue6*. *bEdgeNewValue* is not taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual*.

bDirection: *bDirection* determines the control direction of the function block.

bDirection = FALSE means that *iValueNumber* increments in ascending order from 1 to *iNumberOfValues* and thereby determines the output value *rValueOut* see [Application \[▶ 404\]](#)

bDirection = TRUE means, that *iValueNumber* decrements in decreasing order from *iNumberOfValues* to 1 and thereby determines the output value *rValueOut*, see [Application \[▶ 404\]](#)

bDirection is not taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual*.

eCtrlMode: the operation mode of the function block is specified by means of the enum, see [Application \[▶ 404\]](#)

If the operation mode is *eCtrlMode* = *eHVACCtrlMode_Auto*, the variable *iManualValue* is not taken into account.

If the operation mode is *eCtrlMode* = *eHVACCtrlMode_Manual*, then the variables *iNumberOfValues*, *bEdgeNewValue* and *bDirection* are not taken into account.

iManualValue: if the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual* AND *bEnable* = TRUE AND *bError* = FALSE AND *iManualValue* = 15, then *iValueNumber* = 15. The content of the variable *rValue15* is then output via *rValueOut*, see [Application \[▶ 404\]](#).

iManualValue is only taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual*. The value of *iValueNumber* is determined by *iManualValue*. *iManualValue* should be in the range of 0 and 16. If *iManualValue* falls below the value 0, then 0 is output via *iValueNumber*. If *iManualValue* exceeds the value 16, then 16 is output via *iValueNumber*. If *iManualValue* = 0, then *iValueNumber* = 0 and *rValueOut* = 0.

VAR_OUTPUT

```
rValueOut      : INT;
bActive        : BOOL;
bEdgeNewValueOut : BOOL;
iValueNumber   : INT;
bError         : BOOL;
eErrorCode     : E_HVACErrorCodes;
```

rValueOut: the value of the variable *rValue1-8* is output via *rValueOut*.

If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *iNumberOfValues* > 0 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* increments. If *iValueNumber* = 6, then *rValueOut* = *rValue6*. If *iValueNumber* = *iNumberOfValues* and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* = 1 and *rValueOut* = *rValue1*.

If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *iNumberOfValues* = 7 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* decrements. If *iValueNumber* = 5, then *rValueOut* = *rValue5*. If *iValueNumber* = 1 and there is a rising edge at *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues*(7) and *rValueOut* = *rValue7*.

If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Manual*, then the value of *iValueNumber* is determined by *iManualValue*. If *iManualValue* = 7, then *iValueNumber* = 7, which then means that *rValueOut* = *rValue7*.

The start behavior of the output *rValueOut* looks like this, if *bEnable* = TRUE AND *bError* = FALSE AND

- *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *iNumberOfValues* > 0, so *iValueNumber* = 1 and *rValueOut* = *rValue1*.

- *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *iNumberOfValues* = 7, so *iNumberOfValues* = 7 and *rValueOut* = *rValue7*.

- *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 13, so *iValueNumber* = 13 and *rValueOut* = *rValue13*.

- *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 0, so *iValueNumber* = 0 and *rValueOut* = 0.

bActive: *bActive* becomes TRUE, if

1. *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto*.

2. *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* > 0.

bEdgeNewValueOut: is TRUE for one PLC cycle if $bEnable = TRUE \wedge bError = FALSE \wedge iValueNumber$ changes its value.

iValueNumber: the output $iValueNumber$ always indicates the number of the variable $rValueX$, whose content is output in the output variable $rValueOut$.

If $eCtrlMode = eHVACCtrlMode_Auto$ the variable $bDirection$ determines whether the output variable $iValueNumber$ is incremented or decremented when using the input $bEdgeNewValue$. If $bDirection = FALSE$ it is incremented, if $bDirection = TRUE$ it is decremented.

If $eCtrlMode = eHVACCtrlMode_Manual$ the value of $iValueNumber$ is determined by $iManualValue$.

bError: the output signals with a TRUE that an error is present and an incorrect parameter is present at the variable $iNumberOfValues$. $rValueOut$ and $iValueNumber$ are constantly set to 0 and the enum $eErrorCode$ indicates the error number. The message $bError$ does not have to be acknowledged after rectification of the error.

eErrorCode: returns the error number [► 520] when the $bError$ output is set. The following error can occur in this function block:

$eHVACErrorCodes_InvalidParam_iNumberOfValues$: there is an incorrect value at $iNumberOfValues$. $iNumberOfValues$ must not fall below 1 and must not exceed 8.



To access the enum error numbers in the PLC, $eErrorCode$ can be assigned to a variable of the data type WORD. $eHVACErrorCodes_InvalidParam_iNumberOfValues = 42$

VAR_IN_OUT

```
rValue1-8 : INT;
```

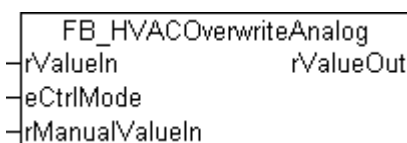
rValue1-8: the value of the output variable $rValueOut$ is determined by the variables $rValue1$ to $rValue8$. If $iValueNumber = 6$, then $rValueOut = rValue6$.

If $iNumberOfValues = 8$, the output $iValueNumber$ moves between 1 and 8. The variables $rValue1$ to $rValue8$ are taken into account for the output $rValueOut$ if the operation mode is $eCtrlMode = eHVACCtrlMode_Auto$, see Application [► 404]: the variable is saved persistently.

Documents about this

example_persistent_e.zip (Resources/zip/11659714827.zip)

3.6.18 FB_HVACOverwriteAnalog



Application

This function block overrides the input $rValueIn$ if $eCtrlMode = eHVACCtrlMode_Manual$ and forwards $rManualValueIn$ to the output $rValueOut$.

VAR_INPUT

```
rValueIn : REAL;
eCtrlMode : E_HVACCtrlMode;
rManualValueIn : REAL;
```

rValueIn: analog input value forwarded to output $rValueOut$ if $eCtrlMode = eHVACCtrlMode_Auto$.

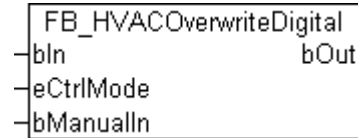
eCtrlMode: the operation mode is selected via this enum. Manual or automatic operation mode.

rManualValueIn: analog manual input value forwarded to output $rValueOut$ if $eCtrlMode = eHVACCtrlMode_Manual$.

VAR_OUTPUT

```
rValueOut      :REAL;
```

rValueOut: analog output value.

3.6.19 FB_HVACOverwriteDigital**Application**

This function block overrides the state of the input *bIn* if *eCtrlMode* = *eHVACCtrlMode_Manual* and forwards it to the output *bOut*.

VAR_INPUT

```
bIn             :BOOL;
eCtrlMode       :E_HVACCtrlMode;
bManualIn       :BOOL;
```

bIn: digital input variable whose state is forwarded to output *bOut* if *eCtrlMode* = *eHVACCtrlMode_Auto*.

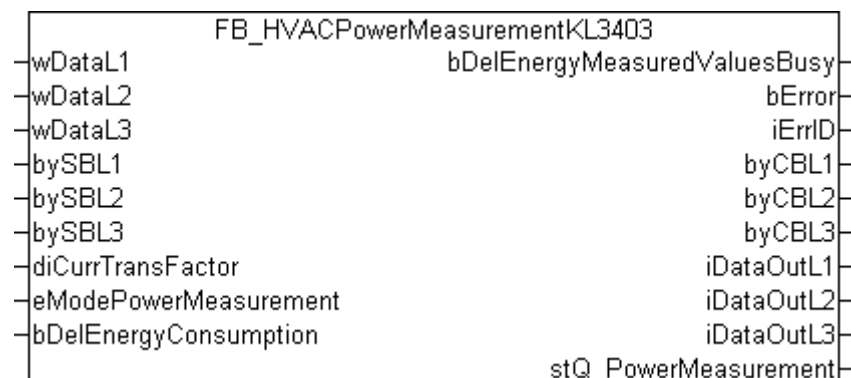
eCtrlMode: the operation mode is selected via this enum. Manual or automatic operation mode.

bManualln: digital input variable whose state is forwarded to the output *bOut* if *eCtrlMode* = *eHVACCtrlMode_Manual*.

VAR_OUTPUT

```
bOut            :BOOL;
```

bOut: digital output variable.

3.6.20 FB_HVACPowerMeasurementKL3403**Application**

This function block serves to control a 3-phase power measurement terminal (KL/KS3403). The terminal data is read out and all variables that depend on it are derived. The Bus Terminal KL3403 enables the measurement of all relevant electrical data of the supply network. The voltage is measured via the direct connection of L1, L2, L3 and N. The current of the three phases L1, L2 and L3 is fed via simple current transformers. All measured currents and voltages are available as RMS values. In the KL3403 version, the active power and the energy consumption for each phase are calculated. Through the relationship of the RMS values of voltage U * current I and the active power P , all other information such as apparent power or phase shift angle $\cos \varphi$ can be derived. For each fieldbus, KL3403 provides a comprehensive network

analysis and an energy management option. The data is read out in 8 groups one after the other. Dependent values are calculated cyclically. The energy measurement is read out from the terminal as a 32-bit value with the overflow from register 1. The input **bDelEnergyConsumption** can be used to delete this value.

The results of the power measurement are available in the output structure *ST_HVACPowerMeasurement*. The structure is 120 bytes long.



The input and output variables *wDataL1*, *wDataL2*, *wDataL3*, *bySBL1*, *bySBL2*, *bySBL3*, *byCBL1*, *byCBL2*, *byCBL3*, *iDataOutL1*, *iDataOutL2* and *iDataOutL3* must be linked with the KL3403 Bus Terminal. They are needed in order to obtain all the data from the terminal.

TYPE ST_HVACPowerMeasurement

Name	: Type	Unit	Description
STRUCT			
<i>diIL1</i> , <i>diIL2</i> , <i>diIL3</i>	: DINT;	A	I _{eff} Strom (Effektivwert) Auflösung: 0,1 A
<i>diUL1</i> , <i>diUL2</i> , <i>diUL3</i>	: DINT;	V	U _{eff} Spannung (Effektivwert) Auflösung: 0,1 V
<i>diPL1</i> , <i>diPL2</i> , <i>diPL3</i>	: DINT;	kW	P Wirkleistung pro Phase Auflösung: 0,1 kW
<i>diPg</i>	: DINT;	kW	P _{ges} Wirkleistung Auflösung: 0,1 kW
<i>diCosPhiL1</i> , <i>diCosPhiL2</i> , <i>diCosPhiL3</i>	: DINT;		cosPhi Leistungsfaktor pro Phase Auflösung: 0,01
<i>diCosPhi</i>	: DINT;		cosPhiges Leistungsfaktor Auflösung: 0,01
<i>diWL1</i> , <i>diWL2</i> , <i>diWL3</i>	: DINT;	kWh	Energieverbrauch Auflösung: 1 kWh
<i>diWg</i>	: DINT;	kWh	Energieverbrauch Auflösung: 1 kWh
<i>diImaxL1</i> , <i>diImaxL2</i> , <i>diImaxL3</i>	: DINT;	A	I _{max} Spitzenwert des Stroms Auflösung: 0,1 A
<i>diUmaxL1</i> , <i>diUmaxL2</i> , <i>diUmaxL3</i>	: DINT;	V	U _{max} Spitzenwert der Spannung Resolution: 0,1 V
<i>diPmaxL1</i> , <i>diPmaxL2</i> , <i>diPmaxL3</i>	: DINT;	kW	P _{max} Spitzenwert der Wirkleistung Resolution: 0,1 kW
<i>diSg</i>	: DINT;	kVA	S _{ges} Scheinleistung Resolution: 0,1 kVA
<i>diQg</i>	: DINT;	kvar	Q _{ges} Blindleistung Resolution: 0,1 kvar
<i>dummy</i>	: DINT;		Reserve, füllt die Struktur auf 120Bytes auf
END_STRUCT			

VAR_INPUT

<i>wDataL1</i> , <i>wDataL2</i> , <i>wDataL3</i>	: WORD;
<i>bySBL1</i> , <i>bySBL2</i> , <i>bySBL3</i>	: BYTE;
<i>diCurrTransFactor</i>	: DINT;
<i>eModePowerMeasurement</i>	: E_HVACPowerMeasurementMode;
<i>bDelEnergyConsumption</i>	: BOOL;

wDataLx: input data from the three channels of the KL3403.

bySBLx: status byte of the three channels of the KL3403. Reports which value can be read via the input (*wDataLx*).

diCurrTransFactor: transformation factor of the current transformer, used for conversion to the actual string current and the associated values.

Since the measured end value always results in dec. 1000 (100.0%) regardless of the terminal type, the primary end value of the transformer must be specified for the factor.

Example KL3403-0000: a 400/1A transformer results in a *diCurrTransFactor* of 400. With an internal resolution of 0.001 A, a measured value of max. 1 A results in an end value of dec. 1000 (100.0%) * *diCurrTransFactor* = 400 A.

Example KL3403-0010: a 400/5A transformer results in a *diCurrTransFactor* of 400. With an internal resolution of 0.005 A, a measured value of max. 5 A results in an end value of dec. 1000 (100.0%) * *diCurrTransFactor* = 400 A.

eModePowerMeasurement: if the value of this parameter lies between 1 and 8, the automatic read-out of all data is interrupted. Only the corresponding selected measured variable is read per cycle.

TYPE E_HVACPowerMeasurementMode:

```
(
eHVACPowerMeasurementMode_AutoAllValues := 0,
eHVACPowerMeasurementMode_Current := 1,
eHVACPowerMeasurementMode_Voltage := 2,
eHVACPowerMeasurementMode_EffectivePower := 3,
```

```

eHVACPowerMeasurementMode_PowerFactor := 4,
eHVACPowerMeasurementMode_EnergyConsumption := 5,
eHVACPowerMeasurementMode_PeakCurrentValu := 6, 1)
eHVACPowerMeasurementMode_PeakVoltageValue := 7, 1)
eHVACPowerMeasurementMode_PeakPowerValue := 81)
);
END_TYPE

```

bDelEnergyConsumption : a positive edge at this input deletes the energy consumption in the EEPROM. The energy consumption is counted in RAM and saved every 15 minutes in the EEPROM. It is retained there even if the KL3403 is switched off.

¹⁾ The minimum and peak values are deleted when the KL3403 is switched off.

VAR_OUTPUT

```

BDelEnergyMeasuredValuesBusy      : BOOL;
bError                             : BOOL;
iErrID                             : UDINT;
byCBL1, byCBL2, byCBL3            : BYTE;
iDataOutL1, iDataOutL2, iDataOutL3: INT;
stQ_PowerMeasurement              : ST_HVACPowerMeasurement;

```

bDelEnergyMeasuredValuesBusy: since the measured energy values are deleted from the internal EEPROM, no value is updated during this time and the flag *bDelEnergyMeasuredValuesBusy* shows TRUE.

bError: if TRUE, an error has occurred in the register communication.

iErrID: error ID in the register communication

0x100 Timeout error. The permissible execution time has been exceeded.

0x200 Parameter error (e.g. with an invalid register number).

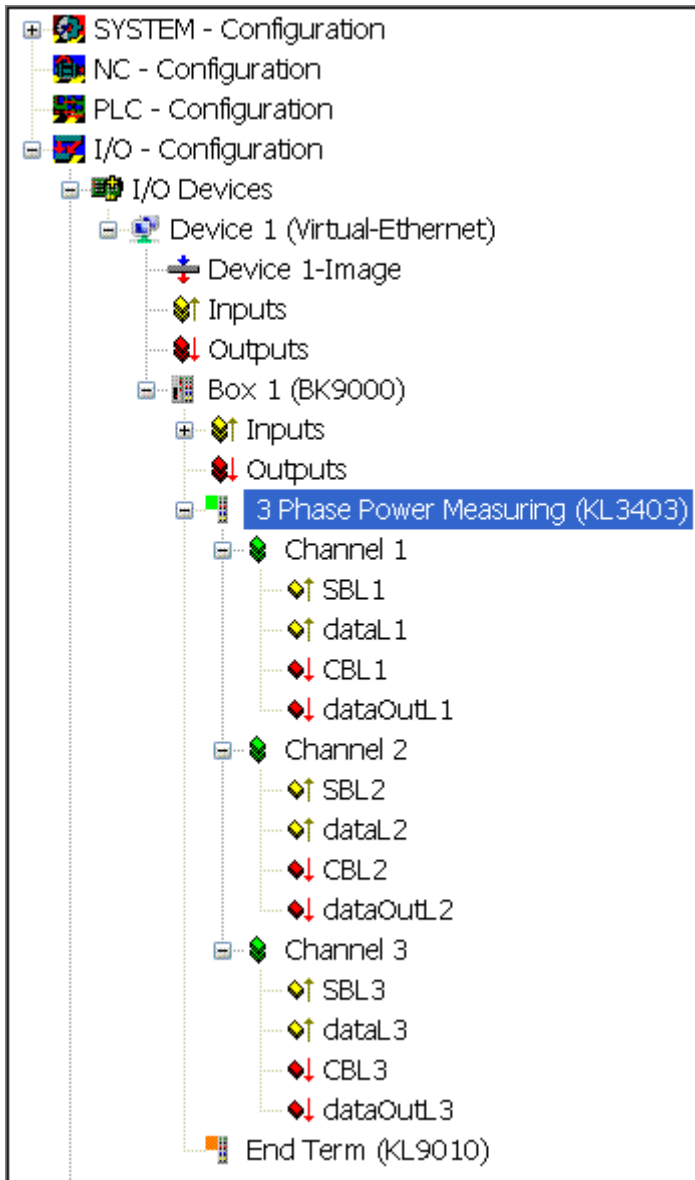
0x300 The read value differs from the written value (write access to this register possibly not allowed or failed)

byCBLx: this output serves the selection of the desired input value and register communication in order to delete the energy consumption.

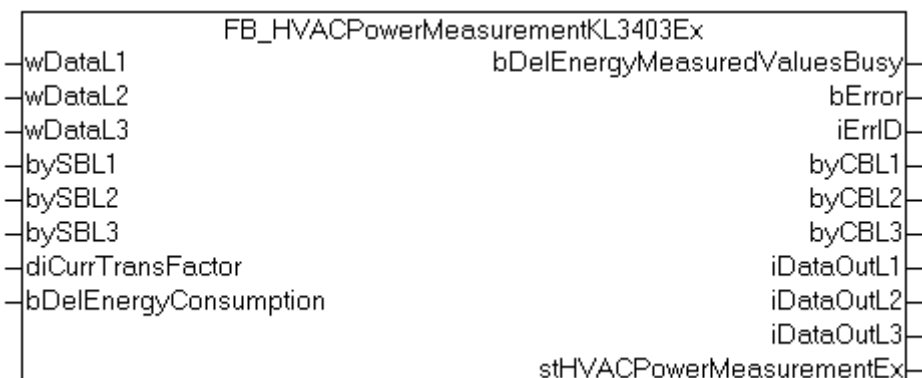
iDataOutLx: this output serves the register communication in order to delete the energy consumption.

stQ_PowerMeasurement: the results of the power measurement as an output data structure.

The example shows how a KL3403 terminal must be linked with the input and output variables of the PLC



3.6.21 FB_HVACPowerMeasurementKL3403Ex



Application

This function block serves to control a 3-phase power measurement terminal (KL/KS3403). The terminal data is read out and all variables that depend on it are derived. The Bus Terminal KL3403 enables the measurement of all relevant electrical data of the supply network. The voltage is measured via the direct

connection of L1, L2, L3 and N. The current of the three phases L1, L2 and L3 is fed via simple current transformers. All measured currents and voltages are available as RMS values. In the KL3403 version, the active power and the energy consumption for each phase are calculated. Through the relationship of the RMS values of voltage U * current I and the active power P , all other information such as apparent power or phase shift angle $\cos \varphi$ can be derived. For each fieldbus, KL3403 provides a comprehensive network analysis and an energy management option. The data is read out in 8 groups one after the other. Dependent values are calculated cyclically. The energy measurement is read out from the terminal as a 32-bit value with the overflow from register 1. The input **bDelEnergyConsumption** can be used to delete this value.

The results of the power measurement are available in the output structure *ST_HVACPowerMeasurementEx*.

The difference compared with the previous function block *FB_HVACPowerMeasurementKL3403* lies in the output format for the results. The results are available in the structure *ST_HVACPowerMeasurementEx* in LREAL format. The output was extended by the frequencies of the three phases.



The input and output variables *wDataL1*, *wDataL2*, *wDataL3*, *bySBL1*, *bySBL2*, *bySBL3*, *byCBL1*, *byCBL2*, *byCBL3*, *iDataOutL1*, *iDataOutL2* and *iDataOutL3* must be linked with the KL3403 Bus Terminal. They are needed in order to obtain all the data from the terminal.

TYPE *ST_HVACPowerMeasurementEx*

Name	: Type	Unit	Description
STRUCT			
<i>fIL1</i> , <i>fIL2</i> , <i>fIL3</i>		: LREAL;	A Ieff Strom (Effektivwert)
<i>fIg</i>	: LREAL;	A	Gesamt Strom (Effektivwert)
<i>fUL1</i> , <i>fUL2</i> , <i>fUL3</i>		: LREAL;	V Ueff Spannung (Effektivwert)
<i>fPL1</i> , <i>fPL2</i> , <i>fPL3</i>		: LREAL;	kW P Wirkleistung pro Phase
<i>fPg</i>	: LREAL;	kW	Pges Wirkleistung
<i>fCosPhiL1</i> , <i>fCosPhiL2</i> , <i>fCosPhiL3</i>		: LREAL;	cosPhi Leistungsfaktor pro Phase
<i>fCosPhi</i>	: LREAL;		cosPhiges Leistungsfaktor
<i>fWL1</i> , <i>fWL2</i> , <i>fWL3</i>		: LREAL;	kWh Energieverbrauch
<i>fWg</i>	: LREAL;	kWh	Energieverbrauch
<i>fImaxL1</i> , <i>fImaxL2</i> , <i>fImaxL3</i>		: LREAL;	A Imax Spitzenwert des Stroms
<i>fUmaxL1</i> , <i>fUmaxL2</i> , <i>fUmaxL3</i>		: LREAL;	V Umax Spitzenwert der Spannung
<i>fPmaxL1</i> , <i>fPmaxL2</i> , <i>fPmaxL3</i>		: LREAL;	kW Pmax Spitzenwert der Wirkleistung
<i>fSg</i>	: LREAL;	kVA	Sges Scheinleistung
<i>fQg</i>	: LREAL;	kvar	Qges Blindleistung
<i>fFrequencyL1</i> , <i>fFrequencyL2</i> , <i>fFrequencyL3</i>		: LREAL;	Hz Frequenz
END_STRUCT			

VAR_INPUT

wDataL1, *wDataL2*, *wDataL3* : WORD;
bySBL1, *bySBL2*, *bySBL3* : BYTE;
diCurrTransFactor : LREAL;
bDelEnergyConsumption : BOOL;

wDataLx: input data from the three channels of the KL3403.

bySBLx: status byte of the three channels of the KL3403. Reports which value can be read via the input (*wDataLx*).

diCurrTransFactor: transformation factor of the current transformer, used for conversion to the actual string current and the associated values.

Since the measured end value is always dec. 1000 (100.0%) irrespective of the terminal type, the primary end value of the transformer must be specified for the factor.

Example KL3403-0000: a 400/1A transformer results in a *diCurrTransFactor* of 400. With an internal resolution of 0.001 A, a measured value of max. 1 A results in an end value of dec. 1000 (100.0%) * *diCurrTransFactor* = 400 A.

Example KL3403-0010: a 400/5A transformer results in a *diCurrTransFactor* of 400. With an internal resolution of 0.005 A, a measured value of max. 5 A results in an end value of dec. 1000 (100.0%) * *diCurrTransFactor* = 400 A.

bDelEnergyConsumption : a positive edge at this input deletes the energy consumption in the EEPROM. The energy consumption is counted in RAM and saved every 15 minutes in the EEPROM. It is retained there even if the KL3403 is switched off.

¹⁾ The minimum and peak values are deleted when the KL3403 is switched off.

VAR_OUTPUT

```
BDelEnergyMeasuredValuesBusy : BOOL;  
bError                        : BOOL;  
iErrID                        : UDINT;  
byCBL1, byCBL2, byCBL3      : BYTE;  
iDataOutL1, iDataOutL2, iDataOutL3: INT;  
stQ_PowerMeasurementEx      : ST_HVACPowerMeasurementEx;
```

bDelEnergyMeasuredValuesBusy: since the measured energy values are deleted from the internal EEPROM, no value is updated during this time and the flag *bDelEnergyMeasuredValuesBusy* shows TRUE.

bError: if TRUE, an error has occurred in the register communication.

iErrID: error ID in the register communication

0x100 Timeout error. The permissible execution time has been exceeded.

0x200 Parameter error (e.g. with an invalid register number).

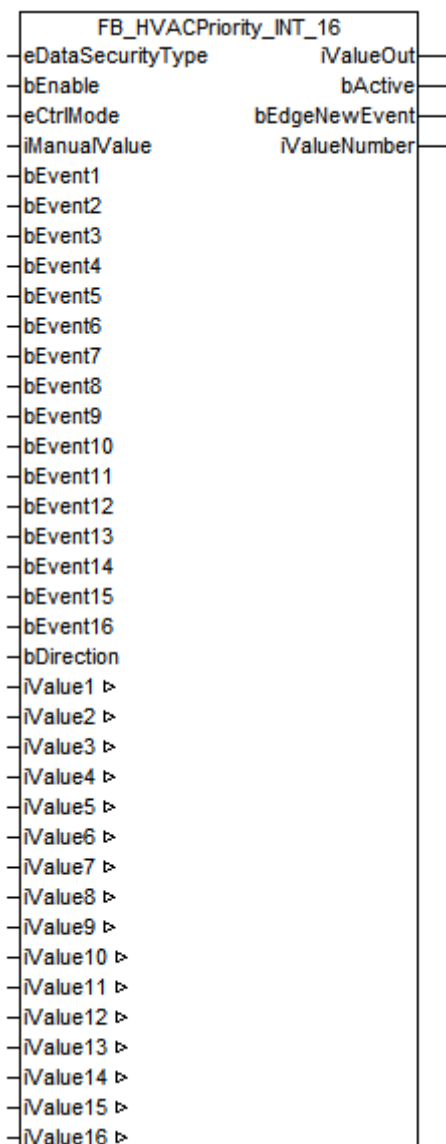
0x300 The read value differs from the written value (write access to this register possibly not allowed or failed)

byCBLx: this output serves the selection of the desired input value and register communication in order to delete the energy consumption.

iDataOutLx: this output serves the register communication in order to delete the energy consumption.

stQ_PowerMeasurementEx: the results of the power measurement as an output data structure.

3.6.22 FB_HVACPriority_INT_16



Application

This function block can be used to prioritize events or as a multiplexer. This selection is made with the help of the Enum *eCtrlMode*.

The function block can be used to prioritize events if *eCtrlMode* = *eHVACCtrlMode_Auto*. The output *iValueOut* is controlled by the occurring events of the inputs *bEvent1-16*, *iValue1-16* and the control direction of the prioritization *bDirection*.

bDirection = FALSE means that the events in ascending order from 1 to 16 have a higher priority. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent15* = TRUE (highest occurred event in [Table 1 \[▶ 415\]](#) column 1), then *iValueOut* has the value of *iValue15*.

bDirection = TRUE means that the events in descending order from 16 to 1 have a higher priority. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in [Table 2 \[▶ 415\]](#) column 1), then *iValueOut* has the value of *iValue1*.

The function block can be used as a multiplexer if *eCtrlMode* = *eHVACCtrlMode_Manual*. The value of *iManualValue* refers to one of the VAR_IN_OUT variables *iValue1-16*, whose value is output via *iValueOut*. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 2, then *iValueOut* = *iValue2*, see [Table 3 \[▶ 416\]](#).

NOTICE

A frequently changing variable may not be applied to the VAR_IN_OUT variables iValue1-16 if eDataSecurityType = eHVACDataSecurityType_Persistent. This would lead to premature wear of the storage medium of the controller. If the VAR_IN_OUT variables iValue1-16 change frequently and are not to be stored persistently, then eDataSecurityType must be eDataSecurityType_Idle.

Table 1: Prioritization of events in ascending order from 1 to 16

Table 1

It can be seen in Table 1 that *bDirection* = FALSE. This means that the events determine the output value *iValueOut* in ascending order from 1 to 16. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent15* = TRUE (highest occurring event in column 1), then *iValueOut* = *iValue15*.

bEnable = TRUE eCtrlMode = eHVACCtrlMode_Auto bDirection = FALSE						
bEvent1	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent2	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent3	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent4	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent5	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent6	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent7	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent8	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent9	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent10	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent11	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
bEvent12	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent13	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent14	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
bEvent15	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent16	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
iValueOut	iValue15	iValue8	iValue16	iValue14	0	iValue16
iValueNumber	15	8	16	14	0	16

Table 2: Prioritization of events in descending order from 16 to 1

Table 2

It can be seen in Table 2 that *bDirection* = TRUE. This means that the events determine the output value *iValueOut* in descending order from 16 to 1. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in [Table 2](#) [▶ 415] column 1), then *iValueOut* = *iValue1*.

bEnable = TRUE eCtrlMode = eHVACCtrlMode_Auto bDirction = TRUE						
bEvent1	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent2	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent3	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent4	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent5	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent6	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent7	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent8	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent9	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent10	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent11	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
bEvent12	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent13	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent14	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
bEvent15	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent16	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
iValueOut	iValue1	iValue1	iValue9	iValue3	0	iValue1
iValueNumber	1	1	9	3	0	1

Table 3: Multiplexer

Table 3

If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 2, then *iValueOut* = 2.

bEnable = TRUE eCtrlMode = eHVACCtrlMode_Manual						
iManualValue	2	3	5	15	0	16
iValueOut	iValue2	iValue3	iValue5	iValue15	0	iValue16
iValueNumber	2	3	5	15	0	16

Application example

Download	Required library
TcHVAC.pro [▶ 531]	TcHVAC.lib


VAR_INPUT

```

eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
bEvent1 - bEvent8 : BOOL;
eCtrlMode         : E_HVACCtrlMode;
iManualValue      : INT;           0..16
bDirection        : BOOL;
    
```

eDataSecurityType: if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bEnable: the function block is enabled via TRUE. If `bEnable = FALSE`, then `iValueOut` is set constantly to 0.

eCtrlMode: the operation mode of the function block is specified by means of the enum.

If `eCtrlMode = eHVACCtrlMode_Auto`, the function block represents a prioritization of events. If `bEnable = TRUE AND eCtrlMode = eHVACCtrlMode_Auto` then the output `iValueOut` is controlled by the occurring events of the inputs `bEvent1-16`, `iValue1-16` and the control direction of the prioritization `bDirection`, see [Table 1 \[▶ 415\]](#) and [Table 2 \[▶ 415\]](#)

If `eCtrlMode = eHVACCtrlMode_Manual`, the function block represents a multiplexer. If `bEnable = TRUE AND eCtrlMode = eHVACCtrlMode_Manual`, then the value of output `iValueOut` is controlled by `iManualValue`. If `iManualProfile = 5`, then `iValueOut = iValue5`, see [Table 3 \[▶ 416\]](#)

iManualProfile: if the operation mode `eCtrlMode = eHVACCtrlMode_Manual` (multiplexer) AND `bEnable = TRUE AND bError = FALSE`, then the value of the output `iValueOut` is controlled by `iManualValue`. If `iManualProfile = 5`, then `iValueOut = iValue5`, see [Table 3 \[▶ 416\]](#)

bEvent1-16: the output `iValueOut` is controlled by the occurring events of the inputs `bEvent1-16`, `iValue1-16` and the control direction of the prioritization `bDirection`.

`bDirection = FALSE` means that the events in ascending order from 1 to 16 determine the output value `iValueOut`. If `bEnable = TRUE AND eCtrlMode = eHVACCtrlMode_Auto AND bDirection = FALSE AND bEvent15 = TRUE` (highest event occurred in [Table 1 \[▶ 415\]](#) column 1), then `iValueOut = iValue15`.

`bDirection = TRUE` means that the events in descending order from 16 to 1 determine the output value `iValueOut`. If `bEnable = TRUE AND eCtrlMode = eHVACCtrlMode_Auto AND bDirection = TRUE AND bEvent1 = TRUE` (lowest occurred event in [Table 2 \[▶ 415\]](#) column 1), then `iValueOut = iValue1`.

The variables `bEvent1-16` and `bDirection` are only taken into account if `eCtrlMode = eHVACCtrlMode_Auto`.

bDirection: the function block can be used to prioritize events if `eCtrlMode = eHVACCtrlMode_Auto`. The output `iValueOut` is controlled by the occurring events of the inputs `bEvent1-16`, `iValue1-16` and the control direction of the prioritization `bDirection`.

`bDirection = FALSE` means that the events in ascending order from 1 to 16 determine the output value `iValueOut`. If `bEnable = TRUE AND eCtrlMode = eHVACCtrlMode_Auto AND bDirection = FALSE AND bEvent15 = TRUE` (highest event occurred in [Table 1 \[▶ 415\]](#) column 1), then `iValueOut = iValue15`.

`bDirection = TRUE` means that the events in descending order from 16 to 1 determine the output value `iValueOut`. If `bEnable = TRUE AND eCtrlMode = eHVACCtrlMode_Auto AND bDirection = TRUE AND bEvent1 = TRUE` (lowest occurred event in [Table 2 \[▶ 415\]](#) column 1), then `iValueOut = iValue1`.

The variables `bEvent1-16` and `bDirection` are only taken into account if `eCtrlMode = eHVACCtrlMode_Auto`.

VAR_OUTPUT

```
iValueOut      : INT;
bActive        : BOOL;
bEdgeNewEvent  : BOOL;
iValueNumber   : INT;
```

iValueOut: this function block can be used to prioritize events or as a multiplexer. This selection is made with the help of the Enum `eCtrlMode`.

The function block can be used to prioritize events if `eCtrlMode = eHVACCtrlMode_Auto`. The output `iValueOut` is controlled by the occurring events of the inputs `bEvent1-16`, `iValue1-16` and the control direction of the prioritization `bDirection`.

`bDirection = FALSE` means that the events in ascending order from 1 to 16 determine the output value `iValueOut`. If `bEnable = TRUE AND eCtrlMode = eHVACCtrlMode_Auto AND bDirection = FALSE AND bEvent15 = TRUE` (highest event occurred in [Table 1 \[▶ 415\]](#) column 1), then `iValueOut = iValue15`.

bDirection = TRUE means that the events in descending order from 16 to 1 determine the output value *iValueOut*. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in [Table 2 \[▶ 415\]](#) column 1), then *iValueOut* = *iValue1*.

The function block can be used as a multiplexer if *eCtrlMode* = *eHVACCtrlMode_Manual*. The value of *iManualValue* refers to one of the VAR_IN_OUT variables *iValue1-16*, whose value is output via *iValueOut*. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 2, then *iValueOut* = *iValue2*, see [Table 3 \[▶ 416\]](#).

bActive: *bActive* becomes TRUE, if

1. *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* and one of the input variables *bEvent1-16* = TRUE.
2. *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* > 0

bEdgeNewEvent: is TRUE for one PLC cycle if

1. *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* and the decisive event (*bEvent1-16*) has changed to control the output *iValueOut* using *iValue1-16*.
2. *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* and with each change of *iManualValue*

iValueNumber: the variable *iValueNumber* indicates the variable from which the value on *iValueOut* is output. If *iValueOut* = *iValue7*, then *iValueNumber* = 7, see [Table 1 \[▶ 415\]](#), [Table 2 \[▶ 415\]](#) and [Table 3 \[▶ 416\]](#)

VAR_IN_OUT

Name	Type	Persistent
<i>iValue1-16</i>	: INT;	X

iValue1-16: the value of the output variable *iValueOut* is determined by the variables *iValue1* to *iValue16*. *iValueOut* = *iValueX*

If the function block is used to prioritize events, each of the variables *iValue1-16* is assigned to an event. *iValue1* is assigned to the event *bEvent1*, *iValue2* to the event *bEvent2*, *iValue3* to the event *bEvent3*,..., *iValue16* to the event *bEvent16*

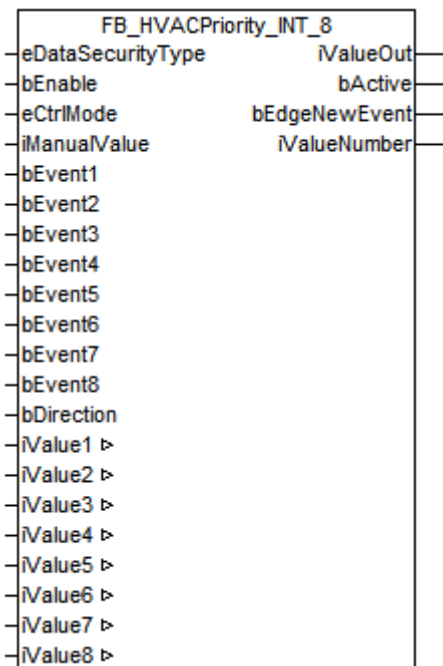
If the function block is used as a multiplexer, then each of the variables *iValue1-16* is assigned to the value of *iManualValue*. If *iManualValue* = 1, then *iValueOut* = *iValue1*. If *iManualValue* = 2, then *iValueOut* = *iValue2*.... If *iManualValue* = 16, then *iValueOut* = *iValue16*.

The variable is saved persistently.

Documents about this

-  [example_persistent_e.zip \(Resources/zip/11659714827.zip\)](#)

3.6.23 FB_HVACPriority_INT_8



Application

This function block can be used to prioritize events or as a multiplexer. This selection is made with the help of the Enum *eCtrlMode*.

The function block can be used to prioritize events if *eCtrlMode* = *eHVACCtrlMode_Auto*. The output *iValueOut* is controlled by the occurring events of the inputs *bEvent1-8*, *iValue1-8* and the control direction of the prioritization *bDirection*.

bDirection = FALSE means that the events in ascending order from 1 to 8 have a higher priority. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent7* = TRUE (highest occurred event in Table 1 [▶ 419] column 1), then *iValueOut* has the value of *iValue7*.

bDirection = TRUE means that the events in descending order from 8 to 1 have a higher priority. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in Table 2 [▶ 419] column 1), then *iValueOut* has the value of *iValue1*.

The function block can be used as a multiplexer if *eCtrlMode* = *eHVACCtrlMode_Manual*. The value of *iManualValue* refers to one of the VAR_IN_OUT variables *iValue1-8*, whose value is output via *iValueOut*. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 2, then *iValueOut* = *iValue2*, see Table 3 [▶ 420].

NOTICE

A frequently changing variable may not be applied to the VAR_IN_OUT variables *iValue1-8* if *eDataSecurityType* = *eHVACDataSecurityType_Persistent*. This would lead to premature wear of the storage medium of the controller. If the VAR_IN_OUT variables *iValue1-8* change frequently and are not to be stored persistently, then *eDataSecurityType* must be *eDataSecurityType_Idle*.

Table 1: Prioritization of events in ascending order from 1 to 8

Table 1

It can be seen in Table 1 that *bDirection* = FALSE. This means that the events determine the output value *iValueOut* in ascending order from 1 to 8. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent7* = TRUE (highest occurring event in column 1), then *iValueOut* = *iValue7*.

bEnable = TRUE eCtrlMode = eHVACCtrlMode_Auto bDirction = FALSE						
bEvent1	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE
bEvent2	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE
bEvent3	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE
bEvent4	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent5	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent6	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
bEvent7	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
bEvent8	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
iValueOut	iValue7	iValue8	0	iValue6	iValue3	iValue5
iValueNumber	7	8	0	6	3	5

Table 2: Prioritization of events in descending order from 8 to 1

Table 2

It can be seen in Table 2 that *bDirection* = TRUE. This means that the events determine the output value *iValueOut* in descending order from 8 to 1. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in [Table 2](#) [▶ 420] column 1), then *iValueOut* = *iValue1*.

bEnable = TRUE eCtrlMode = eHVACCtrlMode_Auto bDirction = TRUE						
bEvent1	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE
bEvent2	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE
bEvent3	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE
bEvent4	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent5	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent6	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
bEvent7	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
bEvent8	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
iValueOut	iValue1	iValue1	0	iValue3	iValue1	iValue1
iValueNumber	1	1	0	3	1	1

Table 3: Multiplexer

Table 3

If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 2, then *iValueOut* = 2.

bEnable = TRUE eCtrlMode = eHVACCtrlMode_Manual						
iManualValue	2	3	5	7	0	8
iValueOut	iValue2	iValue3	iValue5	iValue7	0	iValue8
iValueNumber	2	3	5	7	0	8

Application example


Download	Required library
TcHVAC.pro [▶ 531]	TcHVAC.lib

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
bEvent1 - bEvent8 : BOOL;
eCtrlMode         : E_HVACCtrlMode;
iManualValue      : INT;           0..8
bDirection        : BOOL;
```

eDataSecurityType: if $eDataSecurityType = eHVACDataSecurityType_Persistent$, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If $eDataSecurityType = eHVACDataSecurityType_Idle$ the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if $eDataSecurityType = eHVACDataSecurityType_Persistent$. It would lead to early wear of the flash memory.

bEnable: the function block is enabled via TRUE. If $bEnable = FALSE$, then $iValueOut$ is set constantly to 0.

eCtrlMode: the operation mode of the function block is specified by means of the enum. If $eCtrlMode = eHVACCtrlMode_Auto$, the function block represents a prioritization of events. If $bEnable = TRUE$ AND $eCtrlMode = eHVACCtrlMode_Auto$ the output $iValueOut$ is controlled by the occurring events of the inputs $bEvent1-8$, $iValue1-8$ and the control direction of the prioritization $bDirection$, see [Table 1 \[▶ 419\]](#) and [Table 2 \[▶ 420\]](#)

If $eCtrlMode = eHVACCtrlMode_Manual$, the function block represents a multiplexer. If $bEnable = TRUE$ AND $eCtrlMode = eHVACCtrlMode_Manual$, then the value of output $iValueOut$ is controlled by $iManualValue$. If $iManualProfile = 5$, then $iValueOut = iValue5$, see [Table 3 \[▶ 420\]](#)

iManualProfile: if the operation mode $eCtrlMode = eHVACCtrlMode_Manual$ (multiplexer) AND $bEnable = TRUE$ AND $bError = FALSE$, then the value of the output $iValueOut$ is controlled by $iManualValue$. If $iManualProfile = 5$, then $iValueOut = iValue5$, see [Table 3 \[▶ 420\]](#)

bEvent1-8: the output $iValueOut$ is controlled by the occurring events of the inputs $bEvent1-8$, $iValue1-8$ and the control direction of the prioritization $bDirection$.

$bDirection = FALSE$ means that the events in ascending order from 1 to 8 determine the output value $iValueOut$. If $bEnable = TRUE$ AND $eCtrlMode = eHVACCtrlMode_Auto$ AND $bDirection = FALSE$ AND $bEvent7 = TRUE$ (highest event occurred in [Table 1 \[▶ 419\]](#) column 1), then $iValueOut = iValue7$.

$bDirection = TRUE$ means that the events in descending order from 8 to 1 determine the output value $iValueOut$. If $bEnable = TRUE$ AND $eCtrlMode = eHVACCtrlMode_Auto$ AND $bDirection = TRUE$ AND $bEvent1 = TRUE$ (lowest occurred event in [Table 2 \[▶ 419\]](#) Column 1), then $iValueOut = iValue1$.

The variables $bEvent1-8$ and $bDirection$ are only taken into account if $eCtrlMode = eHVACCtrlMode_Auto$.

bDirection: the function block can be used to prioritize events if $eCtrlMode = eHVACCtrlMode_Auto$. The output $iValueOut$ is controlled by the occurring events of the inputs $bEvent1-8$, $iValue1-8$ and the control direction of the prioritization $bDirection$.

$bDirection = FALSE$ means that the events in ascending order from 1 to 8 determine the output value $iValueOut$. If $bEnable = TRUE$ AND $eCtrlMode = eHVACCtrlMode_Auto$ AND $bDirection = FALSE$ AND $bEvent7 = TRUE$ (highest event occurred in [Table 1 \[▶ 419\]](#) column 1), then $iValueOut = iValue7$.

$bDirection = TRUE$ means that the events in descending order from 8 to 1 determine the output value $iValueOut$. If $bEnable = TRUE$ AND $eCtrlMode = eHVACCtrlMode_Auto$ AND $bDirection = TRUE$ AND $bEvent1 = TRUE$ (lowest occurred event in [Table 2 \[▶ 419\]](#) Column 1), then $iValueOut = iValue1$.

The variables $bEvent1-8$ and $bDirection$ are only taken into account if $eCtrlMode = eHVACCtrlMode_Auto$.

VAR_OUTPUT

```

iValueOut      : INT;
bActive        : BOOL;
bEdgeNewEvent  : BOOL;
iValueNumber   : INT;

```

iValueOut: this function block can be used to prioritize events or as a multiplexer. This selection is made with the help of the Enum *eCtrlMode*.

The function block can be used to prioritize events if *eCtrlMode* = *eHVACCtrlMode_Auto*. The output *iValueOut* is controlled by the occurring events of the inputs *bEvent1-8*, *iValue1-8* and the control direction of the prioritization *bDirection*.

bDirection = FALSE means that the events in ascending order from 1 to 8 determine the output value *iValueOut*. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent7* = TRUE (highest event occurred in [Table 1 \[▶ 419\]](#) column 1), then *iValueOut* = *iValue7*.

bDirection = TRUE means that the events in descending order from 8 to 1 determine the output value *iValueOut*. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in [Table 2 \[▶ 419\]](#) column 1), then *iValueOut* = *iValue1*.

The function block can be used as a multiplexer if *eCtrlMode* = *eHVACCtrlMode_Manual*. The value of *iManualValue* refers to one of the VAR_IN_OUT variables *iValue1-8*, whose value is output via *iValueOut*. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 2, then *iValueOut* = *iValue2*, see [Table 3 \[▶ 420\]](#).

bActive: *bActive* becomes TRUE, if

1. *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* and one of the input variables *bEvent1-8* = TRUE.
2. *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* > 0

bEdgeNewEvent: is TRUE for one PLC cycle if

1. *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* and the decisive event (*bEvent1-8*) has changed to control the output *iValueOut* using *iValue1-8*.
2. *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* and with each change of *iManualValue*

iValueNumber: the variable *iValueNumber* indicates the variable from which the value on *iValueOut* is output. If *iValueOut* = *iValue7*, then *iValueNumber* = 7, see [Table 1 \[▶ 419\]](#), [Table 2 \[▶ 420\]](#) and [Table 3 \[▶ 420\]](#)

VAR_IN_OUT

```

iValue1-8      : INT;

```

iValue1-8: the value of the output variable *iValueOut* is determined by the variables *iValue1* to *iValue8*.
iValueOut = *iValueX*

If the function block is used to prioritize events, each of the variables *iValue1-8* is assigned to an event. *iValue1* is assigned to the event *bEvent1*, *iValue2* to the event *bEvent2*, *iValue3* to the event *bEvent3*,..., *iValue8* to the event *bEvent8*

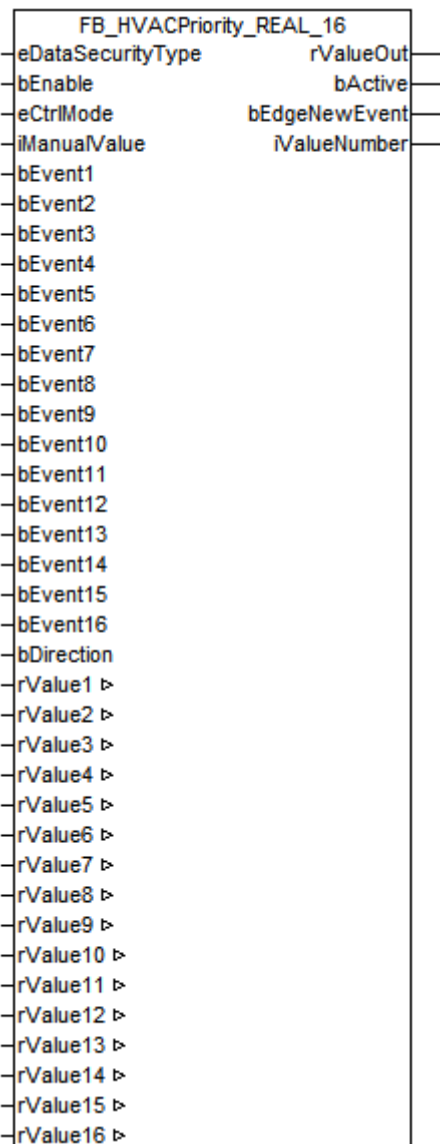
If the function block is used as a multiplexer, then each of the variables *iValue1-8* is assigned to the value of *iManualValue*. If *iManualValue* = 1, then *iValueOut* = *iValue1*. If *iManualValue* = 2, then *iValueOut* = *iValue2*.... If *iManualValue* = 8, then *iValueOut* = *iValue8*.

The variable is saved persistently.

Documents about this

- 📄 [example_persistent_e.zip \(Resources/zip/11659714827.zip\)](#)

3.6.24 FB_HVACPriority_REAL_16



Application

This function block can be used to prioritize events or as a multiplexer. This selection is made with the help of the Enum *eCtrlMode*.

The function block can be used to prioritize events if *eCtrlMode* = *eHVACCtrlMode_Auto*. The output *rValueOut* is controlled by the occurring events of the inputs *bEvent1-16*, *rValue1-16* and the control direction of the prioritization *bDirection*.

bDirection = FALSE means that the events in ascending order from 1 to 16 have a higher priority. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent15* = TRUE (highest occurred event in [Table 1 \[p. 424\]](#) column 1), then *rValueOut* has the value of *rValue15*.

bDirection = TRUE means that the events in descending order from 16 to 1 have a higher priority. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in [Table 2 \[p. 424\]](#) column 1), then *rValueOut* has the value of *rValue1*.

The function block can be used as a multiplexer if *eCtrlMode* = *eHVACCtrlMode_Manual*. The value of *iManualValue* refers to one of the VAR_IN_OUT variables *rValue1-16*, whose value is output via *rValueOut*. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 2, then *rValueOut* = *rValue2*, see [Table 3 \[p. 425\]](#).

NOTICE

A frequently changing variable may not be applied to the VAR_IN_OUT variables rValue1-16 if eDataSecurityType = eHVACDataSecurityType_Persistent. This would lead to premature wear of the storage medium of the controller. If the VAR_IN_OUT variables rValue1-16 change frequently and are not to be stored persistently, then eDataSecurityType must be eDataSecurityType_Idle.

Table 1: Prioritization of events in ascending order from 1 to 16

Table 1

It can be seen in Table 1 that *bDirection* = FALSE. This means that the events determine the output value *rValueOut* in ascending order from 1 to 16. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent15* = TRUE (highest occurring event in column 1), then *rValueOut* = *rValue15*.

bEnable = TRUE eCtrlMode = eHVACCtrlMode_Auto bDirection = FALSE						
bEvent1	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent2	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent3	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent4	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent5	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent6	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent7	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent8	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent9	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent10	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent11	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
bEvent12	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent13	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent14	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
bEvent15	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent16	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
rValueOut	rValue15	rValue8	rValue16	rValue14	0	rValue16
iValueNumber	15	8	16	14	0	16

Table 2: Prioritization of events in descending order from 16 to 1

Table 2

It can be seen in Table 2 that *bDirection* = TRUE. This means that the events determine the output value *rValueOut* in descending order from 16 to 1. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in [Table 2](#) [▶ 424] column 1), then *rValueOut* = *rValue1*.

bEnable = TRUE eCtrlMode = eHVACCtrlMode_Auto bDirction = TRUE						
bEvent1	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent2	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent3	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent4	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent5	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent6	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent7	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent8	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent9	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent10	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent11	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
bEvent12	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent13	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent14	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
bEvent15	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent16	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
rValueOut	rValue1	rValue1	rValue9	rValue3	0	rValue1
iValueNumber	1	1	9	3	0	1

Table 3: Multiplexer

Table 3

If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 2, then *rValueOut* = 2.

bEnable = TRUE eCtrlMode = eHVACCtrlMode_Manual						
iManualValue	2	3	5	15	0	16
rValueOut	rValue2	rValue3	rValue5	rValue15	0	rValue16
iValueNumber	2	3	5	15	0	16

Application example

Download	Required library
TcHVAC.pro [►_531]	TcHVAC.lib


VAR_INPUT

```

eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
bEvent1 - bEvent8 : BOOL;
eCtrlMode         : E_HVACCtrlMode;
iManualValue      : INT;           0..16
bDirection        : BOOL;
    
```

eDataSecurityType: if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bEnable: the function block is enabled via TRUE. If `bEnable = FALSE`, then `rValueOut` is set constantly to 0.

eCtrlMode: the operation mode of the function block is specified by means of the enum.

If `eCtrlMode = eHVACCtrlMode_Auto`, the function block represents a prioritization of events. If `bEnable = TRUE` AND `eCtrlMode = eHVACCtrlMode_Auto` the output `rValueOut` is controlled by the events of the inputs `bEvent1-16`, `rValue1-16` and the control direction of the prioritization `bDirection`, see [Table 1 \[▶ 424\]](#) and [Table 2 \[▶ 424\]](#)

If `eCtrlMode = eHVACCtrlMode_Manual`, the function block represents a multiplexer. If `bEnable = TRUE` AND `eCtrlMode = eHVACCtrlMode_Manual`, then the value of output `rValueOut` is controlled by `iManualValue`. If `iManualProfile = 5`, then `rValueOut = rValue5`, see [Table 3 \[▶ 425\]](#)

iManualProfile: if the operation mode `eCtrlMode = eHVACCtrlMode_Manual` (multiplexer) AND `bEnable = TRUE` AND `bError = FALSE`, then the value of the output `rValueOut` is controlled by `iManualValue`. If `iManualProfile = 5`, then `rValueOut = rValue5`, see [Table 3 \[▶ 425\]](#)

bEvent1-16: the output `rValueOut` is controlled by the occurring events of the inputs `bEvent1-16`, `rValue1-16` and the control direction of the prioritization `bDirection`.

`bDirection = FALSE` means that the events in ascending order from 1 to 16 determine the output value `rValueOut`. If `bEnable = TRUE` AND `eCtrlMode = eHVACCtrlMode_Auto` AND `bDirection = FALSE` AND `bEvent15 = TRUE` (highest event occurred in [Table 1 \[▶ 424\]](#) column 1), then `rValueOut = rValue15`.

`bDirection = TRUE` means that the events in descending order from 16 to 1 determine the output value `rValueOut`. If `bEnable = TRUE` AND `eCtrlMode = eHVACCtrlMode_Auto` AND `bDirection = TRUE` AND `bEvent1 = TRUE` (lowest occurred event in [Table 2 \[▶ 424\]](#) column 1), then `rValueOut = rValue1`.

The variables `bEvent1-16` and `bDirection` are only taken into account if `eCtrlMode = eHVACCtrlMode_Auto`.

bDirection: the function block can be used to prioritize events if `eCtrlMode = eHVACCtrlMode_Auto`. The output `rValueOut` is controlled by the occurring events of the inputs `bEvent1-16`, `rValue1-16` and the control direction of the prioritization `bDirection`.

`bDirection = FALSE` means that the events in ascending order from 1 to 16 determine the output value `rValueOut`. If `bEnable = TRUE` AND `eCtrlMode = eHVACCtrlMode_Auto` AND `bDirection = FALSE` AND `bEvent15 = TRUE` (highest event occurred in [Table 1 \[▶ 424\]](#) column 1), then `rValueOut = rValue15`.

`bDirection = TRUE` means that the events in descending order from 16 to 1 determine the output value `rValueOut`. If `bEnable = TRUE` AND `eCtrlMode = eHVACCtrlMode_Auto` AND `bDirection = TRUE` AND `bEvent1 = TRUE` (lowest occurred event in [Table 2 \[▶ 424\]](#) column 1), then `rValueOut = rValue1`.

The variables `bEvent1-16` and `bDirection` are only taken into account if `eCtrlMode = eHVACCtrlMode_Auto`.

VAR_OUTPUT

```
rValueOut      : REAL;
bActive        : BOOL;
bEdgeNewEvent  : BOOL;
iValueOut      : INT;
```

rValueOut: this function block can be used to prioritize events or as a multiplexer. This selection is made with the help of the Enum `eCtrlMode`.

The function block can be used to prioritize events if `eCtrlMode = eHVACCtrlMode_Auto`. The output `rValueOut` is controlled by the occurring events of the inputs `bEvent1-16`, `rValue1-16` and the control direction of the prioritization `bDirection`.

`bDirection = FALSE` means that the events in ascending order from 1 to 16 determine the output value `rValueOut`. If `bEnable = TRUE` AND `eCtrlMode = eHVACCtrlMode_Auto` AND `bDirection = FALSE` AND `bEvent15 = TRUE` (highest event occurred in [Table 1 \[▶ 424\]](#) column 1), then `rValueOut = rValue15`.

bDirection = TRUE means that the events in descending order from 16 to 1 determine the output value *rValueOut*. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in [Table 2 \[▶ 424\]](#) column 1), then *rValueOut* = *rValue1*.

The function block can be used as a multiplexer if *eCtrlMode* = *eHVACCtrlMode_Manual*. The value of *iManualValue* refers to one of the VAR_IN_OUT variables *rValue1-16*, whose value is output via *rValueOut*. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 2, then *rValueOut* = *rValue2*, see [Table 3 \[▶ 425\]](#).

bActive: *bActive* becomes TRUE, if

1. *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* and one of the input variables *bEvent1-16* = TRUE.
2. *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* > 0

bEdgeNewEvent: is TRUE for one PLC cycle if

1. *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* and the decisive event (*bEvent1-16*) has changed to control the output *rValueOut* using *rValue1-16*.
2. *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* and with each change of *iManualValue*

iValueNumber: the variable *iValueNumber* indicates the variable from which the value on *rValueOut* is output. If *rValueOut* = *rValue7*, then *iValueNumber* = 7, see [Table 1 \[▶ 424\]](#), [Table 2 \[▶ 424\]](#) and [Table 3 \[▶ 425\]](#)

VAR_IN_OUT

```
rValue1-16 : REAL;
```

rValue1-16: the value of the output variable *rValueOut* is determined by the variables *rValue1* to *rValue16*.
rValueOut = *rValueX*

If the function block is used to prioritize events, each of the variables *rValue1-16* is assigned to an event. *rValue1* is assigned to the event *bEvent1*, *rValue2* to the event *bEvent2*, *rValue3* to the event *bEvent3*,..., *rValue16* to the event *bEvent16*

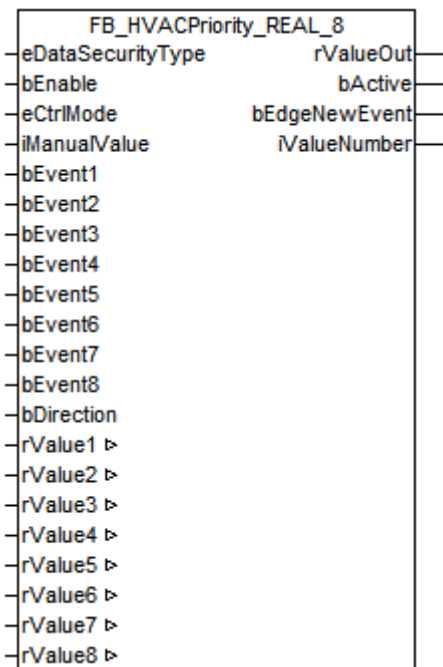
If the function block is used as a multiplexer, then each of the variables *rValue1-16* is assigned to the value of *iManualValue*. If *iManualValue* = 1, then *rValueOut* = *rValue1*. If *iManualValue* = 2, then *rValueOut* = *rValue2*.... If *iManualValue* = 16, then *rValueOut* = *rValue16*.

The variable is saved persistently.

Documents about this

- example_persistent_e.zip (Resources/zip/11659714827.zip)

3.6.25 FB_HVACPriority_REAL_8



Application

This function block can be used to prioritize events or as a multiplexer. This selection is made with the help of the Enum *eCtrlMode*.

The function block can be used to prioritize events if *eCtrlMode* = *eHVACCtrlMode_Auto*. The output *rValueOut* is controlled by the occurring events of the inputs *bEvent1-8*, *rValue1-8* and the control direction of the prioritization *bDirection*.

bDirection = FALSE means that the events in ascending order from 1 to 8 have a higher priority. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent7* = TRUE (highest occurred event in [Table 1 \[▶ 428\]](#) column 1), then *rValueOut* has the value of *rValue7*.

bDirection = TRUE means that the events in descending order from 8 to 1 have a higher priority. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in [Table 2 \[▶ 428\]](#) column 1), then *rValueOut* has the value of *rValue1*.

The function block can be used as a multiplexer if *eCtrlMode* = *eHVACCtrlMode_Manual*. The value of *iManualValue* refers to one of the VAR_IN_OUT variables *rValue1-8*, whose value is output via *rValueOut*. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 2, then *rValueOut* = *rValue2*, see [Table 3 \[▶ 429\]](#).

NOTICE

A frequently changing variable may not be applied to the VAR_IN_OUT variables *rValue1-8* if *eDataSecurityType* = *eHVACDataSecurityType_Persistent*. This would lead to premature wear of the storage medium of the controller. If the VAR_IN_OUT variables *rValue1-8* change frequently and are not to be stored persistently, then *eDataSecurityType* must be *eDataSecurityType_Idle*.

Table 1: Prioritization of events in ascending order from 1 to 8

Table 1

It can be seen in [Table 1](#) that *bDirection* = FALSE. This means that the events determine the output value *rValueOut* in ascending order from 1 to 8. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent7* = TRUE (highest occurring event in column 1), then *rValueOut* = *rValue7*.

bEnable = TRUE eCtrlMode = eHVACCtrlMode_Auto bDirction = FALSE						
bEvent1	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE
bEvent2	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE
bEvent3	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE
bEvent4	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent5	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent6	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
bEvent7	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
bEvent8	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
rValueOut	rValue7	rValue8	0	rValue6	rValue3	rValue5
iValueNumber	7	8	0	6	3	5

Table 2: Prioritization of events in descending order from 8 to 1

Table 2

It can be seen in Table 2 that *bDirection* = TRUE. This means that the events determine the output value *rValueOut* in descending order from 8 to 1. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in [Table 2](#) [► 429] column 1), then *rValueOut* = *rValue1*.

bEnable = TRUE eCtrlMode = eHVACCtrlMode_Auto bDirction = TRUE						
bEvent1	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE
bEvent2	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE
bEvent3	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE
bEvent4	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent5	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent6	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
bEvent7	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
bEvent8	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
rValueOut	rValue1	rValue1	0	rValue3	rValue1	rValue1
iValueNumber	1	1	0	3	1	1

Table 3: Multiplexer

Table 3

If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 2, then *rValueOut* = 2.

bEnable = TRUE eCtrlMode = eHVACCtrlMode_Manual						
iManualValue	2	3	5	1	0	8
rValueOut	rValue2	rValue3	rValue5	rValue1	0	rValue8
iValueNumber	2	3	5	1	0	8

Application example


Download	Required library
TcHVAC.pro [► 531]	TcHVAC.lib

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
bEvent1 - bEvent8 : BOOL;
eCtrlMode        : E_HVACCtrlMode;
iManualValue     : INT;           0..8
bDirection       : BOOL;
```

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bEnable: the function block is enabled via TRUE. If `bEnable = FALSE`, then `rValueOut` is set constantly to 0.

eCtrlMode: the operation mode of the function block is specified by means of the enum.

If `eCtrlMode = eHVACCtrlMode_Auto`, the function block represents a prioritization of events. If `bEnable = TRUE AND eCtrlMode = eHVACCtrlMode_Auto` the output `rValueOut` is controlled by the events of the inputs `bEvent1-8`, `rValue1-8` and the control direction of the prioritization `bDirection`, see [Table 1 \[▶ 428\]](#) and [Table 2 \[▶ 429\]](#)

If `eCtrlMode = eHVACCtrlMode_Manual`, the function block represents a multiplexer. If `bEnable = TRUE AND eCtrlMode = eHVACCtrlMode_Manual`, then the value of output `rValueOut` is controlled by `iManualValue`. If `iManualProfile = 5`, then `rValueOut = rValue5`, see [Table 3 \[▶ 429\]](#)

iManualProfile: if the operation mode `eCtrlMode = eHVACCtrlMode_Manual` (multiplexer) AND `bEnable = TRUE AND bError = FALSE`, then the value of the output `rValueOut` is controlled by `iManualValue`. If `iManualProfile = 5`, then `rValueOut = rValue5`, see [Table 3 \[▶ 429\]](#)

bEvent1-8: the output `rValueOut` is controlled by the occurring events of the inputs `bEvent1-8`, `rValue1-8` and the control direction of the prioritization `bDirection`.

`bDirection = FALSE` means that the events in ascending order from 1 to 8 determine the output value `rValueOut`. If `bEnable = TRUE AND eCtrlMode = eHVACCtrlMode_Auto AND bDirection = FALSE AND bEvent7 = TRUE` (highest event occurred in [Table 1 \[▶ 428\]](#) column 1), then `rValueOut = rValue7`.

`bDirection = TRUE` means that the events in descending order from 8 to 1 determine the output value `rValueOut`. If `bEnable = TRUE AND eCtrlMode = eHVACCtrlMode_Auto AND bDirection = TRUE AND bEvent1 = TRUE` (lowest occurred event in [Table 2 \[▶ 428\]](#) column 1), then `rValueOut = rValue1`.

The variables `bEvent1-8` and `bDirection` are only considered if `eCtrlMode = eHVACCtrlMode_Auto`.

bDirection: the function block can be used to prioritize events if `eCtrlMode = eHVACCtrlMode_Auto`. The output `rValueOut` is controlled by the occurring events of the inputs `bEvent1-8`, `rValue1-8` and the control direction of the prioritization `bDirection`.

`bDirection = FALSE` means that the events in ascending order from 1 to 8 determine the output value `rValueOut`. If `bEnable = TRUE AND eCtrlMode = eHVACCtrlMode_Auto AND bDirection = FALSE AND bEvent7 = TRUE` (highest event occurred in [Table 1 \[▶ 428\]](#) column 1), then `rValueOut = rValue7`.

`bDirection = TRUE` means that the events in descending order from 8 to 1 determine the output value `rValueOut`. If `bEnable = TRUE AND eCtrlMode = eHVACCtrlMode_Auto AND bDirection = TRUE AND bEvent1 = TRUE` (lowest occurred event in [Table 2 \[▶ 428\]](#) column 1), then `rValueOut = rValue1`.

The variables `bEvent1-8` and `bDirection` are only considered if `eCtrlMode = eHVACCtrlMode_Auto`.

VAR_OUTPUT

```
rValueOut      : REAL;
bActive        : BOOL;
bEdgeNewEvent  : BOOL;
iValueNumber   : INT;
```

rValueOut: this function block can be used to prioritize events or as a multiplexer. This selection is made with the help of the Enum *eCtrlMode*.

The function block can be used to prioritize events if *eCtrlMode* = *eHVACCtrlMode_Auto*. The output *rValueOut* is controlled by the occurring events of the inputs *bEvent1-8*, *rValue1-8* and the control direction of the prioritization *bDirection*.

bDirection = FALSE means that the events in ascending order from 1 to 8 determine the output value *rValueOut*. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent7* = TRUE (highest event occurred in [Table 1 \[▶ 428\]](#) column 1), then *rValueOut* = *rValue7*.

bDirection = TRUE means that the events in descending order from 8 to 1 determine the output value *rValueOut*. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in [Table 2 \[▶ 428\]](#) column 1), then *rValueOut* = *rValue1*.

The function block can be used as a multiplexer if *eCtrlMode* = *eHVACCtrlMode_Manual*. The value of *iManualValue* refers to one of the VAR_IN_OUT variables *rValue1-8*, whose value is output via *rValueOut*. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 2, then *rValueOut* = *rValue2*, see [Table 3 \[▶ 429\]](#).

bActive: *bActive* becomes TRUE, if

1. *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* and one of the input variables *bEvent1-8* = TRUE.
2. *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* > 0

bEdgeNewEvent: is TRUE for one PLC cycle if

1. *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* and the decisive event (*bEvent1-8*) has changed to control the output *rValueOut* using *rValue1-8*.
2. *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* and with each change of *iManualValue*

iValueNumber: the variable *iValueNumber* indicates the variable from which the value on *rValueOut* is output. If *rValueOut* = *rValue7*, then *iValueNumber* = 7, see [Table 1 \[▶ 428\]](#), [Table 2 \[▶ 429\]](#) and [Table 3 \[▶ 429\]](#)

VAR_IN_OUT

```
rValue1-8      : REAL;
```

rValue1-8: the value of the output variable *rValueOut* is determined by the variables *rValue1* to *rValue8*.
rValueOut = *rValueX*

If the function block is used to prioritize events, each of the variables *rValue1-8* is assigned to an event. *rValue1* is assigned to the event *bEvent1*, *rValue2* to the event *bEvent2*, *rValue3* to the event *bEvent3*,..., *rValue8* to the event *bEvent8*

If the function block is used as a multiplexer, then each of the variables *rValue1-8* is assigned to the value of *iManualValue*. If *iManualValue* = 1, then *rValueOut* = *rValue1*. If *iManualValue* = 2, then *rValueOut* = *rValue2*.... If *iManualValue* = 8, then *rValueOut* = *rValue8*.

The variable is saved persistently.

Documents about this

-  example_persistent_e.zip (Resources/zip/11659714827.zip)

3.6.26 FB_HVACOptimizedOn

Function block for the optimized switch-on of heating boilers and air conditioning systems in conjunction with switching time function blocks.

The explanations in this document relate to the heating behavior. The function block is used in the same way in conjunction with cooling devices.

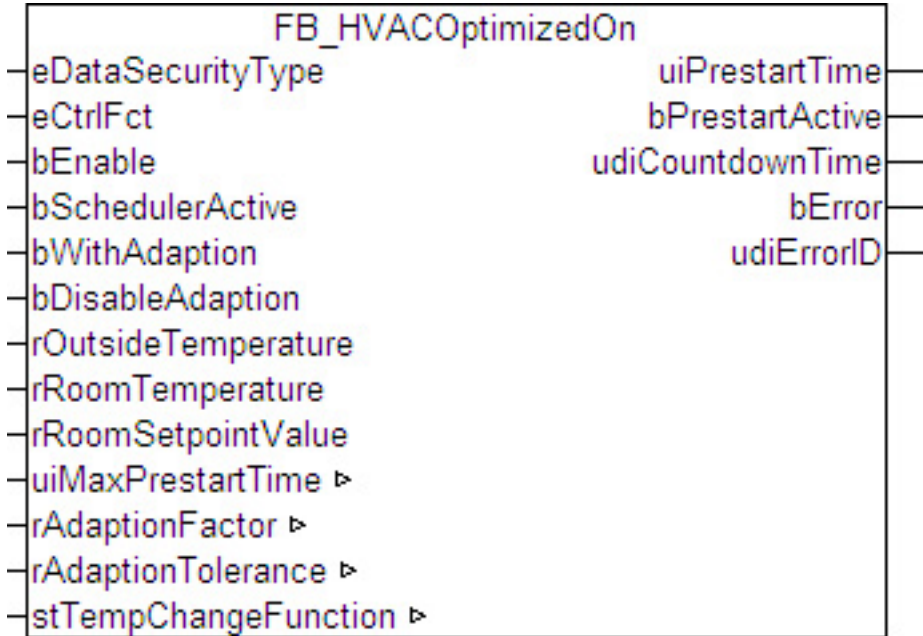
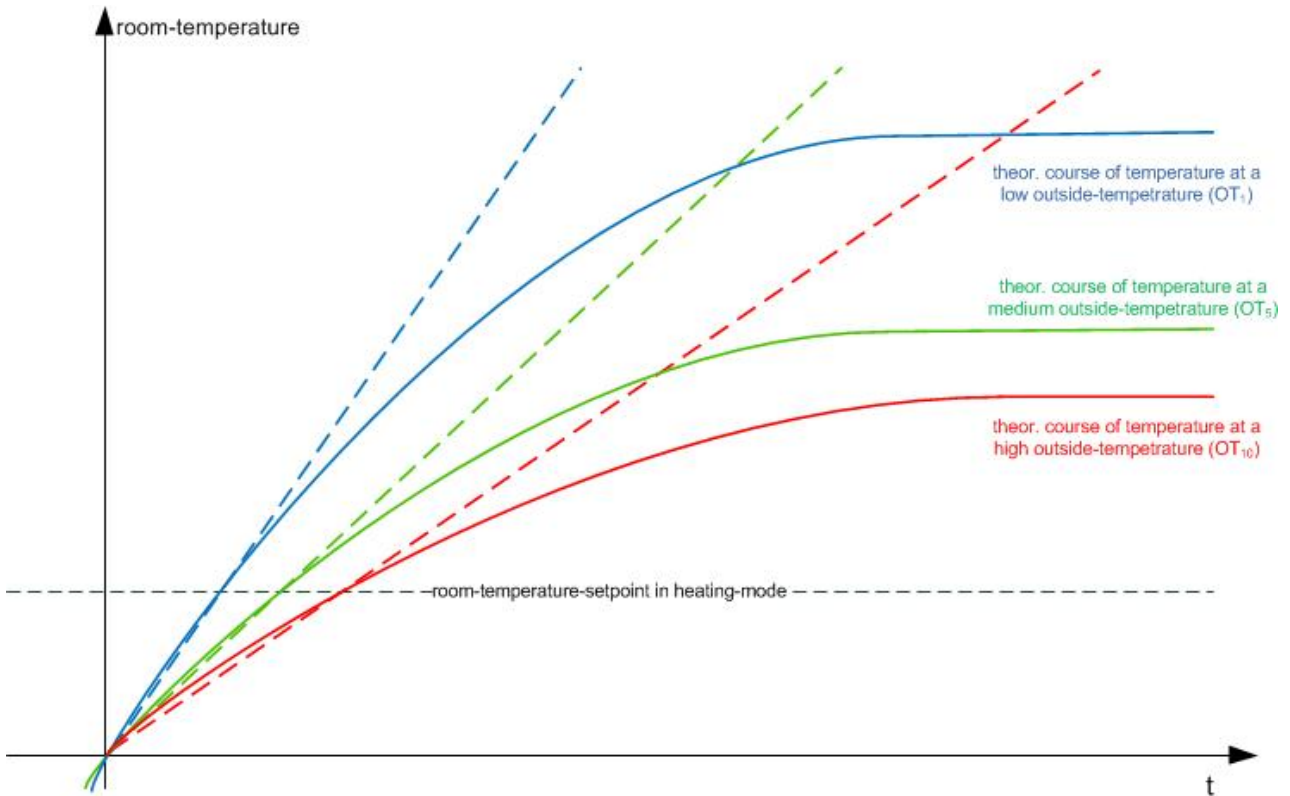


Fig. 17: FB_HVACOptimizedOn

Calculation of the pre-start time

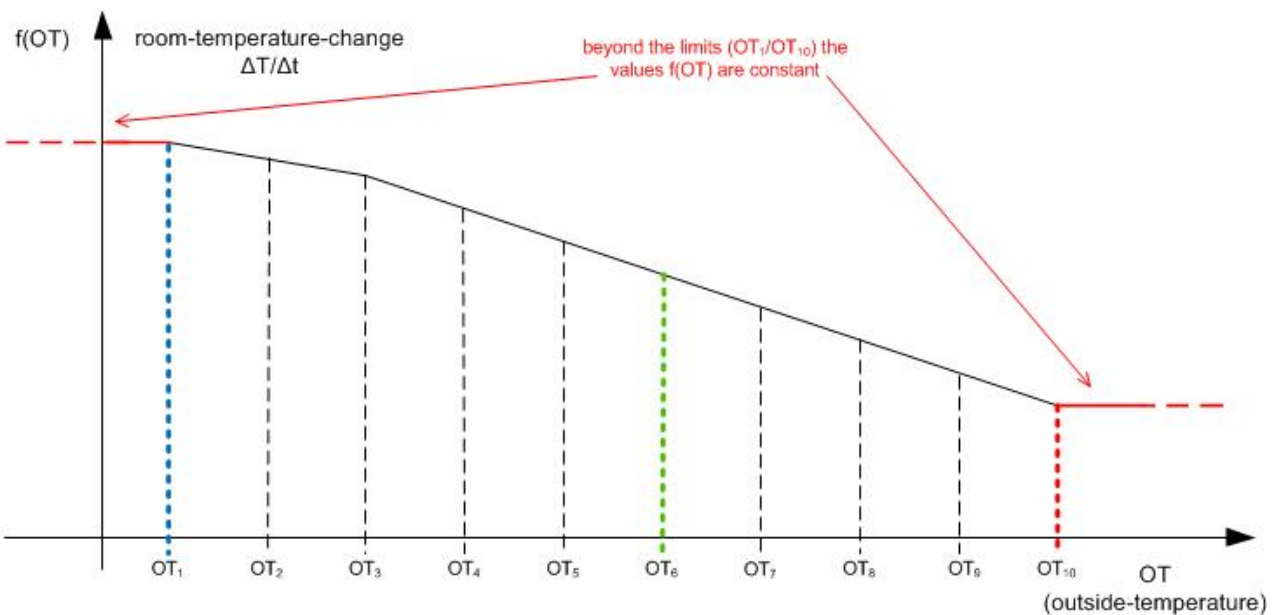
Company buildings and public facilities are generally unoccupied overnight and at weekends, so the heating boilers there often run only in standby mode. The goal of optimized switch-on is to start the heating boiler early enough for the building to have been heated sufficiently at a certain point in time. Conversely, this also applies to the cooling of the building during the summer months. This pre-start time is in fact not constant, but always dependent on various factors, such as outside temperature, room temperature difference (actual value to setpoint), and structural conditions, for example the building mass and the thermal insulation. The development of a universal equation valid for all buildings would be a very complex task and would fail just on account of the input of innumerable parameters. A self-learning (adaptive) approximation method is simpler and ultimately absolutely sufficient.

The flow temperature of the heating boiler depends on the outside temperature. Thus, different outside temperatures result in different heating curves, which essentially correspond to an exponential function:



The flow temperature is always considerably higher than the room temperature to be reached. To determine the pre-start time it is assumed that the area of the functions between the room temperature start value and room temperature set value is linear. This results in a characteristic temperature change $\Delta T/\Delta t$ for each outside temperature, which is shown here by the dotted line.

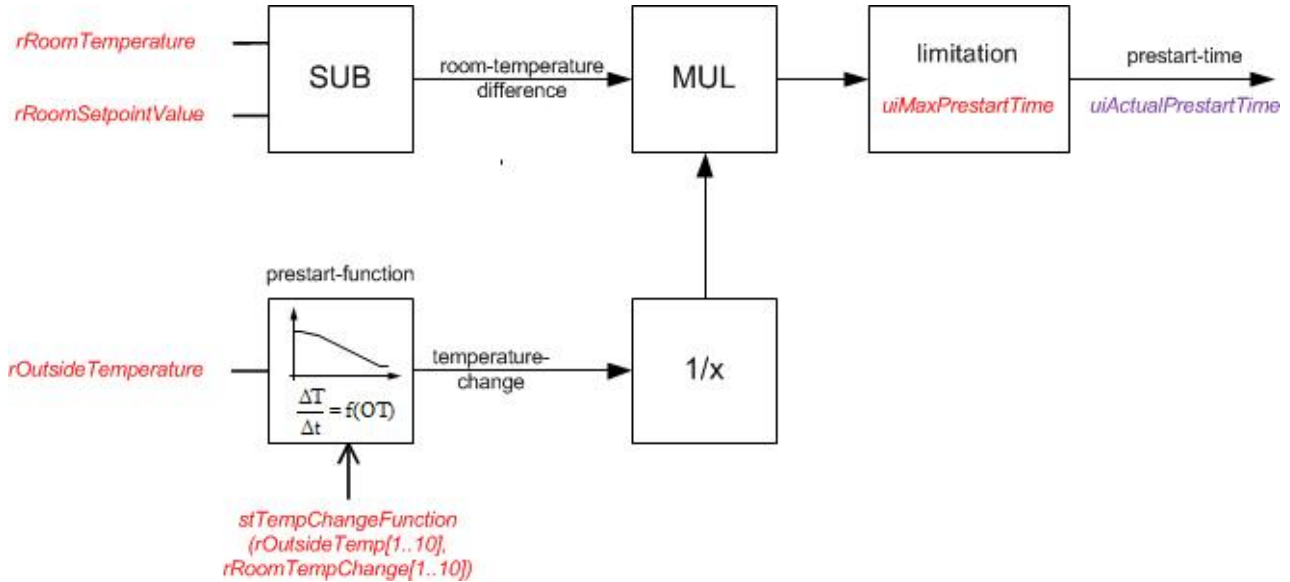
With the structure variable `stTempChangeFunction` [► 529], the function block `FB_HVACOptimizedOn` is based on a table in which the respectively expected room temperature change is assigned for 10 discrete outside temperatures. During the commissioning or the initial starting of the system, the pre-start function must be coarsely predefined - firstly to define the outside temperature range and secondly to accelerate the adaptation procedure. This entry is done with the function block `FB_HVACTempChangeFunctionEntry` [► 452]. The pre-start time can be determined approximately with these values. This temperature change function $f(AT)$ typically adopts the following course:



Function values within these 10 points are defined by linear equations, while values outside correspond to the function value $f(AT_1)$ or $f(AT_{10})$ respectively:

- $AT < AT_1$: $f(AT) = f(AT_1)$
- $AT > AT_{10}$: $f(AT) = f(AT_{10})$

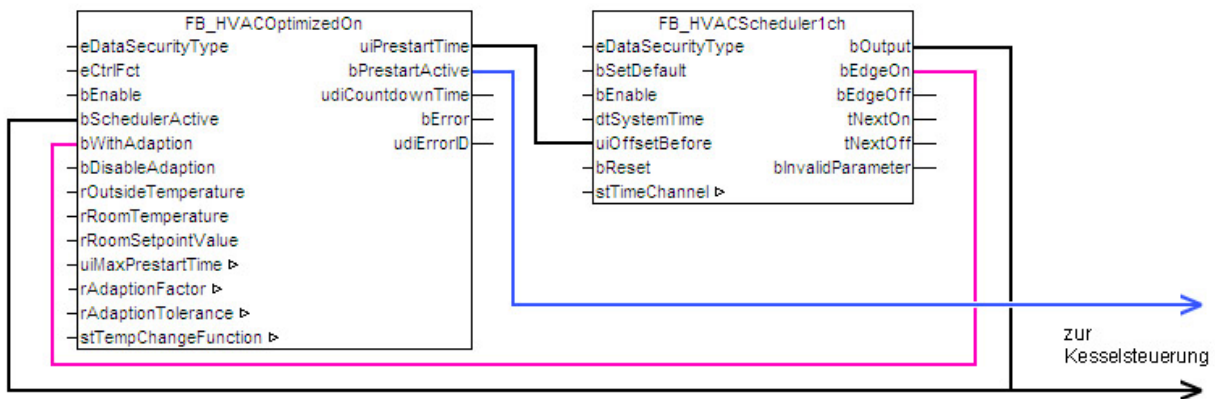
The pre-start time is then determined in accordance with the following sequence:



The input and output variables of the function block are shown here in red. Shown in purple here is the internal flag *uiActualPrestartTime* which contains the current pre-start time at each point in time. The output *uiPrestartTime* corresponds to this flag, but is "frozen" during the pre-start phase. This is explained in more detail below.

Function block linking and pre-start phase

From these parameters the function block will then continuously calculate a pre-start time, which it specifies to the timer function block. If, for example, a boiler is then to have heated up the building at 6 o'clock on Monday morning and the pre-start function block determines 60 minutes pre-start time at 5 o'clock in the morning, the time switch will immediately set the boiler to the appropriate heating mode. The heating behavior is then observed in a reference room over the pre-start phase and the pre-start curve is corrected accordingly.



The pre-start function block specifies a pre-start time for the time switch. If the timer switches the output *bOutput*, this is indicated to the pre-start function block via its input *bSchedulerActive*. Internally the pre-start function block then starts a countdown with the previously output pre-start time *uiPrestartTime* [min]. The

countdown either runs through to the end or is completed prematurely when the room setpoint temperature is reached and represents the pre-start phase. During the countdown, the output *bPrestartActive* is set, which can then be used, for example, for a quick start of the boiler.

The output *uiPrestartTime* follows continuously the above calculation - during a started countdown, however, it is kept constant so that a fluctuation in the outside temperature does not indicate a lower pre-start time and suddenly switch off the boiler.

If, in the case of a pre-start, the timer function block still sends the adaptation order as an edge (red line) to the optimization function block, then the latter will decide following the pre-start phase whether the previously determined pre-start time was precisely within a tolerance, or too short or too long, and correct the temperature change function accordingly. In doing so the pre-start time of the point whose outside temperature was nearest to the actual one at the beginning of the countdown is corrected upwards or downwards respectively. This process is called "[adaptation \[▶ 438\]](#)".

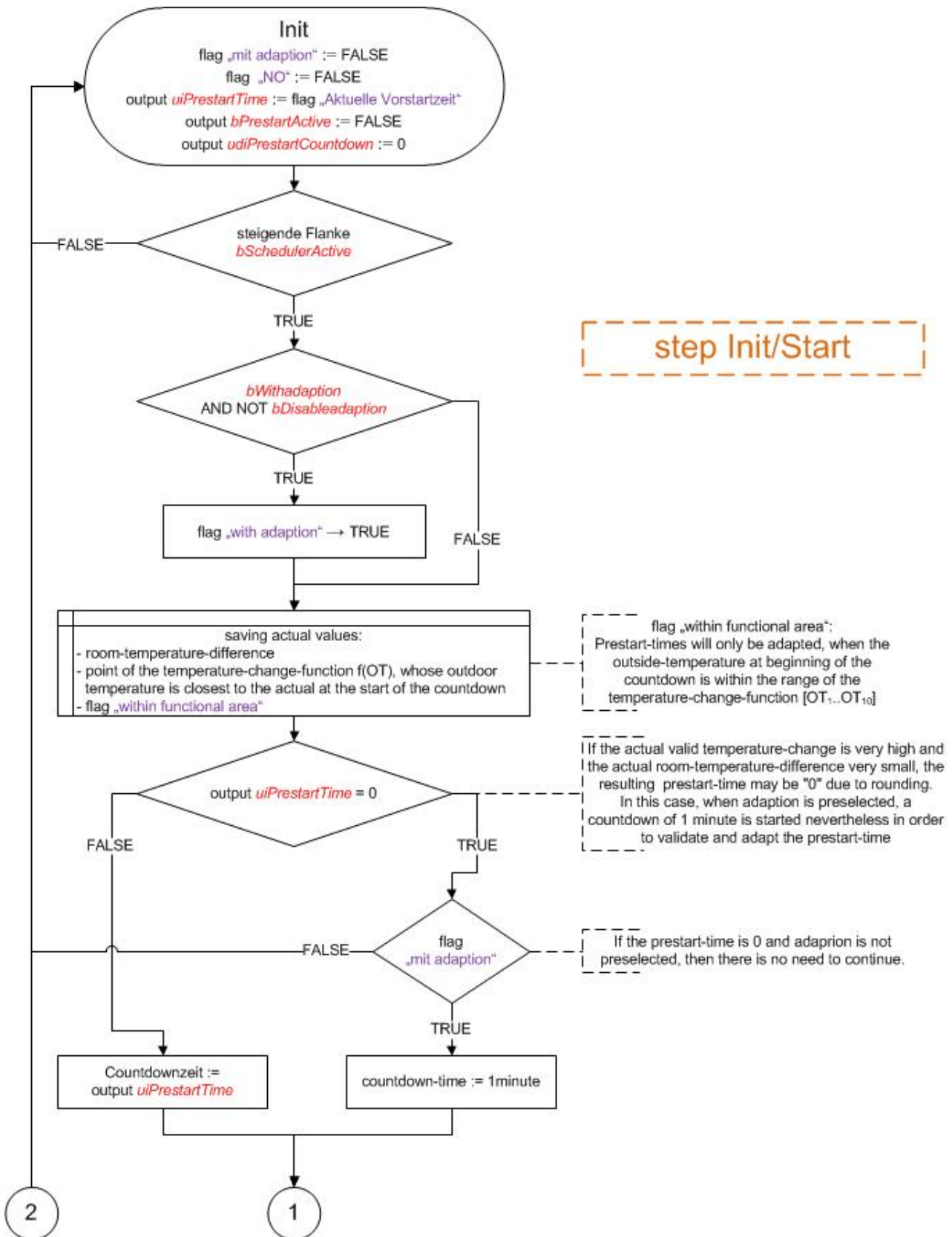


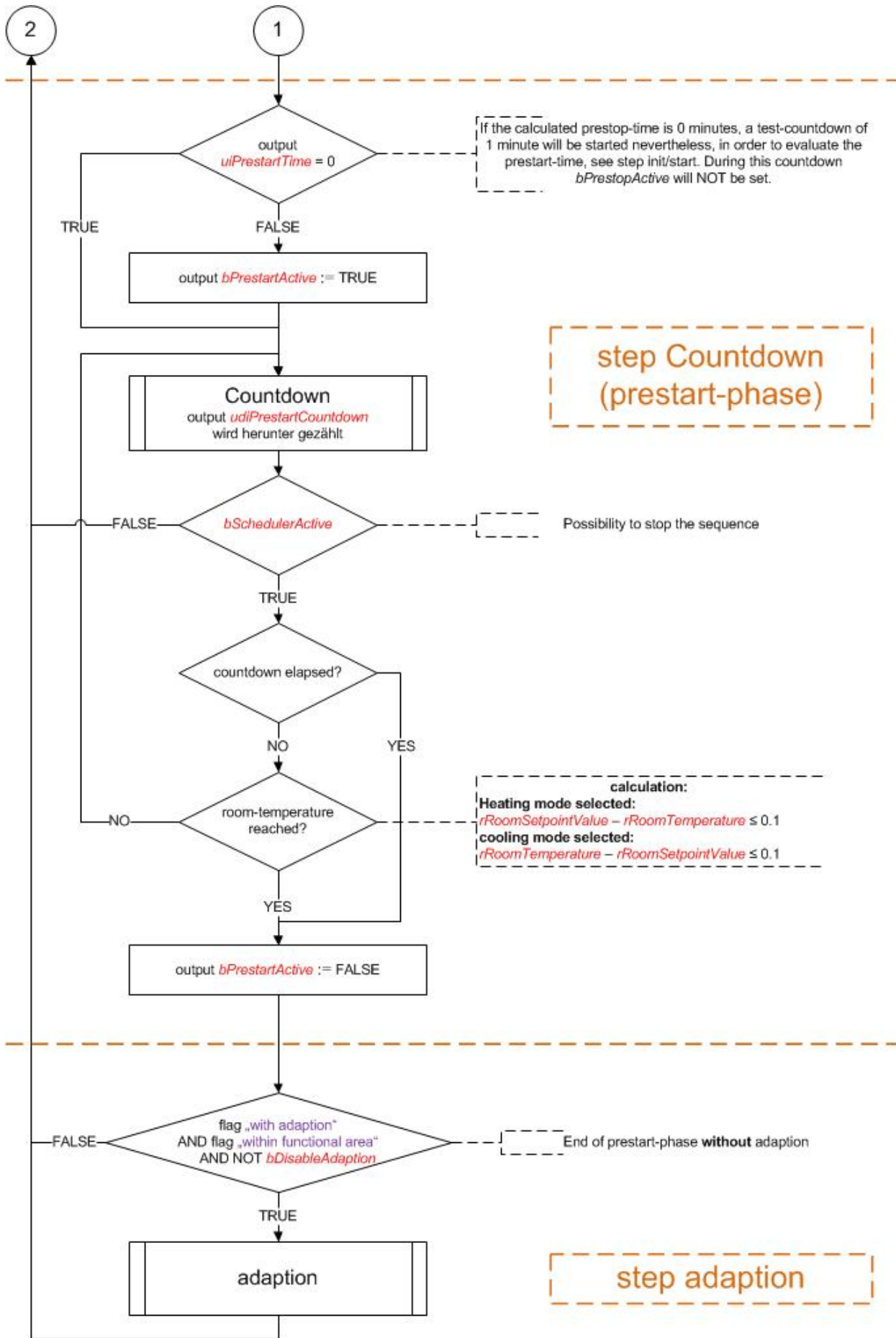
Temperature changes are only corrected if the outside temperature was within the interpolation points, i.e. the function range, at the beginning of the pre-start; see "[Adaptation \[▶ 438\]](#)".

If adaptation is desired, the inputs *bWithAdaption* and *bSchedulerActive* must be set simultaneously. A pure trigger pulse is sufficient for *bWithAdaption*. A previously started countdown is interrupted immediately if the input *bSchedulerActive* releases. The adaptation can be suppressed by the input *bDisableAdaption*. This option should be selected if the temperature change function is no longer to be changed after a certain number of adaptation procedures.

The following program flowchart is intended to better illustrate the behavior; the input and output variables of the function block are shown in red here:

The internal flag "Current pre-start time" is the continuously refreshed result of the calculation in accordance with the diagram shown above.





The input *bEnable*, the function of which is not shown here, effectively disables the function block if it is set to FALSE: "0" is output as the pre-start time *uiPrestartTime* and the sequence shown above is not started or is immediately reset.

Adaptation

Adaptation

If an adaptation is selected and the pre-start phase has ended (countdown = 0 or room temperature reached), then three cases can occur:

1. The countdown has elapsed **and** the room temperature difference lies below the tolerance limit value *rAdaptionTolerance* -> no adaptation takes place.
2. The countdown has not yet elapsed, but the set temperature for the room has been reached -> the pre-start time *uiPrestartTime* was too long.
3. The countdown has elapsed, but there is still a positive temperature difference -> the pre-start time *uiPrestartTime* was too short.

The value $\Delta T/\Delta t$ can now be re-determined on the basis of the elapsed countdown time and the change in the room temperature difference that has taken place

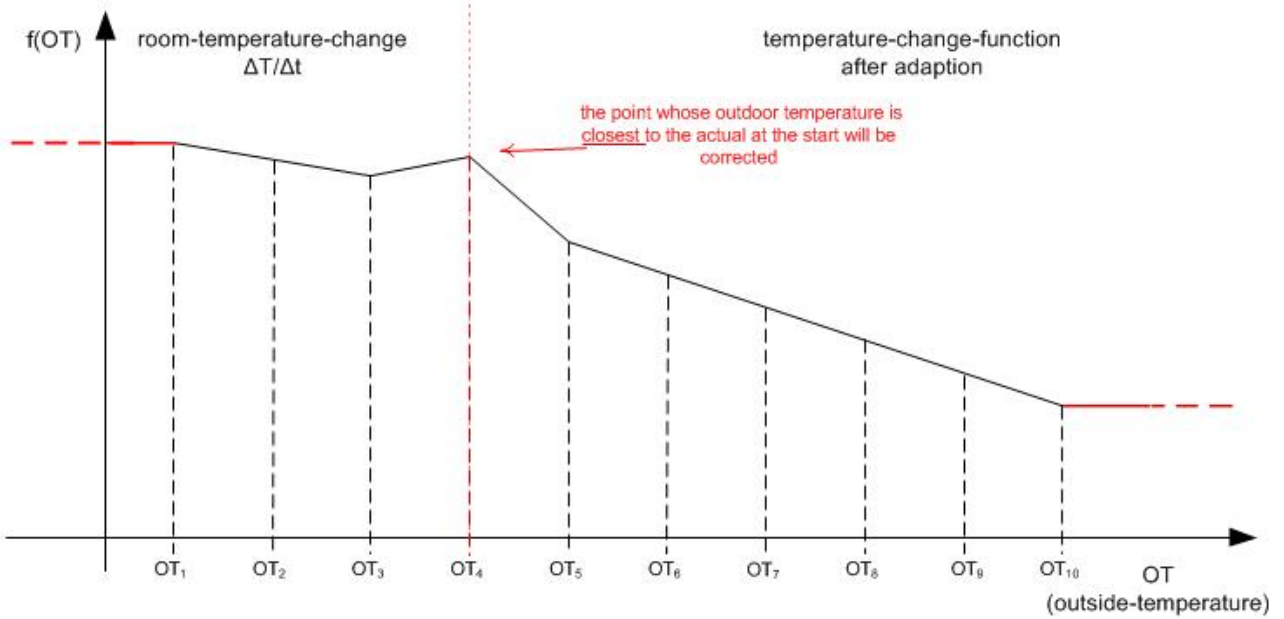
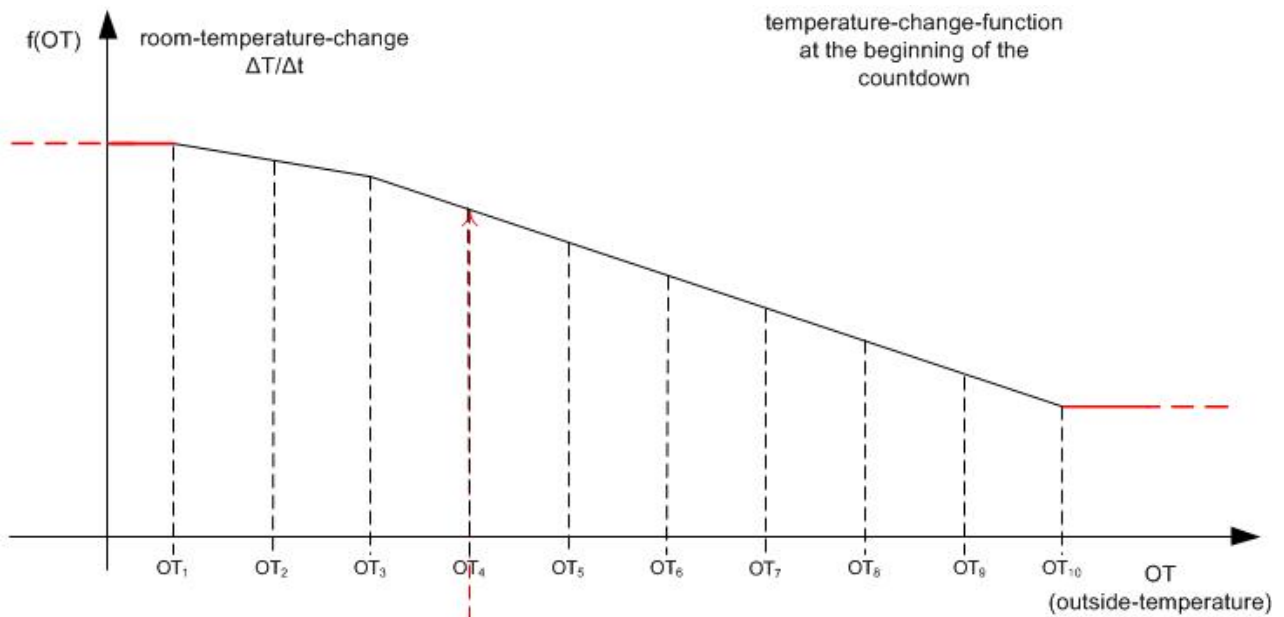
1. no change.
2. too much time was calculated - new value: "room temperature difference at the beginning" / "time elapsed so far"
3. too little time was calculated – new value: ("Room temperature difference at start" - "Room temperature difference at end") / "Countdown time"

The temperature change function is now corrected at the interpolation point whose outside temperature value was closest to that at the start of the pre-stop phase. The relevant point for this was saved before the start of the countdown, see program flow chart.

However, the previously calculated value $\Delta T/\Delta t$ is not necessarily adopted 100% as the new value at the interpolation point. In fact there is an option to mix the new value from a weighting of old and calculated value. This weighting takes place with the aid of the so-called adaptation factor *rAdaptionFactor*.

$$f(AT)_{NEW} := \frac{f(AT)_{OLD} \cdot (100 - rAdaptionFactor) + (\Delta T/\Delta t_{calculated} \cdot rAdaptionFactor)}{100}$$

With an adaptation factor of 100% the newly calculated value is adopted fully, while with 0% the old value is retained.



Because the pre-start time is always changed at the nearest point, an adaptation can only take place if the outside temperature at the beginning of the pre-start phase lies inside AT1 to AT10.


Input/output variables

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
eCtrlFct           : E_BARCtrlFct;
bEnable           : BOOL;
bSchedulerActive  : BOOL;
bWithAdaption     : BOOL;
bDisableAdaption  : BOOL;
rOutsideTemperature: REAL;
rRoomTemperature  : REAL;
rRoomSetpointValue: REAL;
```

eDataSecurityType: if `eDataSecurityType := eDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

eCtrlFct : `eCtrlFct = eBARCtrlFct_Heating` indicates to the function block that it is to be used in heating mode. Cooling mode is indicated by `eCtrlFct = eBARCtrlFct_Cooling`. Any other entry at this input is impermissible and leads to an error message. The two possible entries are for the different calculation of the temperature difference between actual room value and set room value.

bEnable : a FALSE signal at this input suppresses the optimized switch-on of the connected timers. The output value of the pre-start time is also set directly to "0". No countdown starts, nor is an adaptation carried out – the function block is reset.

bSchedulerActive : a rising edge at this input starts the internal countdown of the pre-start time. The output `bPrestartMode` is set to TRUE while the countdown of a pre-start time is running down. If the output `bSchedulerActive` is switched to FALSE again during the countdown, the countdown is interrupted immediately and the output `bPrestartMode` is set to FALSE. The countdown is also stopped and `bPrestartMode` set to FALSE when the room set value is reached.

bWithAdaption : if this input is set *simultaneously* with the input `bSchedulerActive`, an adaptation takes place after the expiry of the countdown. A trigger pulse together with the rising edge of `bSchedulerActive` is sufficient for this. This input can only be used in conjunction with `bSchedulerActive`; setting it alone has no effect at all.

bDisableAdaption : a TRUE signal at this input merely suppresses the adaptation that follows the countdown.

rOutsideTemperature : outside temperature in degrees Celsius.

rRoomTemperature : room temperature in degrees Celsius.

rRoomSetpointValue : room temperature setpoint in degrees Celsius.

VAR_OUTPUT

```
uiPrestartTime   : UINT;
bPrestartActive  : BOOL;
udiCountdownTime : UDINT;
bError           : BOOL;
udiErrorID       : UDINT;
```

uiPrestartTime : output value of the optimized pre-start time to the timers concerned in minutes. This value is formed continuously from the pre-start function that is dependent on the outside temperature. Conversely, if the input `bDisableOptimization` is set to TRUE, then this output is set to "0".

bPrestartActive : the function block is in pre-start mode as long as the internal countdown is running and has not been ended by the expiry of the pre-start time, by the reaching of the room temperature setpoint or by an interruption (`bSchedulerActive = FALSE`). This is indicated by a TRUE signal at this output.


udiCountdownTime : this output indicates the elapsing of the internal countdown in seconds. This output is set to "0" if the function block is no longer in pre-start mode (see `bPrestartMode`).

bError: this output is switched to TRUE if the parameters entered are erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes](#) [▶ 237].

VAR_IN_OUT

The need for entered parameters to be preserved across a control failure makes it necessary for them to be declared as IN-OUT variables. A reference variable is then assigned to them in the program. Each change in the value of this reference variable is persistently saved in the function block and written back to the reference variable after a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>.

```
uiMaxPrestartTime    : UINT;
rAdaptionFactor      : REAL;
rAdaptionTolerance   : REAL;
stTempChangeFunction: ST_HVACTempChangeFunction;
```

uiMaxPrestartTime : due to the adaptation the pre-start times are shortened and lengthened within the temperature change function. While they are logically automatically limited to 0 minutes in the downward direction, the upward limit in minutes can be defined by this input.

rAdaptionFactor : in the adaptation step the temperature change $\Delta T/\Delta t$ that has taken place is calculated for the nearest outside temperature interpolation point. However, this is not necessarily adopted 100% as the new value. In fact there is an option to mix the new value from a weighting of old and calculated value. The adaptation factor (in percent) thereby represents the weighting.

$$f(A\ T)_{NEW} := \frac{f(A\ T)_{OLD} \cdot (100 - rAdaptionFactor) + (\Delta T/\Delta t_{calculated} \cdot rAdaptionFactor)}{100}$$

With an adaptation factor of 100% the newly calculated value is adopted fully, while with 0% the old value is retained.

rAdaptionTolerance : if the countdown has expired and the room temperature setpoint has been reached with adaptation activated, no adaptation is carried out because the pre-start time is precisely right. The value *rAdaptionTolerance* defines a tolerance range: if the actual value lies within the range *rRoomSetpointValue* .. *rRoomSetpointValue* + *rAdaptionTolerance*, then this is regarded as reaching the setpoint.

stTempChangeFunction : structure variable of type ST_HVACTempChangeFunction [▶ 529], which contains the 10 value pairs (outside temperature, internal temperature change). These value pairs, which have to be entered in the field variable in ascending order of the outside temperature, define the 9 pitch lines of the temperature change function. Initial entry through [FB_HVACTempChangeFunctionEntry](#) [▶ 452].

3.6.27 FB_HVACOptimizedOff

Function block for the optimized switch-off (pre-stop) of heating boilers and air conditioning systems in conjunction with switching time function blocks.

The explanations in this document relate to the heating behavior. The function block is used in the same way in conjunction with cooling devices.

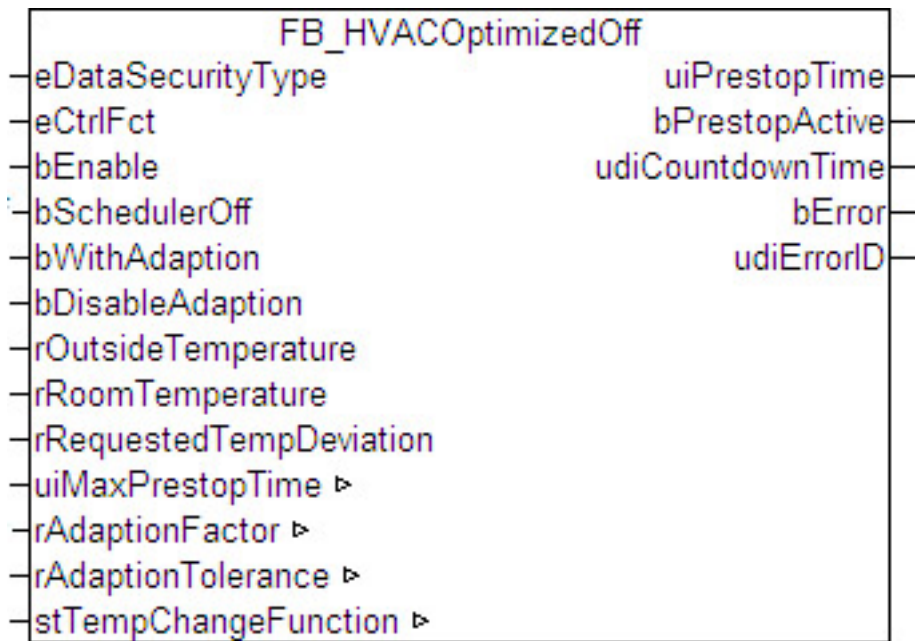
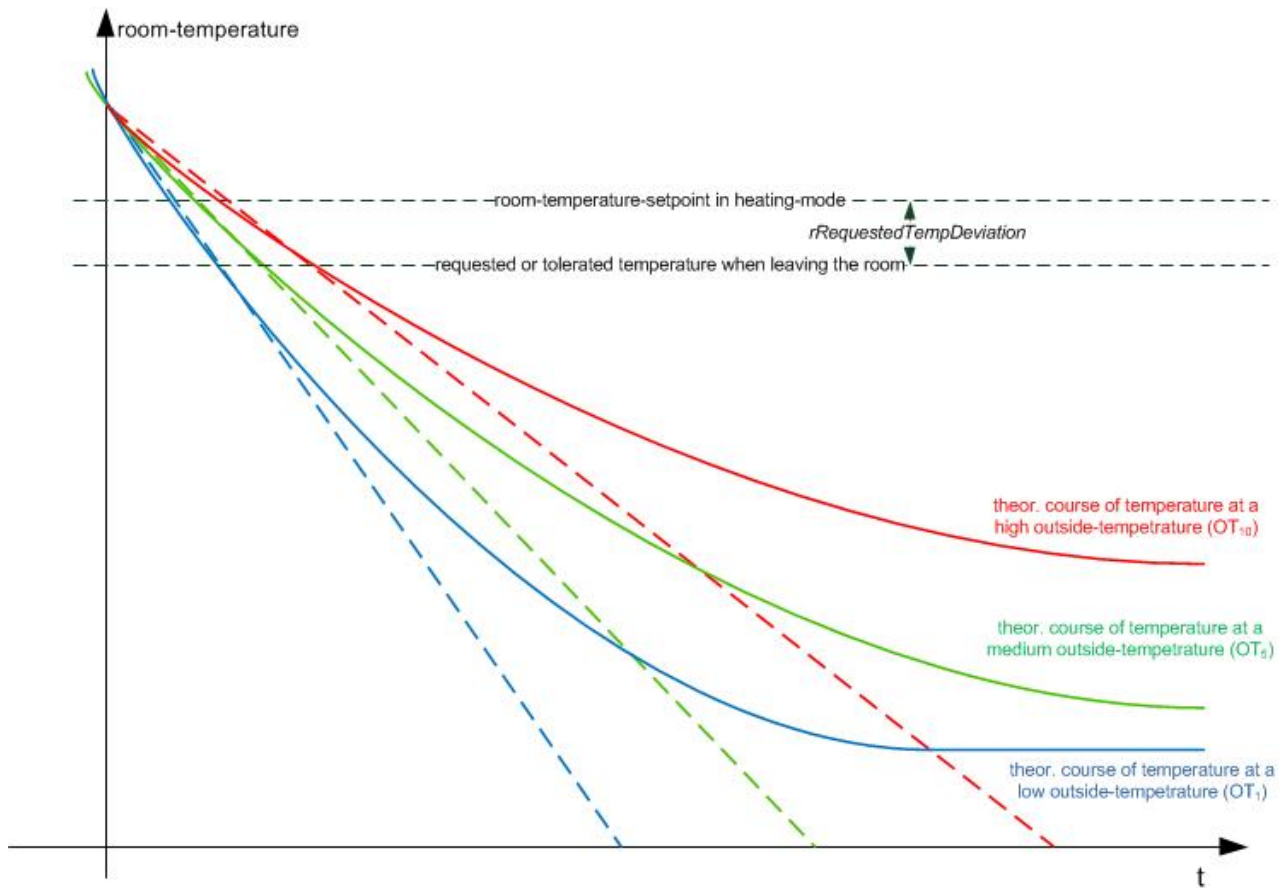


Fig. 18: FB_HVACOptimizedOff

Calculation of the pre-stop time

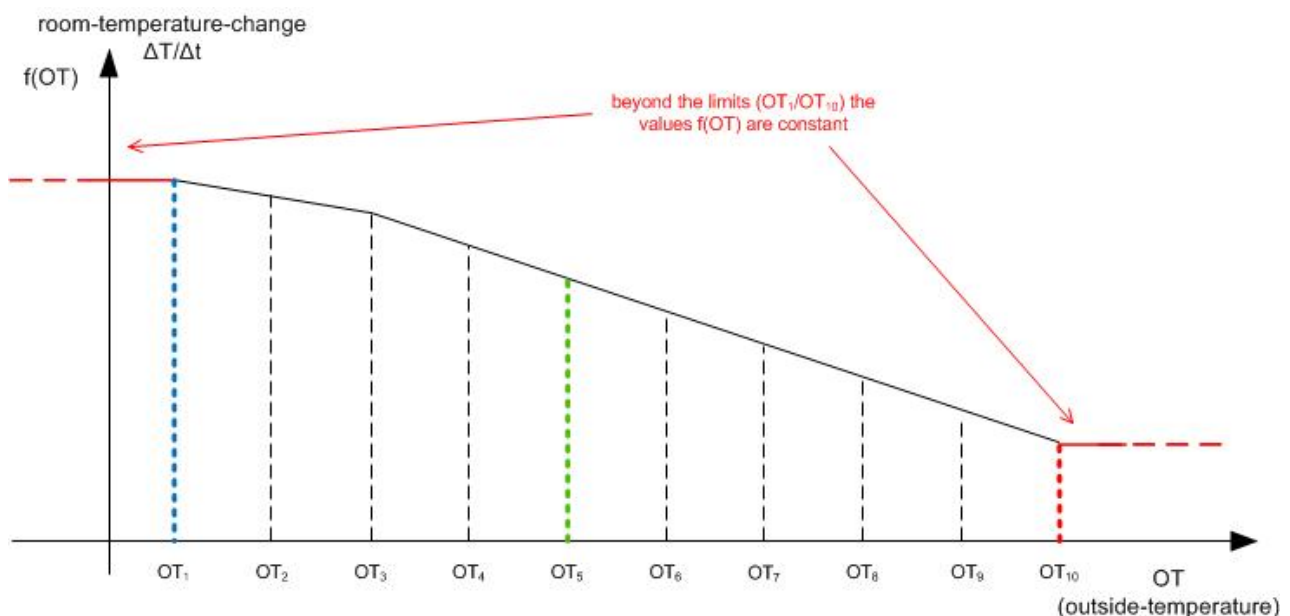
Buildings and parts of buildings that are occupied at a specific and predictable time, such as schools or conference rooms, do not need to be heated for the entire occupancy period. Due to the fact that the buildings store heat, it is possible to turn off the heating a little earlier. The room temperature then drops to a barely noticeable lower temperature level by the end of occupancy. This is a deliberate drop to a target temperature - the drop difference is an input parameter of the function block and is called *rRequestedTempDeviation*.

The flow temperature of the heating boiler depends on the outside temperature. Thus, different outside temperatures result in different cooling curves, which essentially correspond to an exponential function:



To determine the pre-stop time it is assumed that the area of the functions between the room temperature start value and room temperature setpoint is linear. This results in a characteristic temperature change $\Delta T/\Delta t$ for each outside temperature, which is shown here by the dotted line.

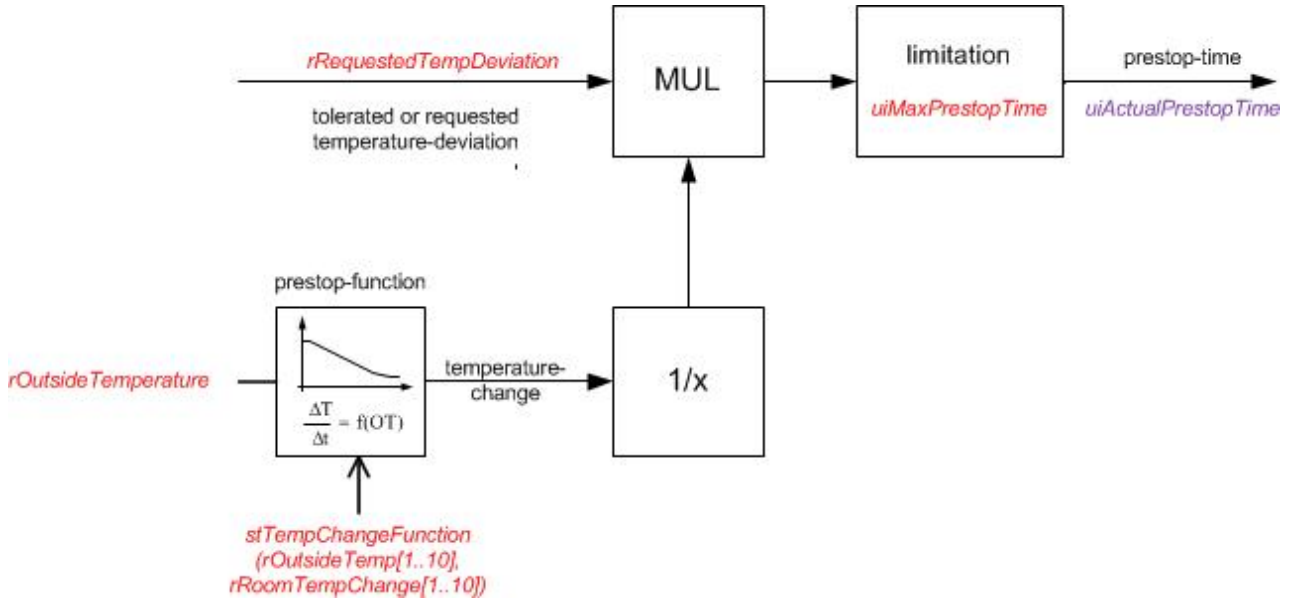
With the structure variable `stTempChangeFunction` [▶ 529], the function block `FB_HVACOptimizedOff` is based on a table in which the respectively expected room temperature change is assigned for 10 discrete outside temperatures. During the commissioning or the initial starting of the system, the pre-start function must be coarsely predefined - firstly to define the outside temperature range and secondly to accelerate the adaptation procedure. This entry is done with the function block `FB_HVACTempChangeFunctionEntry` [▶ 452]. The pre-stop time can be determined approximately with these values. This temperature change function $f(AT)$ typically takes the following course, where the change is shown as magnitude, i.e. positive:



Function values within these 10 points are defined by linear equations, while values outside correspond to the function value $f(AT_1)$ or $f(AT_{10})$ respectively:

- $AT < AT_1$: $f(AT) = f(AT_1)$
- $AT > AT_{10}$: $f(AT) = f(AT_{10})$

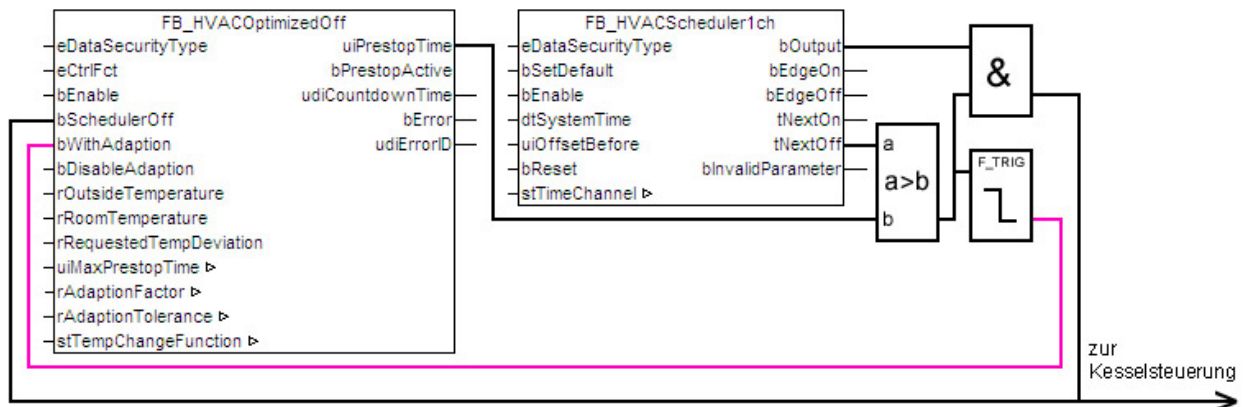
The pre-stop time is then determined in accordance with the following sequence:



The input and output variables of the function block are shown here in red. Shown in purple here is the internal flag *uiActualPrestopTime* which contains the current pre-stop time at each point in time. The output *uiPrestopTime* corresponds to this flag, but is "frozen" during the pre-stop phase. This is explained in more detail below.

Function block linking and pre-stop phase

From these parameters the function block will then continuously calculate a pre-stop time, which it specifies to the timer function block. Then if, for example, a heating boiler is to be switched off at 8 pm and the pre-stop function block determines at 7 pm that the temperature level would drop by the desired value *rRequestedTempDeviation* on switching off the boiler in 60 minutes, the boiler is switched off immediately. The cooling behavior is then observed in a reference room over the pre-stop phase and the temperature change curve is corrected accordingly.



The pre-stop function block specifies a pre-stop time for the timer. If the timer switches the output *bOutput* back to FALSE, this is indicated to the pre-stop function block via its input *bSchedulerOff*. Internally the pre-stop function block then starts a countdown with the previously output pre-stop time *uiPrestopTime* [min].

The countdown either runs to the end or is prematurely ended on reaching the requested or tolerated room temperature. During the countdown, the *bPrestopActive* output is set.

The *uiPrestopTime* output continuously follows the above calculation - however, during a started countdown it is kept constant so that a fluctuation in the outdoor temperature does not indicate a lower pre-stop time and suddenly turn the boiler back on.

If, in the case of a pre-stop, the timer function block still sends the adaptation order as an edge (red line) to the optimization function block, then the latter will decide following the pre-stop phase whether the previously determined pre-stop time was precisely within a tolerance, or too short or too long, and correct the temperature change function accordingly. In doing so the pre-stop time of the point whose outside temperature was nearest to the actual one at the beginning of the countdown is corrected upwards or downwards respectively. This process is called "[adaptation \[▶ 448\]](#)".

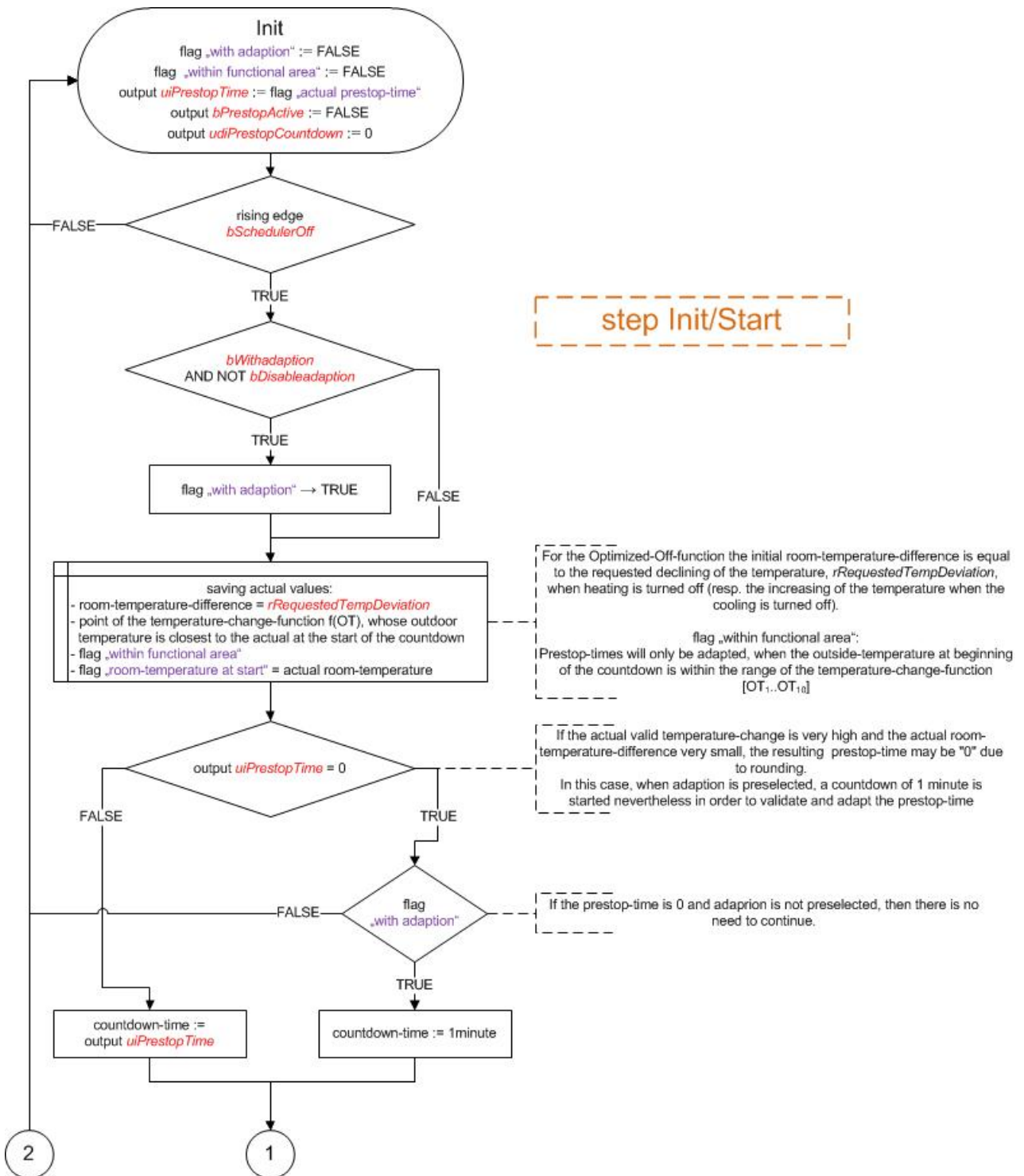


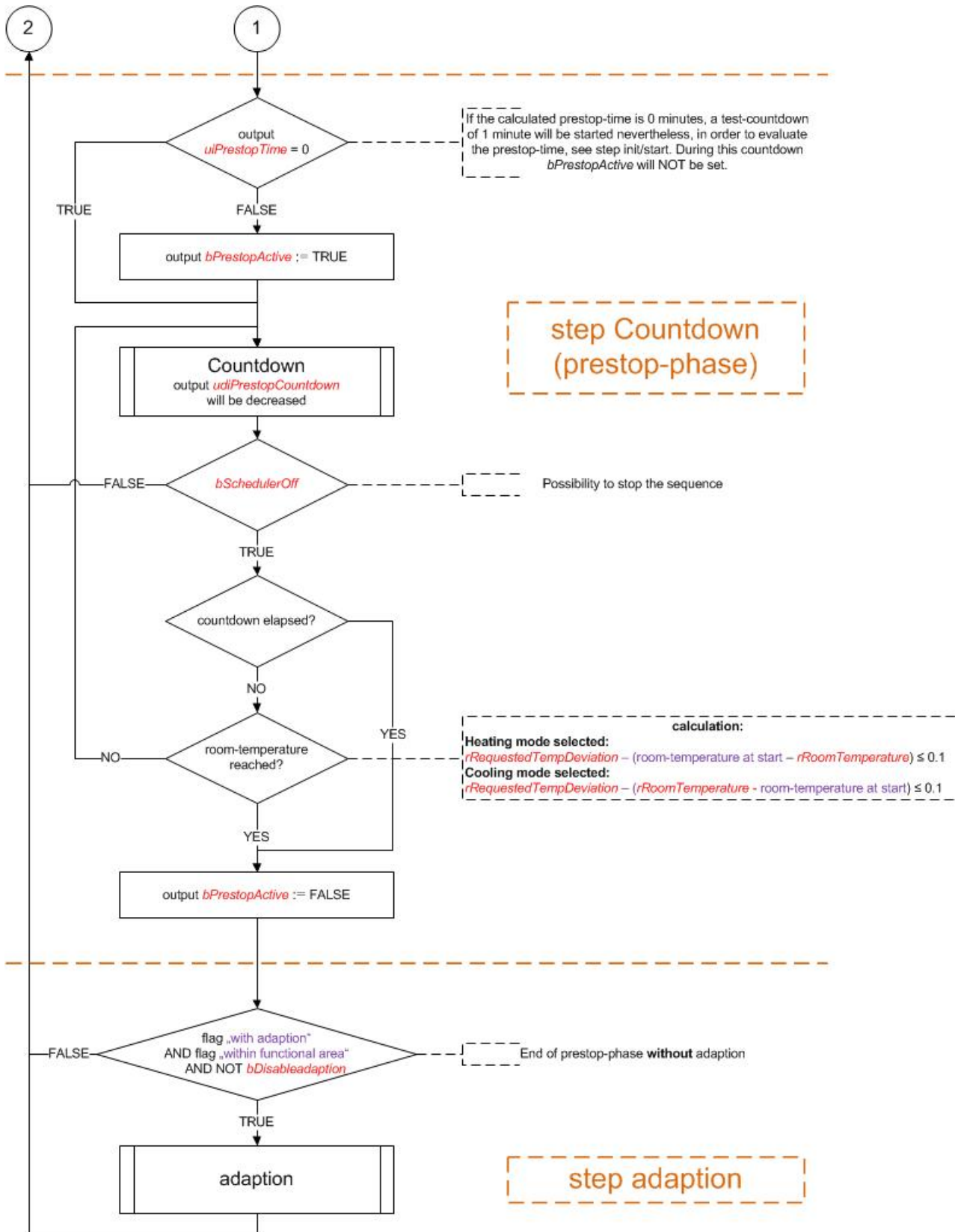
Temperature changes are only corrected if the outside temperature was within the interpolation points, i.e. the function range, at the beginning of the pre-stop; see "[Adaptation \[▶ 448\]](#)".

If adaptation is desired, the inputs *bWithAdaption* and *bSchedulerOff* must be set simultaneously. A pure trigger pulse is sufficient for *bWithAdaption*. A previously started countdown is interrupted immediately if the input *bSchedulerOff* releases. The adaptation can be suppressed by the input *bDisableAdaption*. This option should be selected if the temperature change function is no longer to be changed after a certain number of adaptation procedures.

The following program flowchart is intended to better illustrate the behavior; the input and output variables of the function block are shown in red here:

The internal flag "Current pre-stop time" is the continuously refreshed result of the calculation in accordance with the diagram shown above.





The input *bEnable*, the function of which is not shown here, effectively disables the function block if it is set to FALSE: "0" is output as the pre-stop time *uiPrestopTime* and the sequence shown above is not started or is immediately reset.

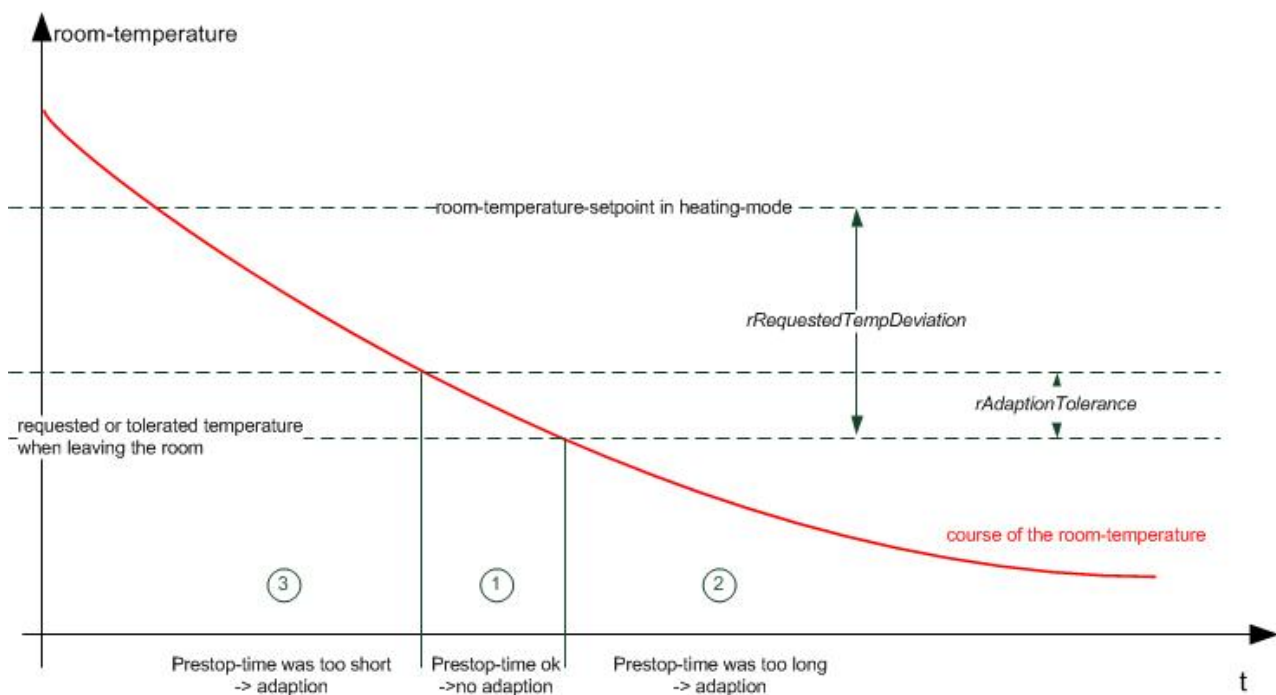
Adaptation

Adaptation

The goal of the adaptation is to adapt the temperature change function so precisely that the requested or tolerated temperature decrease $rRequestedTempDeviation$ is reached in the determined pre-stop time (in cooling mode it would be the temperature increase after switching off the cooling devices).

If an adaptation is selected and the pre-stop phase has ended (countdown = 0 or room temperature reached), then three cases can occur:

1. The countdown has elapsed **and** the requested deviation $rRequestedDeviation$ is exhausted up to a tolerance $rAdaptionTolerance$ -> no adaptation takes place.
2. The countdown has not yet elapsed, but the requested deviation $rRequestedDeviation$ has been exceeded -> the pre-stop time $uiPrestopTime$ was too long.
3. The countdown has elapsed, but the requested deviation $rRequestedDeviation$ is not yet fully exhausted -> the pre-stop time $uiPrestopTime$ was too short, i.e. switch-off can take place even earlier.



The value $\Delta T/\Delta t$ can now be re-determined on the basis of the elapsed countdown time and the change in the room temperature difference that has taken place

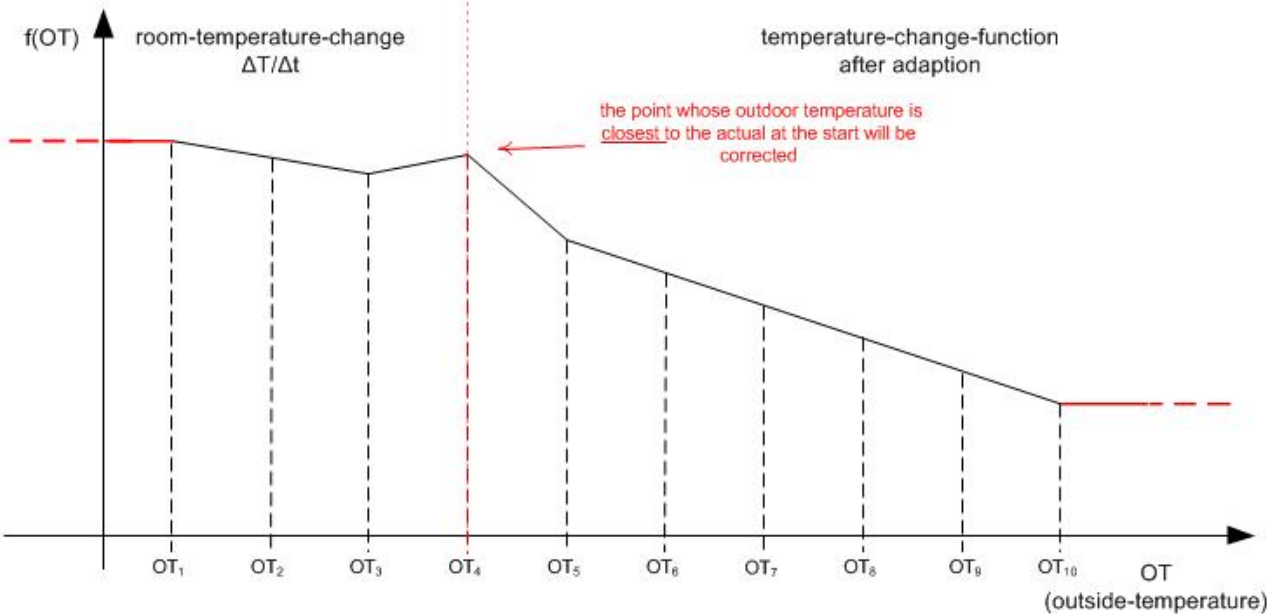
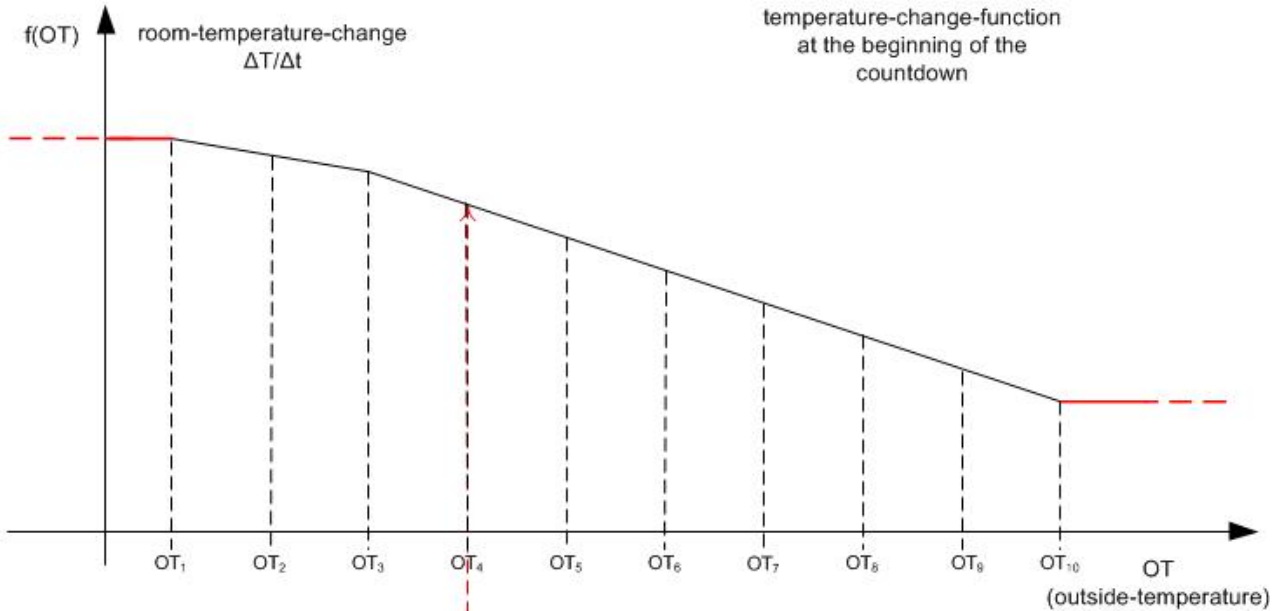
1. no change.
2. too much time was calculated - new value: "desired deviation $rRequestedDeviation$ " / "time elapsed so far"
3. too much time was calculated - new value: ("requested deviation $rRequestedDeviation$ " - "Deviation at the end") / "Countdown time"

The temperature change function is now corrected at the interpolation point whose outside temperature value was closest to that at the start of the pre-stop phase. The relevant point for this was saved before the start of the countdown, see program flow chart.

However, the previously calculated value $\Delta T/\Delta t$ is not necessarily adopted 100% as the new value at the interpolation point. In fact there is an option to mix the new value from a weighting of old and calculated value. This weighting takes place with the aid of the so-called adaptation factor $rAdaptionFactor$.

$$f(AT)_{NEW} := \frac{f(AT)_{OLD} \cdot (100 - rAdaptionFactor) + (\Delta T / \Delta t_{calculated} \cdot rAdaptionFactor)}{100}$$

With an adaptation factor of 100% the newly calculated value is adopted fully, while with 0% the old value is retained.



Note: Because the pre-stop time is always changed at the nearest point, an adaptation can only take place if the outside temperature at the beginning of the pre-stop phase lies *inside* AT_1 to AT_{10} .


Input/output variables

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
eCtrlFct          : E_BARCtrlFct;
bEnable          : BOOL;
bSchedulerOff    : BOOL;
bWithAdaption    : BOOL;
bDisableAdaption : BOOL;
rOutsideTemperature : REAL;
rRoomTemperature : REAL;
rRequestedTempDeviation : REAL;
```

eDataSecurityType: if `eDataSecurityType := eDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

eCtrlFct : `eCtrlFct = eBARCtrlFct_Heating` indicates to the function block that it is to be used in heating mode. Cooling mode is indicated by `eCtrlFct = eBARCtrlFct_Cooling`. Any other entry at this input is impermissible and leads to an error message. The two possible entries are for the different evaluation of the deviation `rRequestedTempDeviation`.

bEnable : a FALSE signal at this input suppresses the optimized switch-off of the connected timers. The output value of the pre-stop time is also set directly to "0". No countdown starts, nor is an adaptation carried out – the function block is reset.

bSchedulerOff : a rising edge at this input starts the internal countdown of the pre-stop time. The output `bPrestopMode` is set to TRUE while the countdown of a pre-stop time is running down. If the output `bSchedulerOff` is switched to FALSE again during the countdown, the countdown is interrupted immediately and the output `bPrestopMode` is set to FALSE. The countdown is also stopped and `bPrestopMode` set to FALSE when the room set value is reached.

bWithAdaption : if this input is set *simultaneously* with the input `bSchedulerOff`, an adaptation takes place after the expiry of the countdown. A trigger pulse together with the rising edge of `bSchedulerOff` is sufficient for this. This input can only be used in conjunction with `bSchedulerOff`, setting it alone has no effect at all.

bDisableAdaption : a TRUE signal at this input merely suppresses the adaptation that follows the countdown.

rOutsideTemperature : outside temperature in degrees Celsius.

rRoomTemperature : room temperature in degrees Celsius.

rRequestedTempDeviation : tolerated downward (heating mode) or upward (cooling mode) temperature deviation after switching off the boiler or the air conditioning system respectively until leaving the room.

VAR_OUTPUT

```

uiPrestopTime   : UINT;
bPrestopActive  : BOOL;
udiCountdownTime : UDINT;
bError          : BOOL;
udiErrorID      : UDINT;

```

uiPrestopTime : output value of the optimized pre-stop time to the timers concerned in minutes. This value is formed continuously from the pre-stop function that is dependent on the outside temperature. Conversely, if the input *bDisableOptimization* is set to TRUE, then this output is set to "0".

bPrestopActive : the function block is in pre-stop mode as long as the internal countdown is running and has not been ended by the expiry of the pre-stop time, by the reaching of the room temperature setpoint or by an interruption (*bSchedulerOff* = FALSE). This is indicated by a TRUE signal at this output.


udiCountdownTime : this output indicates the elapsing of the internal countdown in seconds. This output is set to "0" if the function block is no longer in pre-stop mode (see *bPrestopMode*).

bError: this output is switched to TRUE if the parameters entered are erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes](#) [▶ 237].

VAR_IN_OUT

The need for entered parameters to be preserved across a control failure makes it necessary for them to be declared as IN-OUT variables. A reference variable is then assigned to them in the program. Each change in the value of this reference variable is persistently saved in the function block and written back to the reference variable after a controller failure and restart. If the parameters were only declared as input variables, they would **not** be able to write a reference variable.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

```

uiMaxPrestopTime : UINT;
rAdaptionFactor   : REAL;
rAdaptionTolerance : REAL;
stTempChangeFunction : ST_HVACTempChangeFunction;

```

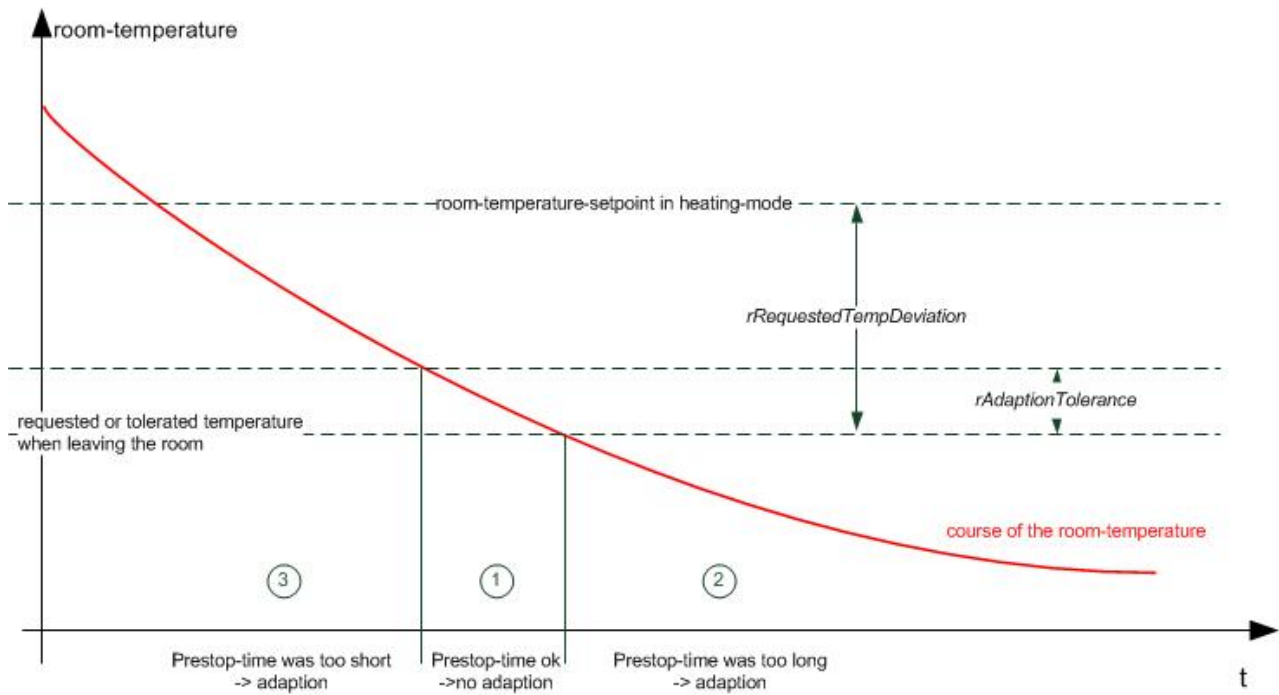
uiMaxPrestopTime : due to the adaptation the pre-stop times are shortened and lengthened within the temperature change function. While they are logically automatically limited to 0 minutes in the downward direction, the upward limit in minutes can be defined by this input.

rAdaptionFactor : in the adaptation step the temperature change $\Delta T/\Delta t$ that has taken place is calculated for the nearest outside temperature interpolation point. However, this is not necessarily adopted 100% as the new value. In fact there is an option to mix the new value from a weighting of old and calculated value. The adaptation factor (in percent) thereby represents the weighting.

$$f(AT)_{\text{NEW}} := \frac{f(AT)_{\text{OLD}} \cdot (100 - rAdaptionFactor) + (\Delta T/\Delta t_{\text{calculated}} \cdot rAdaptionFactor)}{100}$$

With an adaptation factor of 100% the newly calculated value is adopted fully, while with 0% the old value is retained.

rAdaptionTolerance : if the countdown has elapsed and the requested deviation *rRequestedDeviation* is exhausted, no adaptation is carried out because the pre-stop time is precisely right. The value *rAdaptionTolerance* defines an additional tolerance range:



stTempChangeFunction : structure variable of type [ST_HVACTempChangeFunction](#) [▶ 529], which contains the 10 value pairs (outside temperature, internal temperature change). These value pairs, which have to be entered in the field variable in ascending order of the outside temperature, define the 9 pitch lines of the temperature change function. Initial entry through [FB_HVACTempChangeFunctionEntry](#) [▶ 452].

3.6.28 FB_HVACTempChangeFunction

Function block for the input of the interpolation points of the pre-start function.

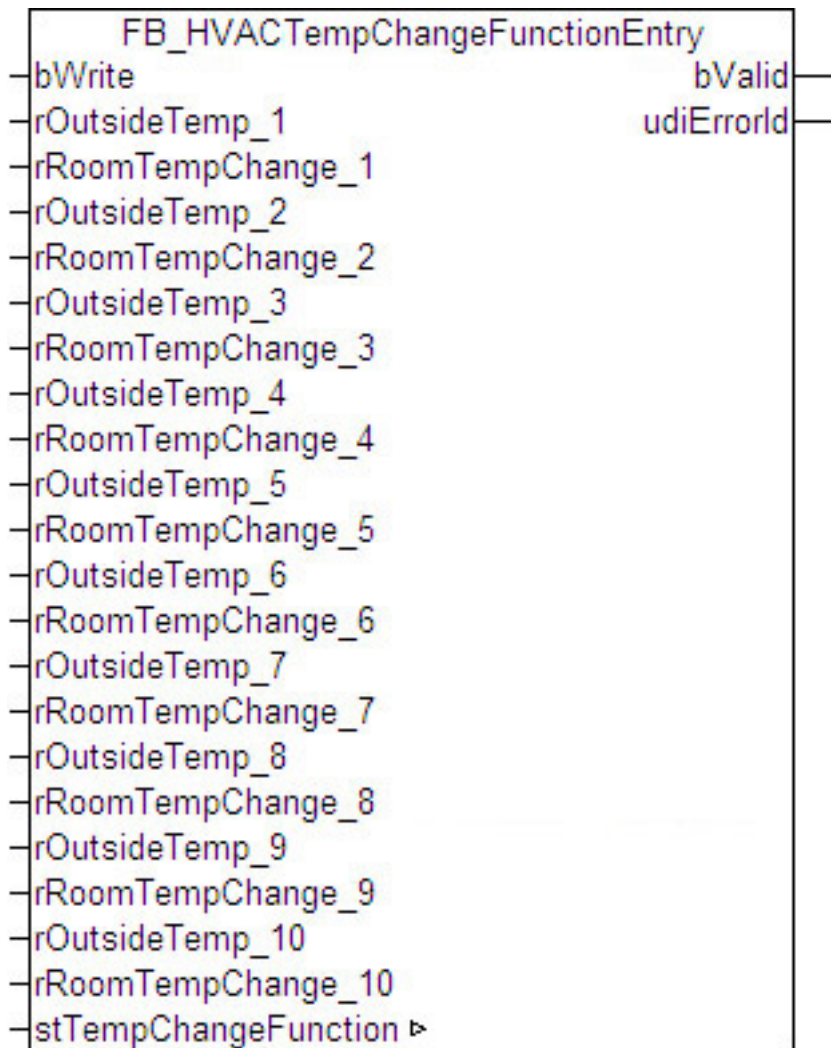


Fig. 19: FB_HVACTempChangeFunctionEntry

To keep the function blocks [FB_HVACOptimizedOn](#) [▶ 431] and [FB_HVACOptimizedOff](#) [▶ 441] uncluttered, they do not contain any input of the individual value pairs for the temperature change function - the function blocks access the structure variable of the temperature change function ([ST_HVACTempChangeFunction](#) [▶ 529]) via an IN-OUT variable. The function block [FB_HVACTempChangeFunction](#) enables the writing of the structure variable in a clear form and also makes sure that the value pairs, as required, are entered in ascending order of the outside temperature and that there are no two points with the same outside temperature. Mathematically speaking, there would be no unambiguous functional correlation in this case. The value pairs are to be entered at the corresponding inputs *rOutsideTemperature_1..rOutsideTemperature_10* (outside temperature) and *rRoomTempChange_1..rRoomTempChange_10* (room temperature change). The function block continuously checks whether the described requirement for ascending order of the outside temperature is fulfilled and whether two value pairs exist with the same outside temperature.



The writing of the temperature change functions should take place **once** in order to give the pre-start/pre-stop function blocks basic values, which are then continually improved by these function blocks over the course of time.

VAR_INPUT

```

bWrite          : BOOL;
rOutsideTemp_1  : REAL;
rRoomTempChange_1 : REAL;
rOutsideTemp_2  : REAL;
rRoomTempChange_2 : REAL;
rOutsideTemp_3  : REAL;
rRoomTempChange_3 : REAL;
rOutsideTemp_4  : REAL;
rRoomTempChange_4 : REAL;

```

```

rOutsideTemp_5      : REAL;
rRoomTempChange_5  : REAL;
rOutsideTemp_6      : REAL;
rRoomTempChange_6  : REAL;
rOutsideTemp_7      : REAL;
rRoomTempChange_7  : REAL;
rOutsideTemp_8      : REAL;
rRoomTempChange_8  : REAL;
rOutsideTemp_9      : REAL;
rRoomTempChange_9  : REAL;
rOutsideTemp_10     : REAL;
rRoomTempChange_10 : REAL;

```

bWrite : a rising edge at this input copies the values entered at the inputs to the pre-start function.

(rOutsideTemp_1 - rRoomTempChange1) ... (rOutsideTemp_10 - rRoomTempChange10): value pairs of the pre-start function: room temperature change (*rRoomTempChange*) in degrees Kelvin per minute at outside temperature (*rOutsideTemp*) in degrees Celsius.

VAR_OUTPUT

```

bValid      : BOOL;
udiErrorID  : UDINT;

```

bValid : this output is switched to TRUE if the parameters entered are **not** erroneous.

udiErrorId : contains the error code if the values entered should be erroneous. See [error codes \[▶ 237\]](#).

VAR_IN_OUT

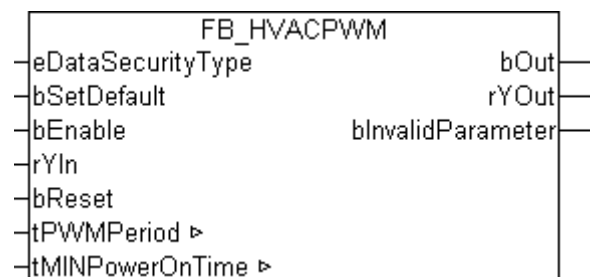
```

stTempChangeFunction : ST_HVACTempChangeFunction;

```

stTempChangeFunction : structure variable of type [ST_HVACTempChangeFunction \[▶ 529\]](#), which contains the 10 value pairs (outside temperature, room temperature change). These value pairs, which have to be entered in the field variable in ascending order of the outside temperature, define the 9 pitch lines of the pre-start function.

3.6.29 FB_HVACPWM



Application

This function block generates a pulse width modulated signal from the analog input signal *rYIn*. Furthermore, a minimum switch-on time can be parameterized.

VAR_INPUT

```

eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
bEnable          : BOOL;
rYIn             : REAL;           0 .. 100    %
bReset           : BOOL;

```

eDataSecurityType: if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block *FB_HVACPersistentDataHandling* must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType := eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled if `bEnable = TRUE`.

rYIn: analog input value of the function block.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
bOut          : BOOL;
rYOut         : REAL;    0 .. 100    %
bInvalidParameter: BOOL;
```

bOut: PWM signal.

rYOut: output of the input value of the function block.

bInvalidParameter: Indicates that an incorrect input parameter is present. `bInvalidParameter` must be acknowledged with `bReset`.

VAR_IN_OUT

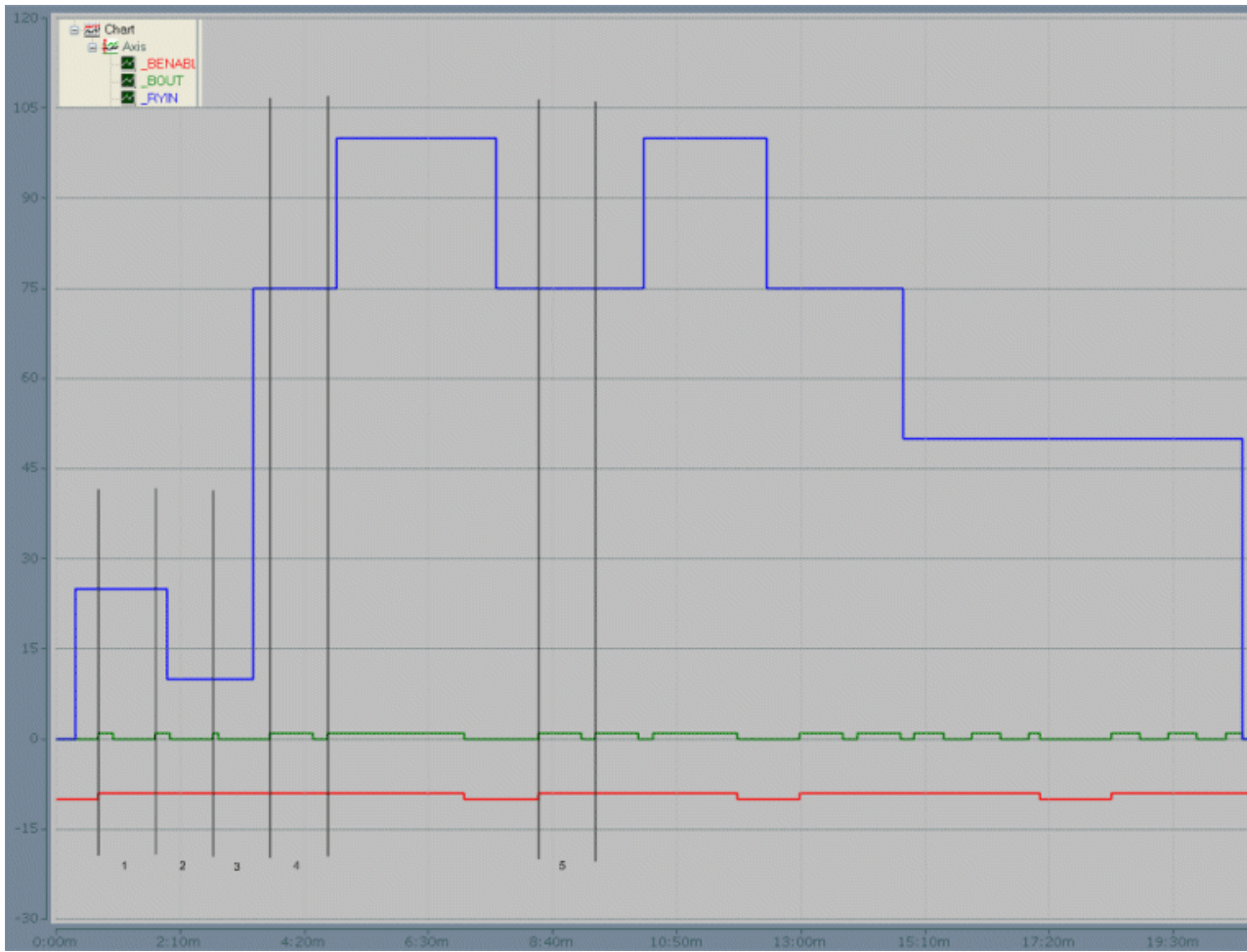
```
tPWMPeriod    : TIME;
tMINPowerOnTime : TIME;
```

tPWMPeriod: period of the PWM signal. The variable is saved persistently. Preset to 30 min.

tMINPowerOnTime: minimum switch-on time of the pulsed output `bOut`. The variable is saved persistently. Preset to 0 s.

Fig. 1: Scope2 recording for additional explanation of the mode of operation of the function block. The output (`bOut = TRUE`) is set on enabling the function block (`bEnable = TRUE`). See also sections 1 and 5.

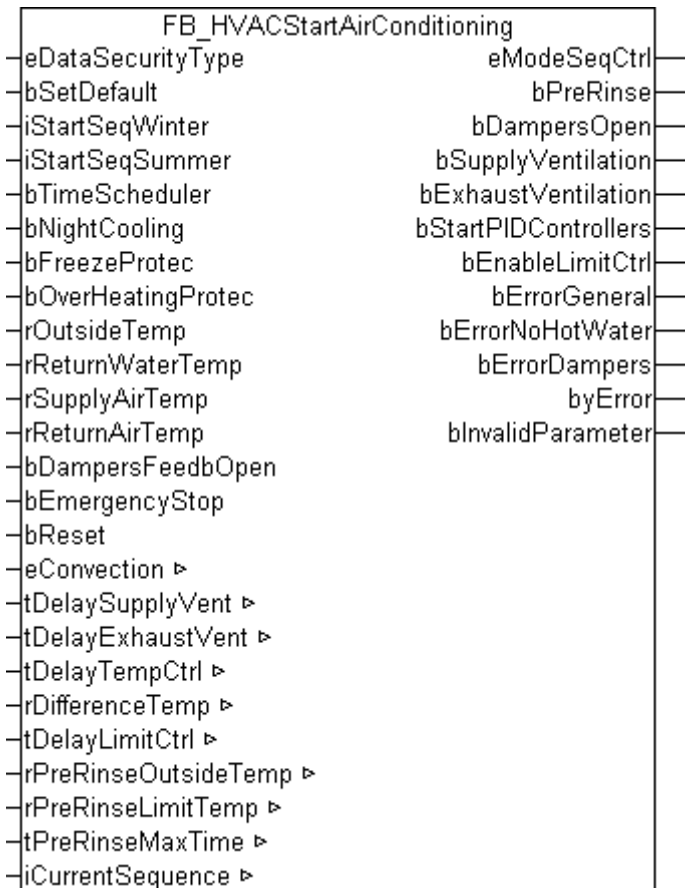
A change in the input value during a running period is only taken into account in the next period. See also sections 2, 3 and 4.



Documents about this

 [example_persistent_e.zip \(Resources/zip/11659714827.zip\)](#)

3.6.30 FB_HVACStartAirConditioning



Application

The air conditioning system is started up stepwise with the start program *FB_HVACStartAirConditioning*. The shut-off dampers, the fans, the controller and the limit value monitoring of the analog inputs are thereby enabled one after the other.

In the case of low outside temperatures below the value of *rPreRinseOutsideTemp*, the heating coil is initially rinsed through with hot water before the outside and exhaust air dampers are opened. In VAC-systems with a temperature sensor in the return flow from the heating coil, the rinsing procedure continues until the temperature *rReturnWaterTemp* has exceeded the value of the input variable *rPreRinseLimitTemp*. The outside and exhaust air dampers are opened when the specified temperature in the return flow from the air heater is reached. If the temperature is not exceeded after the timer *tPreRinseMaxTime* has elapsed, the fault is indicated by a TRUE at the output *bErrorNoHotWater*.

If there is no temperature sensor in the return flow from the heater, a constant of **-100** is applied to the input *rReturnWaterTemp*. The function block **FB_HVACStartAirConditioning** then knows that the pre-rinse procedure must be executed without return flow monitoring. In this case, rinsing is performed until the timer *tPreRinseMaxTime* expires. **Hence, there is no monitoring of the rinsing procedure!**

The outside and exhaust air dampers are opened after completion of the pre-rinsing procedure. The output *bDampersOpen* becomes TRUE as a result.

In order to avoid damage to the air ducts and the dampers, the start-up of the fans is delayed until the feedback "Open" from the dampers is present at the input *bDampersFeedbOpen*. In addition the timer *tDelaySupplyVent* must have elapsed. The set time must be longer than the stroke time of the damper drive. **Since this timer is also effective when the plant is switched off, shutdown ramps of frequency converters must also be taken into account!**

If there is no feedback from the damper drives within the set time, the output *bErrorDampers* becomes TRUE. In order to avoid high start-up currents, the switching on of the exhaust air fan is delayed by the timer *tDelayExhaustVent*.

In systems with a mixed air chamber it makes sense to enable the controller only after the supply air temperature *rSupplyAirTemp* has reached approximately the value of the exhaust air temperature. The maximum deviation is specified by the parameter *rDifferenceTemp*. In systems without a mixed air system a very large value is simply entered here as a constant, e.g. 100. The controller is then enabled without a delay.

After the timer *tDelayLimitCtrl* has expired, the output *bEnableLimitCtrl* is enabled and thus the limit value monitoring of the analog inputs is activated.

If the outside temperature lies below the value of *rPreRinseOutsideTemp*, the value of *iCurrentSequence* is set equal to that of *iStartSeqWinter* at the output. The number of the pre-heater controller must be entered here (see the chapter on sequence controllers). (See the chapter on sequence controllers). If the outside temperature is higher than *rPreRinseOutsideTemp*, start-up can commence with the heat recovery unit or the mixed air chamber. Hence, the number of the sequence controller for the mixed air chamber or for the heat recovery unit is entered at the input *iStartSeqSummer*.

The start-up circuit becomes active in the event of operating requirements of the automatic programs, the time schedules, summer night cooling, back-up operation and overheating protection.

To control the sequence controllers, the enumeration variable *eModeSeqCtrl* is set at the output of the function block as follows:

```
TYPE E_HVACSequenceCtrlMode :
(
eHVACSequenceCtrlMode_Stop := 0,
eHVACSequenceCtrlMode_On := 1,
eHVACSequenceCtrlMode_NightCooling := 2,
eHVACSequenceCtrlMode_FreezeProtection := 3,
eHVACSequenceCtrlMode_OverheatingProtection := 4,
eHVACSequenceCtrlMode_NightCoolingAndOverheatingProtection := 5
);
END_TYPE
```

Requests from all automatic programs are transferred to the system start program via the input variables *bTimeScheduler*, *bNightCooling*, *bFreezeProtect* and *bOverHeatingProtect*. Requests from retention functions are only applied if there is no request from a timer program at the input variable *bTimeScheduler*.

The programs for summer night cooling (*bNightCooling*), cooling protection (*bFreezeProtect*) and overheating protection (*bOverHeatingProtect*) are only usable if a room temperature sensor is used in place of an exhaust air temperature sensor. The cooling protection program always has priority over the other automatic programs unless the operation request from the timer program is present.

Since summer night cooling can take place as forced ventilation with fans or as convection ventilation without fans, the command from the program *FB_HVACSummerNightCooling* is transferred with the counter variable *E_HVACConvectionMode*. This output must be applied to the input variable *eConvection*.

A collection of all air conditioning system error messages that lead to the system switching off is applied to the input *bEmergencyStop*.

When the air conditioning system is switched off, the order of the switch-on steps is reversed.

- Deactivating the limit monitoring at the analog inputs
- Deactivating the controllers (resetting *iCurrentSequence:= 0*)
- Switching off the fans
- Closing the dampers after the timer has expired *tDelaySupplyVent*.


VAR_INPUT

```
eDataSecurityType      : E_HVACDataSecurityType;
bSetDefault             : BOOL;
iStartSeqWinter        : INT;
iStartSeqSummer        : INT;
bTimeScheduler         : BOOL;
bNightCooling          : BOOL;
bFreezeProtect         : BOOL;
bOverHeatingProtect    : BOOL;
rOutsideTemp           : REAL;
rReturnWaterTemp       : REAL;
rSupplyAirTemp         : REAL;
```

```
rReturnAirTemp      : REAL;
bDampersFeedbOpen  : BOOL;
bEmergencyStop      : BOOL;
bReset              : BOOL;
```

eDataSecurityType: if `eDataSecurityType:= eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

iStartSeqWinter: the number of the sequence controller for the preheater must be entered here.

iStartSeqSummer: the number of the sequence controller for the mixed air chamber or for the heat recovery unit must be entered here.

bTimeScheduler: input for the timer switch command.

bNightCooling: input for the night cooling function.

bFreeze Protec: input for the frost protection function.

bOverHeating Protec: input for the overheating protection function.

rOutsideTemp: input for the outside temperature.

rReturnWaterTemp: input for the water temperature from the return flow of the air heater.

rSupplyAirTemp: input for the supply air temperature.

rReturnAirTemp: input for the room temperature or the exhaust air temperature.

bDampersFeedbOpen: input for the feedback from the dampers.

bEmergencyStop: this input can be used to create a collection of all fault messages of the VAC system, which then lead to the emergency shutdown of the system.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
eModeSeqCtrl        : E_HVACSequenceCtrlMode;
bPreRinse           : BOOL;
bDampersOpen        : BOOL;
bSupplyVentilation  : BOOL;
bExhaustVentilation : BOOL;
bStartPIDControllers: BOOL;
bEnableLimitCtrl    : BOOL;
bErrorGeneral       : BOOL;
bErrorNoHotWater    : BOOL;
bErrorDampers       : BOOL;
byError             : BYTE;
bInvalidParameter   : BOOL;
```

eModeSeqCtrl: Enum that notifies the operation request.

bPreRinse: TRUE if the pre-rinsing procedure is active.

bDampersOpen: if TRUE, then the outside and exhaust air dampers are open.

bSupplyVentilation: switch on supply air fan if TRUE.

bExhaustVentilation: switch on exhaust air fan if TRUE.

bStartPIDControllers: enable control.

bEnableLimitCtrl: after the timer *tDelayLimitCtrl* has expired, the output *bEnableLimitCtrl* is enabled and thus the limit value monitoring of the analog inputs is activated.

bErrorGeneral: there is a general error.

bErrorNoHotWater: is set to TRUE if the rinsing procedure has ended and the set temperature in the return flow has not been reached.

bErrorDampers: there is an error in the dampers.

byError: output of the error as a byte.

byError.1:= bInvalidParameter;

byError.2:= bErrorGeneral;

byError.3:= bErrorNoHotWater;

byError.4:= bErrorDampers;

bInvalidParameter: Indicates that an incorrect input parameter is present. *bInvalidParameter* must be acknowledged with *bReset*.

VAR_IN_OUT

```
eConvection      : E_HVACConvectionMode;
tDelaySupplyVent : TIME;
tDelayExhaustVent : TIME;
tDelayTempCtrl   : TIME;
rDifferenceTemp  : REAL;
tDelayLimitCtrl  : TIME;
rPreRinseOutsideTemp : REAL;
rPreRinseLimitTemp : REAL;
tPreRinseMaxTime : TIME;
iCurrentSequence : INT;
```

eConvection: ENUM via which summer night cooling can take place as forced ventilation with fans or as convection ventilation without fans.

TYPE E_HVACConvectionMode :

```
(
eHVACConvectionMode_WithFan := 0,
eHVACConvectionMode_WithoutFan := 1
);
END_TYPE
```

The variable is saved persistently. Preset to 1.

tDelaySupplyVent: time delay that delays the start-up of the supply air fans. The variable is saved persistently. Preset to 120 s.

tDelayExhaustVent: time delay that delays the start-up of the exhaust air fans. The variable is saved persistently. Preset to 5 s.

tDelayTempCtrl: time delay for the controller. The variable is saved persistently. Preset to 10 s.

rDifferenceTemp: the controller is enabled if the difference between the room temperature and the supply air temperature is smaller than *rDifferenceTemp*. The variable is saved persistently. Preset to 5 K.

tDelayLimitCtrl: time delay before limit monitoring of the temperature sensors is enabled. See the variable *bEnableLimitCtrl* in FB_HVACTemperature regarding this point. The variable is saved persistently. Preset to 360 s.

rPreRinseOutsideTemp: outside temperature below which the VAC system should start up with the prerinsing of the heating coil. The variable is saved persistently. Preset to 10 K.

rPreRinseLimitTemp: temperature of the return flow from the heating coil that must be reached in order to end the pre-rinsing procedure and continue with the opening of the dampers. The variable is saved persistently. Preset to 30 K.

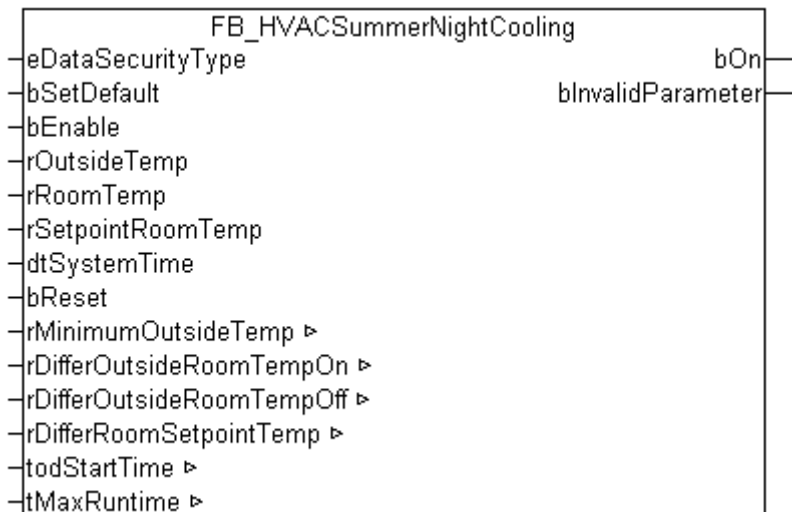
tPreRinseMaxTime: maximum time of the pre-rinse process. The variable is saved persistently. Preset to 300 s.

iCurrentSequence: the starting controller at controller enable is specified here. The value is written by the start program in one cycle only, since afterwards in system operation this parameter must be freely writeable again for the switching on or off of the sequence controllers. The variable is saved persistently.

Documents about this

 example_persistent_e.zip (Resources/zip/11659714827.zip)

3.6.31 FB_HVACSummerNightCooling



Application

With this function block, rooms that were heated up on the day before can be cooled down during the night using cool outside air. The summer night cooling function serves to improve the quality of the air and to save electrical energy. Electrical energy for cooling is saved during the first hours of the next summer day.

The start conditions for the summer night cooling are defined by parameterizing the **FB_HVACSummerNightCooling** function block. The function block can be used to open motor-driven windows or to switch air conditioning systems to summer night cooling mode outside their normal hours of operation. Summer night cooling mode is active if the output variable *bOn* is TRUE.

The following conditions must be met to enable summer night cooling:

- *bEnable* = TRUE
- $rOutsideTemp > (rMinimumOutsideTemp + 0.2K)$
- $((rRoomTemp - rOutsideTemp) > rDifferOutsideRoomTempOn)$
- $((rRoomTemp - rSetpointRoomTemp) > rDifferRoomSetpointTemp)$
- the system time *dtSystemtime* must lie within the timeframe *todStartTime* to 12:00 noon

It is sufficient to disable summer night cooling *bOn* = FALSE if one of the following conditions is met:

- *bEnable* = FALSE
- $(rOutsideTemp < (rMinimumOutsideTemp - 0.2K))$
- $((rRoomTemp - rOutsideTemp) < rDifferOutsideRoomTempOff)$

- the system time *dtSystemtime* is outside of the timeframe from *todStartTime* to 12:00 noon
- the summer night cooling had been activated for the maximum time *tMaxRuntime* within the timeframe of *todStartTime* to 12:00 noon. Summer night cooling can be switched on several times within this timeframe.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
bEnable         : BOOL;
rOutsideTemp    : REAL;
rRoomTemp       : REAL;
rSetpointRoomTemp: REAL;
dtSystemTime    : DT;
bReset          : BOOL;
```

eDataSecurityType: if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If *eDataSecurityType* := *eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled by the PLC program with the input variable *bEnable*.

rOutsideTemp: input for the outside temperature.

rRoomTemp: input for the room temperature.

rSetpointRoomTemp: setpoint for the room temperature

dtSystemTime: this variable transfers the computer system time to the function block.

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```
bOn              : BOOL;
bInvalidParameter: BOOL;
```

bOn: if *bOn* = TRUE, the summer night cooling is activated.

The following conditions must be met to activate summer night cooling:

$bEnable = TRUE \text{ AND } rOutsideTemp > (rMinimumOutsideTemp + 0.2K) \text{ AND } ((rRoomTemp - rOutsideTemp) > rDifferOutsideRoomTempOn) \text{ AND } ((rRoomTemp - rSetpointRoomTemp) > rDifferRoomSetpointTemp) \text{ AND the system time } dtSystemtime \text{ must lie within the timeframe } todStartTime \text{ until } 12:00 \text{ noon.}$

It is sufficient to disable the summer night cooling *bOn* = FALSE if one of the following conditions is met: $bEnable = FALSE \text{ OR } (rOutsideTemp < (rMinimumOutsideTemp - 0.2K)) \text{ OR } ((rRoomTemp - rOutsideTemp) < rDifferOutsideRoomTempOff) \text{ OR the system time } dtSystemtime \text{ is outside of the timeframe from } todStartTime \text{ to } 12:00 \text{ noon OR the summer night cooling had been activated for the maximum time } tMaxRuntime \text{ within the system time of } todStartTime \text{ to } 12:00 \text{ noon. Summer night cooling can be switched on several times within this timeframe.}$

blInvalidParameter: Indicates that an incorrect input parameter is present. *blInvalidParameter* must be acknowledged with *bReset*.

VAR_IN_OUT

```
rMinimumOutsideTemp      : REAL;
rDifferOutsideRoomTempOn : REAL;
rDifferOutsideRoomTempOff : REAL;
rDifferRoomSetpointTemp  : REAL;
todStartTime              : TOD;
tMaxRuntime               : TIME;
```

rMinimumOutsideTemp: if the outside temperature *rOutsideTemp* falls below the limit value (*rMinimumOutsideTemp* - 0.2 K), the summer night cooling function is disabled, i.e. *bOn* = FALSE. If the outside temperature *rOutsideTemp* exceeds the limit value (*rMinimumOutsideTemp* + 0.2K), then one of the conditions for activating the summer night cooling *bOn* = TRUE is satisfied (4..100). The variable is saved persistently. Preset to 10 °C.

rMinimumOutsideTemp must be within its range.

If a variable value is not correct, the last valid variable value is used, if available. If there is no valid last value, then the default value is used. *blInvalidParameter* is set if the parameter is incorrect.

rDifferOutsideRoomTempOn: the difference *rRoomTemp* - *rOutsideTemp* must be greater than the amount of *rDifferOutsideRoomTempOn* so that one of the conditions for activating summer night cooling *bOn* = TRUE is met (0..100).

$((rRoomTemp - rOutsideTemp) > rDifferOutsideRoomTempOn)$

rDifferOutsideRoomTempOn must be 0.4 K higher than *rDifferOutsideRoomTempOff*. In addition, *rDifferOutsideRoomTempOn* must be within its range.

If a variable value is not correct, the last valid variable value is used, if available. If there is no valid last value, then the default value is used. *blInvalidParameter* is set if the parameter is incorrect.

The variable is saved persistently. Preset to 5 °C.

rDifferOutsideRoomTempOff: the difference *rRoomTemp* - *rOutsideTemp* must be less than the amount of *rDifferOutsideRoomTempOff* to disable summer night cooling *bOn* = FALSE.

$((rRoomTemp - rOutsideTemp) < rDifferOutsideRoomTempOff)$

rDifferOutsideRoomTempOff must be 0.4 K smaller than *rDifferOutsideRoomTempOn*. In addition, *rDifferOutsideRoomTempOff* must be within its range (0..100).

If a variable value is not correct, the last valid variable value is used, if available. If there is no valid last value, then the default value is used. *blInvalidParameter* is set if the parameter is incorrect. The variable is saved persistently. Preset to 2 °C.

rDifferRoomSetpointTemp: the room temperature must be greater than the room temperature setpoint by this amount so that one of the conditions for activating summer night cooling *bOn* = TRUE is met.

$((rRoomTemp - rSetpointRoomTemp) > rDifferRoomSetpointTemp)$

rDifferRoomSetpointTemp must be within its range (0..100).

If a variable value is not correct, the last valid variable value is used, if available. If there is no valid last value, then the default value is used. *blInvalidParameter* is set if the parameter is incorrect.

The variable is saved persistently. Preset to 2 °C.

todStartTime: start time for the timeframe within which the summer night cooling can be activated (0..24). The timeframe for enabling summer night cooling starts with *todStartTime* and ends at 12 noon.

todStartTime must be within its range.

If a variable value is not correct, the last valid variable value is used, if available. If there is no valid last value, then the default value is used. *blInvalidParameter* is set if the parameter is incorrect. The variable is saved persistently. Preset to 2 o'clock.

tMaxRuntime: maximum runtime of the summer night cooling function within the timeframe between *todStartTime* and 12 noon (>0s). Summer night cooling can be switched on several times within this timeframe, but the total time cannot exceed *tMaxRuntime*.

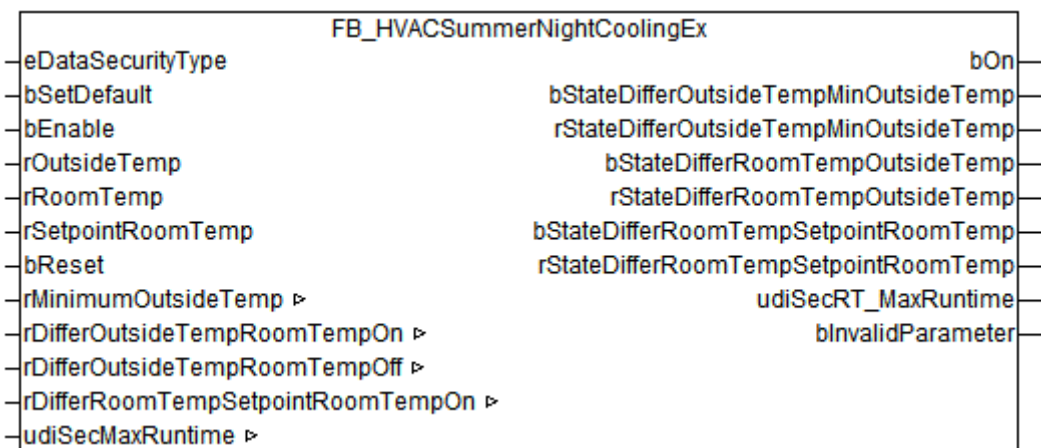
$tMaxRuntime$ must be greater than $T\#0s$.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *blInvalidParameter* is set if the parameter is incorrect. The variable is saved persistently. Preset to 20 min.

Documents about this

 example_persistent_e.zip (Resources/zip/11659714827.zip)

3.6.32 FB_HVACSummerNightCoolingEx



Application

With this function block, rooms that were heated up on the day before can be cooled down during the night using cool outside air. The summer night cooling function serves to improve the quality of the air and to save electrical energy. Electrical energy for cooling is saved during the first hours of the next summer day.

The start conditions for the summer night cooling are defined by parameterizing the **FB_HVACSummerNightCooling** function block. The function block can be used to open motor-driven windows or to switch air conditioning systems to summer night cooling mode outside their normal hours of operation. Summer night cooling mode is active if the output variable *bOn* is TRUE.

The following conditions must be met to activate summer night cooling:

- *bEnable* = TRUE
- $rOutsideTemp > (rMinimumOutsideTemp + 0.2K)$; the value of 0.2K is an internal constant of the function block.
- $(rRoomTemp - rOutsideTemp) > rDifferOutsideTempRoomTempOn$
- $(rRoomTemp - rSetpointRoomTemp) > rDifferRoomTempSetpointRoomTempOn$
- $udiSecRT_MaxRuntime > 0$

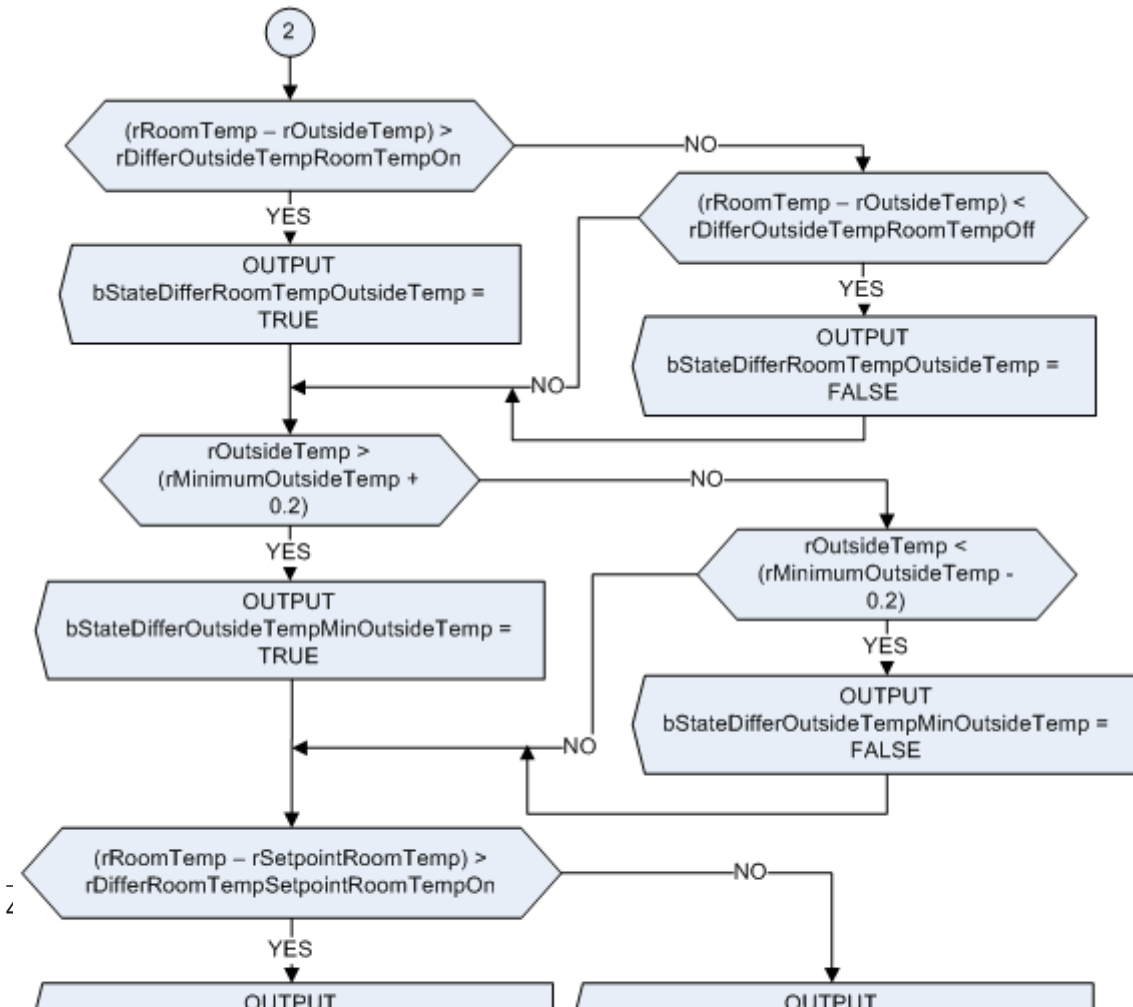
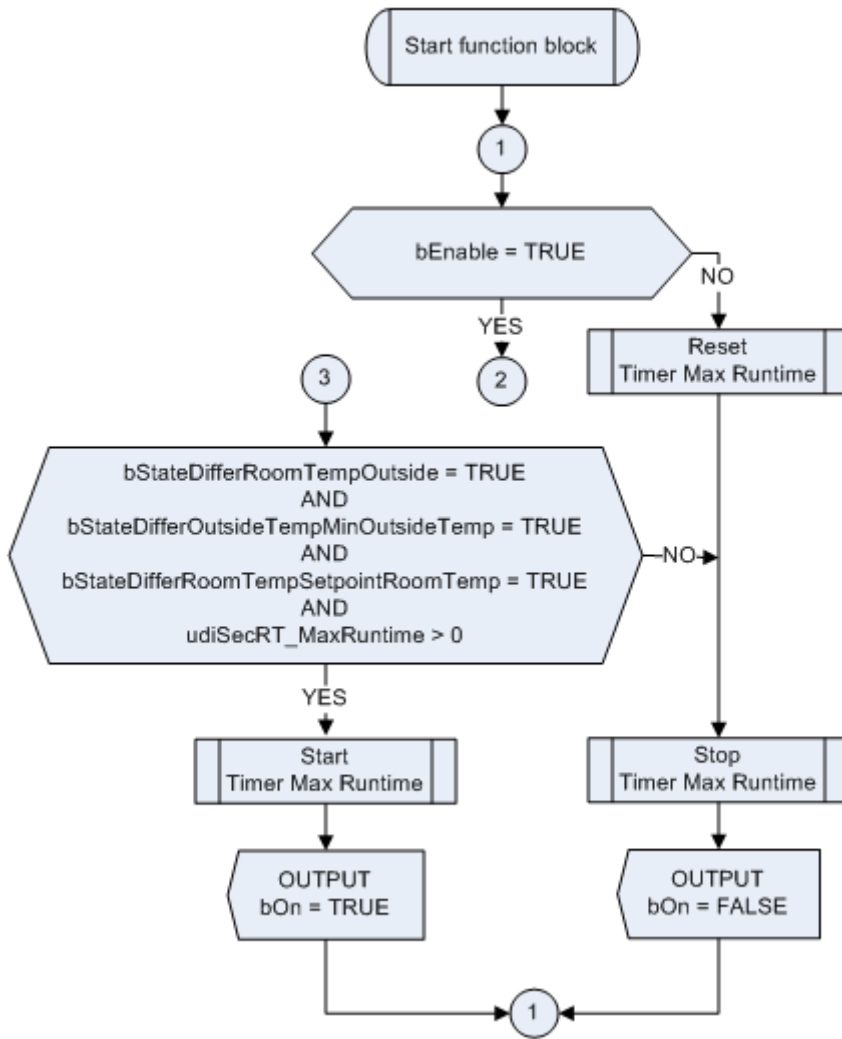
It is sufficient to disable the summer night cooling *bOn* = FALSE if one of the following conditions is met:

- *bEnable* = FALSE
- $rOutsideTemp < (rMinimumOutsideTemp - 0.2K)$; the value of 0.2K is an internal constant of the function block.
- $(rRoomTemp - rOutsideTemp) < rDifferOutsideTempRoomTempOff$
- $udiSecRT_MaxRuntime = 0$



Attention Summer night cooling can be switched on and off several times within the time *udiSecMaxRuntime*. Once *udiSecRT_MaxRuntime* has elapsed the summer night cooling function can only be reactivated if *bEnable* is FALSE for at least one PLC cycle. The function block and the summer night cooling function can be enabled via a timer.

Program flowchart



Application example

The application example shows the function block FB_HVACSummerNightCoolingEx in conjunction with the weekly timer FB_HVACScheduler1ch. The example is available in the programming languages ST and CFC. The program example **P_CFC_SummernightCoolingEx.PRG** for the CFC programming languages can be found in the folder **Language CFC > SpecialFunctions**, the program example **P_ST_SummernightCoolingEx.PRG** for the ST programming languages in the folder **Language Structur Text > SpecialFunctions**.


Download	Required library
TcHVAC.pro [▶ 531]	TcHVAC.lib

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
bEnable          : BOOL;
rOutsideTemp     : REAL;
rRoomTemp        : REAL;
rSetpointRoomTemp: REAL;
bReset           : BOOL;
```

eDataSecurityType: if eDataSecurityType:= *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

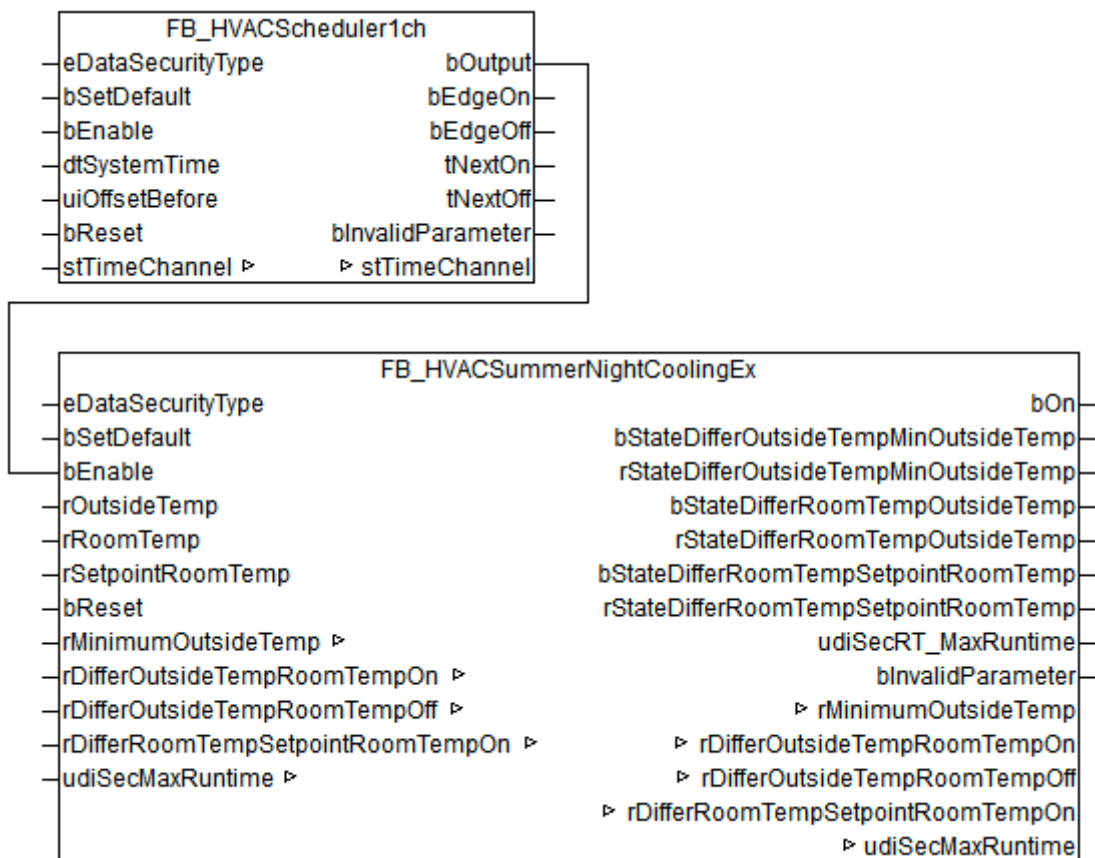
Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If eDataSecurityType:= *eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE
A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType:= <i>eHVACDataSecurityType_Persistent</i> . It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled by the PLC program with the input variable *bEnable*. Once the time *udiSecMaxRuntime* has elapsed the summer night cooling function can only be reactivated if *bEnable* is FALSE for at least one PLC cycle. The function block could be enabled via a timer.



rOutsideTemp: input for the outside temperature.

rRoomTemp: input for the room temperature

rSetpointRoomTemp: setpoint for the room temperature

bReset: acknowledge input in the event of a fault.

VAR_OUTPUT

```

bOn : BOOL;
bStateDifferOutsideTempMinOutsideTemp : BOOL;
rStateDifferOutsideTempMinOutsideTemp : REAL;
bStateDifferRoomTempOutsideTemp : BOOL;
rStateDifferRoomTempOutsideTemp : REAL;
bStateDifferRoomTempSetpointRoomTemp : BOOL;
rStateDifferRoomTempSetpointRoomTemp : REAL;
udiSecRT_MaxRuntime : UDINT;
bInvalidParameter : BOOL;

```

bOn: if *bOn* = TRUE summer night cooling is active.

The following conditions must be met to activate summer night cooling:

- *bEnable* = TRUE
- $rOutsideTemp > (rMinimumOutsideTemp + 0.2K)$; the value of 0.2K is an internal constant of the function block.
- $(rRoomTemp - rOutsideTemp) > rDifferOutsideTempRoomTempOn$
- $(rRoomTemp - rSetpointRoomTemp) > rDifferRoomTempSetpointRoomTempOn$
- $udiSecRT_MaxRuntime > 0$

It is sufficient to disable the summer night cooling *bOn* = FALSE if one of the following conditions is met:

- *bEnable* = FALSE
- $rOutsideTemp < (rMinimumOutsideTemp - 0.2K)$; the value of 0.2K is an internal constant of the function

block.

- $(rRoomTemp - rOutsideTemp) < rDifferOutsideTempRoomTempOff$
 $-udiSecRT_MaxRuntime = 0$

bStateDifferOutsideTempMinOutsideTemp: state display. *bStateDifferOutsideTempMinOutsideTemp* is TRUE if $rOutsideTemp > (rMinimumOutsideTemp + 0.2)$. *bStateDifferOutsideTempMinOutsideTemp* is FALSE if $rOutsideTemp < (rMinimumOutsideTemp - 0.2)$.

rStateDifferOutsideTempMinOutsideTemp: value of difference $rOutsideTemp - rMinimumOutsideTemp$

bStateDifferRoomTempOutsideTemp: state display. *bStateDifferRoomTempOutsideTemp* is TRUE if $(rRoomTemp - rOutsideTemp) > rDifferOutsideTempRoomTempOn$. *bStateDifferRoomTempOutsideTemp* is FALSE if $(rRoomTemp - rOutsideTemp) < rDifferOutsideTempRoomTempOff$ is.

rStateDifferRoomTempOutsideTemp: value of difference $rRoomTemp - rOutsideTemp$

bStateDifferRoomTempSetpointRoomTemp: state display. *bStateDifferRoomTempSetpointRoomTemp* is TRUE if $(rRoomTemp - rSetpointRoomTemp) > rDifferRoomTempSetpointRoomTempOn$. *bStateDifferRoomTempSetpointRoomTemp* is FALSE if $(rRoomTemp - rSetpointRoomTemp) <= rDifferRoomTempSetpointRoomTempOn$ is.

rStateDifferRoomTempSetpointRoomTemp: value of difference $rRoomTemp - rSetpointRoomTemp$

udiSecRT_MaxRuntime: remaining summer night cooling time. If $udiSecRT_MaxRuntime = 0$, the summer night cooling function can be enabled again only if *bEnable* is FALSE for at least one PLC cycle

blInvalidParameter: Indicates that an incorrect input parameter is present. *blInvalidParameter* must be acknowledged with *bReset*.

VAR_IN_OUT

```
rMinimumOutsideTemp      : REAL;
rDifferOutsideTempRoomTempOn  : REAL;
rDifferOutsideTempRoomTempOff : REAL;
rDifferRoomTempSetpointRoomTempOn : REAL;
udiSecMaxRuntime          : UDINT;
```

rMinimumOutsideTemp: if the outside temperature *rOutsideTemp* falls below the limit value $(rMinimumOutsideTemp - 0.2 \text{ K})$, the summer night cooling function is disabled, i.e. *bOn* = FALSE. If the outside temperature *rOutsideTemp* exceeds the limit value $(rMinimumOutsideTemp + 0.2 \text{ K})$, then one of the conditions for activating the summer night cooling *bOn* = TRUE is satisfied (4..100).

rMinimumOutsideTemp must be within its range.

If a variable value is not correct, the last valid variable value is used, if available. If there is no valid last value, then the default value is used. *blInvalidParameter* is set if the parameter is incorrect. The variable is saved persistently. Preset to 10 °C.

rDifferOutsideTempRoomTempOn: the difference $rRoomTemp - rOutsideTemp$ must be greater than the amount of *rDifferOutsideTempRoomTempOn* so that one of the conditions for activating summer night cooling *bOn* = TRUE is met (0..10).

$((rRoomTemp - rOutsideTemp) > rDifferOutsideRoomTempOn)$

rDifferOutsideRoomTempOn must be 0.4 K higher than *rDifferOutsideRoomTempOff*. In addition, *rDifferOutsideRoomTempOn* must be within its range.

If a variable value is not correct, the last valid variable value is used, if available. If there is no valid last value, then the default value is used. *blInvalidParameter* is set if the parameter is incorrect. The variable is saved persistently. Preset to 5 °C.

rDifferOutsideTempRoomTempOff: the difference $rRoomTemp - rOutsideTemp$ must be less than the amount of *rDifferOutsideTempRoomTempOff* to disable summer night cooling *bOn* = FALSE.

$((rRoomTemp - rOutsideTemp) < rDifferOutsideRoomTempOff)$

rDifferOutsideRoomTempOff must be 0.4 K smaller than *rDifferOutsideRoomTempOn*. In addition, *rDifferOutsideRoomTempOff* must be within its range (0..10).

If a variable value is not correct, the last valid variable value is used, if available. If there is no valid last value, then the default value is used. *blInvalidParameter* is set if the parameter is incorrect. The variable is saved persistently. Preset to 2 °C.

rDifferRoomTempSetpointRoomTempOn: the room temperature must be greater than the room temperature setpoint by this amount so that one of the conditions for activating summer night cooling *bOn* = TRUE is met (0..10).

$(rRoomTemp - rSetpointRoomTemp) > rDifferRoomSetpointTemp$

rDifferRoomSetpointTemp must be within its range.

If a variable value is not correct, the last valid variable value is used, if available. If there is no valid last value, then the default value is used. *blInvalidParameter* is set if the parameter is incorrect. The variable is saved persistently. Preset to 2 °C.

udiSecMaxRuntime: maximum runtime of the summer night cooling function. Summer night cooling can be switched on and off several times within the time. The remaining summer night cooling time is indicated via the variable *udiSecRT_MaxRuntime*.

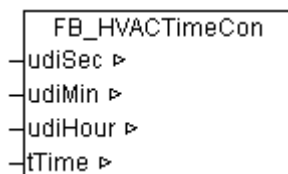
udiSecMaxRuntime must be greater than 0.

If an incorrect variable value is present, the last valid variable value, if available, is used. If there is no valid last value, then the default value is used. *blInvalidParameter* is set if the parameter is incorrect. The variable is saved persistently. Preset to 180.

Documents about this

 [example_persistent_e.zip](#) (Resources/zip/11659714827.zip)

3.6.33 FB_HVACTimeCon



Application

This function block converts a TIME variable *tTime* into the three UDINT variables *udiSec*, *udiMin* and *udiHour*, which indicate the seconds, minutes and hours. This conversion is reversible, so that a TIME variable can be formed from seconds, minutes and hours.

VAR_IN_OUT

```

udiSec      : UDINT;
udiMin      : UDINT;
udiHour     : UDINT;
tTime       : TIME;
  
```

udiSec: variable that displays the seconds (0..59).

udiMin: variable that displays the minutes (0..59).

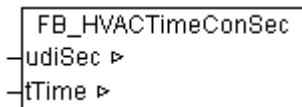
udiHour: variable that displays the hours (0..1191).

tTime: variable of the type Time.

Application example

Download	Required library
TcHVAC.pro [► 531]	TcHVAC.lib

3.6.34 FB_HVACTimeConSec



Application

This function block converts a TIME variable *tTime* into a UDINT variable *udiSec*, which indicates the seconds. This conversion is reversible, so that a TIME variable can be formed from seconds.

VAR_IN_OUT

```

udiSec      : UDINT;
tTime       : TIME;
  
```

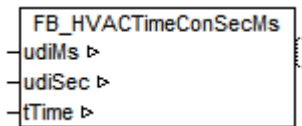
udiSec: variable that displays the seconds (0..4294967)

tTime: variable of the type Time

Application example

Download	Required library
TcHVAC.pro [▶ 531]	TcHVAC.lib

3.6.35 FB_HVACTimeConSecMs



Application

This function block converts a TIME variable *tTime* into two UDINT variables, *udiSec* and *udiMs*, which indicate the seconds and milliseconds respectively. The conversion is also reversible, so that a TIME variable can be formed from the seconds and milliseconds.

VAR_IN_OUT

```

udiMs       : UDINT;
udiSec      : UDINT;
tTime       : TIME;
  
```

udiMs: variable that displays the milliseconds (0..999).

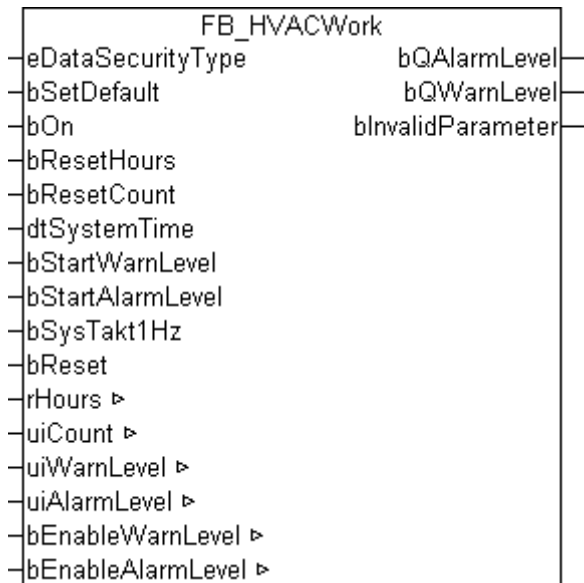
udiSec: variable that displays the seconds (0..4294966).

tTime: variable of the type Time.

Application example

Download	Required library
TcHVAC.pro [▶ 531]	TcHVAC.lib

3.6.36 FB_HVACWork



Application

This function block serves the recording of the operating hours and the switch-on cycles. Furthermore it is possible to report specified operating times being reached.

Recording of operating hours

As soon as the input *bOn* reports operation, the operating time is recorded with an internal resolution of 1 sec. The output is in the form of a real value in hours to two decimal places (1/100 h). If no system time with second resolution is available, or if this is disrupted, the clock input *bSysTakt1Hz* is used instead after 2 s. Triggering takes place on a rising edge at this input. The variant with the system time is more exact than the variant with the clock generator on account of the possibility of synchronization via the network.

Every positive edge at *bOn* is counted as a switch-on cycle and is available at the output as *iCount*.

VAR_INPUT


```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bOn               : BOOL;
bResetHours       : BOOL;
bResetCount       : BOOL;
dtSystemtime      : DT;
bStartWarnLevel   : BOOL;
bStartAlarmLevel  : BOOL;
bSysTakt1Hz       : BOOL;
bReset            : BOOL;
```

NOTICE

If `E_HVACDataSecurityType := eHVACDataSecurityType_Persistent` has been selected, the procedure deviates from the standard. After a change, the data are not written to the flash memory with a delay equivalent to `g_tHVACWriteBackupDataTime` (default `t#5s`), but constantly only once per hour. This serves to protect the flash memory, as otherwise all persistent data would be written to it every 5 sec if the operating hour counter were to be running.

eDataSecurityType: if `eDataSecurityType := eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bOn: operating message; the operating time is totaled up as long as the signal is present. The rising edge is counted as the switch-on cycle.

bResetHours: a rising edge sets the operating hours counter to 0.

bResetCount: a rising edge sets the switch-on cycle counter to 0.

dtSystemtime: system time with seconds display.

bStartWarnLevel: a rising edge at the input resets the output `bQWarnLevel` and restarts the time measurement for this.

bStartAlarmLevel: a rising edge at the input resets the output `bQAlarmLevel` and restarts the time measurement for this.

bSysTakt1Hz: 1 Hz clock signal as a replacement for `dtSystemTime`; if `dtSystemTime` is not available, or does not change its value for longer than 2 s, then the clock signal is used as a replacement.

bReset: acknowledge input in the event of a fault. Resets the flag `blInvalidParameter`.

VAR_IN_OUT

```
rHours          : REAL;
uiCount         : UINT;
uiWarnLevel     : UINT;
uiAlarmLevel    : UINT;
bEnableWarnLevel : BOOL;
bEnableAlarmLevel : BOOL;
```

rHours: operating hours with a resolution of 1/100 hours (internally with 1 s). The variable is saved persistently.

uiCount: switch-on cycle counter. The variable is saved persistently.

uiWarnLevel: number of operating hours after which a warning is output (0..50000). Upon a rising edge at the input `bStartWarnLevel` the current operating hours count is added to the value `uiWarnLevel` and saved as an absolute operating time. As soon as the operating hours counter has reached this value and the signal is enabled via `bEnableWarnLevel`, the output `bQWarnLevel` is set to TRUE. The variable is saved persistently. Preset to 0 h.

uiAlarmLevel: number of operating hours after which an alarm is output (0..50000). Upon a rising edge at the input `bStartAlarmLevel` the current operating hours count is added to the value `uiAlarmLevel` and saved as an absolute operating time. As soon as the operating hours counter has reached this value and the signal is enabled via `bEnableAlarmLevel`, the output `bQAlarmLevel` is set to TRUE. The variable is saved persistently. Preset to 0 h.

bEnableWarnLevel: enables the output `bQWarnLevel`. The variable is saved persistently. Preset to FALSE.

bEnableAlarmLevel: enables the output `bQAlarmLevel`. The variable is saved persistently. Preset to FALSE.

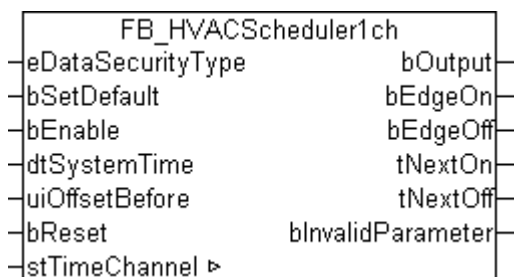
VAR_OUTPUT

```
bQAlarmLevel      : BOOL;
bQWarnLevel       : BOOL;
bInvalidParameter: BOOL;
```

bQAlarmLevel:operating hours counter has reached the alarm level.

bQWarnLevel:operating hours counter has reached the warning level.

bInvalidParameter: an error occurred during the plausibility check. It is deleted again by *bReset*.

3.7 HVAC Time schedule**3.7.1 FB_HVACScheduler1ch****Application**

This function block is a weekly timer switch with one timer switch channel. Any desired combination of days of the week can be assigned to the timer switch channel. The timer switch channel can be defined for one-shot or recurring events. Besides the switching signal, the switch-on and switch-off edges plus the next switch-on or switch-off time as a countdown are available as outputs. The data of the timer switch channel is transferred as a reference. The function block is therefore able to check the data for changes and to save it as persistent data. After booting, the data are thus available to the BMS again. The switch-on and switch-off points can be specified to the exact second via the variables *tiOn_ss* and *tiOff_ss* of the respective time channel.



If the input variable *bEnable* = FALSE, then the output *bOutput* = FALSE. The timer switch channel *stTimeChannel* remains active.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
dtSystemTime      : DT;
uiOffsetBefore    : UINT;           0 .. 240   min
bReset            : BOOL;
```

eDataSecurityType:if *eDataSecurityType*:= *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If *eDataSecurityType*:= *eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled with **bEnable**. If `bEnable = FALSE` there is no switching at the output **bOutput**.

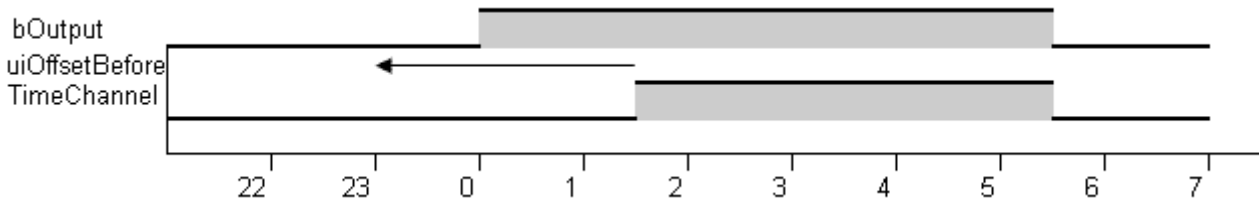
dtSystemTime: the variable **dtSystemTime** transfers the computer system time to the timer program.

uiOffsetBefore: the switch-on time is brought forward by this number of minutes. This function is useful for the time-optimized switching on of heating circuits. A maximum of 240 min, i.e. 4 hours, may be entered. If the resulting switch-on time is before midnight on the previous day, then midnight will be chosen as the switch-on time. The switch-off time is not affected.

bReset: acknowledge input in the event of a fault. Resets the flag `blInvalidParameter`

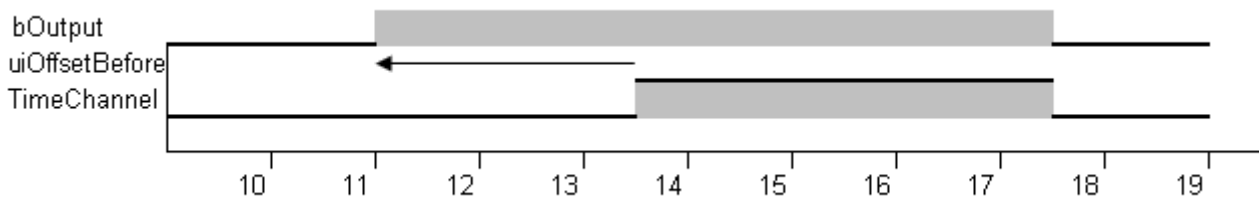
Example 1

Switch-on time from 1:30 to 5:30, `uiOffsetBefore` has the value 150 min. The switch-on time is only advanced by 90 min to 00:00.



Example 2:

Switch-on time from 13:30 to 17:30; `uiOffsetBefore` has the value 150 min. The switch-on time is brought forward by 150 minutes to 11:00.



VAR_IN_OUT

```
stTimeChannel : ST_HVACTimeChannel;
```

stTimeChannel: with the variable `stTimeChannel` it is possible to generate one switch-on time per day of the week.

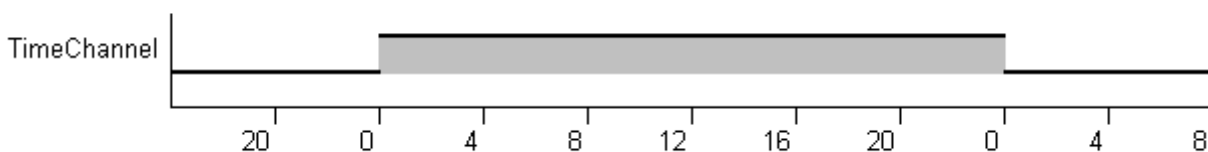
```
TYPE ST_HVACTimeChannel :
STRUCT
  uiOn_hh : UINT;
  uiOn_mm : UINT;
  uiOn_ss : UINT;
  uiOff_hh : UINT;
  uiOff_mm : UINT;
  uiOff_ss : UINT;
  bEnable : BOOL;
  bAllDays : BOOL;
  bMonday : BOOL;
  bTuesday : BOOL;
```

```
bWednesday : BOOL;
bThursday  : BOOL;
bFriday    : BOOL;
bSaturday  : BOOL;
bSunday    : BOOL;
bResetAfterOn: BOOL;
bQ         : BOOL;
END_STRUCT
END_TYPE
```

Special notes regarding time input

The following applies:

- Switch-on time 00:00:00: switch-on begins at 0:00 on the selected day.
- Switch-off time 00:00:00:° switch-off takes place at 23:59:59 on the selected day.



The following applies:

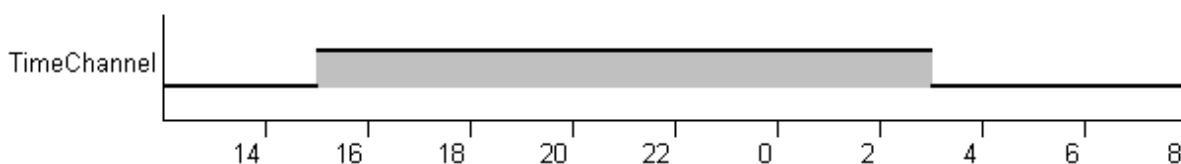
If the switch-on time is later than the switch-off time, the switch-on period is extended through midnight into the next day, regardless of whether the next day is selected.

Enable Reset after On

| all Days | Mo Di Mi Do Fr Sa So OnTime hh:mm:ss OffTime hh:mm:ss

15 : 0 : 0 => 3 : 0 : 0

This setting produces:




For each instance of the function block **FB_HVACScheduler1ch** it is necessary to create a structure variable of type *ST_HVACTimeChannel*, which is transferred as a reference (VAR_IN_OUT).

Sample:

```
VAR_GLOBAL
  stTimeChannel : ST_HVACTimeChannel;
END_VAR
```

Within the weekly planner each timer switch channel is enabled via the variable **bEnable** (On/Off). If *bEnable* = FALSE, then the timer switch channel is deactivated and the times entered have no effect. If *bAllDays* (Mon-Sun) = TRUE, then the times set in *uiOn_hh*, *uiOn_mm*, *uiOff_hh* and *uiOff_mm* apply to all days of the week from Monday to Sunday. If *bAllDays* is FALSE, then the days of the week can be selectively assigned to a timer switch channel using the variables **bMonday** to **bSunday**. For one-shot events, the variable *bResetAfterOn* (non-recurring) is activated. After the time set in this channel has elapsed, the entries created under *uiOn_hh*, *uiOn_mm*, *uiOff_hh*, *uiOff_mm* and the days of the week are retained. However, the variable **bEnable** (On/Off) is reset to FALSE. In this manner a one-shot event can be defined once and reactivated whenever it is required.

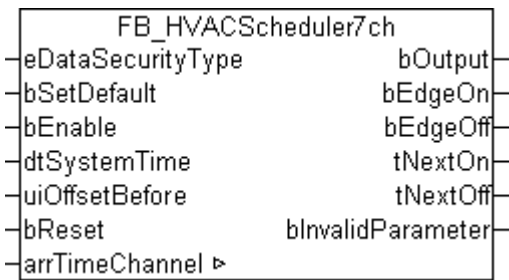
Visualization template for the Target VISU with one timer switch channel:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659727627/.zip>

VAR_OUTPUT

```
bOutput      : BOOL;
bEdgeOn     : BOOL;
bEdgeOff    : BOOL;
tNextOn     : TIME;    0 .. 10080  min    (* Nächster Einschaltpunkt, max 7 Tage * 24 h * 6
0 min = 10080 *)
tNextOff    : TIME;    0 .. 1440   min    (* Nächster Ausschaltpunkt, max gleicher Tage 24
h * 60 min = 1440 *)
bInvalidParameter: BOOL;
```

- bOutput:** is TRUE if one of the timer switch channels has switched on.
- bEdgeOn:** is TRUE for one PLC cycle after bOutput switches on.
- bEdgeOff:** is TRUE for one PLC cycle after bOutput switches off.
- tNextOn:** time until the next switch-on of the timer program. Maximum value 10080 minutes (1 week).
- tNextOff:** time until the next switch-off of the timer program. Maximum forecast until midnight.
- bInvalidParameter:** an error occurred during the plausibility check. It is deleted again by *bReset*.

3.7.2 FB_HVACScheduler7ch



Application

This function block is a weekly timer switch with 7 timer switch channels. Any desired combination of days of the week can be assigned to any timer switch channel. The timer switch channels can be defined for non-recurring or recurring events. Besides the switching signal, the switch-on and switch-off edges plus the next switch-on or switch-off time as a countdown are available as outputs. The data of the timer switch channels is transferred as a reference. The function block is therefore able to check the data for changes and to save it as persistent data. After booting, the data are thus available to the BMS again. The switch-on and switch-off points can be specified to the exact second via the variables *tiOn_ss* and *tiOff_ss* of the respective time channel.




If the input variable *bEnable* = FALSE, then the output *bOutput* = FALSE. The timer switch channels *arrTimeChannel* remain active.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
bEnable         : BOOL;
dtSystemTime    : DT;
uiOffsetBefore  : UINT;    0 .. 240   min
bReset         : BOOL;
```

eDataSecurityType: if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block *FB_HVACPersistentDataHandling* must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled with `bEnable`. If `bEnable = FALSE` there is no switching at the output `bOutput`.

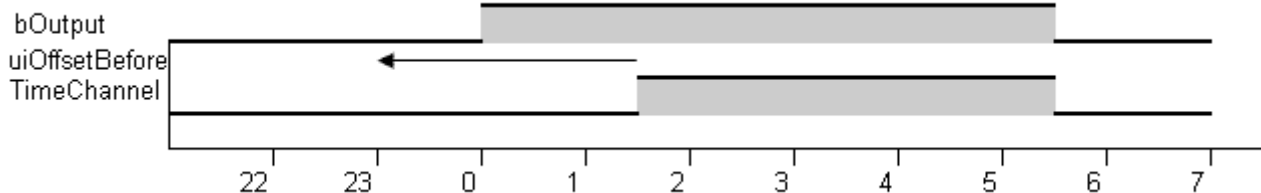
dtSystemTime: the variable `dtSystemTime` transfers the computer system time to the timer program.

uiOffsetBefore: the switch-on time is brought forward by this number of minutes. This function is useful for the time-optimized switching on of heating circuits. A maximum of 240 min, i.e. 4 hours, may be entered. If the resulting switch-on time is before midnight on the previous day, then midnight will be chosen as the switch-on time. The switch-off time is not affected.

bReset: acknowledge input in the event of a fault. Resets the flag `blInvalidParameter`.

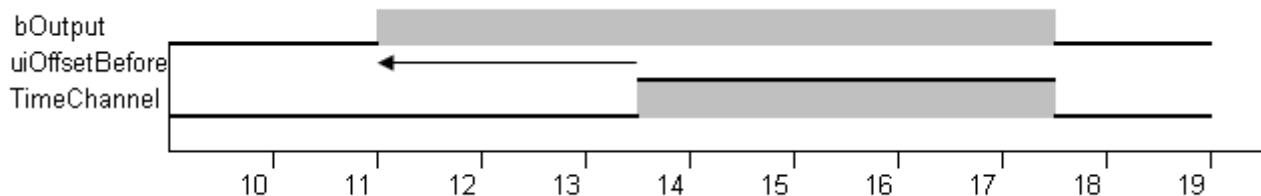
Example 1

Switch-on time from 1:30 to 5:30, `uiOffsetBefore` has the value 150 min. The switch-on time is only advanced by 90 min to 00:00.



Example 2:

Switch-on period from 13:30 to 17:30; `uiOffsetBefore` has the value 150 min. The switch-on time is brought forward by 150 minutes to 11:00.



VAR_IN_OUT

```
arrTimeChannel : ARRAY[1..7] OF ST_HVACTimeChannel;
```

arrScheduleWeekly: the variable *arrScheduleWeekly* is an array with 7 declarations of the structure variable **ST_HVACTimeChannel**. It is possible to generate 1 switch-on time per day of the week with the 7 channels.

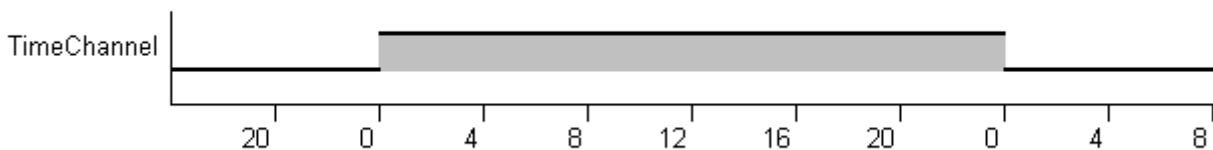
```

TYPE ST_HVACTimeChannel :
STRUCT
  uiOn_hh      : UINT;
  uiOn_mm      : UINT;
  uiOn_ss      : UINT;
  uiOff_hh     : UINT;
  uiOff_mm     : UINT;
  uiOff_ss     : UINT;
  bEnable      : BOOL;
  bAllDays     : BOOL;
  bMonday      : BOOL;
  bTuesday     : BOOL;
  bWednesday   : BOOL;
  bThursday    : BOOL;
  bFriday      : BOOL;
  bSaturday    : BOOL;
  bSunday      : BOOL;
  bResetAfterOn: BOOL;
  bQ           : BOOL;
END_STRUCT
END_TYPE
    
```

Special notes regarding time input

The following applies:

- Switch-on time 00:00:00: switch-on begins at 0:00 on the selected day.
- Switch-off time 00:00:00: switch-off takes place at 23:59:59 on the selected day.



The following applies:

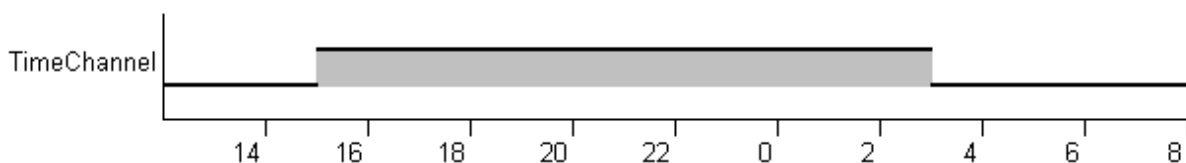
If the switch-on time is later than the switch-off time, the switch-on period is extended through midnight into the next day, regardless of whether the next day is selected.

Enable Reset after On

| all Days | Mo Di Mi Do Fr Sa So OnTime hh:mm:ss OffTime hh:mm:ss

: : => : :

This setting produces:



For each instance of the function block **FB_HVACSchedulerWeekly** it is necessary to create an array of type **ST_HVACTimeChannel**. The size of the array must be 7, since the function block processes a total of 7 weekly timer switch cycles and expects these as a reference (VAR_IN_OUT).

Sample

```
VAR_GLOBAL
  arrTimeChannel : ARRAY[1..7] OF ST_HVACTimeChannel;
END_VAR
```

Within the weekly planner each timer switch channel is enabled via the variable *bEnable* (On/Off). If *bEnable* = FALSE, then the timer switch channel is deactivated and the times entered have no effect. If *bAllDays* (Mon-Sun) = TRUE, then the times set in *uiOn_hh*, *uiOn_mm*, *uiOff_hh* and *uiOff_mm* apply to all days of the week from Monday to Sunday. If *bAllDays* is FALSE, then the days of the week can be selectively assigned to a timer switch channel using the variables *bMonday* to *bSunday*. For one-shot events, the variable *bResetAfterOn* (non-recurring) is activated. After the time set in this channel has elapsed, the entries created under *uiOn_hh*, *uiOn_mm*, *uiOff_hh*, *uiOff_mm* and the days of the week are retained. However, the variable *bEnable* (On/Off) is reset to FALSE. In this manner a one-shot event can be defined once and reactivated whenever it is required.

Example

Enable	Reset after On							OnTime hh:mm:ss	OffTime hh:mm:ss
all Days	Mo	Di	Mi	Do	Fr	Sa	So		
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	7 : 20 : 0	=> 17 : 30 : 0
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	16 : 40 : 0	=> 0 : 0 : 0
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0 : 0 : 0	=> 0 : 0 : 0
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	15 : 0 : 0	=> 3 : 0 : 0
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	12 : 0 : 0	=> 18 : 0 : 0

Monday switched on from 07:20 until 17:30

Tuesday switched on from 16:40 until 00:00

Wednesday switched on from 00:00 until 23:59:59 (around the clock – 24 hours!)

Thursday switched on from 15:00 until 03:00 Friday morning, non-recurring!

Friday Timer switch channel not activated, always switched off!

VAR_OUTPUT

```
bOutput          : BOOL;
bEdgeOn          : BOOL;
bEdgeOff         : BOOL;
tNextOn          : TIME;      0 .. 10080   min   (* Nächster Einschaltpunkt, max 7 Tage * 24 h * 6
0 min = 10080 *)
tNextOff         : TIME;      0 .. 1440    min   (* Nächster Ausschaltpunkt, max gleicher Tage 24
h * 60 min = 1440 *)
bInvalidParameter: BOOL;
```

bOutput: is TRUE if one of the timer switch channels has switched on.

bEdgeOn: is TRUE for one PLC cycle after bOutput switches on.


bEdgeOff: is TRUE for one PLC cycle after bOutput switches off.

tNextOn: time until the next switch-on of the timer program. Maximum value 10080 minutes (1 week).

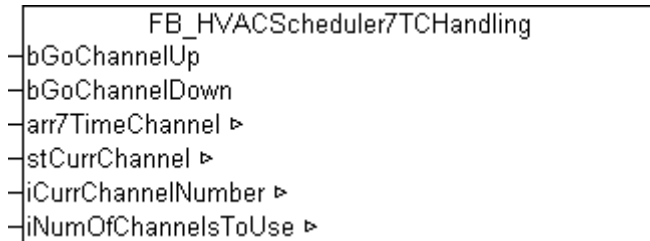
tNextOff: time until the next switch-off of the timer program. Maximum forecast until midnight.

blInvalidParameter: an error occurred during the plausibility check. It is deleted again by *bReset*.

Documents about this

 example_persistent_e.zip (Resources/zip/11659714827.zip)

3.7.3 FB_HVACScheduler7TCHandling



Application

This function block can be used to select and modify an individual line from the data array of a weekly timer. Thus, it is possible to edit the data sets very easily even with a visualization that offers only a low graphical resolution, without having to display all the data of the array.

The data set from **arr7TimeChannel** with the index **iCurrChannelNumber** is copied into data structure **stCurrChannel**. Modifications of the data in structure **stCurrChannel** take effect immediately and are transferred to array **arr7TimeChannel**.

VAR_INPUT

```

bGoChannelUp      : BOOL;
bGoChannelDown    : BOOL;
  
```

bGoChannelUp: a positive edge shifts the focus to the next higher index in the data array *arr7TimeChannel*.

bGoChannelDown: a positive edge shifts the focus to the next lower index in the data array *arr7TimeChannel*.

VAR_IN_OUT

```

arr7TimeChannel  : ARRAY[1..7] OF ST_HVACTimeChannel;
stCurrChannel    : ST_HVACTimeChannel;
iCurrChannelNumber : INT;
iNumOfChannelsToUse: INT;
  
```

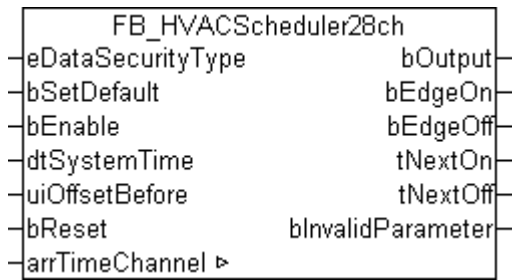
arr7TimeChannel : data set array for timer *FB_HVACScheduler7CH*.

stCurrChannel: contains the contents of the data set from the array **arr7TimeChannel**, which is selected via the variable *iCurrChannelNumber*.

iCurrChannelNumber: index for the array **arr7TimeChannel**.

iNumOfChannelsToUse : maximum permissible index for **iCurrChannelNumber** in case not all channels are needed.

3.7.4 FB_HVACScheduler28ch



Application

This function block is a weekly timer switch with 28 timer switch channels. Any desired combination of days of the week can be assigned to any timer switch channel. The timer switch channels can be defined for non-recurring or recurring events. Besides the switching signal, the switch-on and switch-off edges plus the next switch-on or switch-off time as a countdown are available as outputs. The data of the timer switch channels is transferred as a reference. The function block is therefore able to check the data for changes and to save it as persistent data. After booting, the data are thus available to the BMS again. The switch-on and switch-off points can be specified to the exact second via the variables *tiOn_ss* and *tiOff_ss* of the respective time channel.




If the input variable *bEnable* = FALSE, then the output *bOutput* = FALSE. The timer switch channels *arrTimeChannel* remain active.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
dtSystemTime      : DT;
uiOffsetBefore    : UINT;           0 .. 240   min
bReset            : BOOL;
```

eDataSecurityType: if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If *eDataSecurityType* := *eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled with *bEnable*. If *bEnable* = FALSE there is no switching at the output *bOutput*.

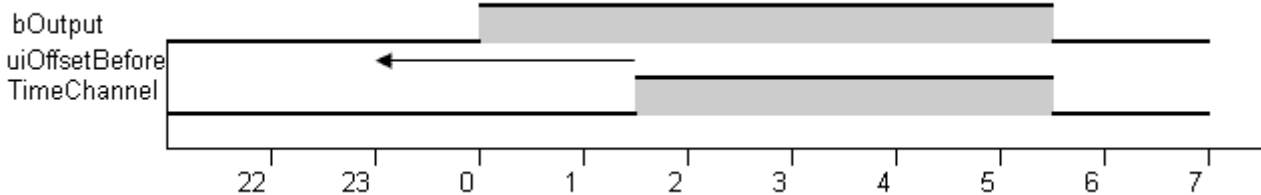
dtSystemTime: the variable *dtSystemTime* transfers the computer system time to the timer program.

uiOffsetBefore: the switch-on time is brought forward by this number of minutes. This function is useful for the time-optimized switching on of heating circuits. A maximum of 240 min, i.e. 4 hours, may be entered. If the resulting switch-on time is before midnight on the previous day, then midnight will be chosen as the switch-on time. The switch-off time is not affected.

bReset: acknowledge input in the event of a fault. Resets the flag *blInvalidParameter*.

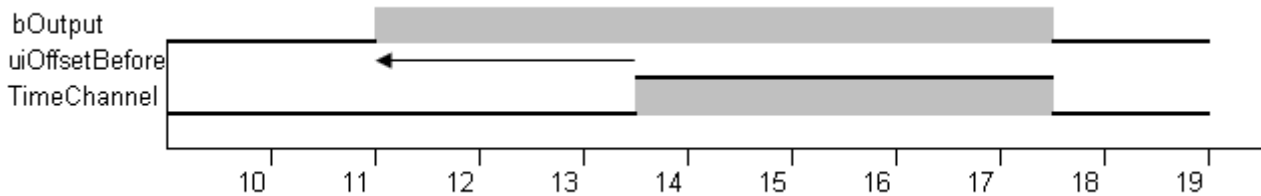
Example 1

Switch-on time from 1:30 to 5:30, uiOffsetBefore has the value 150 min. The switch-on time is only advanced by 90 min to 00:00.



Example 2

Switch-on period from 13:30 to 17:30; uiOffsetBefore has the value 150 min. The switch-on time is brought forward by 150 minutes to 11:00.



VAR_IN_OUT

```
arrTimeChannel : ARRAY[1..28] OF ST_HVACTimeChannel;
```

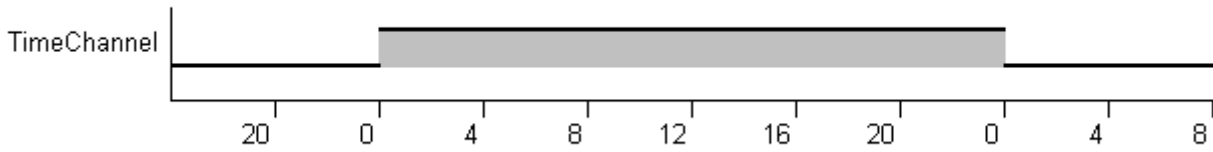
arrTimeChannel: the variable **arrTimeChannel** is an array with 28 declarations of the structure variable **ST_HVACTimeChannel**. It is possible to generate up to 3 switch-on times per day of the week with the 28 channels.

```
TYPE ST_HVACTimeChannel :
STRUCT
  uiOn_hh      : UINT;
  uiOn_mm      : UINT;
  uiOn_ss      : UINT;
  uiOff_hh     : UINT;
  uiOff_mm     : UINT;
  uiOff_ss     : UINT;
  bEnable      : BOOL;
  bAllDays     : BOOL;
  bMonday      : BOOL;
  bTuesday     : BOOL;
  bWednesday   : BOOL;
  bThursday    : BOOL;
  bFriday      : BOOL;
  bSaturday    : BOOL;
  bSunday      : BOOL;
  bResetAfterOn: BOOL;
  bQ           : BOOL;
END_STRUCT
END_TYPE
```

Special notes regarding time input

The following applies:

- Switch-on time 00:00:00: switch-on begins at 0:00 on the selected day.
- Switch-off time 00:00:00: switch-off takes place at 23:59:59 on the selected day.



The following applies:

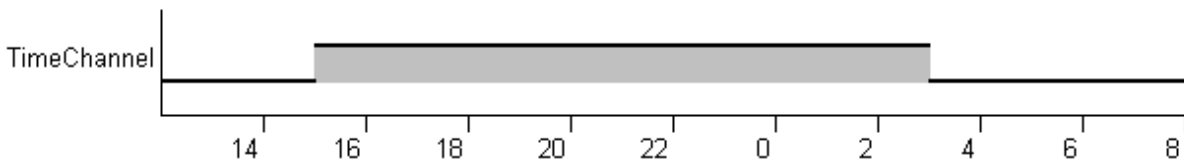
If the switch-on time is later than the switch-off time, the switch-on period is extended through midnight into the next day, regardless of whether the next day is selected.

Enable Reset after On

| all Days | Mo Di Mi Do Fr Sa So OnTime hh:mm:ss OffTime hh:mm:ss

15 : 0 : 0 => 3 : 0 : 0

This setting produces:



For each instance of the function block **FB_HVACScheduler28ch** it is necessary to create an array of type **ST_HVACTimeChannel**. The size of the array must be 28, since the function block processes a total of 28 weekly timer switch cycles and expects these as a reference (VAR_IN_OUT).

Example

```
VAR_GLOBAL
  arrTimeChannel : ARRAY[1..28] OF ST_HVACTimeChannel;
END_VAR
```

Within the weekly planner each timer switch channel is enabled via the variable *bEnable* (On/Off). If *bEnable* = FALSE, then the timer switch channel is deactivated and the times entered have no effect. If *bAllDays* (Mon-Sun) = TRUE, then the times set in *uiOn_hh*, *uiOn_mm*, *uiOff_hh* and *uiOff_mm* apply to all days of the week from Monday to Sunday. If *bAllDays* is FALSE, then the days of the week can be selectively assigned to a timer switch channel using the variables *bMonday* to *bSunday*. For one-shot events, the variable *bResetAfterOn* (non-recurring) is activated. After the time set in this channel has elapsed, the entries created under *uiOn_hh*, *uiOn_mm*, *uiOff_hh*, *uiOff_mm* and the days of the week are retained. However, the variable *bEnable* (On/Off) is reset to FALSE. In this manner a one-shot event can be defined once and reactivated whenever it is required.

Example

Enable	Reset after On							OnTime hh:mm:ss			OffTime hh:mm:ss			
all Days	Mo	Di	Mi	Do	Fr	Sa	So	hh	mm	ss	=>	hh	mm	ss
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	7	20	0	=>	17	30	0
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	16	40	0	=>	0	0	0
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0	0	0	=>	0	0	0
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	15	0	0	=>	3	0	0
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	12	0	0	=>	18	0	0

Monday switched on from 07:20 until 17:30

Tuesday switched on from 16:40 until 00:00

Wednesday switched on from 00:00 until 23:59:59 (around the clock – 24 hours!)

Thursday switched on from 15:00 until 03:00 Friday morning, non-recurring!

Friday Timer switch channel not activated, always switched off!

VAR_OUTPUT

```

bOutput          : BOOL;
bEdgeOn          : BOOL;
bEdgeOff         : BOOL;
tNextOn          : TIME;    0 .. 10080 min    (* Nächster Einschaltpunkt, max 7 Tage * 24 h * 6
0 min = 10080 *)
tNextOff         : TIME;    0 .. 1440 min     (* Nächster Ausschaltpunkt, max gleicher Tage 24
h * 60 min = 1440 *)
bInvalidParameter: BOOL;
    
```

bOutput: is TRUE if one of the timer switch channels has switched on.

bEdgeOn: is TRUE for one PLC cycle after *bOutput* switches on.

bEdgeOff: is TRUE for one PLC cycle after *bOutput* switches off.

tNextOn: time until the next switch-on of the timer program. Maximum value 10080 minutes (1 week).

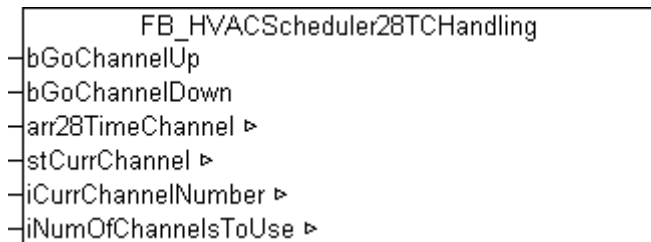
tNextOff: time until the next switch-off of the timer program. Maximum forecast until midnight.

bInvalidParameter: an error occurred during the plausibility check. It is deleted again by *bReset*.

Documents about this

 example_persistent_e.zip (Resources/zip/11659714827.zip)

3.7.5 FB_HVACScheduler28TCHandling



Application

This function block can be used to select and modify an individual line from the data array of a weekly timer. Thus, it is possible to edit the data sets very easily even with a visualization that offers only a low graphical resolution, without having to display all the data of the array.

The data set from **arr7TimeChannel** with the index **iCurrChannelNumber** is copied into data structure **stCurrChannel**. Modifications of the data in structure **stCurrChannel** take effect immediately and are transferred to array **arr28TimeChannel**.

VAR_INPUT

```
bGoChannelUp      : BOOL;
bGoChannelDown    : BOOL;
```

bGoChannelUp: a positive edge shifts the focus to the next higher index in the data array *arr28TimeChannel*.

bGoChannelDown: a positive edge shifts the focus to the next lower index in the data array *arr28TimeChannel*.

VAR_IN_OUT

```
arr28TimeChannel : ARRAY[1..28]OF ST_HVACTimeChannel;
stCurrChannel    : ST_HVACTimeChannel;
iCurrChannelNumber : INT;
iNumOfChannelsToUse : INT;
```

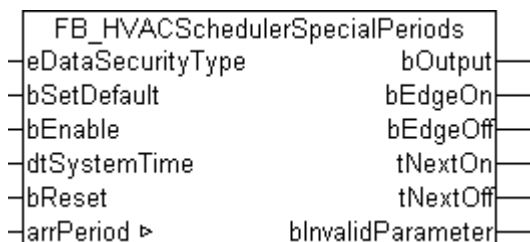
arr28TimeChannel : data set array for timer *FB_HVACScheduler28CH*.

stCurrChannel: contains the contents of the data set from the array **arr28TimeChannel**, which is selected via the variable *iCurrChannelNumber*.

iCurrChannelNumber: index for the array **arr28TimeChannel**.

iNumOfChannelsToUse : maximum permissible index for **iCurrChannelNumber** in case not all channels are needed.

3.7.6 FB_HVACSchedulerSpecialPeriods



Application

This function block is a yearly scheduler in which, e.g., school holidays or factory shutdowns can be entered. The function block is enabled by the input variable *bEnable*. The input *dtSystemTime* is linked to the current system time. The output *bOutput* is set if the timer switch conditions are satisfied. Besides the switching signal, the switch-on and switch-off edges plus the next switch-on or switch-off time as a countdown are available as outputs. The data of the timer switch channels is transferred as a reference. The function block is therefore able to check the data for changes and to save it as persistent data. After booting, the data are thus available to the BMS again. The switch-on and switch-off times on the respective day can be entered to the exact minute.




If the input variable *bEnable* = FALSE, then the output *bOutput* = FALSE. The timer switch channels *arrPeriod* remain active.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
dtSystemTime      : DT;
bReset            : BOOL;
```

eDataSecurityType: if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If *eDataSecurityType* := *eHVACDataSecurityType_Idle* the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if *eDataSecurityType* := *eHVACDataSecurityType_Persistent*. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled with *bEnable*. If *bEnable* = FALSE there is no switching at the output *bOutput*.

dtSystemTime: the variable *dtSystemTime* transfers the computer system time to the timer program.

bReset: acknowledge input in the event of a fault. Resets the flag *bInvalidParameter*.

VAR_IN_OUT

```
arrPeriod          : ARRAY[1..20] OF ST_HVACPeriod;
```

arrPeriod: the variable *arrPeriod* is an array with 20 declarations of the structure variable **ST_HVACPeriod**.

For each instance of the function block **FB_HVACSchedulerSpecialPeriods** it is necessary to create an array of type **ST_HVACPeriod**. The size of the array must be 20, since the function block processes a total of 20 time channels and expects these as a reference (VAR_IN_OUT).

```
TYPE ST_HVACTimeChannel :
STRUCT
  uiOn_hh   : UINT;
  uiOn_mm   : UINT;
```

```

uiOn_ss : UINT;
uiOff_hh : UINT;
uiOff_mm : UINT;
uiOff_ss : UINT;
bEnable : BOOL;
bAllDays : BOOL;
bMonday : BOOL;
bTuesday : BOOL;
bWednesday : BOOL;
bThursday : BOOL;
bFriday : BOOL;
bSaturday : BOOL;
bSunday : BOOL;
bResetAfterOn: BOOL;
bQ : BOOL;
    
```

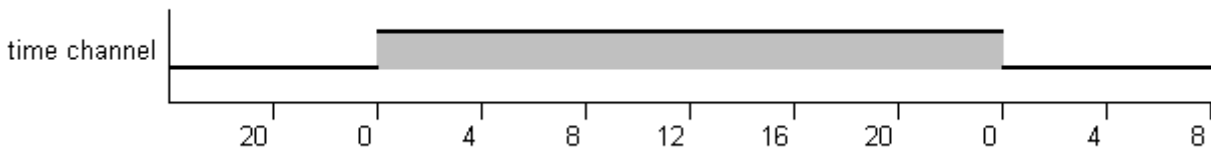
END_STRUCT

END_TYPE Special notes regarding time input

The following applies:

Switch-on time 00:00:00: switch-on begins at 0:00 on the selected day.

Switch-off time 00:00:00: switch-off takes place at 23:59:59 on the selected day.



Each timer switch channel is enabled via the variable *bEnable* (On/Off). If *bEnable* = FALSE, then the timer switch channel is deactivated and the times entered have no effect. For one-shot events, the variable *bResetAfterOn* (non-recurring) is activated. After the time set in this channel has elapsed, the entries created under *uiOn_hh*, *uiOn_mm*, *uiOff_hh*, *uiOff_mm*, *uiOn_Day*, *uiOff_Day*, *uiOn_Month* and *uiOff_Month* are retained. However, the variable *bEnable* (On/Off) is reset to FALSE. In this manner a one-shot event can be defined once and reactivated whenever it is required.

Example:


Reset after On		ON				OFF				
Enable		Day	Month	hh	mm	=>	Day	Month	hh	mm
<input checked="" type="checkbox"/>	<input type="checkbox"/>	12	1	12	0	=>	14	1	12	0
<input checked="" type="checkbox"/>	<input type="checkbox"/>	23	6	0	0	=>	25	8	0	0
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	3	10	12	0	=>	3	10	18	0
<input type="checkbox"/>	<input type="checkbox"/>	1	1	0	0	=>	1	1	0	0

switched on from 12/01 12:20 until 14/01 at 12:00

switched on from 23/06 00:00 until 25/08 23:59:59

switched on from 03/10 12:00 until 03/10 at 18:00, non-recurring!

1.1. to 1.1. Timer switch channel not activated, always switched off!

Visualization template for the Target VISU with one timer switch channel:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659729035/.zip>

VAR_OUTPUT

```
bOutput          : BOOL;
bEdgeOn         : BOOL;
bEdgeOff        : BOOL;
tNextOn         : TIME;      0 .. t#71582m47s295msmin (* Nächster Einschaltpunkt, Range max. =
Obergrenze von Time ca.50 Tage *)
tNextOff        : TIME;      0 .. t#71582m47s295msmin (* Nächster Ausschaltpunkt, Range max. =
Obergrenze von Time ca.50 Tage *)
bInvalidParameter: BOOL;
```

bOutput: is TRUE if one of the timer switch channels has switched on.

bEdgeOn: is TRUE for one PLC cycle after bOutput switches on.

bEdgeOff: is TRUE for one PLC cycle after bOutput switches off.

tNextOn: time until the next switch-on of the timer program.

tNextOff: time until the next switch-off of the timer program.

blInvalidParameter: an error occurred during the plausibility check. It is deleted again by *bReset*.

Documents about this

 [example_persistent_e.zip \(Resources/zip/11659714827.zip\)](#)

3.7.7 FB_HVACSchedulerPublicHolidays



Application

This function block is a yearly scheduler in which, e.g., school holidays or factory shutdowns can be entered. The function block is enabled by the input variable *bEnable*. The input *dtSystemTime* is linked to the current system time. The output *bOutput* is set if the timer switch conditions are satisfied. Besides the switching signal, the switch-on and switch-off edges plus the next switch-on or switch-off time as a countdown are available as outputs. The data of the timer switch channels is transferred as a reference. The function block is therefore able to check the data for changes and to save it as persistent data. After booting, the data are thus available to the BMS again. As opposed to the **FB_HVACSchedulerSpecialPeriods**, only daily switching is possible here.




If the input variable *bEnable* = FALSE, then the output *bOutput* = FALSE. The timer switch channels *arrHoliday* remain active.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
dtSystemTime      : DT;
bReset            : BOOL;
```

eDataSecurityType: if `eDataSecurityType:= eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted, the saved data are automatically read back from the flash into the RAM.

Application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659716235/.zip>

If `eDataSecurityType:= eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTICE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType:= eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

bSetDefault: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

bEnable: the function block is enabled with `bEnable`. If `bEnable = FALSE` there is no switching at the output `bOutput`.

dtSystemTime: the variable `dtSystemTime` transfers the computer system time to the timer program.

bReset: acknowledge input in the event of a fault. Resets the flag `bInvalidParameter`.

VAR_IN_OUT

```
arrHoliday : ARRAY[1..20] OF ST_HVACHoliday;
```

arrHoliday: the variable `arrHoliday` is an array with 20 declarations of the structure variable `ST_HVACHoliday`.

For each instance of the function block `FB_HVACSchedulerPublicHolidays` it is necessary to create an array of type `ST_HVACHoliday`. The size of the array must be 20, since the function block processes a total of 20 time channels and expects these as a reference (VAR_IN_OUT).

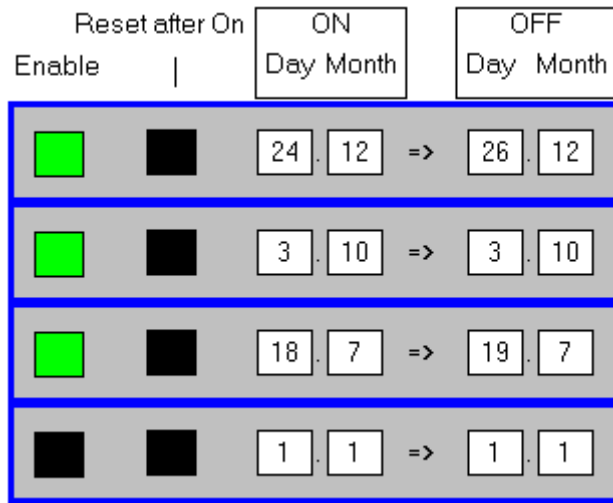
TYPE `ST_HVACTimeChannel` :

```
STRUCT
  uiOn_hh   : UINT;
  uiOn_mm   : UINT;
  uiOn_ss   : UINT;
  uiOff_hh  : UINT;
  uiOff_mm  : UINT;
  uiOff_ss  : UINT;
  bEnable   : BOOL;
  bAllDays  : BOOL;
  bMonday   : BOOL;
  bTuesday  : BOOL;
  bWednesday : BOOL;
  bThursday : BOOL;
  bFriday   : BOOL;
  bSaturday : BOOL;
  bSunday   : BOOL;
  bResetAfterOn: BOOL;
```

```
bQ      : BOOL;
END_STRUCT
END_TYPE
```

Each timer switch channel is enabled via the variable *bEnable* (On/Off). If *bEnable* = FALSE, then the timer switch channel is deactivated and the times entered have no effect. For one-shot events, the variable *bResetAfterOn* (non-recurring) is activated. After the time set in this channel has elapsed, the entries created under *uiOn_Day*, *uiOff_Day*, *uiOn_Month* and *uiOff_Month* are retained. However, the variable *bEnable* (On/Off) is reset to FALSE. In this manner a one-shot event can be defined once and reactivated whenever it is required.

Example




switched on from 24/12 00:00 until 26/12 23:59:59

switched on from 03/10 00:00 until 03/10 23:59:59

switched on from 18/07 00:00 until 19/07 23:59:59

1.1. to 1.1. Timer switch channel not activated, always switched off!

Visualization template for the Target VISU with one timer switch channel:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659730443/.zip>

VAR_OUTPUT

```
bOutput      : BOOL;
bEdgeOn     : BOOL;
bEdgeOff    : BOOL;
tNextOn     : TIME;      0 .. t#71582m47s295msmin    (* Nächster Einschaltpunkt, Range max.
= Obergrenze von Time ca.50 Tage *)
tNextOff    : TIME;      0 .. t#71582m47s295msmin    (* Nächster Ausschaltpunkt, Range max.
= Obergrenze von Time ca.50 Tage *)
bInvalidParameter: BOOL;
```

bOutput: is TRUE if one of the timer switch channels has switched on.

bEdgeOn: is TRUE for one PLC cycle after *bOutput* switches on.

bEdgeOff: is TRUE for one PLC cycle after *bOutput* switches off.

tNextOn: time until the next switch-on of the timer program.

tNextOff: time until the next switch-off of the timer program.

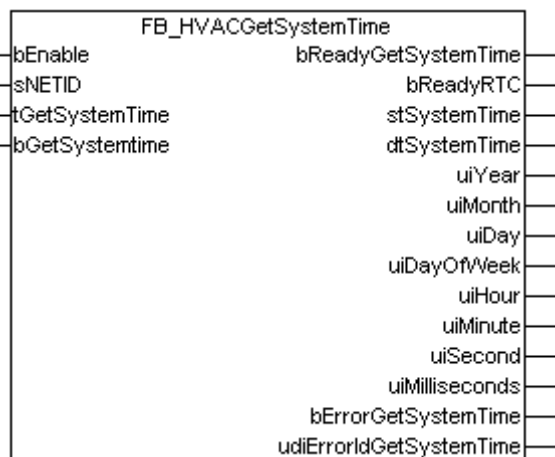
bInvalidParameter: an error occurred during the plausibility check. It is deleted again by *bReset*.

Documents about this

 example_persistent_e.zip (Resources/zip/11659714827.zip)

3.8 HVAC System

3.8.1 FB_HVACGetSystemTime



Application

With this function block an internal clock (Real Time Clock RTC) can be implemented in the TwinCAT PLC. When the function block is enabled via `bEnable`, the RTC clock is initialized with the current NT system time. One system cycle of the CPU is used to calculate the current RTC time. The function block must be called once per PLC cycle in order for the current time to be calculated. Internally an instance of the function blocks **NT_GetTime** and **RTC_EX2** is called in the function block from the Utilities library.



Note Due to system characteristics the RTC time deviates from a reference time. The difference depends on the PLC's cycle time, the value of the basic system ticks, and on the hardware being used. The RTC is therefore synchronized internally in the function block with the NT system time depending on the input variables `sNETID` and `tGetSystemTime`. The local NT system time can, in turn, be synchronized with a reference time with the aid of the SNTP protocol. More information can be found in the Beckhoff Information System under: Beckhoff Information System > Embedded PC > Operating systems > CE > SNTP: Simple Network Time Protocol

Application example

Download	Required library
TcHVAC.pro [► 531]	TcHVAC.lib

VAR_INPUT

```
bEnable      : BOOL;
sNETID       : T_AmsNetId;
tGetSystemTime: TIME;
bGetSystemTime: BOOL;
```

bEnable: enable of the function block. If `bEnable` = TRUE, then the RTC clock is initialized with the NT system time.

sNETID: this parameter can be used to specify the AmsNetID of the TwinCAT computer, whose NT system time is to be determined. If it is to be run on the local computer, an empty string can be entered.

tGetSystemTime: time specification with which the RTC clock is regularly synchronized with the NT system time. This time specification must be greater than or equal to 5 seconds, otherwise the RTC clock will not be synchronized. Parallel to the time specification `tGetSystemTime`, the RTC clock can be synchronized via the input variable `bGetSystemTime`.

bGetSystemTime: the RTC clock is synchronized with the NT system time with a rising edge at this input. This takes place in parallel with the time specification *tGetSystemTime*.

VAR_OUTPUT

```

bReadyGetSystemTime    : BOOL;
bReadyRTC              : BOOL;
stSystemTime           : TIMESTRUCT;
dtSystemTime           : DT;
uiYear                 : UINT;
uiMonth                : UINT;
uiDay                  : UINT;
uiDayOfWeek            : UINT;
uiHour                 : UINT;
uiMinute               : UINT;
uiSecond               : UINT;
uiMilliseconds         : UINT;
bErrorGetSystemTime    : BOOL;
udiErrorIdGetSystemTime : UDINT;
    
```

bReadyGetSystemTime: the function block was successfully initialized or synchronized with the NT system time.

bReadyRTC: this output is set if the function block has been initialized at least once. If this output is set, then the values for date, time and milliseconds at the outputs are valid.

stSystemTime: structure with the current RTC time.

dtSystemTime: date and time of day of the RTC time.

uiYear: the year: 1970 ~ 2106;

uiMonth: the month: 1 ~ 12 (January = 1, February = 2, etc.);

uiDay: the day of the month: 1 ~ 31;

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

uiHour: hour: 0 ~ 23;

uiMinute: minute: 0 ~ 59;

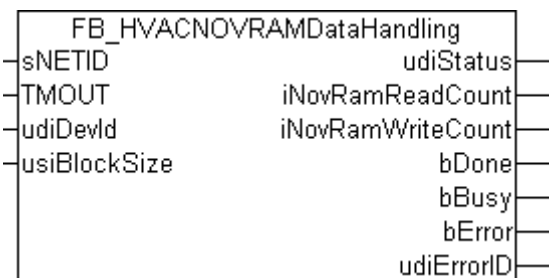
uiSecond: second: 0 ~ 59;

uiMilliseconds: millisecond: 0 ~ 999;

bErrorGetSystemTime: this output is set if an ADS error occurs when transmitting the command. The output indicates with a TRUE that an error has occurred during initialization or synchronization with the NT system time. The function block continues to attempt to initialize or synchronize the RTC clock until the error has been rectified. The RTC clock starts with an incorrect date and time specification and must therefore be synchronized with the NT system time.

udiErrorIdGetSystemTime: returns the ADS error number when the *bErrorGetSystemTime* output is set.

3.8.2 FB_HVACNOVRAMDataHandling



Application

PLC variables are written fail-safe to NOVRAM with this function block. Following a power failure or a restart of the controller, the NOVRAM will be read out completely.

The behavior of the function block looks as follows if everything was correct at startup:


```
udiStatus:= 1;
iNovRamReadCount:= 1;
iNovRamWriteCount:= 0;
bDone:= TRUE
```



Note In the case of ARM-based runtime systems, please make sure that the memory addresses of the PLC variables are divisible by 4!

Comments

The globally declared variables `g_dwHVACVarConfigStart` and `g_dwHVACVarConfigEnd` are allocated to the start address and end address of the memory area. The user must take care to ensure that there are no memory overlaps. The block size is then determined internally from the two addresses. The entire flag area is written automatically to NOVRAM in the event of changes to the flag variables.

see also application example:  <https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659731851/.zip>

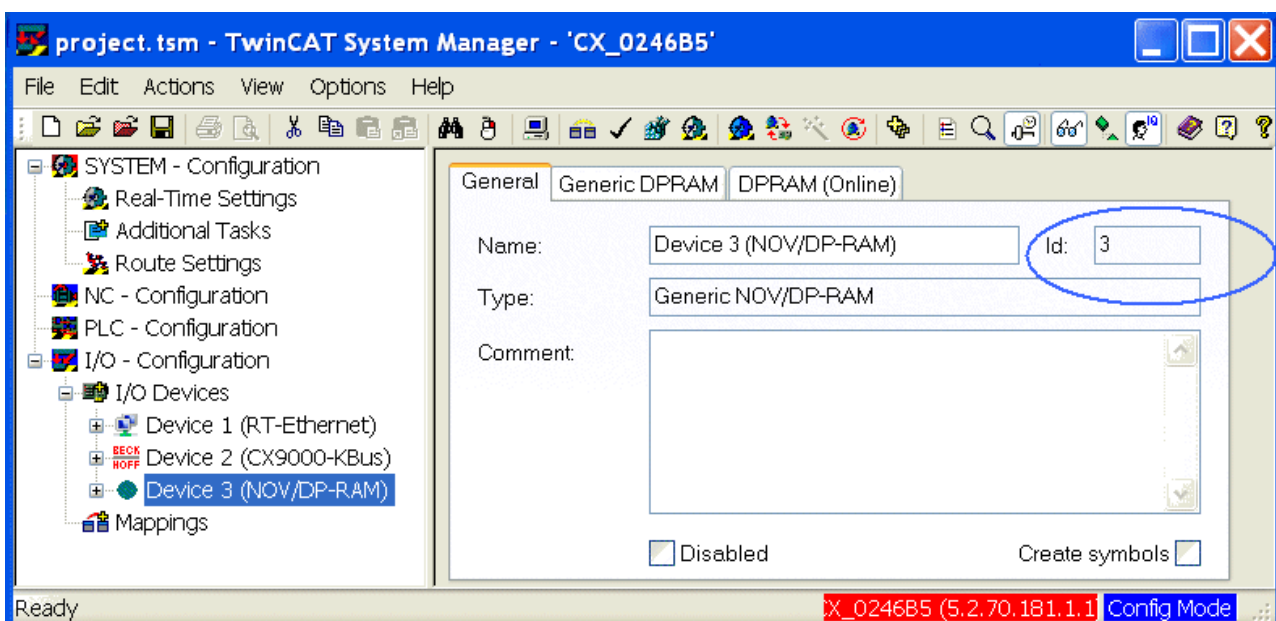
VAR_INPUT

```
sNETID      : T_AmsNetId;
TMOUT       : TIME;
udiDevID    : UDINT;
usiBlockSize : USINT;
```

sNETID : AmsNetId of the TwinCAT computer on which the function should be executed. If it is to be run on the local computer, an empty string can be entered.

TMOUT : states the length of the timeout that may not be exceeded by execution of the ADS command.

udiDevID : the Device ID specifies the NOVRAM of the CX90xx or CX10xx to be accessed with the function block for reading or writing. The device IDs are specified by the TwinCAT System Manager during hardware configuration.



usiBlockSize : the block size to be accessed per read/write cycle is given in percent; e.g. 20, i.e 5 read/write cycles will be required in order to access the entire flag area.

VAR_OUTPUT

```
udiStatus      : UDINT;  
iNovramReadCount : INT;  
iNovramWriteCount : INT;  
bDone         : BOOL;  
bBusy        : BOOL;  
bError       : BOOL;  
udiErrorID   : UDINT;
```

udiStatus : = 0 , no status

= 1, valid data at last reading of NOVRAM

= 2, invalid data at last reading of NOVRAM. Data were discarded.

iNovramReadCount : counter that is incremented by 1 when NOVRAM is read from.

iNovramWriteCount: counter that is incremented by 1 when NOVRAM is written to.

bDone: will be set to TRUE if the function block is executed.

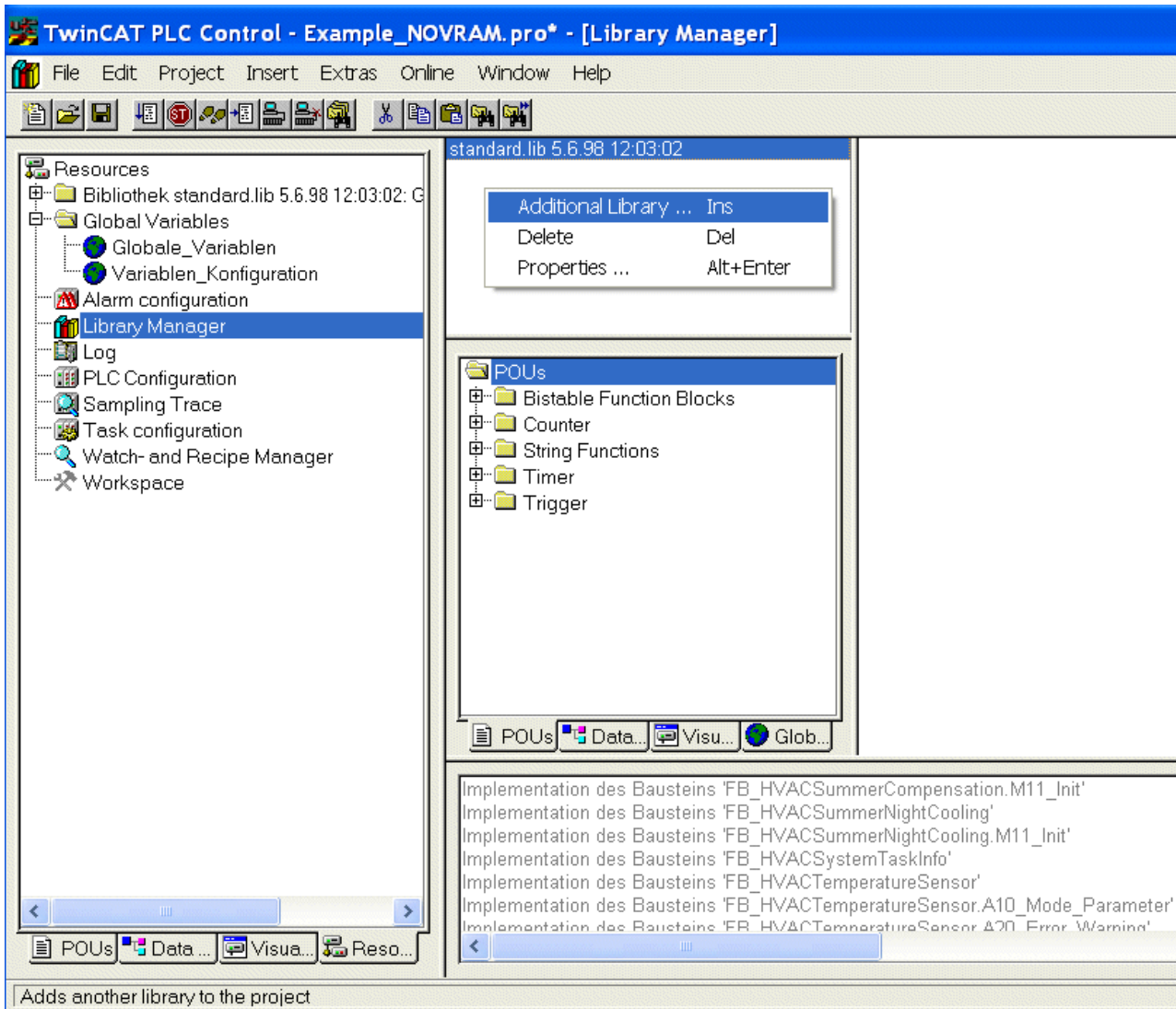
bBusy: when the function block is activated the output is set, and it remains active until execution of the command has been completed.

bError: this output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorID*.

udiErrorID: contains the command-specific error code. Please see [ADS Return Codes](#).

Example of handling inside TwinCAT PLC**1st step**

- Creating a new project
- Insertion of TcHVAC.lib; this takes place via the tab Resources in object Organizer



2nd step

- Translate project; two warnings are received, since two incompletely defined addresses exist in TcHVAC.lib. The user must define the start and end addresses of the memory area via these two addresses.

3rd step

- The menu command **All Instance Paths** in the **Insert** menu can be used via the tab Resources in Object Organizer under Variables_Configuration. The user can define the start and end addresses after performing this step.

4th step

- Define the addresses in Variables_Configuration under the Global Variables.



On ARM-based platforms (e.g. CX90xx) care must be taken that the addresses of the allocated variables are divisible by 4.

5th step

- In this example 0 was defined as the start address and 56 as the end address.

6th step

- The project must be translated again after performing steps 1 to 5.



- For reasons of performance it is useful to address the allocated variables (VAR_CONFIG Object) without gaps.
- If the variable configuration (VAR_CONFIG) is subsequently changed, it is compulsory to load the changes to the target system via an « Online Change » and not via « Load All ». Only an « Online Change » ensures that the NOVRAM data is retained correctly. The NOVRAM data would be recognized as invalid in the case of « Load All ».

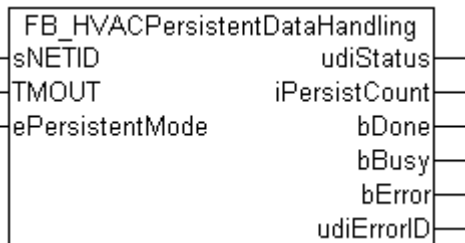
Sample for declaration of allocated variables:

```

Heizkurve.fbRealMinTemp_NOVRAM.rVar_N      AT %MB12 : REAL;
Heizkurve.fbRealMaxTemp_NOVRAM.rVar_N      AT %MB16 : REAL;
Heizkurve.fbRealNightSetback_NOVRAM.rVar_N  AT %MB20 : REAL;
Heizkurve.fbRealSetpoint_Y1_NOVRAM.rVar_N   AT %MB24 : REAL;
Heizkurve.fbRealSetpoint_Y2_NOVRAM.rVar_N   AT %MB28 : REAL;
Heizkurve.fbRealSetpoint_Y3_NOVRAM.rVar_N   AT %MB32 : REAL;
Heizkurve.fbRealSetpoint_Y4_NOVRAM.rVar_N   AT %MB36 : REAL;
Heizkurve.fbRealOutsideTemp_X1_NOVRAM.rVar_N AT %MB40 : REAL;
Heizkurve.fbRealOutsideTemp_X2_NOVRAM.rVar_N AT %MB44 : REAL;
Heizkurve.fbRealOutsideTemp_X3_NOVRAM.rVar_N AT %MB48 : REAL;
Heizkurve.fbRealOutsideTemp_X4_NOVRAM.rVar_N AT %MB52 : REAL;

.g_dwHVACVarConfigStart                    AT %MB0  : DWORD;
.g_dwHVACVarConfigEnd                      AT %MB56 : DWORD;
    
```

3.8.3 FB_HVACPersistentDataHandling




Application

With this function block, all PLC variables that are declared as persistent (VAR PERSISTENT) are written fail-safe to a file. The PLC variables declared as persistent within the function blocks from the TcHVAC.lib are written to this file in case of a change. This file is saved under ..\TwinCAT\Boot directory and is named as follows: TCPLC_T_x.wbp (x = number of the runtime system). So that the new status is not saved immediately after a change of the persistent variables, a timeframe of 5 sec is built into the standard setting. This timeframe can be changed by the user. He must assign the new time to the globally-declared constant variable **g_tHVACWriteBackupDataTime : TIME := t#5s**.

The new status is only saved after the timeframe has elapsed. In the worst case this can mean that changes made within the last timeframe are not saved if a power failure occurs during that time.

Each time the controller is restarted, the status of the persistent data is read from the file. The output variable *udiStatus* indicates whether or not the data is valid.

In order to secure the persistent variables inside the function block voltage failure safe, it is necessary to start an instance of the FB_HVACPersistentDataHandling block in the MAIN program. The handling of this function is described in more detail on the basis of the following application example.

see also application example:  <https://infosys.beckhoff.com/content/1033/tcplclibvac/Resources/11659716235/.zip>

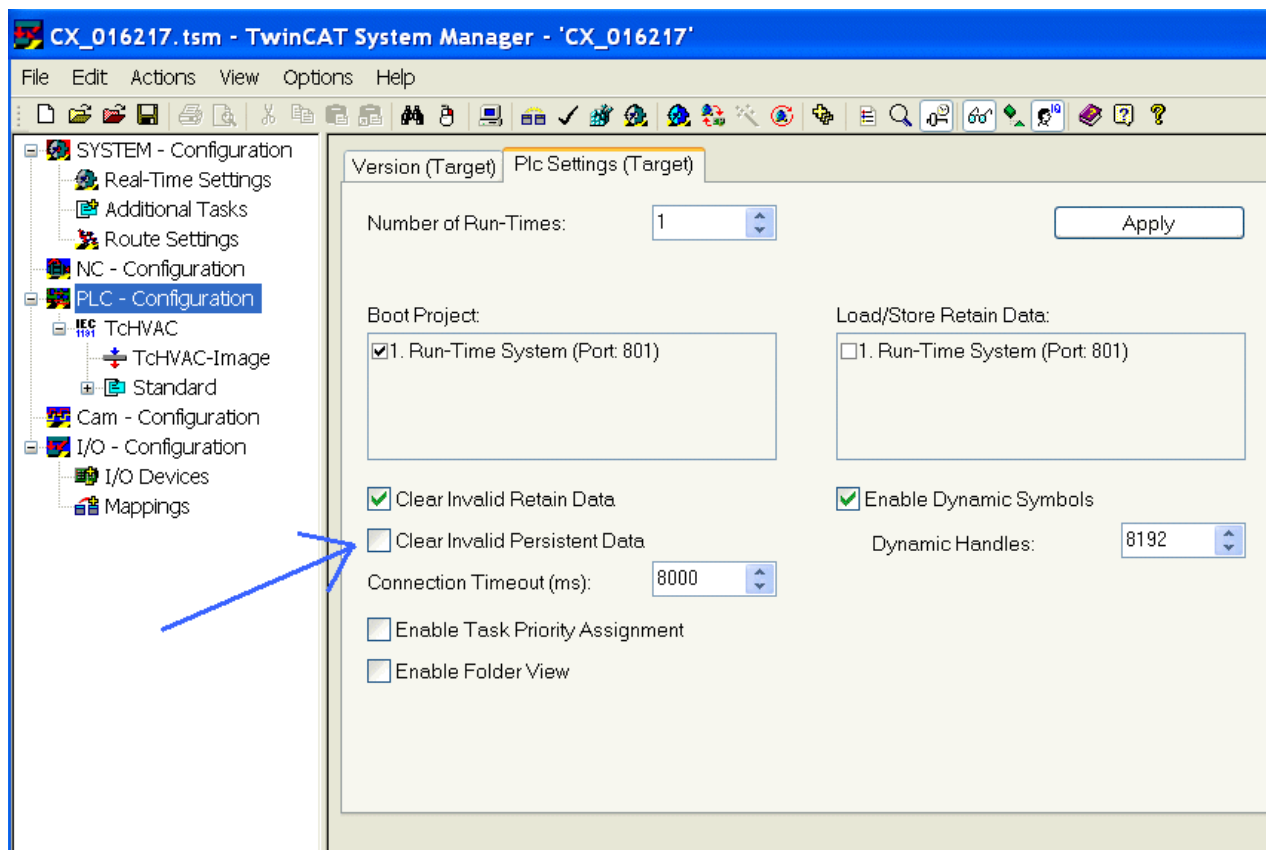
The behavior of the function block looks as follows if everything was correct at startup:
udiStatus:= 1;
iPersistCount:= 1;
bDone:= FALSE

If the result of the query of `usiStatus=1` this means that the persistent variables have been read from memory. `bDone` becomes TRUE only when the persistent variables have been written successfully.

The following setting must be made in the TwinCAT System Manager:

Under PLC Configuration in the PLC Settings (Target System) tab

the option *Delete invalid persistent data* must be **disabled!!** The deactivation must be performed using the **Apply** button.



VAR_INPUT

```
sNETID      : T_AmsNetId;
TMOUT       : TIME;
ePersistentMode : E_PersistentMode;
```

sNETID : AmsNetId of the TwinCAT computer on which the function should be executed. If it is to be run on the local computer, an empty string can be entered.

TMOUT : states the length of the timeout that may not be exceeded by execution of the ADS command.

ePersistentMode: mode in which the persistent data should be written. See also [E_PersistentMode](#).

VAR_OUTPUT

```
udiStatus    : UDINT;
iPersistCount : INT;
bDone        : BOOL;
bBusy        : BOOL;
bError       : BOOL;
udiErrorID   : UDINT;
```

udiStatus : 0 = No status

1 = Persistent data read successfully

2 = Backup from persistent data read

3 = Neither persistent nor backup data read

iPersistCount: counter that is incremented by 1 after successful writing.

bDone: will be set to TRUE if the function block is executed.

bBusy: when the function block is activated the output is set, and it remains active until execution of the command has been completed.

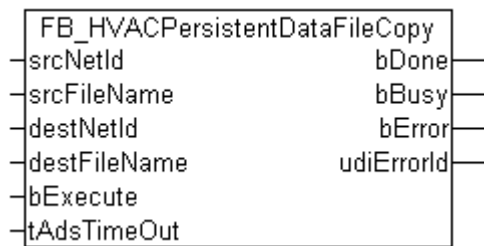
bError: this output is set to TRUE if an error occurs during the execution of a command.

udiErrorID: contains the command-specific error code. Please see ADS Return Codes.

Documents about this

 [example_persistent_e.zip](#) (Resources/zip/11659714827.zip)

3.8.4 FB_HVACPersistentDataFileCopy



Application

This function block can be used, for instance, to copy binary files on the local TwinCAT PC or from a remote TwinCAT PC to the local TwinCAT PC. The function block cannot be used to access network drives. The following steps are executed upon a rising edge at the input *bExecute*:

- a) Open the source and destination files;
- b) Read the source file into a buffer;
- c) Write the bytes that have been read from the buffer into the destination file;
- d) Check whether the end of the source file has been reached. If not, then repeat b) and c). If yes, then jump to e);
- e) Close the source and destination files;

The file is copied one segment at a time. In this function block, the size of the buffer has been specified as 100 bytes, but this can be modified.

Application example

Download	Required library
TcHVAC.pro [► 531]	TcHVAC.lib

VAR_INPUT

```
srcNETID    : T_AmsNetId;
srcFileName : T_MaxString;
destNETID   : T_AmsNetId;
destFileName: T_MaxString;
bExecute    : BOOL;
tAdsTimeOut : TIME;
```

srcNETID: AmsNetId of the TwinCAT computer on which the function should be executed. If it is to be run on the local computer, an empty string can be entered.

srcFileName: contains the path and file name of the file to be opened.

The path can only point to the local computer's file system! This means that network paths cannot be used here!

destNETID: AmsNetId of the TwinCAT computer on which the file should be copied.

destFileName: contains the path and file name of the destination file. Note: the path can only point to the local computer's file system! This means that network paths cannot be used here!

bExecute : the steps listed above are executed upon a rising edge at the input *bExecute*.

tAdsTimeOut : states the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

```
bDone          : BOOL;
bBusy          : BOOL;
bError         : BOOL;
udiErrorID    : UDINT;
```

bDone: will be set to TRUE if the function block is executed.

bBusy: when the function block is activated the output is set, and it remains active until execution of the command has been completed.

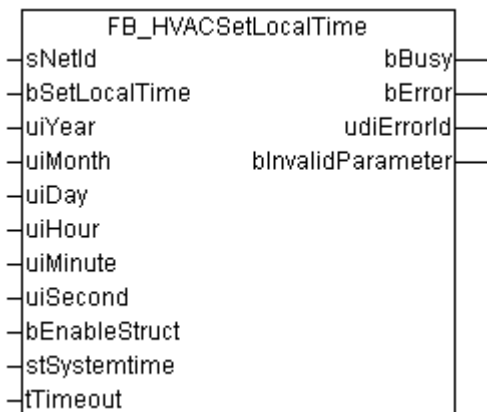
bError: this output is switched to TRUE if an error occurs during the execution of a command.

udiErrorID: contains the command-specific error code. Please see [ADS Return Codes](#).

Documents about this

📄 tchvac.zip (Resources/zip/11659726219.zip)

3.8.5 FB_HVACSetLocalTime



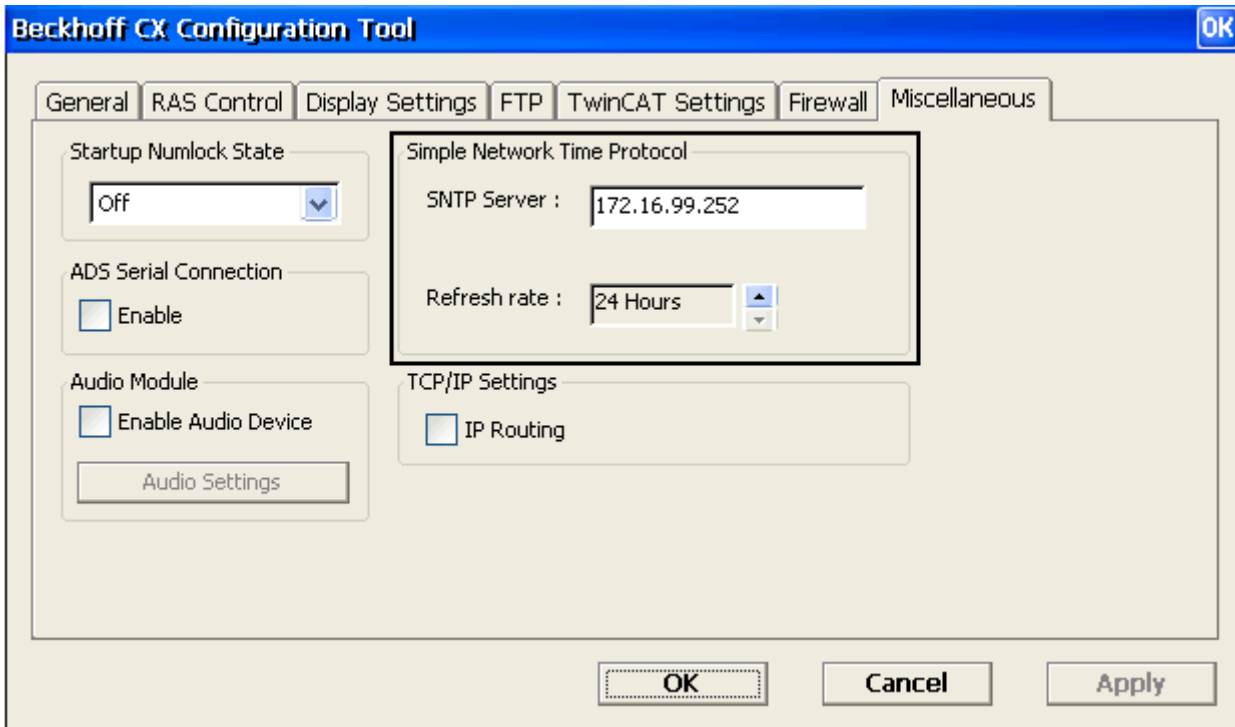
Application

The local NT system time and the date of a TwinCAT system can be set with the function block *FB_HVACSetLocalTime* (the local NT system time is shown in the taskbar). The system time can be specified either via the individual variables *uiYear*, *uiMonth*, *uiDay*, *uiHour*, *uiMinute* and *uiSecond* or the structure *stSystemtime*, see *bEnableStruct*.

Internally, an instance of the function block **NT_SetLocalTime** from the TcUtilities library is called in the function block.



Note The local NT system time can also be synchronized with a reference time with the aid of the SNTP protocol. More information can be found in the Beckhoff Information System under: Beckhoff Information System > Embedded PC > Operating systems > CE > SNTP: Simple Network Time Protocol



Application example

Download	Required library
TcHVAC.pro [▶ 531]	TcHVAC.lib

VAR_INPUT

```

sNetId      : T_AmsNetId;
bSetLocalTime : BOOL;
uiYear      : UINT;          1970 - 2106
uiMonth     : UINT;          1 - 12
uiDay       : UINT;          1 - 31
uiHour      : UINT;          0 - 23
uiMinute    : UINT;          0 - 59
uiSecond    : UINT;          0 - 59
bEnableStruct : BOOL;
stSystemtime : TIMESTRUCT;
tTimeout    : TIME;
    
```

sNETID: this parameter can be used to specify the AmsNetID of the TwinCAT computer, whose local NT system time is to be set. If applicable, an empty string *sNetId := ""* can be specified for the local computer.

bSetLocalTime: activation of the function block with a rising edge.

uiYear: the year: 1970 ~ 2106; ; the variable is active if *bEnableStruct = FALSE*. If specified incorrectly, *blInvalidParameter = TRUE* and the new local NT system time was not set.

uiMonth: the month: 1 ~ 12 (January = 1, February = 2 and so on); the variable is active if *bEnableStruct = FALSE*. If specified incorrectly, *blInvalidParameter = TRUE* and the new local NT system time was not set.

uiDay: day of the month: 1 ~ 31; February with 28 or 29 days and the months with 30 or 31 days are checked. The variable is active if *bEnableStruct = FALSE*. If specified incorrectly, *blInvalidParameter = TRUE* and the new local NT system time was not set.

uiHour: hour: 0 ~ 23; ; the variable is active if *bEnableStruct = FALSE*. If specified incorrectly, *blInvalidParameter = TRUE* and the new local NT system time was not set.

uiMinute: minute: 0 ~ 59; ; the variable is active if *bEnableStruct = FALSE*. If specified incorrectly, *blInvalidParameter = TRUE* and the new local NT system time was not set.

uiSecond: second: 0 ~ 59; ; the variable is active if *bEnableStruct = FALSE*. If specified incorrectly, *blInvalidParameter = TRUE* and the new local NT system time was not set.

bEnableStruct: if *bEnableStruct* = TRUE, the new local NT system time is set via the input variable *stSystemtime*. If *bEnableStruct* = FALSE, the new local NT system time is set via the input variables *uiYear*, *uiMonth*, *uiDay*, *uiHour*, *uiMinute* and *uiSecond*.

stSystemtime: structure with the new local NT system time. The structure is active if *bEnableStruct* = TRUE. The same ranges apply to the structure as to the input variables *uiYear*, *uiMonth*, *uiDay*, *uiHour*, *uiMinute* and *uiSecond*. If specified incorrectly, *bInvalidParameter* = TRUE and the new local NT system time was not set.

tTimeout: indicates the timeout time, which must not be exceeded during execution.

VAR_OUTPUT

```
bBusy          : BOOL;
bError         : BOOL;
udiErrorId     : UDINT;
bInvalidParameter: BOOL;
```

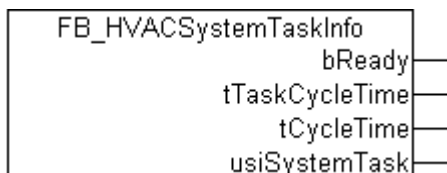
bBusy: if the function block is activated via a rising edge at *bSetLocalTime*, this output is set and remains set until feedback occurs.

bError: if an error occurs during the transfer of the NT system time, this output is set. *bError* is reset with the activation of the function block via the input variable *bSetLocalTime*.

udiErrorId: returns the [ADS error code](#) if *bError* is set.

bInvalidParameter: *bInvalidParameter* is TRUE if the ranges of the variables for the time and date were not observed: *uiYear*, *uiMonth*, *uiDay*, *uiHour*, *uiMinute* and *uiSecond*. The same ranges apply to the time structure *stSystemtime* as to the input variables. If *bInvalidParameter* = TRUE, the new local NT system time was not set. *bInvalidParameter* is reset with the activation of the function block via the input variable *bSetLocalTime*.

3.8.6 FB_HVACSystemTaskInfo



The function block determines system variables of the task with a resolution of 1 ms, in which it is currently called. If the current cycle time is less than 1 ms, the set task cycle time *tTaskCycleTime* is output on the output variable *tCycleTime*.

NOTICE

The *tTaskCycleTime* of the PLC program must not be longer than 100 ms, as otherwise the digital outputs will be deactivated.

This is because the internal K-bus of the Bus Terminals runs synchronously with the PLC program, which is no longer triggered early enough and the watchdog of the Bus Terminals becomes active.

VAR_OUTPUT

```
bReady          : BOOL;
tTaskCycleTime  : TIME;
tCycleTime      : TIME;
usiSystemTask   : USINT;
```

bReady: the variable is TRUE if the system information has been read out.

tTaskCycleTime: set task cycle time.

tCycleTime: cycle time required for the last cycle.

usiSystemTask: task index of the task.

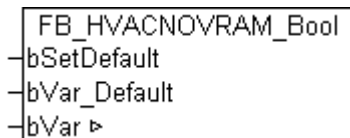
4 Backup Function blocks

4.1 BackupVar NOVRAM

The user can use standard data types (see table) and integrate them in his application in order to save defined PLC variables voltage failure safe.

Data type	BackupVar NOVRAM block
BOOL	FB HVACNOVRAM Bool [▶ 503]
BYTE	FB HVACNOVRAM Byte [▶ 503]
DINT	FB HVACNOVRAM Dint [▶ 504]
DWORD	FB HVACNOVRAM Dword [▶ 504]
INT	FB HVACNOVRAM Int [▶ 505]
LREAL	FB HVACNOVRAM Lreal [▶ 505]
REAL	FB HVACNOVRAM Real [▶ 505]
TIME	FB HVACNOVRAM Time [▶ 506]
SINT	FB HVACNOVRAM Sint [▶ 506]
UDINT	FB HVACNOVRAM Udint [▶ 506]
UINT	FB HVACNOVRAM Uint [▶ 507]
USINT	FB HVACNOVRAM Usint [▶ 507]
WORD	FB HVACNOVRAM Word [▶ 507]

4.1.1 FB_HVACNOVRAM_Bool



VAR_INPUT

```

bSetDefault      : BOOL;
bVar_Default     : BOOL;
    
```

bSetDefault: if the variable is TRUE, the value of the input variable *bVar_Default* is adopted.

bVar_Default: default value defined by the user. The variable is of the data type BOOL.

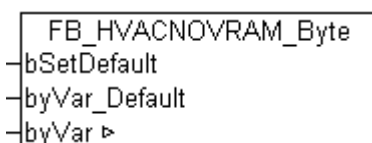
VAR_IN_OUT

```

Name             : Type
bVar             : BOOL;
    
```

bVar: variable that is programmed fail-safe by the user.

4.1.2 FB_HVACNOVRAM_Byte



VAR_INPUT

```
bSetDefault      : BOOL;
byVar_Default    : BYTE;
```

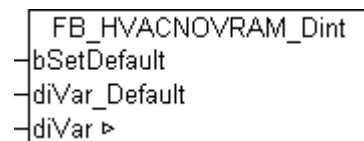
bSetDefault: if the variable is TRUE, the value of the input variable *byVar_Default* is adopted.

byVar_Default: default value defined by the user. The variable is of the data type BYTE.

VAR_IN_OUT

```
byVar            : BYTE;
```

byVar: variable that is programmed fail-safe by the user.

4.1.3 FB_HVACNOVRAM_Dint**VAR_INPUT**

```
bSetDefault      : BOOL;
diVar_Default    : DINT;
```

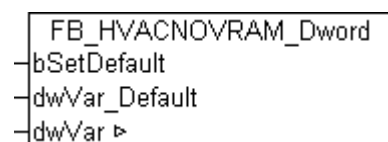
bSetDefault: if the variable is TRUE, the value of the input variable *diVar_Default* is adopted.

diVar_Default: default value defined by the user. The variable is of the data type DINT.

VAR_IN_OUT

```
diVar            : DINT;
```

diVar: variable that is programmed fail-safe by the user.

4.1.4 FB_HVACNOVRAM_Dword**VAR_INPUT**

```
bSetDefault      : BOOL;
dwVar_Default    : DWORD;
```

bSetDefault: if the variable is TRUE, the value of the input variable *dwVar_Default* is adopted.

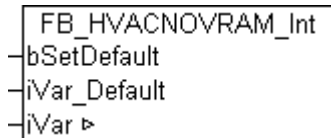
dwVar_Default: default value defined by the user. The variable is of the data type DWORD.

VAR_IN_OUT

```
dwVar            : DWORD;
```

dwVar: variable that is programmed fail-safe by the user.

4.1.5 FB_HVACNOVRAM_Int



VAR_INPUT

```
bSetDefault      : BOOL;
iVar_Default     : INT;
```

bSetDefault: if the variable is TRUE, the value of the input variable *iVar_Default* is adopted.

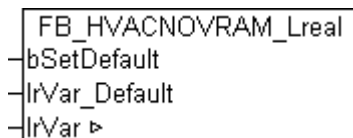
iVar_Default: default value defined by the user. The variable is of the data type INT.

VAR_IN_OUT

```
iVar             : INT;
```

iVar: variable that is programmed fail-safe by the user.

4.1.6 FB_HVACNOVRAM_Lreal



VAR_INPUT

```
bSetDefault      : BOOL;
lrVar_Default     : LREAL;
```

bSetDefault: if the variable is TRUE, the value of the input variable *lrVar_Default* is adopted.

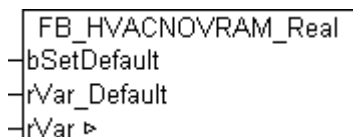
lrVar_Default: default value defined by the user. The variable is of the data type LREAL.

VAR_IN_OUT

```
lrVar            : LREAL;
```

lrVar: variable that is programmed fail-safe by the user.

4.1.7 FB_HVACNOVRAM_Real



VAR_INPUT

```
bSetDefault      : BOOL;
rVar_Default     : REAL;
```

bSetDefault: if the variable is TRUE, the value of the input variable *rVar_Default* is adopted.

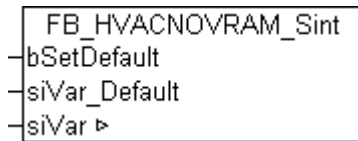
rVar_Default: default value defined by the user. The variable is of the data type REAL.

VAR_IN_OUT

```
rVar             : REAL;
```

rVar: variable that is programmed fail-safe by the user.

4.1.8 FB_HVACNOVRAM_Sint



VAR_INPUT

```
bSetDefault      : BOOL;
siVar_Default    : SINT;
```

bSetDefault: if the variable is TRUE, the value of the input variable *siVar_Default* is adopted.

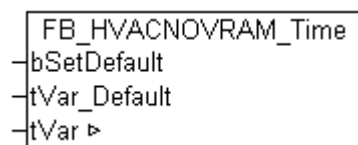
siVar_Default: default value defined by the user. The variable is of the data type SINT.

VAR_IN_OUT

```
siVar            : SINT;
```

siVar: variable that is programmed fail-safe by the user.

4.1.9 FB_HVACNOVRAM_Time



VAR_INPUT

```
bSetDefault      : BOOL;
tVar_Default     : TIME;
```

bSetDefault: if the variable is TRUE, the value of the input variable *tVar_Default* is adopted.

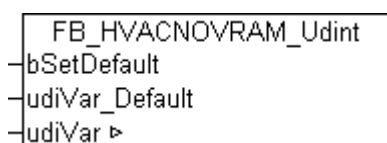
tVar_Default: default value defined by the user. The variable is of the data type TIME.

VAR_IN_OUT

```
tVar            : TIME;
```

tVar: variable that is programmed fail-safe by the user.

4.1.10 FB_HVACNOVRAM_Udint



VAR_INPUT

```
bSetDefault      : BOOL;
udiVar_Default   : UDINT;
```

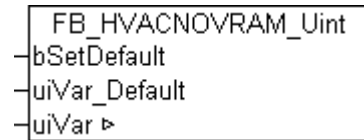
bSetDefault: if the variable is TRUE, the value of the input variable *udiVar_Default* is adopted.

udiVar_Default: default value defined by the user. The variable is of the data type UDINT.

VAR_IN_OUT

```
udiVar      : UDINT;
```

udiVar: variable that is programmed fail-safe by the user.

4.1.11 FB_HVACNOVRAM_Uint**VAR_INPUT**

```
bSetDefault      : BOOL;
uiVar_Default    : UINT;
```

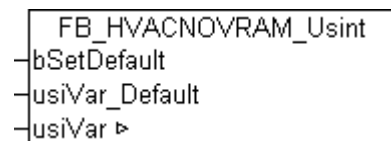
bSetDefault: if the variable is TRUE, the value of the input variable *uiVar_Default* is adopted.

uiVar_Default: default value defined by the user. The variable is of the data type UINT.

VAR_IN_OUT

```
uiVar      : UINT;
```

uiVar: variable that is programmed fail-safe by the user.

4.1.12 FB_HVACNOVRAM_Usint**VAR_INPUT**

```
bSetDefault      : BOOL;
usiVar_Default    : USINT;
```

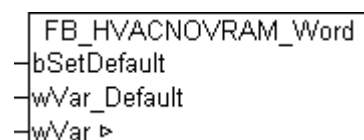
bSetDefault: if the variable is TRUE, the value of the input variable *usiVar_Default* is adopted.

usiVar_Default: default value defined by the user. The variable is of the data type USINT.

VAR_IN_OUT

```
usiVar      : USINT;
```

usiVar: variable that is programmed fail-safe by the user.

4.1.13 FB_HVACNOVRAM_Word**VAR_INPUT**

```
bSetDefault      : BOOL;
wVar_Default     : WORD;
```

bSetDefault: if the variable is TRUE, the value of the input variable *wVar_Default* is adopted.

wVar_Default: default value defined by the user. The variable is of the data type WORD.

VAR_IN_OUT

```
wVar : WORD;
```

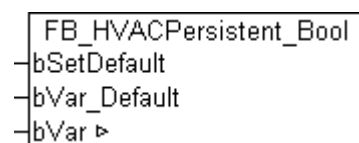
wVar: variable that is programmed fail-safe by the user.

4.2 BackupVar Persistent

The user can use standard data types (see table) and integrate them in his application in order to save defined PLC variables fail-safe.

Data type	BackupVar Persistent function block
BOOL	FB HVACPersistent Bool [▶ 508]
BYTE	FB HVACPersistent Byte [▶ 509]
DINT	FB HVACPersistent Dint [▶ 509]
DWORD	FB HVACPersistent Dword [▶ 509]
INT	FB HVACPersistent Int [▶ 510]
LREAL	FB HVACPersistent Lreal [▶ 510]
REAL	FB HVACPersistent Real [▶ 510]
SINT	FB HVACPersistent Sint [▶ 511]
STRING	FB HVACPersistent String [▶ 511]
STRUCT	FB HVACPersistent Struct [▶ 512]
TIME	FB HVACPersistent Time [▶ 513]
UDINT	FB HVACPersistent Udint [▶ 513]
UINT	FB HVACPersistent Uint [▶ 514]
USINT	FB HVACPersistent Usint [▶ 514]
WORD	FB HVACPersistent Word [▶ 514]

4.2.1 FB_HVACPersistent_Bool



VAR_INPUT

```
bSetDefault : BOOL;
bVar_Default : BOOL;
```

bSetDefault: if the variable is TRUE, the value of the input variable *bVar_Default* is adopted.

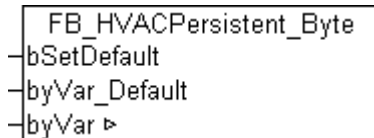
bVar_Default: default value defined by the user. The variable is of the data type BOOL.

VAR_IN_OUT

```
bVar : BOOL;
```

bVar: variable that is programmed fail-safe by the user.

4.2.2 FB_HVACPersistent_Byte



VAR_INPUT

```
bSetDefault      : BOOL;
byVar_Default    : BYTE;
```

bSetDefault: if the variable is TRUE, the value of the input variable *byVar_Default* is adopted.

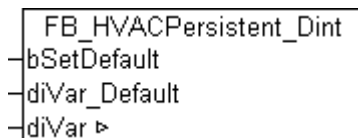
byVar_Default: default value defined by the user. The variable is of the data type BYTE.

VAR_IN_OUT

```
byVar            : BYTE;
```

byVar: variable that is programmed fail-safe by the user.

4.2.3 FB_HVACPersistent_Dint



VAR_INPUT

```
bSetDefault      : BOOL;
diVar_Default    : DINT;
```

bSetDefault: if the variable is TRUE, the value of the input variable *diVar_Default* is adopted.

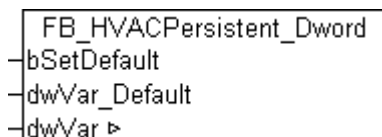
diVar_Default: default value defined by the user. The variable is of the data type DINT.

VAR_IN_OUT

```
diVar            : DINT;
```

diVar: variable that is programmed fail-safe by the user.

4.2.4 FB_HVACPersistent_Dword



VAR_INPUT

```
bSetDefault      : BOOL;
dwVar_Default    : DWORD;
```

bSetDefault: if the variable is TRUE, the value of the input variable *dwVar_Default* is adopted.

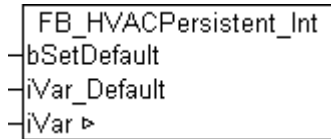
dwVar_Default: default value defined by the user. The variable is of the data type DWORD.

VAR_IN_OUT

```
dwVar            : DWORD;
```

dwVar: variable that is programmed fail-safe by the user.

4.2.5 FB_HVACPersistent_Int



VAR_INPUT

```
bSetDefault      : BOOL;
iVar_Default     : INT;
```

bSetDefault: if the variable is TRUE, the value of the input variable *iVar_Default* is adopted.

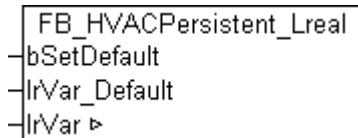
iVar_Default: default value defined by the user. The variable is of the data type INT.

VAR_IN_OUT

```
iVar             : INT;
```

iVar: variable that is programmed fail-safe by the user.

4.2.6 FB_HVACPersistent_Lreal



VAR_INPUT

```
bSetDefault      : BOOL;
lrVar_Default    : LREAL;
```

bSetDefault: if the variable is TRUE, the value of the input variable *lrVar_Default* is adopted.

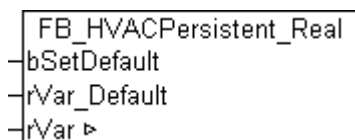
lrVar_Default: default value defined by the user. The variable is of the data type LREAL.

VAR_IN_OUT

```
lrVar            : LREAL;
```

lrVar: variable that is programmed fail-safe by the user.

4.2.7 FB_HVACPersistent_Real



VAR_INPUT

```
bSetDefault      : BOOL;
rVar_Default     : REAL;
```

bSetDefault: if the variable is TRUE, the value of the input variable *rVar_Default* is adopted.

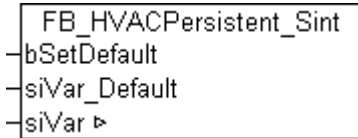
rVar_Default: default value defined by the user. The variable is of the data type REAL.

VAR_IN_OUT

```
rVar : REAL;
```

rVar: variable that is programmed fail-safe by the user.

4.2.8 FB_HVACPersistent_Sint



VAR_INPUT

```
bSetDefault : BOOL;
siVar_Default : SINT;
```

bSetDefault: if the variable is TRUE, the value of the input variable *siVar_Default* is adopted.

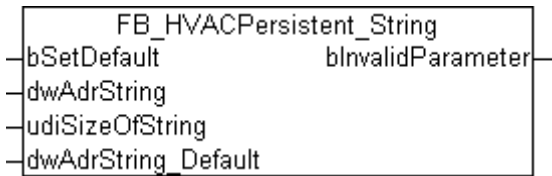
siVar_Default: default value defined by the user. The variable is of the data type SINT.

VAR_IN_OUT

```
siVar : SINT;
```

siVar: variable that is programmed fail-safe by the user.

4.2.9 FB_HVACPersistent_String



VAR_INPUT

```
bSetDefault : BOOL;
dwAdrString : DWORD;
udiSizeOfString : UDINT;
dwAdrString_Default : DWORD;
```

bSetDefault: if the variable is TRUE, the default values of the declared variable *dwAdrString_Default* are adopted and copied to the *dwAdrString* address.

dwAdrString: address of the declared variable to be persistently saved.

udiSizeOfString: size of the declared variable in bytes. (1 byte per character (size at declaration) + 1 byte for null termination)

dwAdrString_Default: address from the string with the default value.

VAR_OUTPUT

```
bInvalidParameter : BOOL;
```

bInvalidParameter: TRUE if *udiSizeOfString > g_udiMaxSizeOfString* OR *dwAdrString = 0* OR *dwAdrString_Default = 0*

VAR_GLOBAL CONSTANT

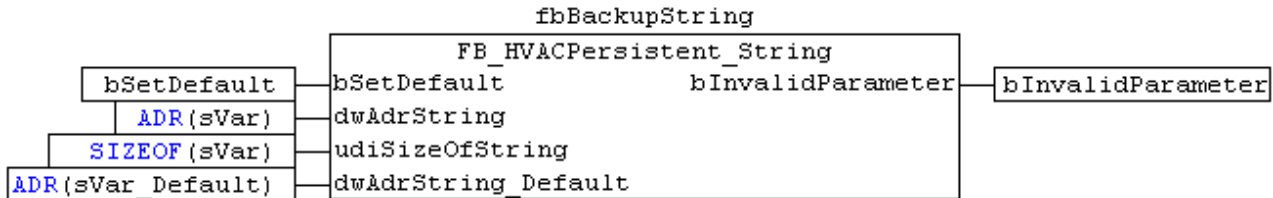
```
Name : Type
```

```
g_udiMaxSizeOfString: UDINT := 255; (* size in number of characters *)
```

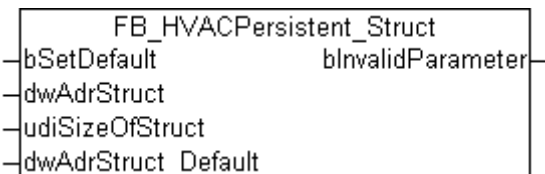
g_udiMaxSizeOfString: any size of the string can be specified via this globally declared constant. Preset value is 255.

Sample:

```
VAR
  fbBackupString      : FB_HVACPersistent_STRING;
  bSetDefault         : BOOL;                (* default flag *)
  sVar                : STRING(55);         (* normal value *)
  sVar_Default        : STRING(55) := 'Store the default String Value';  (* default value *)
  bInvalidParameter   : BOOL;
END_VAR
```



4.2.10 FB_HVACPersistent_Struct



VAR_INPUT

```
bSetDefault          : BOOL;
dwAdrStruct          : DWORD;
udiSizeOfStruct      : UDINT;
dwAdrStruct_Default  : DWORD;
```

bSetDefault: if the variable is TRUE, the default values of the allocated structure instance *dwAdrStruct_Default* are adopted and copied to the *dwAdrStruct* address.

dwAdrStruct: address of the declared structure variable to be persistently saved.

udiSizeOfStruct: size of the instantiated structure in bytes.

dwAdrStruct_Default: address of the allocated structure instance with the default values.

VAR_OUTPUT

```
bInvalidParameter    : BOOL;
```

bInvalidParameter: TRUE if *udiSizeOfStruct* > *g_udiMaxNoOfBytesInStruct* OR *dwAdrStruct* = 0 OR *dwAdrStruct_Default* = 0

VAR_GLOBAL CONSTANT

```
g_udiMaxNoOfBytesInStruct : UDINT := 16; (* size in number of bytes *)
```

g_udiMaxNoOfBytesInStruct: this globally declared constant defines the size of the structure in bytes. By default 16 bytes are defined. This constant can be adapted to the requirements.

Sample:

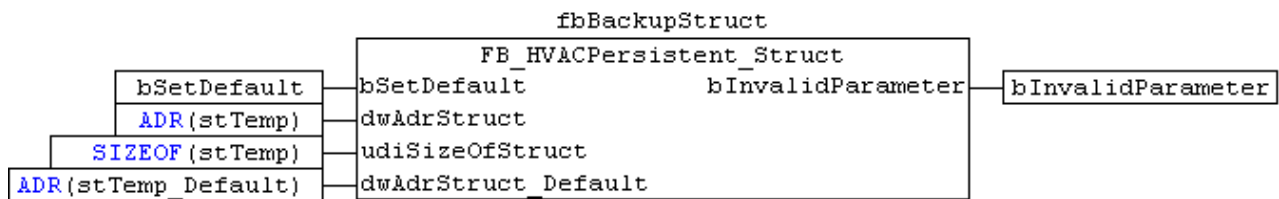
```
VAR
  fbBackupStruct      : FB_HVACPersistent_STRUCT;
  bSetDefault         : BOOL;                (* default flag *)
  stTemp              : ST_Temp;            (* structure values *)
  stTemp_Default      : ST_Temp;            (* default Values *)
  bInvalidParameter   : BOOL;
END_VAR
```


ST_Temp:

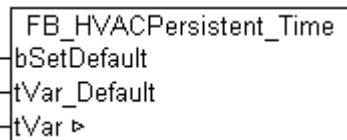
```

TYPE ST_Temp :
STRUCT
  (* total of 8 bytes *)
  byVar1      : BYTE;
  byVar2      : BYTE;
  byVar3      : BYTE;
  byVar4      : BYTE;
  iVar1       : INT;
  iVar2       : INT;
END_STRUCT
END_TYPE

stTemp_Default.byVar1:= 11; (* one byte *)
stTemp_Default.byVar2:= 22; (* one byte *)
stTemp_Default.byVar3:= 33; (* one byte *)
stTemp_Default.byVar4:= 44; (* one byte *)
stTemp_Default.iVar1:= 55; (* two bytes *)
stTemp_Default.iVar2:= 66; (* two bytes *)
    
```



4.2.11 FB_HVACPersistent_Time



VAR_INPUT

```

bSetDefault      : BOOL;
tVar_Default     : TIME;
    
```

bSetDefault: if the variable is TRUE, the value of the input variable *tVar_Default* is adopted.

tVar_Default: default value defined by the user. The variable is of the data type TIME.

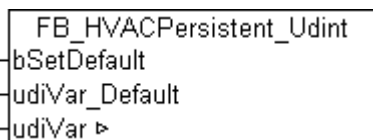
VAR_IN_OUT

```

tVar              : TIME;
    
```

tVar: variable that is programmed fail-safe by the user.

4.2.12 FB_HVACPersistent_Udint



VAR_INPUT

```

bSetDefault      : BOOL;
udiVar_Default   : UDINT;
    
```

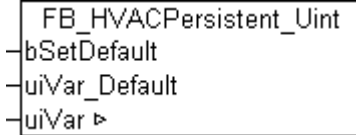
bSetDefault: if the variable is TRUE, the value of the input variable *udiVar_Default* is adopted.

udiVar_Default: default value defined by the user. The variable is of the data type UDINT.

VAR_IN_OUT

```
udiVar          : UDINT;
```

udiVar: variable that is programmed fail-safe by the user.

4.2.13 FB_HVACPersistent_Uint**VAR_INPUT**

```
bSetDefault      : BOOL;
uiVar_Default    : UINT;
```

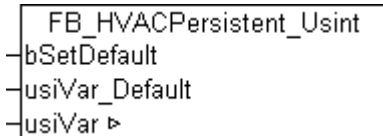
bSetDefault: if the variable is TRUE, the value of the input variable *uiVar_Default* is adopted.

uiVar_Default: default value defined by the user. The variable is of the data type UINT.

VAR_IN_OUT

```
uiVar          : UINT;
```

uiVar: variable that is programmed fail-safe by the user.

4.2.14 FB_HVACPersistent_Usint**VAR_INPUT**

```
bSetDefault      : BOOL;
usiVar_Default    : USINT;
```

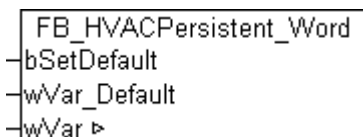
bSetDefault: if the variable is TRUE, the value of the input variable *usiVar_Default* is adopted.

usiVar_Default: default value defined by the user. The variable is of the data type USINT.

VAR_IN_OUT

```
usiVar          : USINT;
```

usiVar: variable that is programmed fail-safe by the user.

4.2.15 FB_HVACPersistent_Word**VAR_INPUT**

```
bSetDefault      : BOOL;
wVar_Default     : WORD;
```

bSetDefault: if the variable is TRUE, the value of the input variable *wVar_Default* is adopted.

wVar_Default: default value defined by the user. The variable is of the data type WORD.

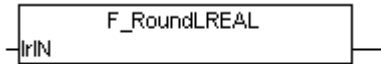
VAR_IN_OUT

```
wVar : WORD;
```

wVar: variable that is programmed fail-safe by the user.

5 Functions

5.1 F_RoundLREAL



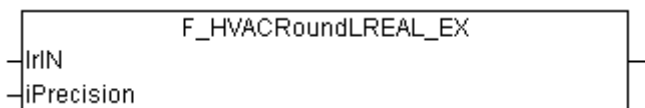
Application

The function *F_RoundLREAL* rounds the input variable *lrIN* of data type LREAL to 1 decimal place. This function can also be used for REAL data types.

VAR_INPUT

```
lrIN : LREAL;
```

5.2 F_RoundLREAL_EX



Application

The function *F_RoundLREAL_EX* rounds the input variable *lrIN* of the data type LREAL to the number of decimal places specified by *iPrecision*. This function can also be used for REAL data types. An instance of the function **LTRUNC** from the library TcMath.lib is used internally in the function.

VAR_INPUT

```
lrIN : LREAL;
iPrecision : INT; 0..5
```

lrIN: floating point number that is to be formatted.



- If $lrIN < 0.1$ AND $lrIN \geq 0.05$, then the return value of the function is 0.1

iPrecision: precision. The value determines the number of digits after the decimal point. The range of decimal places begins with 0 and ends with 5.

Application example

Download	Required library
TcHVAC.pro [► 531]	TcHVAC.lib

6 Enumerations and Structures

6.1 E_HVAC2PointActuatorMode

```

TYPE E_HVAC2PointActuatorMode:
(
  eHVAC2PointActuatorMode_Auto_BMS      := 0,
  eHVAC2PointActuatorMode_Open_BMS      := 1,
  eHVAC2PointActuatorMode_Close_BMS     := 2,
  eHVAC2PointActuatorMode_Auto_OP       := 3,
  eHVAC2PointActuatorMode_Open_OP       := 4,
  eHVAC2PointActuatorMode_Close_OP      := 5
);
END_TYPE

```

..**BMS**: **B**uilding **M**anagement **S**ystem; remote control of the actuator from an BMS.

..**OP**: **O**perator **P**anel; local operation of the actuator from an OP.

6.2 E_HVAC2PointCtrlMode

```

TYPE E_HVAC2PointCtrlMode:
(
  eHVAC2PointCtrlMode_Auto_BMS          := 0,
  eHVAC2PointCtrlMode_On_BMS            := 1,
  eHVAC2PointCtrlMode_Off_BMS           := 2,
  eHVAC2PointCtrlMode_Auto_OP           := 3,
  eHVAC2PointCtrlMode_On_OP             := 4,
  eHVAC2PointCtrlMode_Off_OP            := 5
);
END_TYPE

```

..**BMS**: **B**uilding **M**anagement **S**ystem; remote control of the actuator from an BMS.

..**OP**: **O**perator **P**anel; local operation of the actuator from an OP.

6.3 E_HVAC3PointActuatorMode

```

TYPE E_HVAC3PointActuatorMode:
(
  eHVAC3PointActuatorMode_Auto_BMS      := 0,
  eHVAC3PointActuatorMode_Open_BMS      := 1,
  eHVAC3PointActuatorMode_Close_BMS     := 2,
  eHVAC3PointActuatorMode_Stop_BMS      := 3,
  eHVAC3PointActuatorMode_Auto_OP       := 4,
  eHVAC3PointActuatorMode_Open_OP       := 5,
  eHVAC3PointActuatorMode_Close_OP      := 6,
  eHVAC3PointActuatorMode_Stop_OP       := 7
);
END_TYPE

```

..**BMS**: **B**uilding **M**anagement **S**ystem; remote control of the actuator from an BMS.

..**OP**: **O**perator **P**anel; local operation of the actuator from an OP.

6.4 E_HVACActuatorMode

```

TYPE E_HVACActuatorMode :
(
  eHVACActuatorMode_Auto_BMS            := 0,
  eHVACActuatorMode_Off_BMS             := 1,
  eHVACActuatorMode_Speed1_BMS          := 2,
  eHVACActuatorMode_Speed2_BMS          := 3,
  eHVACActuatorMode_Speed3_BMS          := 4,
  eHVACActuatorMode_Speed4_BMS          := 5,

```

```

eHVACActuatorMode_Auto_OP      := 6,
eHVACActuatorMode_Off_OP       := 7,
eHVACActuatorMode_Speed1_OP    := 8,
eHVACActuatorMode_Speed2_OP    := 9,
eHVACActuatorMode_Speed3_OP    := 10,
eHVACActuatorMode_Speed4_OP    := 11
);
END_TYPE

```

..**BMS**: Building Management System; remote control of the actuator from an BMS.

..**OP**: Operator Panel; local operation of the actuator from an OP.

6.5 E_HVACAIRConditioning2SpeedMode

```

TYPE E_HVACAIRConditioning2SpeedMode:
(
  eHVACAIRConditioning2SpeedMode_Off      := 0,
  eHVACAIRConditioning2SpeedMode_Heating := 1,
  eHVACAIRConditioning2SpeedMode_Cooling  := 2,
  eHVACAIRConditioning2SpeedMode_HeatingAndCooling := 3
);
END_TYPE

```

6.6 E_HVACAnalogOutputMode

```

TYPE E_HVACAnalogOutputMode :
(
  eHVACAnalogOutputMode_Auto_BMS := 0,
  eHVACAnalogOutputMode_Manual_BMS := 1,
  eHVACAnalogOutputMode_Auto_OP := 2,
  eHVACAnalogOutputMode_Manual_OP := 3
);
END_TYPE

```

..**BMS**: Building Management System; remote control of the actuator from an BMS.

..**OP**: Operator Panel; local operation of the actuator from an OP.

6.7 E_HVACAntiBlockingMode

```

TYPE E_HVACAntiBlockingMode:
(
  eHVACAntiBlockingMode_Off      := 0,
  eHVACAntiBlockingMode_Downtime := 1,
  eHVACAntiBlockingMode_Weekly   := 2
);
END_TYPE

```

6.8 E_HVACBusTerminal_KL32xx

```

TYPE E_HVACBusTerminal_KL32xx :
(
  eHVACBusTerminal_None := 0,
  (*=====*)
  eHVACBusTerminal_KL3201_0000 := 1, (*Standard PT100*)
  eHVACBusTerminal_KL3201_0010 := 2, (*PT200*)
  eHVACBusTerminal_KL3201_0011 := 3, (*not supported, PT200 Siemens format*)
  eHVACBusTerminal_KL3201_0012 := 4, (*PT500*)
  eHVACBusTerminal_KL3201_0013 := 5, (*not supported, PT500 Siemens format*)
  eHVACBusTerminal_KL3201_0014 := 6, (*PT1000*)
  eHVACBusTerminal_KL3201_0015 := 7, (*not supported, PT1000 Siemens format*)
  eHVACBusTerminal_KL3201_0016 := 8, (*Ni100*)
  eHVACBusTerminal_KL3201_0017 := 9, (*not supported, Ni100 Siemens format*)
  eHVACBusTerminal_KL3201_0018 := 10, (*not supported*)
  eHVACBusTerminal_KL3201_0020 := 11, (*Ohm10_1200*)
  eHVACBusTerminal_KL3201_0021 := 12, (*not supported, PT100 Siemens format*)
  eHVACBusTerminal_KL3201_0023 := 13, (*Ni120*)
  eHVACBusTerminal_KL3201_0024 := 14, (*not supported, Ni120 Siemens format*)

```

```

eHVACBusTerminal_KL3201_0025 := 15, (*Ni1000*)
eHVACBusTerminal_KL3201_0026 := 16, (*not supported, Ni1000 Siemens format*)
eHVACBusTerminal_KL3201_0027 := 17, (*not supported, Ohm10_10000*)
eHVACBusTerminal_KL3201_0028 := 18, (*not supported, high resolution 0,01°C*)
eHVACBusTerminal_KL3201_0029 := 19, (*Ni1000_LS, Landis&Stefa, TK5000*)
eHVACBusTerminal_KL3201_0030 := 20, (*not supported*)
eHVACBusTerminal_KL3201_0031 := 21, (*PT1000,2 wire connection*)
(*=====*)
eHVACBusTerminal_KL3202_0000 := 22, (*Standard PT100*)
eHVACBusTerminal_KL3202_0010 := 23, (*PT200*)
eHVACBusTerminal_KL3202_0011 := 24, (*not supported, PT200 Siemens format*)
eHVACBusTerminal_KL3202_0012 := 25, (*PT500*)
eHVACBusTerminal_KL3202_0013 := 26, (*not supported, PT500 Siemens format*)
eHVACBusTerminal_KL3202_0014 := 27, (*PT1000*)
eHVACBusTerminal_KL3202_0015 := 28, (*not supported, PT1000 Siemens format*)
eHVACBusTerminal_KL3202_0016 := 29, (*Ni100*)
eHVACBusTerminal_KL3202_0017 := 30, (*not supported, Ni100 Siemens format*)
eHVACBusTerminal_KL3202_0018 := 31, (*not supported*)
eHVACBusTerminal_KL3202_0020 := 32, (*Ohm10_1200*)
eHVACBusTerminal_KL3202_0021 := 33, (*not supported, PT100 Siemens format*)
eHVACBusTerminal_KL3202_0023 := 34, (*Ni120*)
eHVACBusTerminal_KL3202_0024 := 35, (*not supported, Ni120 Siemens format*)
eHVACBusTerminal_KL3202_0025 := 36, (*Ni1000*)
eHVACBusTerminal_KL3202_0026 := 37, (*not supported, Ni1000 Siemens format*)
eHVACBusTerminal_KL3202_0027 := 38, (*not supported, Ohm10_10000*)
eHVACBusTerminal_KL3202_0028 := 39, (*not supported, high resolution 0,01°C*)
eHVACBusTerminal_KL3202_0029 := 40, (*Ni1000_LS, Landis&Stefa, TK5000*)
eHVACBusTerminal_KL3202_0030 := 41, (*not supported*)
eHVACBusTerminal_KL3202_0031 := 42, (*PT1000,2 wire connection*)
(*=====*)
eHVACBusTerminal_KL3204_0000 := 43, (*Standard PT100*)
eHVACBusTerminal_KL3204_0010 := 44, (*PT200*)
eHVACBusTerminal_KL3204_0011 := 45, (*not supported, PT200 Siemens format*)
eHVACBusTerminal_KL3204_0012 := 46, (*PT500*)
eHVACBusTerminal_KL3204_0013 := 47, (*not supported, PT500 Siemens format*)
eHVACBusTerminal_KL3204_0014 := 48, (*PT1000*)
eHVACBusTerminal_KL3204_0015 := 49, (*not supported, PT1000 Siemens format*)
eHVACBusTerminal_KL3204_0016 := 50, (*Ni100*)
eHVACBusTerminal_KL3204_0017 := 51, (*not supported, Ni100 Siemens format*)
eHVACBusTerminal_KL3204_0018 := 52, (*not supported*)
eHVACBusTerminal_KL3204_0020 := 53, (*Ohm10_1200*)
eHVACBusTerminal_KL3204_0021 := 54, (*not supported, PT100 Siemens format*)
eHVACBusTerminal_KL3204_0023 := 55, (*Ni120*)
eHVACBusTerminal_KL3204_0024 := 56, (*not supported, Ni120 Siemens format*)
eHVACBusTerminal_KL3204_0025 := 57, (*Ni1000*)
eHVACBusTerminal_KL3204_0026 := 58, (*not supported, Ni1000 Siemens format*)
eHVACBusTerminal_KL3204_0027 := 59, (*not supported, Ohm10_10000*)
eHVACBusTerminal_KL3204_0028 := 60, (*not supported, high resolution 0,01°C*)
eHVACBusTerminal_KL3204_0029 := 61, (*Ni1000_LS, Landis&Stefa, TK5000*)
eHVACBusTerminal_KL3204_0030 := 62, (*not supported*)
(*=====*)
eHVACBusTerminal_KL3228_0000 := 63, (*Standard PT1000*)
(*=====*)
eHVACBusTerminal_NotSupported := 64, (*error, bus terminal not supported*)
eHVACBusTerminal_KL3201_NotSupported := 65, (*error, bus terminal not supported*)
eHVACBusTerminal_KL3202_NotSupported := 66, (*error, bus terminal not supported*)
eHVACBusTerminal_KL3204_NotSupported := 67, (*error, bus terminal not supported*)
eHVACBusTerminal_KL3228_NotSupported := 68 (*error, bus terminal not supported*)
(*=====*)
eHVACBusTerminal_KL3208_0010 := 69, (*Standard PT1000*)
eHVACBusTerminal_KL3208_NotSupported := 70 (*error, bus terminal not supported*)
);
END_TYPE

```

6.9 E_HVACCtrlMode

```

TYPE E_HVACCtrlMode :
(
  eHVACCtrlMode_Auto := 0,
  eHVACCtrlMode_Manual := 1
);
END_TYPE

```

6.10 E_HVACConvectionMode

```

TYPE E_HVACConvectionMode :
(
    eHVACConvectionMode_WithFan      := 0,
    eHVACConvectionMode_WithoutFan   := 1
);
END_TYPE

```

6.11 E_HVACDataSecurityType

```

TYPE E_HVACDataSecurityType:
(
    eHVACDataSecurityType_Persistent := 0,
    eHVACDataSecurityType_Idle      := 1
);
END_TYPE

```

6.12 E_HVACErrorCodes

```

TYPE E_HVACErrorCodes :
(
    eHVACErrorCodes_NoError                := 0,
    eHVACErrorCodes_Error_tTaskCycleTime   := 1,
    eHVACErrorCodes_Error_tCtrlCycleTime   := 2,
    eHVACErrorCodes_InvalidParam_rDeadRange := 3,
    eHVACErrorCodes_InvalidParam_rDeadBand  := 4,
    eHVACErrorCodes_InvalidParam_rKpIsLessThanZero := 5,
    eHVACErrorCodes_InvalidParam_tTi        := 6,
    eHVACErrorCodes_InvalidParam_tTv        := 7,
    eHVACErrorCodes_InvalidParam_tTd        := 8,
    eHVACErrorCodes_InvalidParam_fBaseTime  := 9,
    eHVACErrorCodes_InvalidParam_rYMaxIsLessThanrYMin := 10,
    eHVACErrorCodes_Error_MEMCPY           := 11,
    eHVACErrorCodes_Error_ModeNotSupported := 12,
    eHVACErrorCodes_Error_rErrHighLimitOverrun := 13,
    eHVACErrorCodes_Error_rErrLowLimitUnderrun := 14,
    eHVACErrorCodes_Error_NoFeedbackActuator1 := 15,
    eHVACErrorCodes_Error_NoFeedbackActuator2 := 16,
    eHVACErrorCodes_Error_NoFeedbackActuator3 := 17,
    eHVACErrorCodes_Error_NoFeedbackActuator4 := 18,
    eHVACErrorCodes_Error_NoFeedbackActuator5 := 19,
    eHVACErrorCodes_Error_NoFeedbackActuator6 := 20,
    eHVACErrorCodes_Error_NoFeedbackActuator7 := 21,
    eHVACErrorCodes_Error_NoFeedbackActuator8 := 22,
    eHVACErrorCodes_InvalidParam_tOverlap1Actuator := 23,
    eHVACErrorCodes_InvalidParam_iCountCtrl := 24,
    eHVACErrorCodes_iMaxOnLevel             := 25,
    eHVACErrorCodes_tNoFeedbActuator        := 26,
    eHVACErrorCodes_Error_iNumberOfSequences := 27,
    eHVACErrorCodes_Error_iMyNumberOfSequences := 28,
    eHVACErrorCodes_Error_iCurrentSequences := 29,
    eHVACErrorCodes_Error_MEMSET           := 30
);
END_TYPE

```

6.13 E_HVACExternalMode

```

TYPE E_HVACExternalMode :
(
    eHVACExternalMode_Off      := 0,
    eHVACExternalMode_On       := 1,
    eHVACExternalMode_ShiftAbsolute := 2
);
END_TYPE

```


6.14 E_HVACExternalRequestMode

```

TYPE E_HVACExternalRequestMode:
(
  eHVACExternalRequestMode_ButtonOn_Off      := 0,
  eHVACExternalRequestMode_ButtonOffDelay    := 1,
  eHVACExternalRequestMode_SwitchOn_Off      := 2
);
END_TYPE

```

6.15 E_HVACPlantMode

```

TYPE E_HVACPlantMode:
(
  eHVACPlantMode_TimeSchedulingOnly          := 0,
  eHVACPlantMode_ExternalRequestOnly        := 1,
  eHVACPlantMode_TimeScheduling_And_ExternalRequest := 2,
  eHVACPlantMode_TimeScheduling_Or_ExternalRequest := 3
);
END_TYPE

```

6.16 E_HVACPowerMeasurementMode

```

TYPE E_HVACPowerMeasurementMode:
(
  eHVACPowerMeasurementMode_AutoAllValues    := 0,
  eHVACPowerMeasurementMode_Current         := 1,
  eHVACPowerMeasurementMode_Voltage         := 2,
  eHVACPowerMeasurementMode_EffectivePower   := 3,
  eHVACPowerMeasurementMode_EnergyConsumption := 4,
  eHVACPowerMeasurementMode_PeakCurrentValue := 5,
  eHVACPowerMeasurementMode_PeakVoltageValue := 6,
  eHVACPowerMeasurementMode_PeakPowerValue  := 7
);
END_TYPE

```

6.17 E_HVACReferencingMode

```

TYPE E_HVACReferencingMode :
(
  eHVACReferencingMode_Emulation      := 0,
  eHVACReferencingMode_AnalogFeedback := 1
);
END_TYPE

```

6.18 E_HVACRegOutsideTemp

```

TYPE E_HVACRegOutsideTemp:
(
  eHVACReqOutsideTemp_NoRequest      := 0,
  eHVACReqOutsideTemp_OTLowerLimit   := 1,
  eHVACReqOutsideTemp_OTHigherLimit  := 2
);
END_TYPE

```

6.19 E_HVACReqPump

```

TYPE E_HVACReqPump:
(
  eHVACReqPump_No           := 0,
  eHVACReqPump_OT_LL       := 1,
  eHVACReqPump_OT_HL       := 2,
  eHVACReqPump_VP          := 3,
  eHVACReqPump_OT_LL_OR_VP := 4,
  eHVACReqPump_OT_HL_OR_VP := 5,
  eHVACReqPump_OT_LL_AND_VP := 6,
  eHVACReqPump_OT_HL_AND_VP := 7
);
END_TYPE

```

eHVACReqPump_No: There is no request on the part of the Enum to control the pump

eHVACReqPump_OT_LL: The outside temperature (OT = *rOutsideTemp*) must be smaller than *rOutsideTempLowLimit* (LL = Lower Limit)

eHVACReqPump_OT_HL: The outside temperature (OT = *rOutsideTemp*) must be larger than *rOutsideTempHighLimit* (HL = Higher Limit)

eHVACReqPump_VP: The valve position (VP = *rValvePosition*) must be larger than *rValvePositionLimitOn*

eHVACReqPump_OT_LL_OR_VP: The outside temperature (OT = *rOutsideTemp*) must be smaller than *rOutsideTempLowLimit* (LL = Lower Limit) OR the valve position (VP = *rValvePosition*) must be larger than *rValvePositionLimitOn*

eHVACReqPump_OT_HL_OR_VP: The outside temperature (OT = *rOutsideTemp*) must be larger than *rOutsideTempHighLimit* (HL = Higher Limit) OR the valve position (VP = *rValvePosition*) must be larger than *rValvePositionLimitOn*

eHVACReqPump_OT_LL_AND_VP: The outside temperature (OT = *rOutsideTemp*) must be smaller than *rOutsideTempLowLimit* (LL = Lower Limit) AND the valve position (VP = *rValvePosition*) must be larger than *rValvePositionLimitOn*

eHVACReqPump_OT_HL_AND_VP: The outside temperature (OT = *rOutsideTemp*) must be larger than *rOutsideTempHighLimit* (HL = Higher Limit) AND the valve position (VP = *rValvePosition*) must be larger than *rValvePositionLimitOn*

6.20 E_HVACRegValve

```
TYPE E_HVACRegValve:
(
    eHVACReqValve_NoRequest           := 0,
    eHVACReqValve_OrValvePosHigherLimit := 1,
    eHVACReqValve_AndValvePosHigherLimit := 2
);
END_TYPE
```

6.21 E_HVACSensorType

```
TYPE E_HVACSensorType :
(
    eHVACSensorType_None           := 0,
    eHVACSensorType_PT100          := 1,
    eHVACSensorType_PT200          := 2,
    eHVACSensorType_PT500          := 3,
    eHVACSensorType_PT1000         := 4,
    eHVACSensorType_NI100          := 5,
    eHVACSensorType_NI120          := 6,
    eHVACSensorType_NI1000         := 7, (*Ni1000,DIN*)
    eHVACSensorType_NI1000_LS      := 8, (*Ni1000_LS, Landis&Staefa, TK5000*)
    eHVACSensorType_Ohm10_1200     := 9,
    eHVACSensorType_Ohm10_5000     := 10,
    eHVACSensorType_Ohm10_10000    := 11,
    eHVACSensorType_NotSupported    := 12
);
END_TYPE
```

6.22 E_HVACSequenceCtrlMode

The controller is enabled and controlled within the sequence via this ENUM, depending on the system operating modes.

```
TYPE E_HVACSequenceCtrlMode :
(
    eHVACSequenceCtrlMode_Stop           := 0,
    eHVACSequenceCtrlMode_On             := 1,
    eHVACSequenceCtrlMode_NightCooling   := 2,
    eHVACSequenceCtrlMode_FreezeProtection := 3,
    eHVACSequenceCtrlMode_OverheatingProtection := 4,
    eHVACSequenceCtrlMode_NightCoolingAndOverheatingProtection := 5
);
END_TYPE
```

6.23 E_HVACSetpointHeatingMode

```

TYPE E_HVACSetpointHeatingMode :
(
  eHVACSetpointHeatingMode_Off           := 0,
  eHVACSetpointHeatingMode_OnOutsideTemp := 1,
  eHVACSetpointHeatingMode_OnDate        := 2,
  eHVACSetpointHeatingMode_OnPermanent   := 3,
  eHVACSetpointHeatingMode_OnFreeze Protec := 4,
  eHVACSetpointHeatingMode_OnNight       := 5,
  eHVACSetpointHeatingMode_OnDay         := 6
);
END_TYPE

```

eHVACSetpointHeatingMode_Off: the heating circuit is completely switched off.



The frost protection function is also deactivated in this operation mode!

eHVACSetpointHeatingMode_OnOutsideTemp: heating period according to outside temperature. The heating circuit is switched on if the average outside temperature *rOutsideTempDamped* is smaller than the value of *rHeatingLimit*. The flow temperature setpoint calculated by the heating characteristic curve or the

room setpoint module is passed through to the output *rSetpoint*.

The output *bHeatingPump* becomes TRUE and thus the heating circuit pump is switched on. Switching on or off following the heating limits being undershot or exceeded can be delayed by the parameters *tDelayHeatingOn* and *tDelayHeatingOff*.

eHVACSetpointHeatingMode_OnDate: heating period by date. The heating circuit is enabled from the date *iOn_Day / iOn_Month* to *iOff_Day / iOff_Month*

. The switching on or off of the heating circuit pump after the switch-on or switch-off limits are reached can be delayed.

eHVACSetpointHeatingMode_OnPermanent: the heating circuit is always switched on.

eHVACSetpointHeatingMode_OnFreeze Protec: the heating circuit is in frost protection mode. If the input variable *bFreeze Protec* is TRUE, the value applied to *rFreeze ProtecSetpoint* is passed through to the output *rSetpointOut*.

eHVACSetpointHeatingMode_OnNight: the setpoint for the flow temperature is calculated depending on the outside temperature. The heating circuit is on continuously, and remains in night setback mode. In this operation mode, the value from the input variable *rNightSetback* is subtracted from the *rSetpointIn*.

eHVACSetpointHeatingMode_OnDay: the heating circuit is in day operation.

6.24 E_HVACSetpointMode

```

TYPE E_HVACSetpointMode :
(
  eHVACSetpointMode_DIN           := 0,
  eHVACSetpointMode_DINLimited    := 1,
  eHVACSetpointMode_ConstantValueBase := 2
);
END_TYPE

```

6.25 E_HVACState

```

TYPE E_HVACState :
(
  eHVACState_Idle   := 0,
  eHVACState_Active := 1,
  eHVACState_Error  := 2
);
END_TYPE

```

This output indicates the current internal state of the block.

eHVACState_Idle: The block has successfully been reset, and is now waiting for selection of the operating mode.

eHVACState_Active: The block is in the active state, which is the normal operating state.

eHVACState_Error: An error has occurred; the block is not executed when in this state.

6.26 E_HVACTemperatureCurve

```

TYPE E_HVACTemperatureCurve :
(
  eHVACTemperatureCurve_None           := 0,
  eHVACTemperatureCurve_NTC1k8         := 1, (*NTC 1.8k, negative, Thermokon*)
  eHVACTemperatureCurve_Ni1000Tk5000  := 2, (*Ni1000 Tk5000, positive, Thermokon*)
  eHVACTemperatureCurve_NTC1k_3_A1    := 3, (*NTC1k, 1K3 A1, negative, S+S*)
  eHVACTemperatureCurve_NTC1k8_3_A1   := 4, (*NTC1.8k, 1.8K3 A1, negative, S+S*)
  eHVACTemperatureCurve_NTC2k2_3_A1   := 5, (*NTC2.2k, 2.2K3 A1, negative, S+S*)
  eHVACTemperatureCurve_NTC3k3_3_A1   := 6, (*NTC3.3k, 3.3K3 A1, negative, S+S*)
  eHVACTemperatureCurve_Ni1000Tk5000_TCR := 7, (*Ni1000 Tk5000 TCR, positive, S+S*)
  eHVACTemperatureCurve_Ni1000_DIN     := 8, (*Ni1000 DIN, positive, S+S*)
  eHVACTemperatureCurve_Pt1000_DIN     := 9 (*Ni1000 DIN, positive, S+S*)
);
END_TYPE

```

6.27 E_HVACTemperatureSensorMode

E_HVACTemperatureSensorMode

```

TYPE E_HVACTemperatureSensorMode :
(
  eHVACTemperatureSensorMode_ReplacementValue := 0,
  eHVACTemperatureSensorMode_LastValue       := 1,
  eHVACTemperatureSensorMode_AutoResetReplacementValue := 2,
  eHVACTemperatureSensorMode_AutoResetLastValue := 3
);
END_TYPE

```

6.28 ST_HVAC2PointCtrlSequence

```

TYPE ST_HVAC2PointCtrlSequence :
STRUCT
  tRemainingTimeIncreaseSequence : TIME;
  tRemainingTimeDecreaseSequence : TIME;
  rX                             : REAL;
  rW_Max                         : REAL;
  rW_Min                         : REAL;
  rE                             : REAL;
  rCtrl_I_HighLimit              : REAL;
  rCtrl_I_LowLimit               : REAL;
  rCtrl_I_Out                    : REAL;
  e2PointCtrlState               : E_HVAC2PointCtrlMode;
  iNumberOfSequences              : INT;
  iMyNumberInSequence             : INT;
  iCurrentSequence                : INT;
  bEnable                        : BOOL;
  bError                          : BOOL;
  bOut                            : BOOL;
  bActiveCtrl                    : BOOL;
  b_rW_Max                       : BOOL;
  b_rW_Min                       : BOOL;
END_STRUCT
END_TYPE

```

6.29 ST_HVACAggregate

```

TYPE ST_HVACAggregate :
STRUCT
  rY_Max : REAL;
  rY_Min : REAL;
  iAggregateStep : INT;

```

```

    bBlock          : BOOL;
END_STRUCT
END_TYPE

```

rY_Max: Parameter specification for constant units.

rY_Min: Parameter specification for constant units.

iAggregateStep: Parameter that specifies the step in which the addressed unit should be fixed or should regulate; see *bBlock*.

bBlock: If *bBlock* = FALSE, then the addressed unit is fixed in the specified step via *iAggregateStep*. If *bBlock* = TRUE, then the control of the addressed unit is released from the off step (0) to the specified step via *iAggregateStep*.

6.30 ST_HVACCmdCtrl_8Param

```

TYPE ST_HVACCmdCtrl_8Param :
STRUCT
    udiSecDelayOn      : UDINT;
    udiSecDelayOff     : UDINT;
    udiSecMinOn        : UDINT;
    udiSecMinOff       : UDINT;
END_STRUCT
END_TYPE

```

udiSecDelayOn: switch-on delay in seconds

udiSecDelayOff: switch-off delay in seconds

udiSecMinOn: minimum switch-on time in seconds

udiSecMinOff: minimum switch-off time in seconds

6.31 ST_HVACCmdCtrl_8State

```

TYPE ST_HVACCmdCtrl_8State :
STRUCT
    udiSecRT_DelayOn   : UDINT;
    udiSecRT_DelayOff  : UDINT;
    udiStep             : UDINT;
    udiSecRT_MinOn     : ARRAY[0..g_iNumOfCmdCtrl_8] OF UDINT;
    udiSecRT_MinOff    : ARRAY[0..g_iNumOfCmdCtrl_8] OF UDINT;
END_STRUCT
END_TYPE

```

udiSecRT_DelayOn: The remaining time of the switch-on delay of the current step *udiStep* is indicated.

udiSecRT_DelayOff: The remaining time of the switch-off delay of the current step *udiStep* is indicated.

udiStep: Status indicating which step the function block is in.

udiSecRT_MinOn[0.. g_iNumOfCmdCtrl_8 [▶ 531]]: The remaining time of the minimum switch-on duration of the outputs is indicated in the one-dimensional field (Table) *udiSecRT_MinOn[0..g_iNumOfCmdCtrl_8 [▶ 531]]*.

udiSecRT_MinOff[0.. g_iNumOfCmdCtrl_8 [▶ 531]]: The remaining time of the minimum switch-off duration of the outputs is indicated in the one-dimensional field (Table) *udiSecRT_MinOff[0..g_iNumOfCmdCtrl_8 [▶ 531]]*.

6.32 ST_HVACHoliday

```

TYPE ST_HVACHoliday :
STRUCT
    uiOn_Day          : UINT;
    uiOn_Month        : UINT;
    uiOff_Day         : UINT;

```

```

uiOff_Month      : UINT;
bEnable         : BOOL;
bResetAfterOn   : BOOL;
bQ              : BOOL;
END_STRUCT
END_TYPE

```

6.33 ST_HVACI_Ctrl

```

TYPE ST_HVACI_Ctrl :
STRUCT
    rIntegralHigh      : REAL;
    rIntegralLow       : REAL;
    udiSecDelayHigh    : UDINT;
    udiSecDelayLow     : UDINT;
END_STRUCT
END_TYPE

```

rIntegralHigh: Positive value for the upper limit at which the integration of the I-transfer element is stopped

rIntegralLow: Positive value for the lower limit at which the integration of the I-transfer element is stopped.

udiSecDelayHigh: Delay time after whose expiry the I-transfer element is activated.

udiSecDelayLow: Delay time after whose expiry the I-transfer element is activated.

6.34 ST_HVACPeriod

```

TYPE ST_HVACPeriod :
STRUCT
    uiOn_hh          : UINT;
    uiOn_mm          : UINT;
    uiOn_Day         : UINT;
    uiOn_Month       : UINT;
    uiOff_hh         : UINT;
    uiOff_mm         : UINT;
    uiOff_Day        : UINT;
    uiOff_Month      : UINT;
    bEnable          : BOOL;
    bResetAfterOn    : BOOL;
    bQ               : BOOL;
END_STRUCT
END_TYPE

```

6.35 ST_HVACParameterScale_nPoint

```

TYPE ST_HVACParameterScale_nPoint :
STRUCT
    rX              : ARRAY [1..g_iMaxNoOfScale_nPoint] OF REAL;
    rY              : ARRAY [1..g_iMaxNoOfScale_nPoint] OF REAL;
    iNumberOfPoint  : INT;
END_STRUCT
END_TYPE

```

rX[1..g_iMaxNoOfScale_nPoint]: Array that includes the values of the single points of the X-axis. The number of points depends on *iNumberOfPoint*.

NOTICE

Please note: The value rX[1] must be higher then rX[2], rX[2] must be higher then rX[n] OR rX[1] must be lesser then rX[2], rX[2] must be lesser then rX[n].

rY[1..g_iMaxNoOfScale_nPoint]: Array that includes the values of the single points of the Y-axis. The number of points depends on *iNumberOfPoint*.

iNumberOfPoint: Number of the single values of the X-Y-coordinates.

NOTICE

Please note: iNumberOfPoint must not be lesser then 2 or bigger then g_iMaxNoOfScale_nPoint(60).

6.36 ST_HVACPowerMeasurement

```

TYPE ST_HVACPowerMeasurement :
STRUCT
  diIL1, diIL2, diIL3           : DINT;
  diUL1, diUL2, diUL3         : DINT;
  diPL1, diPL2, diPL3         : DINT;
  diPg                          : DINT;
  diCosPhiL1, diCosPhiL2, diCosPhiL3 : DINT;
  diCosPhi                      : DINT;
  diWL1, diWL2, diWL3         : DINT;
  diWg                          : DINT;
  diImaxL1, diImaxL2, diImaxL3 : DINT;
  diUmaxL1, diUmaxL2, diUmaxL3 : DINT;
  diPmaxL1, diPmaxL2, diPmaxL3 : DINT;
  diSg                          : DINT;
  diQg                          : DINT;
  dummy                         : DINT;
END_STRUCT
END_TYPE
    
```

6.37 ST_HVACPowerMeasurementEx

```

TYPE ST_HVACPowerMeasurementEx :
STRUCT
  fIL1, fIL2, fIL3           : LREAL;
  fIg                        : LREAL;
  fUL1, fUL2, fUL3         : LREAL;
  fPL1, fPL2, fPL3         : LREAL;
  fPg                        : LREAL;
  fCosPhiL1, fCosPhiL2, fCosPhiL3 : LREAL;
  fCosPhi                    : LREAL;
  fWL1, fWL2, fWL3         : LREAL;
  fWg                        : LREAL;
  fImaxL1, fImaxL2, fImaxL3 : LREAL;
  fUmaxL1, fUmaxL2, fUmaxL3 : LREAL;
  fPmaxL1, fPmaxL2, fPmaxL3 : LREAL;
  fSg                        : LREAL;
  fQg                        : LREAL;
  fFrequencyL1, fFrequencyL2, fFrequencyL3 : LREAL;
END_STRUCT
END_TYPE
    
```

6.38 ST_HVACPowerRangeTable

```

TYPE ST_HVACPowerRangeTable :
STRUCT
  iAggregate      : INT;
  iAggregateStep  : INT;
  rY_Max         : REAL;
    
```

```

rY_Min      : REAL;
rIntegralHigh  : REAL;
rIntegralLow   : REAL;
udiSecDelayHigh : UDINT;
udiSecDelayLow  : UDINT;
bBlock       : BOOL;
END_STRUCT
END_TYPE

```

iAggregate: Parameter that specifies to which output structure *stAggregate1-6* of the function block FB_HVACPowerRangeTable [▶ 275] the variables *rY_Min*, *rY_Max*, *iAggregateStep* and *bBlock* are written.

iAggregateStep: Parameter that specifies the step in which the addressed unit should be fixed or should regulate, depending on *bBlock*. This variable is output via the structure *stAggregateX*.

rY_Max: Parameter specification for constant units. The variable value must not be smaller than *g_rY_Min* [▶ 531] and must not exceed *g_rY_Max* [▶ 531] and *rY_Max* must not be smaller than *rY_Min*. Otherwise an error is indicated by *bError* = TRUE and the execution of the function block FB_HVACPowerRangeTable [▶ 275] is stopped. This variable is output via the structure *stAggregateX*.

rY_Min: Parameter specification for constant units. The variable value must not be smaller than *g_rY_Min* [▶ 531] and must not exceed *g_rY_Max* [▶ 531] and *rY_Max* must not be smaller than *rY_Min*. Otherwise an error is indicated by *bError* = TRUE and the execution of the function block FB_HVACPowerRangeTable [▶ 275] is stopped. This variable is output via the structure *stAggregateX*.

rlIntegralHigh: Positive value for the upper limit at which the integration of the I-transfer element is stopped; see the VAR_IN_OUT-Variable *rlIntegralHigh* in the FB_HVACI_CtrlStep [▶ 248]. This variable is output via the structure *stI_Ctrl*.

rlIntegralLow: Positive value for the lower limit at which the integration of the I-transfer element is stopped; see the VAR_IN_OUT-Variable *rlIntegralHigh* in the FB_HVACI_CtrlStep [▶ 248]. This variable is output via the structure *stI_Ctrl*.

udiSecDelayHigh: Delay time after whose expiry the I-transfer element is activated; see the VAR_IN_OUT-Variable *udiSecDelayHigh* in the FB_HVACI_CtrlStep [▶ 248]. This variable is output via the structure *stI_Ctrl*.

udiSecDelayLow: Delay time after whose expiry the I-transfer element is activated; see the VAR_IN_OUT-Variable *udiSecDelayHigh* in the FB_HVACI_CtrlStep [▶ 248]. This variable is output via the structure *stI_Ctrl*.

bBlock: If *bBlock* = FALSE, then the addressed unit is fixed in the specified step via *iAggregateStep*. If *bBlock* = TRUE, then the control of the addressed unit is released from the off step (0) to the specified step via *iAggregateStep*. This variable is output via the structure *stAggregateX*.

6.39 ST_HVACTimeChannel

```

TYPE ST_HVACTimeChannel :
STRUCT
  uiOn_hh      : UINT;
  uiOn_mm      : UINT;
  uiOn_ss      : UINT;
  uiOff_hh     : UINT;
  uiOff_mm     : UINT;
  uiOff_ss     : UINT;
  bEnable      : BOOL;
  bAllDays     : BOOL;
  bMonday      : BOOL;
  bTuesday     : BOOL;
  bWednesday   : BOOL;
  bThursday    : BOOL;
  bFriday      : BOOL;
  bSaturday    : BOOL;
  bSunday      : BOOL;
  bResetAfterOn : BOOL;
  bQ           : BOOL;
END_STRUCT
END_TYPE

```


6.40 ST_HVACTempChangeFunction

Structure of the node points for the temperature change function in the modules [FB_HVACOptimizedOn](#) [▶ 431] and [FB_HVACOptimizedOff](#) [▶ 441].

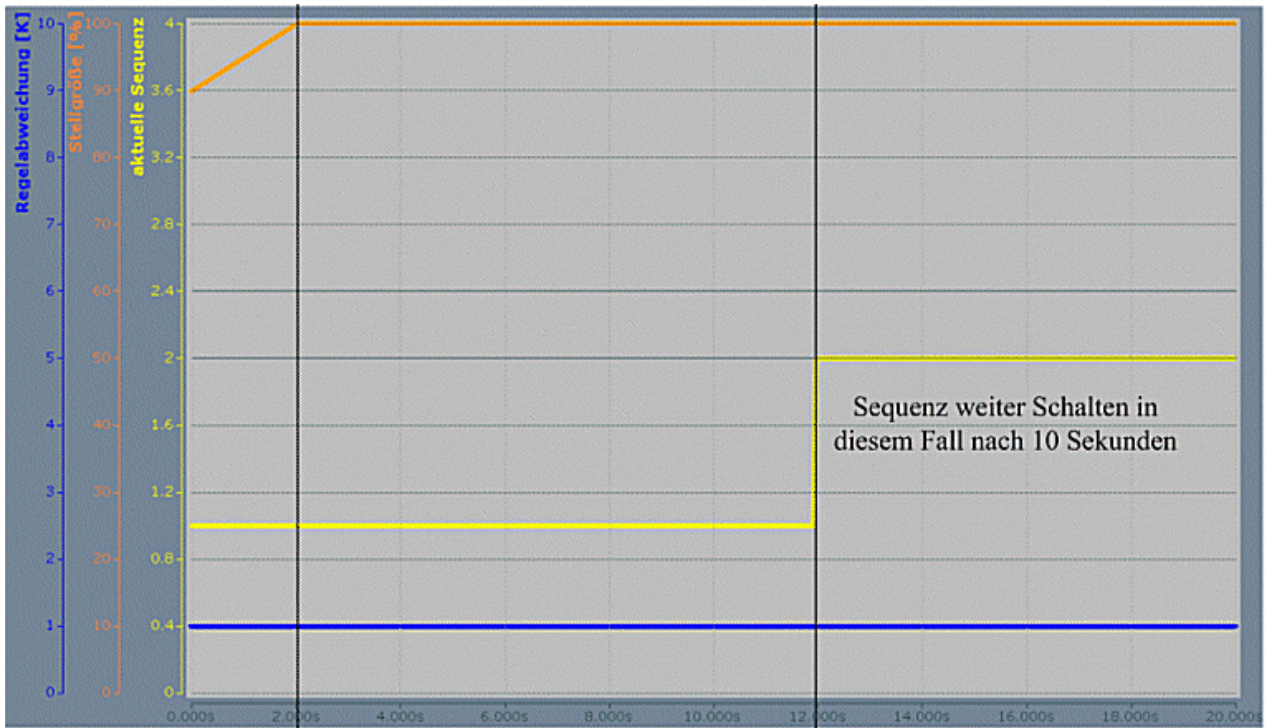
```
TYPE ST_HVACPrestartFunction :  
STRUCT  
    rOutsideTemp      : ARRAY[1..10] OF REAL; (*[degC]*)  
    rRoomTempChange  : ARRAY[1..10] OF REAL; (*[degK/min]*)  
END_STRUCT  
END_TYPE
```

rOutsideTemp / rRoomTempChange : The 10 node points (outdoor temperature in degree Celsius to room temperature change in degree Kelvin / minute), which are passed.

7 Appendix

7.1 Calculation of switching time when changing sequence

Example of the determination of the switching time from one sequence to the next.



$r_{XW}=1$
 $r_{DeadBand}=10$

$\Delta t = 10 \text{ s}$

t_1

t_2




$r_{XW}=2$
 $r_{DeadBand}=10$

$\Delta t = 5 \text{ s}$

t_1

t_2

7.2 Example project

Download	Required library
https://infosys.beckhoff.com/content/1033/tcplclibhvac/Resources/11659726219/.zip 	TcHVAC.lib

7.3 VAR_GLOBAL CONSTANT

```

VAR_GLOBAL CONSTANT
  g_tHVACWriteBackupDataTime : TIME := T#5s;
  g_udiMaxNoOfBytesInStruct  : UDINT := 16;      (* Size in number of bytes *)
  g_udiMaxSizeOfString       : UDINT := 256;      (* Size in number of characters(as in string de
  clARATION) + 1 byte for termination*)
  g_iMaxNoOfScale_nPoint    : INT := 60;        (* FB_HVACScale_nPoint*)
  g_iMaxNumberOfSteps       : INT := 32;        (*FB_HVACI_CtrlStep/FB_HVACPowerRangeTable*)
  g_iMinNumberOfSteps       : INT := 0;        (*FB_HVACI_CtrlStep/FB_HVACPowerRangeTable*)
  g_iMaxNumberOfProfiles    : INT := 16;        (*FB_HVACPowerRangeTable*)
  g_iMinNumberOfProfiles    : INT := 1;        (*FB_HVACPowerRangeTable*)
  g_iMaxNumberOfAggregates  : INT := 6;        (*FB_HVACPowerRangeTable*)
  g_iMinNumberOfAggregates  : INT := 1;        (*FB_HVACPowerRangeTable*)
  g_rYMin                   : REAL := -1000;    (*FB_HVACPowerRangeTable*)
  g_rYMax                   : REAL := 1000;     (*FB_HVACPowerRangeTable*)
  g_iAggregateMinNumberOfSteps : INT := 0;      (*FB_HVACPowerRangeTable*)
  g_iAggregateMaxNumberOfSteps : INT := 6;      (*FB_HVACPowerRangeTable*)
  g_udiMaxSec               : UDINT := 4294967; (*FB_HVACI_CtrlStep / FB_HVACPowerRangeTable*)
  g_iMinNumberOfSequences    : INT := 1;        (*FB_HVAC2PointCtrlSequence*)
  g_iMaxNumberOfSequences    : INT := 16;       (*FB_HVAC2PointCtrlSequence*)
  g_iNumOfCmdCtrl_8         : INT := 8;        (*FB_HVACCmdCtrl_8*)
END_VAR

```

7.4 Table of sequence controller operating modes

Operating modes

A further special feature of the sequence controller is its control by the Enum E_HVACSequenceCtrlMode [▶ 522]

Not only is the controller enabled by the Enum E_HVACSequenceCtrlMode [▶ 522], it also transmits the operating mode of the air conditioning system to the controller blocks in the sequence. Depending on the operating modes, each sequence controller hence reacts specially to the Enum E_HVACSequenceCtrlMode [▶ 522] as illustrated in the table.

Value of: E_HVACSe- quenceC- trlMode	0 (Stop)	1 (On)	2 (night cooling)	3 (support operation)	4 (overheat- ing protec- tion)	5 (night cooling and overheating protection)
FB_HVACMa- sterSequenc eCtrl Master controller	disabled 0 %	Enable	disabled 0 %	Enable rY=ZuluftMax .Temp	Enable rY=ZuluftMin. Temp	Enable rY=ZuluftMin. Temp
FB_HVACPI DPreHeating Pre-heater	disabled 0 %	Enable	disabled 0 %	Enable 0 % - 100 %	disabled 0 %	disabled 0 %
FB_HVACPI DReHeating Re-heater	disabled 0 %	Enable	disabled 0 %	disabled 0 %	disabled 0 %	disabled 0 %

Value of: E_HVACSe- quenceC- trlMode	0 (Stop)	1 (On)	2 (night cooling)	3 (support operation)	4 (overheat- ing protec- tion)	5 (night cooling and overheating protection)
FB_HVACPI DCooling Cooler	disabled 0 %	Enable	disabled 0 %	disabled 0 %	Enable 0 % - 100 %	Enable 0 % - 100 %
FB_HVACPI DEnergyRec overy Heat recovery	disabled 0 %	Enable	disabled 0 %	disabled 0 %	disabled 0 %	disabled 0 %
FB_HVACPI DMixedAir Mixed air flaps	disabled 0 %	Enable	max. outside air rate 100 %	0 % air circulation only	0 % air circulation only	max. outside air rate 100 %

More Information:
www.beckhoff.com/ts8000

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

