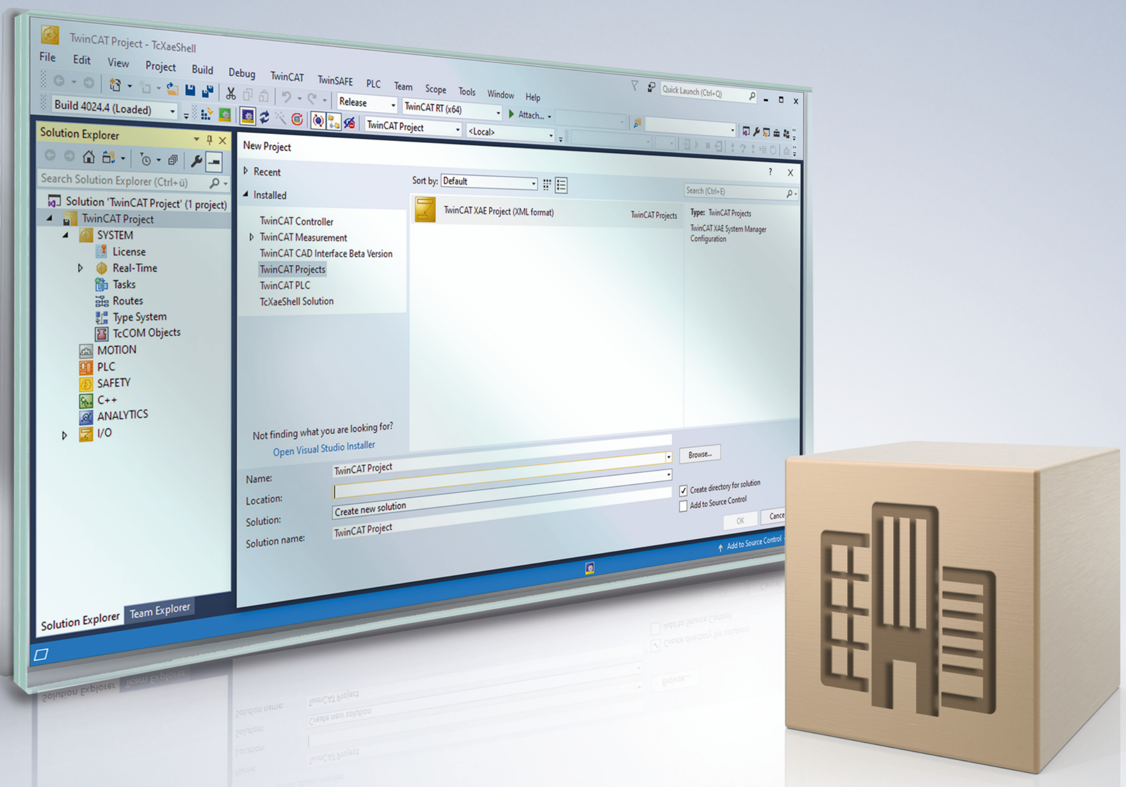
BECKHOFF New Automation Technology

Manual | EN

TF8000

TwinCAT 3 | HVAC



2024-04-08 | Version: 1.2.1

Table of contents

1 Foreword

1.1 Notes on the documentation

This description is intended exclusively for trained specialists in control and automation technology who are familiar with the applicable national standards.
For installation and commissioning of the components, it is absolutely necessary to observe the documentation and the following notes and explanations.
The qualified personnel is obliged to always use the currently valid documentation.

The responsible staff must ensure that the application or use of the products described satisfies all 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 notice.
No claims to modify 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 and licensed trademarks of Beckhoff Automation GmbH.
If third parties make use of designations or trademarks used in this publication for their own purposes, this could infringe upon the rights of the owners of the said designations.

Patents

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

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

Copyright

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

ℹ 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 Overview

Comfort, energy saving, low investment and running costs, and therefore a quick return of investment are the ambitious aims of a building automation system. A basic prerequisite therefore is an integrated, coordinated control system for the automation of all technical building services.

By using the HVAC library, all PLC programs, including the central heating plant, the air conditioning plant and the room automation functions can be programmed with TwinCAT PLC Control and are then available as function blocks within the building automation library.
The Beckhoff system integrator has a wide range of established and approved functions available. This simplifies the implementation of the complex and interoperable building automation functions and reduces the engineering effort. In addition, the life cycle costs of the system are reduced, since the effort for servicing and maintenance can focus on one and the same programming software and hardware platform.

With the room automation functions for air conditioning, lighting and shading devices it is possible to configure systems the meet the highest energy efficiency standard of EN 15232 Class A.

Moreover, the object-oriented encapsulation of the building automation functions offers the following benefits:

- 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.
- Straightforward extension 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, control and regulating systems for the technical equipment of buildings.

2.2 User requirement profile

The user of this library requires basic knowledge of the following.

- TwinCAT PLC
- TwinCAT System
- PC and network knowledge
- Structure and properties of the Beckhoff Embedded PC and its Bus Terminal system.
- 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

Requirements

Development environment	Target platform	TwinCAT PLC function to be implemented	TwinCAT Standard PLC libraries to be implemented

TwinCAT 3.1 build 4022.16 or higher	PC or CX	Tc2_HVAC	Tc2_Coupler;
			Tc2_IoFunctions;
			Tc2_Math;
			Tc2_Standard;
			Tc2_System;
			Tc2_Utilities

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.
The IN_OUT variables are monitored for changes in their values for automatic saving after a parameter change. 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_HVACPersistantDataHandling. Writing of the IN_OUT variables is deactivated if *eDataSecurityType:= eHVACDataSecurityType_Idle*.

Nevertheless, the variables must be located 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 instancing of the function block FB_HVACNOVRAMDataHandling, 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 premissible 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 General Information

Further libraries required

For PC systems and Embedded PCs (CXxxxx):

- Tc2_Coupler
- Tc2_IOFunctions
- Tc2_Math
- Tc2_Standard
- Tc2_System
- Tc2_Utilities

4 Integration in TwinCAT

4.1 System requirements

Technical data	Requirement
Operating system	Windows 7/10, Windows Embedded Standard 7, Windows CE 7
Target platform	PC architecture (x86, x64 or ARM)
TwinCAT version	TwinCAT 3.1 build 4022.16 or higher
Required TwinCAT setup level	TwinCAT 3 XAE, XAR
Required TwinCAT license	TF8000 TC3 HVAC

4.2 Installation

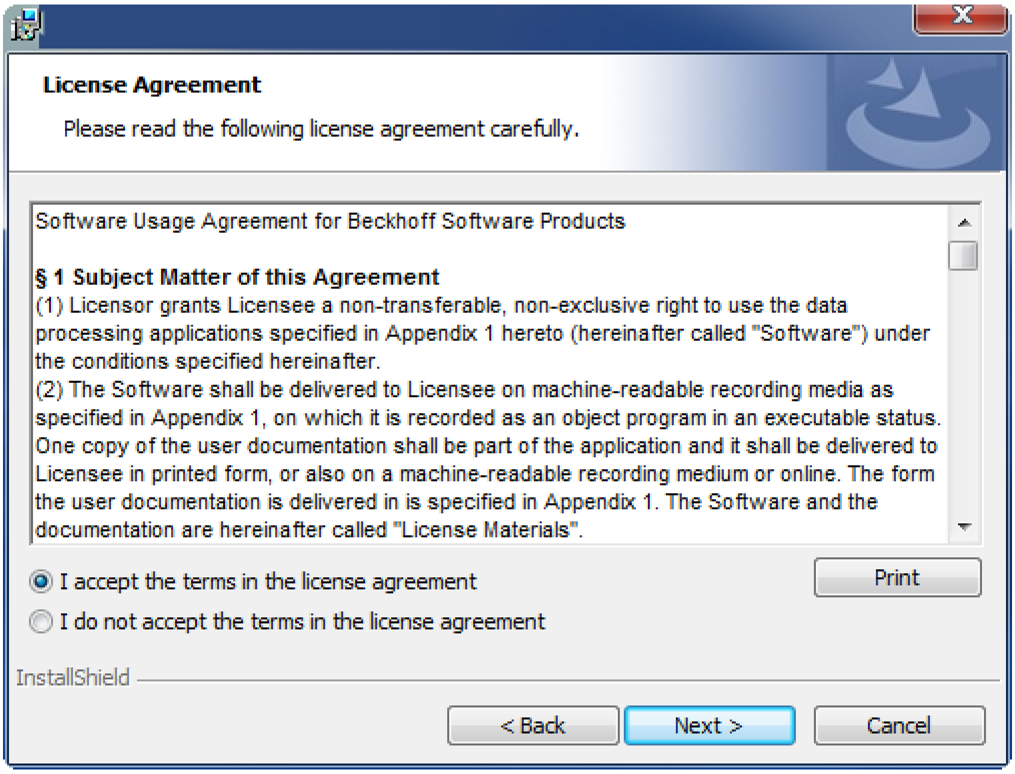
The following section describes how to install the TwinCAT 3 Function for Windows-based operating systems.

✓ The TwinCAT 3 Function setup file was downloaded from the Beckhoff website.

1. Run the setup file as administrator. To do this, select the command **Run as administrator** in the context menu of the file.

   ⇨ The installation dialog opens.

2. Accept the end user licensing agreement and click **Next**.



License Agreement

Please read the following license agreement carefully.

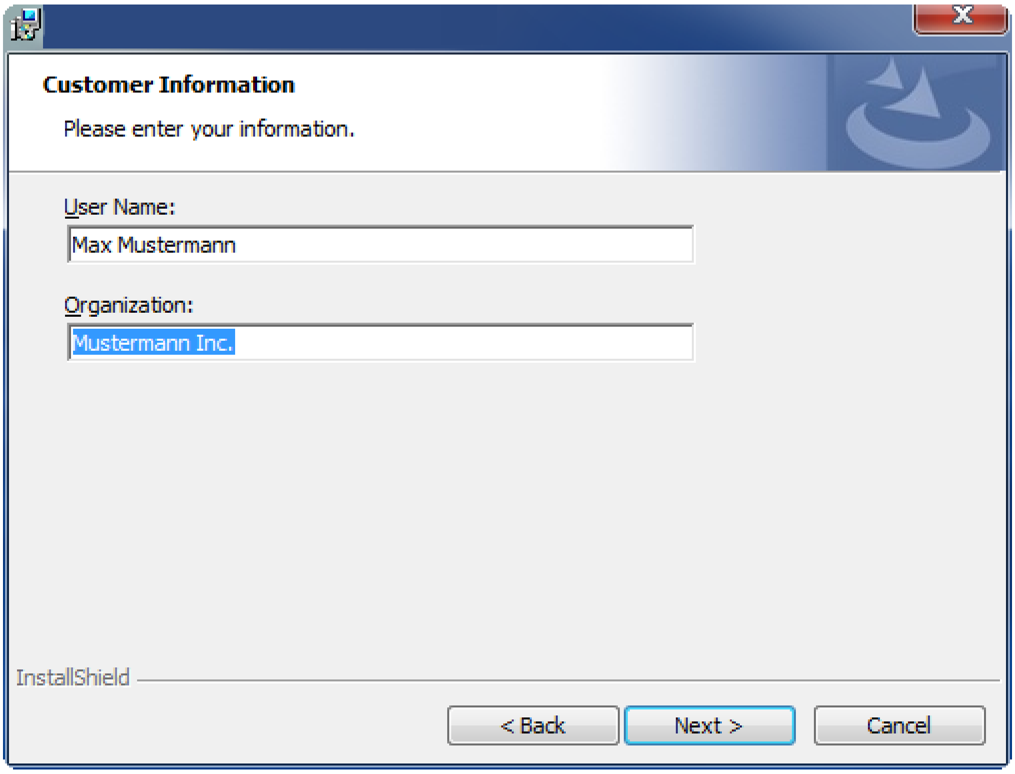
Software Usage Agreement for Beckhoff Software Products

§ 1 Subject Matter of this Agreement
(1) Licensor grants Licensee a non-transferable, non-exclusive right to use the data processing applications specified in Appendix 1 hereto (hereinafter called "Software") under the conditions specified hereinafter.
(2) The Software shall be delivered to Licensee on machine-readable recording media as specified in Appendix 1, on which it is recorded as an object program in an executable status. One copy of the user documentation shall be part of the application and it shall be delivered to Licensee in printed form, or also on a machine-readable recording medium or online. The form the user documentation is delivered in is specified in Appendix 1. The Software and the documentation are hereinafter called "License Materials".

● I accept the terms in the license agreement

○ I do not accept the terms in the license agreement

3. Enter your user data.



Customer Information

Please enter your information.

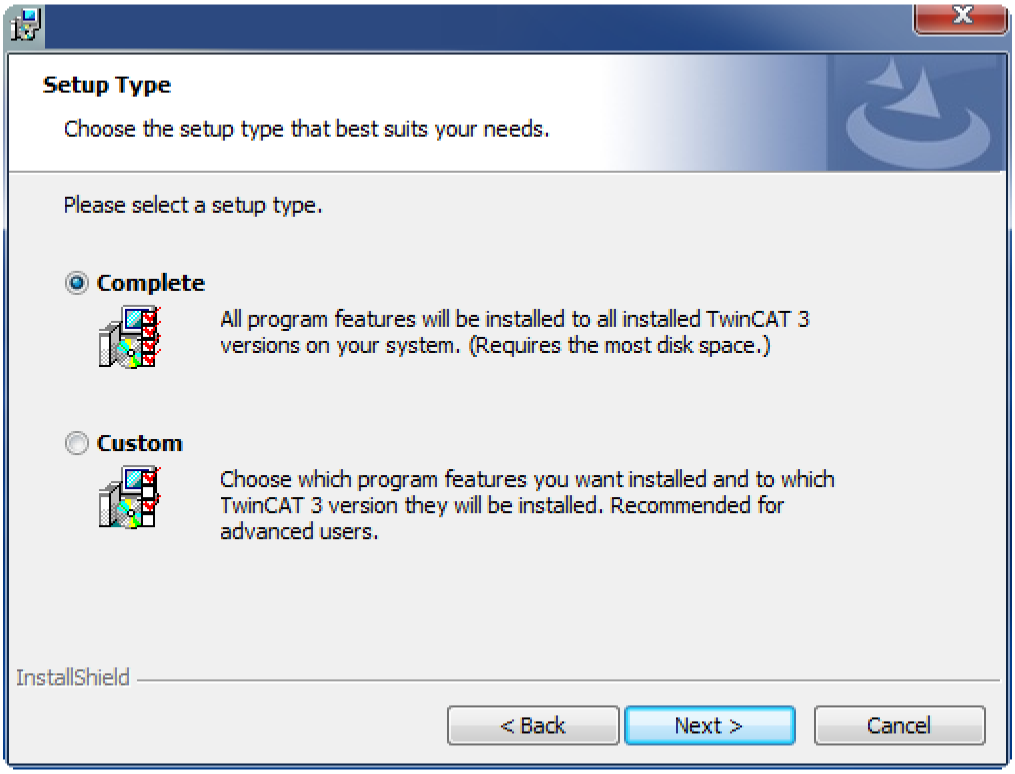
User Name:

Max Mustermann

Organization:

Mustermann Inc.

4. If you want to install the full version of the TwinCAT 3 Function, select **Complete** as installation type. If you want to install the TwinCAT 3 Function components separately, select **Custom**.



Setup Type

Choose the setup type that best suits your needs.

Please select a setup type.

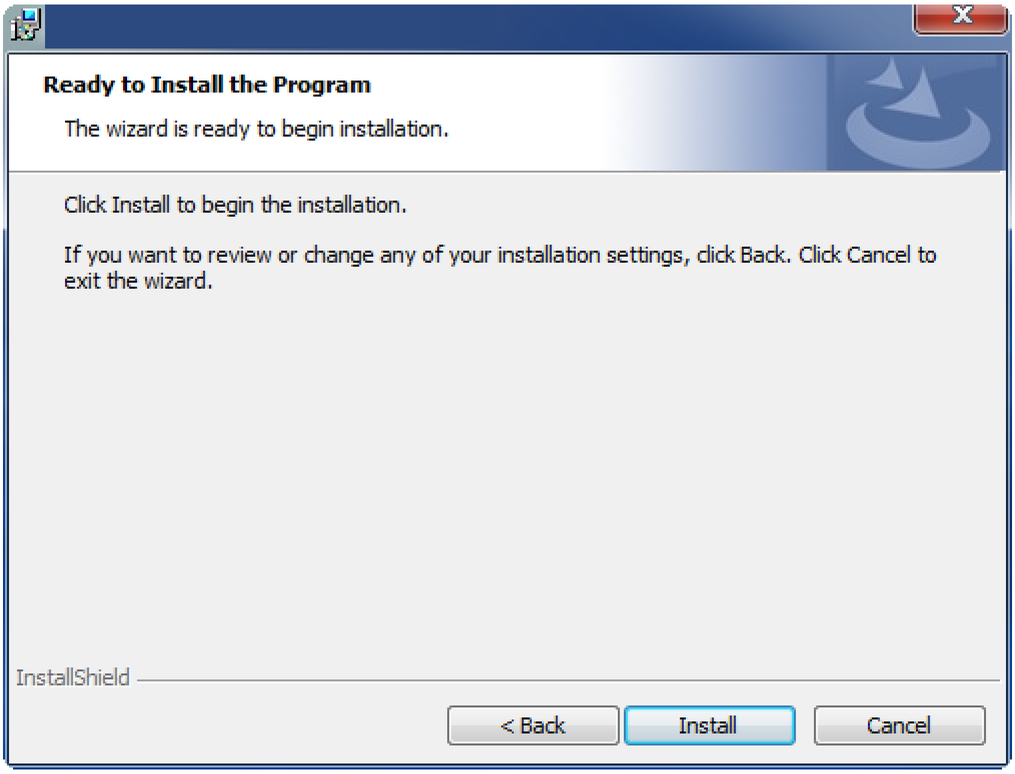
● **Complete**

All program features will be installed to all installed TwinCAT 3 versions on your system. (Requires the most disk space.)

○ **Custom**

Choose which program features you want installed and to which TwinCAT 3 version they will be installed. Recommended for advanced users.

5. Select **Next**, then **Install** to start the installation.



Ready to Install the Program

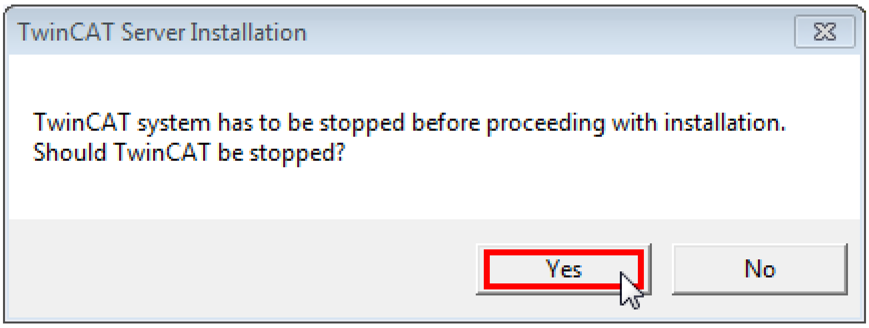
The wizard is ready to begin installation.

Click Install to begin the installation.

If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.

⇨ A dialog box informs you that the TwinCAT system must be stopped to proceed with the installation.

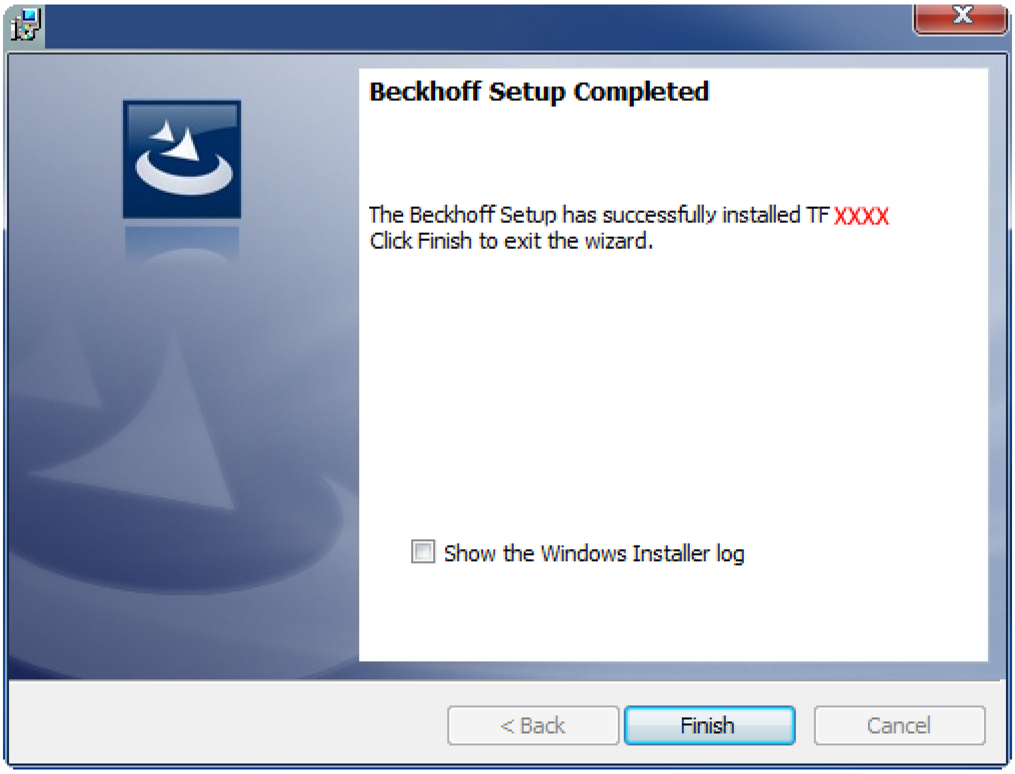
6. Confirm the dialog with **Yes**.



TwinCAT Server Installation

TwinCAT system has to be stopped before proceeding with installation.
Should TwinCAT be stopped?

7. Select **Finish** to exit the setup.



Beckhoff Setup Completed

The Beckhoff Setup has successfully installed TF XXXX
Click Finish to exit the wizard.

□ Show the Windows Installer log

⇨ The TwinCAT 3 Function has been successfully installed and can be licensed (see Licensing [▶ 15]).

4.3 Licensing

The TwinCAT 3 function can be activated as a full version or as a 7-day test version. Both license types can be activated via the TwinCAT 3 development environment (XAE).

Licensing the full version of a TwinCAT 3 Function

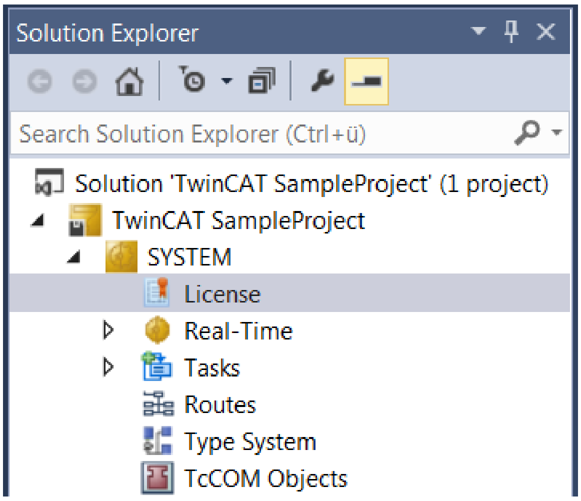
A description of the procedure to license a full version can be found in the Beckhoff Information System in the documentation "TwinCAT 3 Licensing".

Licensing the 7-day test version of a TwinCAT 3 Function

A 7-day test version cannot be enabled for a TwinCAT 3 license dongle.

1. Start the TwinCAT 3 development environment (XAE).
2. Open an existing TwinCAT 3 project or create a new project.
3. If you want to activate the license for a remote device, set the desired target system. To do this, select the target system from the **Choose Target System** drop-down list in the toolbar.
   ⇨ The licensing settings always refer to the selected target system. When the project is activated on the target system, the corresponding TwinCAT 3 licenses are automatically copied to this system.

4. In the **Solution Explorer**, double-click **License** in the **SYSTEM** subtree.



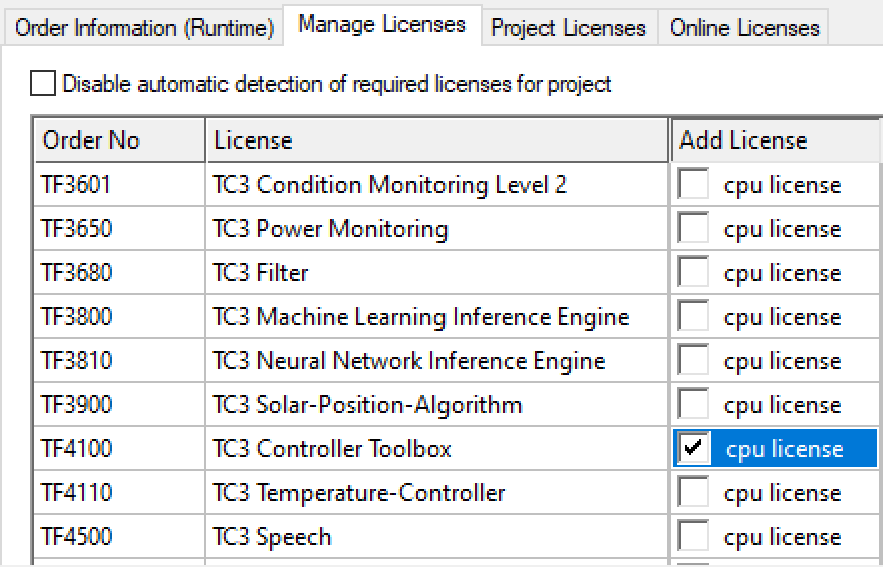
Solution Explorer

Search Solution Explorer (Ctrl+ü)

- Solution 'TwinCAT SampleProject' (1 project)
  - TwinCAT SampleProject
    - SYSTEM
      - License
      - Real-Time
      - Tasks
      - Routes
      - Type System
      - TcCOM Objects

⇨ The TwinCAT 3 license manager opens.

5. Open the **Manage Licenses** tab. In the **Add License** column, check the check box for the license you want to add to your project (e.g. "TF4100 TC3 Controller Toolbox").



Order Information (Runtime) | Manage Licenses | Project Licenses | Online Licenses

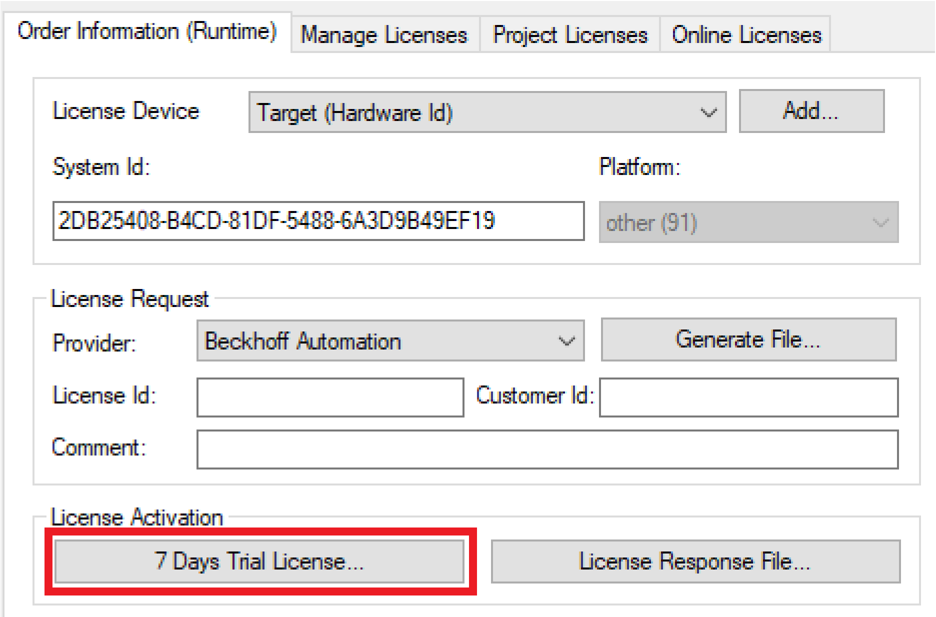
□ Disable automatic detection of required licenses for project

Order No	License	Add License
TF3601	TC3 Condition Monitoring Level 2	□ cpu license
TF3650	TC3 Power Monitoring	□ cpu license
TF3680	TC3 Filter	□ cpu license
TF3800	TC3 Machine Learning Inference Engine	□ cpu license
TF3810	TC3 Neural Network Inference Engine	□ cpu license
TF3900	TC3 Solar-Position-Algorithm	□ cpu license
TF4100	TC3 Controller Toolbox	☑ cpu license
TF4110	TC3 Temperature-Controller	□ cpu license
TF4500	TC3 Speech	□ cpu license

6. Open the **Order Information (Runtime)** tab.

⇨ In the tabular overview of licenses, the previously selected license is displayed with the status “missing”**.**

7. Click **7-Day Trial License...** to activate the 7-day trial license.



Order Information (Runtime) | Manage Licenses | Project Licenses | Online Licenses

License Device: Target (Hardware Id) — Add...

System Id: 2DB25408-B4CD-81DF-5488-6A3D9B49EF19

Platform: other (91)

License Request

Provider: Beckhoff Automation — Generate File...

License Id:

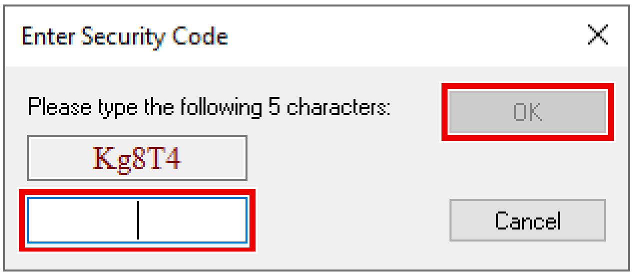
Customer Id:

Comment:

License Activation

7 Days Trial License... — License Response File...

⇨ A dialog box opens, prompting you to enter the security code displayed in the dialog.



Enter Security Code

Please type the following 5 characters:

Kg8T4

OK

Cancel

8. Enter the code exactly as it is displayed and confirm the entry.
9. Confirm the subsequent dialog, which indicates the successful activation.
   ⇨ In the tabular overview of licenses, the license status now indicates the expiry date of the license.
10. Restart the TwinCAT system.

⇨ The 7-day trial version is enabled.

5 Programming

5.1 POUs

5.1.1 Overview

HVAC actuators

Name	Description
FB_HVAC2PointActuator [▶ 25]	Control of two-point dampers or valves
FB_HVAC3PointActuator [▶ 27]	Control of three-point dampers or valves
FB_HVACCirculationPump [▶ 29]	Control of pumps
FB_HVACCirculationPumpEx [▶ 32]	Control of pumps; the control of the output bPump has been changed compared to FB_HVACCirculationPump
FB_HVACMotor1Speed [▶ 43]	Control of single-speed drives
FB_HVACMotor2Speed [▶ 47]	Control of two-speed drives
FB_HVACMotor3Speed [▶ 52]	Control of three-speed drives
FB_HVACRedundancyCtrl [▶ 62]	Control of up to eight actuators depending on the runtime; uses an internal timer for determining the runtime
FB_HVACRedundancyCtrlEx [▶ 65]	Control of up to eight actuators depending on the runtime; the runtimes of the actuators must be created externally as hour values.

HVAC analog modules

Name	Description
FB_HVACAnalogInput [▶ 67]	Acquisition of analog input signals
FB_HVACAnalogOutput [▶ 70]	Control of analog actuators
FB_HVACAnalogOutputEx [▶ 72]	Control of analog actuators with integrated scaling function
FB_HVACAnalogTo3Point [▶ 76]	Conversion of an analog control signal into a three-point step signal
FB_HVACConfigureKL32xx [▶ 82]	Parameterization of the connected sensor type on an input channel from the PLC
FB_HVACScale [▶ 86]	Scaling function block
FB_HVACScale_nPoint [▶ 87]	Simulation of characteristic curves in the PLC
FB_HVACScaleXX [▶ 92]	Mapping of characteristic curves with 2, 4 or 7 interpolation points.
FB_HVACTemperatureCurve [▶ 93]	Different temperature characteristic curves that can be mapped are stored inside the FB.
FB_HVACTemperatureSensor [▶ 99]	Acquisition of temperature values in 1/10 °C, tailored to the KL320x Bus Terminals
FB_HVACTemperatureSensorEx [▶ 102]	Acquisition of temperature values in 1/10°C, tailored to the KL320x Bus Terminals; without 2nd order filter, but with a smoothing function
FB_HVACTemperatureSensorEx2 [▶ 105]	Acquisition of temperature values in 1/10°C or 1/100°C, tailored to the KL320x Bus Terminals; without 2nd order filter, but with a smoothing function

HVAC controller

Name	Description
FB_HVAC2PointCtrl [▸ 109]	2-point controller
FB_HVACI_CtrlStep [▸ 112]	I-transfer element for step control in conjunction with FB_HVACPowerRangeTable
FB_HVACI_CtrlStepEx [▸ 119]	I-transfer element for step control in conjunction with FB_HVACPowerRangeTable
FB_HVACPIDCtrl [▸ 127]	PID controller
FB_HVACPIDCtrl_Ex [▸ 130]	PID controller (extended)
FB_HVACPowerRangeTable [▸ 135]	Power range table for the sequential control of power generators such as boilers or refrigerating machines
FB_HVAC2PointCtrlSequence [▸ 149]	2-point sequence controller
FB_HVACBasicSequenceCtrl [▸ 157]	general sequence controller
FB_HVACMasterSequenceCtrl [▸ 160]	Master controller in a VAC system
FB_HVACPIDCooling [▸ 163]	PID cooling controller
FB_HVACPIDDehumidify [▸ 166]	PID dehumidification controller
FB_HVACPIDEnergyRecovery [▸ 169]	PID heat recovery controller
FB_HVACPIDHumidify [▸ 172]	PID humidification controller
FB_HVACPIDMixedAir [▸ 176]	PID mixed air chamber controller
FB_HVACPIDPreHeating [▸ 179]	PID pre-heating controller
FB_HVACPIDReHeating [▸ 182]	PID reheating controller

Air conditioning room function

Name	Description
FB_BAREnergyLevel [▸ 185]	This function block is for the adaptation of the supply of energy for the use of the building.
FB_BARFanCoil [▸ 187]	This function block maps a 3-speed fan with the corresponding switching hysteresis.
FB_BARFctSelection [▸ 190]	This function block is for enabling room heating or room cooling.
FB_BARSetpointRoom [▸ 193]	This function block assigns a setpoint for cooling operation and another for heating operation to each of the four energy levels.

Room function controller

Name	Description
FB_BARPICtrl [▸ 198]	simple PI Controller

Room functions; lighting

Name	Description
FB_BARAutomaticLight [▸ 204]	Occupancy-controlled automatic light with switch-off delay
FB_BARConstantLightControl [▸ 207]	Constant light controller with continuous output of analog values
FB_BARDaylightControl [▸ 213]	Daylight control without dimming procedures
FB_BARLightActuator [▸ 217]	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.
FB_BARLightCircuit [▸ 220]	Simple light switching circuit without dimming function
FB_BARLightCircuitDim [▸ 222]	Simple light switching circuit with dimming function
FB_BARStairwellAutomatic [▸ 225]	Stairwell lighting with preliminary warning sequence
FB_BARTwilightAutomatic [▸ 228]	Twilight switching

Sun protection room function

Name	Description
FB_BARBlindPositionEntry [▸ 248]	Sun protection function: Input of blind positions.
FB_BARDelayedHysteresis [▸ 251]	External brightness check with delay and hysteresis
FB_BARFacadeElementEntry [▸ 252]	Shading correction: Input of facade elements per function block.
FB_BARReadFacadeElementList [▸ 257]	Shading correction: Input of facade elements via data list (csv).
FB_BARReadShadingObjectsList [▸ 261]	Shading correction: Input of shading objects via data list (csv).
FB_BARShadingCorrection [▸ 264] / FB_BARShadingCorrectionSouth [▸ 267]	Shading correction function block (FB_BarShadingCorrectionSouth applies to the southern hemisphere)
FB_BARShadingObjectsEntry [▸ 270]	Shading correction: Input of shading objects per function block.
FB_BARSunblindActuator [▸ 273] / FB_BARSunblindActuatorEx [▸ 279]	Blind actuator
FB_BARSunblindEvent [▸ 284]	Output of a specified blind position and angle in percent
FB_BARSunblindPrioritySwitch [▸ 286]	Priority controller
FB_BARRollerBlind [▸ 287]	Roller shutter actuator
FB_BARSunblindScene [▸ 289]	Manual operation with scene selection and programming
FB_BARSunblindSwitch [▸ 292]	Manual operation
FB_BARSunblindThermoAutomatic [▸ 294]	Thermo-automatic function
FB_BARSunblindTwilightAutomatic [▸ 301]	Automatic twilight function
FB_BARSunblindWeatherProtection [▸ 303]	Weather protection function
FB_BARSunProtectionEx [▸ 306]	Sun protection function, see Overview of automatic sun protection (shading correction) [▸ 233]
FB_BARWithinRangeAzimuth [▸ 309]	Verification of valid sun position and sun direction range (azimuth angle)
FB_BARWithinRangeElevation [▸ 311]	Verification of valid sun position and sun elevation range (elevation angle)

HVAC Setpoint modules

Name	Description
FB_HVACHeatingCurve [▶ 322]	Calculation the supply temperature depending on the outside temperature with four base
FB_HVACHeatingCurveEx [▶ 324]	Calculation the supply temperature depending on the outside temperature
FB_HVACOutsideTempDamped [▶ 328]	Calculation the damped outside temperature
FB_HVACSetpointHeating [▶ 329]	Control an heating circuit with different operating modes
FB_HVACSetpointRamp [▶ 331]	Moving setpoint ramp
FB_HVACSummerCompensation [▶ 333]	summer compensation

HVAC special functions

Name	Description
FB_HVACAirConditioning2Speed [▸ 336]	Control of air handling units with two-speed fans
FB_HVACAlarm [▸ 340]	Alarm function block
FB_HVACAntiBlockingDamper [▸ 341]	Blocking protection for damper drives
FB_HVACAntiBlockingPump [▸ 343]	Blocking protection for pumps
FB_HVACBlink [▸ 346]	Flash sequence
FB_HVACCmdCtrl_8 [▸ 347]	With this function block, individual units in a system can be sequentially switched on or off in a certain order. *FB_HVACCmdCtrl_8* can be used as the starting function block of a ventilation system.
FB_HVACCmdCtrlSystem1Stage [▸ 355]	Single-stage system switch
FB_HVACCmdCtrlSystem2Stage [▸ 364]	Two-stage system switch
FB_HVACConvertEnum [▸ 377]	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.
FB_HVACEnthalpy [▸ 378]	Determination of the dew point, the specific enthalpy and the absolute humidity.
FB_HVACFixedLimit [▸ 380]	Limit switch
FB_HVACFreezeProtectionHeater [▸ 382]	Frost protection monitoring
FB_HVACMux8 [▸ 59]	evaluates the FIFO memory of FB_HVACRedundancyCtrlEx
FB_HVACMUX_INT_16 [▸ 389]	Function block contains two different types of multiplexer
FB_HVACMUX_INT_8 [▸ 385]	Function block contains two different types of multiplexer
FB_HVACMUX_REAL_16 [▸ 397]	Function block contains two different types of multiplexer
FB_HVACMUX_REAL_8 [▸ 393]	Function block contains two different types of multiplexer
FB_HVACOverwriteAnalog [▸ 400]	analog override in manual operation
FB_HVACOverwriteDigital [▸ 401]	digital override in manual operation
FB_HVACPowerMeasurementKL3403 [▸ 402]	Acquisition of the measured/calculated values of the three-phase power measurement terminal (KL3403)
FB_HVACPowerMeasurementKL3403Ex [▸ 404]	In comparison with the FB_HVACPowerMeasurementKL3403, the results are output in LREAL format. The output was extended by the frequencies of the three phases.
FB_HVACPriority_INT_16 [▸ 410]	This function block can be used to prioritize events or as a multiplexer
FB_HVACPriority_INT_8 [▸ 406]	This function block can be used to prioritize events or as a multiplexer
FB_HVACPriority_REAL_16 [▸ 419]	This function block can be used to prioritize events or as a multiplexer
FB_HVACPriority_REAL_8 [▸ 414]	This function block can be used to prioritize events or as a multiplexer
FB_HVACOptimizedOff [▸ 433]	Function block for the optimized switch-off (pre-stop) of boilers and air conditioning systems in conjunction with switching time function blocks.
FB_HVACOptimizedOn [▸ 423]	Function block for the optimized switch-on of boilers and air conditioning systems in conjunction with switching time function blocks.
FB_HVACTempChangeFunctionEntry [▸ 443]	Function block for the input of the interpolation points of the pre-start function
FB_HVACPWM [▸ 444]	PWM function block
FB_HVACStartAirConditioning [▸ 447]	Start program of an air handling unit
FB_HVACSummerNightCooling [▸ 451]	Summer night cooling

Name	Description
FB_HVACSummerNightCoolingEx [▶ 454]	Summer night cooling
FB_HVACTimeCon [▶ 458]	Converts a TIME variable into three UDINT variables (udiSec, udiMin, udiHour)
FB_HVACTimeConSec [▶ 458]	Converts a TIME variable into a UDINT variable (udiSec)
FB_HVACTimeConSecMs [▶ 459]	Converts a TIME variable into two UDINT variables (udiMs, udiSec)
FB_HVACWork [▶ 459]	Recording of operating hours

HVAC Time schedule

Name	Description
FB_HVACScheduler1ch [▶ 461]	Weekly time switch with 1 time switch channel
FB_HVACScheduler7ch [▶ 464]	Weekly time switch with 7 time switch channel
FB_HVACScheduler7TCHandling [▶ 467]	this FB can be used to select and modify an individual line from the data array of a weekly timer
FB_HVACScheduler28ch [▶ 468]	Weekly time switch with 28 time switch channel
FB_HVACScheduler28TCHandling [▶ 472]	this FB can be used to select and modify an individual line from the data array of a weekly timer
FB_HVACSchedulerSpecialPeriods [▶ 474]	Yearly scheduler with day, month and exactly time
FB_HVACSchedulerPublicHolidays [▶ 472]	Yearly scheduler with day and month

HVAC system

Name	Description
FB_HVACGetSystemTime [▶ 477]	Implementation of an internal clock in the TwinCAT PLC
FB_HVACNOVRAMDataHandling [▶ 478]	Data are written fail-safe to the NOVRAM
FB_HVACPersistentDataHandling [▶ 480]	Data are written fail-safe to a file
FB_HVACPersistentDataFileCopy [▶ 482]	Copying of binary files between local or remote TwinCAT PC and the local TwinCAT PC
FB_HVACSetLocalTime [▶ 483]	Sets the local Windows system time and the date of a TwinCAT system
FB_HVACSystemTaskInfo [▶ 485]	Determination of system variables of the task

HVAC backup function blocks

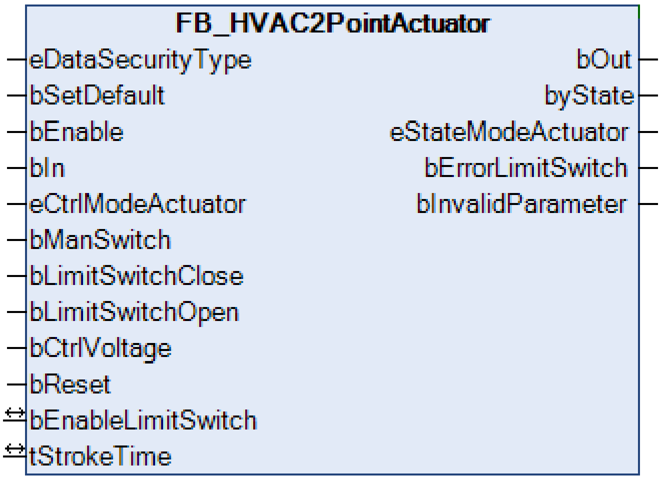
Name	Description
FB_HVACNOVRAM_XX [▶ 486]	FBs for standard data types
FB_HVACPersistent_XX [▶ 486]	FBs for standard data types

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.2 HVAC Actuators

5.1.2.1 FB_HVAC2PointActuator



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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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.

bIn: In automatic mode the actuator is closed by a FALSE and opened by a TRUE.

eCtrlModeActuator: Enum that defines the operation mode (see E_HVAC2PointActuatorMode [▶ 488]).

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: Shows the status of the control of 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 which operation mode the actuator is in (see E_HVAC2PointActuatorMode [▸ 488]).

bErrorLimitSwitch: Goes 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 has occurred during the plausibility check. The message must be acknowledged with *bReset*.

VAR_IN_OUT

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

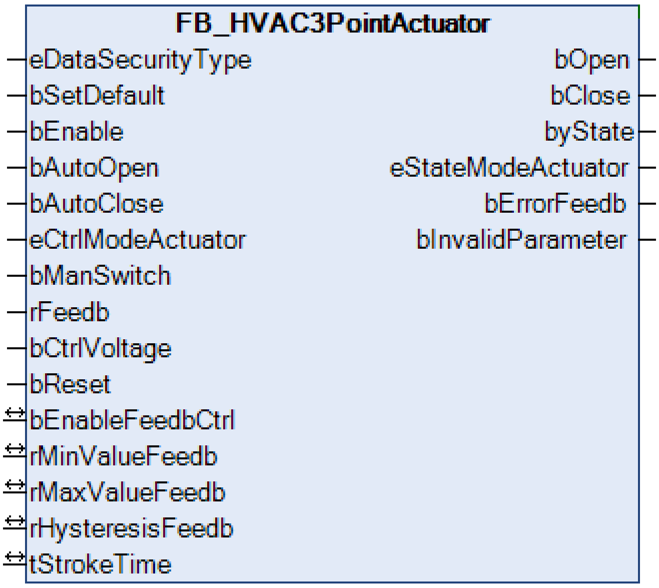
bEnableLimitSwitch: If the input is TRUE, then the function control of the drive is activated by means of the limit switch. The variable is saved persistently.

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.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.2.2 FB_HVAC3PointActuator



This function block serves to control three-point valves or three-point dampers, with or without constant position feedback.
The function block is often used in conjunction with the FB_HVACAnalogTo3Point [▶ 76] function block.

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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 (see E_HVAC3PointActuatorMode [▶ 488]).

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 for controlling 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: Shows the status of the control of the actuator:
byState.0:= Enable
byState.1:= Manual Switch
byState.2:= Enable Feedback Control
byState.3:= Control Voltage
byState.4:= Reset

eStateModeActuator: Indicates which operation mode the actuator is in (see E_HVAC3PointActuatorMode [▶ 488]).

bErrorFeedb: When setting the output *bClose*, the position of the drive must have decreased by at least the value of *rHysteresisFeedb* within the time *tStrokeTime*.
When setting the output *bOpen*, the position of the drive must have increased by at least the value of *rHysteresisFeedb* within the time *tStrokeTime*.
If the actual position is not within the tolerance range within the specified time following a positioning command, this will be indicated by TRUE on the output *bErrorFeedb*. Outputs *bOpen* and *bClose* both go FALSE**.** The fault is acknowledged by a positive edge on the input *bReset*.

bInvalidParameter: TRUE if an error has occurred 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 activated. 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.

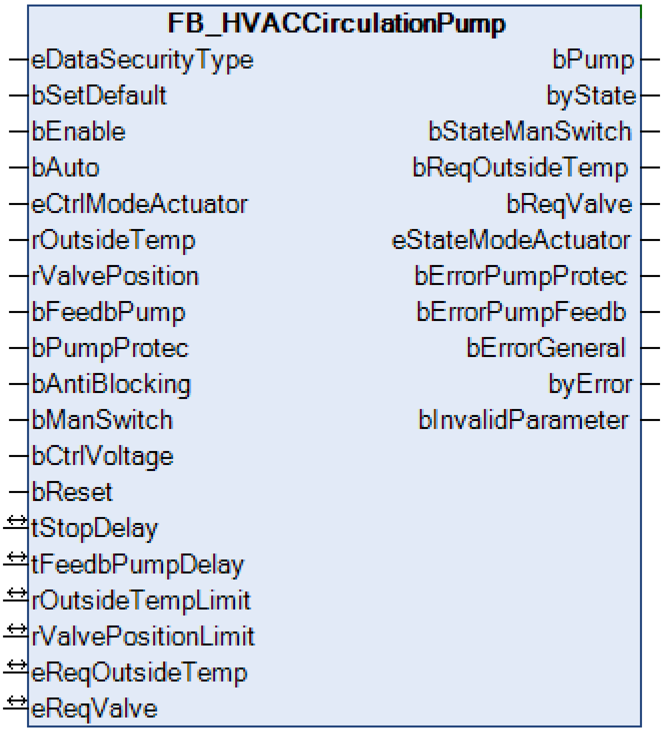
rHysteresisFeedb: Due to the stroke time of the drive, the position feedback always lags in the case of a jump in the set position value. Using the variable *rHysteresisFeedbCtrl*, a range is specified within which the set position value of the actuator can deviate from the feedback signal without the 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 jump of 100 %, at least the stroke time of the drive over its entire range of movement should be entered as a time (0s..3600s). The variable is saved persistently. Preset to 200 s.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.2.3 FB_HVACCirculationPump



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 [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 (see E_HVACActuatorMode [▶ 489]).

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. The output *bPump* is disabled if *bManSwitch*= FALSE. 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 status as a 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: Indicates the state of the manual/emergency switch. A TRUE signals that the manual/ emergency operating level is activated.

bReqOutsideTemp: If the condition for requesting the pump depending on the outside temperature is TRUE, the output variable *bReqOutsideTemp* is TRUE.

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

eStateModeActuator: Indicates which operation mode the actuator is in (see E_HVACActuatorMode [▶ 489]).

bErrorPumpProtec: Pump error.

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 error as a byte.
byError.1 := bInvalidParameter
byError.2 := bErrorGeneral
byError.3 := bErrorPumpProtec
byError.4 := bErrorPumpFeedb

bInvalidParameter: TRUE if an error has occurred 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 given. 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 (see E_HVACRegOutsideTemp [▶ 492]). 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 of *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 (see E_HVACRegValve [▶ 493]). The variable is saved persistently.

The table below shows a summary of all possible combinations:

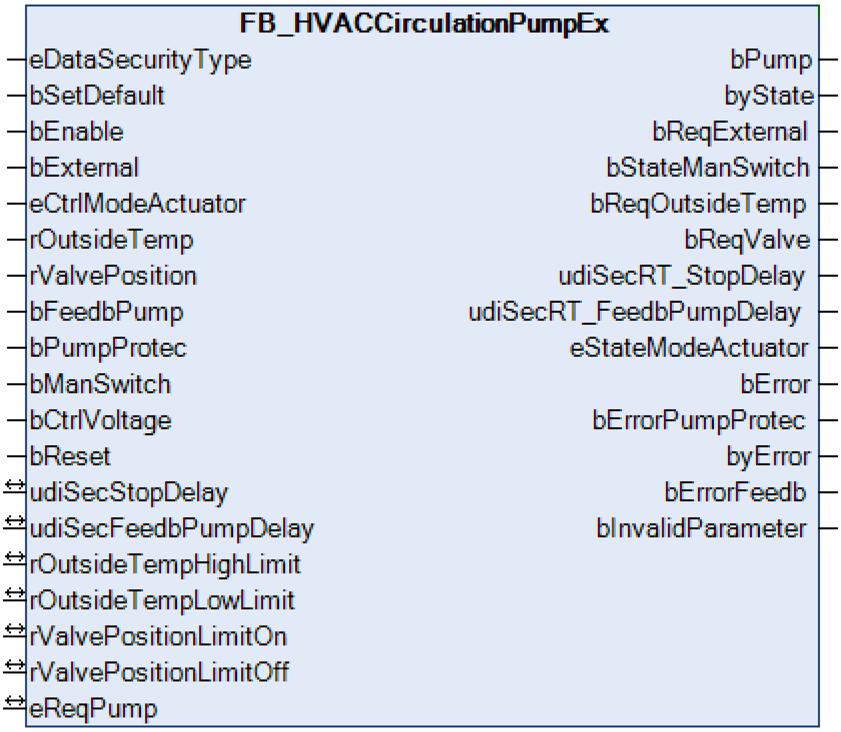
eReqOutsideTemp	eReqValve	Function	Application
OTLowerLimit	NoRequest	outsidetemp lower limit	
OTLowerLimit	OrValvePosHigherLimit	outsidetemp OR valve higher limit	Heating circuit, air heater
OTLowerLimit	AndValvePosHigherLimit	outsidetemp AND valve higher limit	
OTHigherLimit	NoRequest	outsidetemp higher 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.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.2.4 FB_HVACCirculationPumpEx



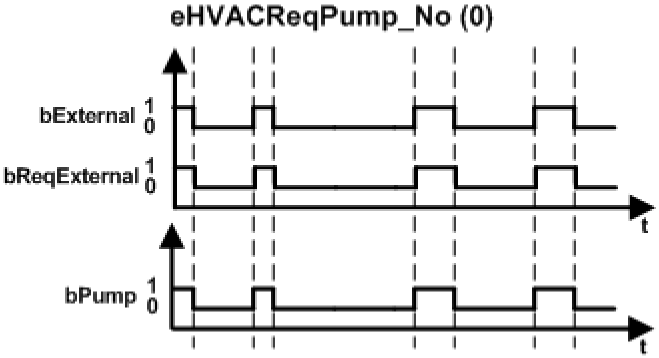
This function block serves to control pumps in HVAC systems.

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

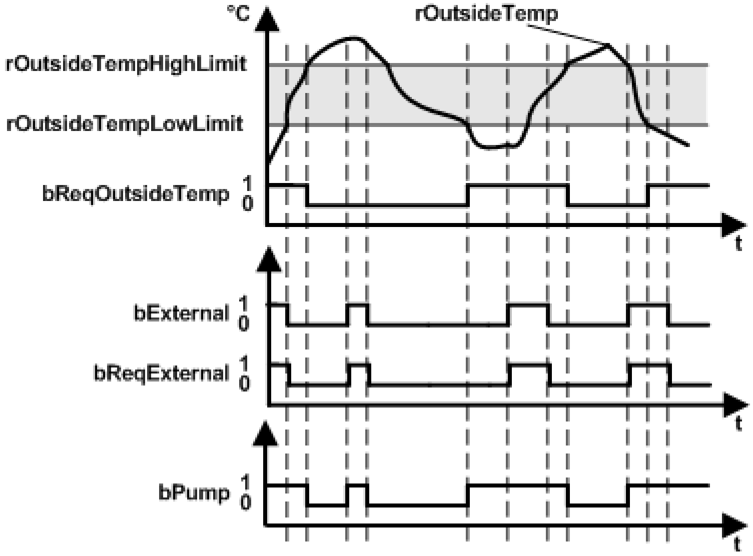
bEnable = TRUE AND *bErrorPumpProtec* = FALSE AND *bManSwitch* = TRUE AND *bCtrlVoltage* = TRUE

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

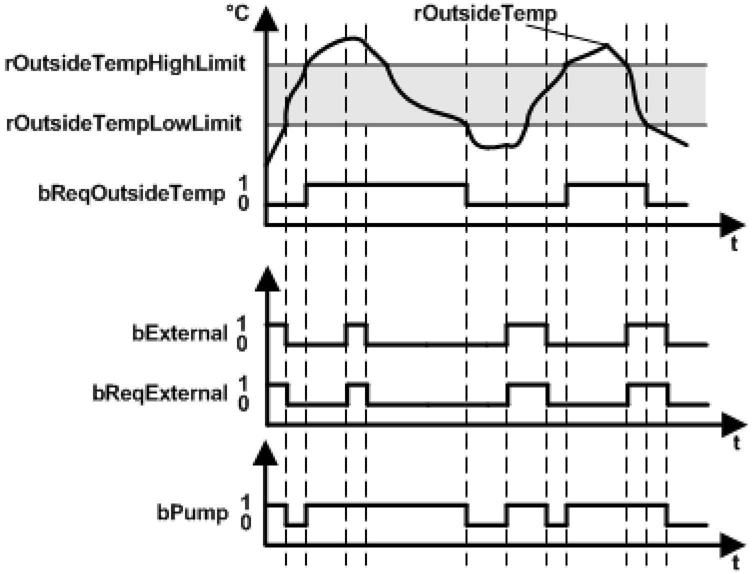
On the basis of the following diagrams it can be seen how the output *bPump* is controlled depending on the Enum *eReqPump*, the input *bExternal*, the outside temperature *rOutSideTemp* and the valve position *rValvePosition*. The switch-on conditions mentioned above must be fulfilled for this, and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



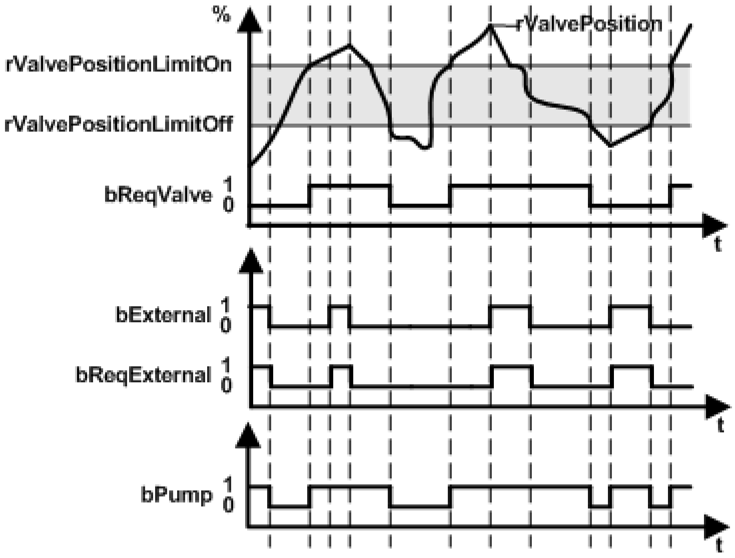
eHVACReqPump_OT_LL (1)



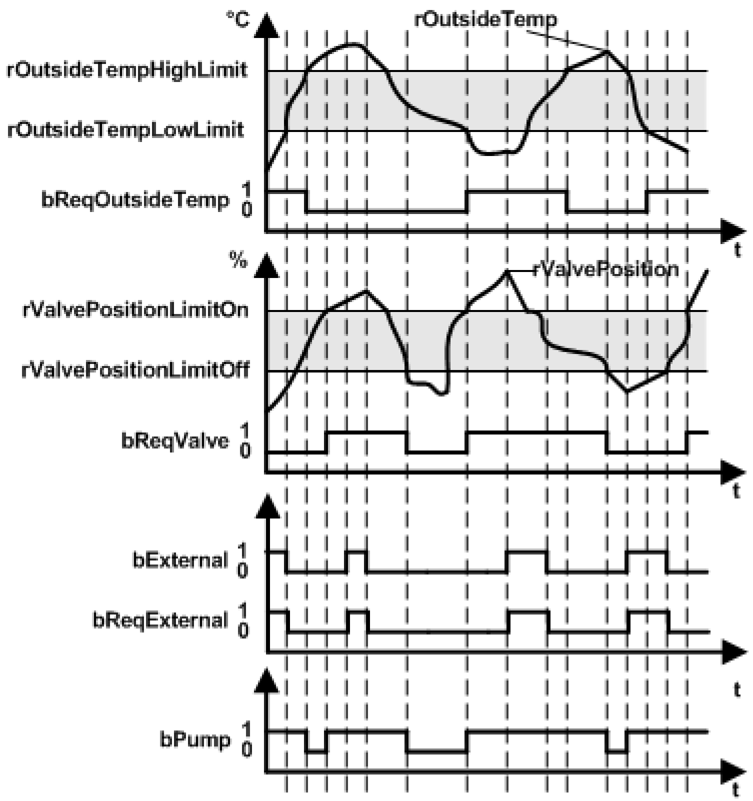
eHVACReqPump_OT_HL(2)



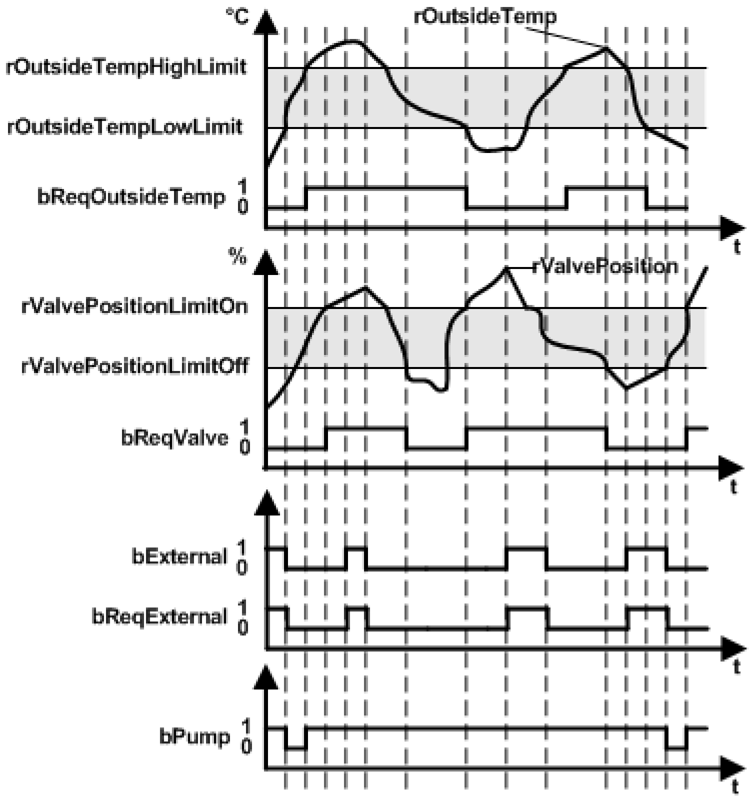
eHVACReqPump_VP (3)

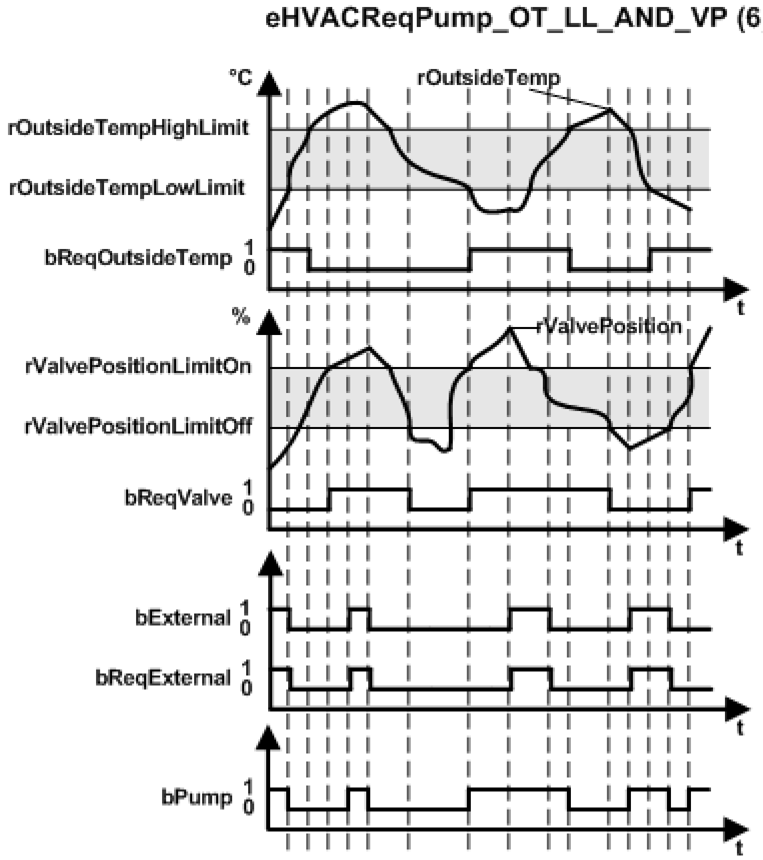


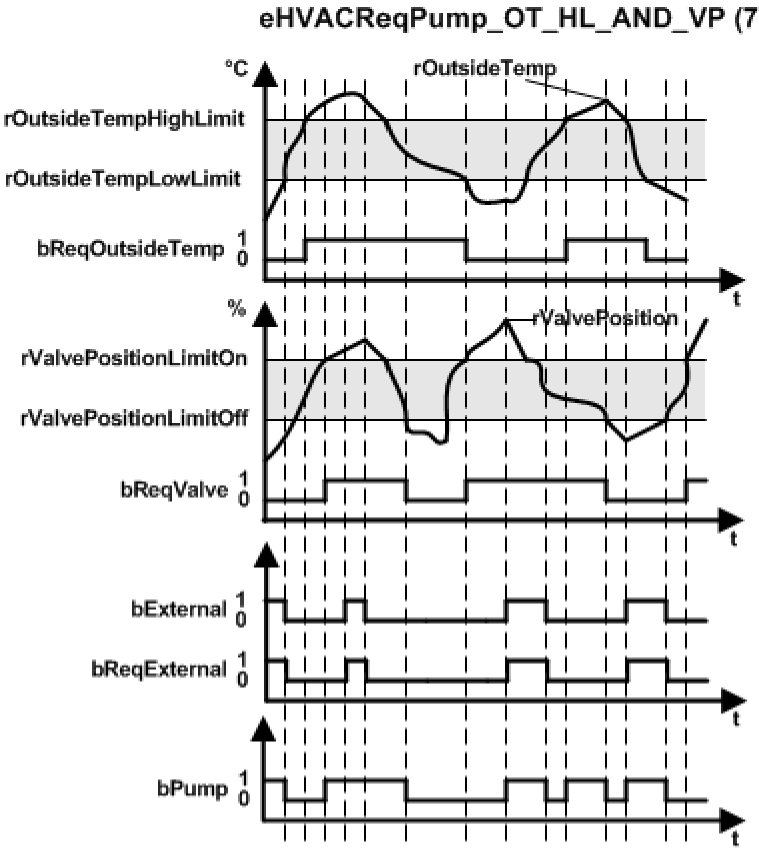
eHVACReqPump_OT_LL_OR_VP (4)



eHVACReqPump_OT_HL_OR_VP (5)







eHVACReqPump_No: There is no request on the part of the Enum to control the pump (see E_HVACReqPump [▸ 492]).

eHVACReqPump_OT_LL: The outside temperature (OT = *rOutsideTemp*) must be lower than *rOutsideTempLowLimit* (LL = Lower Limit) (see E_HVACReqPump [▸ 492]).

eHVACReqPump_OT_HL: The outside temperature (OT = *rOutsideTemp*) must be higher than *rOutsideTempHighLimit* (HL = Higher Limit) (see E_HVACReqPump [▸ 492]).

eHVACReqPump_VP: The valve position (VP = *rValvePosition*) must be larger than *rValvePositionLimitOn* (see E_HVACReqPump [▸ 492]).

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* (see E_HVACReqPump [▸ 492]).

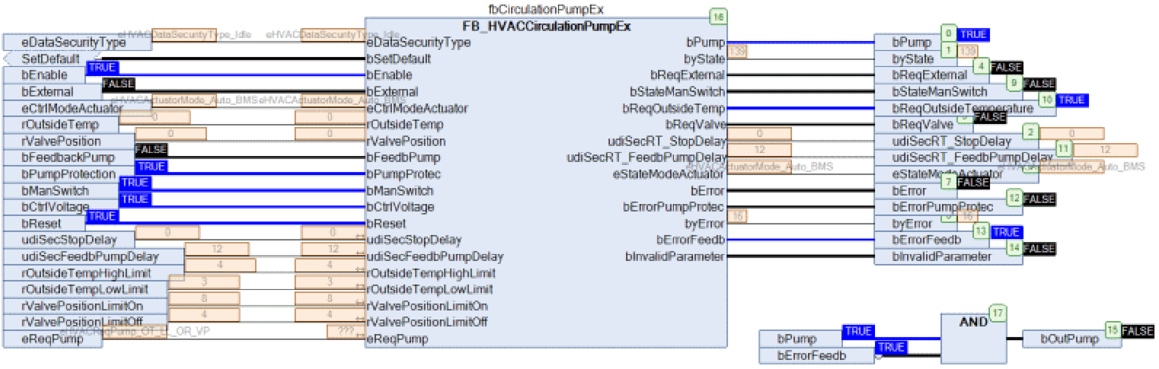
eHVACReqPump_OT_HL_OR_VP: The outside temperature (OT = *rOutsideTemp*) must be higher than *rOutsideTempHighLimit* (HL = Higher Limit) OR the valve position (VP = *rValvePosition*) must be larger than *rValvePositionLimitOn* (see E_HVACReqPump [▸ 492]).

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* (see E_HVACReqPump [▸ 492]).

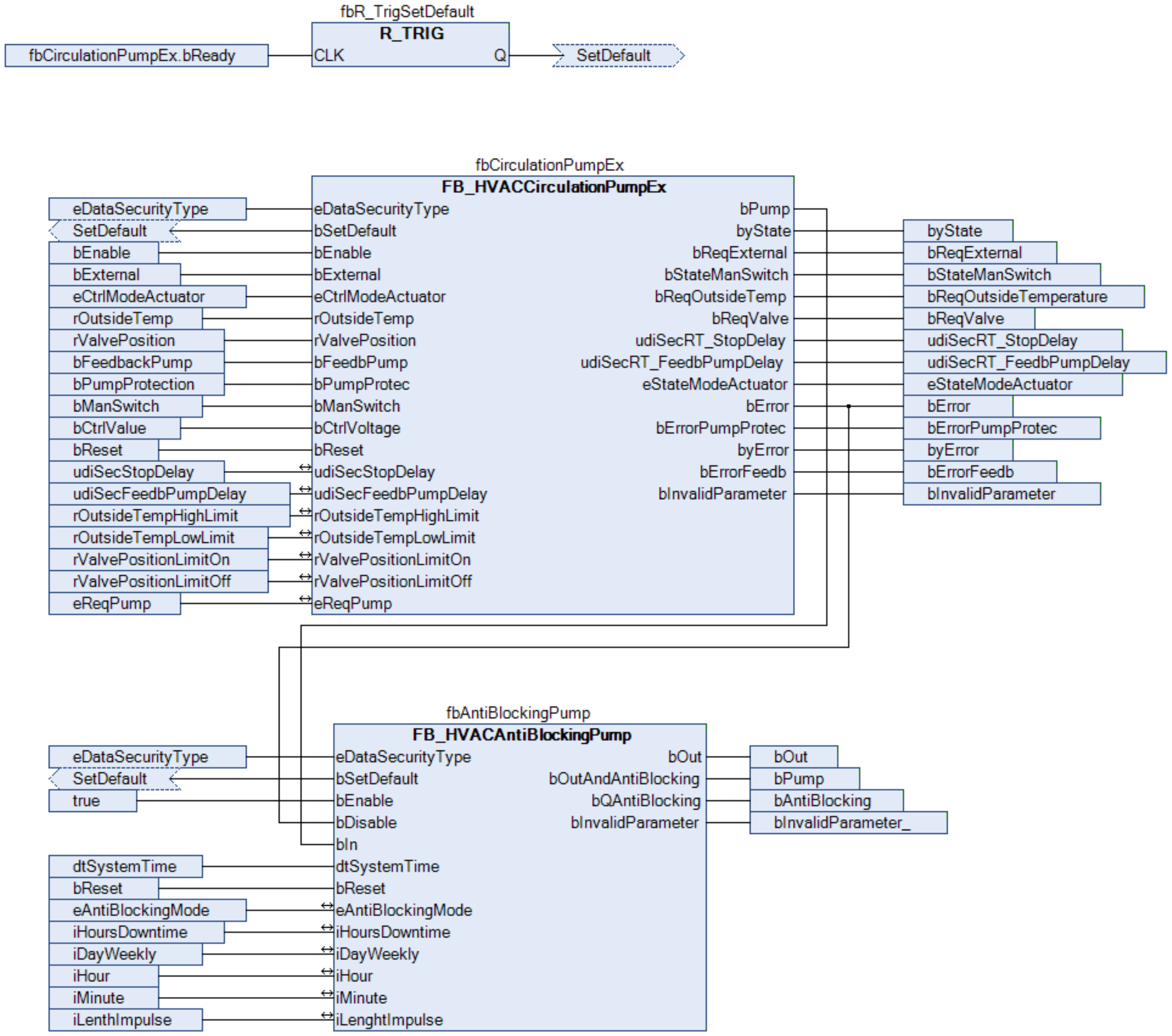
eHVACReqPump_OT_HL_AND_VP: The outside temperature (OT = *rOutsideTemp*) must be higher than *rOutsideTempHighLimit* (HL = Higher Limit) AND the valve position (VP = *rValvePosition*) must be larger than *rValvePositionLimitOn* (see E_HVACReqPump [▸ 492]).

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*.

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 for AntiBlocking function



VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
bExternal         : BOOL;
eCtrlModeActuator : E_HVACActuatorMode;
rOutsideTemp      : REAL;
rValvePosition    : REAL;
bFeedbPump        : BOOL;
bPumpProtec       : BOOL;
bManSwitch        : BOOL;
bCtrlVoltage      : BOOL;
bReset            : BOOL;
```

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 the output *bPump* is constantly 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 *bErrorPumpProtec* = FALSE AND *bManSwitch* = TRUE AND *bCtrlVoltage* = TRUE AND *eCtrlModeActuator* = *eHVACActuatorMode_Auto_BMS* OR *eHVACActuatorMode_Auto_OP* AND *eReqPump*=*eHVACRequestPump_NoRequest*.
Otherwise, *bExternal* controls the output *bPump* depending on *bReqOutsideTemp* and *bReqValve*, see diagrams above.
bExternal is active only in the operation mode *eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP.*

eCtrlModeActuator: Enum that specifies 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* and 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* and *eHVACActuatorMode_Speed1_OP*, provided the following conditions are fulfilled: *bEnable* = TRUE AND *bErrorPumpProtec* = FALSE AND *bManSwitch* = TRUE AND *bCtrlVoltage* = TRUE
The operation modes *eHVACActuatorMode_Off_BMS* and *eHVACActuatorMode_Off_OP* switch the output *bPump* to the FALSE state.
If an incorrect variable value is present on *eCtrlModeActuator*, the last valid variable value is taken. The state of the enum *eCtrlModeActuator* is output via *eStateModeActuator* (see E_HVACActuatorMode [▶ 489]).

rOutsideTemp: Input for transmitting the outside temperature value. Depending on the outside temperature, the pump can be switched on if the temperature falls below the limit value *rOutsideTempLowLimit* or exceeds the limit value rOutsideTempHighLimit. This depends on the enum eReqPump [▶ 492] for requesting the pump, see diagrams above.
rOutsideTemp is active only in the operation mode *eHVACActuatorMode_Auto_BMS* OR *eHVACActuatorMode_Auto_OP*.

rValvePosition: Input for transmitting the valve position of the control loop. Depending on the position *rValvePosition* of the valve associated with the pump, the pump can be switched on when the threshold value *rValvePositionLimitOn* is exceeded. This depends on the enum *eReqPump* for requesting the pump, see diagrams above.
rValvePosition is active only 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 effect on the control of the output *bPump*.
bFeedbPump is active only in the operation mode *eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP* and only if the time *udiSecFeedbPumpDelay* is greater than 0.
If no pump status feedback is present, but the functions are nevertheless realized, then a *TRUE* may not be applied constantly 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 switch for the pump is connected to the input *bPumpProtec*. A pump error has occurred 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. The output *bPump* is disabled if *bManSwitch*= FALSE. 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, then the output *bPump* is constantly FALSE. In order to suppress error message torrents, the error messages *bPumpProtec*, *bErrorFeedb* and *bError* are registered only 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;
bReqOutsideTemp           : BOOL;
bReqValve                 : BOOL;
bStateManSwitch           : BOOL;
udiSecRT_StopDelay        : UDINT;
udiSecRT_FeedbPumpDelay   : UDINT;
eStateModeActuator        : E_HVACActuatorMode;
bError                    : BOOL;
byError                   : BYTE;
bErrorPumpProtec          : BOOL;
bErrorFeedb               : BOOL;
bInvalidParameter         : BOOL;
```

bPump: Output variable for controlling a pump. The following conditions must first be fulfilled in order to control the pump via *bPump*: *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 different possibilities in automatic mode, see diagram above.
The output *bPump* switches itself off, after the switch-on conditions are no longer fulfilled, depending on the Enum E_HVACReqPump [▶ 492], the input *bExternal*, the outside temperature *rOutSideTemp* or the valve position *rValvePosition*, with a delay set by the time *udiSecStopDelay*.

byState: Pump status 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 is fulfilled for switching on the pump via *bPump*, then the output variable *bReqExternal* goes TRUE. All switch-on conditions and the use of the output *bReqExternal* to control *bPump* are described in the diagrams above.

bReqOutsideTemp: If the condition is achieved for requesting the pump via *bPump* depending on the outside temperature *rOutsideTemp*, then the output variable *bReqOutsideTemp* goes TRUE. All switch-on conditions and the use of the output *bReqOutsideTemp* to control *bPump* are described in thediagrams above.

bReqValve: If the condition is achieved for requesting the pump via *bPump* depending on the valve position *rValvePosition*, the output variable *bReqValve* goes TRUE. All switch-on conditions and the use of the output *bReqValve* to control *bPump* are described in the diagrams above.

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

udiSecRT_StopDelay: If the operation mode *eHVACActuatorMode_Auto_BMS* OR *eHVACActuatorMode_Auto_OP* is selected, then the output *bPump* is switched off after the switch-on conditions are no longer satisfied, depending on the enum E_HVACReqPump [▶ 492], the input *bExternal*, the outside temperature *rOutSideTemp* or the valve position *rValvePosition*, delayed by the time set by *udiSecRT_StopDelay*. 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. This error is otherwise indicated via the output variable *bErrorFeedb*.
udiSecRT_FeedbPumpDelay is active only in the operation mode *eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP* and only 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 equivalent to *eCtrlModeActuator* (see E_HVACActuatorMode [▶ 489]).

bError: *bError* goes TRUE if *bErrorPumpProtec* is TRUE. However, only the error *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.

byError: Output of the error as a byte.
byError.1 := bInvalidParameter
byError.2 := bError
byError.3 := bErrorPumpProtec
byError.4 := bErrorFeedb

bErrorPumpProtec: The motor protection switch 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 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 effect on the control of the output *bPump*.
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.

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

VAR_IN_OUT

```
udiSecStopDelay         : UDINT;
udiSecFeedbPumpDelay    : UDINT;
rOutsideTempHighLimit   : REAL;
rOutsideTempLowLimit    : REAL;
rValvePositionLimitOn   : REAL;
rValvePositionLimitOff  : REAL;
eReqPump                : E_HVACReqPump;
```

udiSecStopDelay: The time delay *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, then the output *bPump* is switched off after the switch-on conditions are no longer satisfied, depending on the enum *eReqPump*, the input *bExternal*, the outside temperature *rOutSideTemp* or the valve position *rValvePosition*, delayed by the time set by *udiSecStopDelay*. 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 delay *udiSecFeedbPumpDelay* and must remain so until *bPump* = FALSE. Otherwise this error is indicated via the output variable *bErrorFeedb* (0s..4294967s).
udiSecFeedbPumpDelay is active only in the operation mode *eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP* and only 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*, see diagrams above (-60°C..60°C).
rOutsideTempHighLimit is active only in the operation mode *eHVACActuatorMode_Auto_BMS* OR *eHVACActuatorMode_Auto_OP*.
If an incorrect variable value is present on *rOutsideTempHighLimit*, the last valid variable value is taken. *bInvalidParameter* is set in the case of an incorrect parameter specification. 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*, see diagrams above (-60°C..60°C).
rOutsideTempLowLimit is active only in the operation mode *eHVACActuatorMode_Auto_BMS* OR *eHVACActuatorMode_Auto_OP*.
If an incorrect variable value is present on *rOutsideTempLowLimit*, the last valid variable value is taken. *bInvalidParameter* is set in the case of an incorrect parameter specification. The variable is saved persistently. Preset to 1.

rValvePositionLimitOn: Threshold value for the position *rValvePosition* of a control valve associated with a pump, above which the pump should switch on automatically, e.g. heater pump, see diagrams above (0%..100%).
rValvePositionLimitOn is active only in the operation mode *eHVACActuatorMode_Auto_BMS* OR *eHVACActuatorMode_Auto_OP*.
rValvePositionLimitOn may not be smaller than *rValvePositionLimitOff*. Otherwise the last valid variable value is taken and *bInvalidParameter* is set.
If an incorrect variable value is present on *rValvePositionLimit*, the last valid variable value is taken. *bInvalidParameter* is set in the case of an incorrect parameter specification. The variable is saved persistently. Preset to 5.

rValvePositionLimitOff: Threshold value for the position *rValvePosition* of a control valve associated with a pump, below which the pump should switch off automatically, e.g. heater pump, see diagrams above (0%..100%).
rValvePositionLimitOff is active only in the operation mode *eHVACActuatorMode_Auto_BMS* OR *eHVACActuatorMode_Auto_OP*.
rValvePositionLimitOff may not be larger than *rValvePositionLimitOn.* Otherwise the last valid variable value is taken and *bInvalidParameter* is set.
If an incorrect variable value is present on *rValvePositionLimit*, the last valid variable value is taken. *bInvalidParameter* is set in the case of an incorrect parameter specification. The variable is saved persistently. Preset to 1.

eReqPump: Using the Enum E_HVACReqPump [▸ 492], 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 the limit value *rOutsideTempLowLimit* or exceeds the limit value rOutsideTempHighLimit.

- depending on the position *rValvePosition* of the valve 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 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*

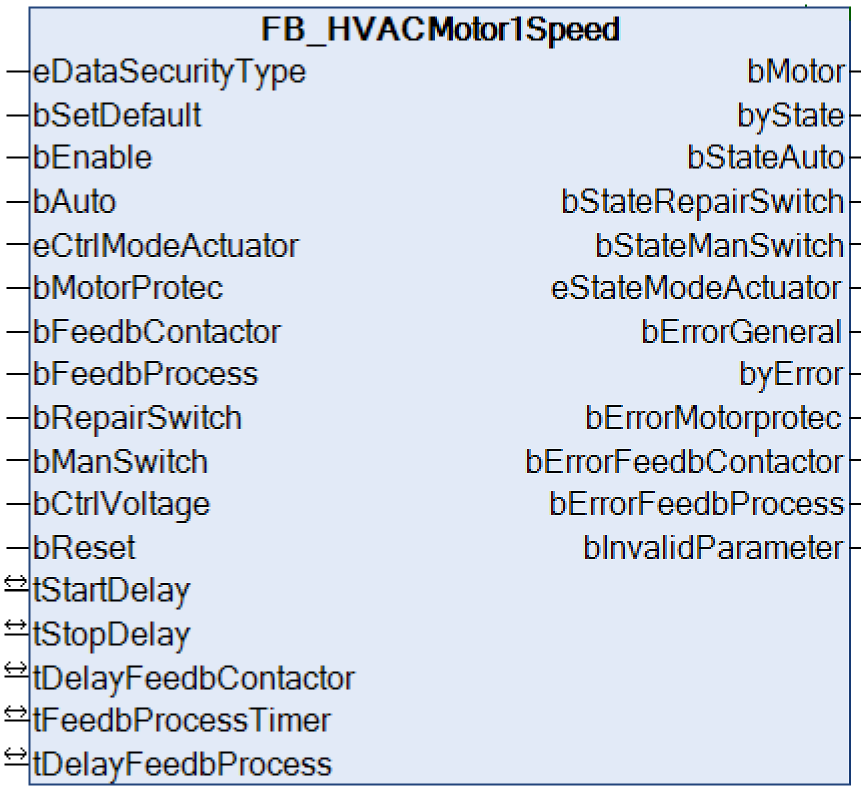
eHVACReqPump_OT_HL_AND_VP: The outside temperature (OT = *rOutsideTemp*) must be larger than *rOutsideTempHighLimit* (HL = Higher Limit) AND the valve position (VP = *rValvePosition*) must be larger than *rValvePositionLimitOn*

Prerequisites for the use of the enum *eReqPump* are that *bEnable* = TRUE AND *bErrorPumpProtec* = FALSE AND *bManSwitch* = TRUE AND *bCtrlVoltage* = TRUE AND *eCtrlModeActuator* = *eHVACActuatorMode_Auto_BMS* OR *eHVACActuatorMode_Auto_OP*.
If an incorrect variable value is present on *eReqPump*, the last valid variable value is taken.
bInvalidParameter is set in the case of an incorrect parameter specification. The variable is saved persistently.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.2.5 FB_HVACMotor1Speed



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

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 [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 then only active if the operation mode *eCtrlModeActuator* is set either to *eHVACActuatorMode_Auto_BMS* or to *eHVACActuatorMode_Auto_OP*.
If the input variable *bAuto* = TRUE, the function block will be instructed that the drive is to run. If *bAuto* = FALSE, the drive will be switched off with a delay via the specified time variable *tStopDelay*.

eCtrlModeActuator: Enum that specifies the operation mode of the motor (see E_HVACActuatorMode [▸ 489]). If specified incorrectly, operation continues internally with the last valid operation mode. This is *eHVACActuatorMode_Auto_BMS* in the case of initial commissioning. *bInvalidParameter* is set in the case of an incorrect parameter specification.

bMotorProtec: Input for the motor protection switch. There is a motor protection switch error 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 for 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*.

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 state of 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 single-speed 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 [▶ 489] is either *eHVACActuatorMode_Auto_BMS* or *eHVACActuatorMode_Auto_OP* and stage 1 was activated via the input variable *bAuto*.

bStateRepairSwitch: Indicates the state of the repair switch. TRUE indicates that the repair switch is switched off.

bStateManSwitch: Indicates the state 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 (see E_HVACActuatorMode [▶ 489]).

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: Delivers all error messages and warnings from the function block.
byError.1 := *bInvalidParameter;*
byError.2 := *bErrorGeneral*;
byError.3 := *bErrorMotorprotec*;
byError.4 := *bErrorFeedbContactor;*
byError.5 := *bErrorFeedbProcess;*

bErrorMotorprotec: Motor protection switch error, 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*.

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

bInvalidParameter: 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 *bInvalidParameter* 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 no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

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 no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

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 no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

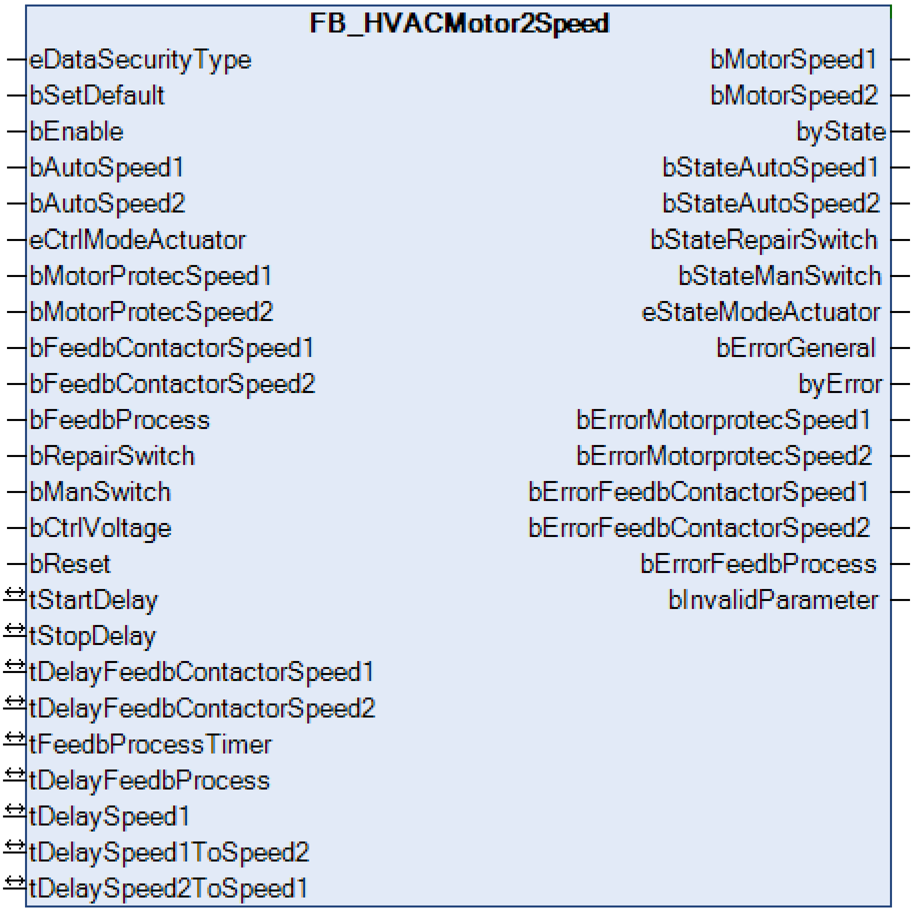
tFeedbProcessTimer: Time delay of the process feedback *bFeedbProcess* [s] after switching on the motor. If this time has elapsed and *bFeedbProcess* = FALSE, then this is fed back to the controller via the error message *bErrorFeedbProcess* (0s..3600s). The variable is saved persistently. Preset to 0 s.
If no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

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 response of the input *bFeedbProcess* can be delayed by the time *tDelayFeedbProcess* [s] (0s..3600s). The variable is saved persistently. Preset to 0 s.
If no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.2.6 FB_HVACMotor2Speed



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*.

VAR_INPUT

```
eDataSecurityType     : E_HVACDataSecurityType;
bSetDefault           : BOOL;
bEnable               : BOOL;
bAutoSpeed1           : BOOL;
bAutoSpeed2           : BOOL;
eCtrlModeActuator     : E_HVACActuatorMode;
bMotorProtecSpeed1    : BOOL;
bMotorProtecSpeed2    : BOOL;
bFeedbContactorSpeed1 : BOOL;
bFeedbContactorSpeed2 : BOOL;
bFeedbProcess         : BOOL;
bRepairSwitch         : BOOL;
bManSwitch            : BOOL;
bCtrlVoltage          : BOOL;
bReset                : BOOL;
```

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 via the specified time variable *tStopDelay*. If an enable is present, a restart of the motor is blocked for the duration of *tDelaySpeed2toSpeed1*.

bAutoSpeed1: *bAutoSpeed1* is then only active if the operation mode *eCtrlModeActuator* is either *eHVACActuatorMode_Auto_BMS* or *eHVACActuatorMode_Auto_OP*.
If the input variable *bAutoSpeed1* = TRUE and the input variable *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 via the specified time variable *tStopDelay*.

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

eCtrlModeActuator: Enum that specifies the operation mode of the motor (see E_HVACActuatorMode [▶ 489]). If specified incorrectly, operation continues internally with the last valid operation mode. This is *eHVACActuatorMode_Auto_BMS* in the case of initial commissioning. *bInvalidParameter* is set in the case of an incorrect parameter specification.

bMotorProtecSpeed1: Input for the motor protection speed 1. There is a motor protection error if the input *bMotorProtecSpeed1* = FALSE (quiescent current principle). In the event of a fault, the outputs *bMotorSpeed1* and *bMotorSpeed2* are both FALSE; the error 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 the motor protection speed 2. There is a motor protection error if the input *bMotorProtecSpeed2* = FALSE (quiescent current principle). In the event of a fault, the outputs *bMotorSpeed1* and *bMotorSpeed2* are both 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 both 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*.

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

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 both 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.

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*.
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 and the outputs *bMotorSpeed1* and *bMotorSpeed2* are both 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 and 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 state of 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 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

```
bMotorSpeed1               : BOOL;
bMotorSpeed2               : BOOL;
byState                    : BYTE;
bStateAutoSpeed1           : BOOL;
bStateAutoSpeed2           : BOOL;
bStateRepairSwitch         : BOOL;
bStateManSwitch            : BOOL;
eStateModeActuator         : E_HVACActuatorMode;
bErrorGeneral              : BOOL;
byError                    : BYTE;
bErrorMotorprotecSpeed1    : BOOL;
bErrorMotorprotecSpeed2    : BOOL;
bErrorFeedbContactorSpeed1 : 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;*
yState.5 := *bStateRepairSwitch;*
byState.6 := *bStateManSwitch;*
byState.7 := *bCtrlVoltage;*

bStateAutoSpeed1: Indicates the state for automatic preselection of 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 of 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: Indicates the state of the repair switch. TRUE indicates that the repair switch is switched off.

bStateManSwitch: Indicates the state 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 (see E_HVACActuatorMode [⏵ 489]).

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: Delivers all error messages and warnings from 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: Motor protection error, see input variable *bMotorProtecSpeed1*

bErrorMotorprotecSpeed2: Motor protection error, 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.

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.

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*, *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 *bInvalidParameter* 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 no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

tStopDelay: The switch-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 no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

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 no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

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 no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

tFeedbackProcessTimer: Time delay [s] of the process feedback *bFeedbProcess* during the motor start-up phase. 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 no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

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 response of the input *bFeedbProcess* can be delayed by the time *tDelayFeedbProcess* [s] (0s..3600s). The variable is saved persistently. Preset to 0 s.
If no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

tDelaySpeed1: Time delay [s] for the start-up phase of the motor in speed 1 (1s..3600s). The variable is saved persistently. Preset to 3 s.
After this time has elapsed the motor can be switched from the first to the second speed if the operation mode for speed 2 has been selected.
If no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

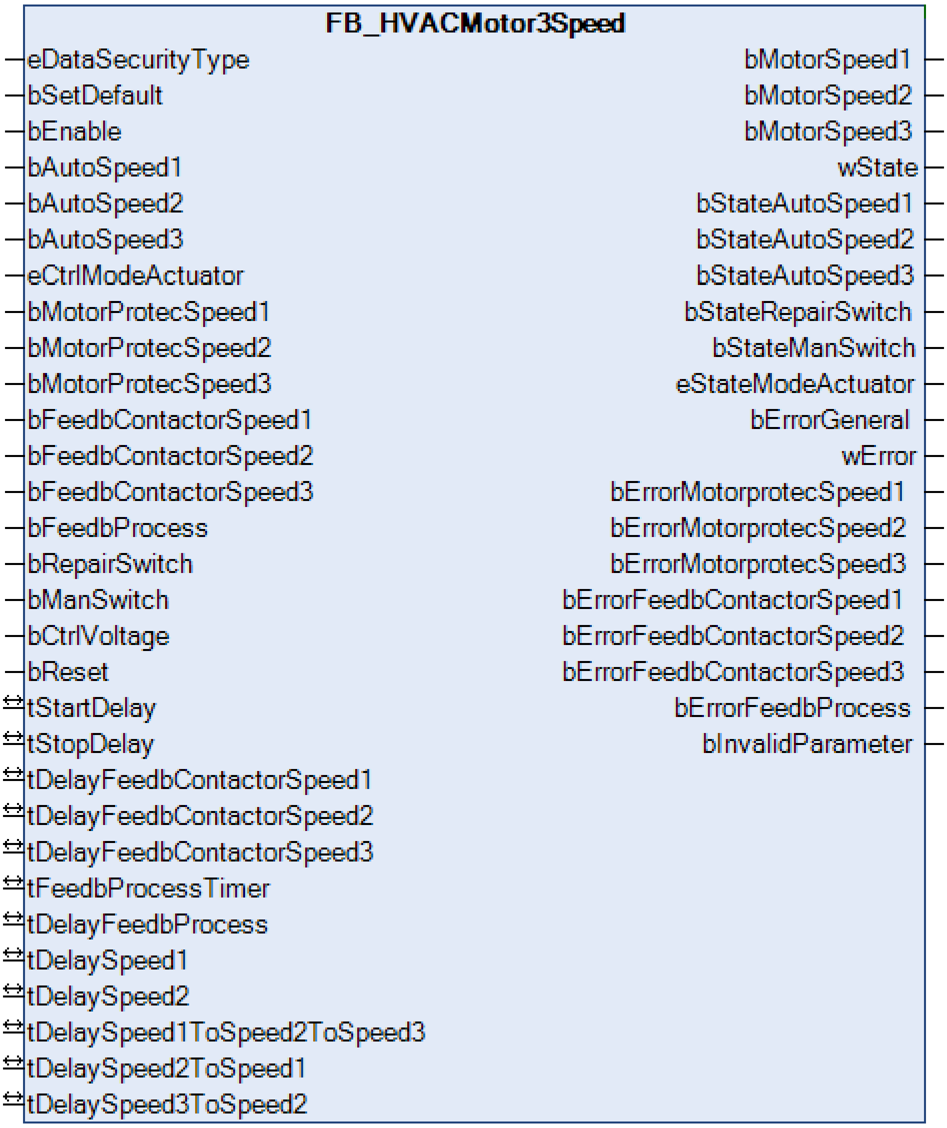
tDelaySpeed1ToSpeed2: Time delay for the motor switchover phase from speed 1 to speed 2, so that both outputs *bMotorSpeed1* and *bMotorSpeed2* are FALSE for a short while (100ms..10s). The variable is saved persistently. Preset to 250 ms. The purpose of this time delay is to preserve the motor windings.
If no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

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 no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.2.7 FB_HVACMotor3Speed



This function block serves to control a three-speed drive in HVAC systems. The function block always runs in stage 1, the lowest power stage, and switches to stage 2 or stage 3 depending on requirements. It cannot be switched on directly in stage 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*.

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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 via the specified time variable *tStopDelay*.
If an enable is present, a restart of the motor is blocked for the duration of *tDelaySpeed2ToSpeed1* + *tDelaySpeed3ToSpeed2*.

bAutoSpeed1: *bAutoSpeed1* is then only active if the operation mode eCtrlModeActuator [▶ 489] is either *eHVACActuatorMode_Auto_BMS* or *eHVACActuatorMode_Auto_OP*.
If the input variable *bAutoSpeed1* is TRUE and the input variables *bAutoSpeed2* and *bAutoSpeed3* are 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 via the specified time variable *tStopDelay*.

bAutoSpeed2: *bAutoSpeed2* is then only active if the operation mode eCtrlModeActuator [▶ 489] is either *eHVACActuatorMode_Auto_BMS* or *eHVACActuatorMode_Auto_OP*.
If the input variables *bAutoSpeed1* and *bAutoSpeed2* are TRUE and the input variable *bAutoSpeed3* is FALSE, then the function block is instructed that the drive should run at speed 2.
If *bAutoSpeed1* and *bAutoSpeed2* are both FALSE, the drive will be switched off with a delay via the specified time variable *tStopDelay*.

bAutoSpeed3: *bAutoSpeed3* is then only active if the operation mode eCtrlModeActuator [▶ 489] is either *eHVACActuatorMode_Auto_BMS* or *eHVACActuatorMode_Auto_OP*.
If the input variables *bAutoSpeed1, bAutoSpeed2* and *bAutoSpeed3* are TRUE, then the function block is instructed that the drive should run at speed 3.
If *bAutoSpeed1, bAutoSpeed2* and *bAutoSpeed3* are all FALSE, the drive will be switched off with a delay via the specified time variable *tStopDelay*.

eCtrlModeActuator: Enum that specifies the operation mode of the motor (see E_HVACActuatorMode [▶ 489]). If specified incorrectly, operation continues internally with the last valid operation mode. This is *eHVACActuatorMode_Auto_BMS* in the case of initial commissioning. *bInvalidParameter* is set in the case of an incorrect parameter specification.

bMotorProtecSpeed1: Input for the motor protection speed 1. There is a motor protection error if the input *bMotorProtecSpeed1* = 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 the motor protection speed 2. There is a motor protection error if the input *bMotorProtecSpeed2* = 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 the motor protection speed 3. There is a motor protection error if the input *bMotorProtecSpeed3* = 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*.

ℹ 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*.

ℹ 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*.

ℹ 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, 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 and 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, then *bManSwitch* is FALSE and 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 state of 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 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: Status word 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 of speed 1 if the operation mode eCtrlModeActuator [▶ 489] 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 of speed 2 if the operation mode eCtrlModeActuator [▶ 489] 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 of speed 3 if the operation mode eCtrlModeActuator [▶ 489] is either *eHVACActuatorMode_Auto_BMS* or *eHVACActuatorMode_Auto_OP* and speed 3 was activated via the input variable *bAutoSpeed3*.

bStateRepairSwitch: Indicates the state of the repair switch. TRUE indicates that the repair switch is switched off.

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

eStateModeActuator: Enum via which the operation mode status 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, bErrorFeedbContactorSpeed23* 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: Delivers all error messages and warnings from 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: Motor protection error, see input variable *bMotorProtecSpeed1.*

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

bErrorMotorprotecSpeed3: Motor protection error, 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.

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.

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.

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 no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

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 no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

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 no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

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 no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

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 no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

tFeedbackProcessTimer: Time delay [s] of the process feedback *bFeedbProcess* during the motor start-up phase. 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 no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

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 response of the input *bFeedbProcess* can be delayed by the time *tDelayFeedbProcess* [s] (0s..3600s). The variable is saved persistently. Preset to 0 s.
If no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

tDelaySpeed1: Time delay [s] for the start-up phase of the motor in speed 1 (1s..3600s). The variable is saved persistently. Preset to 3 s.
After this time has elapsed the motor can be switched from the first to the second speed if the operation mode for speed 2 has been selected.
If no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

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

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 all FALSE for a short while (1s..3600s). The variable is saved persistently. Preset to 250 ms. The purpose of this time delay is to preserve the motor windings.
If no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

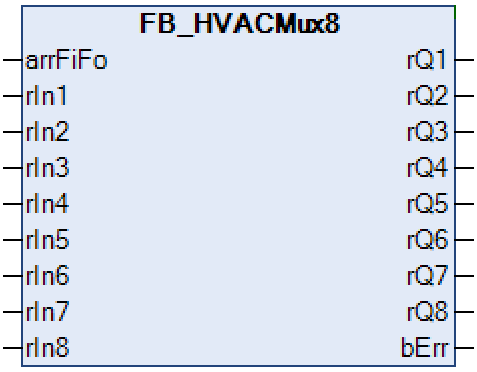
tDelaySpeed2ToSpeed1: Time delay [s] for the motor switchover phase from speed 2 to speed 1 (1s..3600s). The variable is saved persistently. Preset to 10 s. In this phase the outputs *bMotorSpeed1, bMotorSpeed2* and *bMotorSpeed3* are FALSE for the time *tDelaySpeed2ToSpeed1* in order to reduce the speed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

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 *tDelaySpeed2ToSpeed2* in order to reduce the speed of the motor when switching to speed 2.
If no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.2.8 FB_HVACMux8



This function block evaluates the FIFO memory from FB_HVACRedundancyCtrlEx [▶ 65]. 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 example 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;
rIn1 - rIn8         : REAL;
```

arrFiFo: Contains the assignment table with a maximum of eight values. The first value indicates where the first input was copied to, the second value where the second input was copied to and so on. Identical values are permitted. No assignment takes place with "0".

rIn1 - 8: Setpoints to be mapped.

VAR_OUTPUT

```
rQ1 - rQ8       : REAL;
bErr            : BOOL;
```

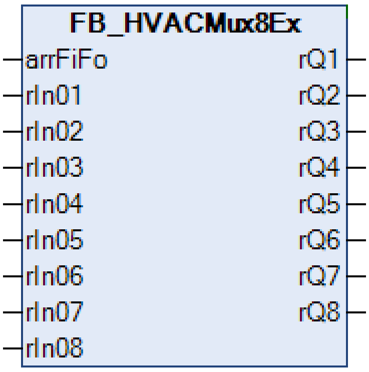
rQ1 -rQ8: Actuator setpoint, input value mapped according to the FiFo table.

bErr: Value range in the FiFo table was exceeded; 0 to 8 is permissible.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.2.9 FB_HVACMux8Ex



This function block evaluates the FIFO memory from FB_HVACRedundancyCtrlEx [▶ 65]. The inputs are mapped to the respective outputs according to a FIFO table.

Sample:

In the sample the array contains: 4,3,1,2,0,0,0,0. As a result the following are now output in *FB_HVACMux8Ex*

rIn1 on the output *rQ4*

rIn2 on the output *rQ3*

rIn3 on the output *rQ1*

rIn4 on the output *rQ2*

VAR_INPUT

```
arrFiFo         : Array[1..8] of INT;
rIn1 - rIn8     : REAL;
```

arrFiFo: Contains the assignment table with a maximum of eight values. The first value indicates where the first input was copied to, the second value where the second input was copied to and so on. No assignment takes place with "0".

rIn1- rIn8: Setpoints to be mapped.

VAR_OUTPUT

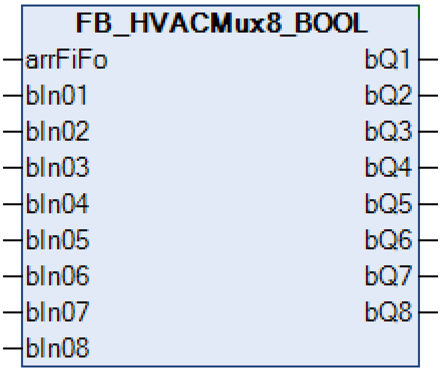
```
rQ1 - rQ8      : REAL;
```

rQ1 – rQ8: Actuator setpoint, input value mapped according to the FIFO table.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.22 or higher	Tc2_HVAC V3.3.4.0	TF8000 | TC3 HVAC V1.0.1.0

5.1.2.10 FB_HVACMux8_BOOL



This function block evaluates the FIFO memory from FB_HVACRedundancyCtrlEx [▸ 65]. The inputs are mapped to the respective outputs according to a FIFO table.

The functions blocks FB_HVACRedundancyCtrlEx [▸ 65], *FB_HVACMux8_BOOL* and FB_HVACMux8Ex [▸ 60] enable the sequential control of up to 8 units with automatic changing of the switch-on sequence.

Sample:

In the sample the array contains: 4,3,1,2,0,0,0,0. As a result the following are now output in *FB_HVACMux8_BOOL*

bIn1 on the output *bQ4*

bIn2 on the output *bQ3*

bIn3 on the output *bQ1*

bIn4 on the output *bQ2*

ausgegeben.

VAR_INPUT

```
arrFiFo          : Array[1..8] of INT;
bIn1 - bIn8      : BOOL;
```

arrFiFo: Contains the assignment table with a maximum of eight values. The first value indicates where the first input was copied to, the second value where the second input was copied to and so on. Identical values are permitted. No assignment takes place with "0".

bIn1- bIn8: Variables to be mapped.

VAR_OUTPUT

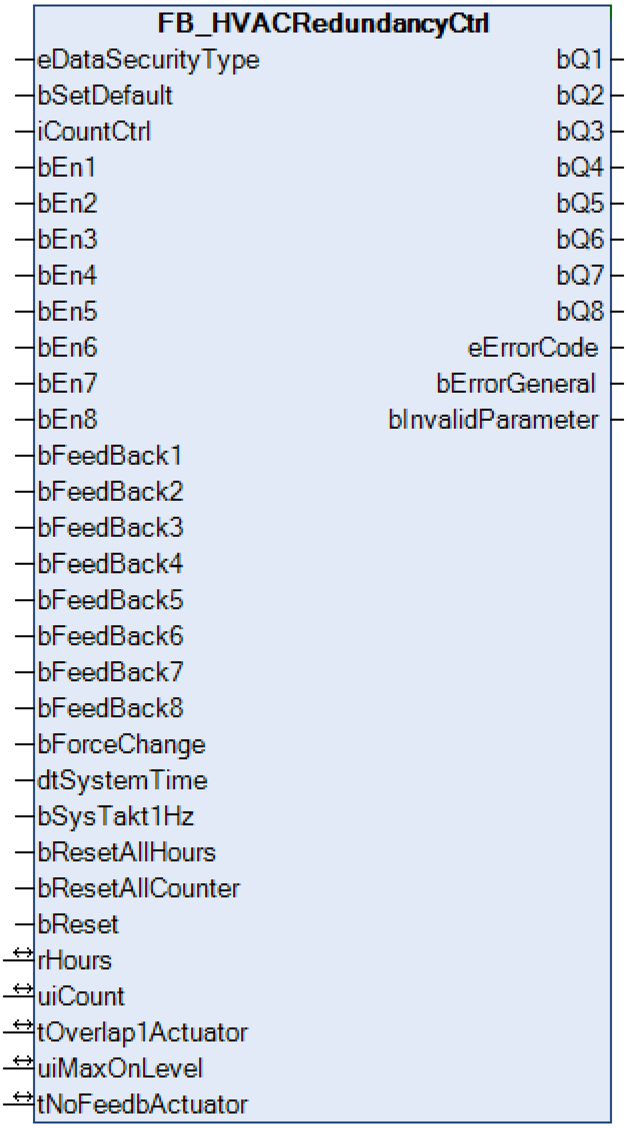
```
bQ1 – bQ8        : BOOL;
```

bQ1 – bQ8: Output variable; mapped input variable according to FIFO table.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.22 or higher	Tc2_HVAC V3.3.4.0	TF8000 | TC3 HVAC V1.0.1.0

5.1.2.11 FB_HVACRedundancyCtrl



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 *bForceChange*. 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 shock in the pumps, an overlap time can be set via *tOverlap1Actuator*. This applies only to cases where an exchange between two actuators takes place.

In order to determine the runtimes, the function block FB_HVACWork [▸ 459] is instanced internally and the IN-OUT variables are relayed to the outside (*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

```
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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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* > *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.

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 hour meters.

bResetAllCounter: Resets all switch-on meters.

bReset: Acknowledge input in the event of a fault.

VAR_OUTPUT

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

bQ1 - 8: Actuator feedback.

eErrorCode: Indicates which actuator has not returned an operating feedback within the prespecified time interval (see E_HVACErrorCodes [▶ 491]). The recognition of this group of errors 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 has occurred during the plausibility check. The message must be acknowledged with *bReset*.

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.

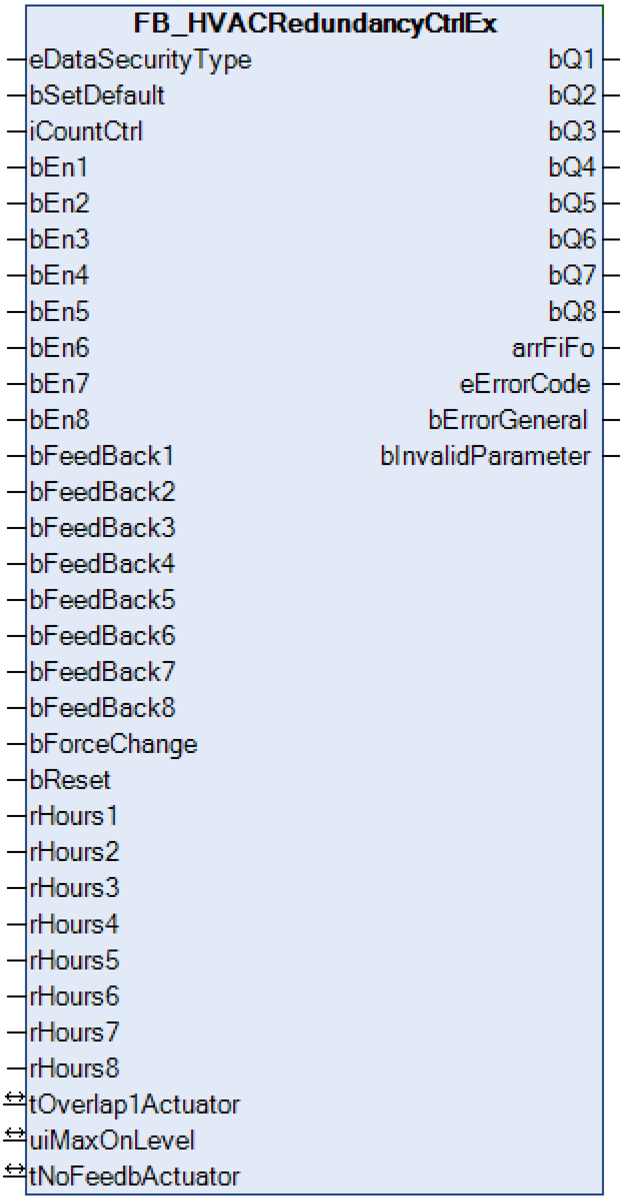
uiMaxOnLevel: Max. time in hours [h] that an actuator may be switched on (0h..1000h). Forces an actuator change only if an actuator is available to be switched on. The variable is saved persistently. Preset to 200 h.

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.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.2.12 FB_HVACRedundancyCtrlEx



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 take place with the time specification *uiMaxOnLevel*, a change of *iCountCtrl* or by the trigger *bForceChange*. 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 applies only to cases where an exchange between two actuators takes place.
In contrast to FB_HVACRedundancyCtrl [▶ 62] no internal timer is used for determining the runtimes. Instead, the times have to be specified externally as hourly values via 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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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). A change in the sequence of the actuators to be switched on is created by a change of *iCountCtrl*.

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_OUTPUT

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

bQ1 - 8: Actuator feedback.

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

eErrorCode: Indicates which actuator has not returned an operating feedback within the prespecified time interval (see E_HVACErrorCodes [▶ 491]). The recognition of this group of errors 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 has occurred during the plausibility check. The message must be acknowledged with *bReset*.

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.

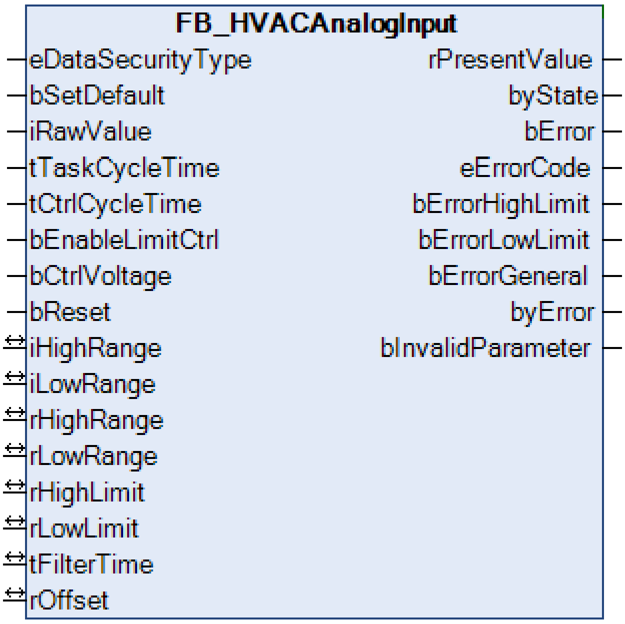
tNoFeedbActuator: Time that is allowed to elapse before the lack of an operating feedback from the actuator leads to *bErrorGeneral* = TRUE (0ms..60s). There is no evaluation if the time = 0. The variable is saved persistently. Preset to 3 s

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.3 HVAC Analog modules

5.1.3.1 FB_HVACAnalogInput



This function block serves the acquisition and scaling of analog input signals. Using the KL30xx, KL31xx and KL32xx terminals, the standard signals 0-20 mA, 4-20 mA, 0-10 V and 10-5000 Ω 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 [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 TaskCycleTime. The function block uses this input variable to calculate internally whether the state and the output variables have to be updated in the current cycle.

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 [▸ 447]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 supply voltage is present.

bError: The variable *bError* changes to TRUE in the event of an internal error in the function block.

eErrorCode*:* Contains the specific error code related to the output *bError* (see E_HVACErrorCodes [▸ 491]).

bErrorHighLimit: TRUE if the upper limit value has been 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 error as a 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.

bInvalidParameter: TRUE if an error has occurred 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 *HighRange/rHighRange* and *iLowRange/rLowRange*, the raw value is linearly converted to the physical unit. *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.

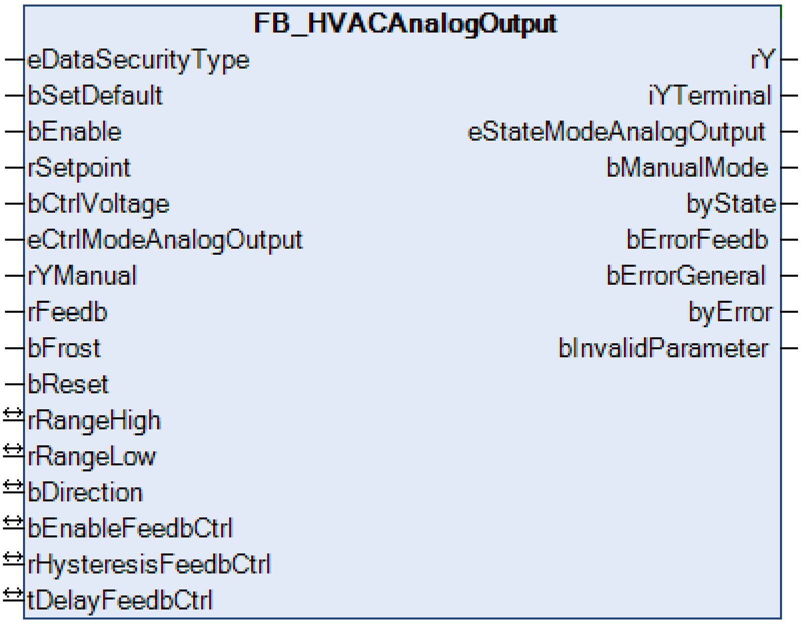
The output value is given by:

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

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.3.2 FB_HVACAnalogOutput



This function block serves to control analog 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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 (see E_HVACAnalogOutputMode [▶ 489]).
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 positional 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 once again FALSE.

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: Shows 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 (see E_HVACAnalogOutputMode [▶ 489]).

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: Delivers all error messages and warnings.
byError.1 := *bInvalidParameter*
byError.2 := *bErrorGeneral*
byError.3 := *bErrorFeedb*

bInvalidParameter: TRUE if an error has occurred 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,
==> rY = 50
==> rYTerminal = 16384

rSetpoint = 200, rRangeLow = 0, rRangeHigh = 200,
==> rY = 100
==> 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: Analog 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.

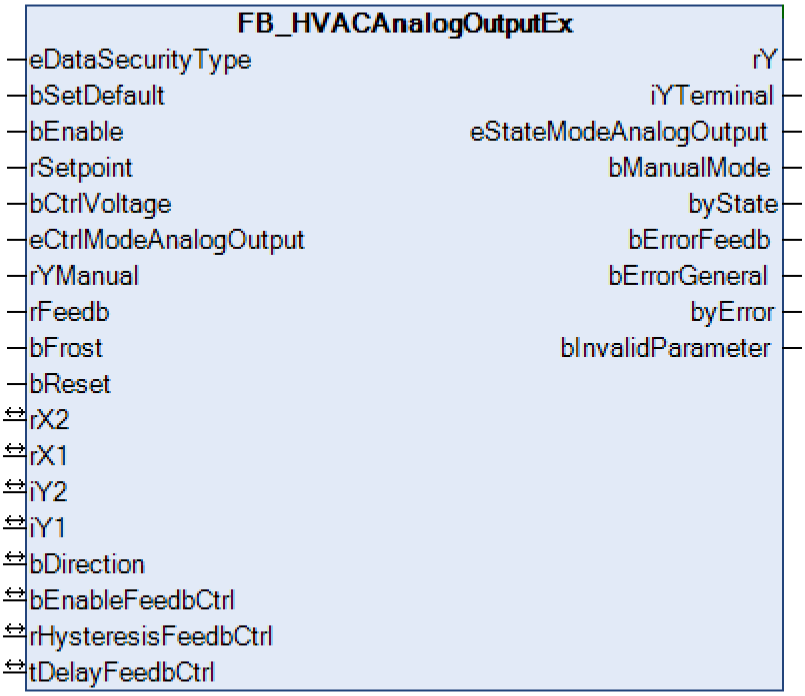
rHysteresisFeedbCtrl: Due to the stroke time of typical drives used in heating and air conditioning systems, the positional feedback always lags in the case of a jump in the set position value. Using the variable *rHysteresisFeedbCtrl*, a range is specified within which the position setpoint *rY* of 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* [s] (0s..50s). The variable is saved persistently.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.3.3 FB_HVACAnalogOutputEx/FB_HVACAnalogOutputEx2



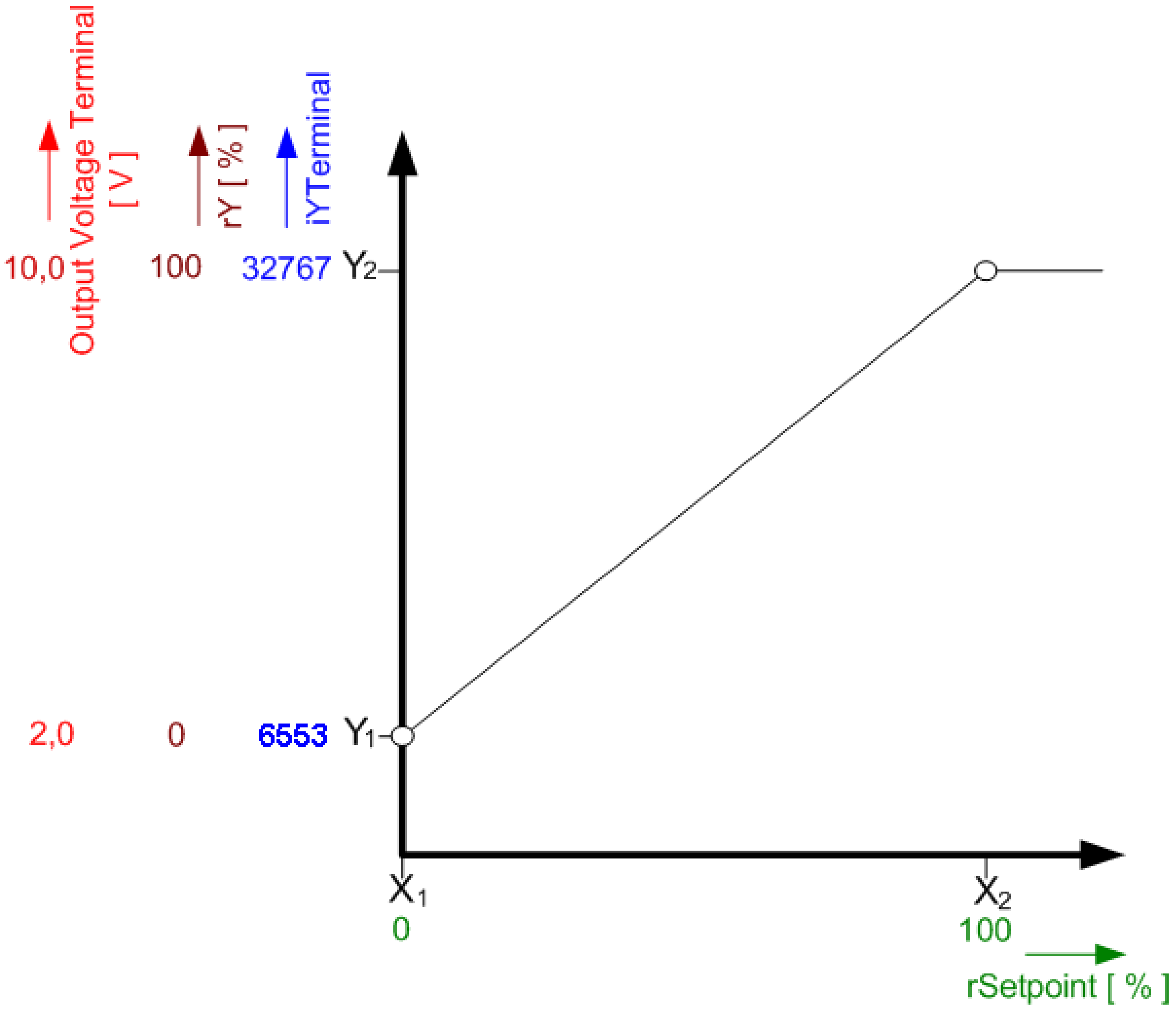
This function block serves to control analog actuators, such as valves or dampers, with positional feedback. Unlike the FB_HVACAnalogOutput [▶ 70], a scaling function is integrated in this function block. The function is: y=m*x+b.

The only difference between the function blocks *FB_HVACAnalogOutputEx* and *FB_HVACAnalogOutputEx2* is that with *FB_HVACAnalogOutputEx2* the control direction is queried after the response of *bFrost*. If the control direction is inverse, *rY* is set to the smallest value.

Application example:

Damper actuator working range 2 to 10 V with the KL4404 (signal voltage 0 to 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 [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 (see E_HVACAnalogOutputMode [▸ 489]).
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 positional 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 once again FALSE.

With *FB_HVACAnalogOutputEx2* the control direction is queried after the response of *bFrost*. If the control direction is inverse, *rY* is set to the smallest value.

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: Shows 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 (see E_HVACAnalogOutputMode [▸ 489]).

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: Delivers all error messages and warnings.
byError.1 := bInvalidParameter
byError.2 := bErrorGeneral
byError.3 := bErrorFeedb

bInvalidParameter: TRUE if an error has occurred 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: Analog 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.

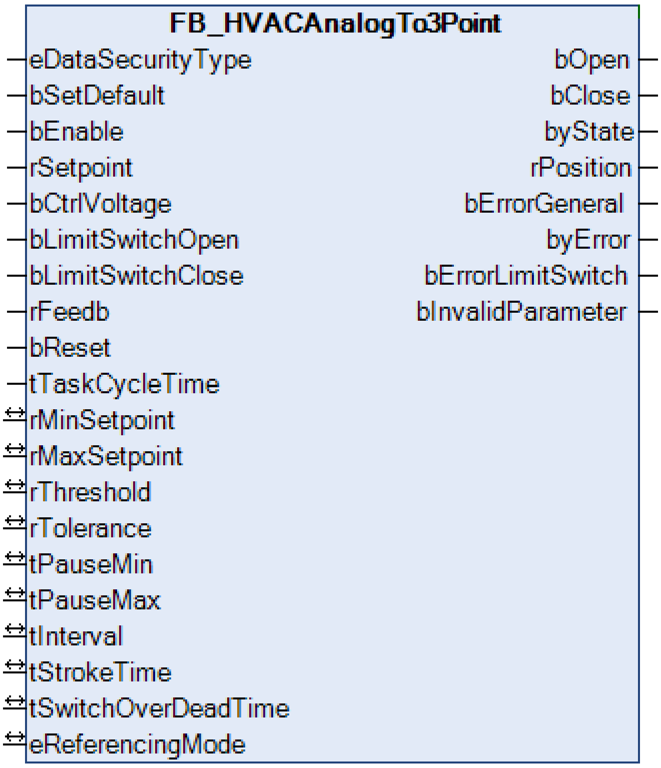
rHysteresisFeedbCtrl: Due to the stroke time of typical drives used in heating and air conditioning systems, the positional feedback always lags in the case of a jump in the set position value. Using the variable *rHysteresisFeedbCtrl*, a range is specified within which the position setpoint *rY* of 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.

Requirements

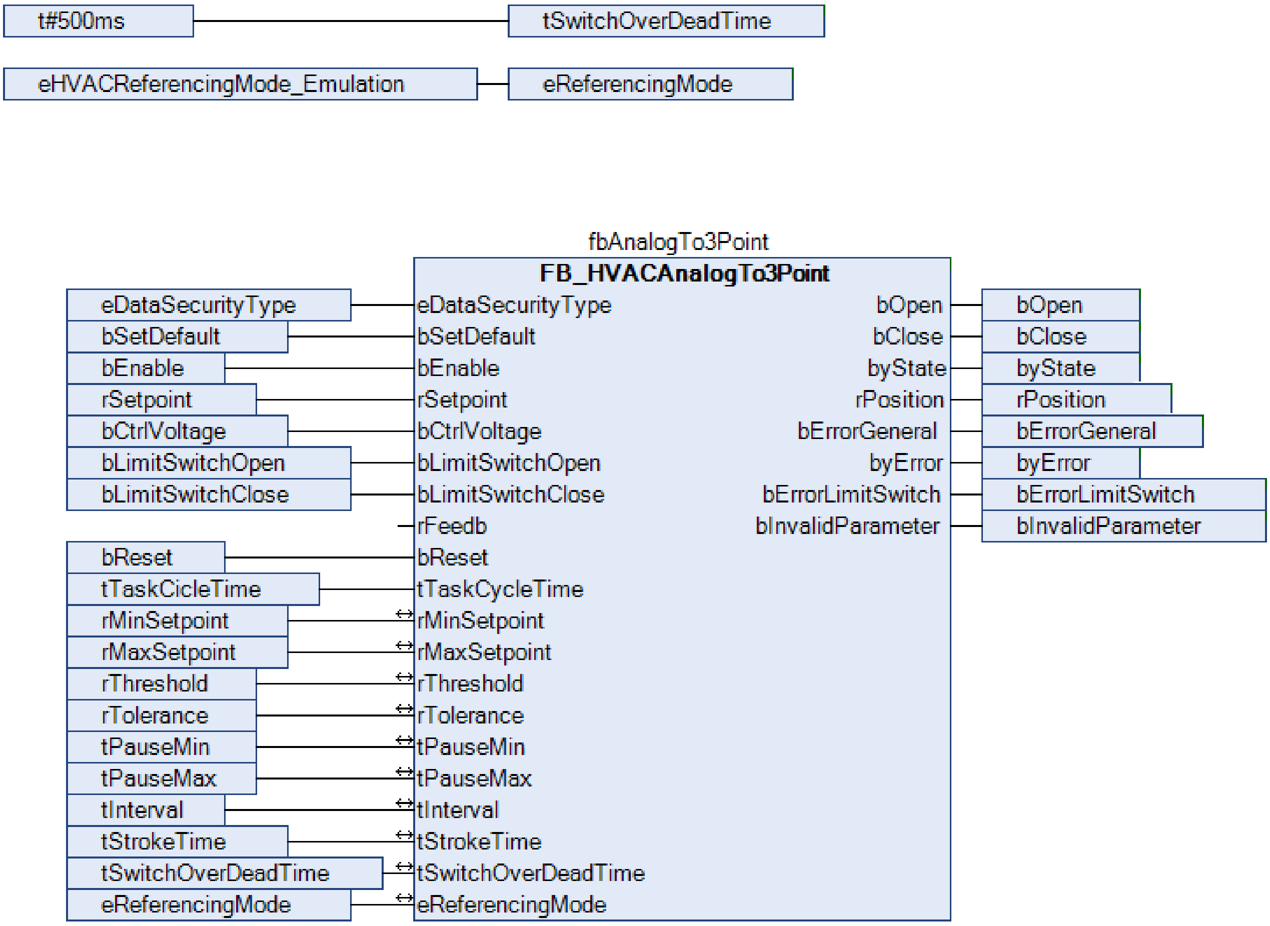
Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.3.4 FB_HVACAnalogTo3Point



This function block serves to convert an analog control signal into a three-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: 3-Point-Actuator without feedback and limit switches



VAR_INPUT

```
eDataSecurityType     : E_HVACDataSecurityType;
bSetDefault           : BOOL;
bEnable               : BOOL;
rSetpoint             : REAL;
rFeedb                : REAL;
bLimitSwitchOpen      : BOOL;
bLimitSwitchClose     : BOOL;
bCtrlVoltage          : BOOL;
bReset                : BOOL;
tTaskCycleTime        : TIME;
```

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 value range for *rSetpoint* is specified by the parameters *rMaxSetpoint* and *rMinSetpoint* (0.0..32767).
If no correct variable value is present at *rSetpoint*, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues internally with *rMinSetpoint*. *bInvalidParameter* will be set in the event of an incorrect variable value; the function block continues to work normally.

rFeedb: The input variable *rFeedb* (0.0..32767) 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 value range for *rFeedb* is specified by the parameters *rMaxSetpoint* and *rMinSetpoint. rFeedb* is feedback to the controller via the output variable *rPosition.*
If no correct variable value is present at *rFeedb* , then the last valid variable value, if one exists, will be taken. If no valid last value exists, operation continues internally with *rMinSetpoint*. *bInvalidParameter* will be set in the event of an incorrect variable value; the function block continues to work normally.

bLimitSwitchOpen/bLimitSwitchClose: If the three-point actuator is equipped with limit switches, these can be connected to the inputs *bLimitSwitchOpen* and *bLimitSwitchClose*.
The limit switches serve as references for the function block in the referencing mode *eReferencingMode* = *eHVACReferencingMode_Emulation*. 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 *bLimitSwitchOpen/bLimitSwitchClose*. 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 at TRUE after the respective limit switch has been reached.
The limit switches must be connected to the function block as normally closed contacts. If none exist, a TRUE must be applied to the two inputs *bLimitSwitchOpen/bLimitSwitchClose*.

bCtrlVoltage: A check is made of whether control voltage is present via the input variable *bCtrlVoltage*. Both *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 *FB_HVACAnalogTo3Point* in the event of control voltage failure.
The control voltage is present if a TRUE is present at the input *bCtrlVoltage*.

bReset: Acknowledge input in the event of a fault.

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.
If an incorrect time value of T#0 s is present on *tTaskCycleTime*, then *tTaskCycleTime* is set internally to T#1 ms. *bInvalidParameter* is set in the case of an incorrect variable value.

VAR_OUTPUT

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

bOpen: Output for opening drive of the three-point actuator.

bClose: Output for closing drive of 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 [▶ 492] = *eHVACReferencingMode_AnalogFeedback*, then *rFeedb* is fed back to the controller via the output variable *rPosition*. If *rFeedb* > *rMaxSetpoint*, then *rPosition* = *rMaxSetpoint* and *bInvalidParameter* is set. If *rFeedb* < *rMinSetpoint*, then *rPosition* = *rMinSetpoint* and *bInvalidParameter* 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: Delivers all error messages and warnings from the function block
byError.1 := *bInvalidParameter*
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 if ((rMinSetpoint + rThreshold) > rSetpoint) and bClose = TRUE). The error *bErrorLimitSwitch* can only occur if *eReferencingMode* = *eEmulation*.

bInvalidParameter: 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 *bInvalidParameter* must be acknowledged via *bReset*.

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 value range for *rSetpoint* is specified by the parameters *rMaxSetpoint* (0..32767) and *rMinSetpoint* (0..32767). *rMaxSetpoint* must be greater than *rMinSetpoint.* In addition the two variables *rMaxSetpoint* and *rMinSetpoint* are integrated in the calculation of the variable pulse/pause modulation; see fig. 1.1.
If an incorrect variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the preset value. *bInvalidParameter* will be set in the event of an incorrect parameter entry; the function block continues to work normally. The variables are saved persistently. *rMinSetpoint* preset to 0. *rMaxSetpoint* preset to 100.

rThreshold/rTolerance: If the difference between the set position value *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* and *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*: *bOpen* is activated

rPosition- rSetpoint > = *rThreshold: bClose* is activated
If an incorrect variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the preset value. *bInvalidParameter* will be set in the event of an incorrect parameter entry; the function block continues to work normally. The variables are saved persistently. *rThreshold* preset to 5.0. *rTolerance* preset to 2.0.

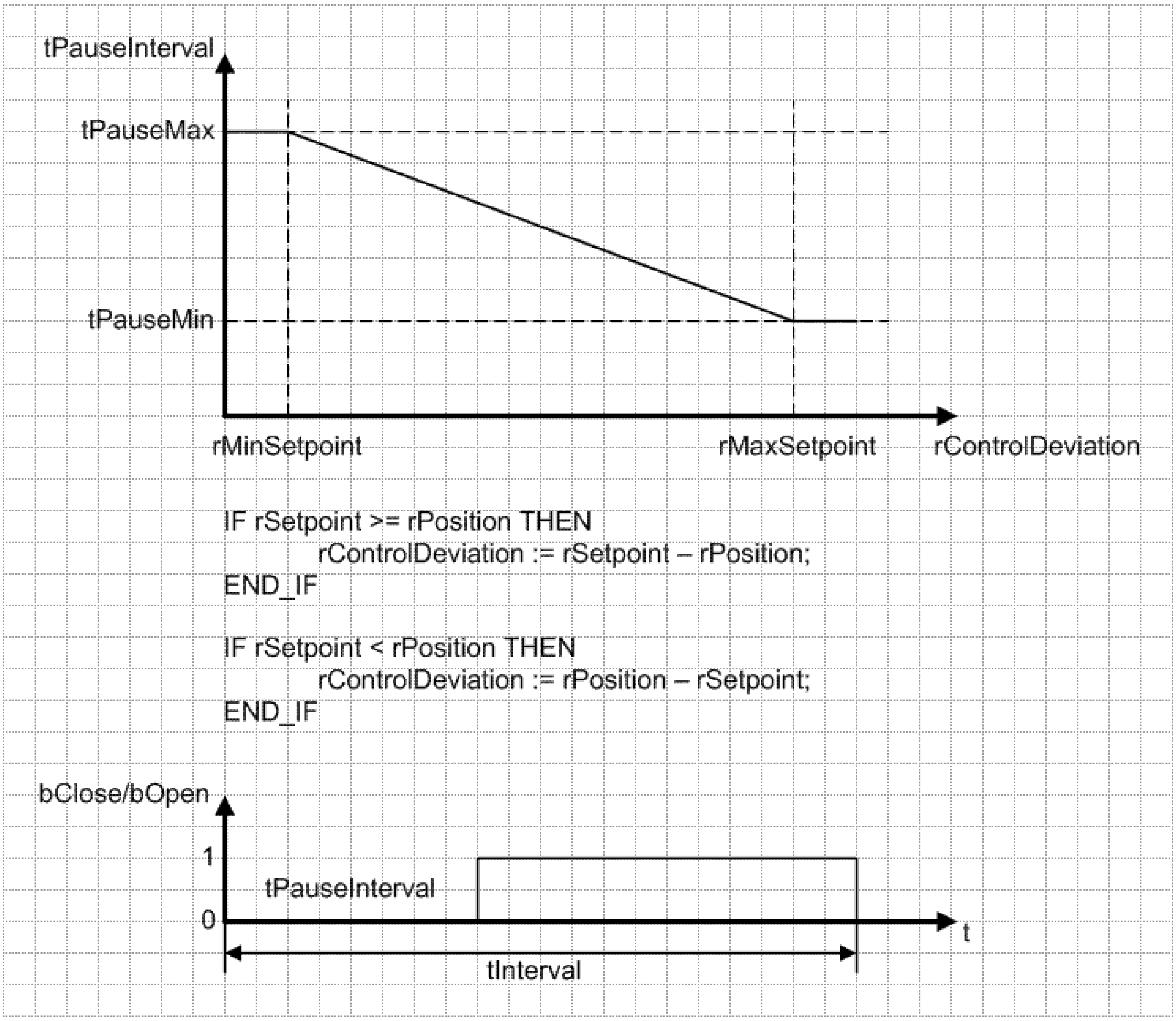
tPauseMin/tPauseMax/tInterval: A variable pulse/pause modulation for cycling the three-point actuator can be set with the parameters *tPauseMin* [s] (0..3600), *tPauseMax* [s] (0..3600) and *tInterval* [s] (0..3600). The pause length is defined as a function of the control deviation by the parameters *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 satisfied in all cases: *tPauseMin* < *tPauseMax* < *tInterval*
If an incorrect variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the preset value. *bInvalidParameter* will be set in the event of an incorrect parameter entry; the function block continues to work normally. The variables are saved persistently. *tPauseMin* preset to 2 s. *tPauseMax* preset to 8 s. *tInterval* preset to 10 s.

tStrokeTime: The variable *tStrokeTime* [s] 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 satisfied in all cases: *tStrokeTime* > *tInterval*
If an incorrect variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the preset value. *bInvalidParameter* will be set in the event of an incorrect parameter entry; the function block continues to work normally. The variable is saved persistently. Preset to 200 s.

tSwitchOverDeadTime: Dwell time when changing direction (0..3600). Both outputs are reset during this period.
If an incorrect variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the preset value. *bInvalidParameter* will be set in the event of an incorrect parameter entry; the function block continues to work normally. The variable is saved persistently. Preset to 500 ms.

eReferencingMode: Enum that specifies the referencing mode of the function block (see E_HVACReferencingMode [▸ 492]).
Depending on the features of the actuator used, the position is referenced depending on *eReferencingMode*. If *eReferencingMode = eHVACReferencingMode_Emulation*, the position of the drive will be calculated on the basis of the activation 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*. The associated output *bOpen* or *bClose* remains at TRUE after the respective limit switch has been reached.
If *eReferencingMode = eHVACReferencingMode_AnalogFeedback*, the position of the drive will be transferred on the basis of the signal *rFeedback*. The associated output *bOpen* or *bClose* remains at TRUE after the respective limit switch has been reached.
If an incorrect variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the preset value. *bInvalidParameter* will be set in the event of an incorrect parameter entry; the function block continues to work normally. The variable is saved persistently. Preset to 0.

Figure 1.1



```
IF rSetpoint >= rPosition THEN
        rControlDeviation := rSetpoint – rPosition;
END_IF

IF rSetpoint < rPosition THEN
        rControlDeviation := rPosition – rSetpoint;
END_IF
```

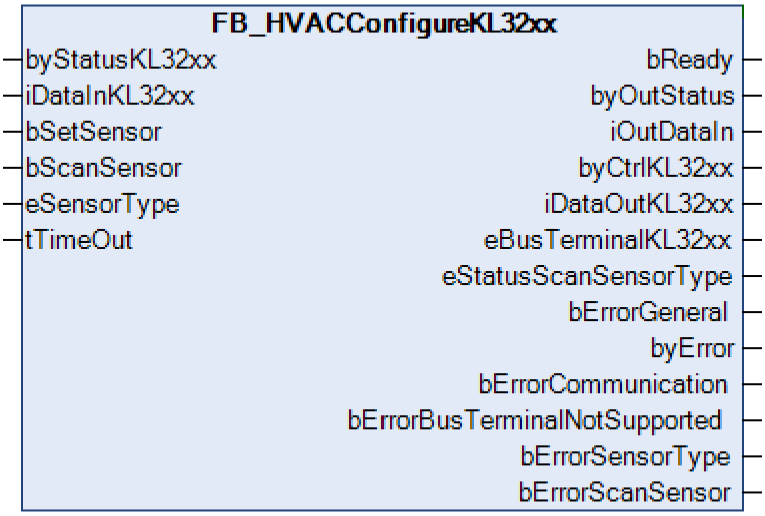
Table 1: Example

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

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.3.5 FB_HVACConfigureKL32xx



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 I/O area 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 [▶ 493].
The KS2000 configuration software is no longer needed for setting the temperature sensor.

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.

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.

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 I/O area 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 I/O area 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 [▸ 493] 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 [▸ 493].

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 [▸ 493].

eSensorType: Enum with which the sensor type for the respective channel is specified for setting via *bSetSensor* (see E_HVACSensorType [▸ 493]).

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 I/O area 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* (see E_HVACBusTerminal_KL32xx [▸ 489]).

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* (see E_HVACSensorType [▸ 493]).

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

byError: Delivers all error messages and warnings from 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 types.

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*.

Types of sensor selectable via the Enum eTemperatureCharacteristic

(see E_HVACBusTerminal_KL32xx [▶ 489])

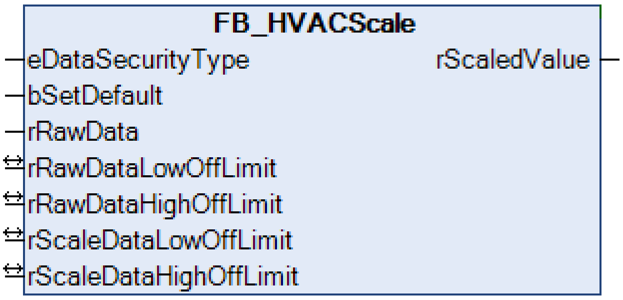
Sensor type	KL3201/2/4-0000		KL3201/2/4-0010		KL3201/2/4-0012		KL3201/2/4-0014		KL3201/2/4-0016		KL3201/2/4-0020	
		Raw value		Raw value		Raw value		Raw value		Raw value		Raw value
PT100	x	1/10°C	x	1/10°C	x	1/10°C	x	1/10°C	x	1/10°C	x	1/10°C
PT200	x	1/10°C	x	1/10°C	x	1/10°C	x	1/10°C	x	1/10°C	x	1/10°C
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(Tk5000,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	
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	x	1/10°C	not supported		not supported	
PT200	x	1/10°C	x	1/10°C	x	1/10°C	x	1/10°C	not supported		not supported	
PT500	x	1/10°C	x	1/10°C	x	1/10°C	x	1/10°C	not supported		not supported	
PT1000	x	1/10°C	x	1/10°C	x	1/10°C	x	1/10°C	x	1/10°C	x	1/100°C
Ni100	x	1/10°C	x	1/10°C	x	1/10°C	x	1/10°C	not supported		not supported	
Ni120	x	1/10°C	x	1/10°C	x	1/10°C	x	1/10°C	not supported		not supported	
Ni1000(DIN)	x	1/10°C	x	1/10°C	x	1/10°C	x	1/10°C	x	1/10°C	x	1/100°C
Ni1000(Tk5000,LS)	not supported		not supported		x	1/10°C	not supported		x	1/10°C	x	1/100°C
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	
Resistance measurement 10 ... 10000 Ω	not supported		not supported		not supported		not supported		not supported		not supported	

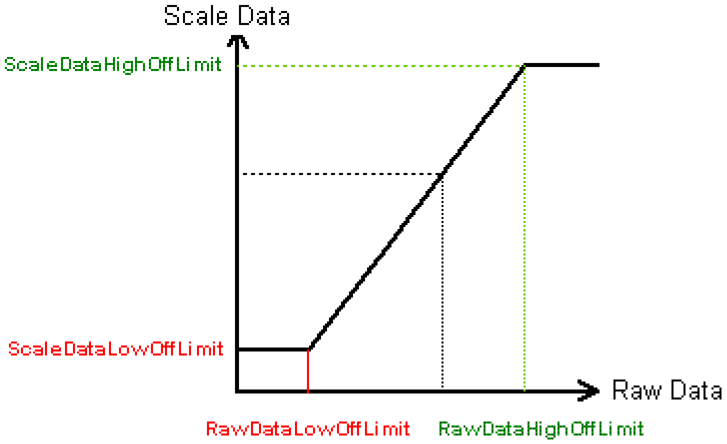
Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.3.6 FB_HVACScale



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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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.

rRawData: Raw value from the terminal.

VAR_OUTPUT

```
rScaledValue          :  REAL;
```

rScaledValue: Scaled value.

VAR_IN_OUT

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

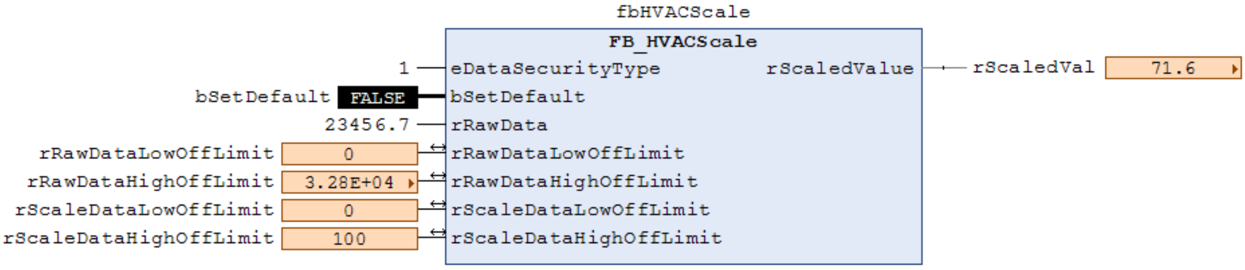
rRawDataLowOffLimit: Lower limit for raw value. The variable is saved persistently. Preset to 0.

rRawDataHighOffLimit: Upper limit for raw value. The variable is saved persistently. Preset to 32767.

rScaleDataLowOffLimit: Lower limit of scaled measured value. The variable is saved persistently. Preset to 0.

rScaleDataHighOffLimit: Upper limit of scaled measured value. The variable is saved persistently. Preset to 100.

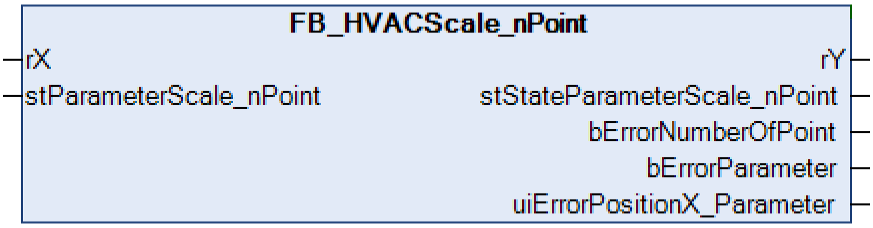
Example in FBD:



Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.3.7 FB_HVACScale_nPoint



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 on the XY axis of the characteristic curve are specified by the structure stParameterScale_nPoint [▸ 497].
Analog raw resistance values can be represented in the PLC with the KL32xx Bus Terminals. Detailed information can be found in the documentation on FB_HVACConfigureKL32xx [▸ 82].

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.

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.

VAR_INPUT

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

rX: Raw value that scales to the indicated measured value of the structure stParameterScale_nPoint [▶ 497] 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 [▶ 82].

stParameterScale_nPoint: Structure via which the individual points of the XY coordinates are given their valency (see ST_HVACParameterScale_nPoint [▶ 497]). 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 containing the valency of the individual points on the X axis. The number of points specified depends on *iNumberOfPoint*.
stParameterScale_nPoint.rY[1 ..NumberOfPoint] – 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 *bErrorParamter* = TRUE, then the state of the individual parameters in the structure *stStateParameterScale_nPoint* = 0. If *stParameterScale_nPoint.iNumberofPoint* = 20, then the individual parameters of *stStateParameterScale_nPoint.rX[1..20]* / *stStateParameterScale_nPoint.rY[1..20]* are displayed; from 21 to 60 the state = 0 (see ST_HVACParameterScale_nPoint [▶ 497]).

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: Care must be taken when parameterizing the structure stParameterScale_nPoint [▶ 497] 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, then *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 ST_HVACParameterScale_nPoint [▸ 497] (see Parameter [▸ 505]).

Formulas for the linear equation, two-point form:

m = gradient

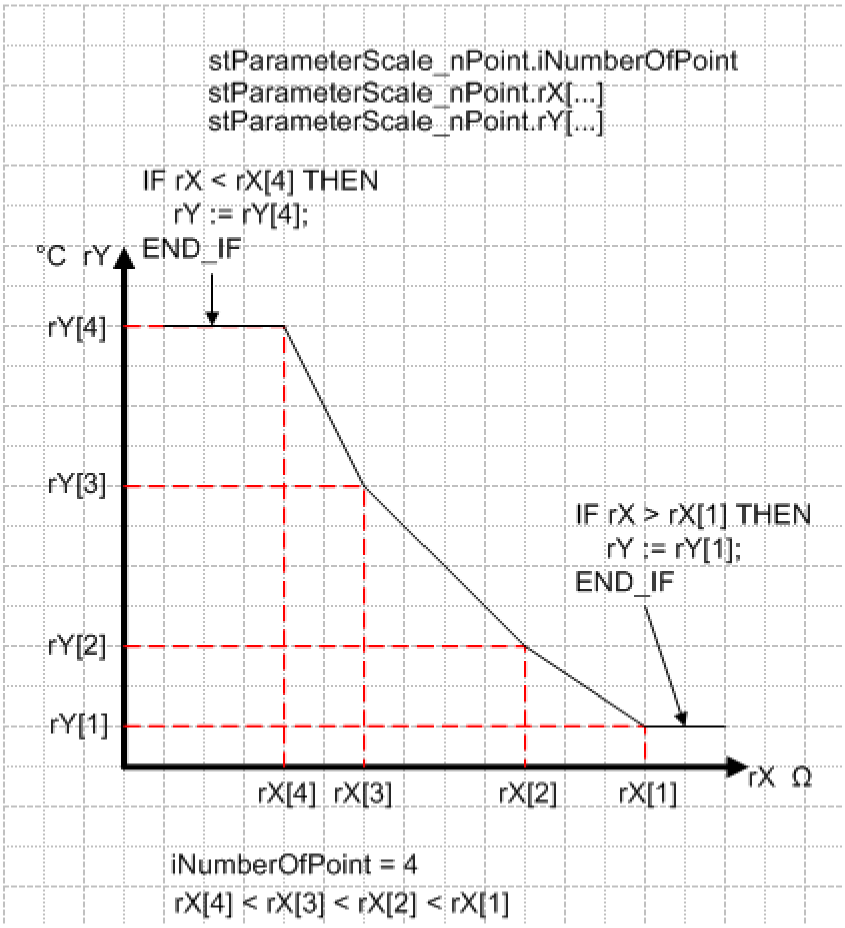
m = (Y2 - Y1) / (X2 - X1)

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

Y = Y1 + m * (X - X1)

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

Example curve 1 with calculation



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])

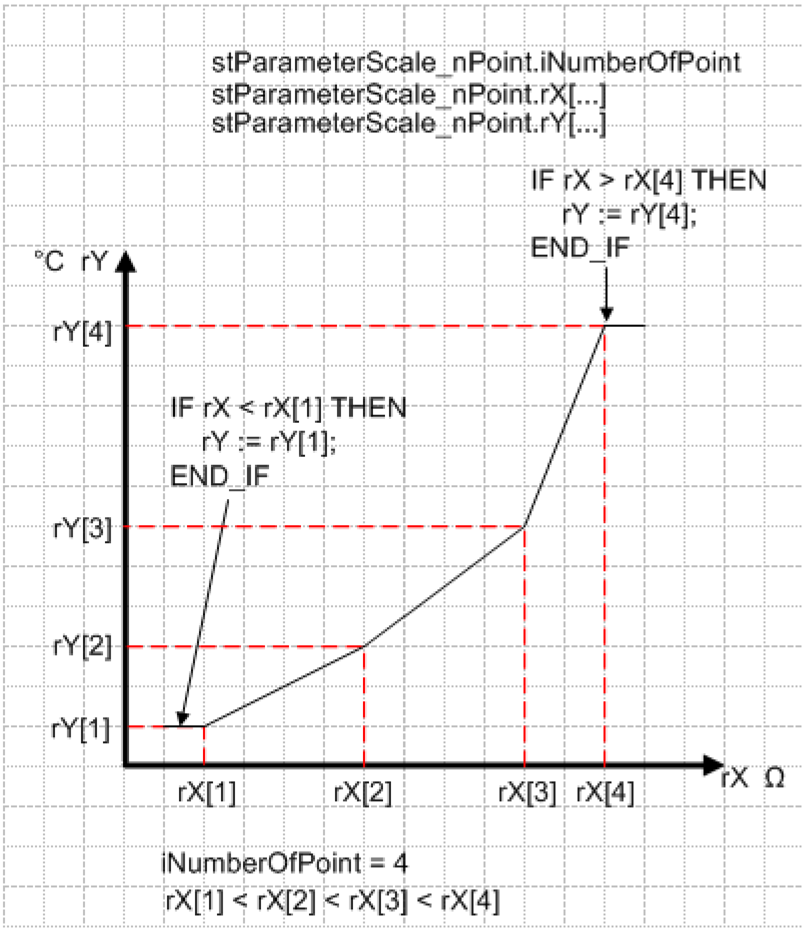
rM = (21 - 20) / (1163 - 1209) = -0.02174

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])

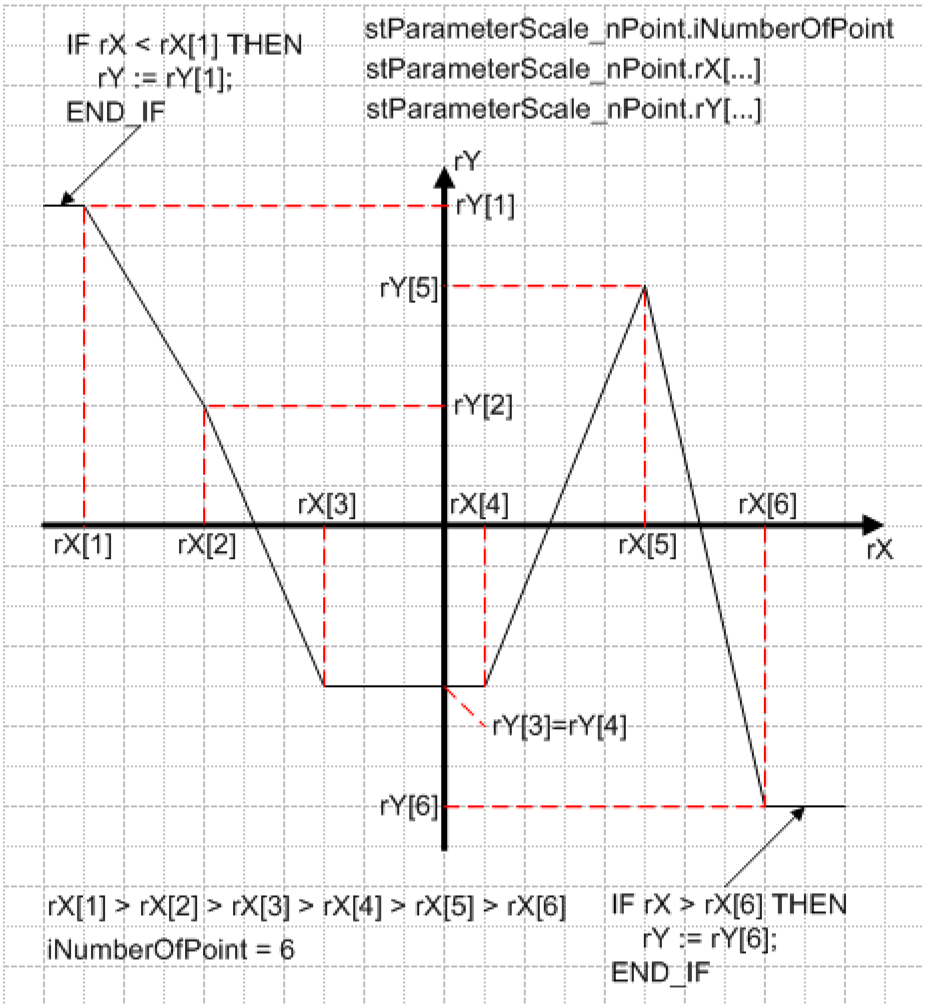
rM = (25 - 20) / 1142 - 1112) = 0.166

Y = Y3 + m * (X - X3)

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

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

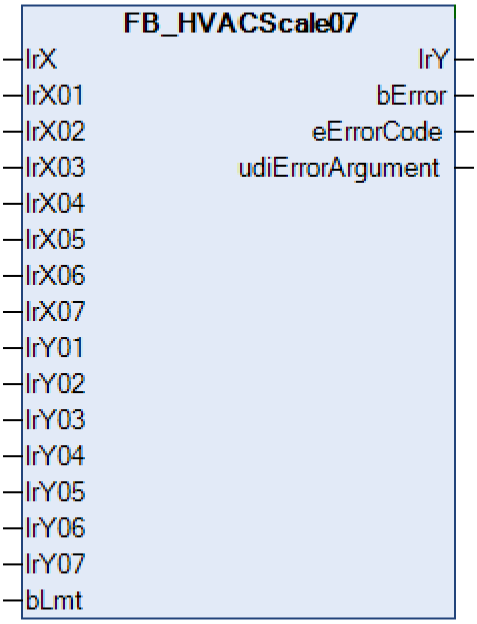
Example curve 3



Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.3.8 FB_HVACScaleXX



Characteristic curves with 2, 4 or 7 interpolation points can be mapped with the function blocks *FB_HVACScale02, FB_HVACScale04* and *FB_HVACScale07* (shown here). The analog raw value is applied to the input lrX and output as the calculated Y value at the output lrY. Different slopes of the characteristic curve can be specified in sections by specifying various interpolation points. The range is limited internally to the values lrY01 and lrY07 with the aid of the input *bLmt*.

VAR_INPUT

```
lrX   : LREAL;
lrX0n : LREAL;
lrY0n : LREAL;
bLmt  : BOOL;
```

lrX: Applied raw value.

lrX0n: X interpolation point of the characteristic curve. The characteristic curve has 2, 4 or 7 interpolation points, depending on the scaling function block.

lrY0n: X interpolation point of the characteristic curve. The characteristic curve has 2, 4 or 7 interpolation points, depending on the scaling function block.

bLmt: Input for limiting the output Y-range to values between lrY01 and lrY0n, depending on the maximum number of interpolation points.

VAR_OUTPUT

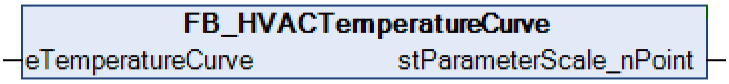
```
lrY               : LREAL;
bError            : BOOL;
eErrorCode        : E_HVACErrorCodes;
udiErrorArgument  : UDINT;
```

lrY: Calculated Y-value of the characteristic curve.
bError: Incorrect entry of the X interpolation points. The values must be entered in ascending order. No two X-values may be the same.
eErrorCode: The error message eHVACErrorCodes_ErrIdlrX0x [▶ 491] will be displayed.
udiErrorArgument: Indication of which interpolation point has been entered incorrectly.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.3.9 FB_HVACTemperatureCurve



A temperature curve stored internally in the function block is selected from Tables 1 and 2 via the Enum eTemperatureCurve [▸ 495] 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 [▸ 87]. 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 [▸ 82].

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.

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.

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 (see E_HVACTemperatureCurve [▸ 495]).

VAR_OUTPUT

```
stParameterScale_nPoint   : ST_HVACParameterScale_nPoint;
```

stParameterScale_nPoint: Structure containing the number and valences of the points of the XY coordinates (see ST_HVACParameterScale_nPoint [▸ 497]). *stParameterScale_nPoint* contains the curves stored in 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;

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) Ni1000Tk5000
eHVACTemperatureCharacteristic_	NTC1k_3_A1	NTC1k8_3_A1	NTC2k2_3_A1	NTC3k3_3_A1	Ni1000_DIN	Pt1000_DIN	Ni1000Tk5000_TCR
°C	Ω	Ω	Ω	Ω	Ω	Ω	Ω
- 50	32886				743	803	790.8
- 40	18641				791	843	826.8
- 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

	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) Ni1000Tk5000
eHVACTemperatureCharacteristic_	**NTC1k_3_A1**	**NTC1k8_3_A1**	**NTC2k2_3_A1**	**NTC3k3_3_A1**	**Ni1000_DIN**	**Pt1000_DIN**	**Ni1000Tk5000_TCR**
°C	**Ω**	**Ω**	**Ω**	**Ω**	**Ω**	**Ω**	**Ω**
35	694	1235	1500	1960	1200	1136	1161.5
40	583	1031	1248	1597	1230	1155	1185.7
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		81	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	

	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) Ni1000Tk5000
eHVACTemperatureCharacteristic_	NTC1k_3_A1	NTC1k8_3_A1	NTC2k2_3_A1	NTC3k3_3_A1	Ni1000_DIN	Pt1000_DIN	Ni1000Tk5000_TCR
°C	Ω	Ω	Ω	Ω	Ω	Ω	Ω
320						2191	
330						2227	
340						2262	
350						2297	
360						2332	
370						2367	
380						2401	
390						2436	
400						2470	

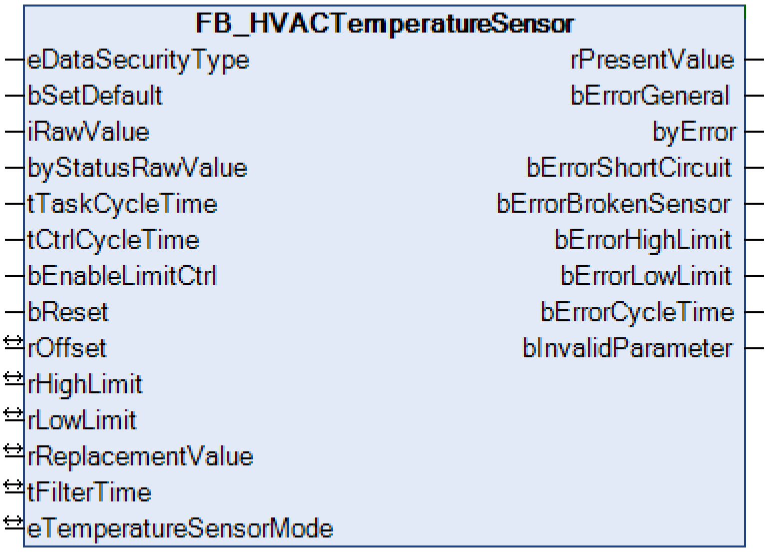
Specified temperature curves, selectable via the Enum eTemperatureCurve

	Thermokon Sensor type NTC 1.8 kOhm	Thermokon Sensor type Ni1000 Tk5000
eHVACTemperatureCharacteristic_	**NTC1K8**	**Ni1000Tk5000**
°C	**Ω**	**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

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.3.10 FB_HVACTemperatureSensor



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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 fault diagnosis, e.g. wire breakage 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 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: Delivers 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 in the connected temperature sensor. After rectification of the error, acknowledgment depends on the operation mode.

bErrorBrokenSensor: Error, wire breakage 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* >= *rHighLimit.* The warning that the upper limit value has been exceeded can only be acknowledged if *rPresentValue* <= *rHighLimit* -1.0 for a period of 5 seconds.

bErrorLowLimit: Warning: lower limit value undershot; goes 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 period of 5 seconds

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

bInvalidParameter: Indicates that an incorrect parameter is present at one of the variables *rOffset, rHighLimit, rLowLimit, rReplacementValue, tFilterTime or 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 no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value.
bInvalidParameter is set in the case of an incorrect parameter specification.

rHighLimit: Upper limit value (-250..+850); the output *bErrorHighLimit* is set if *rPresentValue* >= *rHighLimit*. *rHighLimit* must be greater than *rLowLimit*. The variable is saved persistently. Preset to 120.
If no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

rLowLimit: Lower limit value (-250..+850); the output *bErrorLowLimit* is set if *rPresentValue* <= *rLowLimit*. The variable is saved persistently. Preset to -60.
If no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

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 *eHVACTemperatureSensorMode_AutoResetReplacementValue.* The variable is saved persistently. Preset to 0.
If no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

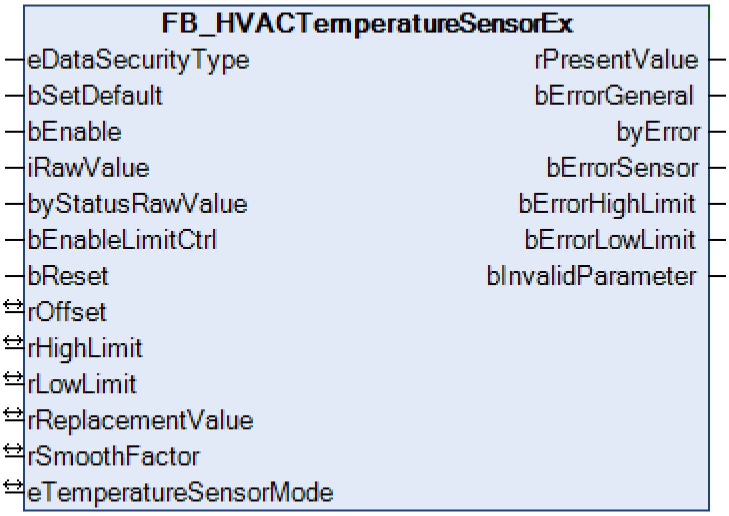
tFilterTime: Filter constant (1ms..100s). To avoid large fluctuations and jumps in the measuring signal, the function block is provided with a 2nd order filter. Upon a restart of the controller, or following correction of the error *bErrorShortCircuit* or *bErrorBrokenSensor*, the 2nd order filter is 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 no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

eTemperatureSensorMode: Enum that specifies the operation mode of the function block (see E_HVACTemperatureSensorMode [▶ 495]). 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 no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.3.11 FB_HVACTemperatureSensorEx



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 adapted to the Bus Terminals KL3201/02/04 and KL3228. These Bus Terminals can either be ordered preconfigured or set to the respective sensor type via the 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*, for example, can be linked directly with the raw temperature value of the following Bus Terminals: KL3201/02/04 and KL3228.

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

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 has been corrected and *bErrorSensor* = FALSE, then *rPresentValue* = rPresentValueOld for one PLC cycle.

The state 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 be linked, for example, directly with the status byte of the following Bus Terminals: KL3201/02/04 and KL3228. *rHighLimit/rLowLimit* can be used to specify temperature limit values.

In contrast to the FB_HVACTemperatureSensor [▶ 99] this function block has the input variable *bEnable*, which is useful if the characteristic sensor curves in the Bus Terminals KL3201/02/04 and KL3228 are to be set from the PLC via the function block FB_HVACConfigureKL32xx [▶ 82]. In this function block the second order filter in FB_HVACTemperatureSensor [▶ 99] 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*.

VAR_INPUT

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

eDataSecurityType: If eDataSecurityType [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 fault diagnosis, e.g. wire breakage or short circuit. If the KL32xx is set to resistance measurement (Ω) there is no fault 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 mode eTemperatureSensorMode [▶ 495], 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 has been corrected and *bErrorSensor* = FALSE, then *rPresentValue* = rPresentValueOld for one PLC cycle.
If *bErrorSensor* = TRUE, the value of *rPresentValue* depends on the operation mode eTemperatureSensorMode [▶ 495].

bErrorGeneral: The error message *bErrorGeneral* becomes TRUE if one of the error messages *bErrorHighLimit, bErrorLowLimit* or *bErrorSensor* isTRUE. The value of the output variable *rPresentValue* is then dependent on the operation mode eTemperatureSensorMode [▶ 495] and is enabled once the error has been rectified and, depending on the operation mode eTemperatureSensorMode [▶ 495], acknowledged with *bReset*.

byError: Delivers 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 in the connected temperature sensor. Once the fault has been corrected, the message is acknowledged either with *bReset* or automatically, depending on mode eTemperatureSensorMode [▶ 495].
byError.6 := *bErrorBrokenSensor:* Error, wire breakage in the connected temperature sensor. Once the fault has been corrected, the message is acknowledged either with *bReset* or automatically, depending on mode eTemperatureSensorMode [▶ 495].

bErrorSensor: Becomes TRUE if *byError*.5/*bErrorShortCircuit* or *byError*.6/*bErrorBrokenSensor* is TRUE. Once the fault has been corrected, the message is acknowledged either with *bReset* or automatically, depending on mode eTemperatureSensorMode [▶ 495]. 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 period of 5 seconds. Depending on mode eTemperatureSensorMode [▶ 495], the warning is acknowledged either with *bReset* or automatically.

bErrorLowLimit: Warning: lower limit value undershot; goes 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 period of 5 seconds. Depending on mode *eTemperatureSensorMode*, the warning is acknowledged either with *bReset* or automatically.

bInvalidParameter: Indicates that there is an incorrect parameter to one of the variables *rHighLimit, rLowLimit, rSmoothFactor* or eTemperatureSensorMode [▶ 495]. 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 [▶ 495].

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 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, then the last valid variable value, if one, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

rLowLimit: Lower limit value. If *rPresentValue* <= *rLowLimit* , then output *bErrorLowLimit* is set. The variable is saved persistently. Preset to -60.
If an incorrect variable value is present, then the last valid variable value, if one, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

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

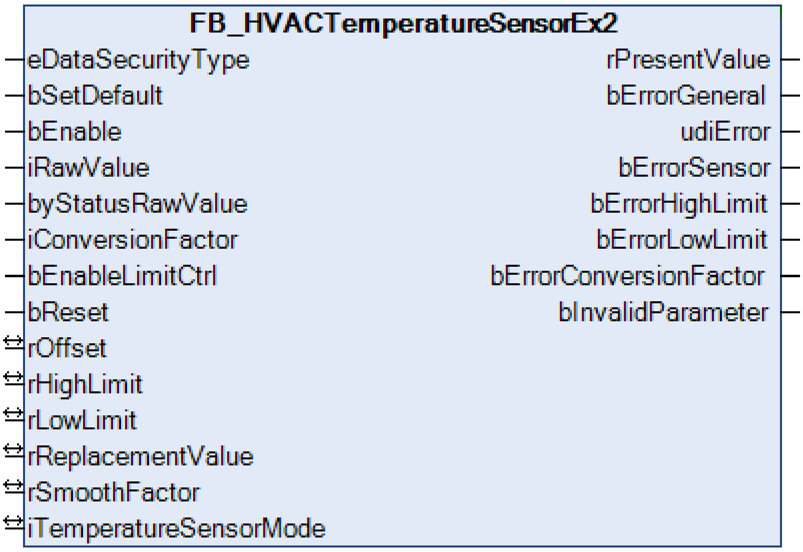
rSmoothFactor: Smoothing factor (>= 1) for the output value *rPresentValue*. The variable is saved persistently. Preset to 100.
The value for *rPresentValue* is calculated and output according to the following formula:
rPresentValue := ((iRawValue / 10 + rOffset) - rPresentValueOld) / rSmoothFactor + rPresentValueOld;
rPresentValueOld is the value of *rPresentValue* that was output in the previous PLC cycle.
If *rSmoothFactor* = 1, then *rPresentValue* := ((*iRawValue* / 10 + *rOffset*)
If an incorrect variable value is present at *rSmoothFactor* , then the last valid variable value, if one, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

eTemperatureSensorMode: Enum that specifies the operation mode of the function block (see E_HVACTemperatureSensorMode [▶ 495]).
eTemperatureSensorMode = *eHVACTemperatureSensorMode_ReplacementValue*: If *bErrorSensor* = 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 *bErrorSensor* = TRUE, then the last valid temperature value that was present 10 seconds before is output at the output variable *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 *bErrorSensor* = TRUE, then the last valid temperature value that was present 10 seconds before is output at the output variable *rPresentValue*. The function block acknowledges itself automatically following correction of the error.
If an incorrect variable value is present at *eTemperatureSensorMode*, the default value is used. *bInvalidParameter* is set in the case of an incorrect parameter specification.
The variable is saved persistently. Preset to 3.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.3.12 FB_HVACTemperatureSensorEx



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 respective sensor type via the software.
The raw temperature value is transferred either in 1/10 or 1/100 °C to the function block via the input variable

iRawValue and output via *rPresentValue* as a floating point number. *iRawValue*, for example, can be linked directly with 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:

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 has been corrected and *bErrorSensor* = FALSE, then *rPresentValue* = rPresentValueOld for one PLC cycle.

The state 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 be linked, for example, directly with 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 [▸ 99], 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 [▸ 82]. In this function block the second order filter in FB_HVACTemperatureSensor [▸ 99] is replaced by the smoothing function described above. The output *bErrorSensor* is new and replaces the two outputs bErrorShortCircuit/bErrorBrokenSensor. However, these outputs still exist in the error byte *udiError*.

Unlike FB_HVACTemperatureSensorEx [▸ 102], sensors with the raw temperature value 1/10 or 1/100 °C can be transferred on this function block.

VAR_INPUT

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

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 or 1/100 °C from the bus terminal.

byStatusRawValue: Status byte of the temperature sensor from the bus terminal. Serves for fault diagnosis, e.g. wire breakage or short circuit. If the KL32xx is set to resistance measurement (Ω) there is no fault diagnosis.

iConversionFactor: Conversion factor (0..1) 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 at *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 has been corrected and *bErrorSensor* = FALSE, then *rPresentValue* = rPresentValueOld for one PLC cycle.
If *bErrorSensor* = TRUE, the value of *rPresentValue* depends on the operation mode *iTemperatureSensorMode*.

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

udiError: Delivers 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 in the connected temperature sensor. Once the fault has been corrected, the message is acknowledged either with *bReset* or automatically, depending on mode *iTemperatureSensorMode*.
byError.6 := *bErrorBrokenSensor:* error, wire breakage in the connected temperature sensor. Once the fault has been corrected, the message is acknowledged either with *bReset* or automatically, depending on mode *iTemperatureSensorMode*.

bErrorSensor: Goes 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 period of 5 seconds. Depending on mode *iTemperatureSensorMode*, the warning is acknowledged either with *bReset* or automatically.

bErrorLowLimit: Warning: lower limit value undershot; goes 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 period 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 there is an incorrect parameter on 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 output *bErrorHighLimit* is set. *rHighLimit* must be greater than *rLowLimit*.
If an incorrect variable value is present, then the last valid variable value, if one, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

rLowLimit: Lower limit value. The variable is saved persistently. Preset to -60.
If *rPresentValue* <= *rLowLimit* , then output *bErrorLowLimit* is set.
If an incorrect variable value is present, then the last valid variable value, if one, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

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 *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, if one, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

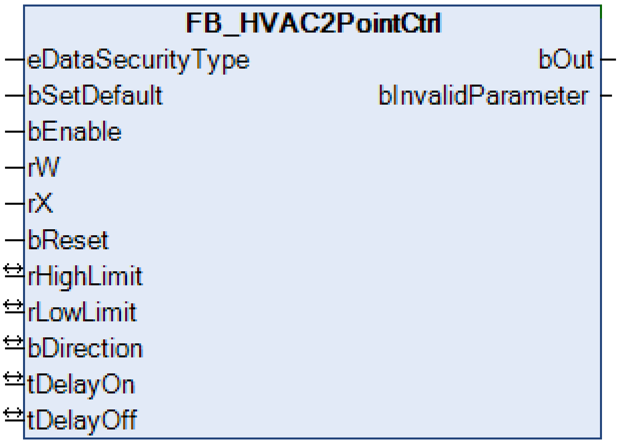
iTemperatureSensorMode: Specifies the operation mode of the function block.
iTemperatureSensorMode = *0*: If *bErrorSensor* = TRUE, then *rPresentValue* = *rReplacementValue*. After correction of the error, the function block must be acknowledged with a rising edge at the input variable *bReset*.
iTemperatureSensorMode = *1*: If *bErrorSensor* = TRUE, then the last valid temperature value that was present 10 seconds before is output at the output variable *rPresentValue*. After correction of the error, it must be acknowledged with a rising edge at the input variable *bReset*.
iTemperatureSensorMode = *2* : If *bErrorShortCircuit* or *bErrorBrokenSensor* = TRUE, then *rPresentValue* = *rReplacementValue.* The function block acknowledges itself automatically following correction of the error.
iTemperatureSensorMode = *3*: If *bErrorSensor* = TRUE, then the last valid temperature value that was present 10 seconds before is output at the output variable *rPresentValue*. The function block acknowledges itself automatically following correction of the error.
If there is an incorrect variable value on *iTemperatureSensorMode*, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.
The variable is saved persistently. Preset to 3.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.4 HVAC Controller

5.1.4.1 FB_HVAC2PointCtrl



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 [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 *rW* variable.

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 two-point controller.

bInvalidParameter: TRUE if an error has occurred 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 two-point controller in relation to the upper/lower limits of the control deviation, with no switch-on or release delay.

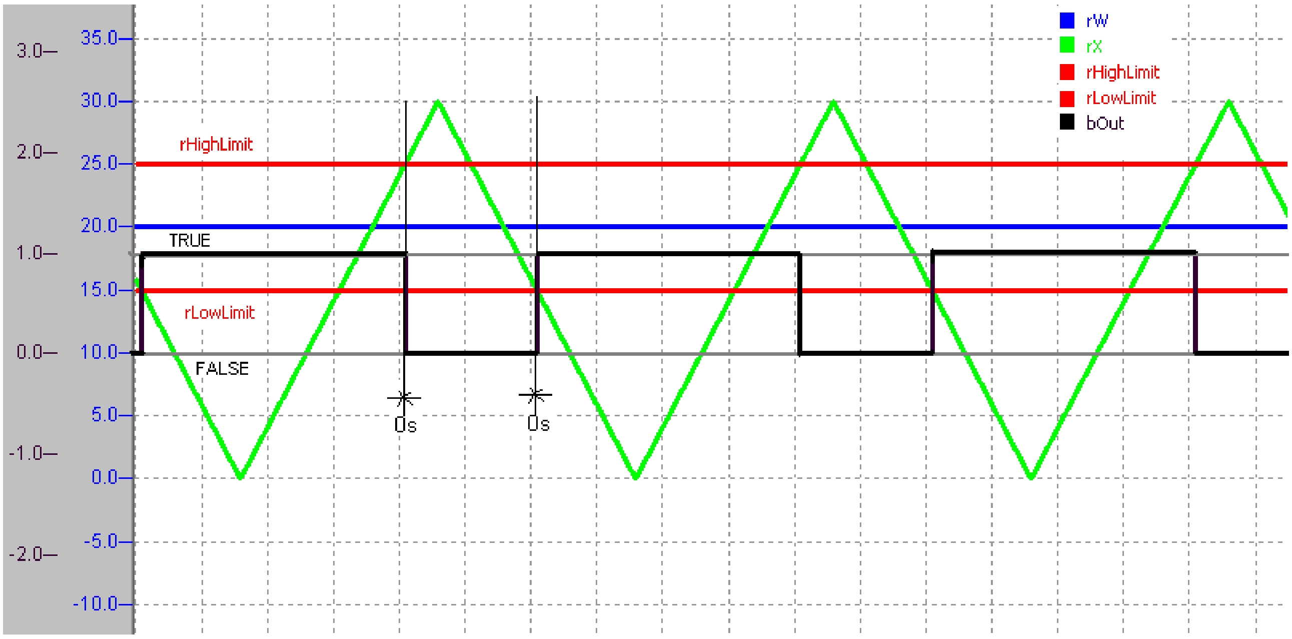


Figure: FB_HVAC2PointCtrl timing diagram

rHighLimit = 5
rLowLimit = 5
bDirection = FALSE
tDelayOn = 0s
tDelayOff = 0s

Figure 2:

Curve of switching output *bOut* for a two-point controller in relation to the upper limit *rHighLimit* = 3 and the lower limit *rLowLimit* = 8 with a switch-on delay of *tDelayOn* = 4s and a release delay of *tDelayOff* = 6s.

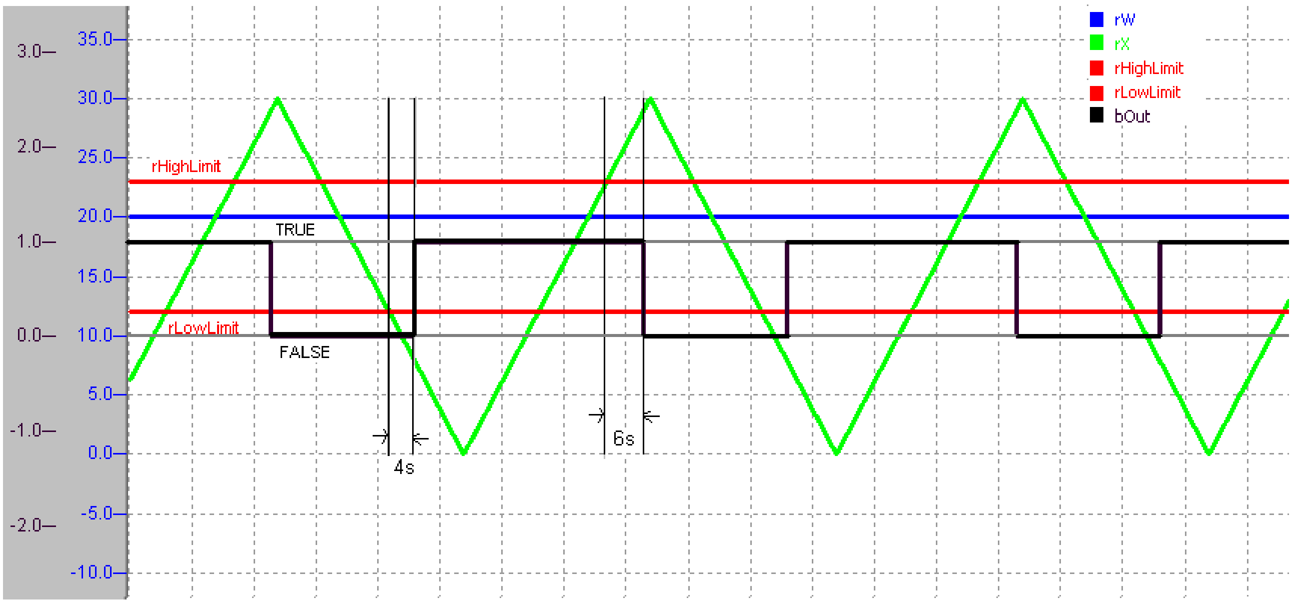


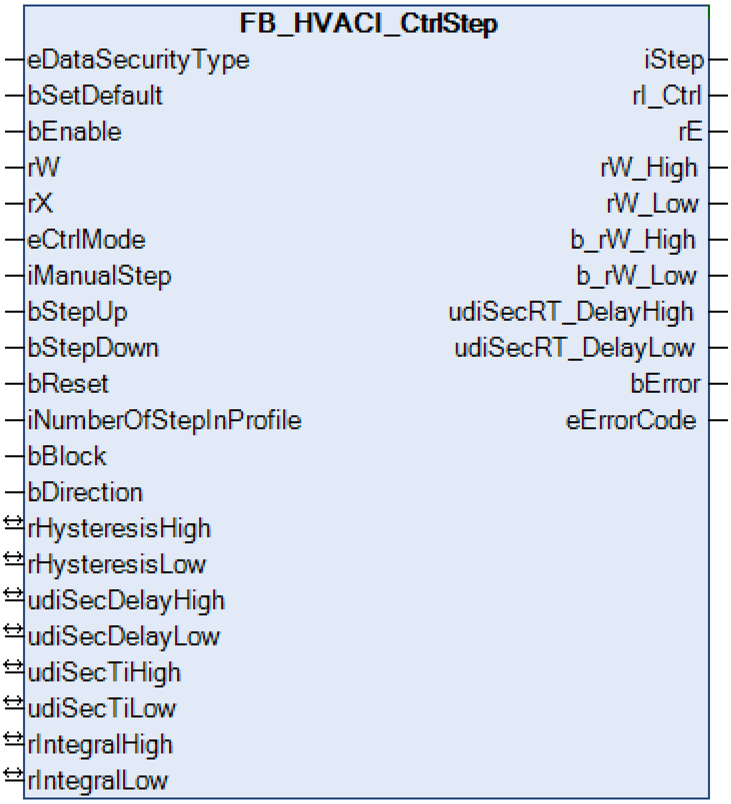
Figure: FB_HVAC2PointCtrl timing diagram

rHighLimit = 3
rLowLimit = 8
bDirection = FALSE
tDelayOn = 4s
tDelayOff = 6s

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.4.2 FB_HVACI_CtrlStep



The function block serves the sequential control of power generators.

In conjunction with the power range table FB_HVACPowerRangeTable [▶ 135], the power step controller can be used for the control of several 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 [▶ 115]

An integration time setting of udiSecTiHigh/udiSecTiLow = 60 seconds means that the I-transfer element changes by one Kelvin per minute. In the case of a control deviation of *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
For the remaining control deviation of 2 and the lower integral limit of 10, *rI_Ctrl* = *rIntegralLow* after 5 minutes. 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.

iStep is set to 1 for one PLC cycle if *bEnable* = TRUE, *bError* = FALSE, eCtrlMode [▶ 490] = 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 / udiSecDelay*Low 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

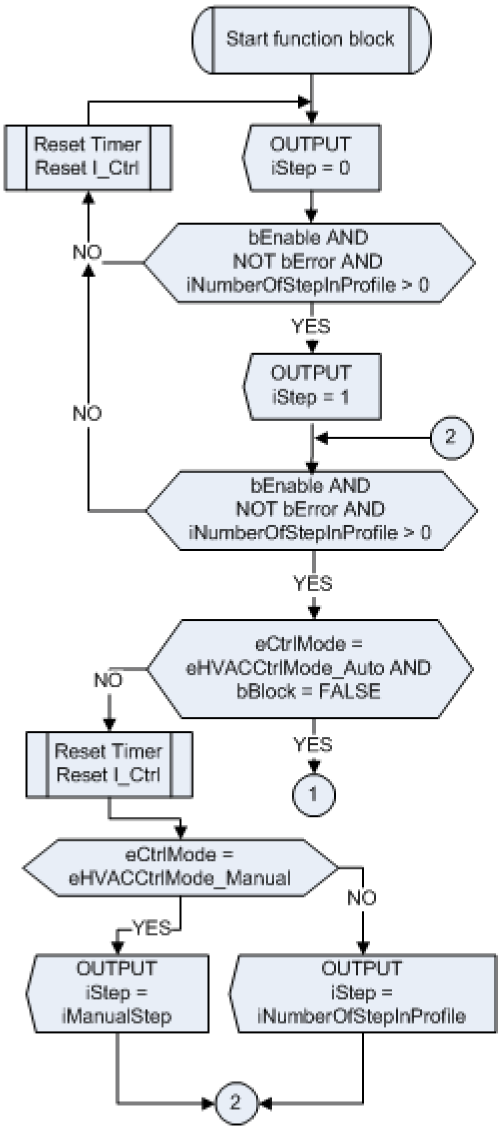
(*rX* < *rW_Low* AND *iStep* > 0 AND *bDirection*)
OR
(*rX* >= *rW_High* AND *iStep* > 0 AND NOT *bDirection*)

The I-transfer element and the internal time delays (*udiSecDelayLow*, *udiSecDelayHigh*) are reset after each change of *iStep*.

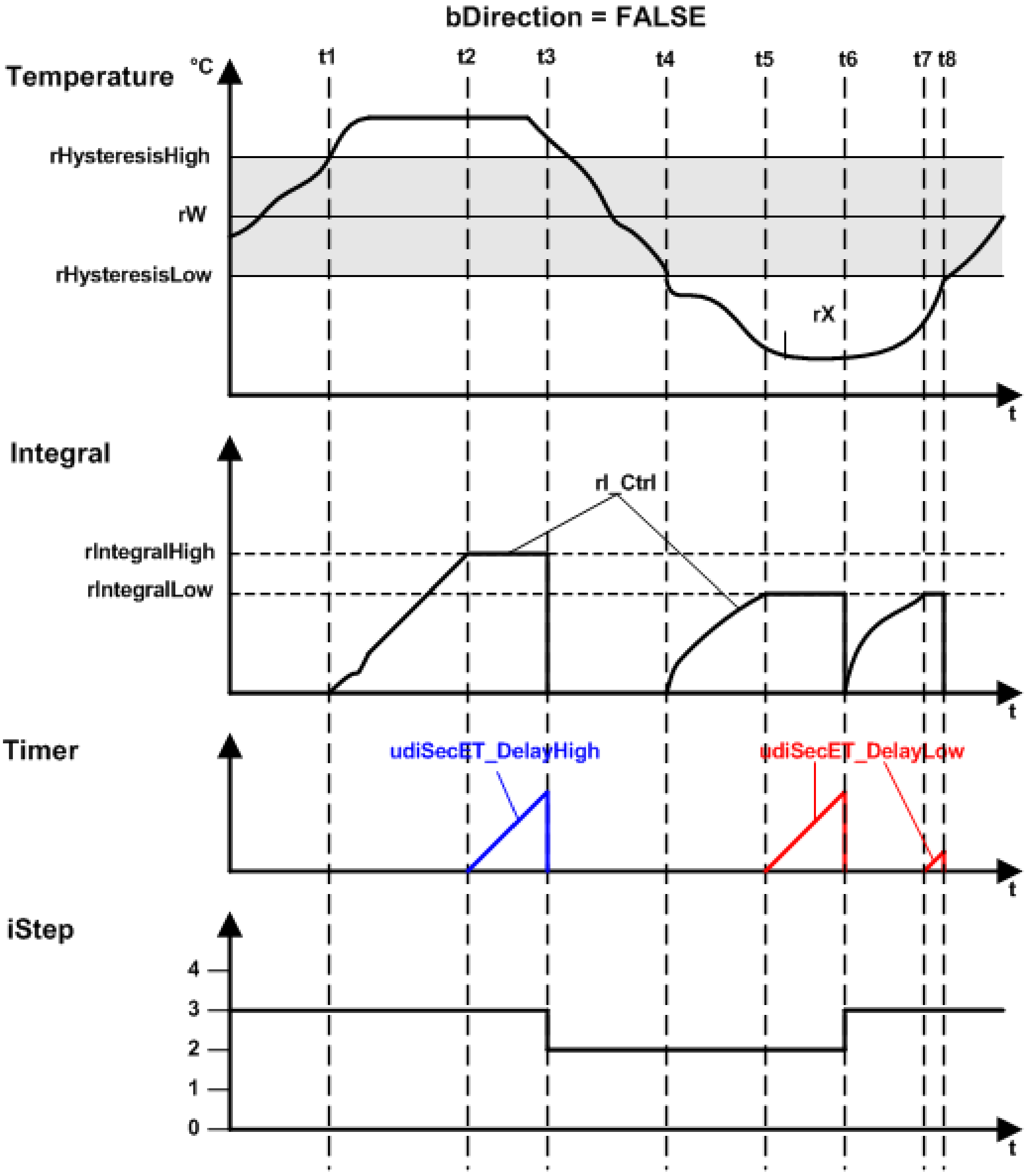
Transfer function of the I-transfer element (I_Ctrl)

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

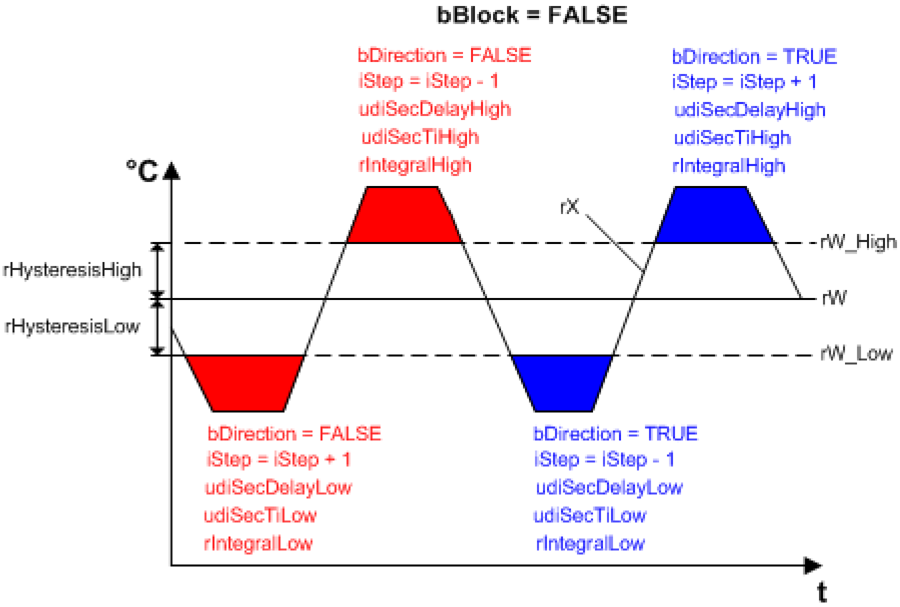
Program flowchart



Program flowchart



Behavior of the different variables



The figure shows when each variable is used and how.

VAR_INPUT

```
eDataSecurityType         : E_HVACDataSecurityType;
bSetDefault               : BOOL;
bEnable                   : BOOL;
rW                        : REAL;
rX                        : REAL;
eCtrlMode                 : E_HVACCtrlMode;
iManualStep               : INT;
bStepUp                   : BOOL;
bStepDown                 : BOOL;
bDirection                : BOOL;
iNumberOfStepInProfile    : INT;
bBlock                    : BOOL;
bReset                    : BOOL;
```

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 *rW* variable.

rX: The variable *rX* is used to transfer the actual value.

eCtrlMode: Enum that specifies the operation mode of the function block (see E_HVACCtrlMode [▸ 490]). 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: *bDirection* determines the control direction of the function block. FALSE = heating mode; TRUE = cooling mode

iNumberOfStepInProfile: Number of steps in the power step sequence. *iNumberOfStepInProfile* can be 0 at the least and the value of g_iMaxNumberOfSteps at the most. 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 to 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;
rI_Ctrl                 : REAL;
rE                      : REAL;
rW_High                 : REAL;
rW_Low                  : REAL;
b_rW_High               : BOOL;
b_rW_Low                : BOOL;
udiSecRT_DelayHigh      : UDINT;
udiSecRT_DelayLow       : UDINT;
bError                  : BOOL;
eErrorCode              : E_HVACErrorCodes;
```

iStep: The output variable *iStep* indicates the step in a power step sequence. On starting the function block, *iStep* = 1 if *bEnable* = TRUE, *bError* = FALSE, *eCtrlMode* = *eHVACCtrlMode_Auto* AND *iNumberOfStepInProfile* > 0. *iStep* can reach the value of *iNumberOfStepInProfile* at the most and can be 0 at the least.

The value of *iStep* can be specified manually 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* >= *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*), see Program flowchart [▸ 114]

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* >= *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 [▸ 114].

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 F [▸ 112]unctional description or Behavior of different variables [▸ 115].

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 F [▸ 112]unctional description or Behavior of different variables [▸ 115].

b_rW_High: *b_rW_High* goes TRUE if *rX* > *rW_High*.

b_rW_Low: *b_rW_Low* goes TRUE if *rX* < *rW_Low*.

udiSecRT_DelayHigh: Remaining time of the delay time *udiSecDelayHigh*.

udiSecRT_DelayLow: Remaining time of the delay time *udiSecDelayLow*.

bError: This output signals with a TRUE that there is an error and that there is a wrong parameter on the variable *iNumberOfStepInProfile*. *iStep* is set constantly to 0 and the Enum *eErrorCode* indicates the error code. Once the fault has been corrected the *bError* message must be acknowledged with *bReset*.

eErrorCode: returns the error number if an output *bError* is set (*see* E_HVACErrorCodes [▸ 491]). The following error can occur in this function block: *eHVACErrorCodes_InvalidParam_NumberOfStepInProfile*.

To access the enum error number in the PLC eErrorCode [▸ 491] can assigned to a variable of 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 Functional description [▸ 112] or Behavior of different variables [▸ 115]. *rW_High* := *rW* + *rHysteresisHigh.* The variable is saved persistently. Preset to 5.

rHysteresisLow: Positive value of the lower limit of the control deviation; see Functional description [▸ 112] or Behavior of different variables [▸ 115]. *rW_Low* := *rW* – *rHysteresisLow.* The variable is saved persistently. Preset to 5.

udiSecDelayHigh: Delay time (0..g_udiMaxSec) after whose expiry *iStep* is incremented or decremented, see Functional description [▸ 112] or Behavior of different variables [▸ 115]. The variable is saved persistently. Preset to 300 s.

udiSecDelayLow: Delay time (0..g_udiMaxSec) after whose expiry *iStep* is incremented or decremented, see Functional description [▸ 112] or Behavior of different variables [▸ 115]. The variable is saved persistently. Preset to 300 s.

udiSecTiHigh: Integration time (0..g_udiMaxSec) for the upper limit of the I-transfer element in seconds; see Functional description [▸ 112] or Behavior of different variables [▸ 115]. *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 (0..g_udiMaxSec) for the lower limit of the I-transfer element; see Functional description [▸ 112] or Behavior of different variables [▸ 115]. *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.

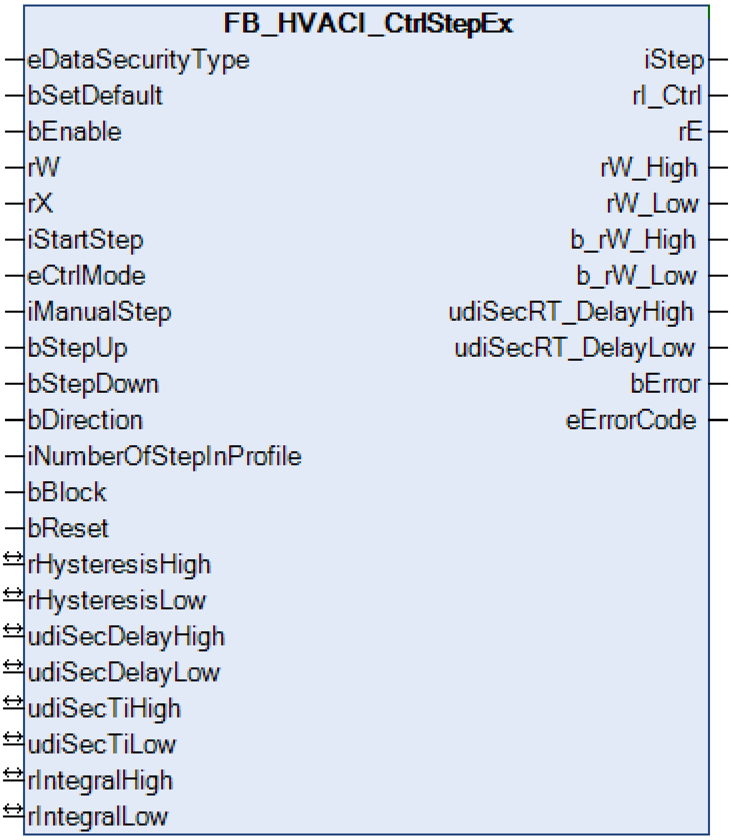
rIntegralHigh: Positive value for the upper limit at which the integration of the I-transfer element is stopped (anti-reset-windup measure) and one of the delay times starts; see Functional description [▸ 112] or Behavior of different variables [▸ 115]. The variable is saved persistently. Preset to 15.

rIntegralLow: Positive value for the lower limit at which the integration of the I-transfer element is stopped (anti-reset-windup measure) and one of the delay times starts; see Functional description [▸ 112] or Behavior of different variables [▸ 115]. The variable is saved persistently. Preset to 15.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.4.3 FB_HVACI_CtrlStepEx



Functional description

The function block serves the sequential control of power generators. It differs from FB_HVACI_CtrlStep [▸ 112] 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 [▸ 135], the power step controller can be used for the stepped control of several 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 [▸ 122]

An integration time setting of *udiSecTiHigh*/*udiSecTiLow* = 60 seconds means that the I-transfer element changes by one Kelvin per minute. In the case of a control deviation of *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
For the remaining control deviation of 2 and the lower integral limit of 10, *rI_Ctrl* = *rIntegralLow* after 5 minutes. 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.

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 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 / udiSecDelay*Low 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

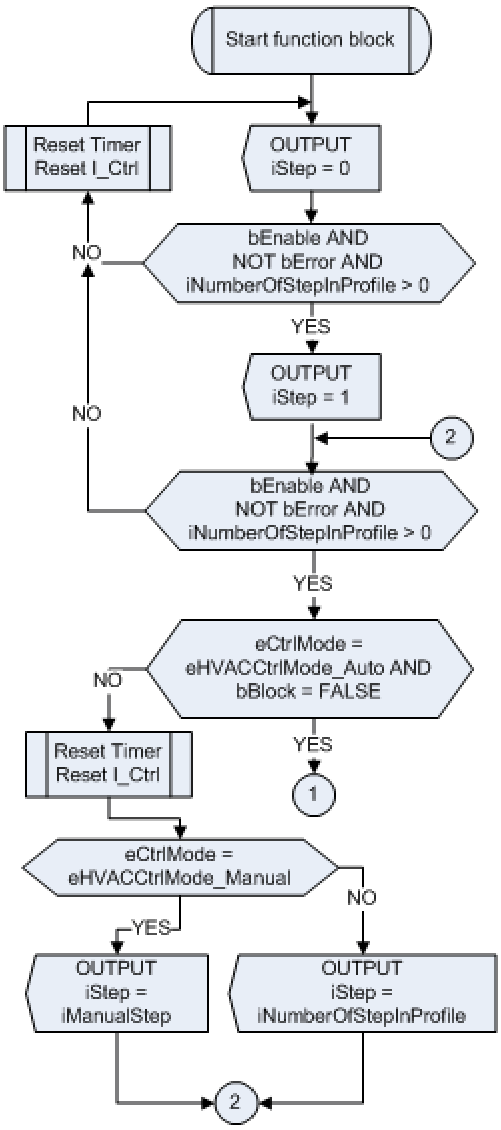
(*rX* < *rW_Low* AND *iStep* > 0 AND *bDirection*)
OR
(*rX* >= *rW_High* AND *iStep* > 0 AND NOT *bDirection*)

The I-transfer element and the internal time delays (*udiSecDelayLow*, *udiSecDelayHigh*) are reset after each change of *iStep*.

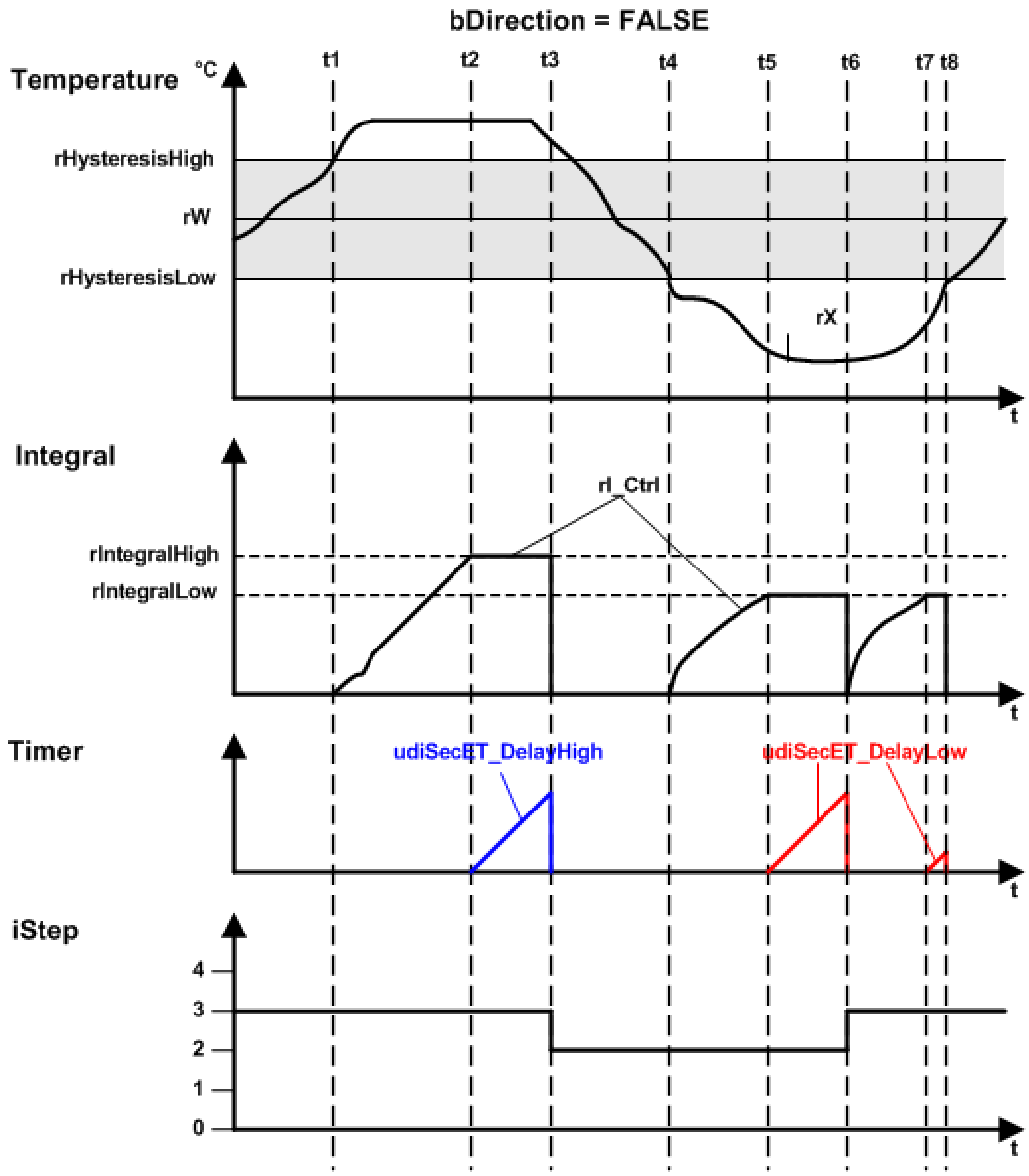
Transfer function of the I-transfer element (I_Ctrl)

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

Program flowchart

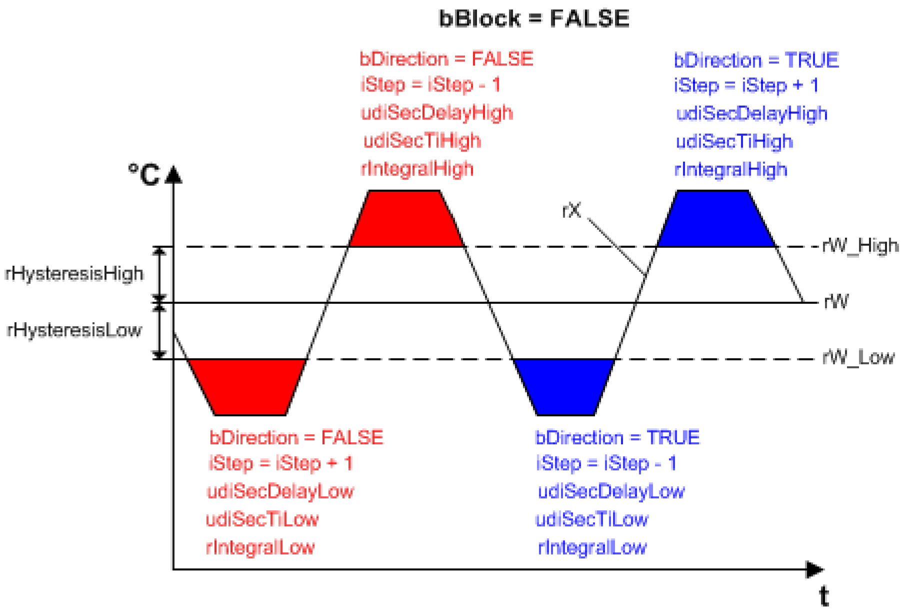


Program flowchart



Program flowchart

Behaviour of the different variables



The figure shows when each variable is used and how.

VAR_INPUT

```
eDataSecurityType          : E_HVACDataSecurityType;
bSetDefault                : BOOL;
bEnable                    : BOOL;
rW                         : REAL;
rX                         : REAL;
iStartStep                 : INT;
eCtrlMode                  : E_HVACCtrlMode;
iManualStep                : INT;
bStepUp                    : BOOL;
bStepDown                  : BOOL;
bDirection                 : BOOL;
iNumberOfStepInProfile     : INT;
bBlock                     : BOOL;
bReset                     : BOOL;
```

eDataSecurityType: If eDataSecurityType [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 *rW* variable.

rX: The variable *rX* is used to transfer the actual value.

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: *bDirection* determines the control direction of the function block. FALSE = heating mode; TRUE = cooling mode

iNumberOfStepInProfile: Number of steps in the power step sequence. *iNumberOfStepInProfile* can be 0 at the least and the value of g_iMaxNumberOfSteps at the most. 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 to 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;
rI_Ctrl              : REAL;
rE                   : REAL;
rW_High              : REAL;
rW_Low               : REAL;
b_rW_High            : BOOL;
b_rW_Low             : BOOL;
udiSecRT_DelayHigh   : UDINT;
udiSecRT_DelayLow    : UDINT;
bError               : BOOL;
eErrorCode           : E_HVACErrorCodes;
```

iStep: The output variable *iStep* indicates the step in a power step sequence. *iStep* = *iStartStep* on starting the function block if *bEnable* = TRUE AND *bError* = FALSE. *iStep* can reach the value of *iNumberOfStepInProfile* at the most and can be 0 at the least.

The value of *iStep* can be specified manually 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* >= *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*), see Program flowchart [▶ 121]

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* >= *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 [▶ 121].

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 Functional description [▶ 119] or Behavior of different variables [▶ 123].

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 Functional description [▶ 119] or Behavior of different variables [▶ 123].

b_rW_High: *b_rW_High* goes TRUE if *rX* > *rW_High*.

b_rW_Low: *b_rW_Low* goes TRUE if *rX* < *rW_Low*.

udiSecRT_DelayHigh: Remaining time of the delay time *udiSecDelayHigh*.

udiSecRT_DelayLow: Remaining time of the delay time *udiSecDelayLow*.

bError: This output signals with a *TRUE* that there is an error and that there is a wrong parameter on the variable iNumberOfStepInProfile. *iStep* is set constantly to 0 and the Enum *eErrorCode* indicates the error code. Once the fault has been corrected the *bError* message must be acknowledged with *bReset*.

eErrorCode: Returns the error number if a *bError* output is set (see E_HVACErrorCodes [▶ 491]). The following error can occur in this function block: *eHVACErrorCodes_InvalidParam_NumberOfStepInProfile*

To access the enum error number in the PLC eErrorCode [▶ 491] can assigned to a variable of 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 Functional description [▶ 119] or Behavior of different variables [▶ 123]. *rW_High* := *rW* + *rHysteresisHigh.* The variable is saved persistently. Preset to 5.

rHysteresisLow: Positive value of the lower limit of the control deviation; see Functional description [▶ 119] or Behavior of different variables [▶ 123]. *rW_Low* := *rW* – *rHysteresisLow.* The variable is saved persistently. Preset to 5.

udiSecDelayHigh: Delay time (0..g_udiMaxSec) after whose expiry *iStep* is incremented or decremented, see Functional description [▶ 119] or Behavior of different variables [▶ 123]. The variable is saved persistently. Preset to 300 s.

udiSecDelayLow: Delay time (0..g_udiMaxSec) after whose expiry *iStep* is incremented or decremented, see Functional description [▶ 119] or Behavior of different variables [▶ 123]. The variable is saved persistently. Preset to 300 s.

udiSecTiHigh: Integration time (0..g_udiMaxSec) for the upper limit of the I-transfer element in seconds; see Functional description [▶ 119] or Behavior of different variables [▶ 123]. *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 (0..g_udiMaxSec) for the lower limit of the I-transfer element; see Functional description [▶ 119] or Behavior of different variables [▶ 123]. *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.

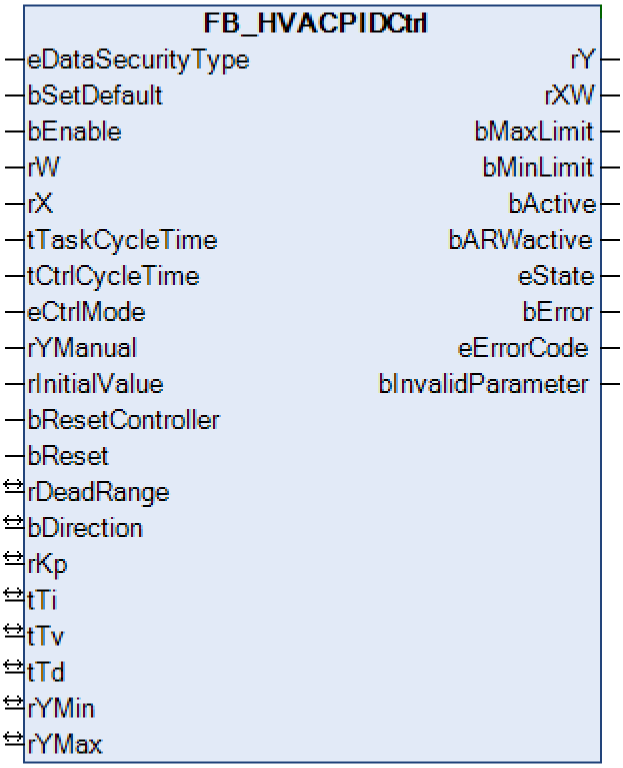
rIntegralHigh: Positive value for the upper limit at which the integration of the I-transfer element is stopped (anti-reset-windup measure) and one of the delay times starts; see Functional description [▶ 119] or Behavior of different variables [▶ 123]. The variable is saved persistently. Preset to 15.

rIntegralLow: Positive value for the lower limit at which the integration of the I-transfer element is stopped (anti-reset-windup measure) and one of the delay times starts; see Functional description [▶ 119] or Behavior of different variables [▶ 123]. The variable is saved persistently. Preset to 15.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.4.4 FB_HVACPIDCtrl



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 anti-reset-windup function. 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 [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 enabled 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 (see E_HVACCtrlMode [▶ 490]).

rYManual: The controller output *rY* can be overridden for test purposes. The setpoint 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 on a rising edge at the controller enable input *bEnable*.
0 = start with the value zero at 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 variable limit.

eState: State of the controller (see E_HVACState) [▶ 494].

bError: This output indicates with a TRUE that there is an error.

eErrorCode: Contains the command-specific error code (see E_HVACErrorCodes [▶ 491]).

bInvalidParameter: TRUE if an error has occurred 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: Integration time [s]. The I-component corrects the residual control deviation following correction of the P-component. 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 actuator. 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.

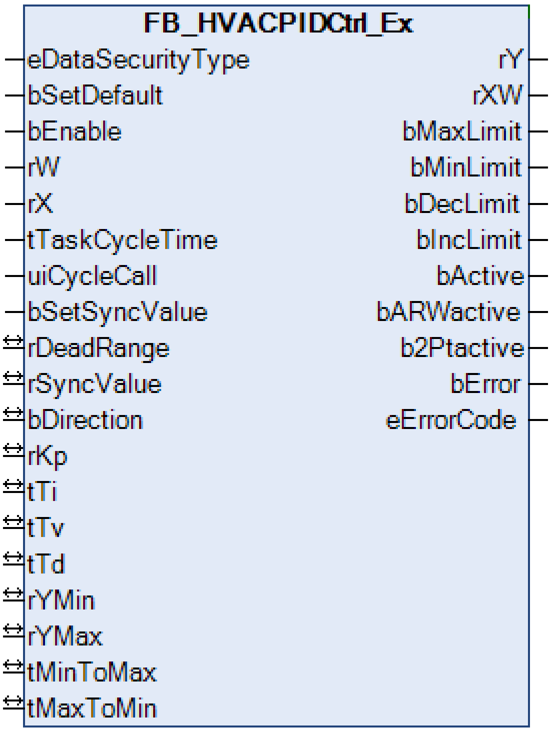
tTd: Damping time [s]. The variable is saved persistently. Preset to 0 s.

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 setpoint for an exhaust/supply air cascade control. The variables are saved persistently. *rYMin* preset to 0. *rYMax* preset to 100.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.4.5 FB_HVACPIDCtrl_Ex



Universal PID controller.

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 anti-reset-windup function. This prevents continuous integration in the event of a perpetual pending control deviation. In addition to FB_HVACPIDCtrl [▶ 127], the controller provides a slope compensation at the output.

Passive-behaviour (bEnable = FALSE or bError = TRUE)

The outputs are set in the followiog scheme:

rY	0,0
rXW	0,0
bMaxLimit	FALSE
bMinLimit	FALSE
bDecLimit	FALSE
bIncLimit	FALSE
bActive	FALSE
bARWactive	FALSE
b2Ptactive	FALSE

In case of an error the output *bError* is set to TRUE - *eErrorCode* displays the actual errorcode. The internal values for the P-, I-, and D-part are set to "0", as well as the repective values for the previous cycle. Thus the controller can restart "clean" without any past 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, then the controller ensures that this is held constant if the controller output *rY* should be at the limits *rYMin* or *rYMax* and attempts to fall or rise further. This procedure is referred to as Anti-Reset-Windup. 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

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	lrX-lrW (actual value setpoint)	direct (cooling)
FALSE	lrW-lrX (setpoint 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 if the minimum values is reached at the output, while the control deviation is still negative.
To prevent this in the event of leaving the control range, the I-component is set in such a way that, in addition to the P and D-components, it reaches the respective limit value *lrYmin* or *lrYMax* at the control output.
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 lrY 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 limitation of controller output for increase: *tMinToMax* in seconds in reference to a change from *rYMin* to *rYMax*.

tMaxToMin: Slope limitation of controller output for decrease: *tMaxToMin* in seconds in reference 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).

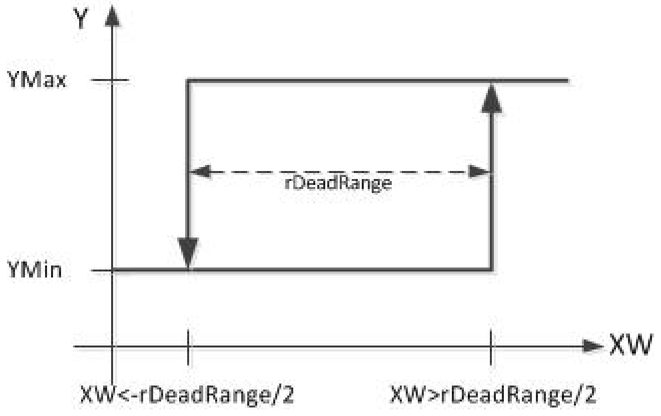
Deadrange

If the setpoint is finally reached, the actual value may still swing around it, which causes swinging at the control-output and constant movement at the actuator. This can be surpressed by defining a deadrange around the setpoint. If the absolute value of the control-deviation is smaller than half of the defined *rDeadRange*, the P- I- and D-Part will be frozen and not calculated any more until the deviation gets above 0,5**rDeadRange* again.

A zero-entry at this input deactivates the deadrange.

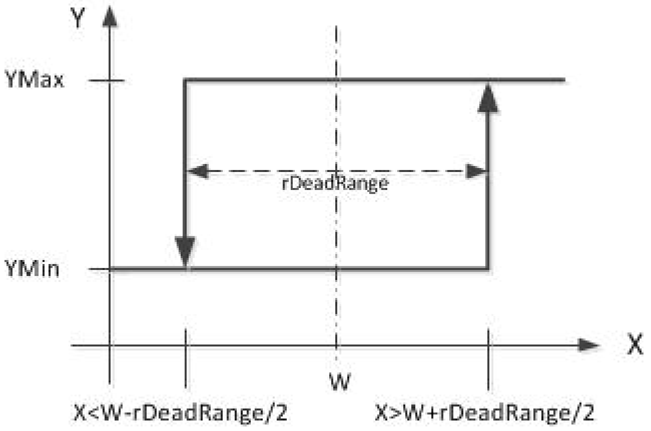
2-point control behavior

If the control parameters *Ir*, *rKp*, *tTn*, *tTv* and *tTd* are set to 0.0 or t#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:*

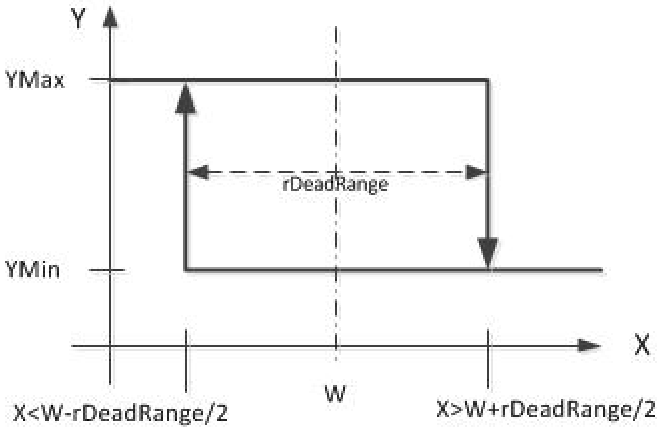


The different calculation methods for the control deviation for direct and indirect control direction result in the following switching characteristics in relation to the actual value *rX*:

- direct control direction (cooling): Control deviation = actual value - setpoint *rXW* = *rX* - *rW*



- indirect control direction (heating): Control deviation = actual value - setpoint *rXW* = *rW-rX*



VAR_INPUT

```
eDataSecurityType    : E_HVACDataSecurityType;
bSetDefault          : BOOL;
bEnable              : BOOL;
rW                   : REAL;
```

```
rX                  : REAL;
tTaskCycleTime      : TIME;
uiCycleCall         : UINT;
bSetSyncValue       : BOOL;
```

eDataSecurityType: If eDataSecurityType [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 interpreted 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: The control value *rY* is set to the value *rSyncValue* (VAR_IN_OUT) on a rising edge at the input *bSetSyncValue*. 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 [▶ 131])

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 [▶ 491]).

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: The control value *rY* is set to this value on a rising edge at the input *bSync*. 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 [▶ 131] 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 amplification. Only affects the P-component. The variable is saved persistently. Preset to 1.0.

tTi: Integration time [s]. The I-component corrects the residual control deviation following correction of the P-component. 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 actuator. A zero value at this parameter disables the I-component. 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-component. The variable is saved persistently. Preset to 0 s.

tTd: Damping time [s]. The variable is saved persistently. Preset to 0 s.

rYMin / rYMax: Limit 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 setpoint for an exhaust/supply air cascade control. The variables are saved persistently. *rYMin* preset to 0. *rYMax* preset to 100.

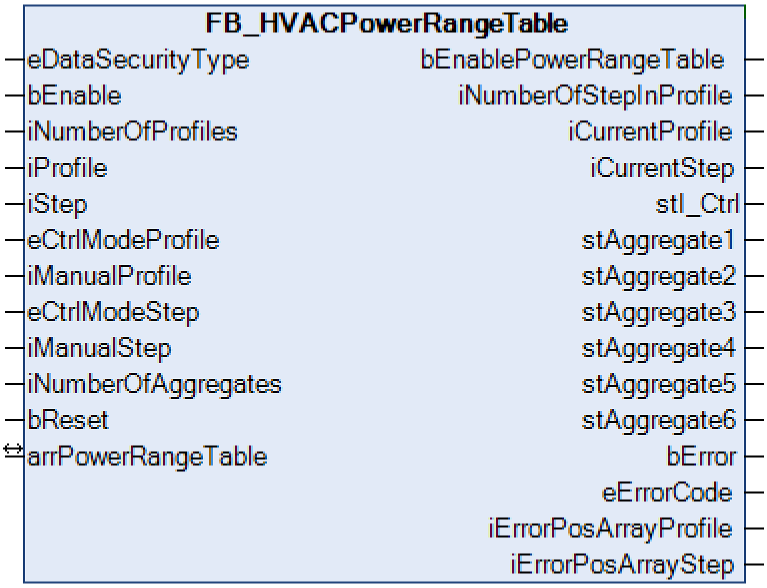
tMinToMax: Slope limitation [s] of the controller output for increase: *tMinToMax* in seconds in reference to a change from *rYMin* to *rYMax*. The variable is saved persistently. Preset to 0 s.

tMaxToMin: Slope limitation [s] of the controller output for decrease: *tMaxToMin* in seconds in reference to a change from *rYMax* to *rYMin.* The variable is saved persistently. Preset to 0 s.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.4.6 FB_HVACPowerRangeTable



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_HVACI_Ctrl_Step [▸ 112] 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 [▸ 498]. 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 to g_iMaxNumberOfProfiles. This range is called the profile and is addressed via the input variable *iProfile*. The field range 0 to g_iMaxNumberOfSteps represents the lines. This range is called the step and is addressed via the input variable *iStep*.

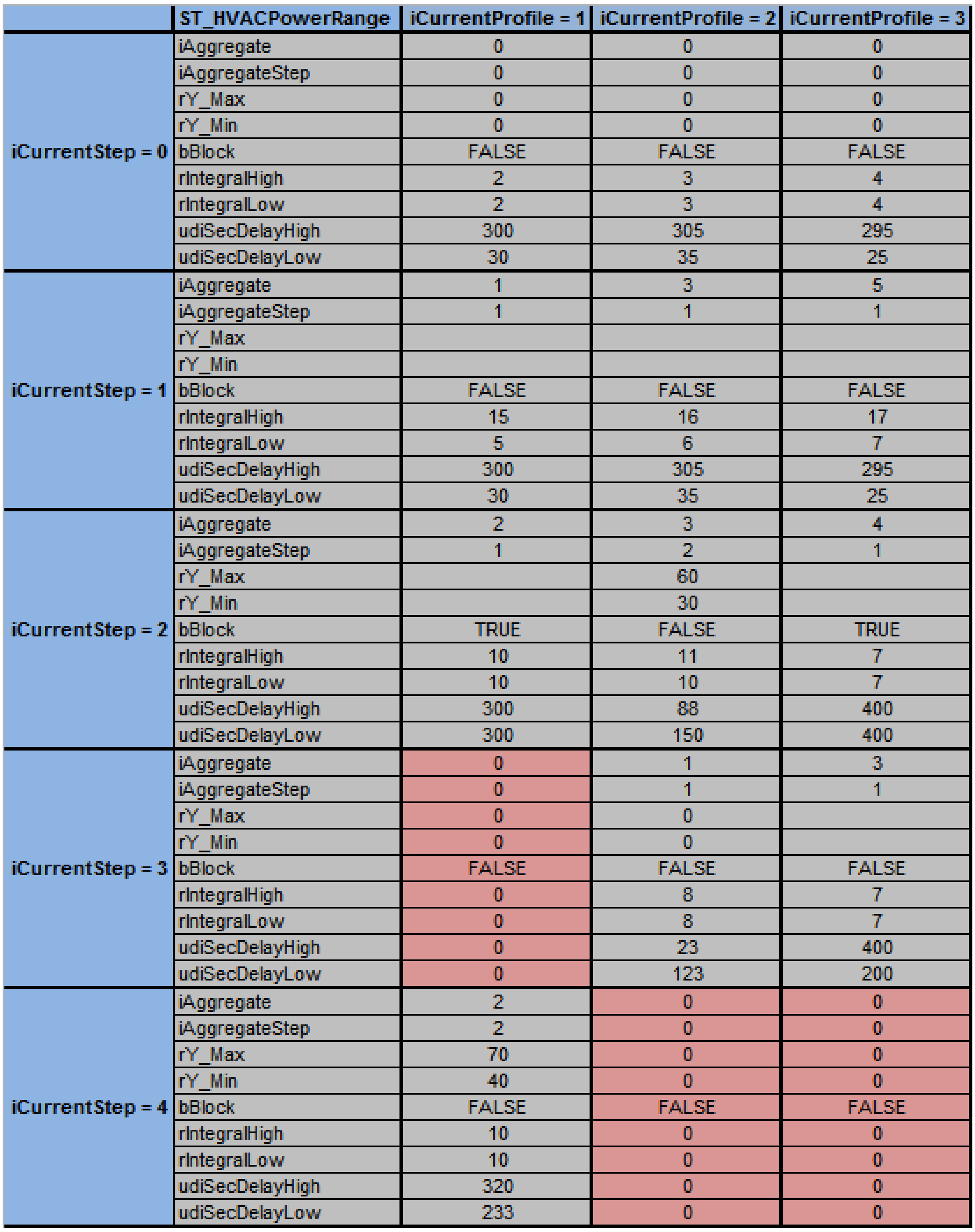
The power range table consists of 16 profiles. Each profile can have up to 33 steps, from 0 to 32. Each individual step contains the parameter structure ST_HVACPowerRange [▸ 498] needed for system control. The parameters indicated here are transferred via the output structures *stI_Ctrl* and *stAggregate1-6* to the function block FB_HVACI_CtrlStep [▸ 112] for controlling the steps and 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 specification in the selected profile *siCurrentProfile*. The selected profile is checked internally from step 1 up to the step in which all variables of the structure ST_HVACPowerRange [▸ 498] have the value 0. *iNumberOfStepInProfile* is always intended only for the specified profile. *iNumberOfStepInProfile* can thus be used to limit the steps in a profile for other function blocks such as for the input *iNumberOfStepInProfile* of FB_HVACI_CtrlStep [▸ 112], 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_HVACI_Ctrl_Step [▸ 112] via the output structure *stI_Ctrl*.

Table example for the power range table arrPowerRangeTable

A unit is assigned to each power step within a profile (table columns).
Hence, the order is specified in which the power generators are switched on or off within a sequence.
The power of the units is specified as 0 to 6 in the case of stepped power generators or 0 to 100% in the case of constant power generators.
After switching to a higher or lower power step, the integral component of the upstream function block FB_HVACI_Ctrl_Step [▶ 112] is re-initialized. The current values for the step-up and step-down integrals are then transferred to the controller by means of the structure *stI_Ctrl*.
It is possible to switch to a different profile for a change of order of the units within the sequence. By skilfully entering different orders of the units 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.
In the case of power generators with different rated outputs, the power profile is changed in relation to the load.

If a unit 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, unit 1 is switched to the first power step in power step 1. In the second power step, unit 2 is added with its first power step.

Profile 2:
In profile 2, unit 3 is switched to the first power step in power step 1. In the second power step, unit 3 is switched in its second power step. In the third power step, unit 1 is added with its first power step.

Profile 3:
In profile 3, unit 5 is switched to the first power step in power step 1. In the second power step, unit 4 is added with its first power step. In the third power step, unit 3 is added with its first power step.

By skillful arrangement of the power profiles, it is possible to react to faults in individual units, 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 *stI_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 *iNumberOfStepInProfile* has the value 2, because no variable value in the structure ST_HVACPowerRange [▶ 498] (marked red) is greater than 0 in the line **iCurrentStep = 3**.
   If one assumes that the current power step *iCurrentStep* has the value 2, then the variable values are output as follows on the output structures:

   ```
   arrPowerRangeTable[iCurrentProfile,iCurrentStep];

   stAggregate1.iAggregateStep :=arrPowerRangeTable[1,1].iAggregateStep;
   stAggregate1.rY_Max := arrPowerRangeTable[1,1].rY_Max;
   stAggregate1.rY_Min :=arrPowerRangeTable[1,1].rY_Min;
   stAggregate1.bBlock := arrPowerRangeTable[1,1].bBlock;

   stAggregate2.iAggregateStep :=arrPowerRangeTable[1,2].iAggregateStep;
   stAggregate2.rY_Max := arrPowerRangeTable[1,2].rY_Max;
   stAggregate2.rY_Min :=arrPowerRangeTable[1,2].rY_Min;
   stAggregate2.bBlock := arrPowerRangeTable[1,2].bBlock;

   stI_Ctrl.rIntegralHigh  := arrPowerRangeTable[1,2].rIntegralHigh;
   stI_Ctrl.rIntegralLow  := arrPowerRangeTable[1,2].rIntegralLow;
   stI_Ctrl.udiSecDelayHigh  :=arrPowerRangeTable[1,2,].udiSecDelayHigh;
   stI_Ctrl.udiSecDelayLow  :=arrPowerRangeTable[1,2].udiSecDelayLow;
   ```

   Contents of the output structures *stAggregate1, stAggregate2* and *stI_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;

   stI_Ctrl.rIntegralHigh := 10;
   stI_Ctrl.rIntegralLow := 10;
   stI_Ctrl.udiSecDelayHigh :=300;
   stI_Ctrl.udiSecDelayLow :=300;
   ```

2. The column **iCurrentProfile = 2** represents the second profile of the power range table *arrPowerRange*. In this profile the output *iNumberOfStepInProfile* has the value 3, because no variable value in the structure ST_HVACPowerRange [▶ 498] (marked red) is greater than 0 in the line **iCurrentStep = 4**.
   If one assumes that the current power step *iCurrentStep* has the value 3, then the variable values are output as follows on the output structures:

   ```
   arrPowerRangeTable[iCurrentProfile,iCurrentStep];

   stAggregate1.iAggregateStep :=arrPowerRangeTable[2,3].iAggregateStep;
   stAggregate1.rY_Max := arrPowerRangeTable[2,3].rY_Max;
   stAggregate1.rY_Min :=arrPowerRangeTable[2,3].rY_Min;
   stAggregate1.bBlock := arrPowerRangeTable[2,3].bBlock;

   stAggregate3.iAggregateStep :=arrPowerRangeTable[2,2].iAggregateStep;
   stAggregate3.rY_Max := arrPowerRangeTable[2,2].rY_Max;
   stAggregate3.rY_Min :=arrPowerRangeTable[2,2].rY_Min;
   stAggregate3.bBlock := arrPowerRangeTable[2,2].bBlock;

   stI_Ctrl.rIntegralHigh  := arrPowerRangeTable[2,3].rIntegralHigh;
   stI_Ctrl.rIntegralLow  := arrPowerRangeTable[2,3].rIntegralLow;
   stI_Ctrl.udiSecDelayHigh  :=arrPowerRangeTable[2,3].udiSecDelayHigh;
   stI_Ctrl.udiSecDelayLow  :=arrPowerRangeTable[2,3].udiSecDelayLow;
   ```

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;
```

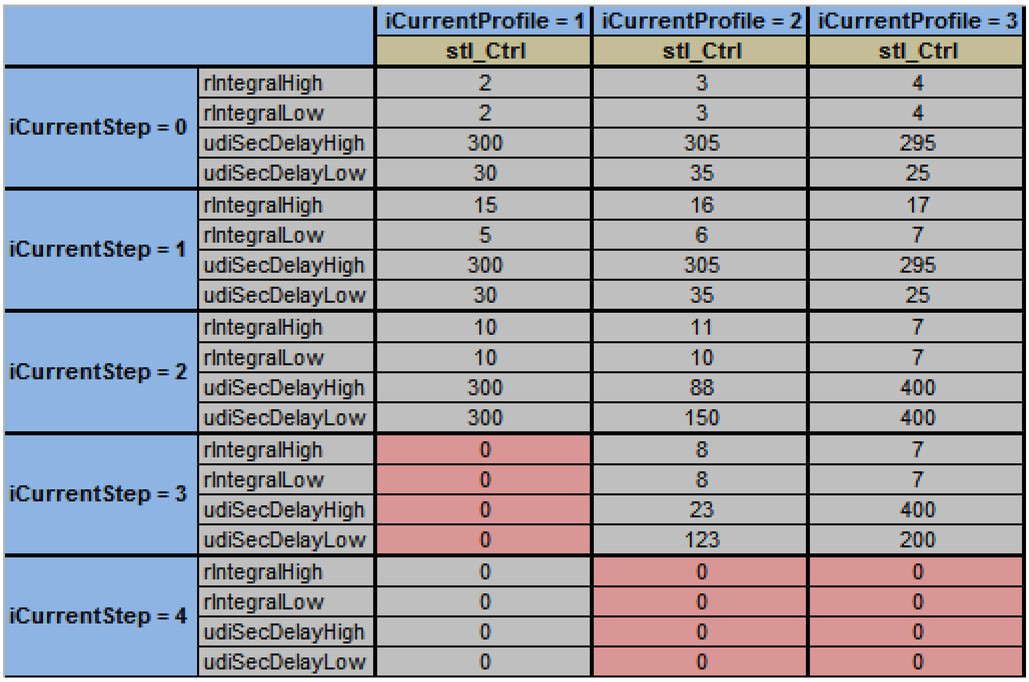
3. The column **iCurrentProfile = 3** represents the third profile of the power range table *arrPowerRange*. In this profile the output *iNumberOfStepInProfile* has the value 3, because no variable value in the structure ST_HVACPowerRange [▸ 498] (marked red) is greater than 0 in the line **iCurrentStep = 4**. If one assumes that the current power step *iCurrentStep* has the value 3, then the variable values are output as follows on the output structures:

```
arrPowerRangeTable[iCurrentProfile,iCurrentStep];

stAggregate3.iAggregateStep :=arrPowerRangeTable[3,3].iAggregateStep;
stAggregate3.rY_Max := arrPowerRangeTable[3,3].rY_Max;
stAggregate3.rY_Min :=arrPowerRangeTable[3,3].rY_Min;
stAggregate3.bBlock := arrPowerRangeTable[3,3].bBlock;

stAggregate4.iAggregateStep :=arrPowerRangeTable[3,2].iAggregateStep;
stAggregate4.rY_Max := arrPowerRangeTable[3,2].rY_Max;
stAggregate4.rY_Min :=arrPowerRangeTable[3,2].rY_Min;
stAggregate4.bBlock := arrPowerRangeTable[3,2].bBlock;


stAggregate5.iAggregateStep :=arrPowerRangeTable[3,1].iAggregateStep;
stAggregate5.rY_Max := arrPowerRangeTable[3,1].rY_Max;
stAggregate5.rY_Min :=arrPowerRangeTable[3,1].rY_Min;
stAggregate5.bBlock := arrPowerRangeTable[3,1].bBlock;

stl_Ctrl.rIntegralHigh  := arrPowerRangeTable[3,3].rIntegralHigh;
stl_Ctrl.rIntegralLow  := arrPowerRangeTable[3,3].rIntegralLow;
stl_Ctrl.udiSecDelayHigh  :=arrPowerRangeTable[3,3].udiSecDelayHigh;
stl_Ctrl.udiSecDelayLow  :=arrPowerRangeTable[3,3].udiSecDelayLow;
```

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.

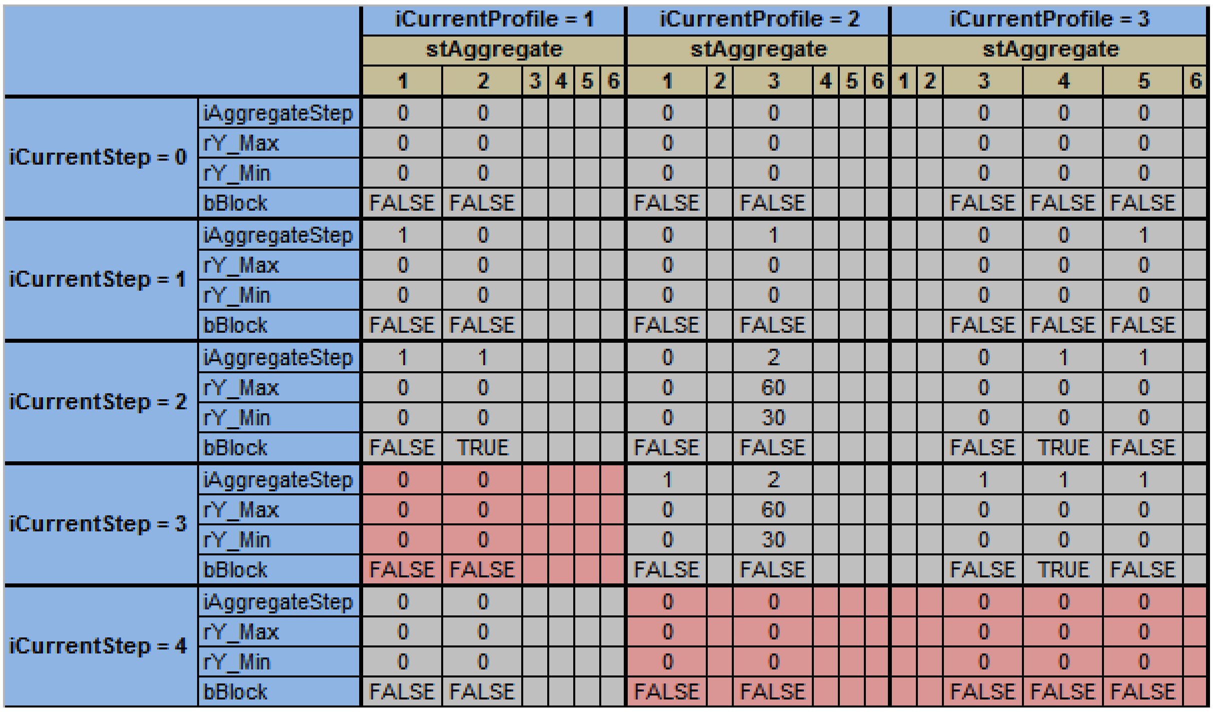
```
stAggregate3.iAggregateStep :=1;
stAggregate3.rY_Max := 0;
stAggregate3.rY_Min :=0;
stAggregate3.bBlock := FALSE;

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.udiSecDelayHigh  :=400;
stl_Ctrl.udiSecDelayLow  :=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	

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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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: The number of parameterized profiles in the power range table *arrPowerRangeTable*. The indication of the number of profiles must not be lower than g_iMinNumberOfProfiles and must not be greater than g_iMaxNumberOfProfiles. 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* 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 smaller than *g_iMinNumberOfSteps* 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 (see E_HVACCtrlMode [▶ 490]). 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 and must not be greater than *iNumberOfProfiles*. Otherwise *iManualProfile* is set internally to g_iMinNumberOfProfiles if it is lower than g_iMinNumberOfProfiles 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 (see E_HVACCtrlMode [▶ 490]). 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 smaller than g_iMinNumberOfSteps and must not be greater than *iNumberOfStepInProfile*. Otherwise *iManualStep* is set internally to g_iMinNumberOfSteps if less than g_iMinNumberOfProfiles or to *iNumberOfStepInProfile* if greater than *iNumberOfStepInProfile*.

iNumberOfAggregates: *iNumberOfAggregates* specifies the number of units 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 specification must not be smaller than g_iMinNumberOfAggregates and must not be greater than g_iMaxNumberOfAggregates. 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 released. *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 [▸ 498] 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 [▸ 498] 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 responsible for starting the power step sequence of the energy generators are transferred to the upstream controller FB_HVACI_Ctrl_Step [▸ 112] 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 [▸ 112]. In turn, the output variable *iStep* of FB_HVACI_CtrlStep [▸ 112] 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*.

stl_Ctrl: Output structure of the parameter ST_HVACI_Ctrl [⏵ 496] for the function block FB_HVACI_CtrlStep [⏵ 112].
The values adopted by the variables of the output structure *stl_Ctrl* from the power range table *arrPowerRangeTable* depend on the selected profile and the current step. If the selected profile *iCurrentProfile* **= 2** and the step *iCurrentStep* **= 3**, then the contents of the following variables from the power range table are written to the structure *stl_Ctrl*:

stl_Ctrl.rIntegralHigh := *arrPowerRangeTable*[**iCurrentProfile**, **iCurrentStep**].*rIntegralHigh*;
stl_Ctrl.rIntegralLow := *arrPowerRangeTable*[**iCurrentProfile**, **iCurrentStep**].*rIntegralLow*;
stl_Ctrl.udiSecDelayHigh :=*arrPowerRangeTable*[**iCurrentProfile**, **iCurrentStep**].*udiSecDelayHigh*;
stl_Ctrl.udiSecDelayLow :=*arrPowerRangeTable*[**iCurrentProfile**, **iCurrentStep**].*udiSecDelayLow*;

stl_Ctrl.rIntegralHigh := *arrPowerRangeTable*[**2**,**3**].*rIntegralHigh*;
stl_Ctrl.rIntegralLow := *arrPowerRangeTable*[**2**,**3**].*rIntegralLow*;
stl_Ctrl.udiSecDelayHigh :=*arrPowerRangeTable*[**2**,**3**].*udiSecDelayHigh*;
stl_Ctrl.udiSecDelayLow :=*arrPowerRangeTable*[**2**,**3**].*udiSecDelayLow*;

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 parameter ST_HVACAggregate [⏵ 495] for controlling units 1 to 6. *iNumberOfAggregates* specifies the number of units 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 *iCurrentProfile* **= 8** and the step *iCurrentStep* **= 5**, then it is crucial that the variable *arrPowerRangeTable*[**8**, **5**].*iAggregate* has the value 1. The values parameterized in the power range table are then output via the output structure *stAggregate1*.

stAggregate1.rY_Max := *arrPowerRangeTable*[**iCurrentProfile**, **iCurrentStep**].*rY_Max*;
stAggregate1.rY_Min :=*arrPowerRangeTable*[**iCurrentProfile**, **iCurrentStep**].*rY_Min*;
stAggregate1.iAggregateStep :=*arrPowerRangeTable*[**iCurrentProfile**, **iCurrentStep**].*iAggregateStep*;
stAggregate1.bBlock := *arrPowerRangeTable*[**iCurrentProfile**, **iCurrentStep**].*bBlock*;

stAggregate1.rY_Max := *arrPowerRangeTable*[**8**,**5**].*rY_Max*;
stAggregate1.rY_Min :=*arrPowerRangeTable*[**8**,**5**].*rY_Min*;
stAggregate1.iAggregateStep :=*arrPowerRangeTable*[**8**,**5**].*iAggregateStep*;
stAggregate1.bBlock := *arrPowerRangeTable*[**8**,**5**].*bBlock*;

If the value of the variable *arrPowerRangeTable* [**8**,**3**].*iAggregate* is also 1, then the values of the highest power step *iCurrentStep* = 5 are written to the corresponding output structure *stAggregate1*, because *arrPowerRangeTable* [**8**,**5**].*iAggregate* 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*[**8**,**3**].*rY_Max*;
stAggregate1.rY_Min :=*arrPowerRangeTable*[**8**,**3**].*rY_Min*;
stAggregate1.iAggregateStep :=*arrPowerRangeTable*[**8**,**3**].*iAggregateStep*;
stAggregate1.bBlock := *arrPowerRangeTable*[**8**,**3**].*bBlock*;

If *iCurrentStep* = 4 and *arrPowerRangeTable* [**8**,**4**].*iAggregate* = 2, then this means that the following values are output via the output structures *stAggregate1* and *stAggregate2*:

stAggregate2.rY_Max := *arrPowerRangeTable*[**8**,**4**].*rY_Max*;
stAggregate2.rY_Min :=*arrPowerRangeTable*[**8**,**4**].*rY_Min*;
stAggregate2.iAggregateStep :=*arrPowerRangeTable*[**8**,**4**].*iAggregateStep*;
stAggregate2.bBlock := *arrPowerRangeTable*[**8**,**4**].*bBlock*;

stAggregate1.rY_Max := *arrPowerRangeTable*[**8**,**3**].*rY_Max*;
stAggregate1.rY_Min :=*arrPowerRangeTable*[**8**,**3**].*rY_Min*;
stAggregate1.iAggregateStep :=*arrPowerRangeTable*[**8**,**3**].*iAggregateStep*;
stAggregate1.bBlock := *arrPowerRangeTable*[**8**,**3**].*bBlock*;

bError: This output indicates with a TRUE that there is an error. The execution of the function block is stopped. The *eErrorCode* enum indicates the error number.
Once the fault has been corrected the *bError* message must be acknowledged with *bReset*.

eErrorCode: Returns the error number if a *bError* output is set (see E_HVACErrorCodes [▶ 491]).

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 or smaller than or equal to *iNumberOfStepInProfile*.

eHVACErrorCodes_InvalidParam_iNumberOfStepInProfile: Error when checking the number of profiles, whether each of the specified profiles has parameterized steps. The profiles are checked in step 0. If none of the variable values of the structure ST_HVACPowerRange [▶ 498] there is larger than 0, then there is an error. The variable *iErrorPosArrayProfile* indicates the incorrect profile in the power range table *arrPowerRangeTable*[**x1**,**x2**]: **x1 =** *iErrorPosArrayProfile;*

eHVACErrorCodes_InvalidParam_iNumberOfAggregates: Error when checking the specified number of units. The value of *iNumberOfAggregates* must be smaller than g_iMinNumberOfAggregates and larger than g_iMaxNumberOfAggregates.

eHVACErrorCodes_InvalidParam_iAggregateSteps: Error when checking the specified steps for the unit. The value of *arrPowerRangeTable*[**x1,x2**].*iAggregateStep* must be greater than or equal to g_iAggregateMinNumberOfSteps and smaller than or equal to g_iAggregateMaxNumberOfSteps.
The variables *iErrorPosArrayProfile* and *iErrorPosArrayStep* indicate the incorrect point in the power range table *arrPowerRangeTable*[**x1**,**x2**]: **x1 =** *iErrorPosArrayProfile;* **x2 =** *iErrorPosArrayStep*

eHVACErrorCodes_InvalidParam_iNumberOfProfiles: Error when checking the number of profiles. The value of *iNumberOfProfiles* must be greater than or equal to g_iMinNumberOfProfiles and smaller than or equal to g_iMaxNumberOfProfiles.

eHVACErrorCodes_InvalidParam_iProfile: Error when checking the specified profiles. The value of *iProfile* must be greater than or equal to *g_iMinNumberOfProfiles* and smaller than or equal to iNumberOfProfiles.

eHVACErrorCodes_InvalidParam_iAggregate: Error when checking the specified units. The value of *arrPowerRangeTable*[**x1,x2**].*iAggregate* must be greater than or equal to g_iMinNumberOfAggregates and smaller than or equal to g_iMaxNumberOfAggregates.
The variables *iErrorPosArrayProfile* and *iErrorPosArrayStep* indicate the incorrect point in the power range table *arrPowerRangeTable*[**x1**,**x2**]: **x1 =** *iErrorPosArrayProfile;* **x2 =** *iErrorPosArrayStep*

eHVACErrorCodes_InvalidParam_udiSecDelayUp: Error when checking *arrPowerRangeTable*[**x1**,**x2**].*udiSecDelayUp*. The value of *arrPowerRangeTable*[**x1,x2**].*udiSecDelayUp* must not be larger than g_udiMaxSec.
The variables *iErrorPosArrayProfile* and *iErrorPosArrayStep* indicate the incorrect point in the power range table *arrPowerRangeTable*[**x1**,**x2**]: **x1 =** *iErrorPosArrayProfile;* **x2 =** *iErrorPosArrayStep*

eHVACErrorCodes_InvalidParam_udiSecDelayDown: Error when checking *arrPowerRangeTable*[**x1**,**x2**].*udiSecDelayDown*. The value of *arrPowerRangeTable*[x1,**x2**].*udiSecDelayDown* must not be larger than g_udiMaxSec.
The variables *iErrorPosArrayProfile* and *iErrorPosArrayStep* indicate the incorrect point in the power range table *arrPowerRangeTable*[**x1**,**x2**]: **x1 =** *iErrorPosArrayProfile;* **x2 =** *iErrorPosArrayStep*

To access the enum error number in the PLC eErrorCode [▶ 491] can assigned to a variable of data type WORD.
eHVACErrorCodes_InvalidParam_iStep = 31,
eHVACErrorCodes_InvalidParam_iNumberOfStepInProfil = 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* indicates the profile in which the incorrect place in the power range table *arrPowerRangeTable*[*iErrorPosArrayProfile, iErrorPosArrayStep*] is to be found. If *iErrorPosArrayProfile* > 0, then one of the following errors is present on *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*.

Example 1 for the localization of an error:
The following values are present on the output variables:

bError =TRUE*;*
eErrorCode= eHVACErrorCodes_InvalidParam_iNumberOfStepInProfile;
iErrorPosArrayProfile = **2**;

No variable value of the structure ST_HVACPowerRange [▶ 498] in profile 2 in step 0 of the power range table *arrPowerRangeTable***[2,x]** is greater than 0. *iErrorPosArrayStep* is not taken into account with this error.

Example 2 for the localization of an error:
The following values are present on 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.

Example for the localization of an error:
The following values are present on 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;
```

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 array with the structure ST_HVACPowerRange [▶ 498]. 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 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 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 [▶ 498] needed for system control. The parameters indicated here are transferred via the output structures *stI_Ctrl* to the function block FB_HVACI_CtrlStep [▶ 112] 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 up to the step in which all variables of the structure ST_HVACPowerRange [▶ 498] have the value 0. *iNumberOfStepInProfile* is always intended only for the specified profile. 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 [▶ 498]

arrPowerRangeTable[x,x].iAggregate: Parameter that specifies to which output structure *stAggregate1-6* of the function block FB_HVACPowerRangeTable [▶ 135] the variables *rY_Min, rY_Max, iAggregateStep* and *bBlock* are written.

arrPowerRangeTable[x,x].iAggregateStep: Parameter that specifies the step in which the addressed unit should be fixed or should regulate; see *bBlock*.
iAggregateStep is output via the structure *stAggregateX.*

arrPowerRangeTable[x,x].rY_Max: Parameter specification for continuous units. *rY_Max* is output via the structure *stAggregateX*.

arrPowerRangeTable[x,x].rY_Min: Parameter specification for continuous units. *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 the FB_HVACI_CtrlStep [▶ 112].
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 the FB_HVACI_CtrlStep [▶ 112].
rIntegralLow is output via the structure *stI_Ctrl.*

arrPowerRangeTable[x,x].udiSecDelayHigh: Delay time after whose expiry the I-transfer element is activated; see the VAR_IN_OUT-Variable *udiSecDelayHigh* in the FB_HVACI_CtrlStep [▶ 112].
udiSecDelayHigh is output via the structure *stI_Ctrl*.

arrPowerRangeTable[x,x].udiSecDelayLow: Delay time after whose expiry the I-transfer element is activated; see the VAR_IN_OUT-Variable *udiSecDelayHigh* in the FB_HVACI_CtrlStep [▶ 112].
udiSecDelayLow is output via the structure *stI_Ctrl*.

arrPowerRangeTable[x,x].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.
bBlock is output via the structure *stAggregateX*.
The variable is saved persistently.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.4.7 Sequence-Controller

Introduction – sequence controller

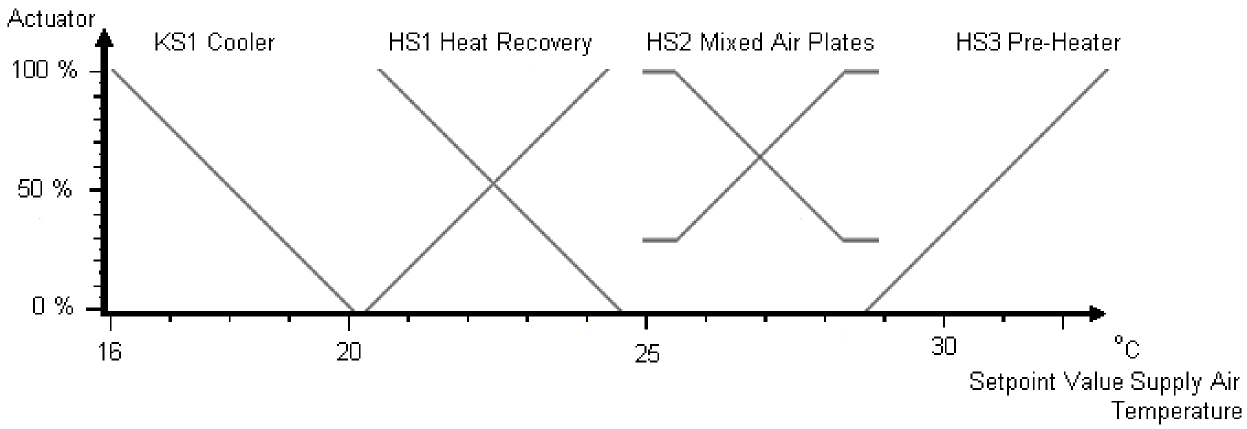
In heating, ventilation and air conditioning systems, it is often the case that several actuators, working in a so-called controller sequence, are used in order to achieve a control variable. In the air conditioning system shown below, four actuators are involved in the regulation of the supply air temperature. 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 an 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 at the end position of *rYmax* or *rYmin*, depending on

the control direction.
This is repeated with the remaining actuators until the setpoint or the left or right 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.

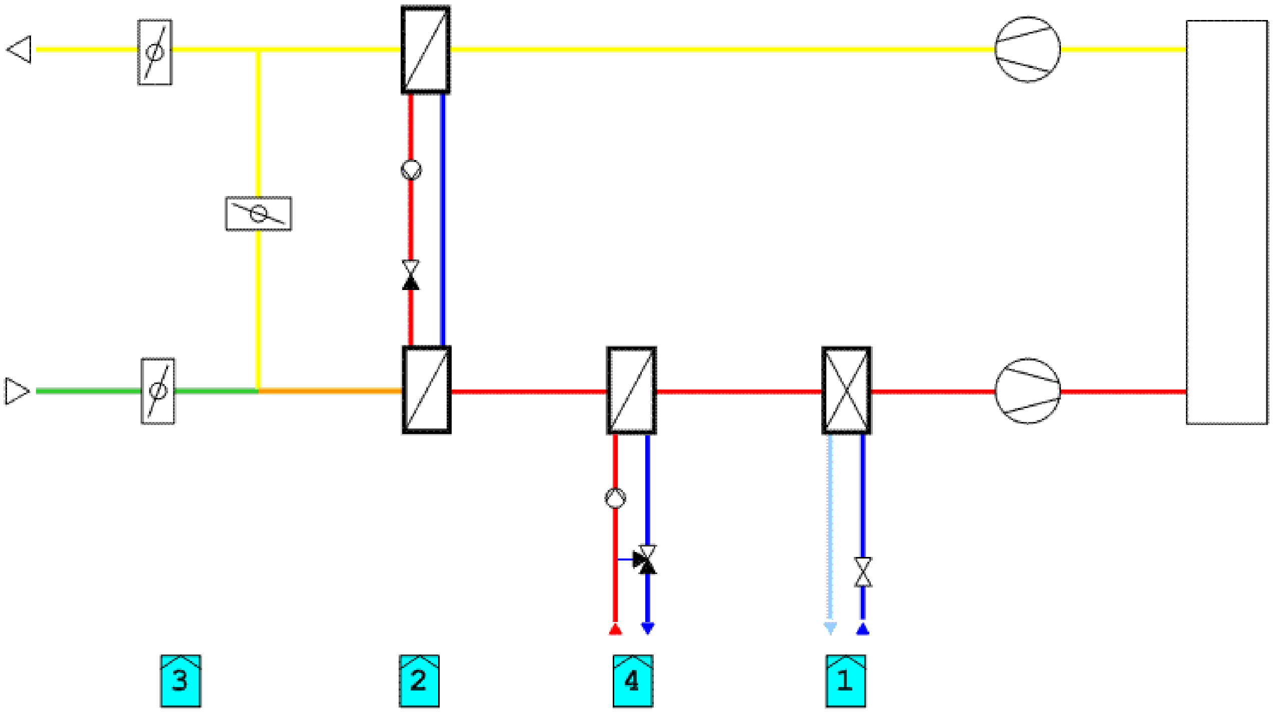
Actuators that should drive in parallel or in opposite directions, e.g. outside air and circulating air dampers (HS2) are implemented once only. The individual actuators can have a positive control direction (more actuation results in a larger control variable, e.g. a heater) or a negative control direction (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 diagram above.

```
iMyNumberInSequence
```

1. KS1 cooler 1
2. HS1 heat recovery 2
3. HS2 mixed air dampers 3
4. HS3 pre-heater 4



Each sequence controller is given the total number of controllers in the sequence.
Four in this case. 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 [▸ 169] would start control 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 [▸ 176]. The parameter *eHVACSequenceCtrlMode* is thus raised 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 pre-heater or the sequence controller FB_HVACPIDPreHeating [▸ 179] respectively is activated. If the control variable is too high, the sequence is reversed until the cooler is activated by means of FB_HVACPIDCooling [▸ 163] being enabled.

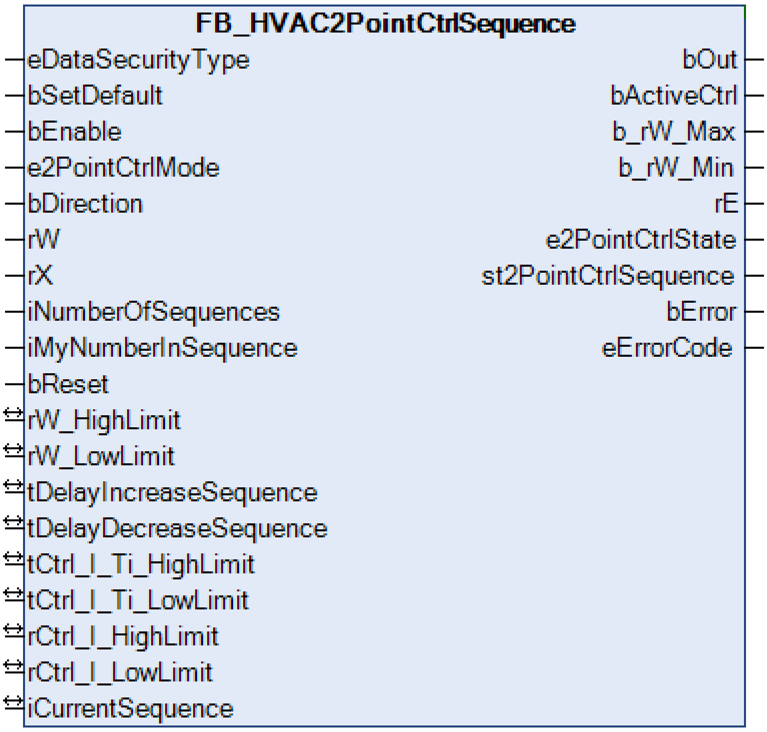
If the control value is at the lower or upper limit of a controller, the actual value of the controlled system oscillates around the setpoint with a small amplitude, frequent switching back and forth between two sequence controllers can be damped by an additional parameter for the switchover. To this end, the difference between the actual and 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

A further special feature of the sequence controller is its control by the Enum E_HVACSequenceCtrlMode [▸ 493]
Not only is the controller enabled by the Enum E_HVACSequenceCtrlMode [▸ 493], it also transmits the operation mode of the air conditioning system to the controller blocks in the sequence. Depending on the operation modes, each sequence controller hence reacts specially to the Enum E_HVACSequenceCtrlMode [▸ 493] 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 %

5.1.4.7.1 FB_HVAC2PointCtrlSequence



This function block represents a 2-point sequence controller. It is used for sequential staged control of discontinuous units. If 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

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

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

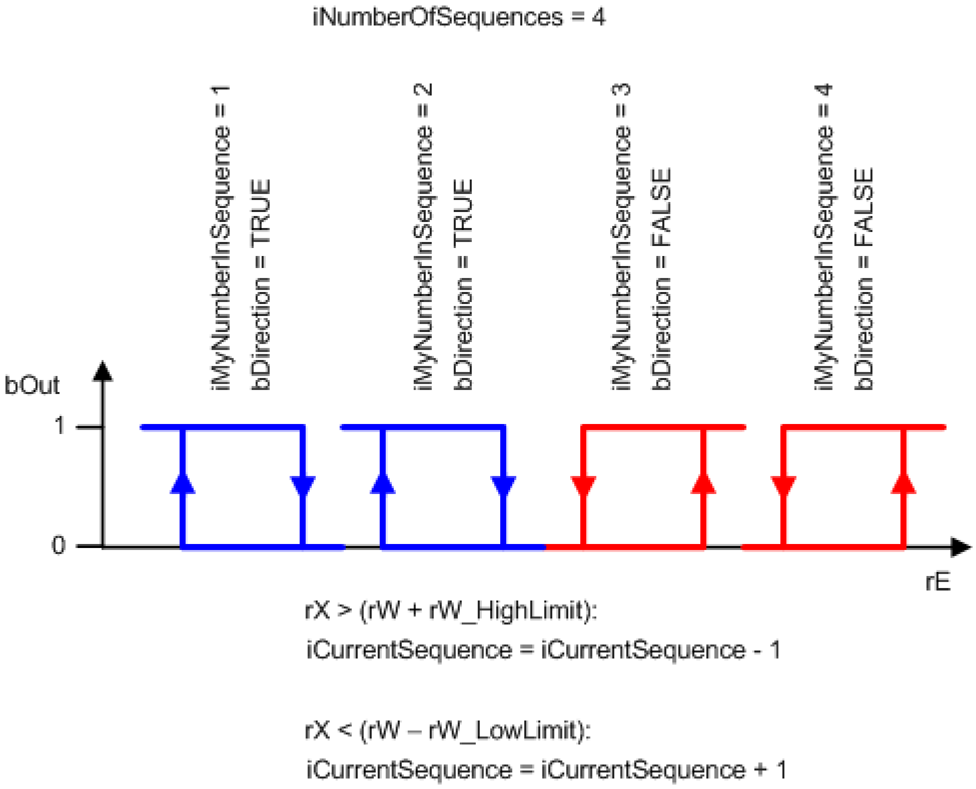
```
bActiveCtrl = TRUE AND
( (*Decrease*)
(
rX > st2PointCtrlSequence.rW_Max AND ((iMyNumberInSequence <= 1) = FALSE) AND
(
(bDirection = TRUE AND bOut = TRUE) OR
(bDirection = FALSE AND bOut = FALSE)
)
)
OR
( (*Increase*)
rX < st2PointCtrlSequence.rW_Min AND((iMyNumberInSequence >= iNumberOfSequences) = 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 when *st2PointCtrlSequence.rCtrl_I_Out* <= *st2PointCtrlSequence.rCtrl_I_LowLimit* AND *st2PointCtrlSequence.tRemainingTimeDecreaseSequence* = T#0s.

iCurrentSequence = *iCurrentSequence* + 1 when *st2PointCtrlSequence.rCtrl_I_Out* >= *st2PointCtrlSequence.rCtrl_I_HighLimit* AND *st2PointCtrlSequence.tRemainingTimeIncreaseSequence* = T#0s.

Behavior of the outputs of four FB_HVAC2PointCtrlSequence in a control sequence



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.

If *bEnable* =FALSE OR *bError* = TRUE OR (*e2PointCtrlMode* =eHVAC2PointCtrlMode_On_BMS OR eHVAC2PointCtrlMode_On_OP OR eHVAC2PointCtrlMode_Off_BMS OR eHVAC2PointCtrlMode_Off_OP), then the automatic switchover of the controller from the active controller to the neighboring controller on the left or right in the control sequence remains in operation. 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.

VAR_INPUT

```
eDataSecurityType   : E_HVACDataSecurityType;
bSetDefault         : BOOL;
bEnable             : BOOL;
e2PointCtrlMode     : E_HVAC2PointCtrlMode;
bDirection          : BOOL;
rW                  : REAL;
rX                  : REAL;
iNumberOfSequences  : INT;
iMyNumberInSequence: INT;
bReset              : BOOL;
```

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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.

If *bEnable* =FALSE OR *bError* = TRUE OR (*e2PointCtrlMode* =eHVAC2PointCtrlMode_On_BMS OR eHVAC2PointCtrlMode_On_OP OR eHVAC2PointCtrlMode_Off_BMS OR eHVAC2PointCtrlMode_Off_OP), then the automatic switchover of the controller from the active controller to the neighboring controller on the left or right in the control sequence remains in operation. 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 (see E_HVAC2PointCtrlMode [▶ 488]). If *bEnable* = TRUE AND *bError* = FALSE, then the output *bOut* can be switched on or off directly via the enum. On starting the PLC *e2PointCtrlMode* = *eHVAC2PointCtrlMode_Auto_BMS*.
Up- or down-switching of the sequence via *iCurrentSequence* does not depend on mode *e2PointCtrlMode* of the 2-point sequence controller.

If *bEnable* =FALSE OR *bError* = TRUE OR (*e2PointCtrlMode* =eHVAC2PointCtrlMode_On_BMS OR eHVAC2PointCtrlMode_On_OP OR eHVAC2PointCtrlMode_Off_BMS OR eHVAC2PointCtrlMode_Off_OP), then the automatic switchover of the controller from the active controller to the neighboring controller on the left or right in the control sequence remains in operation. 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 *rW* variable.

rX: The variable *rX* is used to transfer the actual value.

iNumberOfSequences: Number of controllers in the sequence (1..32). If *iNumberOfSequences* <= 0 an error is issued and indicated with *bError* = TRUE. The number of device in a sequence is not exceeded by the active sequence controller via *iCurrentSequence*.

iMyNumberInSequence: The 2-point controller's own number in the sequence (1..32). If *iMyNumberInSequence* > *iNumberOfSequences* OR *iMyNumberInSequence* <= 0, then an error is output and indicated by *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_BMS* OR *e2PointCtrlMode* = *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_BMS* OR *e2PointCtrlMode* = *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_BMS* OR *e2PointCtrlMode* = *eHVAC2PointCtrlMode_On_OP*)

bOut becomes FALSE if

1.
bEnable = FALSE OR *bError* = TRUE

2.
bEnable = TRUE AND *bError* = FALSE AND
(*e2PointCtrlMode* = *eHVAC2PointCtrlMode_Auto_BMS* OR *e2PointCtrlMode* = *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_BMS* OR *e2PointCtrlMode* = *eHVAC2PointCtrlMode_Auto_OP*) AND
(
(*bDirection* = TRUE AND (*iCurrentSequence* > *iMyNumberInSequence*)) OR (*bDirection* = FALSE AND (*iCurrentSequence* < *iMyNumberInSequence*))
)

4.
bEnable = TRUE AND *bError* = FALSE AND
(*e2PointCtrlMode* = *eHVAC2PointCtrlMode_Off_BMS* OR *e2PointCtrlMode* = *eHVAC2PointCtrlMode_Off_OP*)

bActiveCtrl: *bActiveCtrl* indicates with TRUE that the function block is the active one in the sequence. *bActiveCtrl* goes TRUE if *bEnable* = TRUE, *bError* = FALSE AND *iCurrentSequence* = *iMyNumberInSequence*.

b_rW_Max: *b_rW_Max* goes TRUE if *rX* > *st2PointCtrlSequence.rW_Max*.

b_rW_Min: *b_rW_Min* goes TRUE if *rX* < *st2PointCtrlSequence.rW_Min.*

rE: Control deviation: *rE* = *rW* - *rX*

e2PointCtrlState: Enum that indicates the state of the operation mode *e2PointCtrlMode* of the 2-point sequence controller (see E_HVAC2PointCtrlMode [▸ 488]).

st2PointCtrlSequence: The structure indicates various states, inputs and outputs of the function block (see ST_HVAC2PointCtrlSequence [▸ 495]). 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 time of delay time *tDelayIncreaseSequence*.

st2PointCtrlSequence.tRemainingTimeDecreaseSequence: Remaining time of 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 [▸ 150] 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 [▸ 150] in this document

st2PointCtrlSequence.rE: Control deviation: *rE* = *rW* - *rX*

st2PointCtrlSequence.rCtrl_I_HighLimit: Upper limit value 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 value 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 *iNumberOfSequences*

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* is *TRUE*, output bOut becomes FALSE.

If *bEnable* =FALSE OR *bError* = TRUE OR (*e2PointCtrlMode* =eHVAC2PointCtrlMode_On_BMS OR eHVAC2PointCtrlMode_On_OP OR eHVAC2PointCtrlMode_Off_BMS OR eHVAC2PointCtrlMode_Off_OP), then the automatic switchover of the controller from the active controller to the neighboring controller on the left or right in the control sequence remains in operation. 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.

eErrorCode: returns the error number if an output *bError* is set (E_HVACErrorCodes [▸ 491] E_HVACErrorCodes). The following errors can occur in this function block: *eHVACErrorCodes_Error_iMyNumberInSequence*, *eHVACErrorCodes_Error_iNumberOfSequences*, *eHVACErrorCodes_Error_iCurrentSequences*

To access the enum error number in the PLC eErrorCode [▸ 491] can 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 control deviation limit. *st2PointCtrlSequence.rW_Max =rW_Max* := *rW* + *rW_HighLimit.*
The variable is saved persistently. Preset to 5.

rW_LowLimit: Positive value of the lower control deviation limit. *st2PointCtrlSequence.rW_Min =rW_Min* := *rW* - *rW_LowLimit.*
The variable is saved persistently. Preset to 5.

tDelayIncreaseSequence: Delay time after whose expiry *iCurrentSequence* is incremented by 1, see Transfer function of the internal I-transfer element [▸ 150] 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 [▸ 150] 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 [▸ 150] 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 [▸ 150] 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 [▸ 150] 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 [▶ 150] 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 over *iCurrentSequence*. If *iCurrentSequence* > *iNumberOfSequences* or *iCurrentSequence* < 0 an error is indicated with *bError* = TRUE.
Up- or down-switching of the sequence via *iCurrentSequence* depending on the control deviation *rE* takes place if the function block is active in the sequence *bActiveCtrl* = TRUE.

1. *iCurrentSequence* = *iCurrentSequence - 1 if st2PointCtrlSequence.rCtrl_I_Out* >= *st2PointCtrlSequence.rLimit_Ctrl_I_Min* AND *st2PointCtrlSequence.tRemainingTimeDecreaseSequence* = T#0s AND *bActiveCtrl* = TRUE

2. *iCurrentSequence* = *iCurrentSequence* + 1 if *st2PointCtrlSequence.rCtrl_I_Out* >= *st2PointCtrlSequence.rLimit_Ctrl_I_Max* AND *st2PointCtrlSequence.tRemainingTimeIncreaseSequence* = T#0s AND *bActiveCtrl* = TRUE

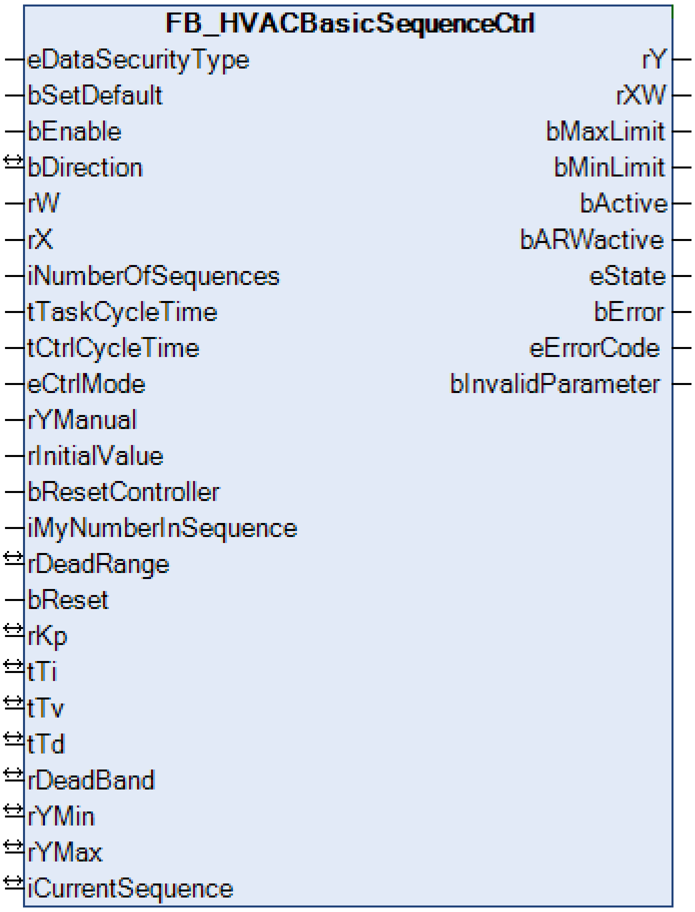
To ensure correct functioning of the automatic switchover of the controller from the active controller to the neighboring controller on the left or right in the control sequence, *iCurrentSequence* may not be written to continuously from the outside in a control sequence. The active sequence controller must be defined when starting a control sequence. *iCurrentSequence* must be written for one PLC cycle and must be > 0 and <= *iNumberOfSequences*.

If *bEnable* =FALSE OR *bError* = TRUE OR (*e2PointCtrlMode* =eHVAC2PointCtrlMode_On_BMS OR eHVAC2PointCtrlMode_On_OP OR eHVAC2PointCtrlMode_Off_BMS OR eHVAC2PointCtrlMode_Off_OP), then the automatic switchover of the controller from the active controller to the neighboring controller on the left or right in the control sequence remains in operation. 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.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.4.7.2 FB_HVACBasicSequenceCtrl



Unlike the sequence controllers FB_HVACPIDCooling [▶ 163], FB_HVACDehumidify [▶ 166], FB_HVACEnergyRecovery [▶ 169], FB_HVACPIDHumidify [▶ 172], FB_HVACPIDMixedAir [▶ 176], FB_HVACPIDPreHeating [▶ 179] and FB_HVACPIDReHeating [▶ 182], the function block *FB_HVACBasicSequenceCtrl* has no special system-specific extension 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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 enabled if *bEnable* = TRUE.

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

eCtrlMode: The operation mode is selected via this enum (see E_HVACCtrlMode [▸ 490]).

rYManual: Manual value.

rInitialValue: The restart behavior of the controller is influenced by *rInitialValue*.

bResetContoller: 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 variable limit.

eState: State of the controller (see E_HVACState) [▶ 494].

bError: This output indicates with a TRUE that there is an error.

eErrorCode: Contains the command-specific error code (see E_HVACErrorCodes [▶ 491]).

bInvalidParameter: TRUE if an error has occurred 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, 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.

rKp: Proportional factor amplification. The variable is saved persistently. Preset to 1.

tTi: Integration time. The I-component corrects the residual control deviation following correction of the P-component. 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 actuator. 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 avoided by an additional parameter for the switchover (0..32767). To this end, the difference between the actual and 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 Example [▶ 507]).
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.

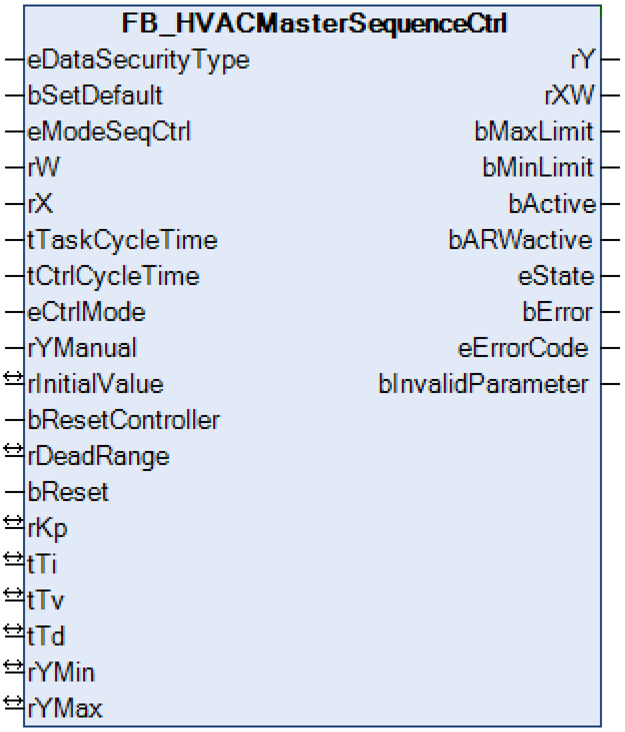
rYMax: Upper limit of the working range of the controller. The variable is saved persistently. Preset to 100.

iCurrentSequence: Currently active controller in the sequence.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.4.7.3 FB_HVACMasterSequenceCtrl



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 setpoints 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 the backup mode of the air conditioning system *(eModeSeqCtrl = eHVACSequenceCtrlMode_FreezeProtection)*, the value of *rYMax* is switched directly through to the output *rY.* If overheating protection is active *(eModeSeqCtrl = eHVACSequenceCtrlMode_ OverheatingProtection)*, the value of *rYMin* is 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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▸ 491]:= 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 system operation mode is transferred to the controller with the enum *eModeSeqCtrl* (see E_HVACSequenceCtrlMode [▸ 493]).

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 variable to calculate internally whether the state and the output variables have to be updated in the current cycle.

eCtrlMode: The operation mode is selected via this Enum. Manual or automatic mode (see E_HVACCtrlMode [▸ 490]).

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: Set temperature value 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 variable limit.

eState: State of the controller (see E_HVACState [▸ 494]).

bError: This output indicates with a TRUE that there is an error.

eErrorCode: Contains the command-specific error code (see E_HVACErrorCodes [▸ 491]).

bInvalidParameter: TRUE if an error has occurred 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 amplification. The variable is saved persistently. Preset to 1.

tTi: Integration time. The I-component corrects the residual control deviation following correction of the P-component. 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.

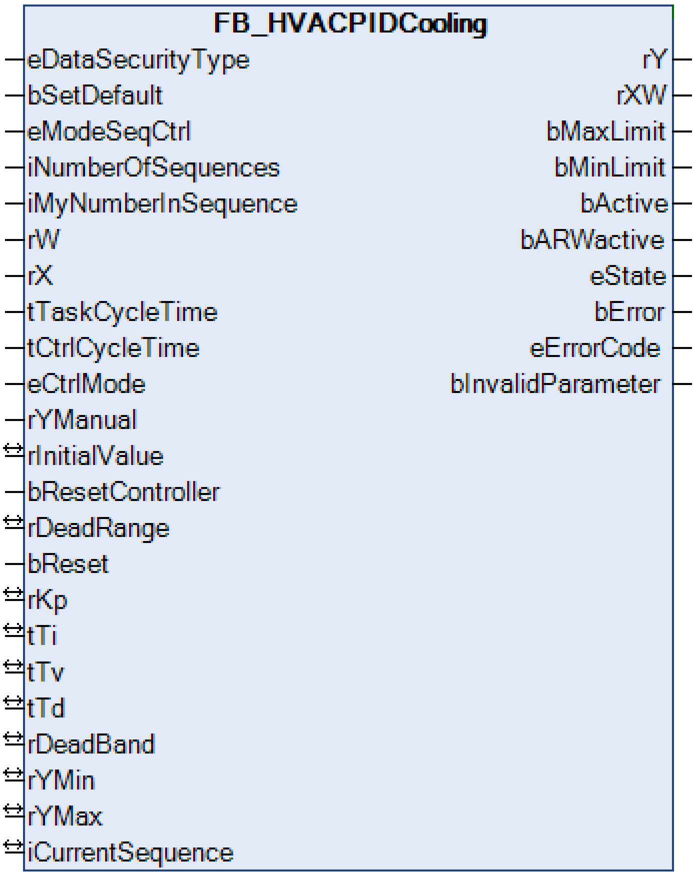
rYMin: Lower limit of the working range of the controller. The variable is saved persistently. Preset to 16.

rYMax: Upper limit of the working range of the controller. The variable is saved persistently. Preset to 25.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.4.7.4 FB_HVACPIDCooling



The control direction of the cooling controller is inverted compared to the heating controller. If the supply air setpoint 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 [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▶ 491]:= 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 enumeration (see E_HVACSequenceCtrlMode [▶ 493]). In addition, the operation mode of the VAC system is transmitted to the controller blocks in the sequence. See also Table operation modes [▶ 509].

iNumberOfSequences: Number of sequence controllers in the system.

iMyNumberInSequence: The controller's own number 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.

eCtrlMode: The operation mode is selected via this enumeration (see E_HVACCtrlMode [▶ 490]). 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 variable limit.

eState: State of the controller (see E_HVACState [▶ 494]).

bError: This output indicates with a TRUE that there is an error.

eErrorCode: E_HVACErrorCodes (see E_HVACErrorCodes [▶ 491]).

bInvalidParameter: TRUE if an error has occurred 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 amplification. The variable is saved persistently. Preset to 0.

tTi: Integration time. The I-component corrects the residual control deviation following correction of the P-component. 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 actuator. 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 set value 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 and set values 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 Example [▶ 507]). 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.

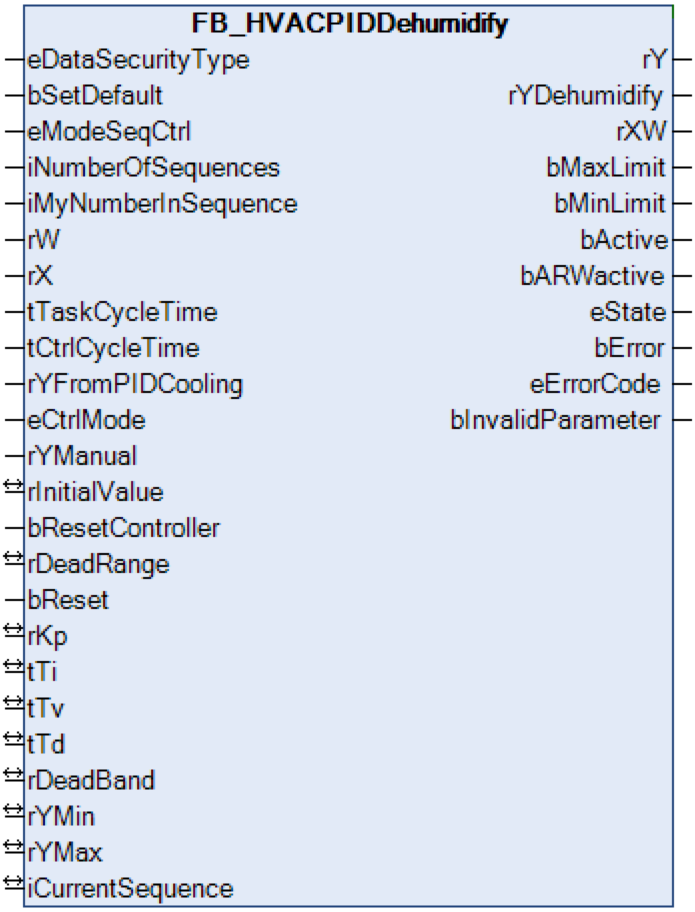
rYMax: Upper limit of the working range of the controller. The variable is saved persistently. Preset to 100.

iCurrentSequence: Number of the active controller from the sequence.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.4.7.5 FB_HVACPIDDehumidify



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 [▶ 163] and *FB_HVACPIDDehumidify*, act on the cooler valve. First of all the control signal *rY* is relayed by FB_HVACPIDCooling [▶ 163] 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 pre-heater 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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 (see E_HVACSequenceCtrlMode [▸ 493]). In addition, the operation mode of the VAC system is transmitted to the controller function block in the sequence. See also Table operation modes [▸ 509].

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 (see E_HVACCtrlMode [▸ 490]). 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 variable limit.

eState: State of the controller (see E_HVACState [▸ 494]).

bError: This output indicates with a TRUE that there is an error.

eErrorCode: E_HVACErrorCodes (see E_HVACErrorCodes [▸ 491]).

bInvalidParameter: TRUE if an error has occurred 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 amplification. The variable is saved persistently. Preset to 0.

tTi: Integration time. The I-component corrects the residual control deviation following correction of the P-component. 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 actuator. 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 set value 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 and set values 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 Example [▸ 507]). 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.

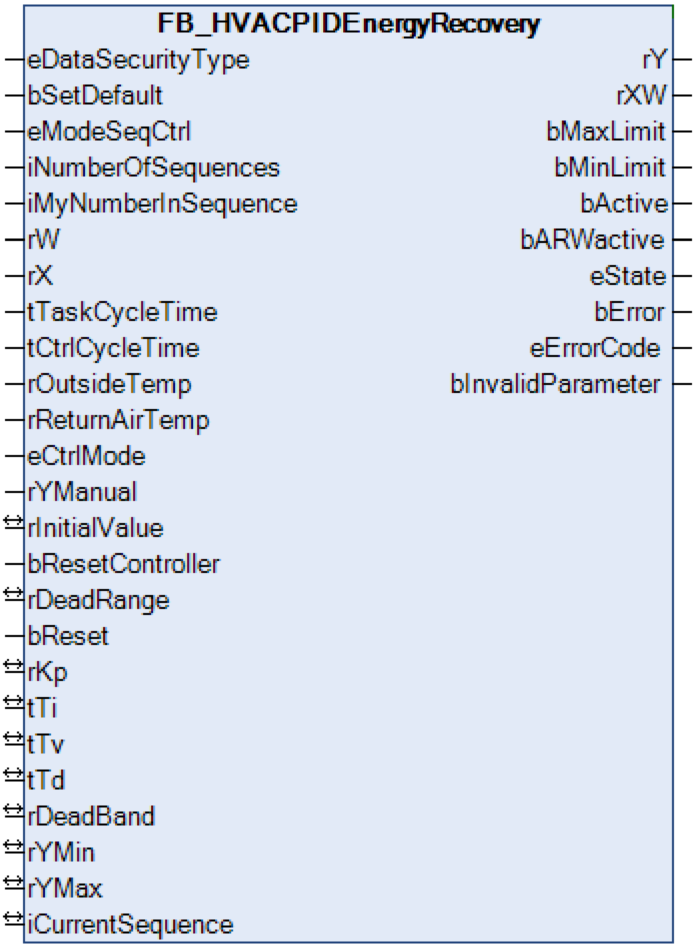
rYMax: Upper limit of the working range of the controller. The variable is saved persistently. Preset to 100.

iCurrentSequence: Number of the active controller from the sequence.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.4.7.6 FB_HVACPIDEnergyRecovery



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 [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 (see E_HVACSequenceCtrlMode [▸ 493]). In addition, the operation mode of the VAC system is transmitted to the controller function block in the sequence. See also Table operation modes [▸ 509].

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 mode (see E_HVACCtrlMode [▸ 490]).

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 variable limit.

eState: State of the controller (see E_HVACState [▸ 494]).

bError: This output indicates with a TRUE that there is an error.

eErrorCode: E_HVACErrorCodes (see E_HVACErrorCodes [▸ 491]).

bInvalidParameter: TRUE if an error has occurred 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 amplification. The variable is saved persistently. Preset to 0.

tTi: Integration time. The I-component corrects the residual control deviation following correction of the P-component. 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 actuator. 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 set value 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 and set values 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 Example [▸ 507]). 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.

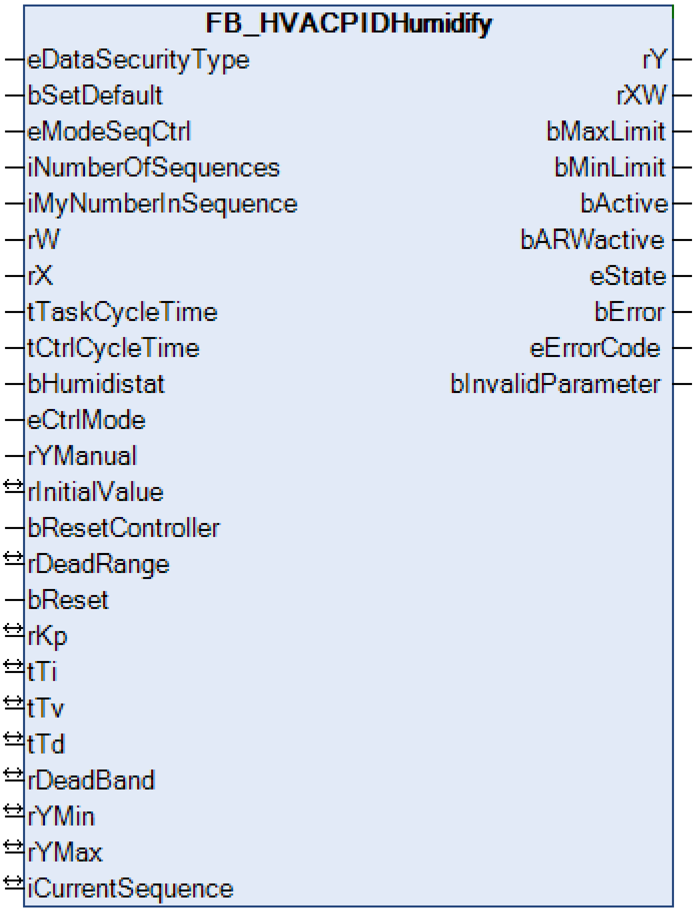
rYMax: Upper limit of the working range of the controller. The variable is saved persistently. Preset to 100.

iCurrentSequence: Number of the active controller from the sequence.

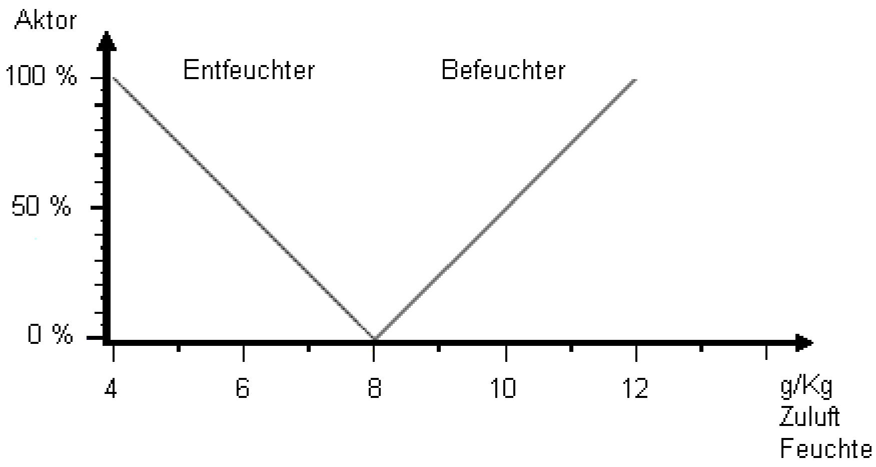
Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.4.7.7 FB_HVACPIDHumidify



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 incuding 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;
bResetContoller         : BOOL;
bReset                  : BOOL;
```

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 (see E_HVACSequenceCtrlMode [▸ 493]). In addition, the operation mode of the VAC system is transmitted to the controller blocks in the sequence. See also Table operation modes [▸ 509].

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.

bHumidistat: A TRUE on this input enables the humidity control. The hygrostat is connected to this input so that over-humidification cannot occur.

eCtrlMode: The operation mode is selected via this Enum. Manual or automatic mode (see E_HVACCtrlMode [▶ 490]).

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 variable limit.

eState: State of the controller (see E_HVACState [▶ 494]).

bError: This output indicates with a TRUE that there is an error.

eErrorCode: E_HVACErrorCodes (see E_HVACErrorCodes [▶ 491]).

bInvalidParameter: TRUE if an error has occurred 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 amplification. The variable is saved persistently. Preset to 0.

tTi: Integration time. The I-component corrects the residual control deviation following correction of the P-component. 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 actuator. 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 set value 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 and set values 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 Example [▶ 507]). 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.

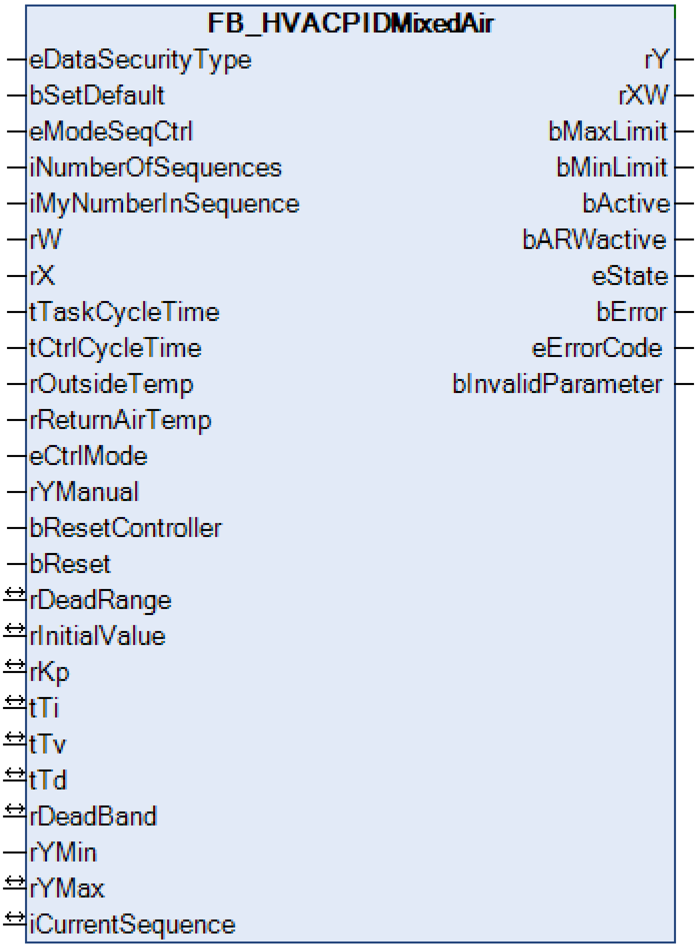
rYMax: Upper limit of the working range of the controller. The variable is saved persistently. Preset to 100.

iCurrentSequence: Number of the active controller from the sequence.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.4.7.8 FB_HVACPIDMixedAir



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 setpoint 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 setpoint 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 operation mode* (eHVACSequenceCtrlMode_NightCooling) is transmitted to the function block by means of the Enum eModeSeqCtrl [▸ 493]. 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 [▸ 451].

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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 (see E_HVACSequenceCtrlMode [▶ 493]). In addition, the operation mode of the VAC system is transmitted to the controller blocks in the sequence. See also Table operation modes [▶ 509].

iNumberOfSequences: Number of sequence controllers in the system.

iMyNumberInSequence: The controller's own number 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 (see E_HVACCtrlMode [▶ 490]). 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 variable limit.

eState: State of the controller (see E_HVACState [▶ 494]).

bError: This output indicates with a TRUE that there is an error.

eErrorCode: E_HVACErrorCodes (see E_HVACErrorCodes [▶ 491]).

bInvalidParameter: TRUE if an error has occurred 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 amplification. The variable is saved persistently. Preset to 0.

tTi: Integration time. The I-component corrects the residual control deviation following correction of the P-component. 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 actuator. 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 and 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 Example [▶ 507]). The variable is saved persistently. Preset to 0.

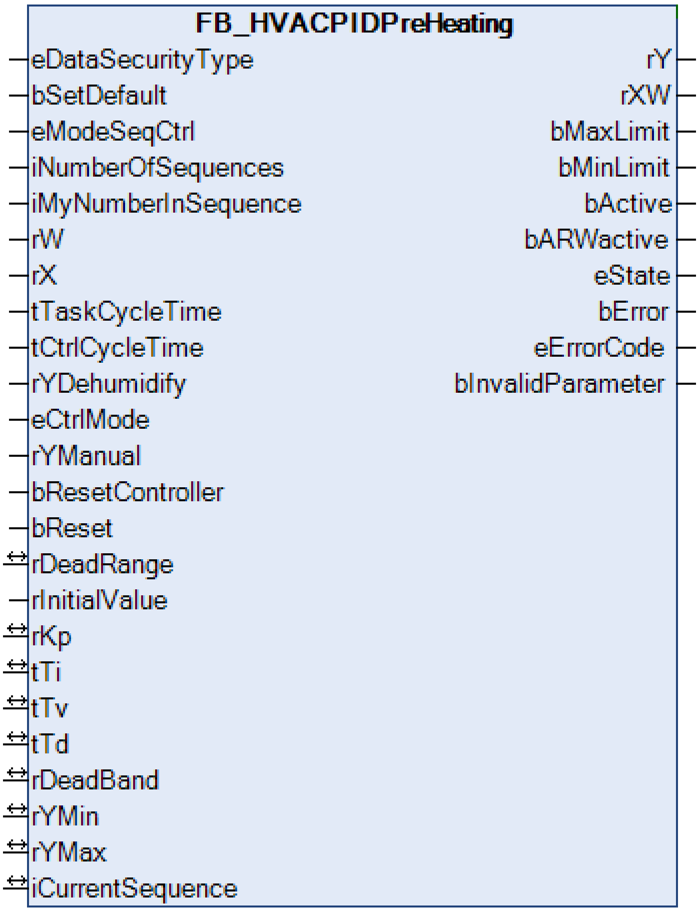
rYMax: Upper limit of the working range of the controller. The variable is saved persistently. Preset to 100.

iCurrentSequence: Number of the active controller from the sequence.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.4.7.9 FB_HVACPIDPreHeating



This function block is a special application of a PID controller.
Besides the PID control algorithm, the block contains the conditions for changing to the sequence of the previous or next sequence controller. A special feature of the pre-heating 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. As a result of this the reheater FB_HVACPIDReHeating [▶ 182] is activated in place 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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 (see E_HVACSequenceCtrlMode [▶ 493]). In addition, the operation mode of the VAC system is transmitted to the controller function block in the sequence. See also Table operation modes [▶ 509].

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 t*CtrlCycleTime* 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 [▶ 166] 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 (see E_HVACCtrlMode [▶ 490]). 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 variable limit.

eState: State of the controller (see E_HVACState [▶ 494]).

bError: This output indicates with a TRUE that there is an error.

eErrorCode: E_HVACErrorCodes (see E_HVACErrorCodes [▶ 491]).

bInvalidParameter: TRUE if an error has occurred 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 amplification. The variable is saved persistently. Preset to 0.

tTi: Integration time. The I-component corrects the residual control deviation following correction of the P-component. 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 actuator. 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 and 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 Example [▶ 507]). 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.

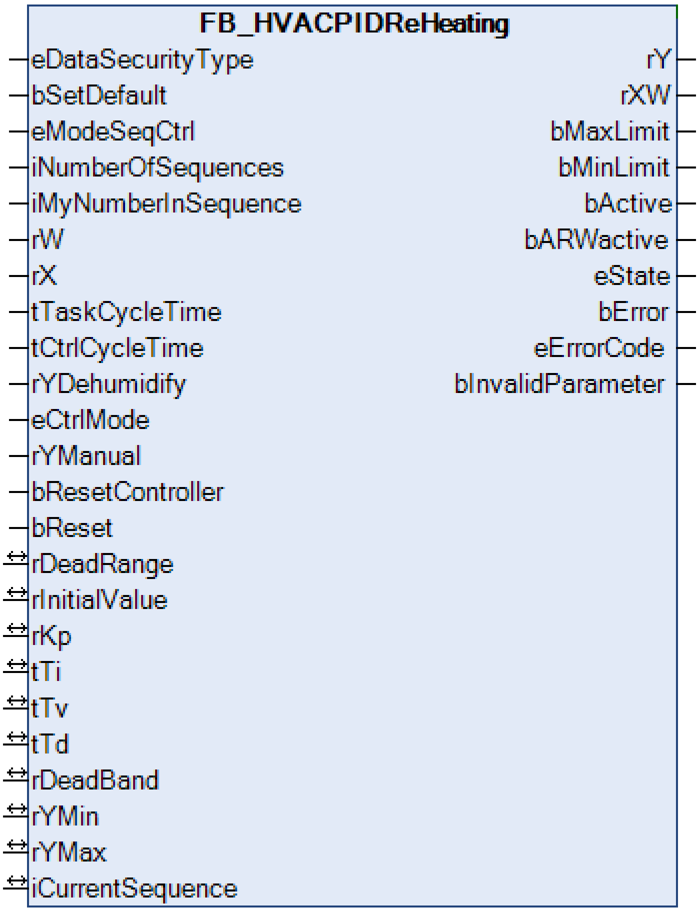
rYMax: Upper limit of the working range of the controller. The variable is saved persistently. Preset to 100.

iCurrentSequence: Number of the active controller from the sequence.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.4.7.10 FB_HVACPIDReHeating



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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 (see E_HVACSequenceCtrlMode [▶ 493]). In addition, the operation mode of the VAC system is transmitted to the controller blocks in the sequence. See also Table operation modes. [▶ 509]

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 (see E_HVACCtrlMode [▶ 490]). 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 variable limit.

eState: State of the controller (see E_HVACState [▶ 494]).

bError: This output indicates with a TRUE that there is an error.

eErrorCode: E_HVACErrorCodes (see E_HVACErrorCodes [▶ 491]).

bInvalidParameter: TRUE if an error has occurred 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 amplification. The variable is saved persistently. Preset to 0.

tTi: Integration time. The I-component corrects the residual control deviation following correction of the P-component. 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 actuator. 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 set value 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 and set values 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 Example [▶ 507]). 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 from the sequence.

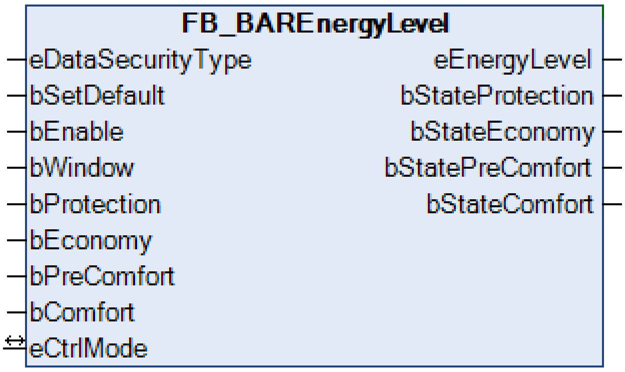
Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5 HVAC Room Functions

5.1.5.1 Air conditioning

5.1.5.1.1 FB_BAREnergyLevel



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 scheduler.

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 scheduler or by presence.

VAR_INPUT

```
eDataSecurityType        : E_HVACDataSecurityType;
bSetDefault              : BOOL;
bEnable                  : BOOL;
bWindow                  : BOOL;
bProtection              : BOOL;
bEconomy                 : BOOL;
bPreComfort              : BOOL;
bComfort                 : BOOL;
```

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 (see E_BAREnergyLevel [▸ 499]).

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

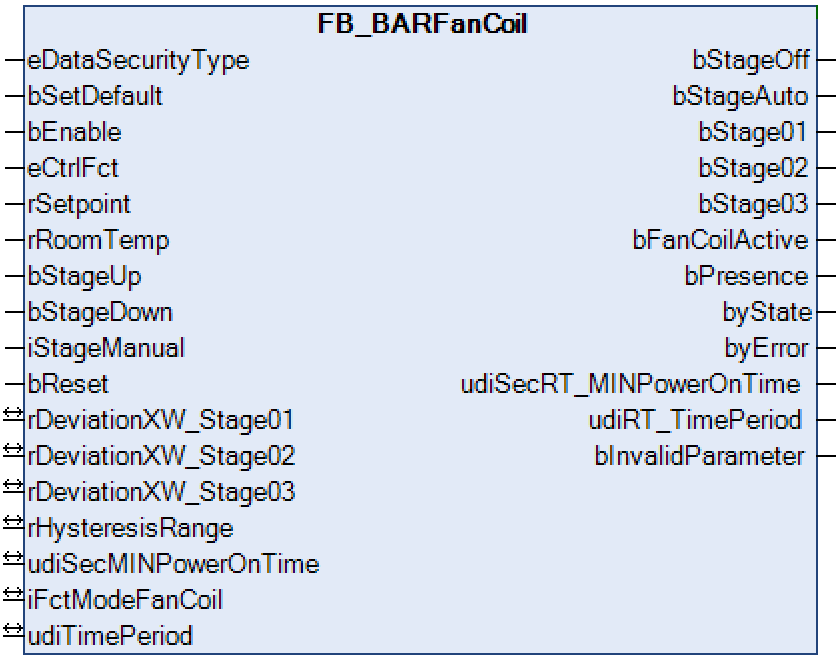
```
eCtrlMode      : E_BAREnergyLevel;
```

eCtrlMode: Using this ENUM the operation mode can be preselected from the building management level (see E_BAREnergyLevel [▸ 499]). The variable is saved persistently. Preset to automatic.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.1.2 FB_BARFanCoil



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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 output eCtrlFct [▶ 499] of **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 pushbutton.

bStageDown: Local adjustment of the fan stage, stepwise decrease by pushbutton.

iStageManual: The manual fan stage can be/is centrally adjusted via this input.
iStageManual: = 0, corresponds to the fan stage OFF
iStageManual: = 1, corresponds to the fan stage AUTO
iStageManual: = 2, corresponds to fan stage 01 active
iStageManual: = *3,* corresponds to fan stage 02 active
iStageManual: = *4*, corresponds to fan stage 03 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 stage 01 active.

bStage02: TRUE, fan stage 02 active.

bStage03: TRUE, fan stage 03 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 specification is active
byState.4:= bReset

byState.5 = fan stage 01 active
byState.6 = fan stage 02 active
byState.7 = fan stage 03 active

byError: Output of the error as a 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 there is an incorrect input parameter. *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 stage 01. The variable is saved persistently. Preset to 0.7.

rDeviationXW_Stage02: Limit value of the control deviation for fan stage 02. The variable is saved persistently. Preset to 1.7.

rDeviationXW_Stage03: Limit value of the control deviation for fan stage 03. The variable is saved persistently. Preset to 2.1.

rHysteresisRange: Hysteresis range placed around the limit value.
Example: A limit value of 0.7 and a hysteresis range of 0.2 results in fan stage 01 being switched on at a control deviation of > 0.8.
At a control deviation of < 0.6 fan stage 01 is switched off.
The variable is saved persistently. Preset to 0.2.

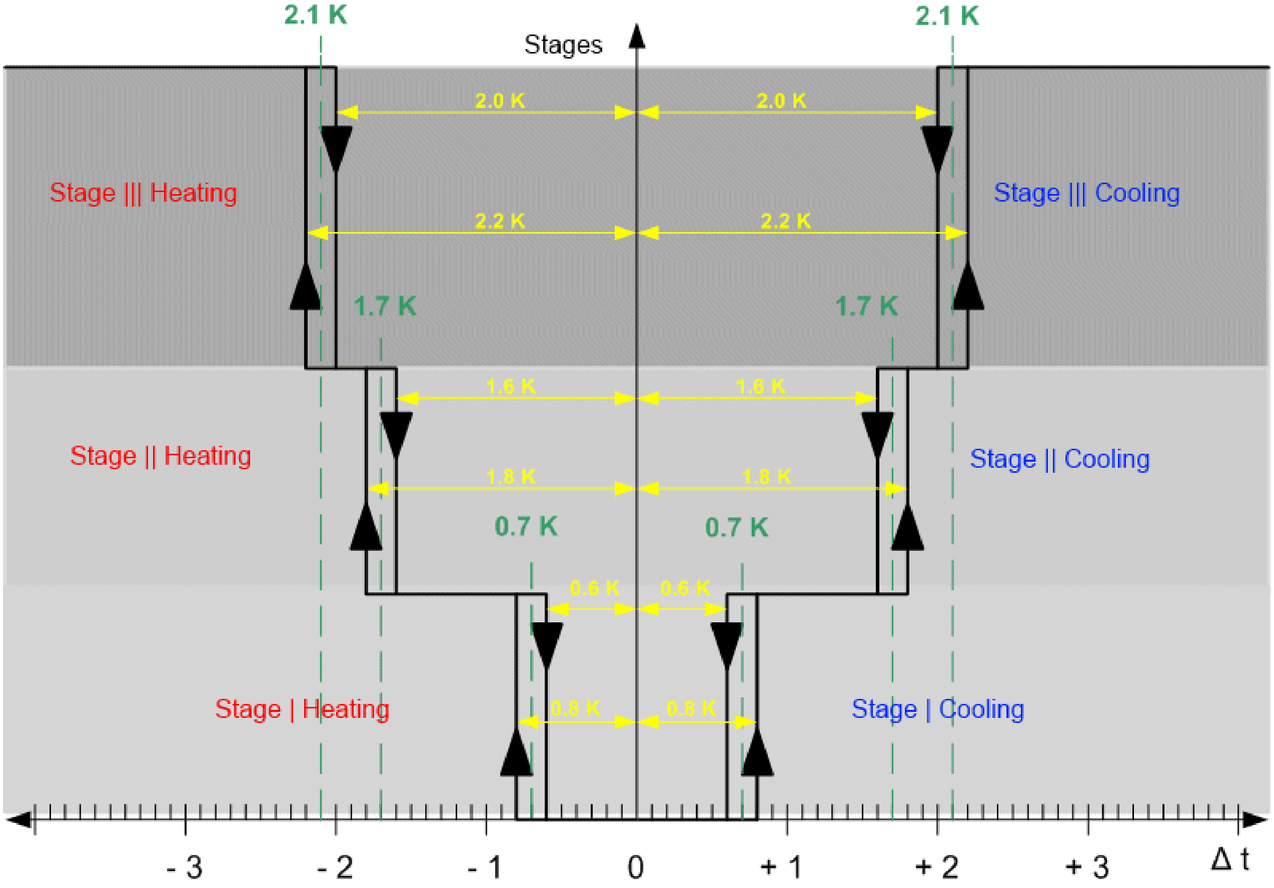
udiSecMINPowerOnTime: Minimum switch-on time that a fan must run for in a stage before switching to a different stage or switching off. Input in seconds (e.g. 120 corresponds to 120 s). The variable is saved persistently. Preset to 120 s.

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. The variable is saved persistently. Preset to 60 min.

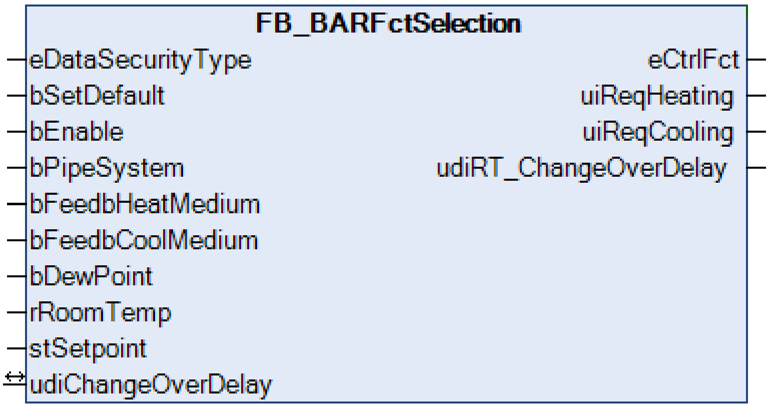
Illustration of the fan control with the default parameters:



Requirements

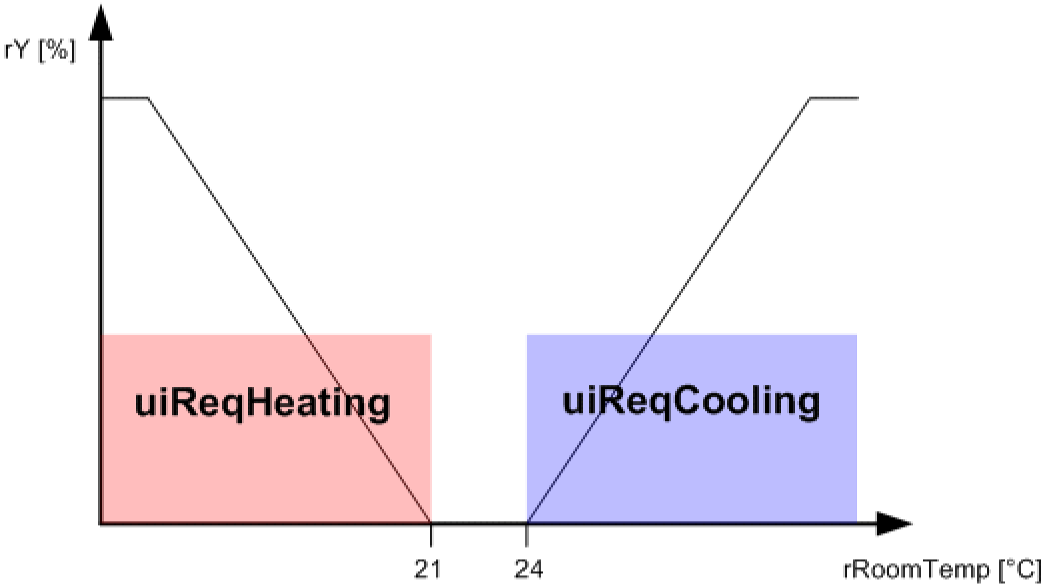
Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.1.3 FB_BARFctSelection



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 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**	**bFeedbHeatMedium**	**bFeedbCoolMedium**	**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	OFF
					FALSE	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	OFF
					FALSE	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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 (see ST_BARSetpointRoom [▶ 503]).

VAR_OUTPUT

```
eCtrlFct             : E_BARCtrlFct;
uiReqHeating         : UINT;
uiReqCooling         : UINT;
udiRT_ChangeOverDelay : UDINT;
```

eCtrlFct: This output contains the current control function (see E_BARCtrlFct [▶ 499]).

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

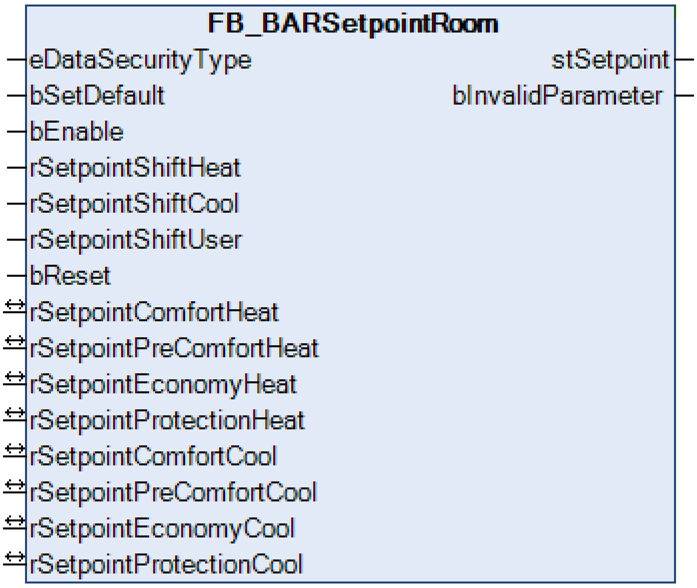
```
uiChangeOverDelay  : UINT;
```

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.

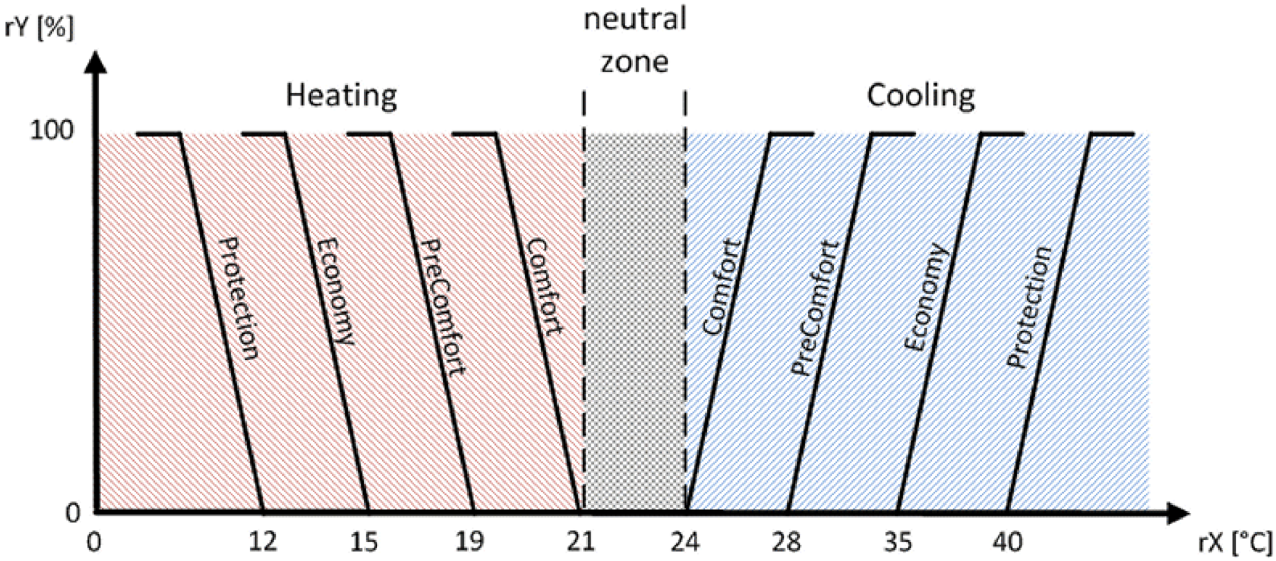
Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.1.4 FB_BARSetpointRoom



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 controller is provided with an energetical optimum setpoint regarding the use of the room and the selected heating or cooling operation.



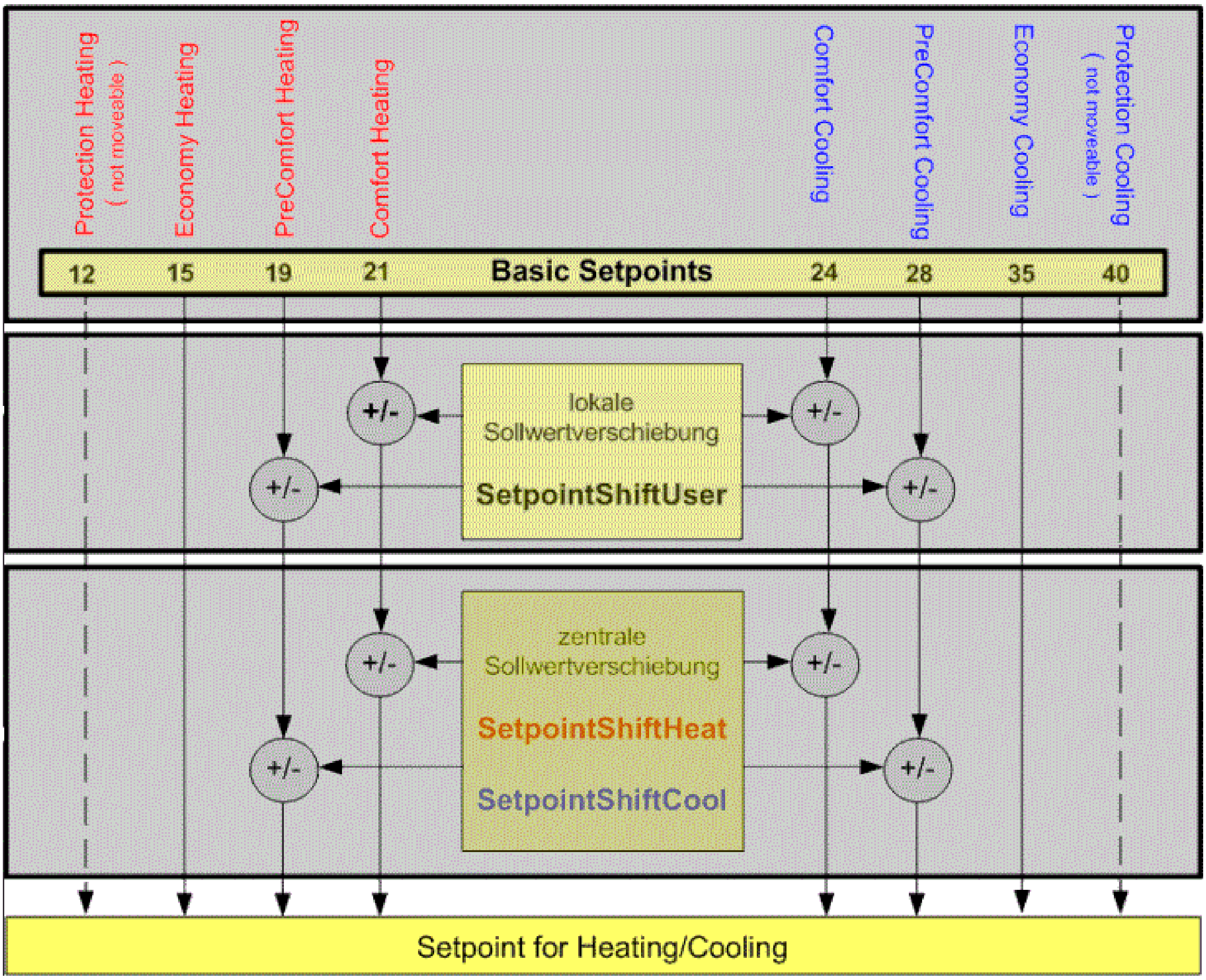
The resulting setpoint for the different energy levels is made up of:

1. the base setpoint value

2. the local setpoint value shift (not in the case of the Protection and Economy setpoints)

3. the central setpoint value shift (not in the case of the Protection and Economy 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 [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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		Protection-Heating	Economy-Heating	Pre-ComfortHeating	ComfortHeating	ComfortCooling	Pre-ComfortCooling	Economy-Cooling	Protection-Cooling
Base setpoint value [°C]		12	15	19	21	24	28	35	40
SetpointShiftHeat [K]	+3	-	-	-	+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		Protection-Heating	Economy-Heating	Pre-ComfortHeating	ComfortHeating	ComfortCooling	Pre-ComfortCooling	Economy-Cooling	Protection-Cooling
Base setpoint value [°C]		12	15	19	21	24	28	35	40
SetpointShiftHeat [K]	-3	-	-	-3	-3	-	-	-	-
Resulting setpoint [°C]		12	15	16	18	24	28	35	40

rSetpointShiftCool: The rSetpointShiftCool variable 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		Protection-Heating	Economy-Heating	Pre-ComfortHeating	ComfortHeating	Comfort-Cooling	Pre-Comfort-Cooling	Economy-Cooling	Protection-Cooling
Base setpoint value [°C]		12	15	19	21	24	28	35	40
SetpointShiftCool [K]	+3	-	-	-	-	+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		Protection-Heating	Economy-Heating	Pre-ComfortHeating	ComfortHeating	Comfort-Cooling	Pre-Comfort-Cooling	Economy-Cooling	Protection-Cooling
Base setpoint value [°C]		12	15	19	21	24	28	35	40
SetpointShiftCool [K]	-3	-	-	-	-	-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		Protection-Heating	Economy-Heating	Pre-ComfortHeating	ComfortHeating	Comfort-Cooling	Pre-Comfort-Cooling	Economy-Cooling	Protection-Cooling
Base setpoint value [°C]		12	15	19	21	24	28	35	40
SetpointShiftCool [K]	+8	-	-	-	-	+8	+8	-	-
Resulting setpoint [°C]		12	15	19	21	32	36	36	40

rSetpointShiftUser: The *rSetpointShiftUser* variable is used for the user's local setpoint value shift. A positive value of *rSetpointShiftUser* affects the setpoints of ComfortHeating, ComfortCooling and PreComfortCooling.
Example:

Energy levels		Protection-Heating	EconomyHeating	PreComfortHeating	ComfortHeating	Comfort-Cooling	PreComfortCooling	EconomyCooling	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	19	24	27	31	35	40

A negative value of *rSetpointShiftUser* affects the setpoints of PreComfortHeating, ComfortHeating and ComfortCooling.
Example:

Energy levels		Protection-Heating	EconomyHeating	PreComfortHeating	ComfortHeating	Comfort-Cooling	PreComfortCooling	EconomyCooling	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 the energy levels (see ST_BARSetpointRoom [▶ 503]).

bInvalidParameter: Indicates that there is an incorrect input parameter. *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.

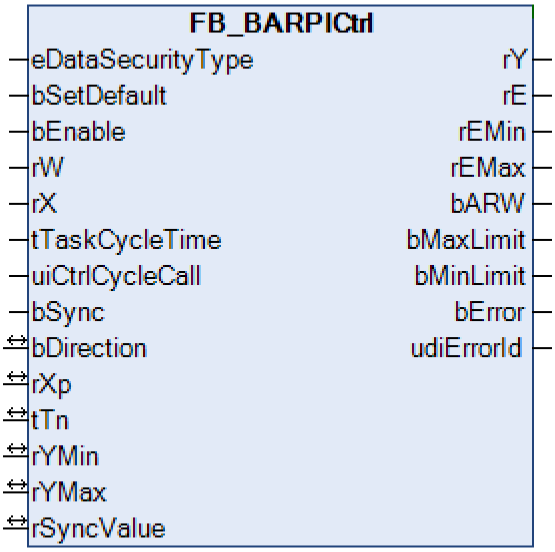
rSetpointProtectionCool: Setpoint for the Protection cooling energy level. The variable is saved persistently. Preset to 40.0.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.2 Controller

5.1.5.2.1 FB_BARPICtrl



Simple PI controller. The control gain has no influence on the I-component.

This PI controller does not work directly with an adjustable gain factor K_p, but with 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.

VAR_INPUT

```
eDataSecurityType  : E_HVACDataSecurityType;
bSetDefault        : BOOL;
bEnable            : BOOL;
rW                 : REAL;
rX                 : REAL;
tTaskCycleTime     : TIME;
uiCtrlCycleCall    : UINT;
bSync              : BOOL;
```

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 evaluated 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*=0 ms, 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 [▸ 202])

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 [▸ 318].

VAR_IN_OUT

```
bDirection      : BOOL;
rXp             : REAL;
tTn             : TIME;
rYMin           : REAL;
rYMax           : REAL;
rSyncValue      : REAL;
```

bDirection: The control direction [▶ 202] 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-component. The variable is saved persistently. Preset to 100.0.

tTn: Integral action time in seconds. The I-component corrects the residual control deviation following the correction of the P-component. 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 actuator. A zero value at this input deactivates the I-component. The variable is saved persistently. Preset to 30 s.

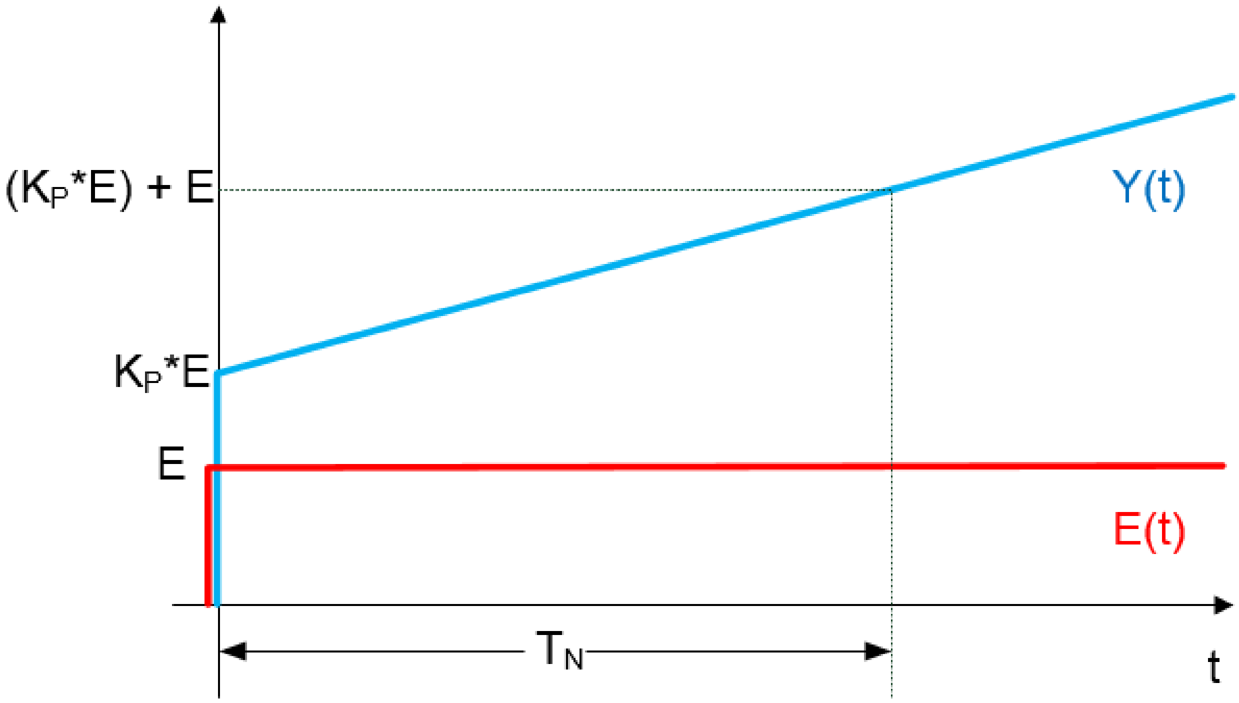
rYMin / rYMax: Limit the working range of the controller. Several other function blocks, e.g. sequencers, require a symmetrical control range (-100 to +100). In the case of a cascade structure, the working range of the master controller determines the setpoint of the slave controller. For example, 15° to 25° as the limitation of the supply air setpoint for an exhaust/supply air cascade control. The variable is saved persistently. Preset to 0.0 or 100.0 respectively.

rSyncValue: The control value *rY* is set to this value on a rising edge at the input *bSync*. 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; the control gain has no influence on the integral component. Reaction of the output Y(t) to a control deviation jump by E: In the case of a jump of the control deviation by E, the output Y initially jumps to K_P*E due to the proportional component and then grows by a further E in each interval T_N.

The controller is designed so that it begins at 0, i.e. without the K_P*E jump, each time the PLC is reset or restarted.



Basic Function

A TRUE signal at the input *bEnable* enables the module. Then the internal controlling algorithm is processed. The input value *uiCtrlCycleCall* indicates, after what number of PLC cycles the internal controlling algorithm is going to be processed.
If *uiCtrlCycleCall* =1, a recalculation is executed in each PLC cycle, if it is set to 100, a new calculation is executed only each 100 PLC cycles. The PLC cycle time is part of the calculation of the actuating variable. A wrong input value causes a wrong calculation.
The inputs *rW* (desired value), *rX* (actual value), *rXp* (proportional band) and *rTn* (reset time) are the input parameters of the PI-controller. Via them the output parameters rY (actuating variable) and *rE* (control deviation) are evaluated in each calculation cycle. The actuating variable can additionally be limited via the inputs *rYMin* and *rYMax* .

Setting via the proportional band

The adjustment of the gain factor K_p of a controller often includes 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-component 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→ $Y=E*K_p$

This relationship also applies to the changes in the control deviation and the control value:

- Control value change = control deviation change •gain factor→ $\Delta Y=\Delta E*K_p$

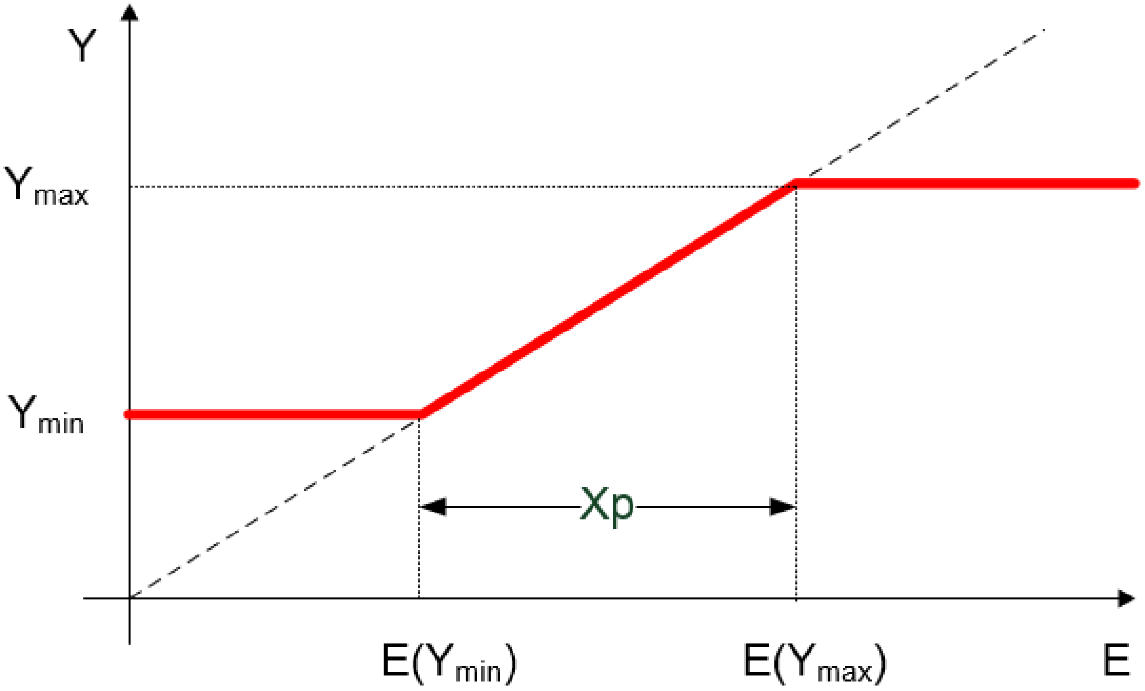
Referenced to the minimum and maximum value of the control value, Y_{min} and Y_{max}:

- $Y_{max}-Y_{min}=(E(Y_{max})-E(Y_{min}))*K_p$

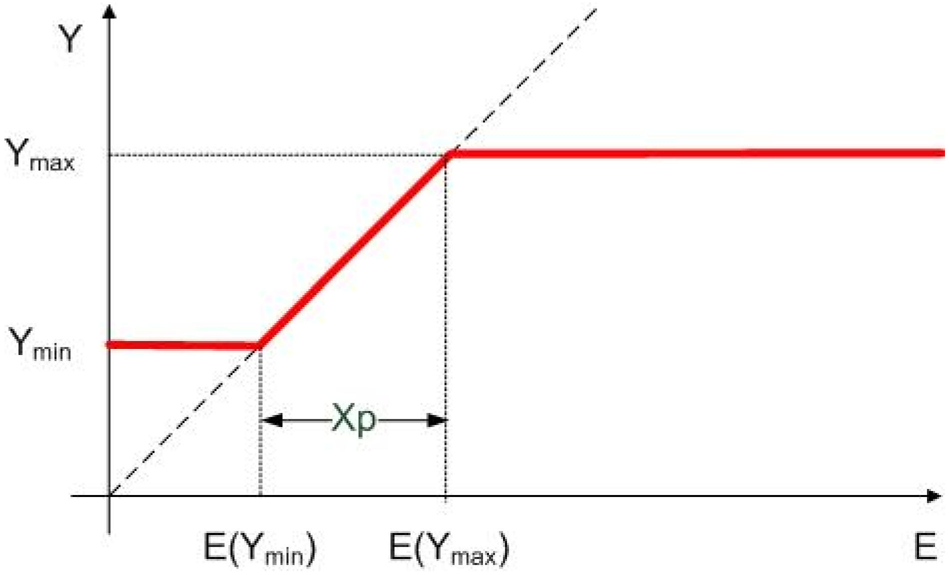
This difference, $E(Y_{max})$ - $E(Y_{min})$, is called the proportional band (Xp). Transposed, the equation is then:

- $K_p=(Y_{max}-Y_{min})/X_p$

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 in the Kp factor. However, the control deviation limit values $E(Y_{max})$-$E(Y_{min})$ are shifted:



Control-direction

Control-direction

If *bDirection* is set to FALSE, the control-direction will be changed, so a negative control-deviation will cause a positive value at the control-output. This is done by changing the internal calculation of the control-deviation.

bDirection	rE (control-deviation)	control-direction
TRUE	lrX-lrW (actual value-setpoint)	direct (cooling)
FALSE	lrW-lrX (setpoint-actual value)	reverse (heating)

Anti-Reset-Windup (ARW)

If the controller runs in this limitation, the I-component is hold internally at the last value. Without this, the I-component-values could increase unhindered. In case of a direction reversal of the controller, a huge I-component has to be eliminated, before the controller can operate within its normal limitations again. If calculation of the I-component is stopped, the output bARW is set.

Special Case Tn=0 to switch-off the I-component

In the step response diagram above you see, that the I-component's influence is the weaker, the bigger the integral-time T_n has been set. If the integral-time goes to infinity, the I-component's influence goes to zero. Reverse a constantly decreasing reset time would nurture the I-component, by T_n=0 the actuating variable would go to infinity. This special case is used to **switch-off** the I-component. This is an internal exception, because the integral-time directly belongs to the I-component and should switch-off the I-component via "0-item" metaphorically.

Synchronisation

A rising edge at *bSync* sets the controller output *rY* directly to *rSyncValue*, if the controller has been activated via a TRUE signal at *bEnable*. If not, the rising edge at *bSync* is ignored.

Passive-behaviour (bEnable = FALSE or bError = TRUE)

If the controller is wrong parameterized, the execution is stopped, the output *bError* is set and the corresponding error code is provided to *udiErrorID*, see Error codes [▸ 318]. The Module 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

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.3 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 [▸ 220] - simple light switching circuit without dimming function.

FB_BARLightCircuitDim [▸ 222] - simple light switching circuit with dimming function.

FB_BARAutomaticLight [▸ 204] - presence-controlled automatic light with switch-off delay.

FB_BARStairwellAutomatic [▸ 225] - stairwell lighting with early warning sequence.

FB_BARTwilightAutomatic [▸ 228] - twilight circuit.

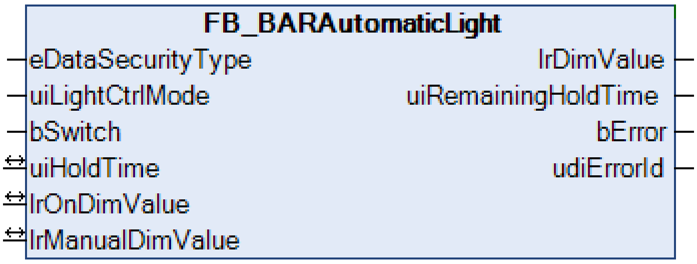
FB_BARDaylightControl [▸ 213] - daylight controller without dimming processes

FB_BARConstantLightControl [▸ 207] - constant light control with continuous output of analog values.

The group of **actuator functions** is currently represented by only one function block.

FB_BARLightActuator [▸ 217] - 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.

5.1.5.3.1 FB_BARAutomaticLight

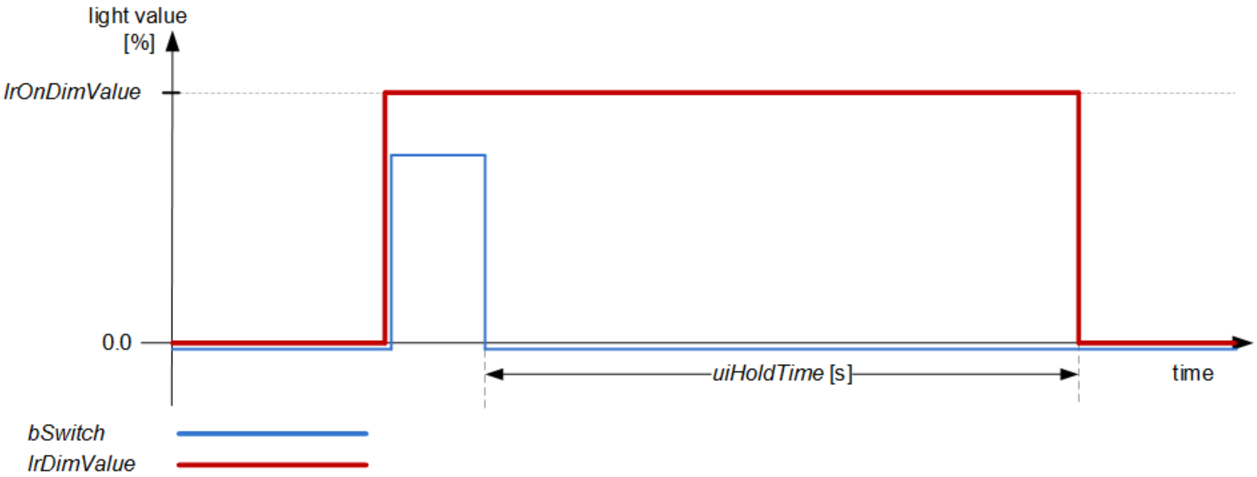


Function block for an automatic light circuit as used in corridors or sanitary facilities.

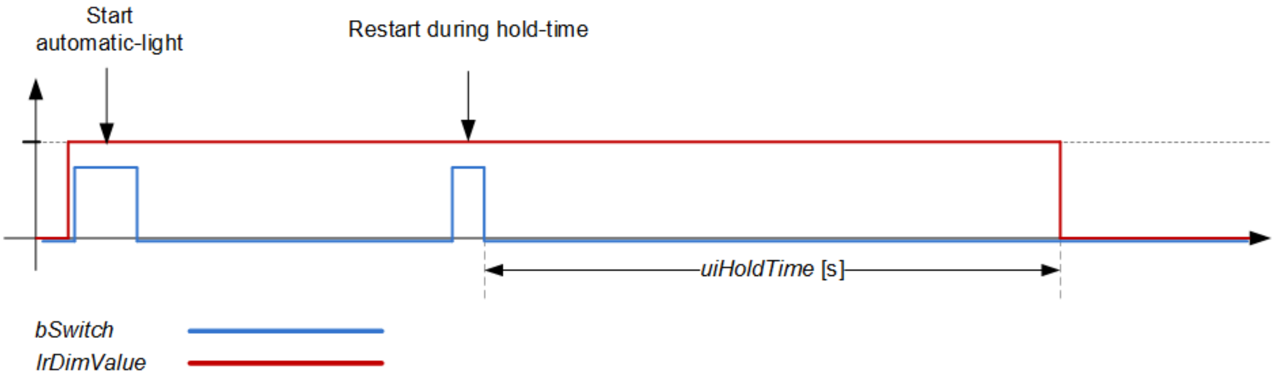
The function block knows 3 different modes, which can be set via the input *uiLightCtrlMode*:

- 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 *lrDimValue* to the value entered under *lrManualDimValue*. A negative edge starts the holding time generator. If the holding time *uiHoldTime* [s] has expired, the output *lrDimValue* is reset to 0.0.



The sequence can be restarted at any time:



The *bSwitch* input has no function in the manual modes: if *uiLightCtrlMode*=1 the output value *lrDimValue* is set constantly to *lrManualDimValue* and if *uiLightCtrlMode*=2 it is set constantly to 0.0.

Resetting the holding time

Changing to manual mode resets any holding time that had started up to that point.

VAR_INPUT

```
eDataSecurityType    : E_HVACDataSecurityType;
uiLightCtrlMode      : UINT;
bSwitch              : BOOL;
```

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 holding 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 holding time (*uiHoldTime*), in order to illustrate the countdown of the holding 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 [▶ 318].

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

```
uiHoldTime       : UINT;
lrOnDimValue     : LREAL;
lrManualDimValue : LREAL;
```

uiHoldTime : Holding time [s] of the automatic light controller after a falling edge on *bSwitch*.

lrOnDimValue : A dimming value in automatic mode (uiLightCtrlMode=0).

lrManualDimValue : Output dimming value in Manual On mode (uiLightCtrlMode=1).

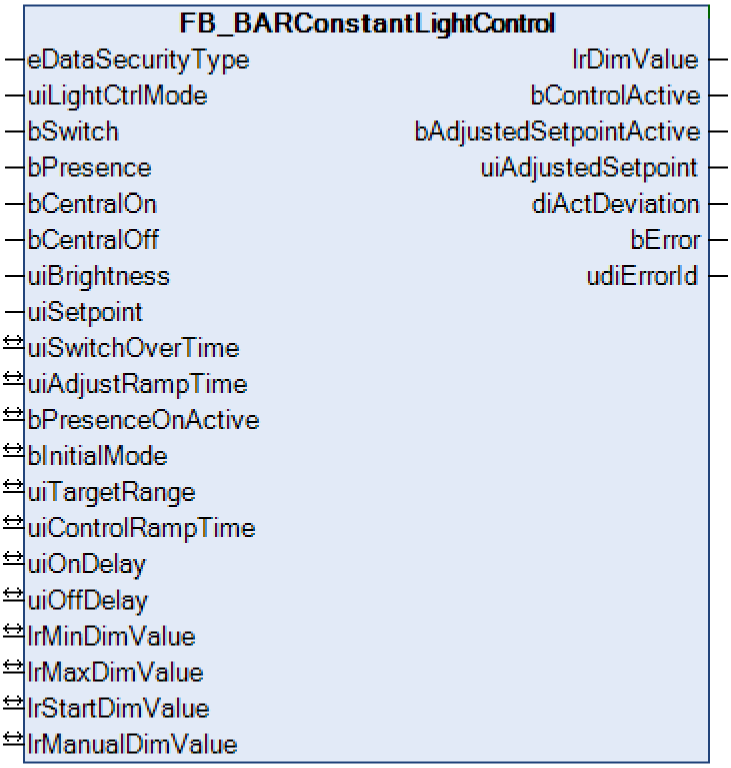
Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

Documents about this

🗎 example_persistent.zip (Resources/zip/5090272523.zip)

5.1.5.3.2 FB_BARConstantLightControl



The constant light control 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 control is enabled when the room is entered or by a rising edge on the input *bPresence*. Optionally the constant light control can also be operated by a pushbutton. A short pulse on the input *bToggle* enables the constant light control 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 pushbutton 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 stored as a setpoint value. 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 control 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 value or the value of *uiSetpointlValue* 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 external 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*.
If the external brightness decreases and the light should be turned on again, it can be delayed by *uiOnDelay* [sec].

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 control can be deactivated and the lighting can be switched on and off in manual mode. This deactivates the edge detection at the inputs *bCentrolOn,bCentalOff* and *bToggle*.

The function block knows three different modes, which can be set via the input *uiLightCtrlMode*:

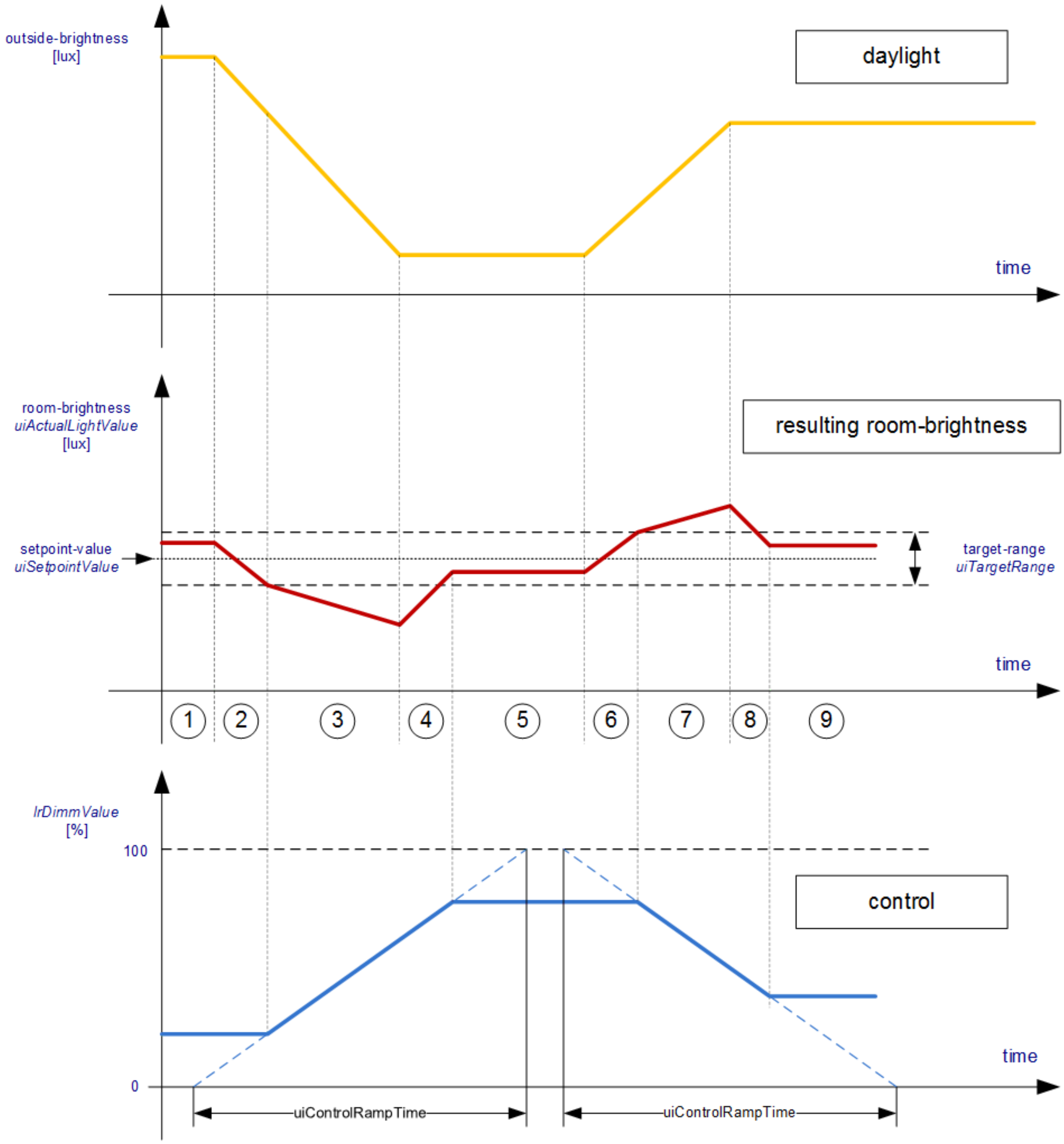
- Automatic mode
- Manual On mode
- Manual Off mode

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 control if it had been inactive before. Another short signal at *bSwitch* deactivates the constant light control again. A long TRUE signal on *bSwitch* switches the function block to setpoint value 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 *lrDimValue* is initially set in the active state to the starting value *lrStartDimValue*. 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 *lrDimValue* 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 succeds 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 succeds in damping the gain of room-brightness.

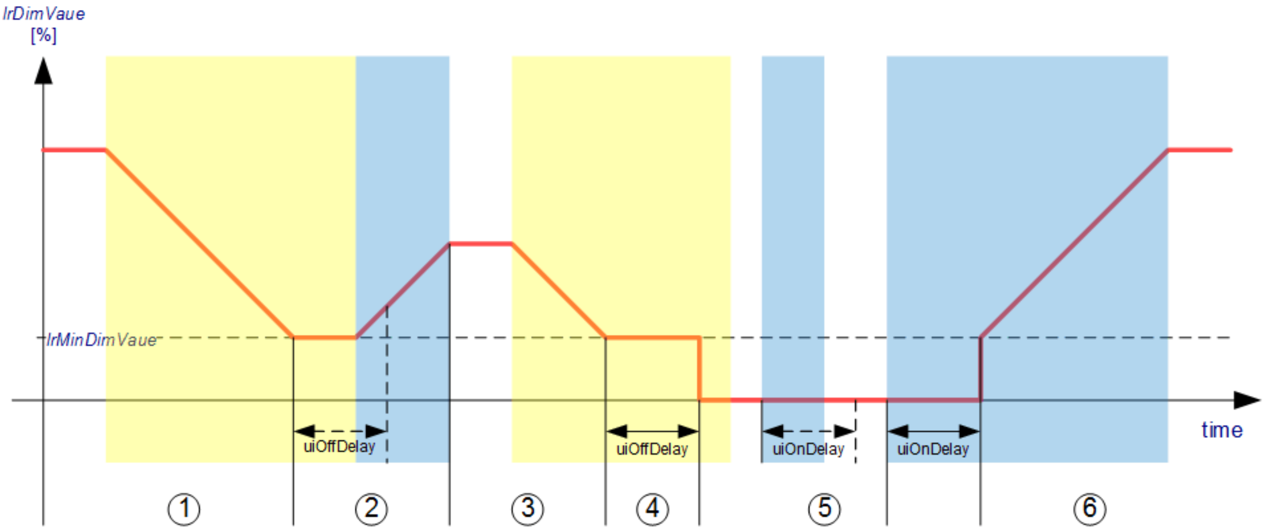
8 The outside-brightness does not increase any more. The control-ramp is now able to compensate the room-brightness.

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 external brightness increases, then less and less artificial light is necessary in order to obtain the desired total brightness. The lighting can be switched off completely if the external brightness is sufficient. 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 *lrMinDimValue*. 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 *lrDimValue*. 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 *lrMinDimValue*. The following diagram is intended to clarify the behavior:

The yellow areas indicate the ranges where the external brightness permits the lighting to be dimmed down; in the blue areas, conversely, the external brightness alone is not sufficient to obtain the desired room brightness. In the white areas the output control value *lrDimValue* 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 value

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 operation

The inputs *bSwitch, bCentralOn* and *bCentralOff* have no function in the manual operation modes: if *uiLightCtrlMode*=1 the output value *lrDimValue* is set constantly to *lrManualDimValue* and if *uiLightCtrlMode*=2 it is set constantly to 0.0.

Manual mode / automatic mode

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 controller must be restarted.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
uiLightCtrlMode   : UINT;
bSwitch           : BOOL;
bPresence         : BOOL;
bCentralOn        : BOOL;
bCentralOff       : BOOL;
uiBrightness      : UINT;
uiSetpoint        : UINT;
```

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 control is inactive – the value *lrManualDimValue* is output without a delay at the output *lrDimValueOut*, all other command inputs are ineffective.
- 2: Manual Off mode; constant light control is inactive – the value 0.0 is output without a delay at the output *lrDimValueOut*, all other command inputs are ineffective.

bSwitch: Rising edges on this input switch the control on and off alternately. When switching off the output *lrDimValue* 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 *lrDimValue* to 0.0. This command input is active only in automatic mode (*uiLightCtrlMode*=0).

uiBrightness: Light actual value [lux].

uiSetpointValue: Light setpoint value [lux].

VAR_OUTPUT

```
lrDimValue               : LREAL;
bControlActive           : BOOL;
bAdjustedSetpointActive  : BOOL;
uiAdjustedSetpoint       : UINT;
diActDeviation           : DINT;
bError                   : BOOL;
udiErrorId               : UDINT;
```

lrDimValue: 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 *lrDimValue*=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 [▶ 318].

VAR_IN_OUT

```
uiSwitchOverTime   : UINT;
uiAdjustRampTime   : UINT;
bPresenceOnActive : BOOL;
bInitialMode       : BOOL;
uiTargetRange      : UINT;
uiControlRampTime : UINT;
uiOnDelay          : UINT;
uiOffDelay         : UINT;
lrMinDimValue      : LREAL;
lrMaxDimValue      : LREAL;
lrStartDimValue    : LREAL;
lrManualDimValue   : LREAL;
```

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

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 [▶ 207].

lrMaxDimValue: Upper limit to which the output *lrDimValue* 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).

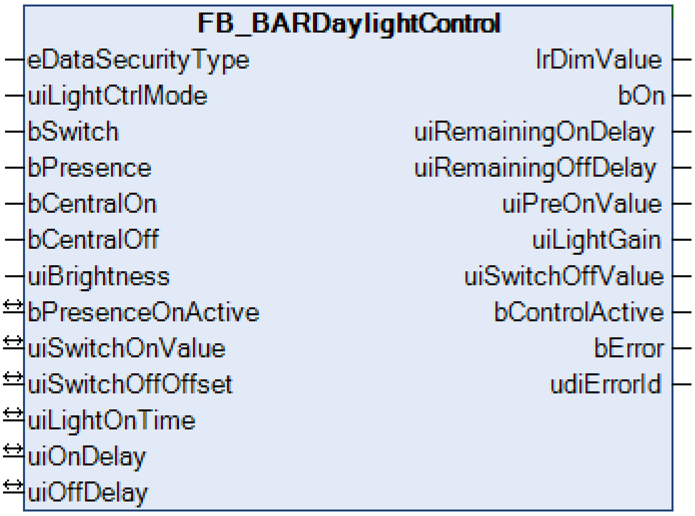
Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

Documents about this

example_persistent.zip (Resources/zip/5090272523.zip)

5.1.5.3.3 FB_BARDaylightControl



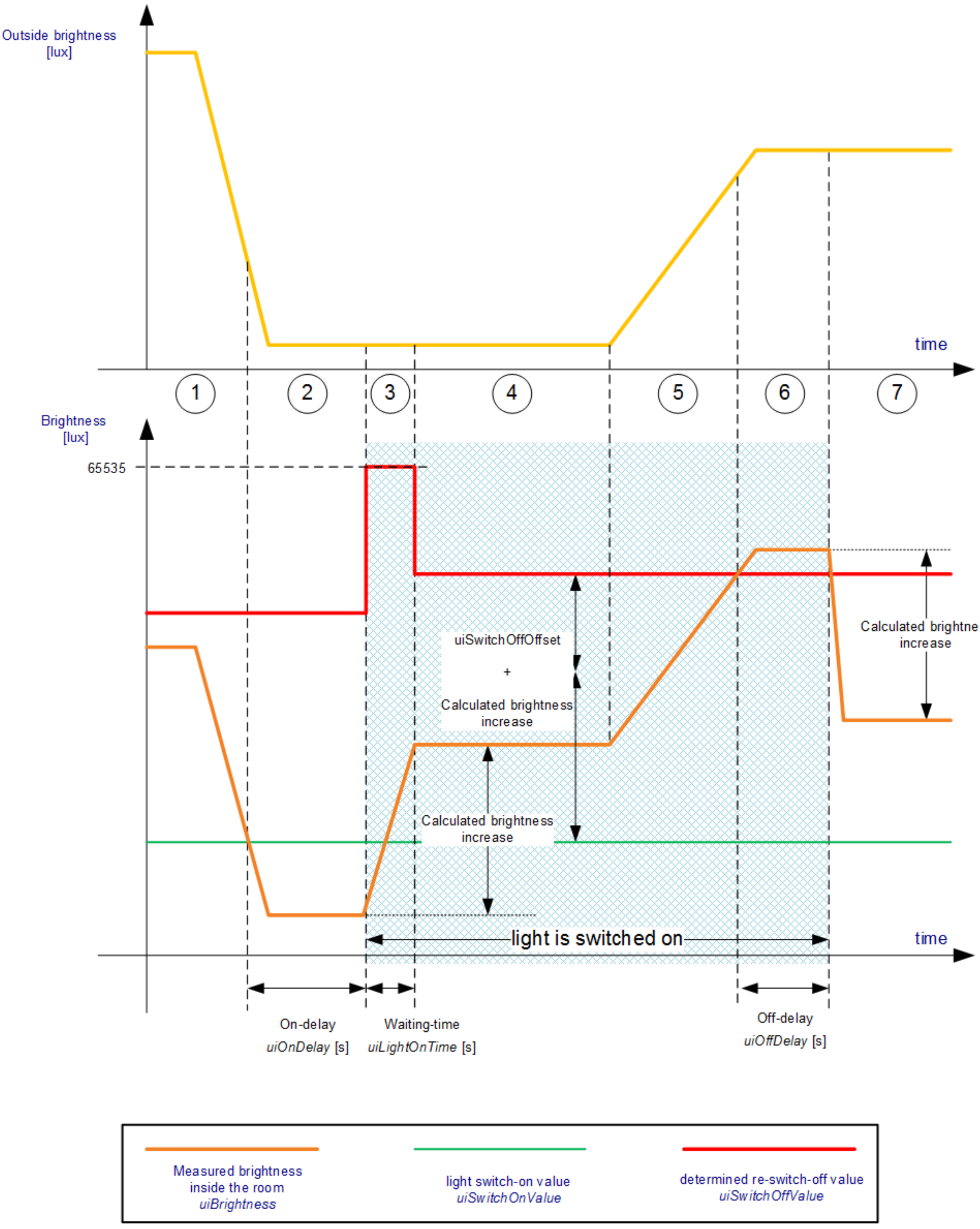
Daylight switch. Unlike the constant light control [▶ 207], this automatic switch operates not with dimming values, but merely switches the light on or off depending on the measured brightness.

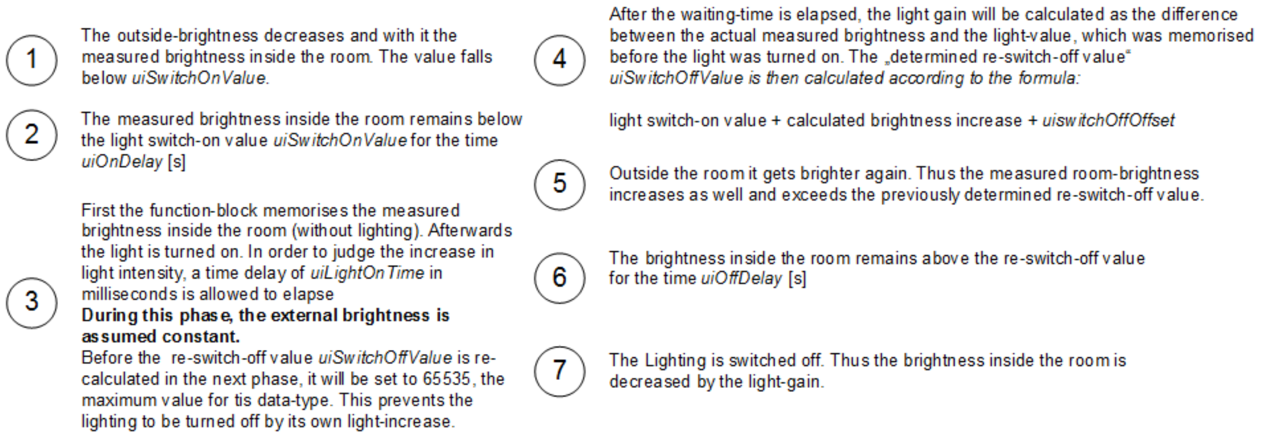
The function block knows three different modes, which can be set via the input *uiLightCtrlMode*:

- 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 external 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 external 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 external 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.


Outside brightness [lux]
time
1
2
3
4
5
6
7
Brightness [lux]
65535
uiSwitchOffOffset
+
Calculated brightness increase
Calculated brightness increase
Calculated brightness increase
light is switched on
time
On-delay
uiOnDelay [s]
Waiting-time
uiLightOnTime [s]
Off-delay
uiOffDelay [s]
Measured brightness inside the room
uiBrightness
light switch-on value
uiSwitchOnValue
determined re-switch-off value
uiSwitchOffValue



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]

3 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 re-calculated in the next phase, it will be set to 65535, the maximum value for tis data-type. This prevents the lighting to be turned off by its own light-increase.

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:*

light switch-on value + calculated brightness increase + *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 operation

The inputs bSwitch, bCentralOn and bCentralOff have no function in the manual operation modes: the output value *bOn* is set to *TRUE* if *uiLightCtrlMode*=1 and to FALSE if *uiLightCtrlMode*=2.

> **Manual mode / automatic mode**
>
> Changing to manual mode resets any control process that had previously started. On re-entering automatic mode the output value bOn is FALSE and the controller must be restarted.

VAR_INPUT

```
eDataSecurityType  : E_HVACDataSecurityType;
uiLightCtrlMode    : UINT;
bSwitch            : BOOL;
bCentralOn         : BOOL;
bCentralOff        : BOOL;
uiBrightness       : UINT;
```

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 control is inactive – the value *lrManualDimValue* is output without a delay at the output *lrDimValueOut*, all other command inputs are ineffective.
- 2: Manual Off mode; constant light control is inactive – the value 0.0 is output without a delay at the output *lrDimValueOut*, all other command inputs are ineffective.

bSwitch: Rising edges on this input switch the control on and off alternately. When switching off the output *lrDimValue* 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 *lrDimValue* to 0.0. This command input is active only in automatic mode (*uiLightCtrlMode*=0).

uiBrightness: Light actual value [lux].

VAR_OUTPUT

```
lrDimValue          : LREAL;
bOn                 : BOOL;
uiRemainingOnDelay  : UINT;
uiRemainingOffDelay : UINT;
uiPreOnValue        : UINT;
uiLightGain         : UINT;
uiSwitchOffValue    : UINT;
bControlActive      : BOOL;
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: Switch 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 expiry of an additional waiting time of 1s. 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 [▶ 318].

VAR_IN_OUT

```
uiSwitchOnValue   : UINT;
uiSwitchOffOffset : UINT;
uiLightOnTime     : UINT;
uiOnDelay         : UINT;
uiOffDelay        : UINT;
```

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

uiSwitchOnValue: Light switch-on value. The lighting is switched on after the expiry of the switch-on delay if the external 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 light 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 external brightness sinks in such a manner that the constant light control 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 external brightness increases in such a manner that the constant light control needs to be switched off, then this should take place only after this time delay in order to mask out short-term fluctuations.

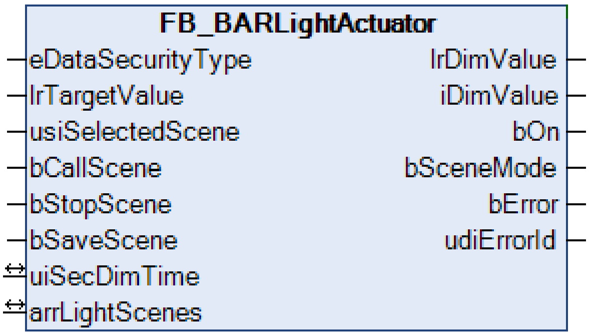
Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

Documents about this

example_persistent.zip (Resources/zip/5090272523.zip)

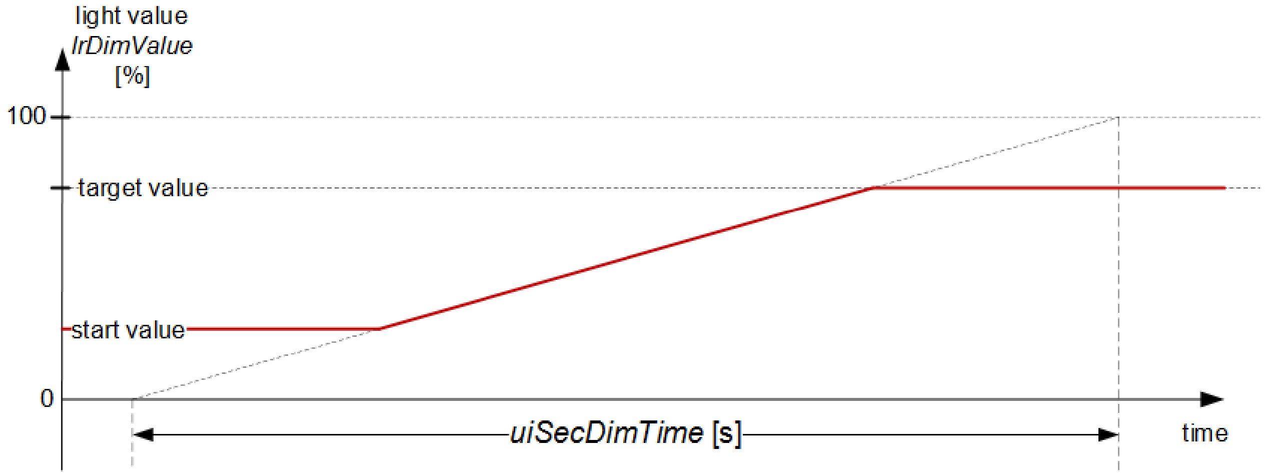
5.1.5.3.4 FB_BARLightActuator

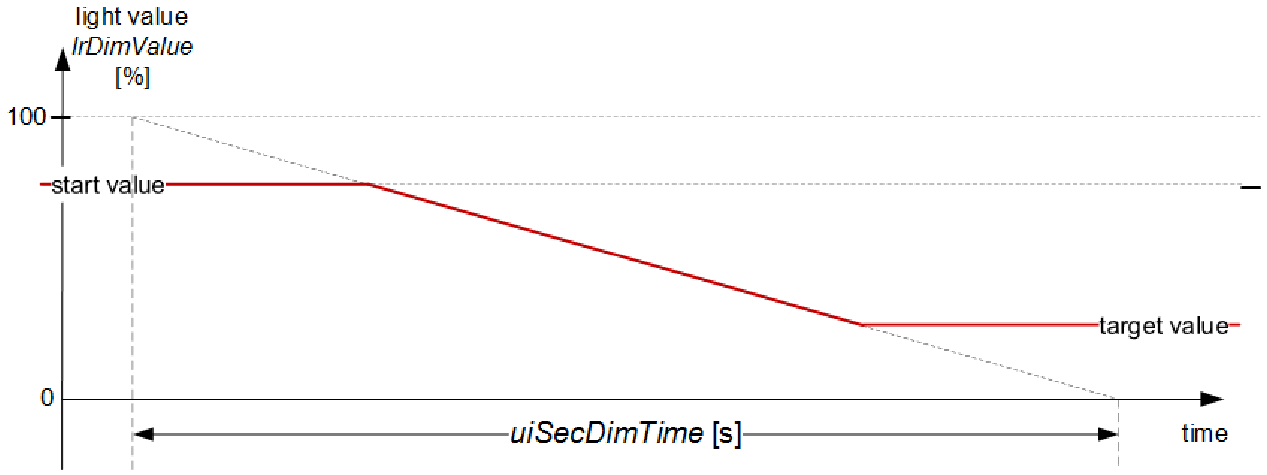


This function block serves to control a conventional light actuator. The outputs cover the value ranges 0..100%, 0..32767 and On/Off.
On top of that the function block contains a scene memory in which up to 21 different light values can be saved.

In principle the function block passes the values on the input *lrTargetValue* through to the output *lrDimValue*. Conversely, a rising 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 rising edge on *bCallScene*. The scene mode is quit again by a change of value on the input *lrTargetValue* or a rising edge on *bStopScene*. The output *bSceneMode* goes back to FALSE and the output *lrDimValue* once again follows the input *lrTargetValue*. A TRUE signal on bSaveScene saves the current light output value into the scene table *arrLightScenes* under the index *usiSelectedScene*.

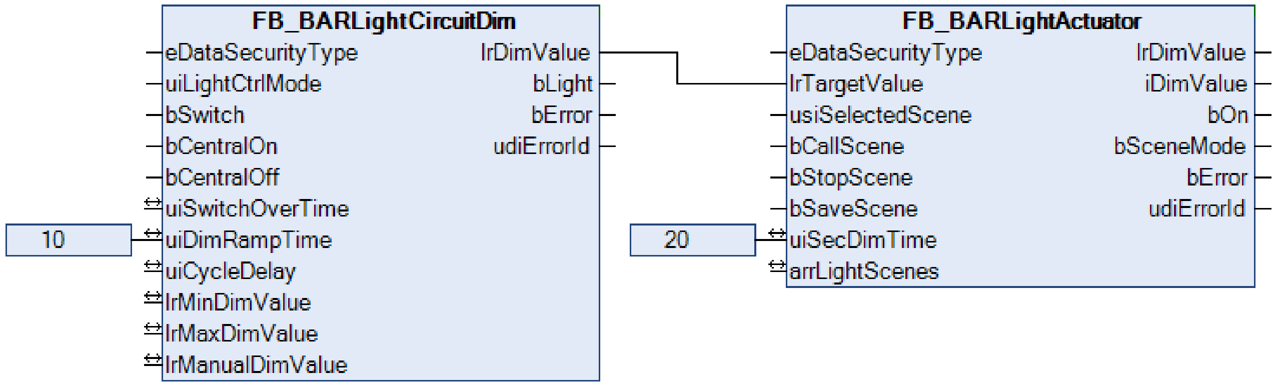
The light output value *lrDimValue* always follows the target values specified to it via a ramp. As in the case of the user function FB_BARLightCircuitDim [▶ 222], this is defined by a ramp time – in this case *uiSecDimTime* – which indicates the time interval in seconds that the light output value should require in order to change by 100%.





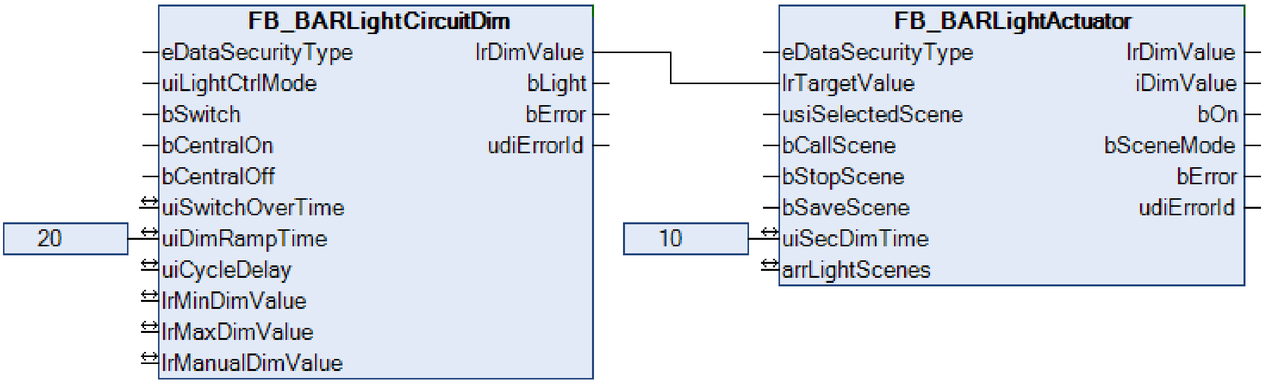
Since the light sensor function block FB_BARLightCircuitDim [▶ 222] also contains a ramp function for presetting the target value, the light actuator function described here is called the "secondary dimtime" - *uiSecDimTime*.
If the two have different values, then the longer ramp time is always the relevant one:

Example 1:



In this constellation the function block FB_BARLightCircuitDim [▶ 222] will change its output value *lrDimValue* 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.

Example 2:



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 [▶ 222] 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 *lrDimValue* to the internally-saved value (see FB_BARLightCircuitDim [▶ 222]). 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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.

lrTargetValue: Target value of the 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 state: *lrDimmValue*=0.0 => *bOn*=FALSE - *lrDimmValue*>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 [▶ 318].

VAR_IN_OUT

```
uiSecDimTime    : UINT;
arrLightScenes : ARRAY[0..20] OF LREAL;
```

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

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.

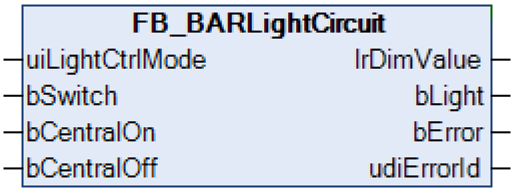
Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

Documents about this

🗎 example_persistent.zip (Resources/zip/5090272523.zip)

5.1.5.3.5 FB_BARLightCircuit



This function block represents a simple light switching circuit without a dimming function.

The function block knows three 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 *lrDimValue* to 0.0, while a rising edge on *bCentralOn* sets the output to 100.0. Rising edges on *bSwitch* make the output *lrDimValue* change each time between 0.0 and 100.0.

In the manual operation modes the output value *lrDimValueOut* is set constantly to the value *100.0* if *uiLightCtrlMode*=1 and constantly to the value 0.0 if *uiLightCtrlMode*=2.

VAR_INPUT

```
uiLightCtrlMode     : UINT;
bSwitch             : BOOL;
bCentralOn          : BOOL;
bCentralOff         : BOOL;
```

uiLightCtrlMode: Operation mode.

- 0: Automatic mode; the output value *lrDimmValue* 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 *lrDimValueOut*, all other command inputs are ineffective.
- 2: Manual Off mode; the value 0.0 is output without a delay at the output *lrDimValueOut*, all other command inputs are ineffective.

bSwitch: Rising edges on *bSwitch* make the output *lrDimValue* change each time between 0.0 and 100.0.

bCentralOn: Switches the output *lrDimValueOut* to 100.0.

bCentralOff: Switches the output *lrDimValueOut* to 0.0.

Note: All switching commands, *bSwitch, bCentralOn* and *bCentralOff* are effective only in automatic mode.

VAR_OUTPUT

```
lrDimValue  : LREAL;
bLight      : BOOL;
bError      : BOOL;
udiErrorId  : UDINT;
```

lrDimValue: 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 *lrDimmValue* = 0.0, otherwise TRUE.

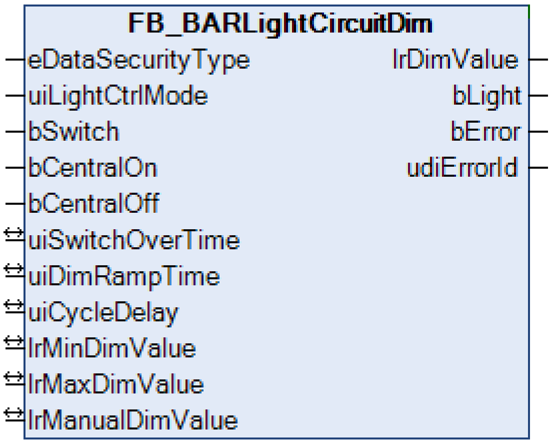
bError: This output is switched to TRUE if the parameters entered are erroneous.

udiErrorId: Contains the error code if the values entered should be erroneous. See Error codes [▶ 318].

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.3.6 FB_BARLightCircuitDim



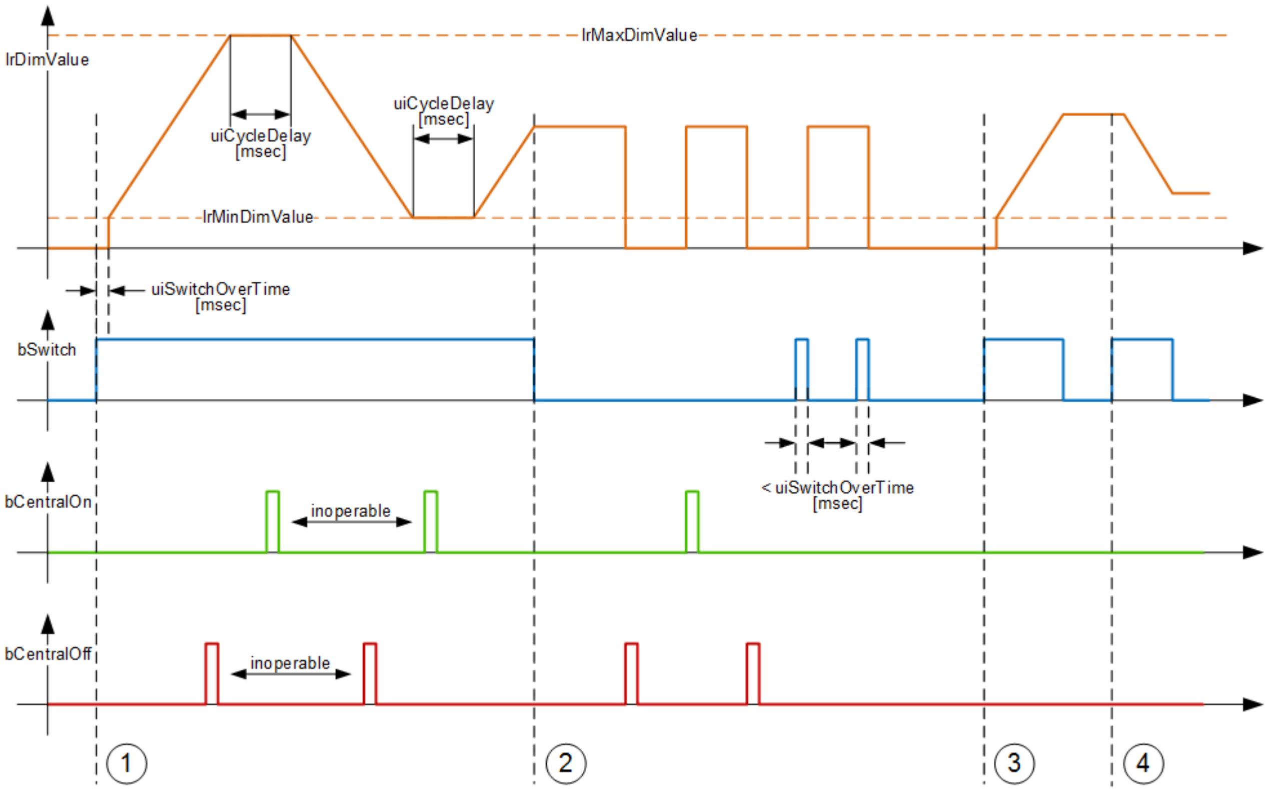
This function block represents a light switching circuit with a dimming function.

The function block knows three 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 percent, even if the limits *lrMinDimValue* and *lrMaxDimValue* are set differently. If the output value *lrDimmValue* 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 *lrDimmValue* will be initially set to *lrMinValue*. Afterwards the dimming-cycle begins and *lrDimmValue* will be continuously increased until it reaches the maximum value *lrMaxValue*. With *bSwitch* still set to TRUE it will remain at this level for *uiCycleDelay* in milliseconds before the output is decreased. When *lrDimmValue* reaches *lrMinValue*, 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 *lrDimValue* to 0 respectively to the last saved value. Rising edges at *bSwitch* will also toggle the output *lrDimValue* 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 (*lrDimmValue*=0.0) the output *lrDimmValue* 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 the manual operation modes the output value *lrDimValueOut* is set constantly to the value *lrManualDimValue* if *uiLightCtrlMode*=1 and constantly to the value 0.0 if *uiLightCtrlMode*=2.

VAR_INPUT

```
eDataSecurityType  : E_HVACDataSecurityType;
uiLightCtrlMode    : UINT;
bSwitch            : BOOL;
bCentralOn         : BOOL;
bCentralOff        : BOOL;
```

eDataSecurityType: If eDataSecurityType [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.

uiLightCtrlMode: Operation mode.

- 0: Automatic mode; the output value *lrDimmValue* can be influenced by the command inputs *bSwitch*, *bCentralOn* and *bCentralOff*
- 1: Manual On mode; the value *lrManualDimValue* is output without a delay at the output *lrDimValueOut*, all other command inputs are ineffective.
- 2: Manual Off mode; the value 0.0 is output without a delay at the output *lrDimValueOut*, all other command inputs are ineffective.

bSwitch: 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.
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 *lrDimValueOut* to the previously saved (at the last switch-off) dimming value, see *bSwitch*.

bCentralOff: Switches the output *lrDimValueOut* to 0.0.

Note: All switching commands, *bSwitch, bCentralOn* and *bCentralOff* are effective only in automatic mode.

VAR_OUTPUT

```
lrDimValue  : LREAL;
bLight      : BOOL;
bError      : BOOL;
udiErrorId  : UDINT;
```

lrDimValue: Light output value; can adopt values between *lrMinDimValue* and *lrMaxDimValue*, see VAR_IN_OUT, but maximally 0..100%.

bLight: FALSE if *lrDimmValue* = 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 [▸ 318].

VAR_IN_OUT

```
uiSwitchOverTime : UINT;
uiDimRampTime    : UINT;
uiCycleDelay     : UINT;
lrMinDimValue    : LREAL;
lrMaxDimValue    : LREAL;
lrManualDimValue : LREAL;
```

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 to be defined as input

variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

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).

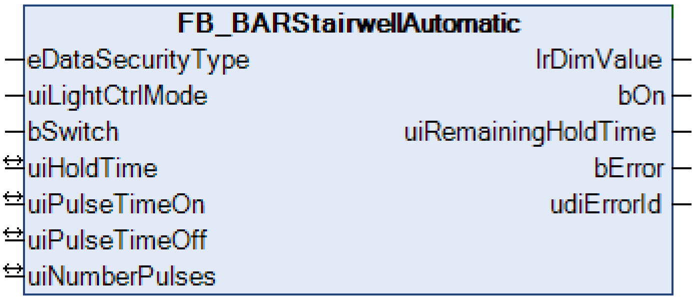
Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

Documents about this

example_persistent.zip (Resources/zip/5090272523.zip)

5.1.5.3.7 FB_BARStairwellAutomatic

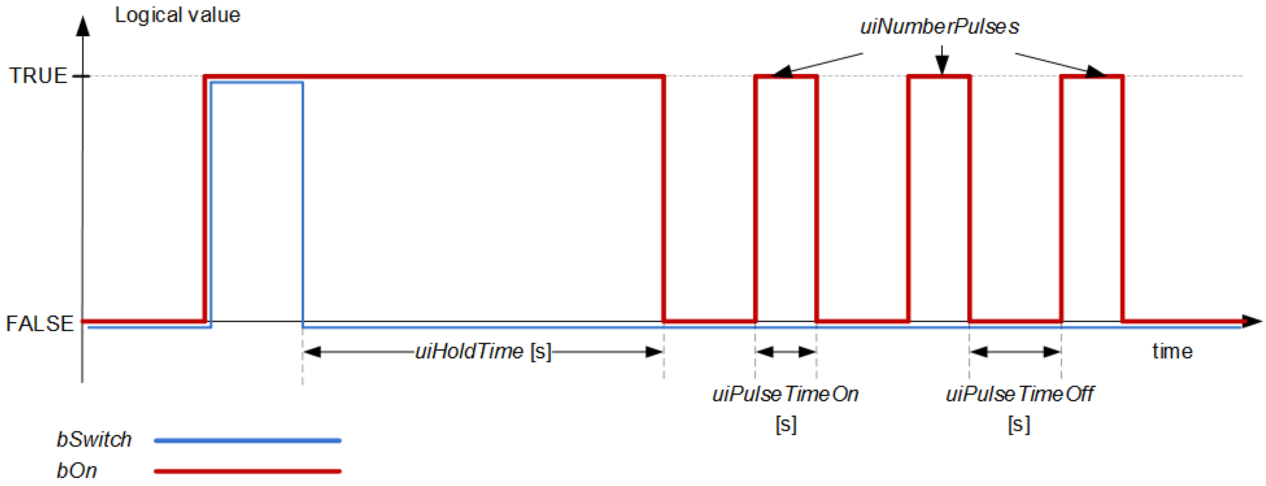


Function block for a stairwell light circuit.

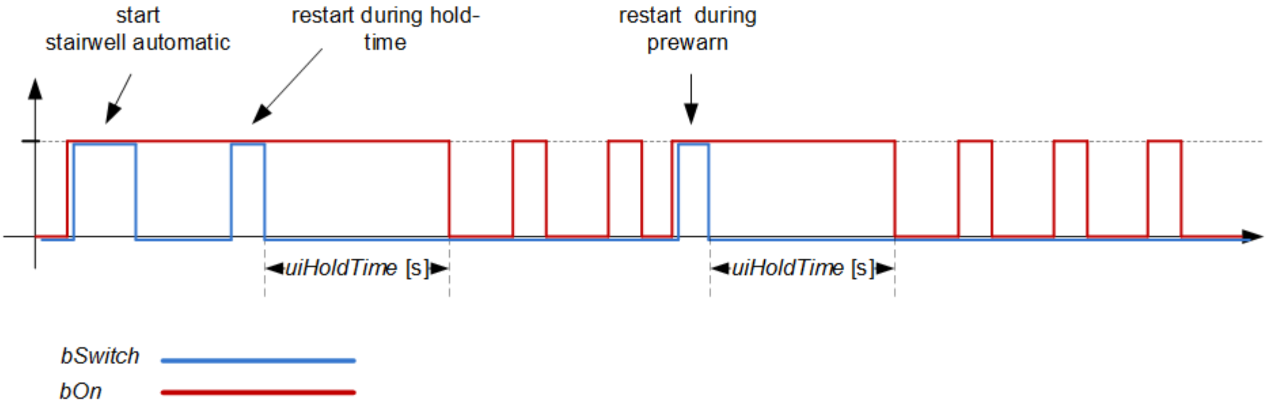
The function block knows 3 different modes, which can be set via the input *uiLightCtrlMode*:

- 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 holding 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:



The *bSwitch* input has no function in the manual modes: the output value *bOn* is set constantly to TRUE if *uiLightCtrlMode*=1 and constantly to FALSE if *uiLightCtrlMode*=2.

Manual mode

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 [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▶ 491]:= eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.

uiLightCtrlMode: Operation mode.

- 0: Automatic mode; the stairwell light 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: Switch output for the lighting.

uiRemainingTimeHold: Remaining holding 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 holding time (*uiHoldTime*), in order to illustrate the countdown of the holding 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 [▶ 318].

VAR_IN_OUT

```
uiHoldTime       : UINT;
uiPulseTimeOn    : UINT;
uiPulseTimeOff   : UINT;
uiNumberPulses   : UINT;
```

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

uiHoldTime: Holding 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.

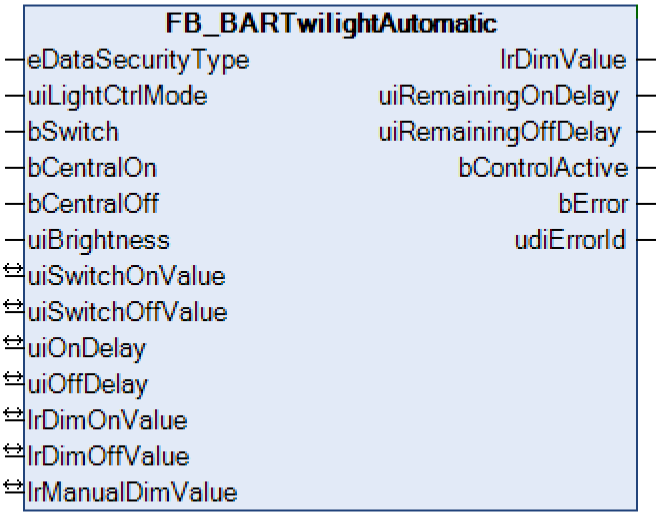
Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

Documents about this

example_persistent.zip (Resources/zip/5090272523.zip)

5.1.5.3.8 FB_BARTwilightAutomatic



Automatic twilight function

The function block knows three 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 automatic twilight function, if it had been inactive beforehand. A further edge on *bSwitch* deactivates the automatic twilight function 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 automatic twilight function 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 operation

The inputs bSwitch, bCentralOn and bCentralOff have no function in the manual operation modes: if *uiLightCtrlMode*=1 the output value *lrDimValue* is set constantly to *lrManualDimValue* and if *uiLightCtrlMode*=2 it is set constantly to 0.0.

Switch-on and switch-off delay

The switch-on and switch-off delays, as described above (“automatic mode”), are always run through irrespective of the state of the automatic function (active or inactive) and the operation mode, i.e. the timers are not reset by these operating states.

VAR_INPUT

```
eDataSecurityType  : E_HVACDataSecurityType;
uiLightCtrlMode    : UINT;
bSwitch            : BOOL;
bCentralOn         : BOOL;
bCentralOff        : BOOL;
uiBrightness       : UINT;
```

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 automatic twilight function on and off.
- 1: Manual On mode; automatic twilight function is inactive – the value *lrManualDimValue* is output without a delay at the output *lrDimValueOut*, all other command inputs are ineffective.
- 2: Manual Off mode; automatic twilight function is inactive – the value 0.0 is output without a delay at the output *lrDimValueOut*, all other command inputs are ineffective.

bSwitch: Positive edges on this input switch the automatic twilight function on and off alternately. When switching off the output *lrDimValue* 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 automatic twilight function on. This command input is active only in automatic mode (*uiLightCtrlMode*=0).

bCentralOff: A positive edge on this input switches the automatic twilight function off and sets the output *lrDimValue* to 0.0. This command input is active only in automatic mode (*uiLightCtrlMode*=0).

uiBrightness: Light actual value [lux].

VAR_OUTPUT

```
lrDimValue           : LREAL;
uiRemainingOnDelay   : UINT;
uiRemainingOffDelay  : UINT;
bControlActive       : BOOL;
bError               : BOOL;
udiErrorId           : UDINT;
```

lrDimValue: 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 automatic twilight function is activated. This is intended to serve as an additional feedback signal if switch-on has taken place but the controller outputs a light value of *lrDimValue*=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 [▸ 318].

> 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

```
uiSwitchOnValue  : UINT;
uiSwitchOffValue : UINT;
uiOnDelay        : UINT;
uiOffDelay       : UINT;
lrDimOnValue     : LREAL;
lrDimOffValue    : LREAL;
lrManualDimValue : LREAL;
```

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

uiSwitchOnValue: Switch-on threshold value. 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 value. 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).

Requirements

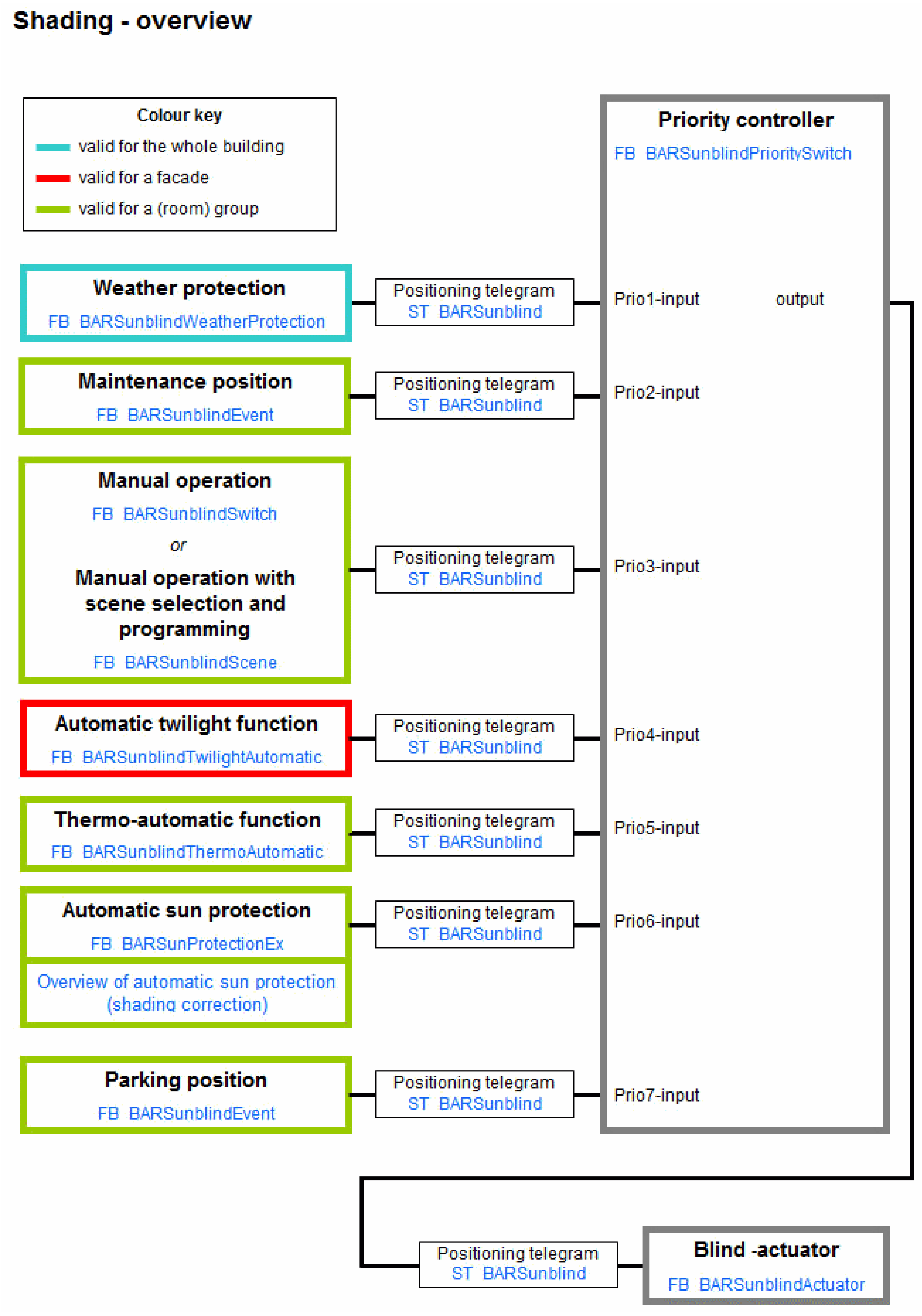
Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

Documents about this

🗎 example_persistent.zip (Resources/zip/5090272523.zip)

5.1.5.4 Sun protection

5.1.5.4.1 Overview shading



Shading - overview

Colour key

- 🟦 valid for the whole building
- 🟥 valid for a facade
- 🟩 valid for a (room) group

Weather protection
FB BARSunblindWeatherProtection

Maintenance position
FB BARSunblindEvent

Manual operation
FB BARSunblindSwitch
or
Manual operation with scene selection and programming
FB BARSunblindScene

Automatic twilight function
FB BARSunblindTwilightAutomatic

Thermo-automatic function
FB BARSunblindThermoAutomatic

Automatic sun protection
FB BARSunProtectionEx
Overview of automatic sun protection (shading correction)

Parking position
FB BARSunblindEvent

Positioning telegram
ST BARSunblind

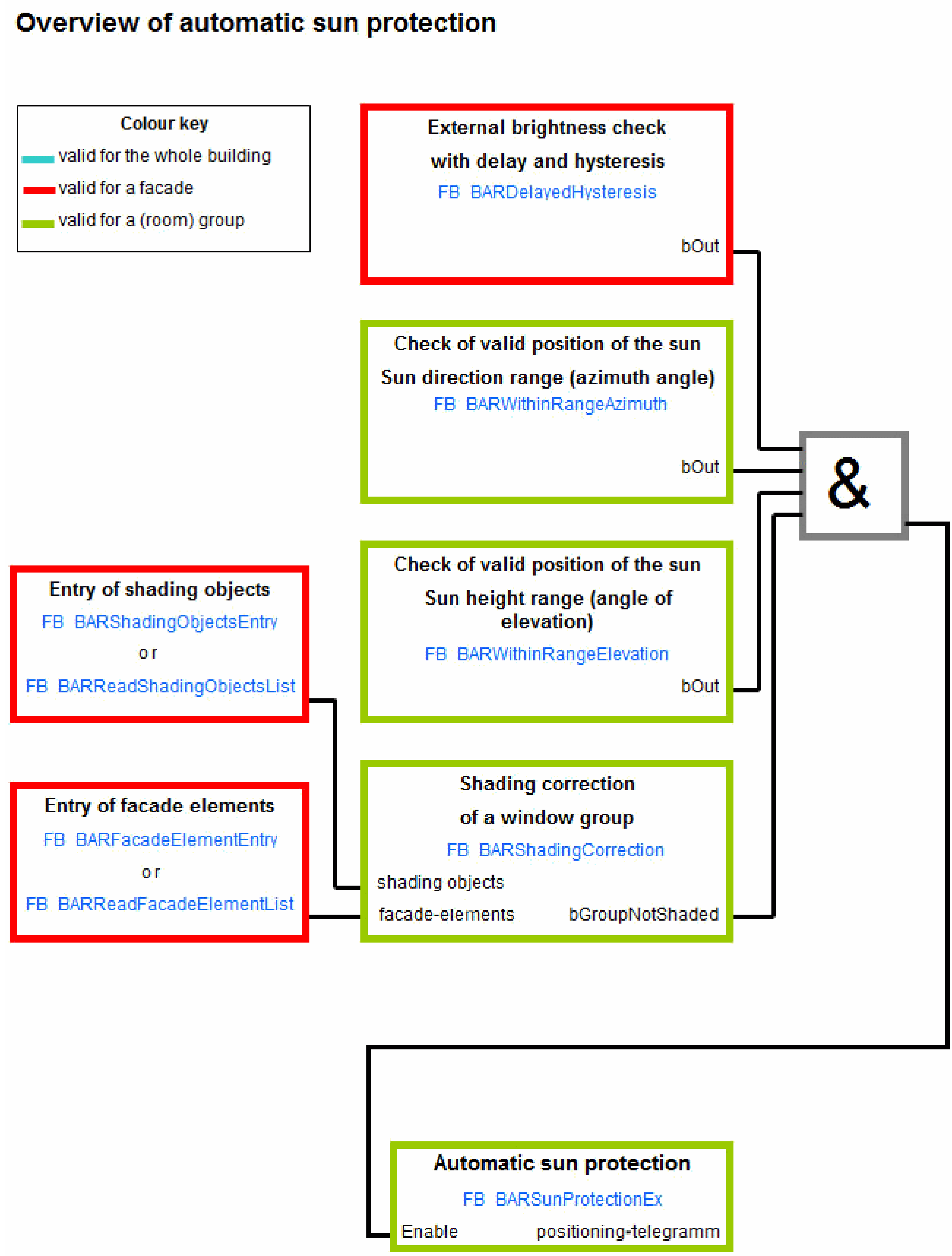
Priority controller
FB BARSunblindPrioritySwitch

Prio1-input
Prio2-input
Prio3-input
Prio4-input
Prio5-input
Prio6-input
Prio7-input
output

Positioning telegram
ST BARSunblind

Blind -actuator
FB BARSunblindActuator

5.1.5.4.2 Overview sun protection



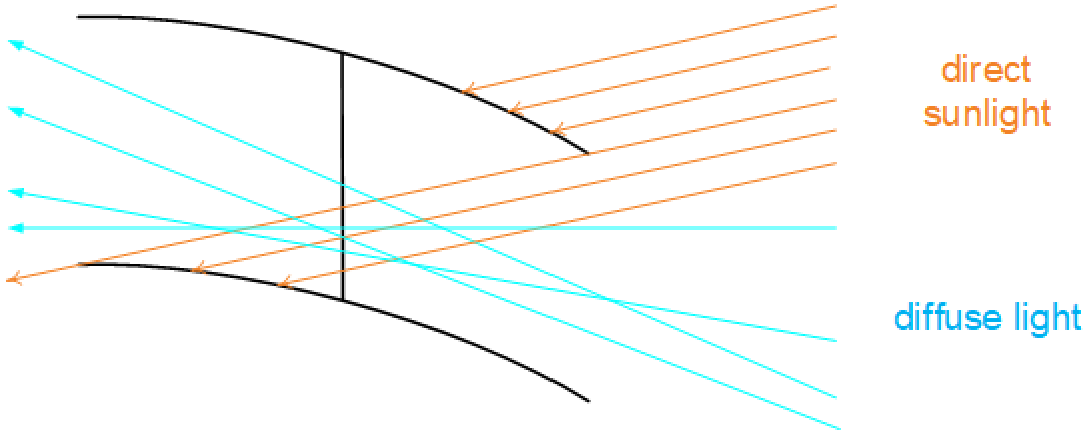
5.1.5.4.3 Sun protection: Basic principles and definitions

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:

- Slat adjustment
- Height adjustment

Lamella setpoint tracing

A blind with lamellas that can be adjusted offers the option of intelligent sun protection here. The position of the lamellas 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 lamella angle:

- the current sun elevation (elevation angle)
- the sun position, i.e. the azimuth angle
- the facade orientation
- the lamella width
- the lamella spacing

Effective elevation angle

If the blind is viewed in section as above, the angle of incidence does not depend solely on the sun elevation, but also on the direction of the sun:

- If the facade orientation and the sun position (azimuth) are the same, i.e. the sunlight falls directly onto the facade, the effective light incidence angle is the same as the current elevation angle.
- 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 elevation angle.

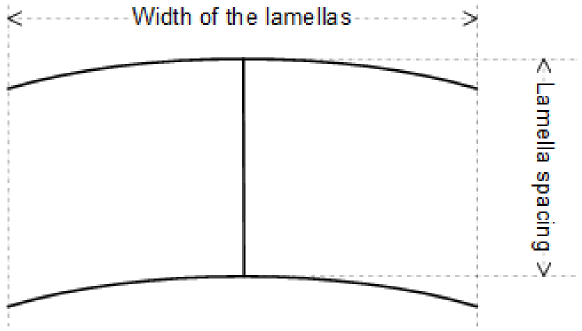
This relationship can easily be illustrated with a set square positioned upright on the table: Viewed directly from the side you can see a triangle with two 45° angles and one 90° angle. If the triangle is rotated, the side on the table appears to become shorter and the two original 45° angles change. The triangle appears to be getting steeper.

We therefore refer to the "effective elevation angle", which describes the proportion of light that falls directly onto the blind.

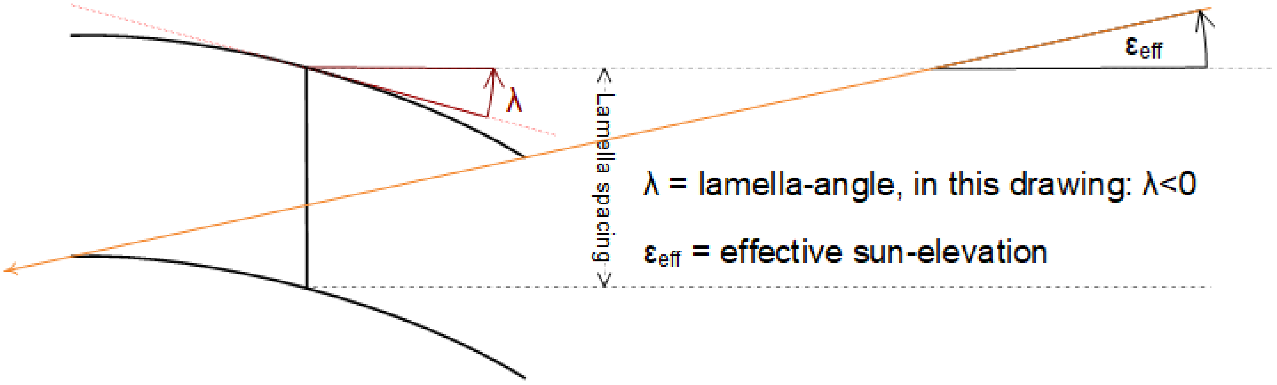
The following three images illustrate the relationship between the effective elevation angle and the blind dimensions, and how the resulting lamella angle λ changes during the day:

Lamella-angle

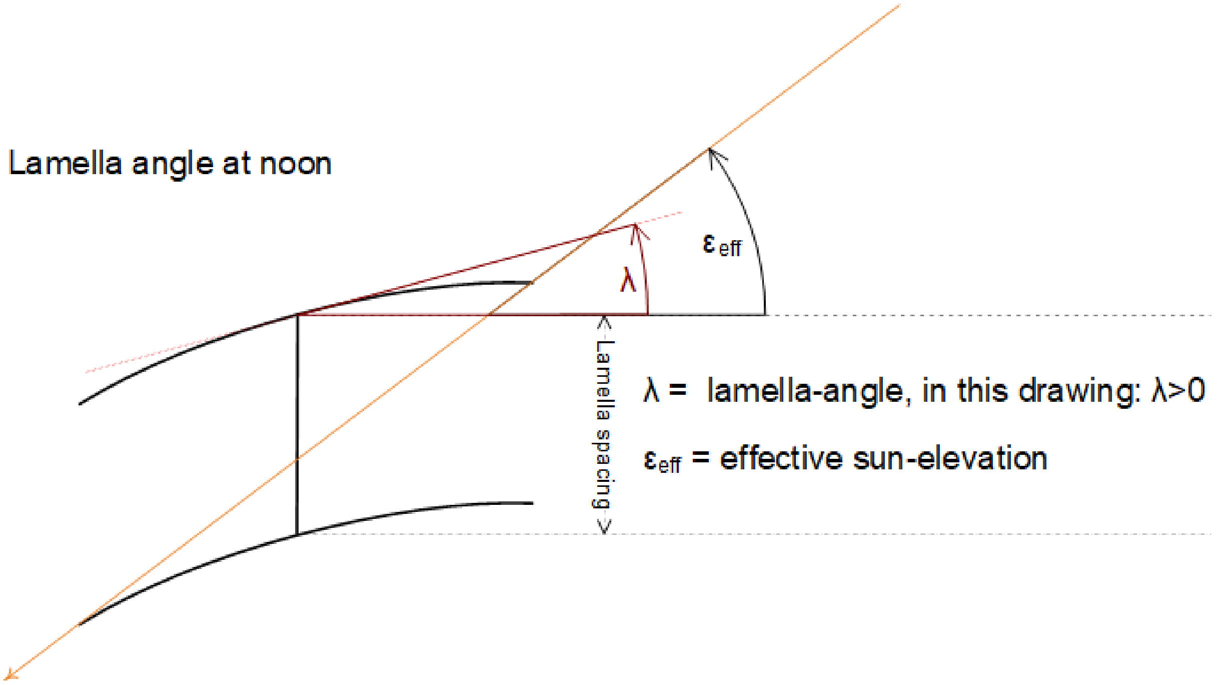
Lamella at an angle of λ=0



Lamella-angle in the morning and in the evening


ε_eff
λ
Lamella spacing
λ = lamella-angle, in this drawing: λ<0
ε_eff = effective sun-elevation

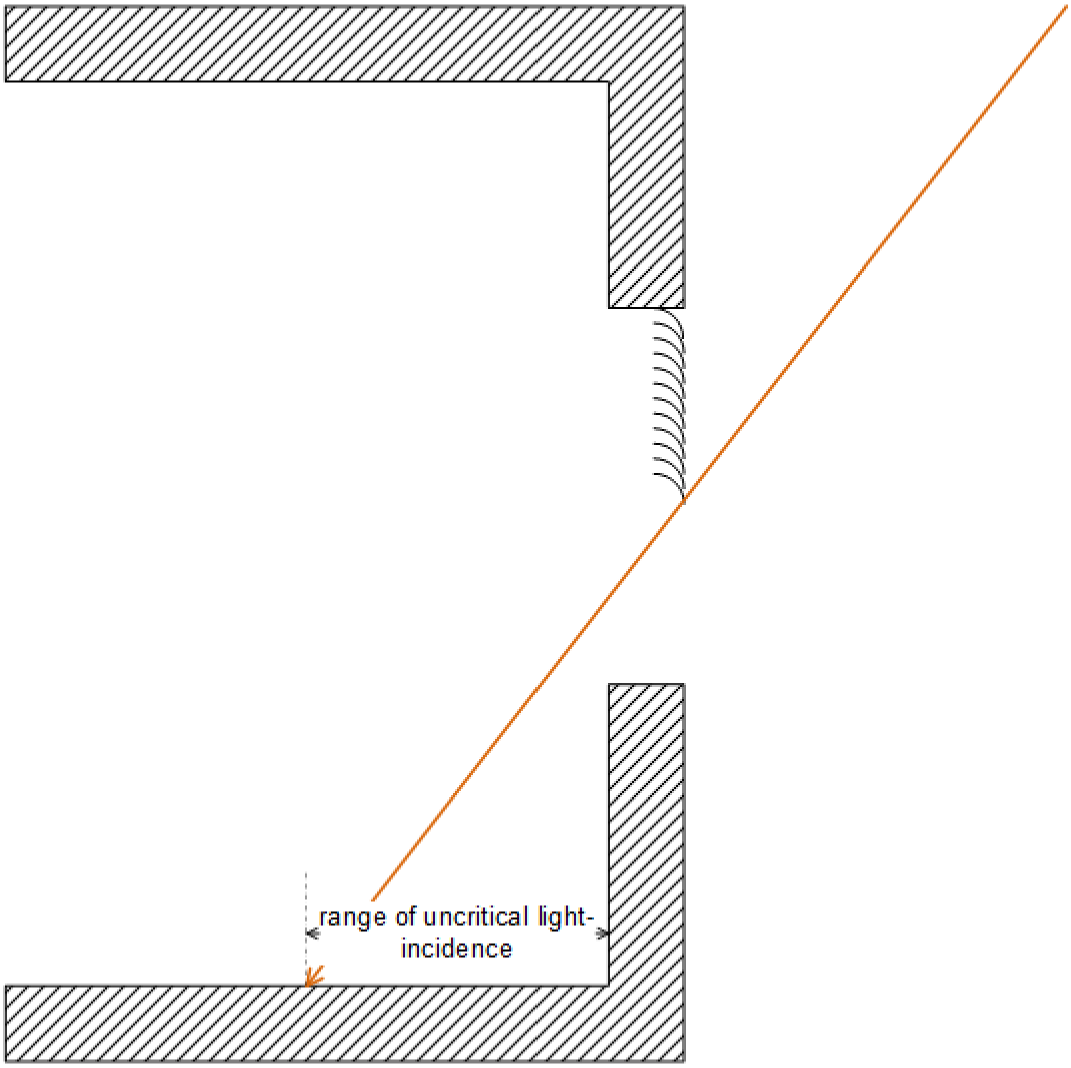
Lamella angle at noon


ε eff
λ
Lamella spacing
λ = lamella-angle, in this drawing: λ>0
ε_eff = effective sun-elevation

More detailed information on this topic can be found in the chapter Effective elevation angle [▶ 239].

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.

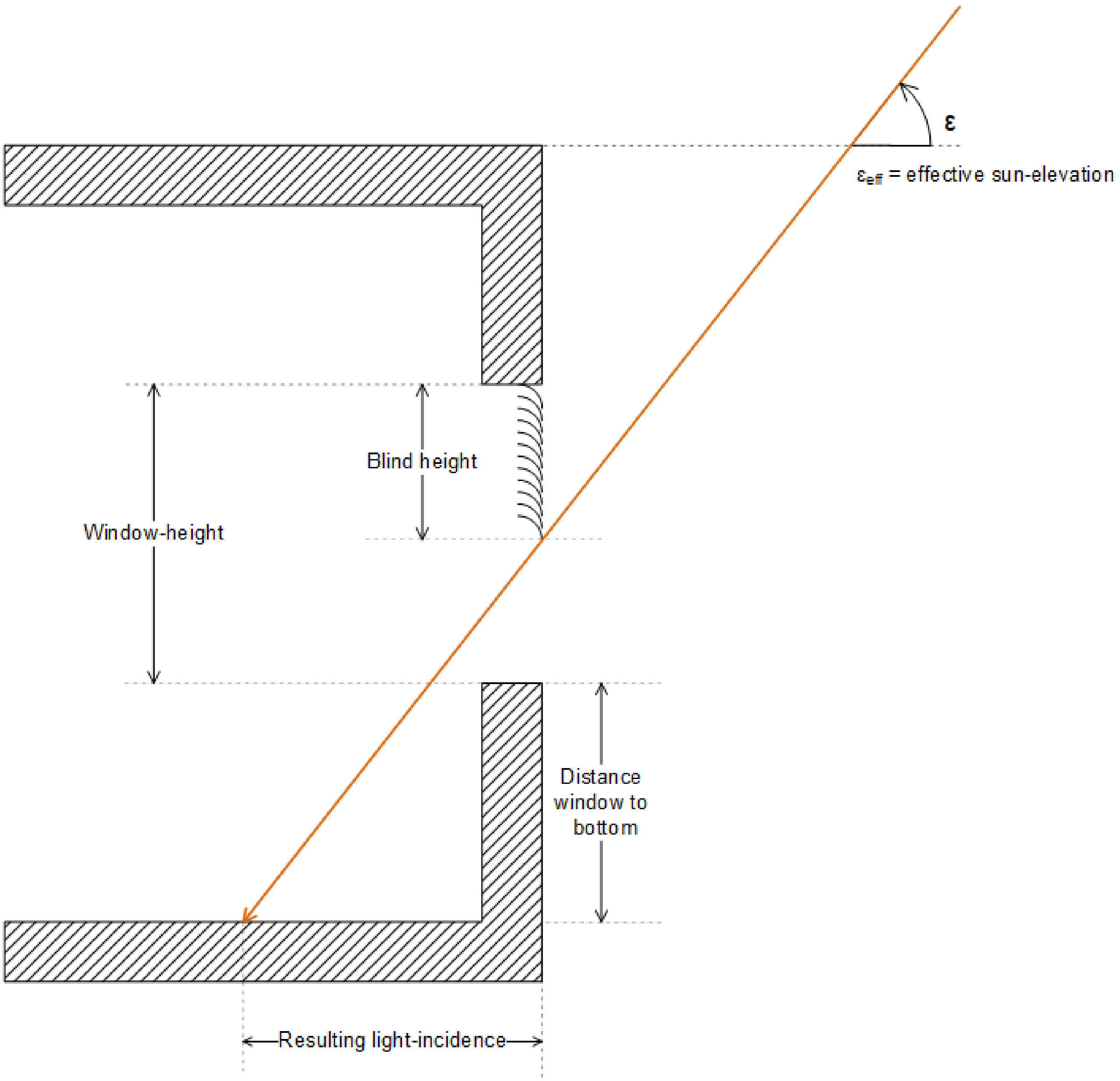


In order to be able to calculate at any time the appropriate blind height that guarantees that the incidence of sunlight does not exceed a certain value, the following values are necessary.

Required for the calculation of the respective blind height:

- Height of the sun (elevation)
- Window height
- Distance between the window and the floor

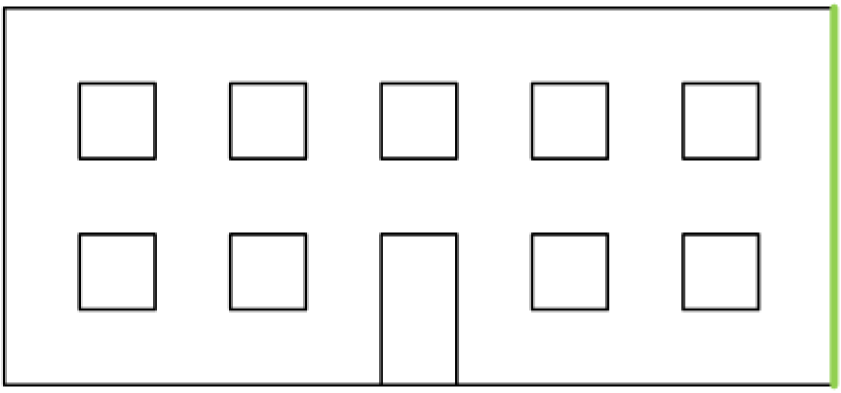
The following illustration shows where these parameters are to be classified:


ε
ε_{eff} = effective sun-elevation
Blind height
Window-height
Distance window to bottom
Resulting light-incidence

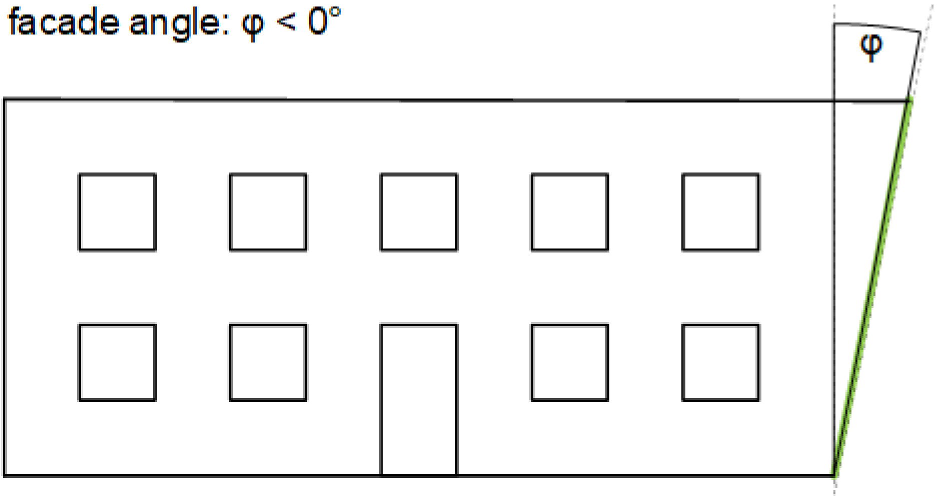
Influence of the facade inclination

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:

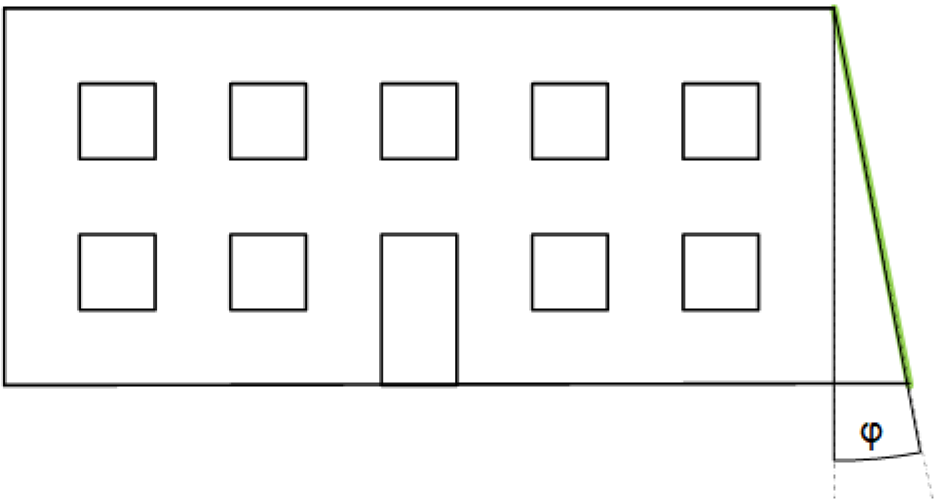
facade angle: φ = 0°



facade angle: φ < 0°

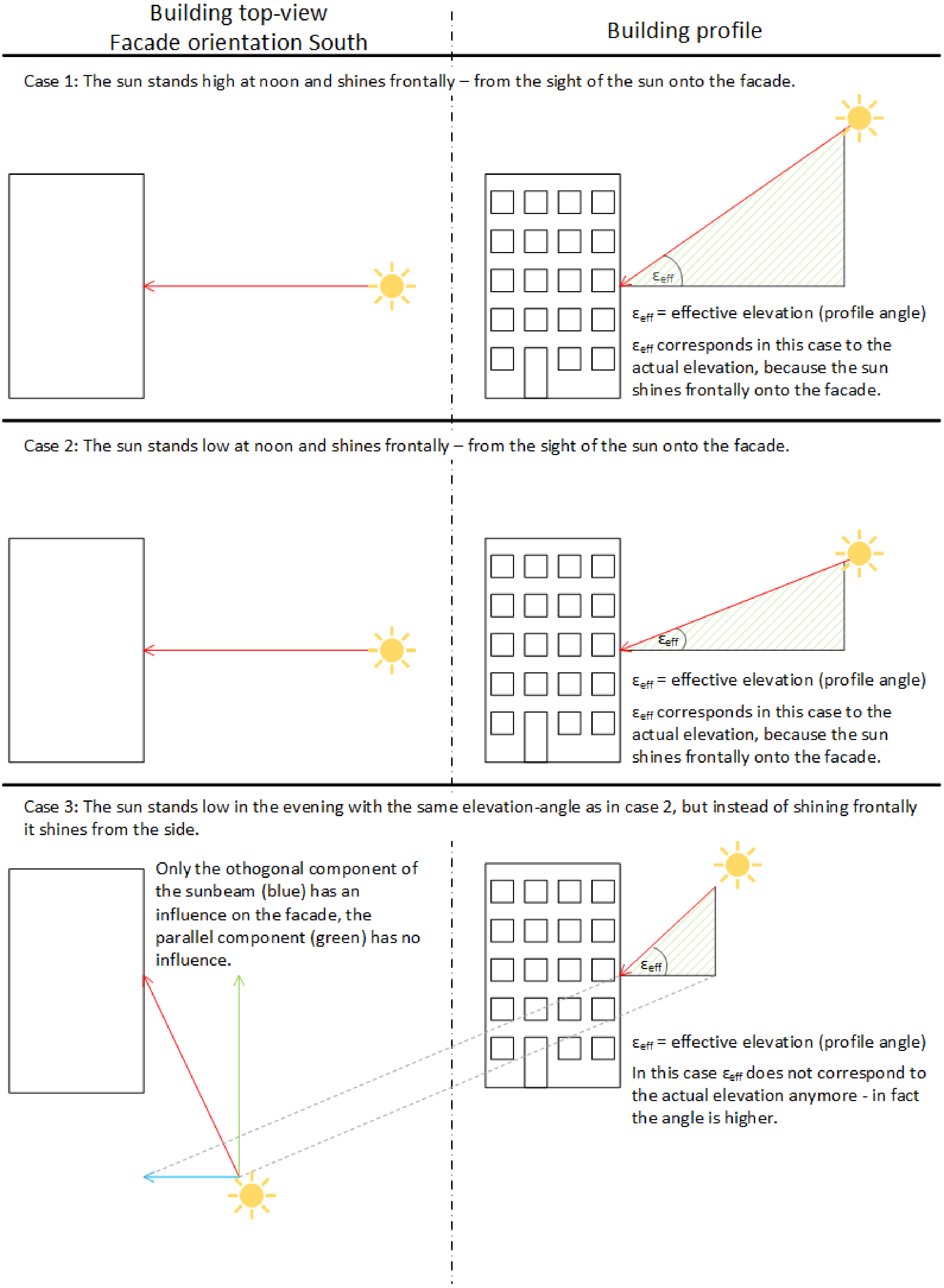


facade angle: φ > 0°



5.1.5.4.3.1 Effective elevation angle

This chapter describes the relationship between the incidence of sunlight on the facade and the elevation angle.



Case 1: The sun stands high at noon and shines frontally – from the sight of the sun onto the facade.

ε_{eff} = effective elevation (profile angle)

ε_{eff} corresponds in this case to the actual elevation, because the sun shines frontally onto the facade.

Case 2: The sun stands low at noon and shines frontally – from the sight of the sun onto the facade.

ε_{eff} = effective elevation (profile angle)

ε_{eff} corresponds in this case to the actual elevation, because the sun shines frontally onto the facade.

Case 3: The sun stands low in the evening with the same elevation-angle as in case 2, but instead of shining frontally it shines from the side.

Only the othogonal component of the sunbeam (blue) has an influence on the facade, the parallel component (green) has no influence.

ε_{eff} = effective elevation (profile angle)

In this case ε_{eff} does not correspond to the actual elevation anymore - in fact the angle is higher.

Case 1 shows a typical incidence of sunlight on a south-facing facade at midday.

The sun is high compared to the morning and evening.

The green shaded triangle represents the elevation angle ε.

Seen from the direction of the sun (azimuth), only the orthogonal component has a direct effect on the facade!

In this case, the elevation angle ε is therefore equal to the effective elevation angle ε_{eff} or the profile angle for the calculation.

Case 2 shows a typical incidence of sunlight at midday on a south-facing facade, but lower.

The green shaded triangle represents the elevation angle ε, which is smaller compared to case 1.

Here too, the elevation angle ε is equal to the effective elevation angle ε_{eff} or the profile angle for the calculation.

Case 3 shows a typical incidence of sunlight on a south-facing facade in the evening. The elevation angle is the same as in case 2, so case 3 is the continuation of case 1 on the same day.

The green shaded triangle from case 2 (building in profile) is now tilted forwards.

However, it represents the orthogonal lighting component. The elevation angle for this component is visibly larger than in case 2, where the sun shines orthogonally onto the facade.

5.1.5.4.4 Shading correction: basic principles and definitions

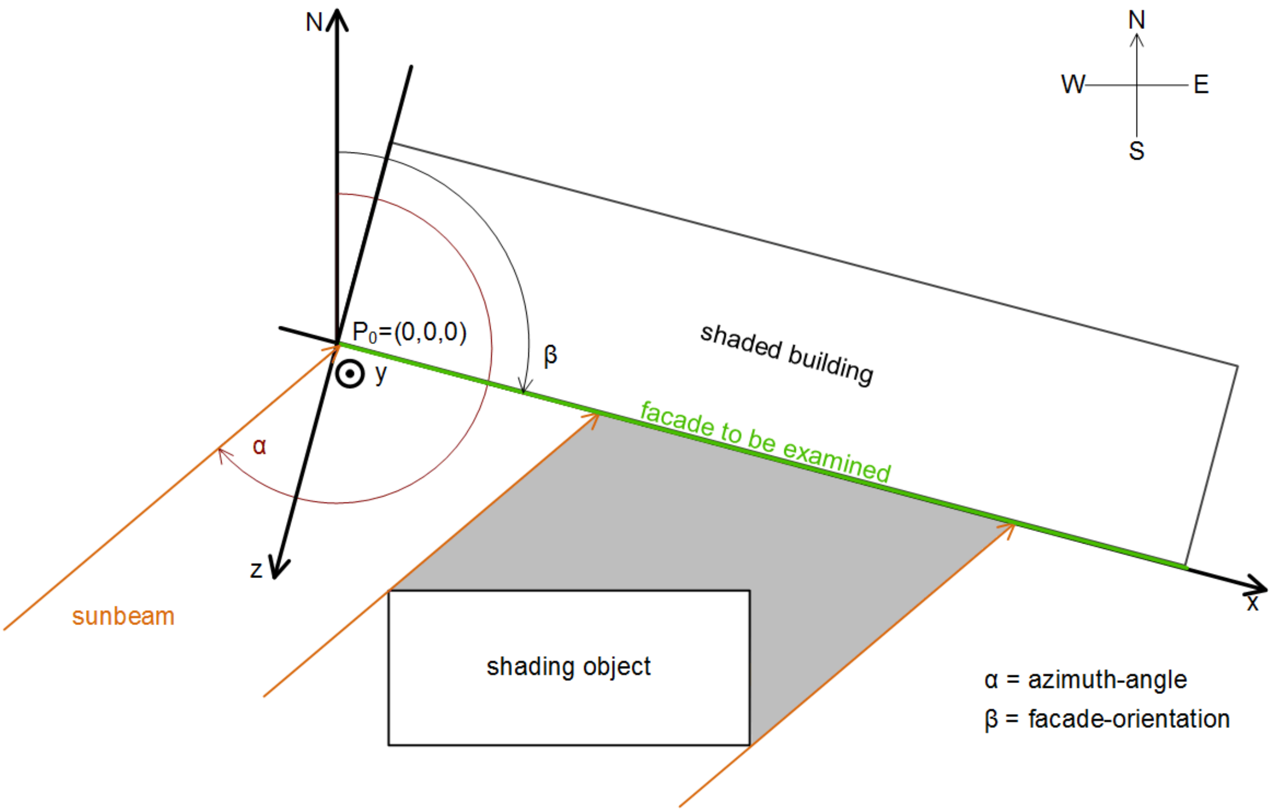
The shading correction can be used in conjunction with the automatic sun function or louvre adjustment. The function checks whether a window or a window group that is assigned to a room, for example, is temporarily placed in the shade by surrounding buildings or parts of its own building. Sun shading 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



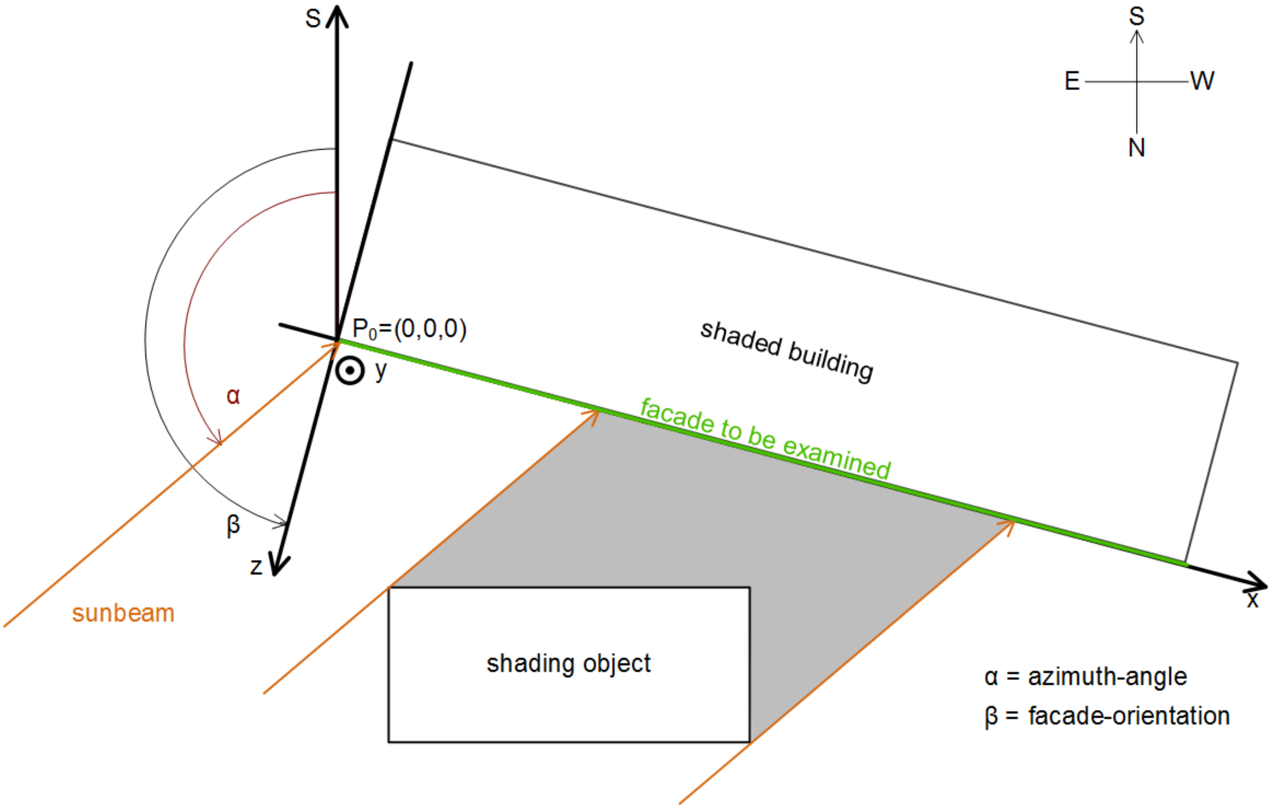
For the pure observation of the shadow thrown on the facade, a two-dimensional coordinate system is ultimately required, therefore the x and y axis were placed on the facade. The zero point is thereby at the bottom left on the base, as if one were regarding the facade from the front. For the calculation 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.

The horizontal position of the sun (azimuth angle)in the northern hemisphere is by definition measured from the north. 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	β=0°
East	β=90°
South	β=180°
West	β=270°

The course of the sun in the southern hemisphere is different. The sun rises in the east as well. but at noon it is located in the north. Thus the definition of the facade-orientation is changed as well:

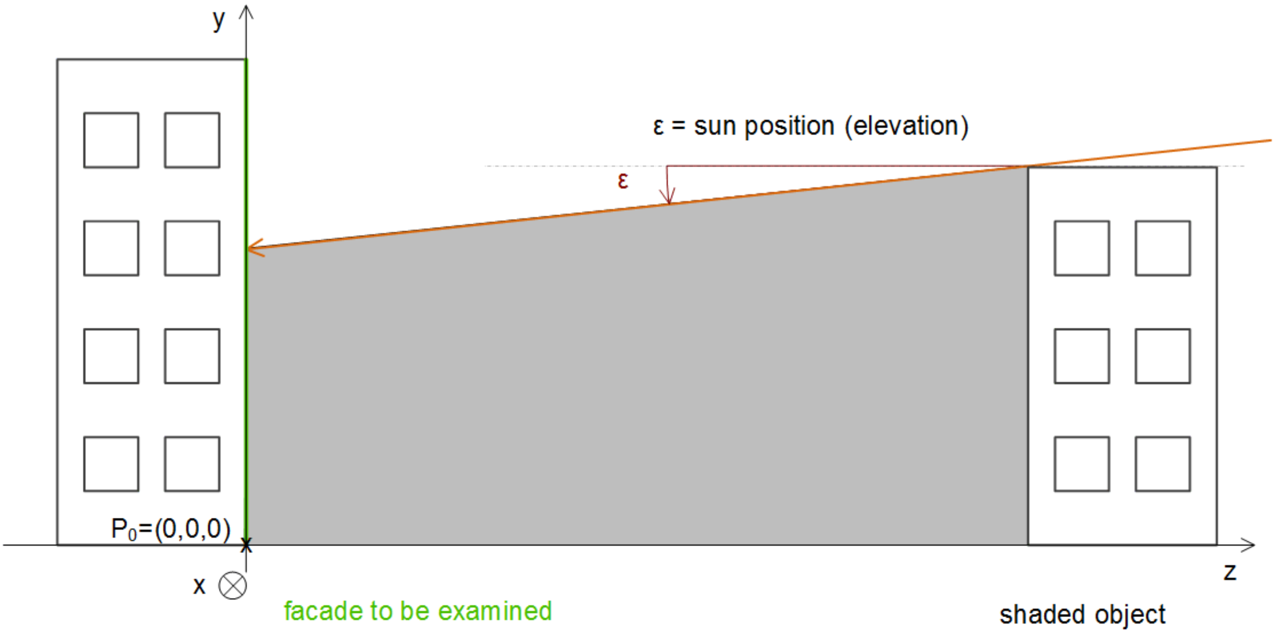
Line of sight	Facade orientation
South	β=0°
East	β=90°
North	β=180°
West	β=270°



All further explanations in this chapter are referring to the characteristics of the northern hemisphere. For the the subsequent parameterization of the shading-correction (FB_BARShadingCorrection [▸ 264] / FB_BARShadingCorrectionSouth [▸ 267]) only the facade-orientation is important, which can be taken from the above mentioned tables for either the northern or southern hemisphere.

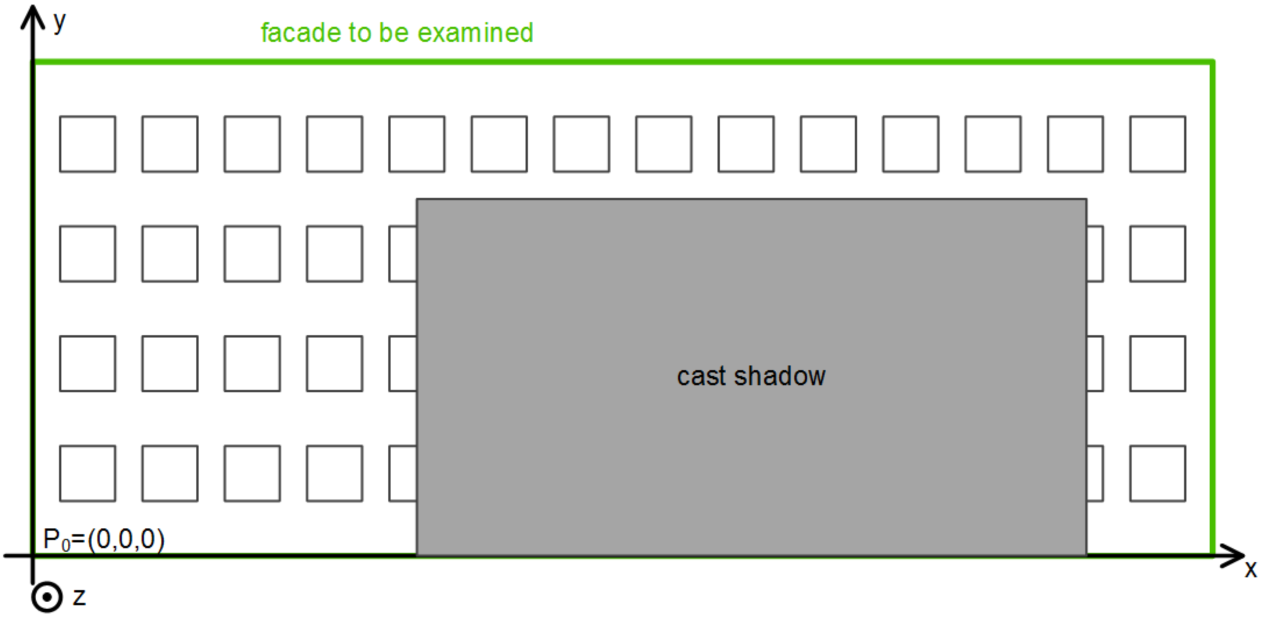
The following two illustrations are intended to further clarify the position of the point of origin P_0 as well as the orientation of the coordinate system:

Observation from the side



The angle of elevation (height of the sun) 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.

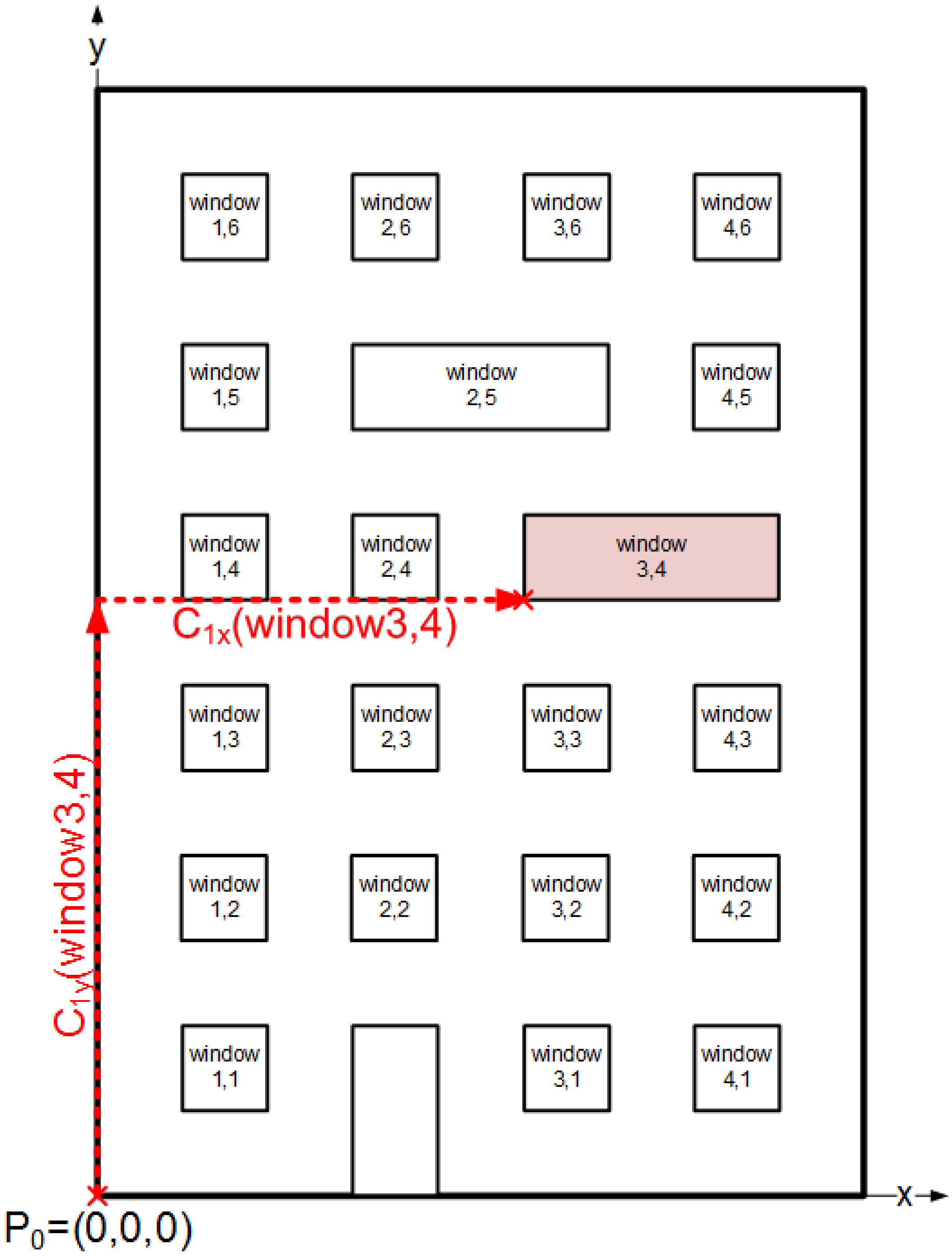
Observation from the front



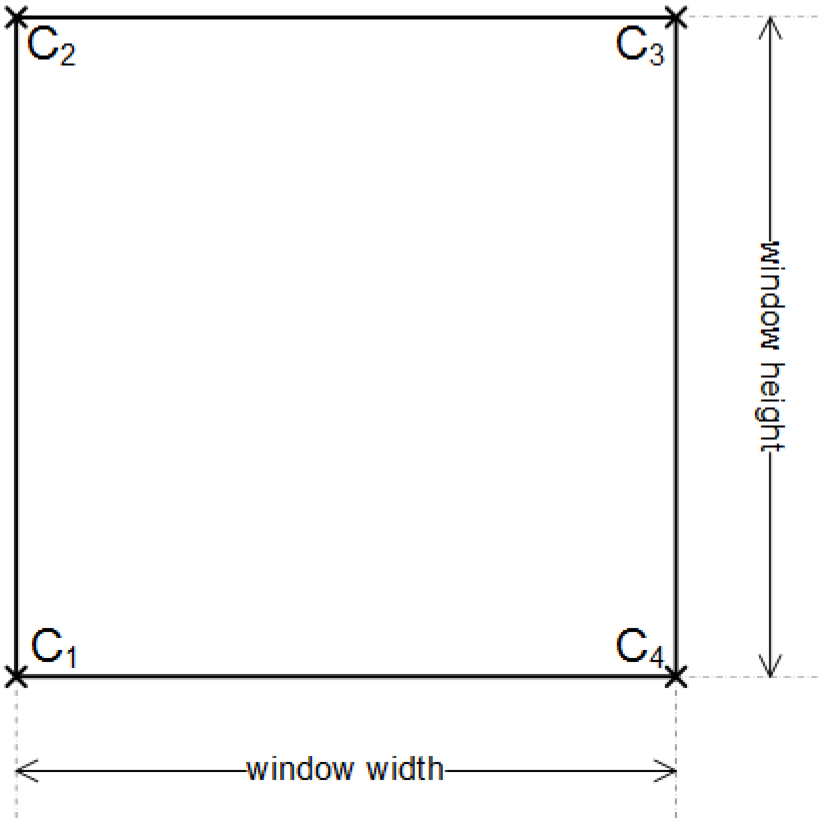
Here, the position of the point of origin, P_0, at the bottom left base point of the facade is once more very clear. Beyond that the X-Y orientation is illustrated, which is important later for the entry of the window elements.

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.



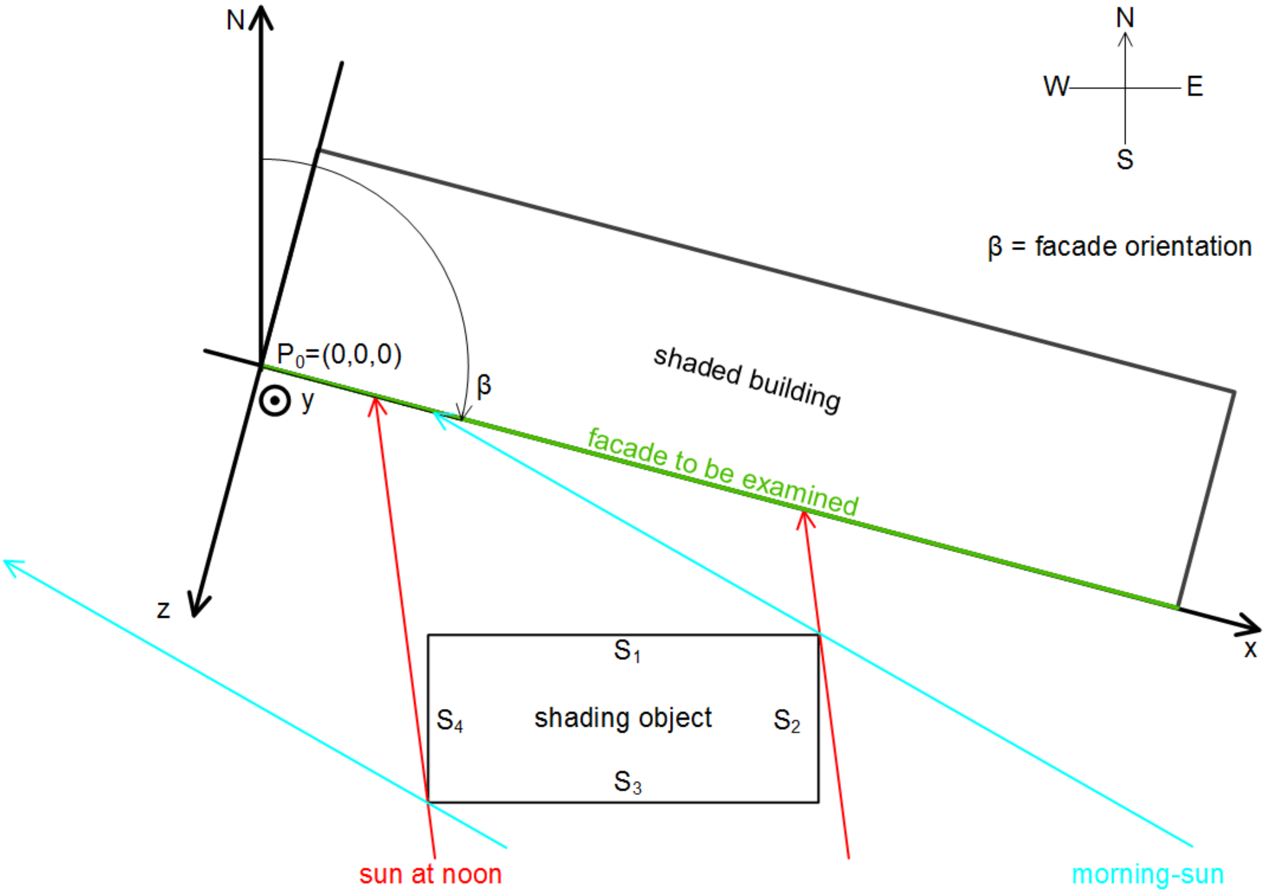
C_n=comer n

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

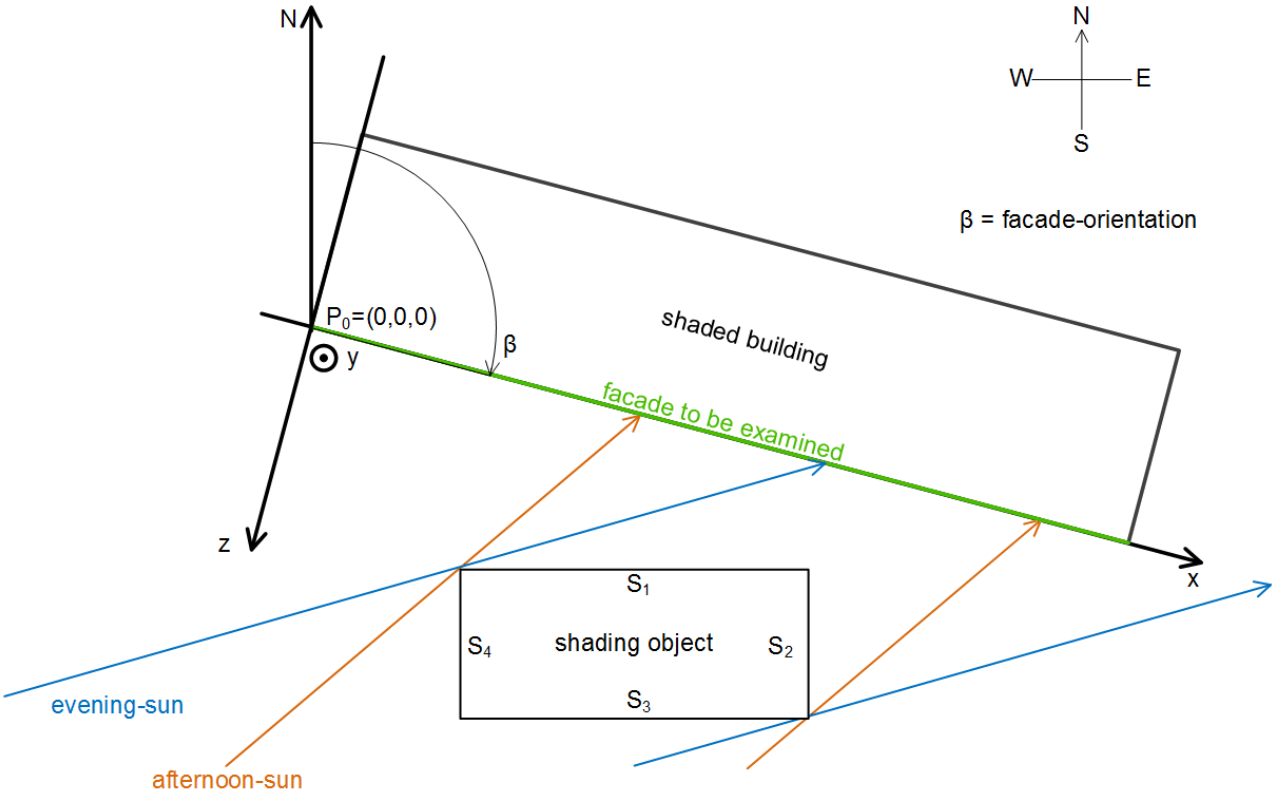
When describing the shading objects, distinction is made between angular objects (building, column) and objects that are approximately spherical (e.g. trees). Angular objects can be categorised according to the shadow they cast into square, shadow-casting facades, wherein one must consider which ones cast the main shadows over the course of the day:

Morning/midday

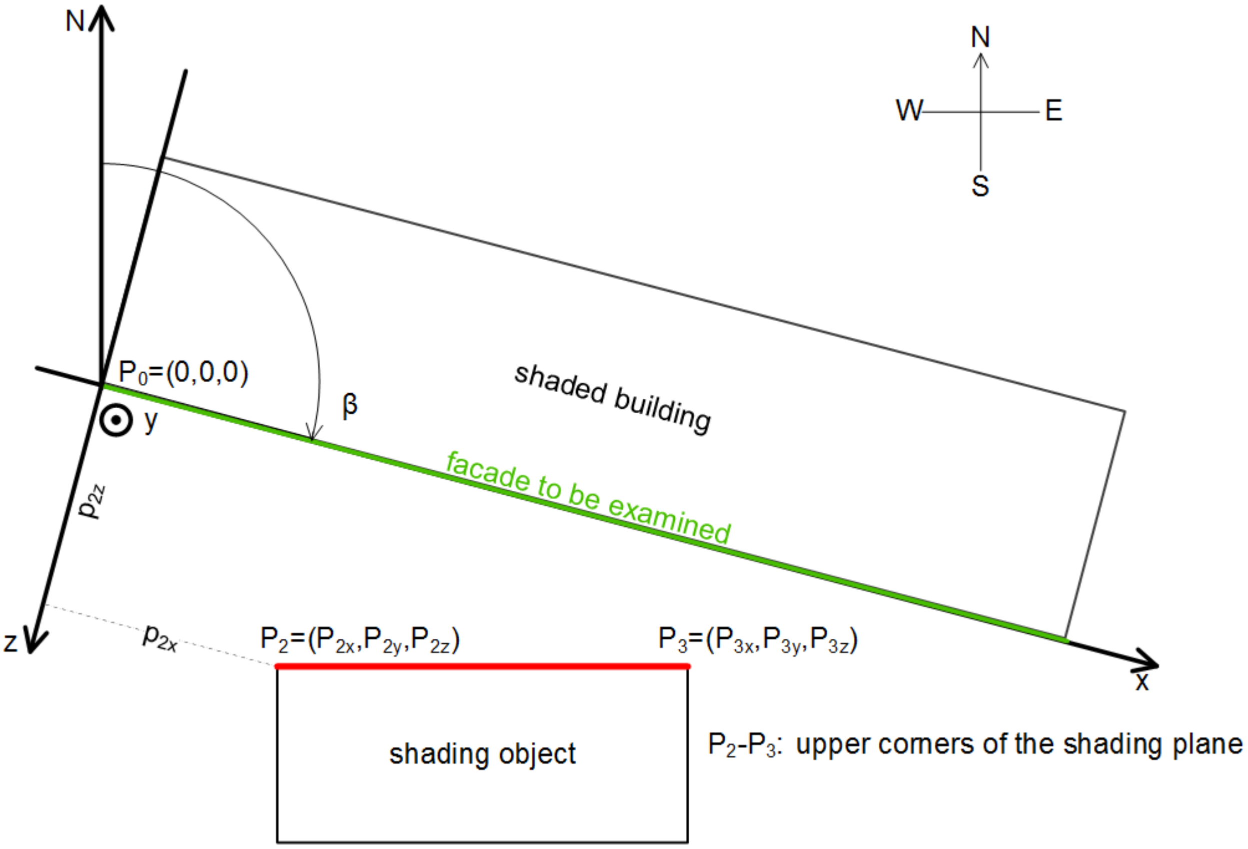


In the morning and at midday, the shadows are mainly cast by the sides S_1 and S_4; S_2 und S_3 need not be considered if they are not higher.

Afternoon / evening



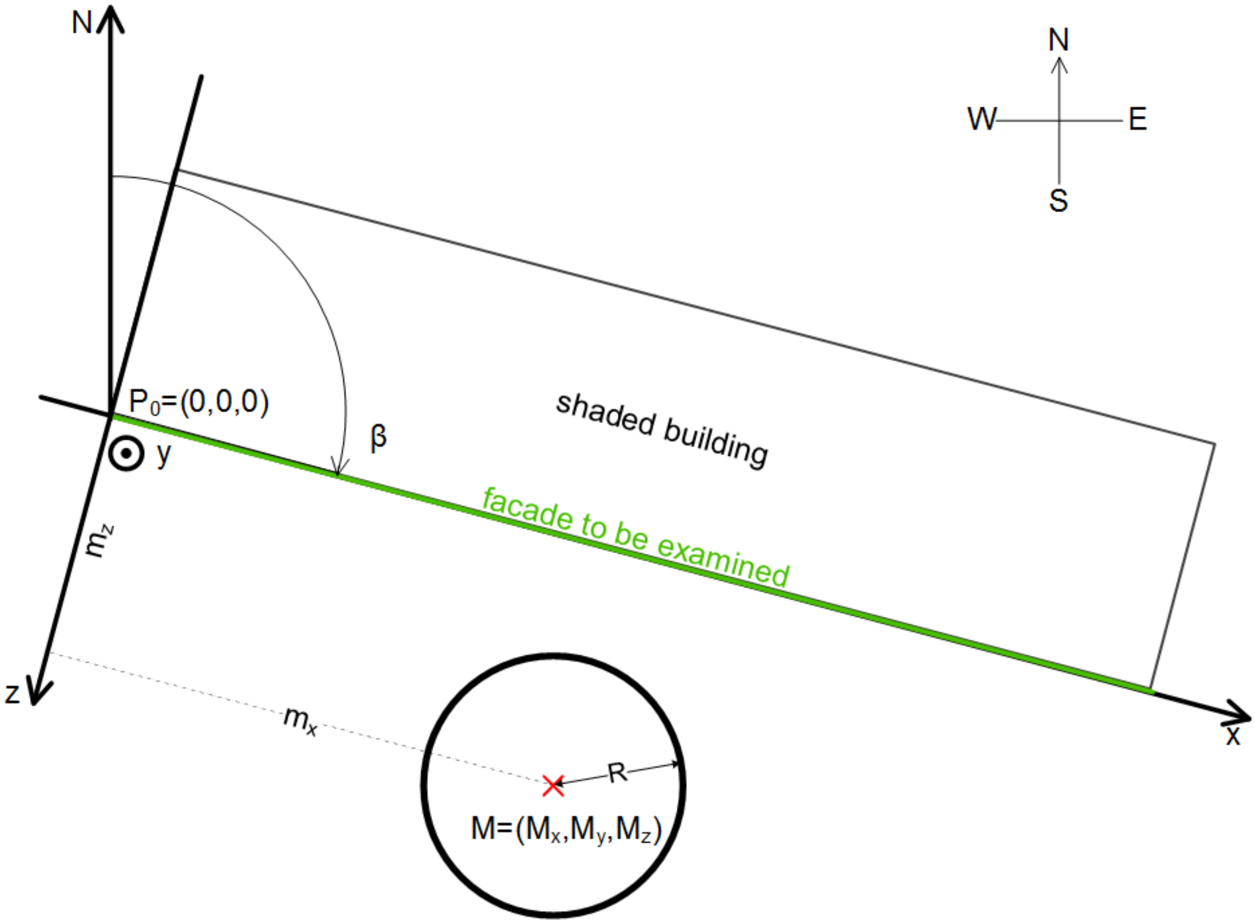
In the afternoon and in the evening too, the total shade can be determined alone by the observation of S_1 and S_2. In this case, therefore, it is sufficient to specify S_1 and S_2 as shadow-casters. The entry is made on the basis of the four corners or their coordinates in relation to the zero point 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.

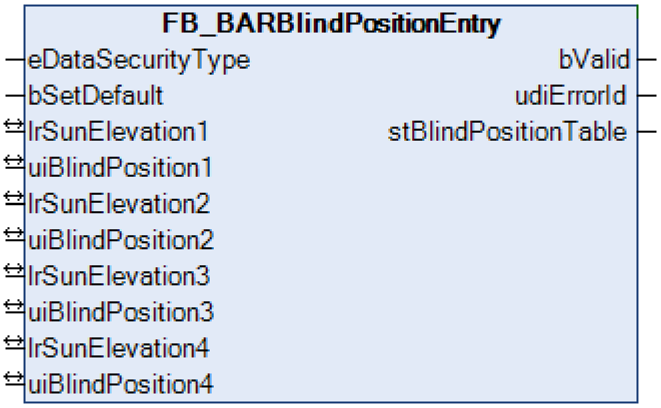
The input of shadow-casting ball elements is done by entering the centre of the ball and its radius:

Ball elements



A "classification" of the ball element as in the case of the angular building is of course unnecessary, since the shadow cast by a ball changes only its direction, but not its size.

5.1.5.4.5 FB_BARBlindPositionEntry



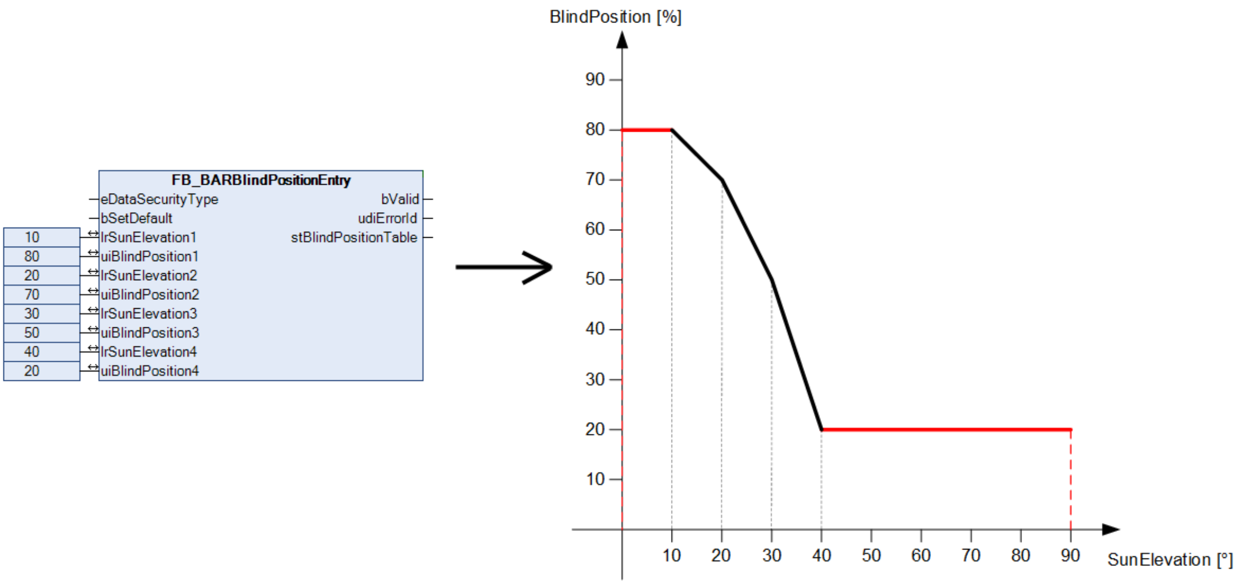
```
FB_BARBlindPositionEntry
eDataSecurityType                bValid
bSetDefault                      udiErrorId
lrSunElevation1                  stBlindPositionTable
uiBlindPosition1
lrSunElevation2
uiBlindPosition2
lrSunElevation3
uiBlindPosition3
lrSunElevation4
uiBlindPosition4
```

This function block serves for the input of interpolation points for the function block FB_BARSunProtectionEx [▶ 306], if this should be operated in the height positioning mode with the help of a table, see E_BARPosMode [▶ 501].

In addition to the operation modes "Fixed shutter height" and "Maximum light incidence", the function block FB_BARSunProtectionEx [▸ 306] 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 shutter height relative to the respective sun position is calculated by linear interpolation. However, since incorrectly entered values can lead to malfunctions in FB_BARSunProtectionEx [▸ 306] , 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 selected values for *rSunElevation1* .. *rSunElevation4* must also be unique; for example, they may not be:
[*rSunElevation1* = 10; *uiBlindPosition1* = 50] and at the same time [*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.
On top of that the entries for the position of the sun and shutter 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 ≤*rSunElevation*≤ 90 (in degrees - range of validity of source values)
- 0 ≤*uiBlindPosition*≤ 100 (in percent - range of validity of target values)

The function block checks the values entered for these conditions and outputs an error code [▸ 318] if they are not satisfied. In addition, the output *bValid* is set to FALSE.
Furthermore the function block independently ensures that the boundary areas are filled out: Internally a further interpolation point is setup at *rSunElevation* = 0 with *uiBlindPosition1* and a further one above *rSunElevation4* at *rSunElevation* = 90 with *uiBlindPosition4*. This ensures that a sensible target value exists for all valid input values 0 ≤*rSunElevation*≤ 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 [▸ 306] thus increases to 6, see ST_BARBlindPositionTable [▸ 502].

The interpolation of the values takes place in the glare protection block.

VAR_INPUT

```
eDataSecurityType   : E_HVACDataSecurityType;
bSetDefault         : BOOL;
```

eDataSecurityType: If eDataSecurityType [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 [▶ 318].

stBlindPositionTable : Transfer structure of the interpolation points, see ST_BARBlindPositionTable [▶ 502].

VAR_IN_OUT

```
lrSunElevation1   : REAL;;
uiBlindPosition1 : UINT;
lrSunElevation2   : REAL;
uiBlindPosition2 : UINT;
lrSunElevation3   : REAL;
uiBlindPosition3 : UINT;
lrSunElevation4   : REAL;
uiBlindPosition4 : UINT;
```

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

lrSunElevation1: Sun position at the first interpolation point (0°..90°).

uiBlindPosition1: Blind position (degree of closure) at the first interpolation point (0%..100%).

lrSunElevation2: Sun position at the second interpolation point (0°..90°).

uiBlindPosition2: Blind position (degree of closure) at the second interpolation point (0%..100%).

lrSunElevation3: Sun position at the third interpolation point (0°..90°).

uiBlindPosition3: Blind position (degree of closure) at the third interpolation point (0%..100%).

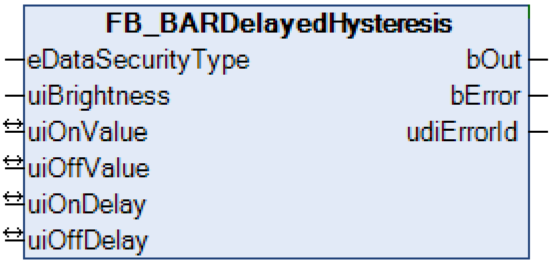
lrSunElevation4: Sun position at the fourth interpolation point (0°..90°).

uiBlindPosition4: Blind position (degree of closure) at the fourth interpolation point (0%..100%).

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.6 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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.

uiBrightness: External 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 [▶ 318].

VAR_IN_OUT

```
uiOnValue  : UINT;
uiOffValue : UINT;
uiOnDelay  : UINT;
uiOnDelay  : UINT;
```

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

uiOnValue: Switch-on threshold value in lux. This must be larger than the switch-off threshold value *uiOffValue*.

uiOffValue: Switch-off threshold value in lux. This must be smaller than the switch-on threshold value *uiOnValue*.

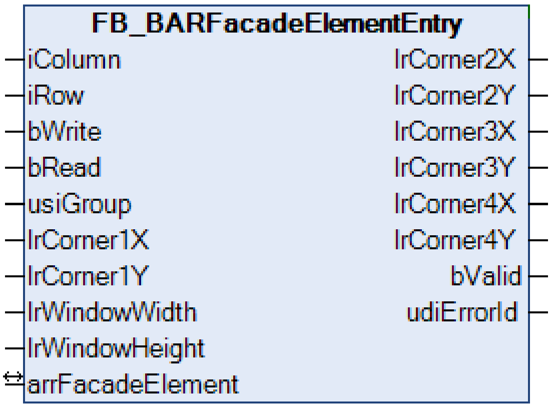
uiOnDelay: Switch-on delay in seconds.

uiOffDelay: Switch-off delay in seconds.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.7 FB_BARFacadeElementEntry

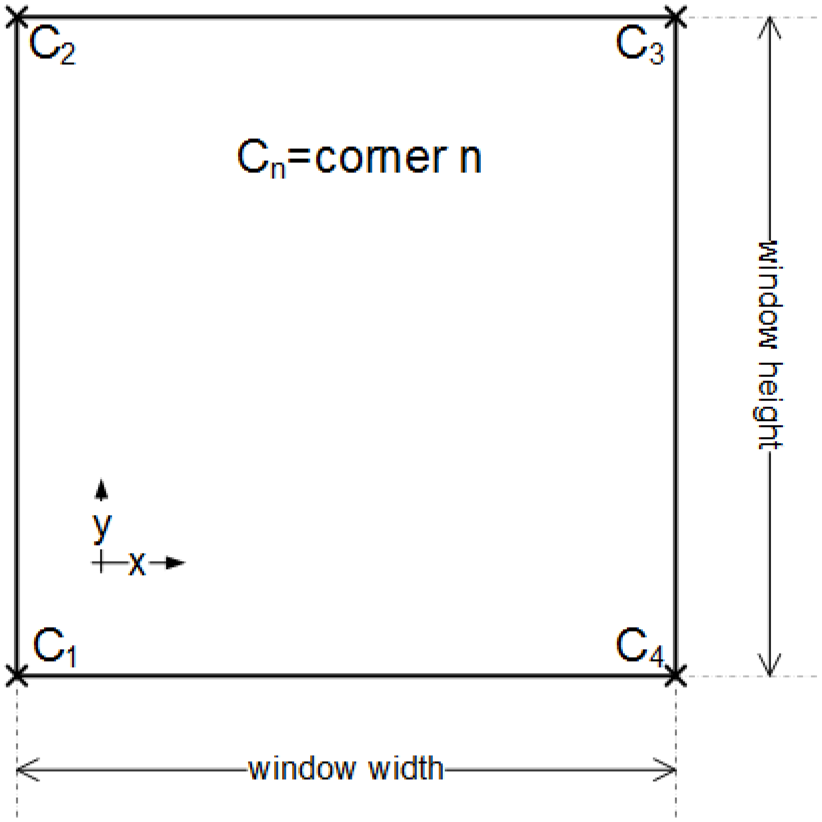


This function block serves the administration of all facade elements (windows) in a facade, which are saved globally in a list of facade elements [▸ 316]. It is intended to facilitate the input of the element information - also with regard to the use of the target visualization. A schematic illustration of the objects with description of the coordinates is given in Shading correction: principles and definitions [▸ 240].
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 [▸ 502]). The information includes the group affiliation, the dimensions (width, height) and the coordinates of the corners. The function block thereby accesses this field directly via the IN-OUT variable *arrFacadeElement*.

Note: 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 metres
- lrCorner1Y Y-coordinate of corner 1 in metres
- lrWindowWidth Window width in metres
- lrWindowHeight Window height in metres

These are then assigned to the corresponding input variables of the 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 visualisation.

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 [▶ 318] is output at the output *udiErrorId*. See also below "Errors (bValid=FALSE)".

Write

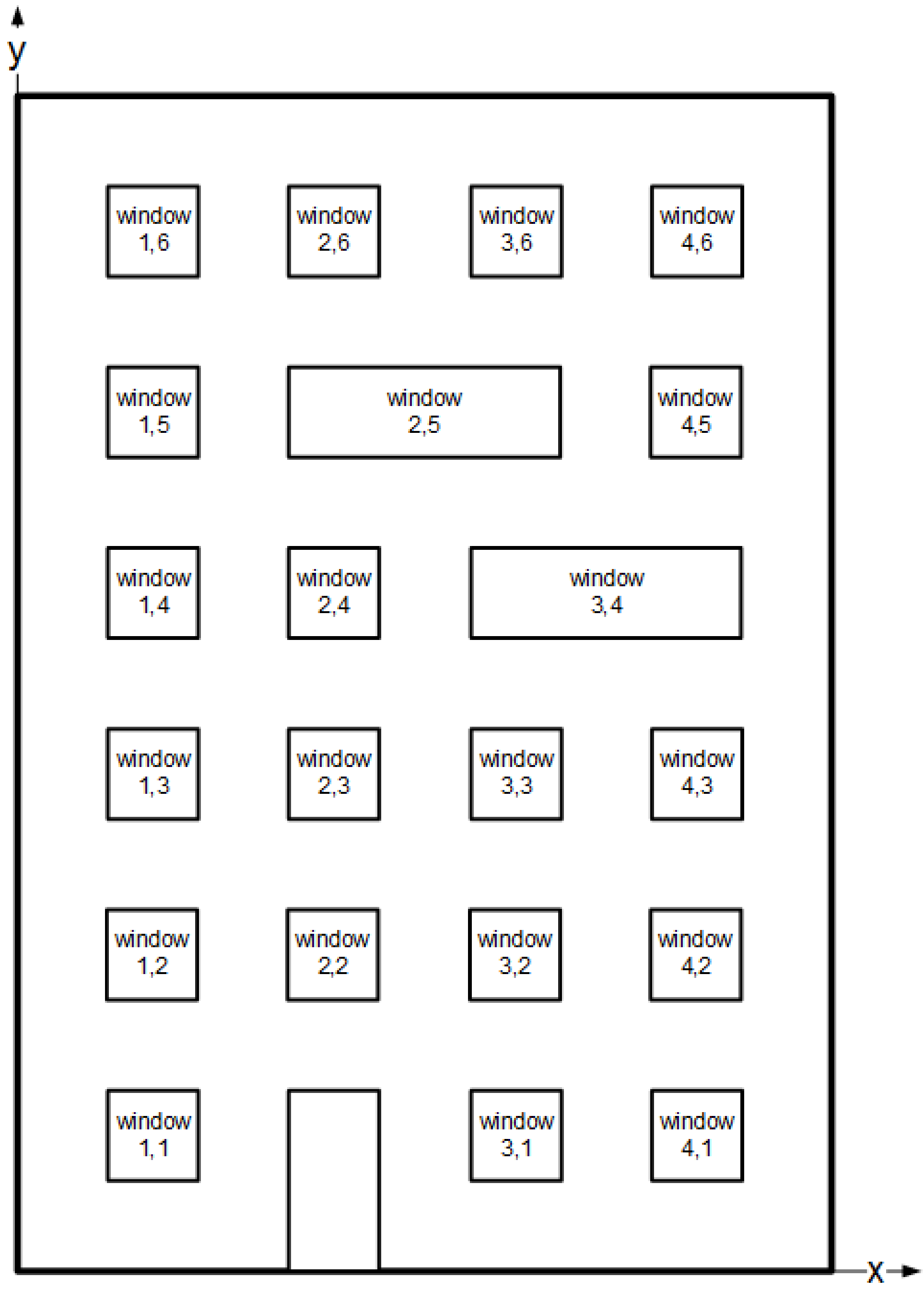
The parameterised data are written to the list element with the index *nId* upon a rising edge on *bWrite*, regardless of whether they represent valid values or not. Therefore the element structure ST_BARFacadeElement [▶ 502] also contains a plausibility bit, *bValue*, that relays precisely this information to the function block FB_BARShadingCorrection [▶ 264] / FB_BARShadingCorrectionSouth [▶ 267] and prevents incorrect calculations there.

Error (bValid=FALSE)

The function block FB_BARShadingCorrection [▶ 264] / FB_BARShadingCorrectionSouth [▶ 267], 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
- lrCorner1X must be greater than or equal to 0.0
- lrCorner1Y must be greater than or equal to 0.0
- lrWindowWidth must be greater than 0
- lrWindowHeight must be greater than 0

If one of these criteria is not fulfilled, then this is interpreted as an incorrect input and *bValid* is set to FALSE at the block output of FB_BARFacadeElementEntry as well as in the window element ST_BARFacadeElement [▶ 502].
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;
```

eDataSecurityType: If eDataSecurityType [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.

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: dto. Row index. **iRow and iColumn may not be zero!** This arises from the field definition, see above.

bRead: With a positive edge on this input, the information from 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 rising edge writes both the entered values and the calculated values into the selected field element *arrFacadeElement [iColumn,iRow]*.

usiGroup: Group membership,

lrCorner1X: X-coordinate of corner point 1 in meters.

lrCorner1Y: Y-coordinate of corner point 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 metres. See "Note" above.

lrCorner2Y: determined Y-coordinate of corner 2 of the window in metres. See "Note" above.

lrCorner3X: determined X-coordinate of corner 3 of the window in metres. See "Note" above.

lrCorner3Y: determined Y-coordinate of corner 3 of the window in metres. See "Note" above.

lrCorner4X: determined X-coordinate of corner 4 of the window in metres. See "Note" above.

lrCorner4Y: determined Y-coordinate of corner 4 of the window in metres. See "Note" above.

bValid: result check for the entered values.

udiErrorId: contains the error code if the values entered are not OK. See Error codes [▸ 318].

VAR_IN_OUT

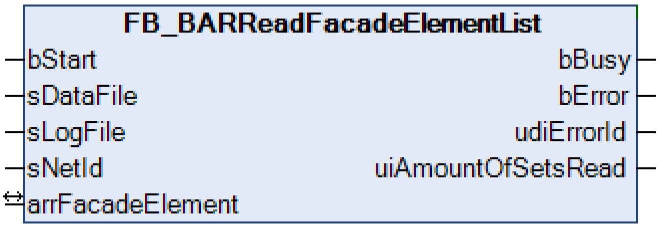
```
arrFacadeElement : ARRAY[1..iColumnsPerFacade, 1..iRowsPerFacade] OF ST_BARFacadeElement;
```

arrFacadeElement: List of facade elements [▸ 316].

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.8 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 [▸ 316]. 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 for the data 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: dto. 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 [▸ 318] or ADS_Error codes.

uiAmountOfSetsRead: Number of data sets read

VAR_IN_OUT

```
arrFacadeElement : ARRAY[1..iColumnsPerFacade, 1..iRowsPerFacade] OF ST_BARFacadeElement;
```

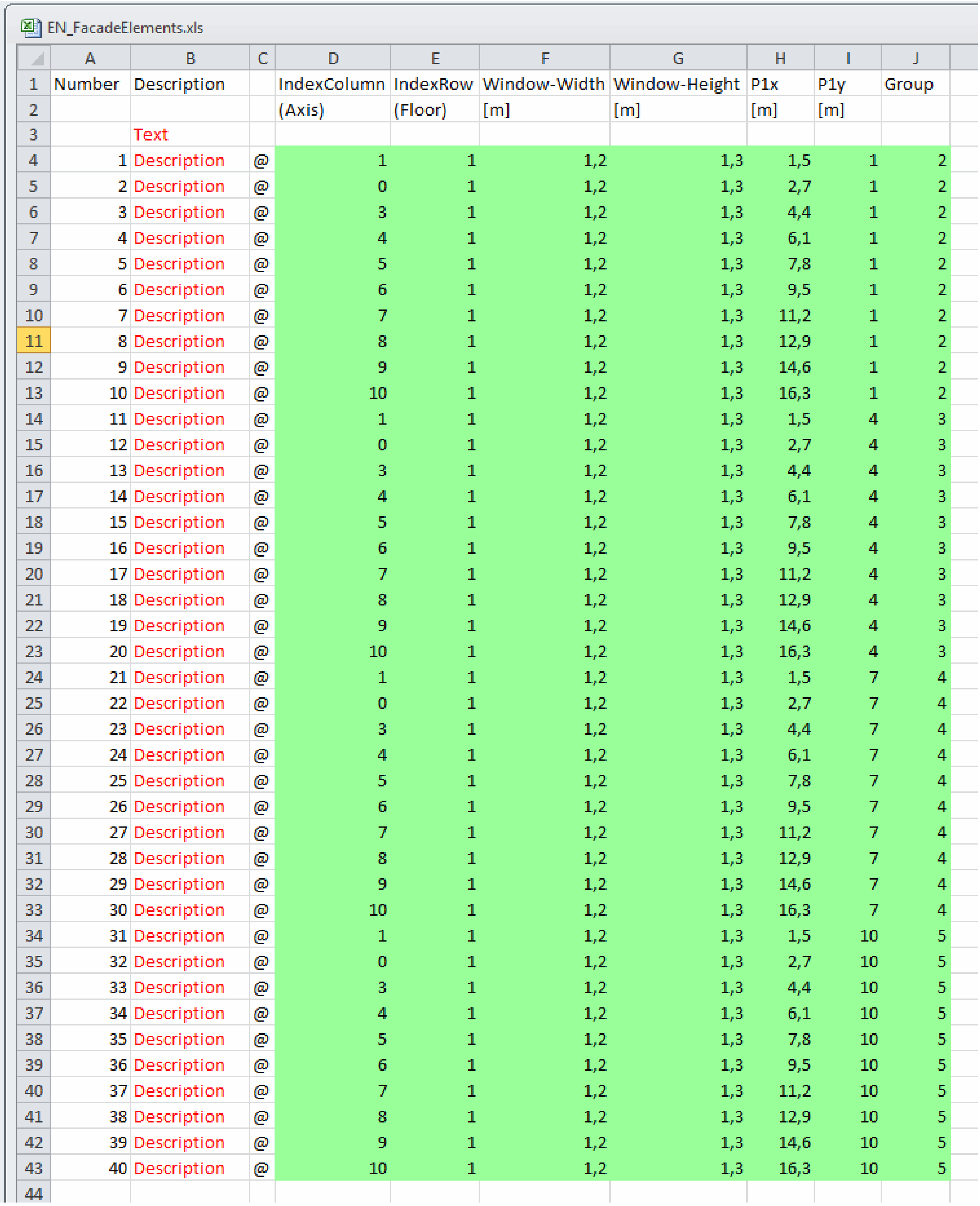
arrFacadeElement: List of facade elements [▶ 316].

Excel table

The following example shows the Excel table with the entries of the window elements.
All text fields are freely writeable; important are the green marked fields, wherein each line of a data set is marked there.
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 [▶ 316]. 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.
- The table has to be 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 [▶ 306], does not accept elements with the group entry '0'.



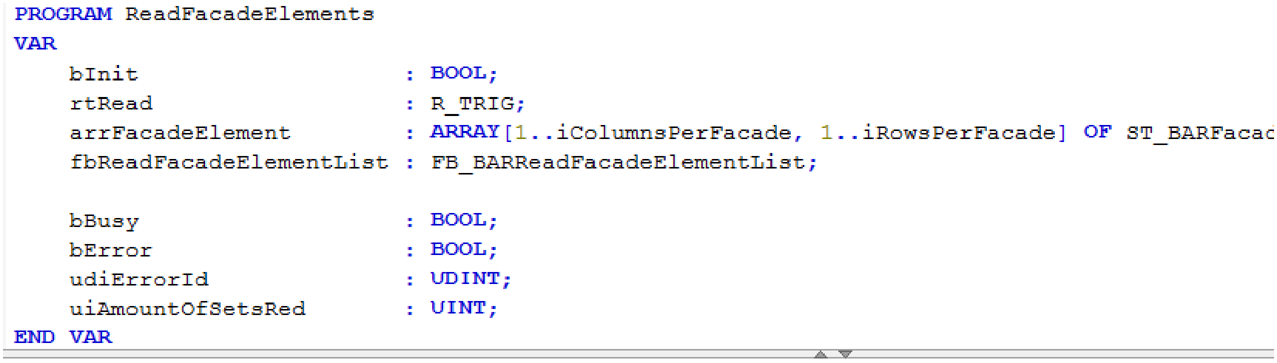
EN_FacadeElements.xls

	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										

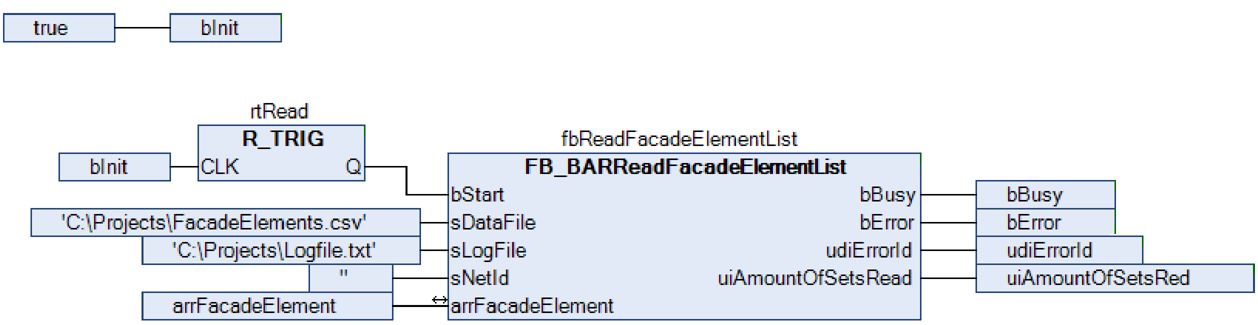
Log file

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

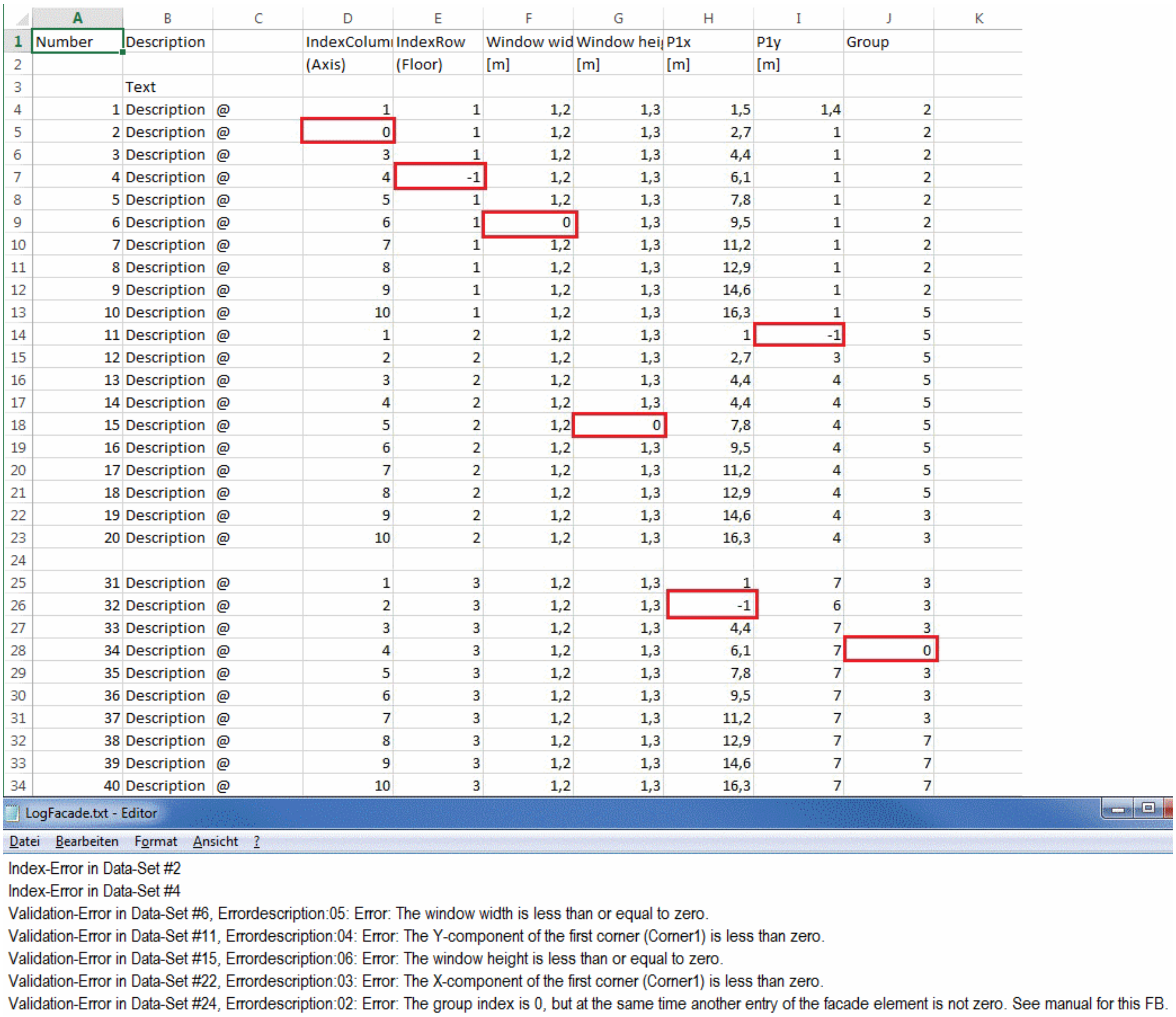


```
PROGRAM ReadFacadeElements
VAR
    bInit                     : BOOL;
    rtRead                    : R_TRIG;
    arrFacadeElement          : ARRAY[1..iColumnsPerFacade, 1..iRowsPerFacade] OF ST_BARFacac
    fbReadFacadeElementList : FB_BARReadFacadeElementList;

    bBusy                     : BOOL;
    bError                    : BOOL;
    udiErrorId                : UDINT;
    uiAmountOfSetsRed         : UINT;
END VAR
```



In this sample the variable *bInit* 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 TRUE as long as reading and writing are taking place. The error that occurred last is indicated on *udiErrorId*; *bError* is then TRUE. The number of data lines found and read is indicated on *uiAmountOfSetsRead* for checking purposes.

The errors marked were "built into" the following Excel list. This results in the shown log file:



	A	B	C	D	E	F	G	H	I	J	K
1	Number	Description		IndexColumı	IndexRow	Window wid	Window hei	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	0	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	

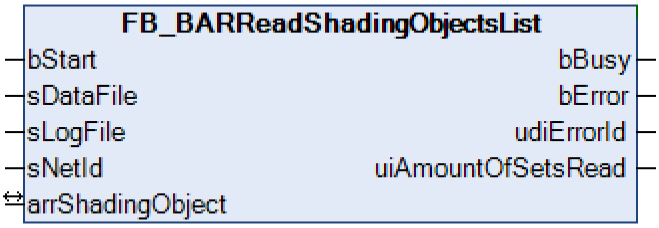
```
Index-Error in Data-Set #2
Index-Error in Data-Set #4
Validation-Error in Data-Set #6, Errordescription:05: Error: The window width is less than or equal to zero.
Validation-Error in Data-Set #11, Errordescription:04: Error: The Y-component of the first corner (Corner1) is less than zero.
Validation-Error in Data-Set #15, Errordescription:06: Error: The window height is less than or equal to zero.
Validation-Error in Data-Set #22, Errordescription:03: Error: The X-component of the first corner (Corner1) is less than zero.
Validation-Error in Data-Set #24, Errordescription:02: Error: The group index is 0, but at the same time another entry of the facade element is not zero. See manual for this FB.
```

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 [▶ 270] and was therefore provided in greater detail with an error number, which is explained in greater detail under error codes [▶ 318]. The third and the fourth errors likewise occurred after the internal validation. Important here is 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.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.9 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 [▶ 317]. 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 for the data 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 [▶ 318] 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 [▶ 317].

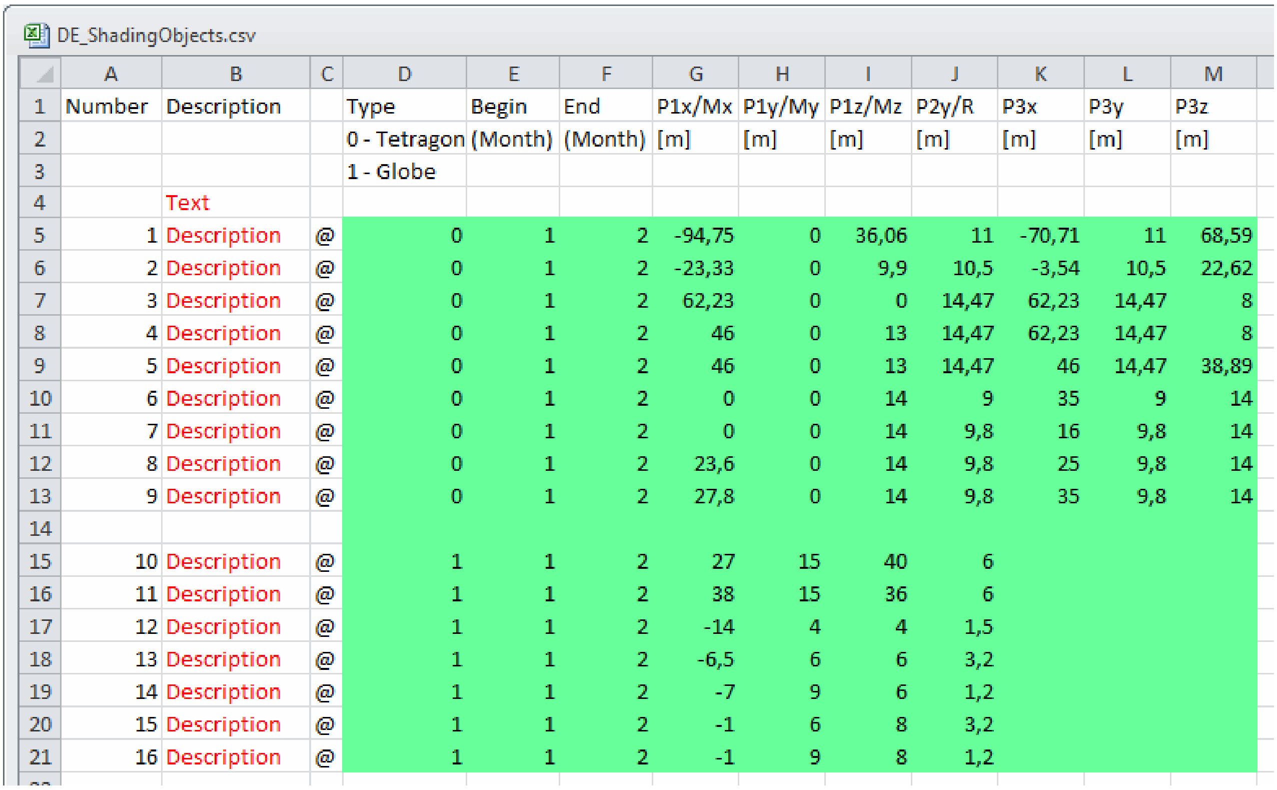
Excel table

The following example shows the Excel table with the entries of the window elements.
All text fields are freely writeable; important are the green marked fields, wherein each line of a data set is marked there. The columns G to J have a different meaning depending on whether the type square or ball is concerned. The columns K to M are to be left empty in the case of balls. With regard to the square coordinates, only the relevant data are entered and the remainder are internally calculated, see FB_BARShadingObjectsEntry [▶ 270].
The following rules are to be observed:

- A data set must always start with a '@'.
- The monthly entries may not be 0 or greater than 12, otherwise all 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 (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.
- The table has to be saved in Excel as file type "CSV (comma-separated values) (*.csv)".

Is not necessary to describe all shading objects that are possible per facade. Only those contained in the list ultimately take effect.



DE_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	@	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			

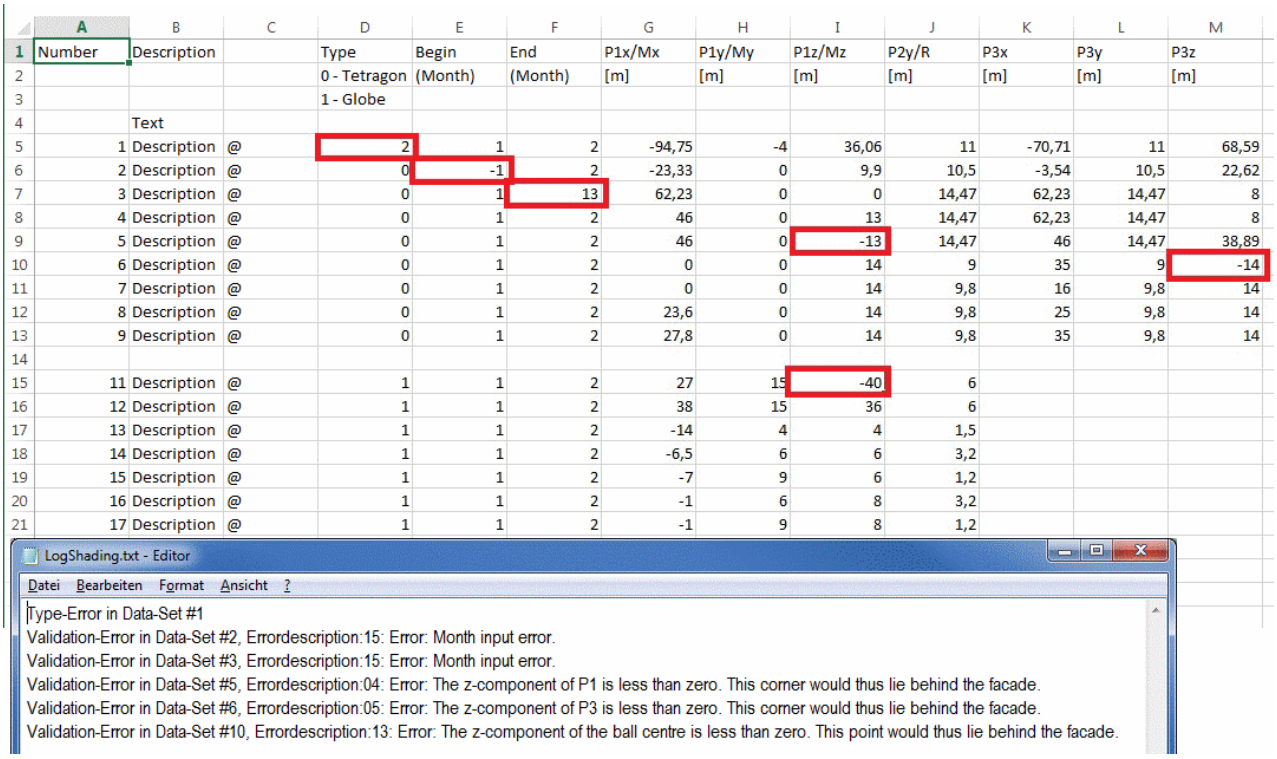
Log file

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 Example

In this example the variable *bInit* 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 takes 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 TRUEs as long as reading and writing are taking place. The error that occurred last is indicated on *udiErrorId*; *bError* is then TRUE. The number of data lines found and read is indicated on *uiAmountOfSetsRead* for checking purposes.

The errors marked were built into the following Excel list. This results in the shown log file:



	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	@	2	1	2	-94,75	-4	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	13	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	2	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			

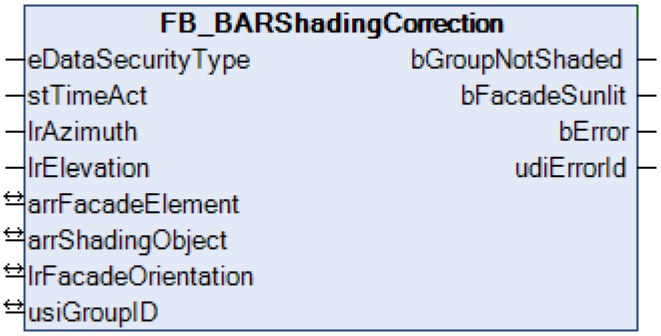
```
Type-Error in Data-Set #1
Validation-Error in Data-Set #2, Errordescription:15: Error: Month input error.
Validation-Error in Data-Set #3, Errordescription:15: Error: Month input error.
Validation-Error in Data-Set #5, Errordescription:04: Error: The z-component of P1 is less than zero. This corner would thus lie behind the facade.
Validation-Error in Data-Set #6, Errordescription:05: Error: The z-component of P3 is less than zero. This corner would thus lie behind the facade.
Validation-Error in Data-Set #10, Errordescription:13: Error: The z-component of the ball centre is less than zero. This point would thus lie behind the facade.
```

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 [▶ 270] and was therefore provided in greater detail with an error number, which is explained in greater detail under error codes [▶ 318]. 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 number entered in the list, but by the actual sequential number: only 16 data sets were read in here.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.10 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 [▶ 267].

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 [▸ 316] 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 [▸ 317] is likewise globally defined. The IN-OUT variable *arrShadingObject* accesses it directly.

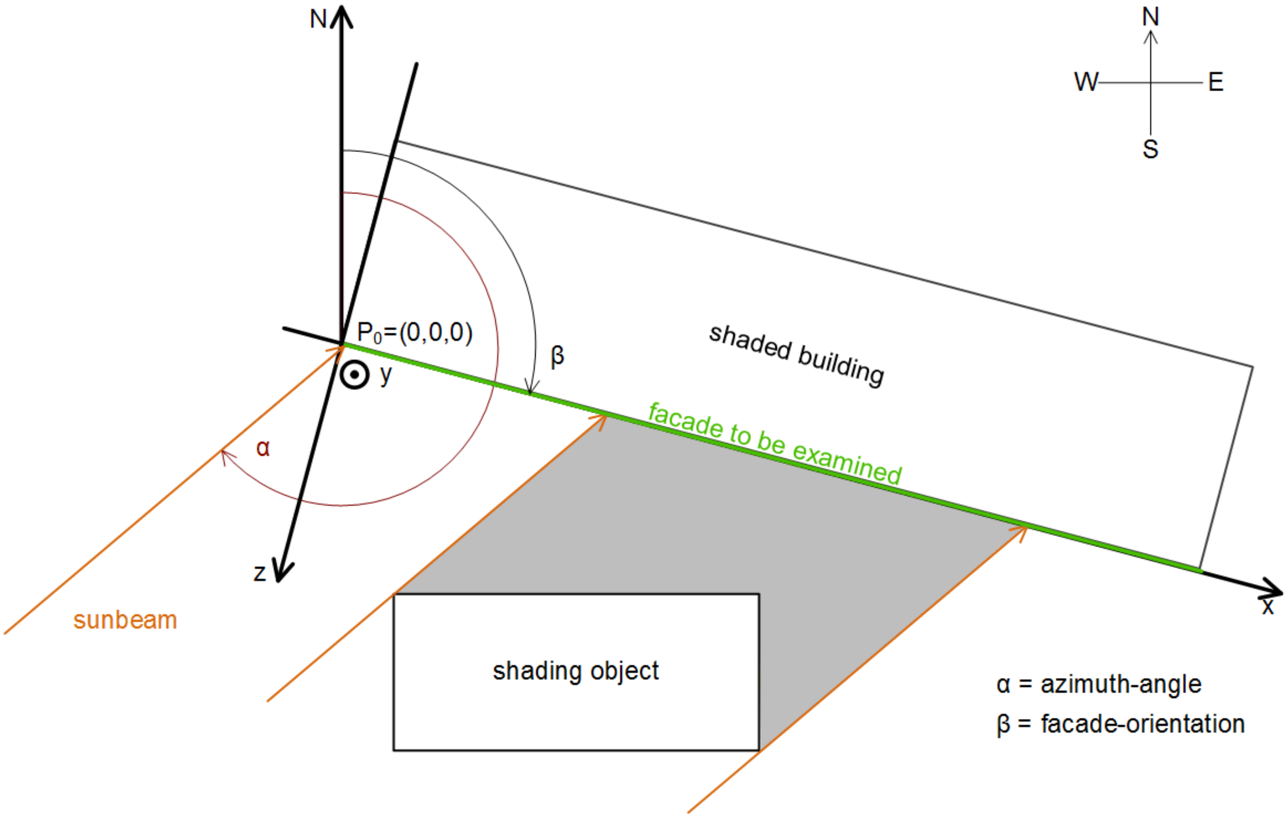
On the basis of the facade orientation (*lrFacadeOrientation*), the direction of the sun (*lrAzimuth*) and the height of the sun (*lrElevation*), a calculation can be performed for each corner of a window to check whether this lies in a shaded area. A window group is considered to be completely shaded 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	β=0°
East	β=90°
South	β=180°
West	β=270°

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 solar 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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.

stTimeAct: Entry of the current time in GMT (Greenwich Mean Time).

lrAzimuth: Sun direction in degrees at the time of observation.

lrElevation: Sun elevation in degrees at the time of observation.

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 if the sun is shining 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 [▶ 318].

VAR_IN_OUT

```
arrFacadeElement      : ARRAY[1..iColumnsPerFacade, 1..iRowsPerFacade] OF ST_BARFacadeElement;
arrShadingObject      : ARRAY[1..iShadingObjects] OF ST_BARShadingObject;
lrFacadeOrientation   : LREAL;
usiGroupID            : USINT;
```

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

arrFacadeElement: List of facade elements [▶ 316].

arrShadingObject: List of shading objects [▶ 317].

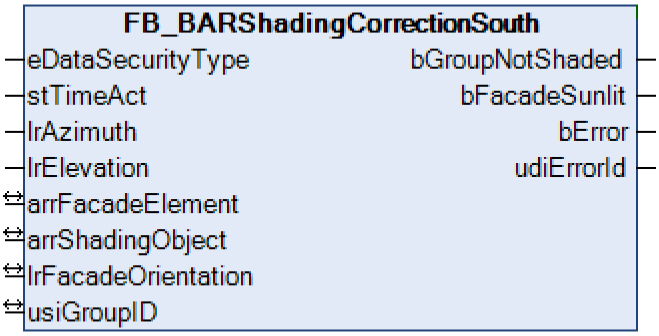
lrFacadeOrientation: Facade orientation, see illustration above.

usiGroupId: Window group regarded. The group 0 is reserved here for unused window elements, see FB_BARFacadeElementEntry [▶ 252]. 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.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.11 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 [▶ 264].

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 [▶ 316] 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 [▶ 317] is likewise globally defined. The IN-OUT variable *arrShadingObject* accesses it directly.

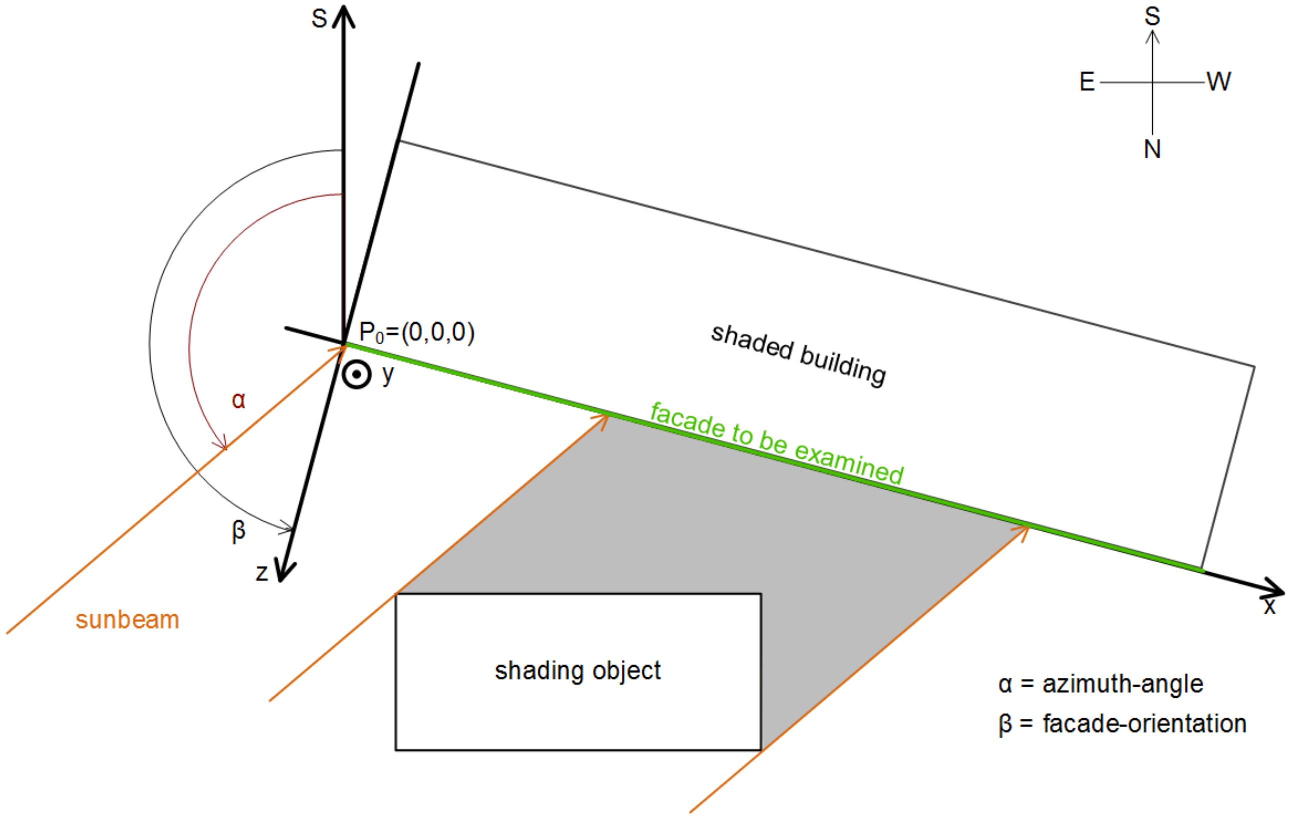
On the basis of the facade orientation (*lrFacadeOrientation*), the direction of the sun (*lrAzimuth*) and the height of the sun (*lrElevation*), a calculation can be performed for each corner of a window to check whether this lies in a shaded area. A window group is considered to be completely shaded 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	β=0°
East	β=90°
North	β=180°
West	β=270°

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 solar 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 [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.

stTimeAct: Entry of the current time in GMT (Greenwich Mean Time).

lrAzimuth: Sun direction in degrees at the time of observation.

lrElevation: Sun elevation in degrees at the time of observation.

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 if the sun is shining 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 [▶ 318].

VAR_IN_OUT

```
arrFacadeElement      : ARRAY[1..iColumnsPerFacade, 1..iRowsPerFacade] OF ST_BARFacadeElement;
arrShadingObject      : ARRAY[1..iShadingObjects] OF ST_BARShadingObject;
lrFacadeOrientation   : LREAL;
usiGroupID            : USINT;
```

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

arrFacadeElement: List of facade elements [▶ 316].

arrShadingObject: List of shading objects [▶ 317].

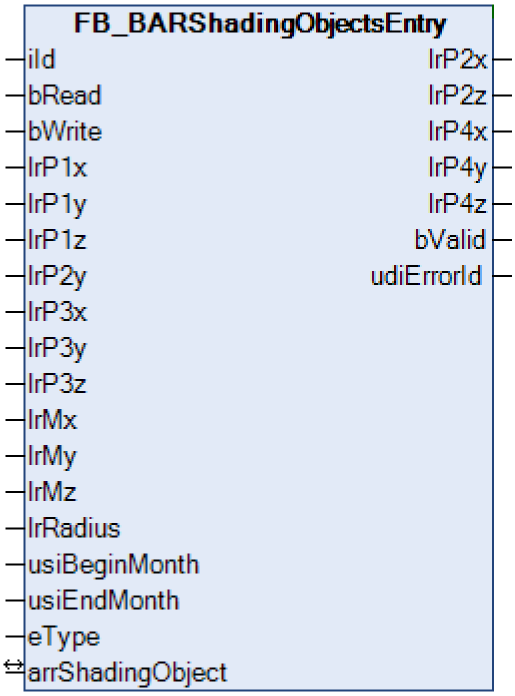
lrFacadeOrientation: Facade orientation, see illustration above.

usiGroupId: Window group regarded. The group 0 is reserved here for unused window elements, see FB_BARFacadeElementEntry [▶ 252]. 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.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.12 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 [▶ 317]. It is intended to facilitate the input of the element information - also with regard to the use of the target visualization. A schematic illustration of the objects with description of the coordinates is given in Shading correction: principles and definitions [▶ 240].
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 [▶ 503]). This information consists of the selected type of shading (rectangle or ball) 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 ball the center *(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*.

Note: The fact that the square 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*;

This limits the input options for a rectangle such that the side edges are perpendicular to the floor (*lrP2x* = *lrP1x* and *lrP4x* = *lrP3x*), that the rectangle is not tilted (*lrP2z* = *lrP1z* and *lrP4z* = *lrP3z*), and that it can only have a positive height (y-direction) (*lrP4y* = *lrP1y*).

The function block is used in three steps:

- Read
- Change
- Write

Read

With the entry to *iId* the appropriate element is selected from the list *arrShadingObject, [iId]*. A rising edge on *bRead* reads the data. These values are assigned to the input and output variables of the function block. This concerns the input values *lrP1x*, *lrP1y*, *lrP1z*, *lrP2y*, *lrP3x*, *lrP3y*, *lrP3z*, *lrMx*, *lrMy*, *lrMz* and *lrRadius* and 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 visualisation.

Change

In a next program step the listed input values can then be changed. If the use of a square is preselected via the value "*eObjectTypeTetragon"* at the input eType [▶ 501], then the output values *lrP2x, lrP2z, lrP4x, lrP4y* and *lrP4z* result from the square coordinates 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 [▶ 318] is output at the output *udiErrorId*.
If a square is defined, then only the inputs *lrP1x*, *lrP1y*, *lrP1z*, *lrP2y*, *lrP3x*, *lrP3y* and *lrP3z* need to be described; the inputs *lrMx*, *lrMy*, *lrMz* and *lrRadius* do not need to be linked. In case of a ball definition, only *lrMx, lrMy, lrMz* and *lrRadius* need to be described and the square coordinates can remain unlinked.

The parameterised data are written to the list element with the index *iId* upon a rising edge on *bWrite*, regardless of whether they represent valid values or not. Therefore the element structure ST_BARShadingObject [▶ 503] also contains a plausibility bit, *bValid*, that relays precisely this information to the function block FB_BARShadingCorrection [▶ 264] / FB_BARShadingCorrectionSouth [▶ 267] and prevents incorrect calculations there.

This approach is to be regarded only as a proposal. It is naturally also possible to parameterise 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;
```

iId: Index of the selected element. This refers to the selection of a field element of the array saved in the IN-OUT variable *arrShadingObject*. **iId may not be zero!** This results from the field definition [▶ 270], *arrShadingObject: ARRAY[1..iShadingObjects] OF ST_BARShadingObject;*

bRead: With a rising edge on this input the information of the selected element, *arrShadingObject [iId]*, is read into the function block and assigned to the input variables *lrP1x* to *eType* as well as the output variables *lrP2x* to *lrP4z*. If data are already present on the inputs *lrP1x* to *eType* at this time, then the data previously read are immediately overwritten with these data.

bWrite: A positive edge writes the values on the inputs *rP1x* to *eType* as well as the values determined and assigned to the outputs *lrP2x* to *lrP4z* into the selected field element *arrShadingObject[iId].*

lrP1x: X-coordinate of point 1 of the shading element (square) in meters.

lrP1y: Y-coordinate of point 1 of the shading element (square) in meters.

lrP1z: Z-coordinate of point 1 of the shading element (square) in meters.

lrP2y: Y-coordinate of point 2 of the shading element (square) in meters.

lrP3x: X-coordinate of point 3 of the shading element (square) in meters.

lrP3y: Y-coordinate of point 3 of the shading element (square) in meters.

lrP3z: Z-coordinate of point 3 of the shading element (square) in meters.

lrMx: X-coordinate of the center of the shading element (ball) in meters.

lrMy: Y-coordinate of the center of the shading element (ball) in meters.

lrMz: Z-coordinate of the center of the shading element (ball) in meters.

lrRadius: Radius of the shading element (ball) in meters.

usiBeginMonth: Beginning of the shading period (month).

usiEndMonth: End of the shading period (month).

eType: Selected type of element: Rectangle or ball. See E_BARShadingObjectType [▶ 501].

Remark about the shading period:

The entries for the months may not be 0 or greater than 12, otherwise all 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 (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 (square) in metres. See "Note" above.

lrP2z: determined Z-coordinate of point 2 of the shading element (square) in metres. See "Note" above.

lrP4x: determined X-coordinate of point 4 of the shading element (square) in metres. See "Note" above.

lrP4y: determined Y-coordinate of point 4 of the shading element (square) in metres. See "Note" above.

lrP4z: determined Z-coordinate of point 4 of the shading element (square) in metres. See "Note" 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 ball the centre 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 criteria mentioned. See Error codes [▶ 318].

VAR_IN_OUT

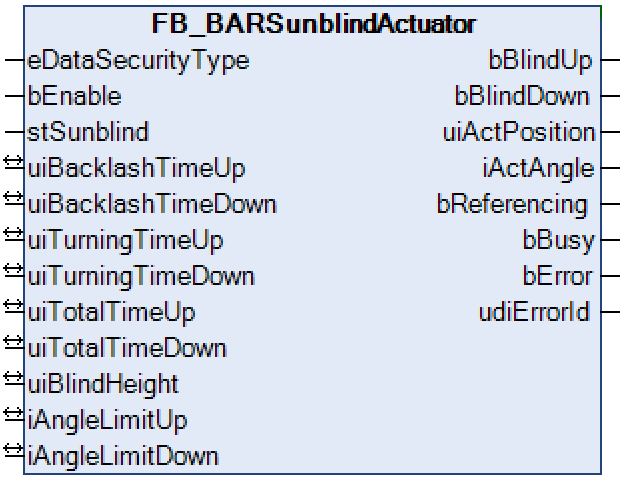
```
arrShadingObject : ARRAY[1..iShadingObjects] OF ST_BARShadingObject;
```

arrShadingObject : List of shading objects [▶ 317]. Values are persistently saved.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.13 FB_BARSunblindActuator



This function block is used for positioning of a slatted blind via two outputs: drive up and drive down. The blind can be driven to any desired (height) position and slat angle via the positioning telegram stSunblind [⏵ 504]. On top of that, the positioning telegram stSunblind [⏵ 504] 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 [⏵ 292].

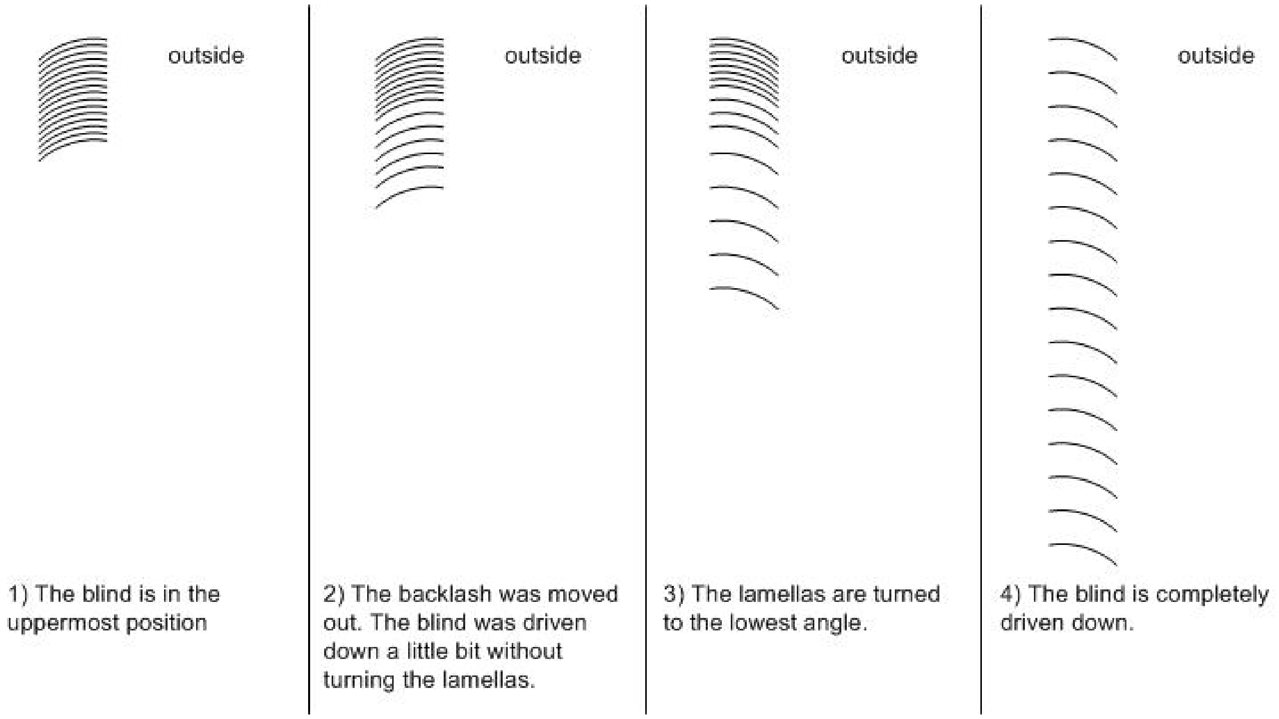
The total stroke time in milliseconds is limited here by the UINT format to 65535 ms. FB_BARSunblindActuatorEx [⏵ 279] should be used for longer stroke times as it represents these times in UDINT format (*udiTotalTimeUp/udiTotalTimeDown*).

Structure of the blind positioning telegram stSunblind [⏵ 504].

```
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 drive profile:



1) The blind is in the uppermost position

2) The backlash was moved out. The blind was driven down a little bit without turning the lamellas.

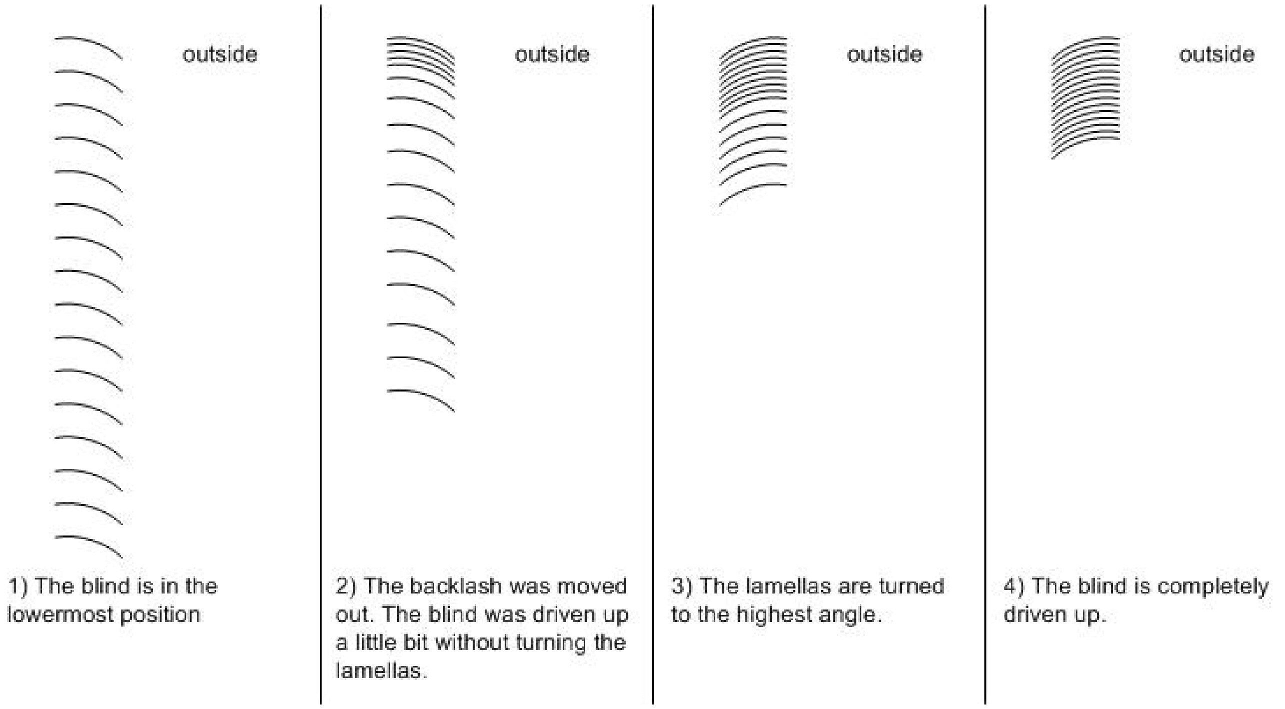
3) The lamellas are turned to the lowest angle.

4) The blind is completely driven down.

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 louvre 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 louvres start to turn from the "inward low point" to the "outward low point". During this time the louvre 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 travelled, 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 time interval of the subsequent turning of the louvres 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:



1) The blind is in the lowermost position

2) The backlash was moved out. The blind was driven up a little bit without turning the lamellas.

3) The lamellas are turned to the highest angle.

4) The blind is completely driven up.

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). This time is also 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 reversal backlash has already been traversed, the most secure way to measure the reversal backlash of the upward movement or its extension 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.

Parameterisation

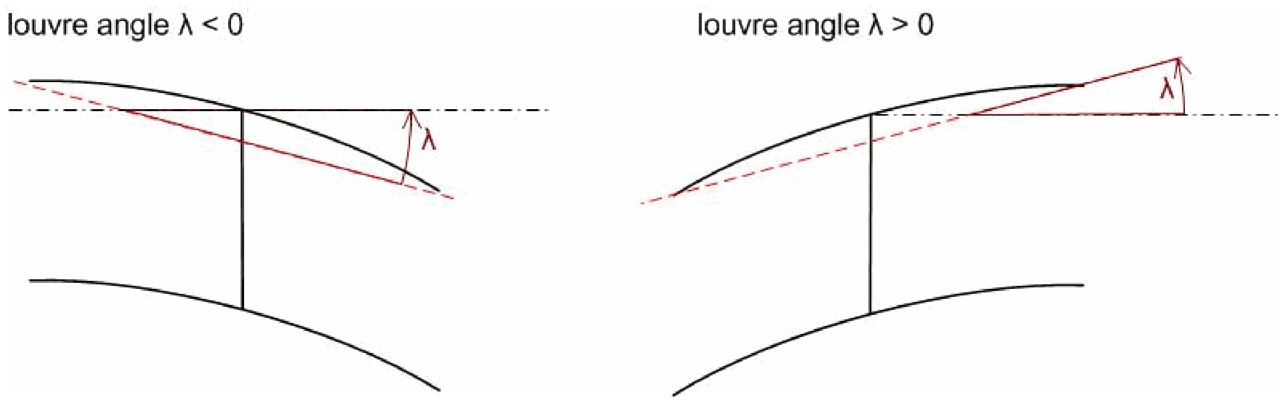
For the calculation of the (height) position and the louvre angle, the following times now have to be determined for both the upward and downward movement:

- 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 louvre angle after turning upwards (*iAngleLimitUp* in degrees)
- the lowest louvre angle after turning downwards (*iAngleLimitDown* in degrees)

The louvre angle λ is defined by a notional straight line through the end points of the louvre to the horizontal.



Functioning

The function block fundamentally controls the blind via the information from the positioning telegram stSunblind [▸ 504]. If automatic mode is active (*bManualMode*=FALSE), then the current position and louvre angle are always driven to, wherein changes are immediately accounted for. The height positioning takes priority: First the entered height and afterwards the louvre angle are driven to. 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 drive-up time. The position is then in any case "0" and the louvre angle is at its maximum. Since blind positioning without an encoder is naturally always susceptible to error, it is important to automatically reference as often as possible: each time the "0" position is to be driven to (the angle is unimportant), the blind initially drives upward quite normally with continuous position calculation. Once the calculated position value of 0% has been reached, the output *bBlindUp* is kept active until another complete drive-up time + 5 s has elapsed. For reasons of flexibility there are now two possibilities to interrupt the referencing procedure: Until the calculated 0% position has been reached, a change of position is still accepted and executed; after reaching this 0% position, the manual command "down" can still move the blind differently. 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 [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 [▶ 504].

VAR_OUTPUT

```
bBlindUp        : BOOL;
bBlindDown      : BOOL;
uiActPosition   : UINT;
uiActAngle      : UINT;
bReferencing    : BOOL;
bBusy           : BOOL;
bError          : BOOL;
udiErrorId      : UDINT;
```

bBlindUp: control output for blind up.

bBlindDown: control output for blind down.

uiActPosition: current position in percent.

uiActAngle: current louvre angle in degrees.

bReferencing: the blind is in the reference-process, which means, that the output *bBlindUp* will be set for the complete drive-up time + 5 s. Only the manual command "down" can move the blind in the opposite direction an terminate the reference-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 [▶ 318].

VAR_IN_OUT

```
uiBacklashTimeUp   : UINT;
uiBacklashTimeDown : UINT;
uiTurningTimeUp    : UINT;
uiTurningTimeDown  : UINT;
uiTotalTimeUp      : UINT;
uiTotalTimeDown    : UINT;
uiBlindHeight      : UINT;
iAngleLimitUp      : INT;
iAngleLimitDown    : INT;
```

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

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: Complete time for driving up in ms.

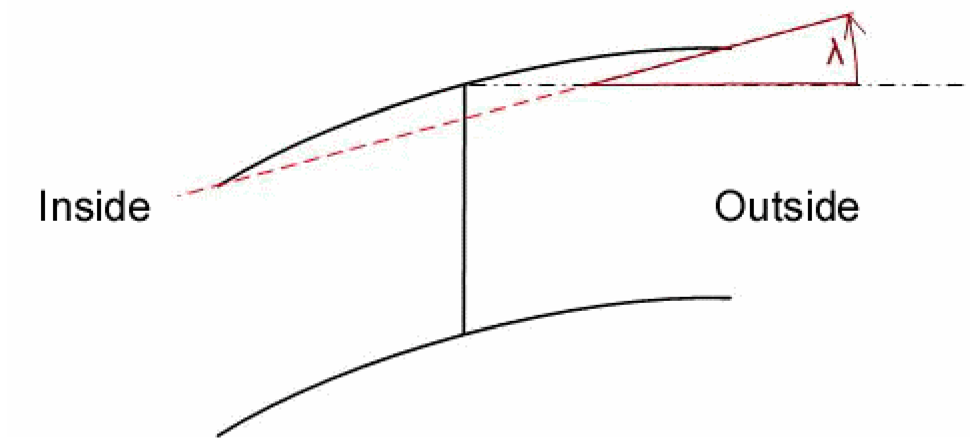
uiTotalTimeDown: Complete time for driving down in ms.

uiBlindHeight: Shutter height in mm.

iAngleLimitUp: Highest position of the slats in degrees.

This position is reached once the blind has moved to the top position.

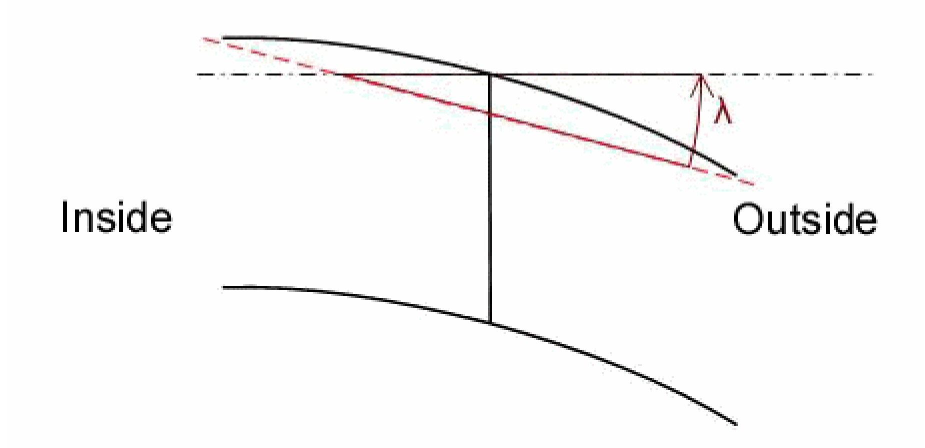
The slat angle λ, as defined above, is then typically greater than zero.



iAngleLimitDown: Lowest position of the slats in degrees.

This position is reached once the blind has moved to the bottom position.

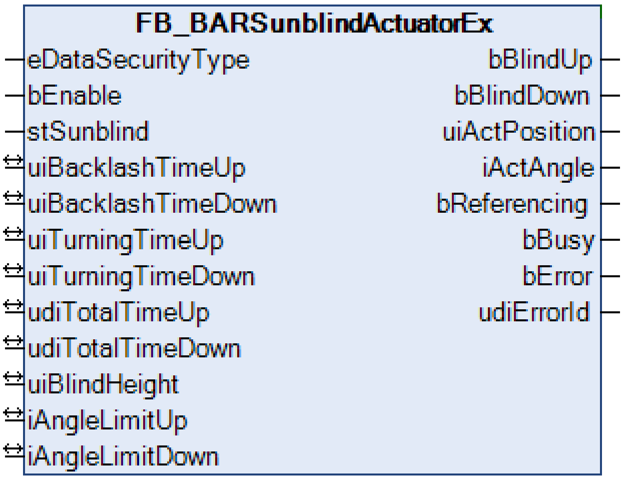
The slat angle λ, as defined above, is then typically less than zero.



Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.14 FB_BARSunblindActuatorEx



This function block is used for positioning of a slatted blind via two outputs: drive up and drive down. The blind can be driven to any desired (height) position and slat angle via the positioning telegram stSunblind [▸ 504]. On top of that, the positioning telegram stSunblind [▸ 504] 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 [▸ 292].

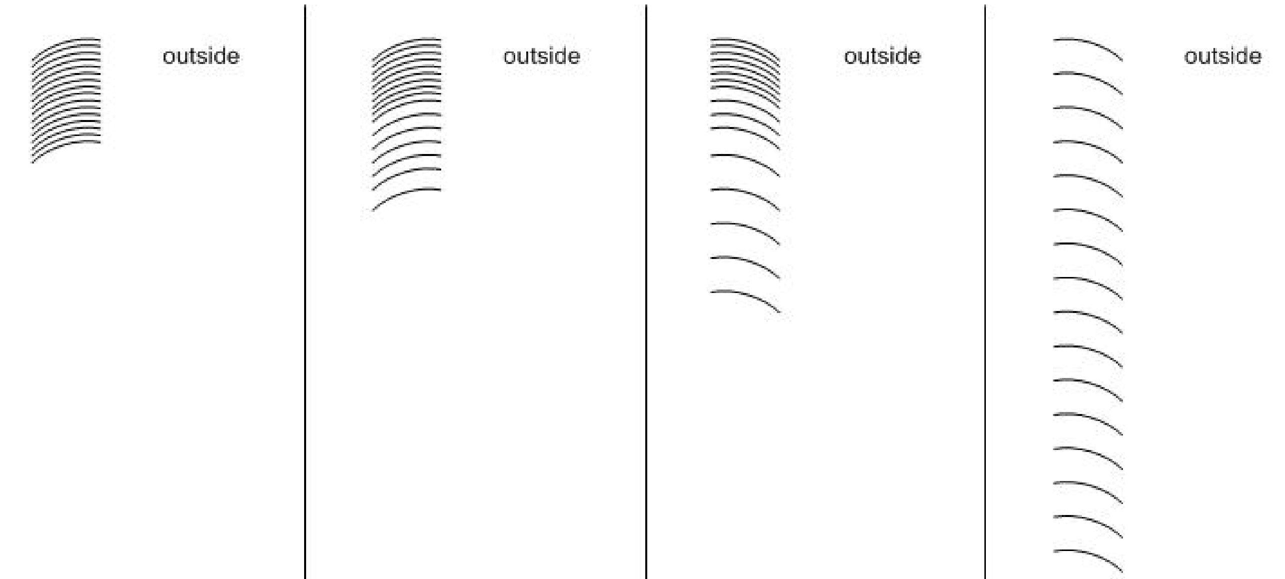
As opposed to the function block FB_BARSunblindActuator [▸ 273], this allows the input of longer stroke times because they are represented here not in UINT, but in the UDINT format (*udiTotalTimeUp/ udiTotalTimeDown*).

Structure of the blind positioning telegram stSunblind [▸ 504].

```
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:



1) The blind is in the uppermost position

2) The backlash was moved out. The blind was driven down a little bit without turning the lamellas.

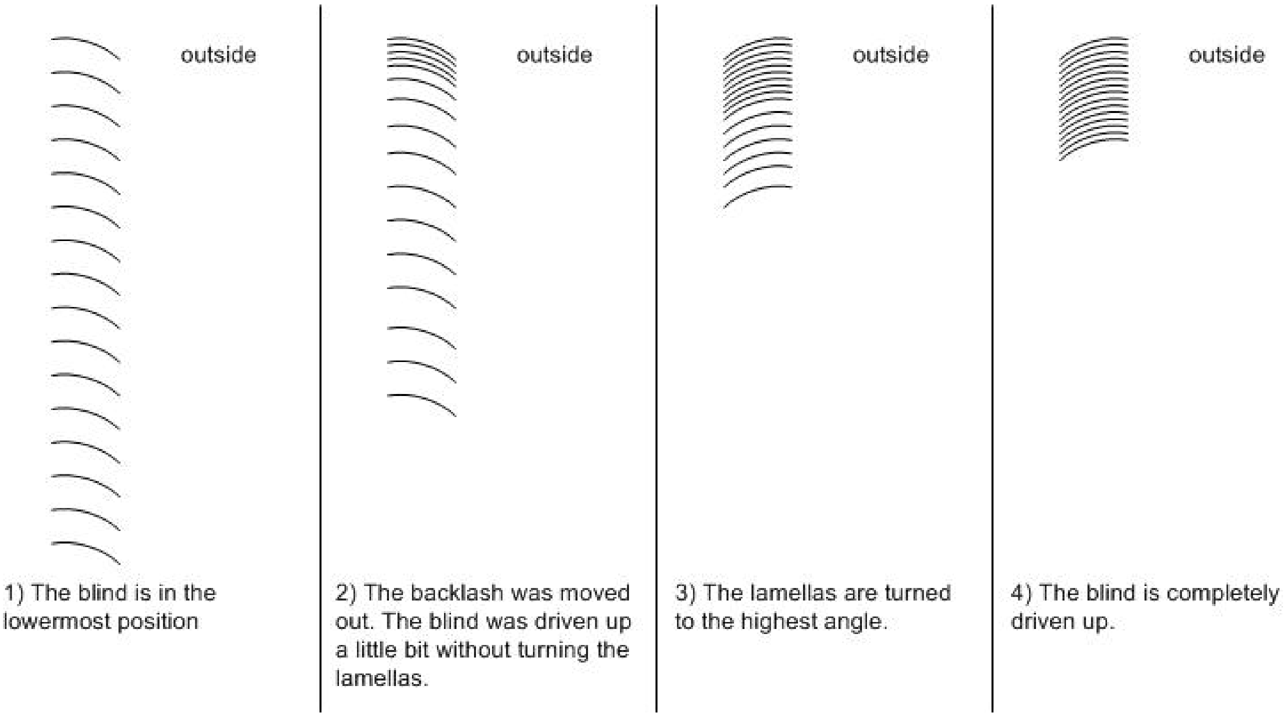
3) The lamellas are turned to the lowest angle.

4) The blind is completely driven down.

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 reversal backlash has already been traversed, the most secure way to measure the reversal backlash of the downward movement or its extension time is when the blind has initially been driven fully upward. A further important parameter is the time interval 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:



1) The blind is in the lowermost position

2) The backlash was moved out. The blind was driven up a little bit without turning the lamellas.

3) The lamellas are turned to the highest angle.

4) The blind is completely driven up.

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). This time is also 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 reversal backlash has already been traversed, the most secure way to measure the reversal backlash of the upward movement or its extension 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

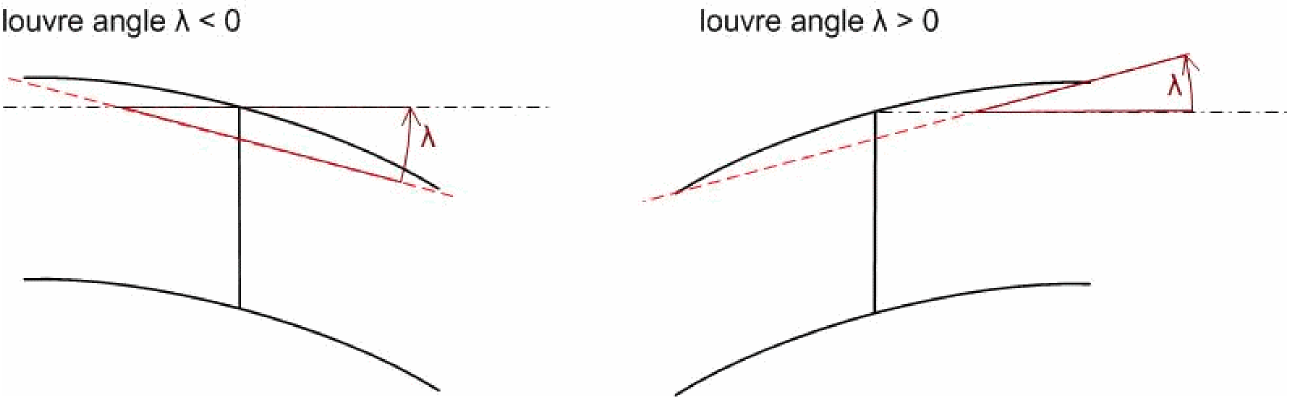
For the calculation of the (height) position and the slat angle, the following times now have to be determined for both the upward and downward movement:

- the travel time of the backlash (*uiBacklashTimeUp* / *uiBacklashTimeDown* in ms)
- the turning duration (*uiTurningTimeUp* / *uiTurningTimeDown* in ms)
- the total stroke 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.



Functioning

The function block fundamentally controls the blind via the information from the positioning telegram stSunblind [▶ 504]. If automatic mode is active (*bManualMode*=FALSE), then the current position and louvre angle are always driven to, wherein changes are immediately accounted for. The height positioning takes priority: First the entered height and afterwards the louvre angle are driven to. 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 drive-up time. The position is then in any case "0" and the louvre angle is at its maximum. Since blind positioning without an encoder is naturally always susceptible to error, it is important to automatically reference as often as possible: each time the "0" position is to be driven to (the angle is unimportant), the blind initially drives upward quite normally with continuous position calculation. Once the calculated position value of 0% has been reached, the output *bBlindUp* is kept active until another complete drive-up time + 5 s has elapsed. For reasons of flexibility there are now two possibilities to interrupt the referencing procedure: Until the calculated 0% position has been reached, a change of position is still accepted and executed; after reaching this 0% position, the manual command "down" can still move the blind differently. 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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 [▶ 504].

VAR_OUTPUT

```
bBlindUp        : BOOL;
bBlindDown      : BOOL;
uiActPosition   : UINT;
uiActAngle      : UINT;
bReferencing    : BOOL;
bBusy           : BOOL;
bError          : BOOL;
udiErrorId      : UDINT;
```

bBlindUp: control output for blind up.

bBlindDown: control output for blind down.

uiActPosition: current position in percent.

uiActAngle: current louvre angle in degrees.

bReferencing: the blind is in the reference-process, which means, that the output *bBlindUp* will be set for the complete drive-up time + 5 s. Only the manual command "down" can move the blind in the opposite direction and terminate the reference-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 [▶ 318].

VAR_IN_OUT

```
uiBacklashTimeUp    : UINT;
uiBacklashTimeDown  : UINT;
uiTurningTimeUp     : UINT;
uiTurningTimeDown   : UINT;
udiTotalTimeUp      : UDINT;
udiTotalTimeDown    : UDINT;
uiBlindHeight       : UINT;
iAngleLimitUp       : INT;
iAngleLimitDown     : INT;
```

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

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: Complete time for driving up in ms.

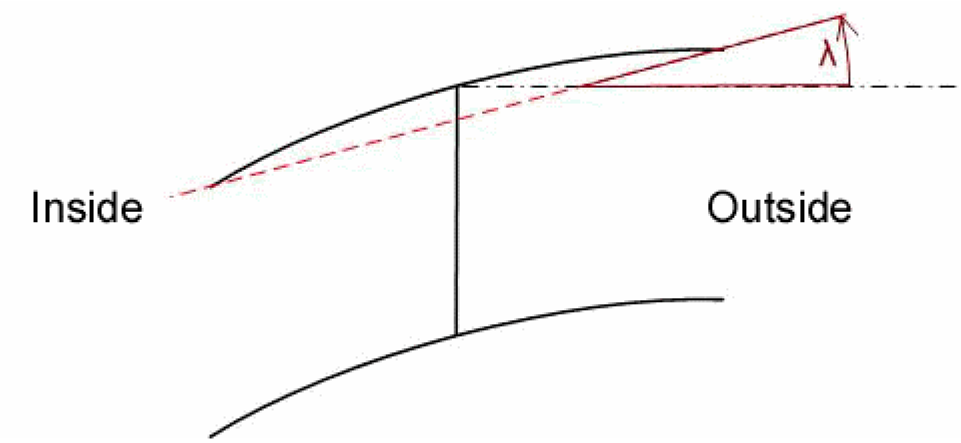
udiTotalTimeDown: Complete time for driving down in ms.

uiBlindHeight: Shutter height in mm.

iAngleLimitUp: Highest position of the slats in degrees.

This position is reached once the blind has moved to the top position.

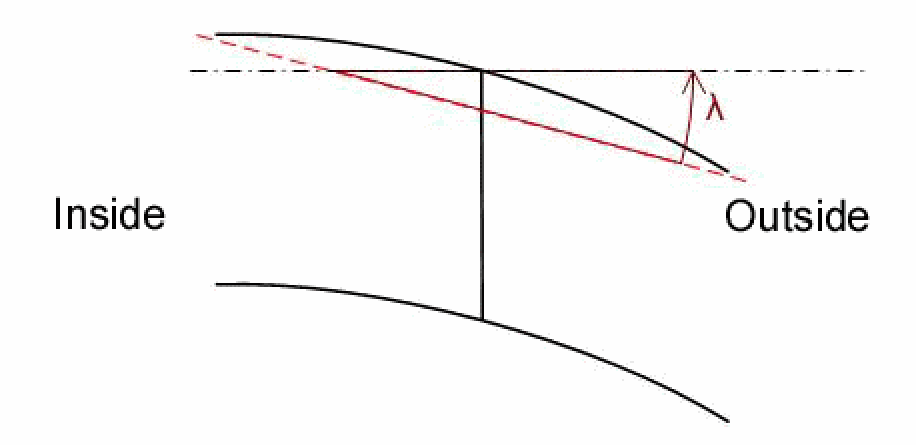
The slat angle λ, as defined above, is then typically greater than zero.



iAngleLimitDown: Lowest position of the slats in degrees.

This position is reached once the blind has moved to the bottom position.

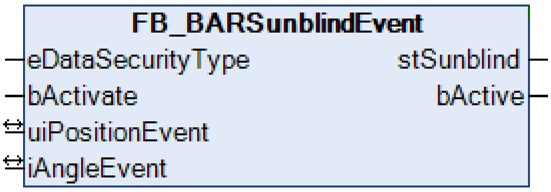
The slat angle λ, as defined above, is then typically less than zero.



Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.15 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 [▶ 504] and the values *uiPositionEvent* for the shutter 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 [▶ 504] 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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 setpoint values together with the active flag in the positioning telegram ST_BARSunblind [▶ 504]. 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 [▶ 504] and serves merely to indicate whether the function block is sending an active telegram.

stSunblind : Output structure of the blind positions, see ST_BARSunblind [▶ 504]

VAR_IN_OUT

```
uiPositionPark : UINT;
iAnglePark     : INT;
```

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

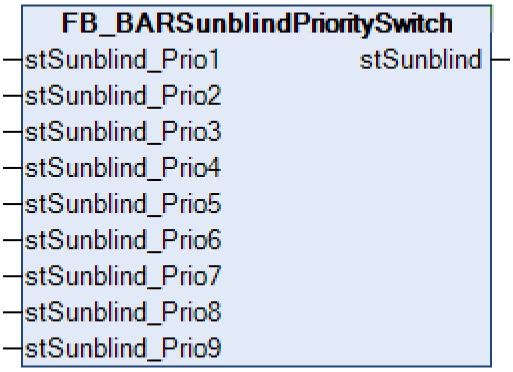
uiPositionEvent: Height position of the blind in % in case of activation.

iAngleEvent: Slat angle of the blind in degrees in case of activation.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.16 FB_BARSunblindPrioritySwitch



Priority controller for up to 9 positioning telegrams (*stSunblind_Prio1* ... *stSunblind_Prio9*) of the type ST_BARSunblind [▸ 504].

Structure of the blind positioning telegram ST_BARSunblind [▸ 504].

```
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 the output *stSunblind*. "Active" means that the variable *bActive* is set within the structure of the positioning telegram.

Note: This function block is to be programmed in such a way that one of the applied telegrams is always active. If this is not the case, then an empty telegram is output, i.e. *uiPosition*=0, *iAngle*=0, *bManUp*=FALSE, *bManDown*=FALSE, *bManualMode*=FALSE and *bActive*=FALSE. Since the blind function block FB_BARSunblindActuator [▸ 273] or the roller shutter function block FB_BARRollerblind [▸ 287] for its part ignores the *bActive* flag, this telegram would thus be evaluated as a command to drive 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

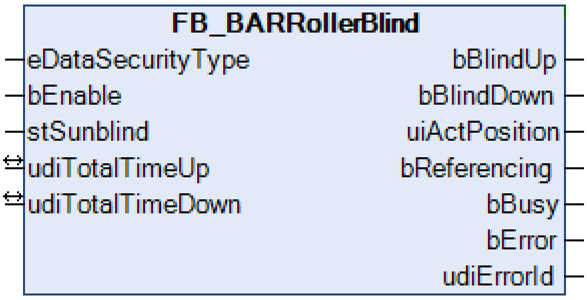
```
stSunblind     : ST_BARSunblind;
```

stSunblind: resulting positioning telegram.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.17 FB_BARRollerBlind



This function block is used for positioning a roller blind via two outputs: blind up and blind down. The blind can be driven to any desired position with the position telegram stSunblind [▶ 504]. In addition, the positioning telegram stSunblind [▶ 504] offers manual commands, which can be used to move the blind to particular positions. These manual commands are controlled by the function block FB_BARSunblindSwitch [▶ 292].

Structure of the blind positioning telegram stSunblind [▶ 504].

```
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.
Allowance is made for the different driving characteristics through the two different runtime parameters *udiTotalTimeUp* (upward runtime of the blind in ms) and *udiTotalTimeDown* (downward runtime of the blind in ms).

Functioning

The function block fundamentally controls the blind via the information from the positioning telegram stSunblind [▶ 504]. If automatic mode is active (*bManualMode*=FALSE), then the current position is always driven to, wherein changes are immediately accounted for. 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 drive-up time. The position is then in any case "0". Since blind positioning without an encoder is naturally always susceptible to error, it is important to automatically reference as often as possible: each time the "0" position is to be driven to, the blind initially drives upward quite normally with continuous position calculation. Once the calculated position value of 0% has been reached, the output *bBlindUp* is kept active until another complete drive-up time + 5 s has elapsed.
For reasons of flexibility there are now two possibilities to interrupt the referencing procedure: Until the calculated 0% position has been reached, a change of position is still accepted and executed; after reaching this 0% position, the manual command "down" can still move the blind differently. 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 [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 [▸ 504].

VAR_OUTPUT

```
bBlindUp        : BOOL;
bBlindDown      : BOOL;
uiActPosition   : UINT;
bReferencing    : BOOL;
bBusy           : BOOL;
bError          : BOOL;
udiErrorId      : UDINT;
```

bBlindUp: control output for blind up.

bBlindDown: control output for blind down.

uiActPosition: current position in percent.

bReferencing: the blind is in the reference-process, which means, that the output *bBlindUp* will be set for the complete drive-up time + 5 s. Only the manual command "down" can move the blind in the opposite direction an terminate the reference-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 [▸ 318].

VAR_IN_OUT

```
udiTotalTimeUp      : UDINT;
udiTotalTimeDown    : UDINT;
```

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

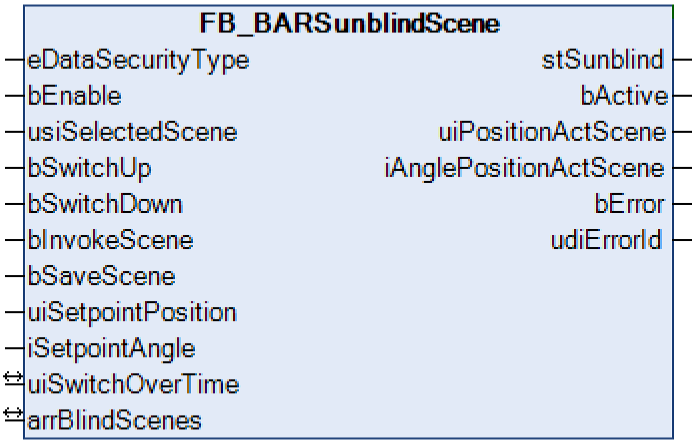
udiTotalTimeUp: Complete time for driving up in ms.

udiTotalTimeDown: Complete time for driving down in ms.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.18 FB_BARSunblindScene



This function block represents an extension of the manual operation FB_BARSunblindSwitch [▸ 292] by a scene memory and a call function. The blind actuator FB_BARSunblindActuator [▸ 273] or the roller shutter actuator FB_BARRollerblind [▸ 287] 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 [▸ 504].

```
TYPE ST_BARSunblind:
STRUCT
     uiPosition       : UINT;
     iAngle           : INT;
     bManUp           : BOOL;
     bManDown         : BOOL;
     bManualMode      : BOOL;
     bActive          : BOOL;
END_STRUCT
END_TYPE
```

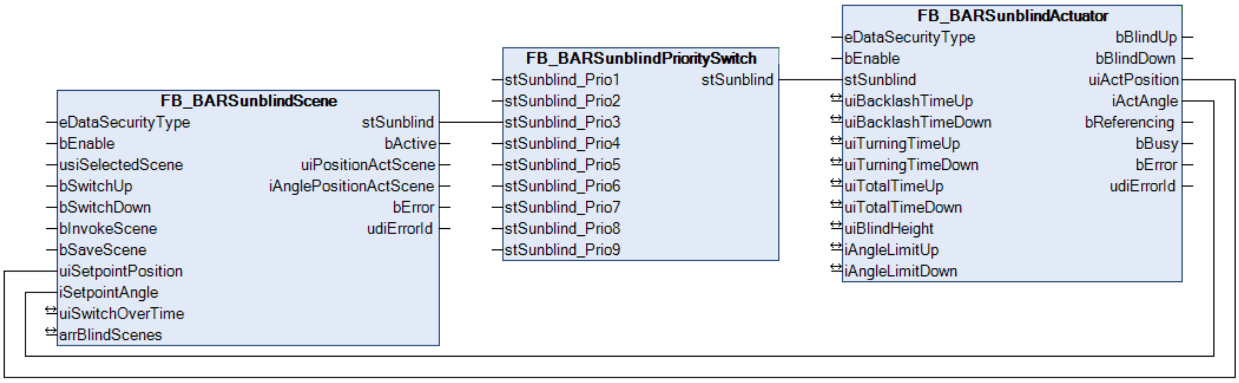
Mode of operation

This function block controls the blind function blocks FB_BARSunblindActuator [▸ 273] and FB_BARRollerBlind [▸ 287] in manual mode via the command inputs *bSwitchUp* and *bSwitchDown*, wherein *bSwitchUp* takes priority. The commands are passed on to the respective commands *bManUp* and *bManDown* in the positioning telegram. If a command input is activated that is longer than the entered time *uiSwitchOverTime* (in ms), then the corresponding control command latches. Activating a command input again releases this latch.
A rising edge on *bSaveScene* saves the current position and louvre angle in the scene selected in *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 driven to.

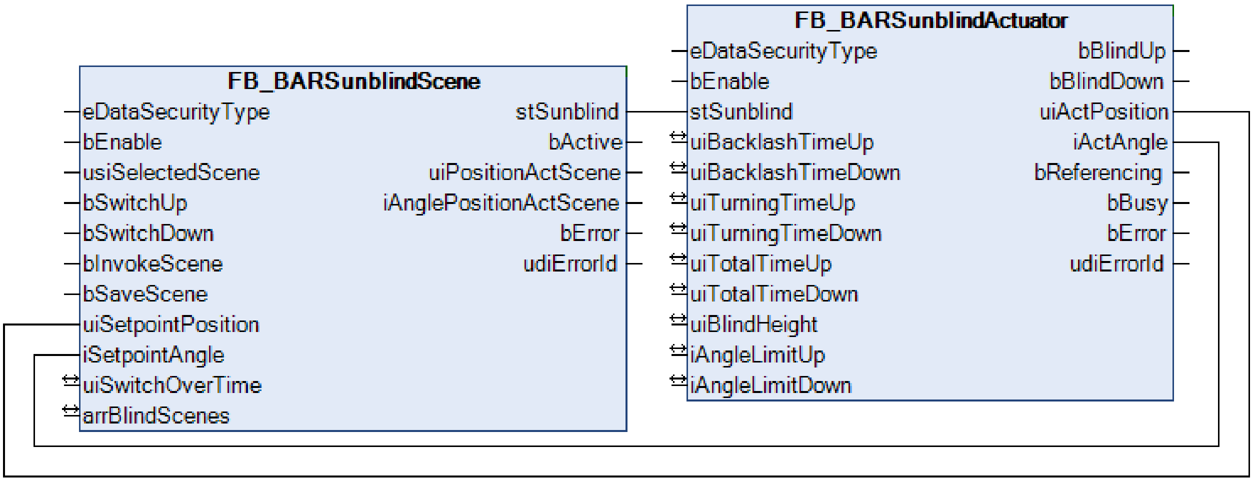
Linking to the blind function block

Like the "normal" manual control function block, FB_BARSunblindSwitch [▸ 292], the scene selection function block can be connected to the blind function block either directly or via an upstream priority controller FB_BARSunblindPrioritySwitch [▸ 286]. The connection is made via the positioning telegram stSunblind [▸ 504]. 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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 both position and angle in the positioning telegram stSunblind [▸ 504] - *bManualMode* and *bActive* are both 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 is to be either stored (*bSaveScene*) or called (*bInvokeScene*).

bSwitchUp: Command input for blind up.

bSwitchDown: Command input for blind down.

bInvokeScene: Call selected scene.

bSaveScene: Save selected scene.

uiSetpointPosition: Setpoint position in % that is to be saved in the selected scene. Must be linked to the actual position of the actuator function block FB_BARSunblindActuator [▸ 273] or FB_BARRollerblind [▸ 287] for the reference blind/roller shutter in order to be able to save a position that was previously driven to manually.

iSetpointAngle: dto. 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 [▸ 504].

bActive : Corresponds to the Boolean value *bActive* in the blind telegram ST_BARSunblind [▸ 504] and serves merely to indicate whether the function block is sending an active telegram.

uiPositionActScene : Indicates the saved relative blind height position in % for the currently selected scene.

iAnglePositionActScene : dto. 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 [▸ 318].

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 [▸ 231]) 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

```
uiSwitchOverTime  : UINT;
arrBlindScenes    : ARRAY[0..20] OF ST_BARSunblindScene;
```

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

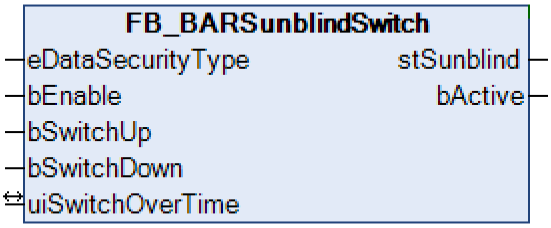
uiSwitchOverTime : Time in milliseconds until the corresponding manual command in the positioning telegram stSunblind [▸ 504] latches in the case of a continuous active command input.

arrBlindScenes : Table with the scene entries of the type ST_BARSunblindScene [▸ 504]. Up to 21 scenes can be saved: 0..20.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.19 FB_BARSunblindSwitch



With the help of this function block the blind controller FB_BARSunblindActuator [▸ 273] or the roller shutter controller FB_BARRollerblind [▸ 287] can be controlled in manual operation mode. The connection takes place via the positioning telegram stSunblind [▸ 504] either directly or with an additional priority controller.

Structure of the blind positioning telegram stSunblind [▸ 504].

```
TYPE ST_BARSunblind:
STRUCT
     uiPosition       : UINT;
     iAngle           : INT;
     bManUp           : BOOL;
     bManDown         : BOOL;
     bManualMode      : BOOL;
     bActive          : BOOL;
END_STRUCT
END_TYPE
```

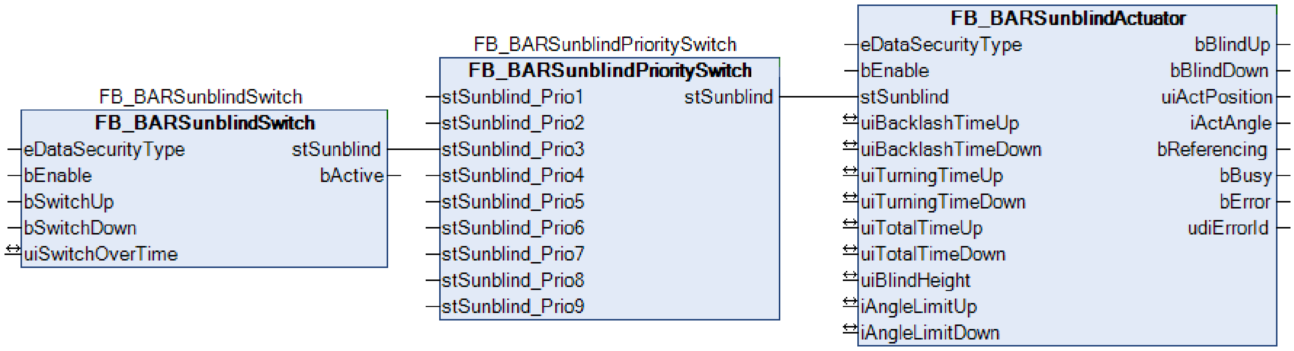
Operation

This function block controls the blind function blocks FB_BARSunblindActuator [▸ 273] and FB_BARRollerBlind [▸ 287] in manual mode via the command inputs *bSwitchUp* and *bSwitchDown*, wherein *bSwitchUp* takes priority. The commands are passed on to the respective commands *bManUp* and *bManDown* in the positioning telegram. If a command input is activated that is longer than the entered time *uiSwitchOverTime* (in ms), then the corresponding control command latches. Activating a command input again releases this latch.

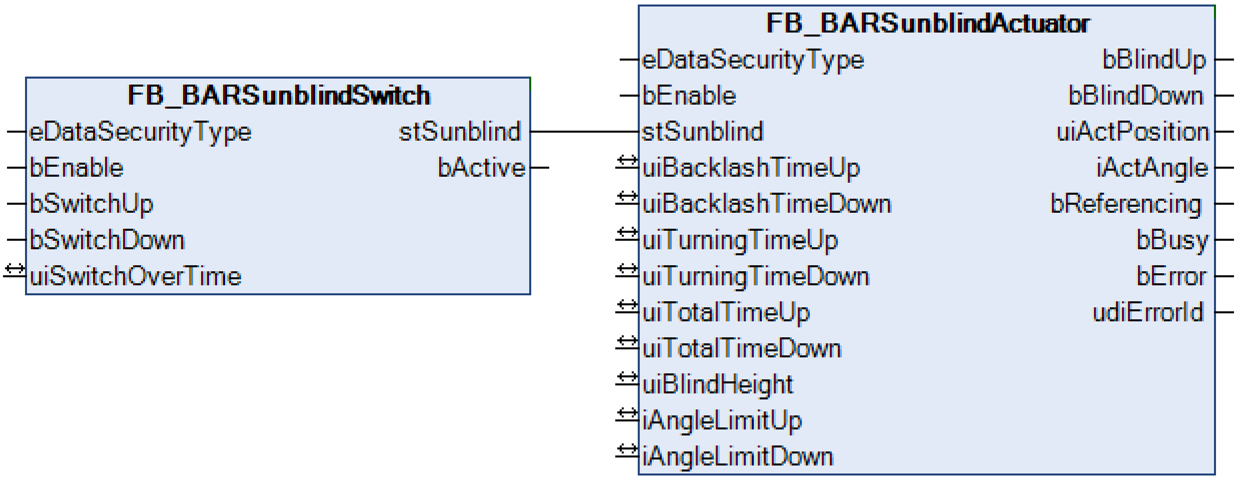
Linking to the blind function block

The manual control function block can be connected to the blind function block either directly or via an upstream priority controller FB_BARSunblindPrioritySwitch [▸ 286]. The connection is made via the positioning telegram stSunblind [▸ 504].

Use of a priority controller:



Direct connection:



VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
bSwitchUp         : BOOL;
bSwitchDown       : BOOL;
```

eDataSecurityType: If eDataSecurityType [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 both position and angle in the positioning telegram stSunblind [▶ 504] - *bManualMode* and *bActive* are both 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 for blind up.

bSwitchDown: Command input for blind down.

VAR_OUTPUT

```
stSunblind : ST_BARSunblind;
bActive    : BOOL;
```

stSunblind: positioning telegram, see ST_BARSunblind [▶ 504].

bActive: corresponds to the Boolean value *bActive* in the blind telegram ST_BARSunblind [▶ 504] and serves merely to indicate whether the function block is sending an active telegram.

VAR_IN_OUT

```
uiSwitchOverTime : UINT;
```

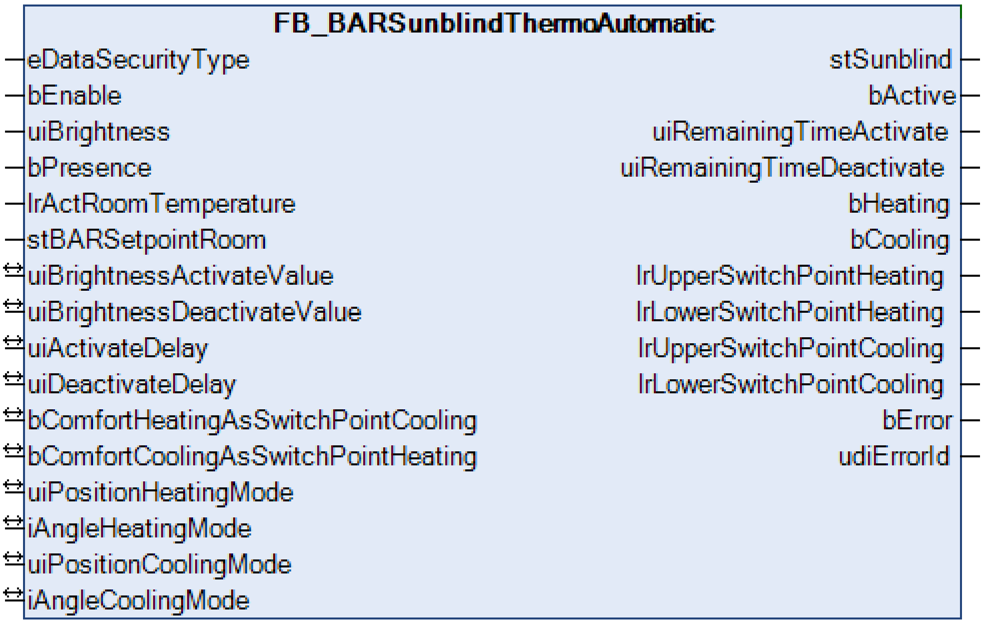
In order for the registered parameters to be retained beyond a controller failure, it is necessary to declare them as In-Out variables. A reference variable is then assigned to them in the program. Each change of the value of these reference variables is persistently stored in the function block and written back to the reference variable following a controller failure and restart. If the parameters were only to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

uiSwitchOverTime : Time in milliseconds until the corresponding manual command in the positioning telegram stSunblind [▶ 504] latches in the case of a continuous active command input.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.20 FB_BARSunblindThermoAutomatic


FB_BARSunblindThermoAutomatic
eDataSecurityType
bEnable
uiBrightness
bPresence
lrActRoomTemperature
stBARSetpointRoom
uiBrightnessActivateValue
uiBrightnessDeactivateValue
uiActivateDelay
uiDeactivateDelay
bComfortHeatingAsSwitchPointCooling
bComfortCoolingAsSwitchPointHeating
uiPositionHeatingMode
iAngleHeatingMode
uiPositionCoolingMode
iAngleCoolingMode
stSunblind
bActive
uiRemainingTimeActivate
uiRemainingTimeDeactivate
bHeating
bCooling
lrUpperSwitchPointHeating
lrLowerSwitchPointHeating
lrUpperSwitchPointCooling
lrLowerSwitchPointCooling
bError
udiErrorId

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 thermo-automatic function 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 three 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*.

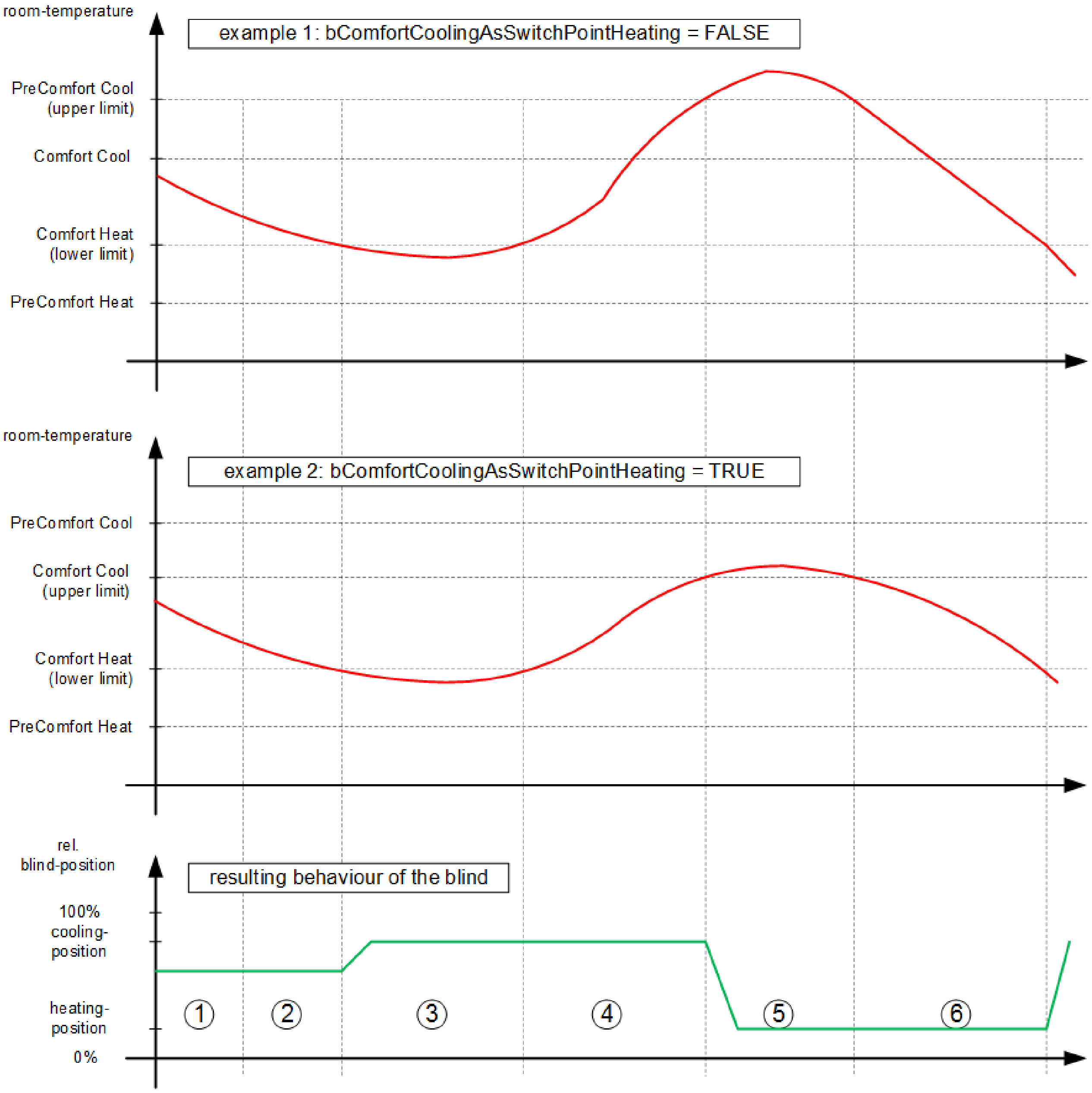
Once the heating or the cooling case has been reached, the thermo-automatic function remembers this until it becomes inactive itself once again due to renewed presence detection or insufficient external brightness (see above).
In both cases, however – heating or cooling – the mechanism operates like a 2-point 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* ; the lower limit value is not selectable:

	bComfortCoolingAsSwitchPointHeating = FALSE	bComfortCoolingAsSwitchPointHeating = TRUE
upper limit value	stBARSetpointRoom_PreComfortCool	stBARSetpointRoom_ComfortCool
lower limit value	stBARSetpointRoom_ComfortHeat	stBARSetpointRoom_ComfortHeat

Diagram



① Initial situation:
The thermo-automatic is not active yet. Another function is contolling 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 an 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.

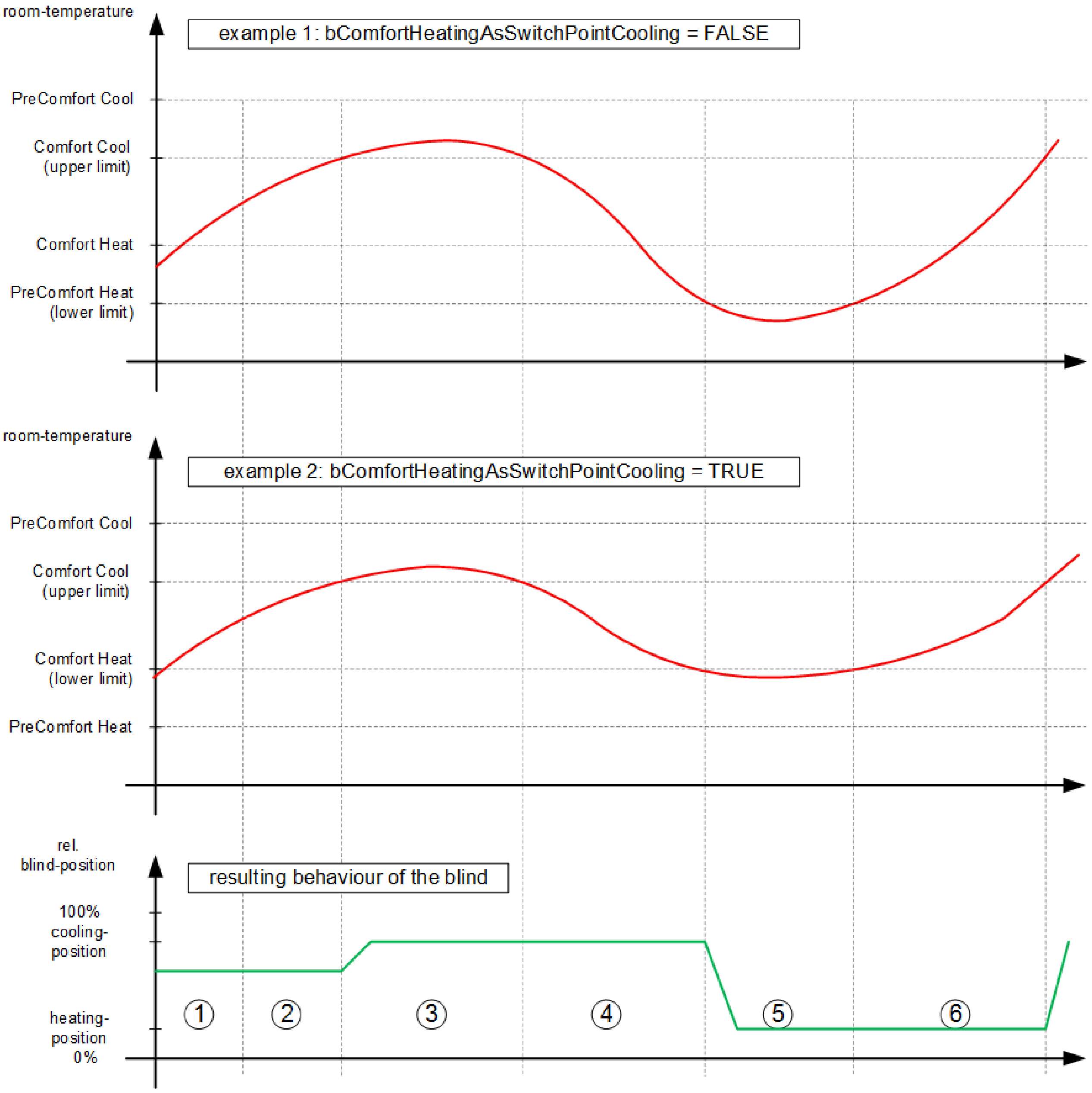
⑥ The blind will stay in the cooling-position until the temperature falls below the lower setpoint (ComfortHeat) again.

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, then the blind drives to the heating position *uiPositionHeatingMode* and *iAngleHeatingMode*.
The lower limit value is selectable by the parameter *bComfortHeatingAsSwitchPointCooling* ; 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 contolling 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 an 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 [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 both position and angle in the positioning telegram stSunblind [▸ 504] and *bActive* is FALSE. This means that another function takes over control of the blind via the priority controller.

uiBrightness: External brightness in lux.

bPresence: The automatic function is active only it the room is unoccupied, since otherwise the sun shading takes priority. If this input is FALSE, the room is considered to be unoccupied and the thermo-automatic function is active. A TRUE signal at *bPresence* deactivates the thermo-automatic function **immediately**.

lrActRoomTemperature: Input for the room temperature in °C.

stBARSetpointRoom: Structure of the type ST_BARSetpointRoom [▸ 294], 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 [▸ 504].

bActive : Corresponds to the Boolean value *bActive* in the blind telegram ST_BARSunblind [▸ 504] and serves merely to indicate whether the function block is sending an active telegram.

uiRemainingTimeActivate: The thermo-automatic function 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 thermo-automatic function 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.

lrUpperSwitchPointHeating: 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).

lrLowerSwitchPointHeating: 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).

lrUpperSwitchPointCooling: Indicates the selected 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).

lrLowerSwitchPointCooling: 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 [▶ 318].

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 [▶ 231]) 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

```
uiBrightnessActivateValue           : UINT;(*lux*)
uiBrightnessDeactivateValue         : UINT;(*lux*)
uiActivateDelay                     : UINT;(*sec*)
uiDeactivateDelay                   : UINT;(*sec*)
bComfortHeatingAsSwitchPointCooling : BOOL;
bComfortCoolingAsSwitchPointHeating : BOOL;
uiPositionHeatingMode               : UINT;
iAngleHeatingMode                   : INT;
uiPositionCoolingMode               : UINT;
iAngleCoolingMode                   : INT;
```

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

uiBrightnessActivateValue / uiActivateDelay : Activation limit value. The thermo-automatic function is active only if the external 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 external 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 : Deactivation limit value of the brightness: If the external brightness *uiBrightness* [lux] falls below the value *uiBrightnessDeactivateValue* [lux] for the time *uiDeactivateDelay* [s], then the automatic function 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 upper 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 for the height position: 0% = fully open, 100% = fully closed.

iAngleHeatingMode : Slat angle of the blind in degrees in the heating case (intended heat irradiation).

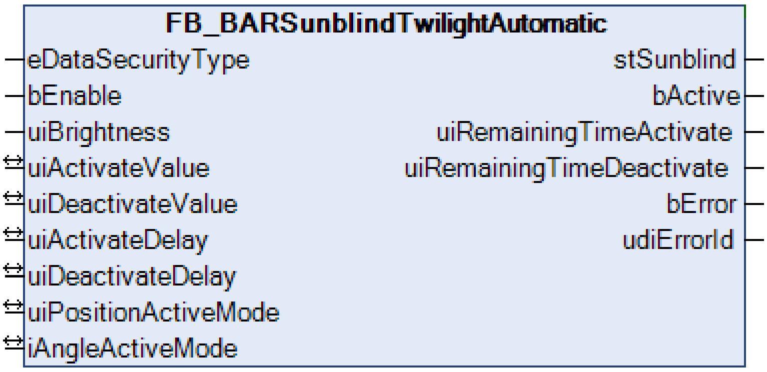
uiPositionCoolingMode : Height position of the blind in % in the cooling case (sun protection). The following applies for the height position: 0% = fully open, 100% = fully closed. The value of the height position must be higher than in the heating case, i.e. the blind must be closed more in the cooling 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).

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.21 FB_BARSunblindTwilightAutomatic



This function block controls the blind if the outdoor brightness has fallen below a limit value.

The automatic twilight function operates with both a value hysteresis and a temporal hysteresis: If the external 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 position telegram stSunblind [▶ 504]. Conversely, if the external 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 [▶ 504] 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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 both position and angle in the positioning telegram stSunblind [▶ 504] and *bActive* is FALSE. This means that another function takes over control of the blind via the priority controller.

uiBrightness: External 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 [▶ 504].

bActive : Corresponds to the Boolean value *bActive* in the blind telegram ST_BARSunblind [▶ 504] and serves merely to indicate whether the function block is sending an active telegram.

uiRemainingTimeActivate: Indicates the time remaining in seconds until the activation of the automatic function after falling below the switch value *usiActivateValue*. This output is 0 as long as no countdown of the time is taking place.

uiRemainingTimeDeactivate: Indicates the time remaining in seconds until the deactivation of the automatic function after the switch value *usiDeactivateValue* has been exceeded. 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 [▶ 318].

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 [▶ 231]) 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

```
uiActivateValue      : UINT;
uiDeactivateValue    : UINT;
uiActivateDelay      : UINT;
uiDeactivateDelay    : UINT;
uiPositionActiveMode : UINT;
iAngleActiveMode     : INT;
```

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 to be defined as input

variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

uiActivateValue: Activation limit value in lux.

uiDeactivateValue: Deactivation limit value in lux.

uiActivateDelay: Activation delay in seconds.

uiDeactivateDelay: Deactivation delay in seconds.

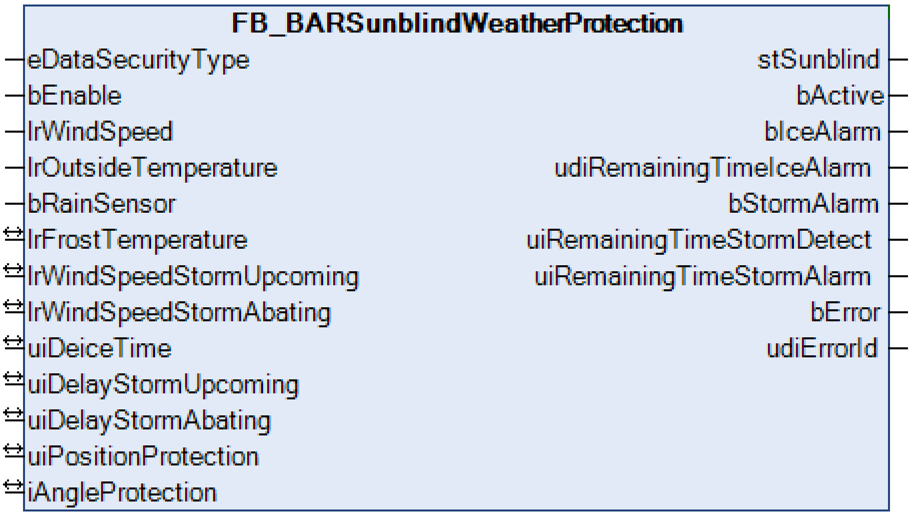
uiPositionActiveMode: Height position of the blind in % if the automatic twilight function is active.

iAngleActiveMode: Slat angle of the blind in degrees if the automatic twilight function is active.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.22 FB_BARSunblindWeatherProtection



The weather protection has the highest priority in the blind controller (see overview [▶ 231]) 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 *lrOutsideTemperature* falls below the frost limit value *lrFrostTemperature* 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 *lrWindSpeedStormUpcoming* for the time *uiDelayStormUpcoming* [s], then it is assumed that a storm is directly impending. Only if the wind speed falls below the value *lrWindSpeedStormAbating* 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 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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 both position and angle in the positioning telegram stSunblind [▶ 504] 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 entry is arbitrary, but it is important that no value is smaller than 0 and that the values become larger with increasing speed.

lrOutsideTemperature: 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 [▶ 504].

bActive : Corresponds to the Boolean value *bActive* in the blind telegram ST_BARSunblind [▶ 504] and serves merely to indicate whether the function block is sending an active telegram.

bIceAlarm: Indicates the icing up alarm.

udiRemainingTimeIceAlarm: 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 (*lrFrostTemperature*), the remaining number of seconds until the 'all-clear' signal is given (*bIceAlarm*=FALSE) are indicated here. This output is 0 as long as no countdown of the time is taking place.

bStormAlarm: Indicates the storm alarm.

uiRemainingTimeStormDetect: In an uncritical case this second counter constantly indicates the alarm delay time *uiDelayStormUpcoming*. If the measured wind force *lrWindSpeed* lies above the activation limit value *lrWindSpeedStormUpcoming*, then the seconds until the alarm is given 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 initiated, this second counter initially constantly indicates the deactivation time delay of the storm alarm *uiDelayStormAbating*. If the measured wind force *lrWindSpeed* falls below the deactivation limit value *lrWindSpeedStormAbating*, then the seconds remaining until the 'all-clear' signal is given (*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 [▶ 318].

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 [▶ 231]) 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

```
lrFrostTemperature        : LREAL;
lrWindSpeedStormUpcoming  : LREAL;
lrWindSpeedStormAbating   : LREAL;
uiDeiceTime               : UINT;
uiDelayStormUpcoming      : UINT;
uiDelayStormAbating       : UINT;
uiPositionProtection      : UINT;
iAngleProtection          : INT;
```

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

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 be not smaller than 0 and must lie above the value for the deactivation. Otherwise an error is output. The unit of entry must be the same as that of the input *lrWindSpeed*. 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 entry must be the same as that of the input *lrWindSpeed*. 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 in seconds until the storm alarm is triggered.

uiDelayStormUpcoming: Time delay in seconds until the storm alarm is reset.

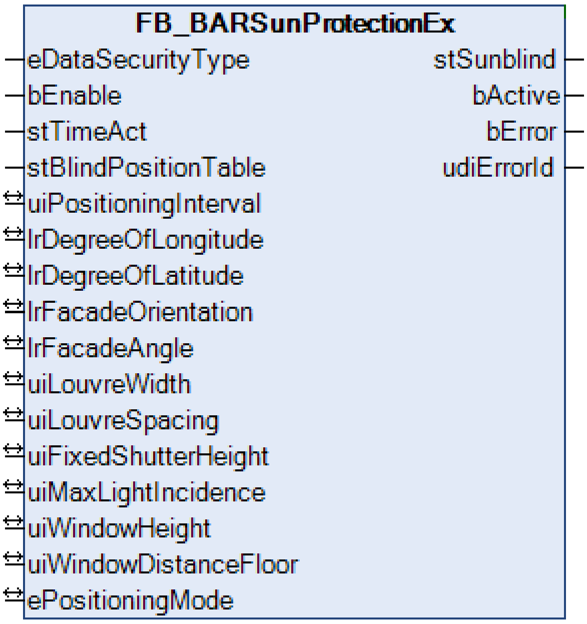
uiPositionProtection: Height position of the blind in % in the case of protection.

iAngleProtection: Slat angle of the blind in degrees in the case of protection.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.23 FB_BARSunProtectionEx



Function block for the control of glare protection with the aid of a slatted blind.

The function block FB_BARSunProtectionEx enables glare protection in two ways that operate in parallel:

- Slat adjustment, so that the direct incident light cannot quite enter through the lowered part of the blind.
- Control of the shutter height, with three different options (adjustable via the enumerator on *ePositioningMode*):
  1) Fixed shutter height, i.e. no change (preset)
  2) Shutter height in relation to the position of the sun, defined via a table (stBlindPosTable [▶ 502]), see also description of the function block FB_BARBlindPositionEntry [▶ 248]
  3) Maximum desired incidence of light

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 [▶ 504]. Of course, the output does not take place continuously, since a constant blind movement would be perceived as extremely distracting. The interval in seconds at which new position values are to be output can be set on the input *uiPositioningInterval*.
However, the shading criteria must always be fulfilled between two positioning times: no direct light may enter through the slats and the desired incidence of light must remain limited by the shutter height, if one initially assumes that the shutter height is controlled via the mode "maximum desired incidence of light". Therefore two blind and slat positions are internally calculated: those for the current switching point and those for the next. 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 actually be done by the user by programming the input *bEnable*; see Overview of automatic sun protection (shading correction) [▶ 233].

If these three conditions are not satisfied, then the active bit (*bActive*) is set to FALSE, the shutter 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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= eHVACDataSecurityType_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 [▶ 504] 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 [▶ 501]). See FB_BARBlindPositionEntry [▶ 248] for further description.

VAR_OUTPUT

```
stSunblind    : ST_BARSunblind;
bError        : BOOL;
udiErrorId    : UDINT;
```

stSunblind: Output structure of the blind positions, see ST_BARSunblind [▶ 504]

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 [▶ 318].

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 [▶ 231]) 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

```
uiPositioningInterval  : UINT;
lrDegreeOfLongitude    : LREAL;
lrDegreeOfLatitude     : LREAL;
lrFacadeOrientation    : LREAL;
lrFacadeAngle          : LREAL;
uiLouvreWidth          : UINT;
uiLouvreSpacing        : UINT;
uiFixedShutterHeight   : UINT;
uiMaxLightIncidence    : UINT;
uiWindowHeight         : UINT;
uiWindowDistanceFloor  : UINT;
ePositioningMode       : E_BARPOSMODE;
```

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

uiPositioningInterval: Positioning interval in seconds – time interval between two outputs of blind positions. 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	β=0°
East	β=90°
South	β=180°
West	β=270°

The following applies for the southern hemisphere:

Line of sight	Facade orientation
South	β=0°
East	β=90°
North	β=180°
West	β=270°

lrFacadeAngle: Facade inclination in degrees. See facade inclination.

uiLouvreWidth: Width of the slats in mm, see sketch.

uiLouvreSpacing: Slat spacing in mm, see sketch.

uiFixedShutterHeight: Fixed (constant) shutter height [0..100%]. Valid if *ePositioningMode* = *ePosModeFixed* (see enumerator E_BARPosMode [▸ 501]).

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 [▸ 501]).

uiWindowHeight: Window height in mm for the calculation of the shutter height if the mode "maximum desired incidence of light" is selected.

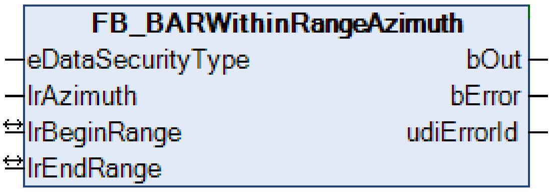
uiWindowDistanceFloor: Distance between the floor and the window sill in mm for the calculation of the shutter height if the mode "maximum desired incidence of light" is selected.

ePositioningMode: Selection of the positioning mode, see enumerator E_BARPosMode [▸ 501].

Requirements

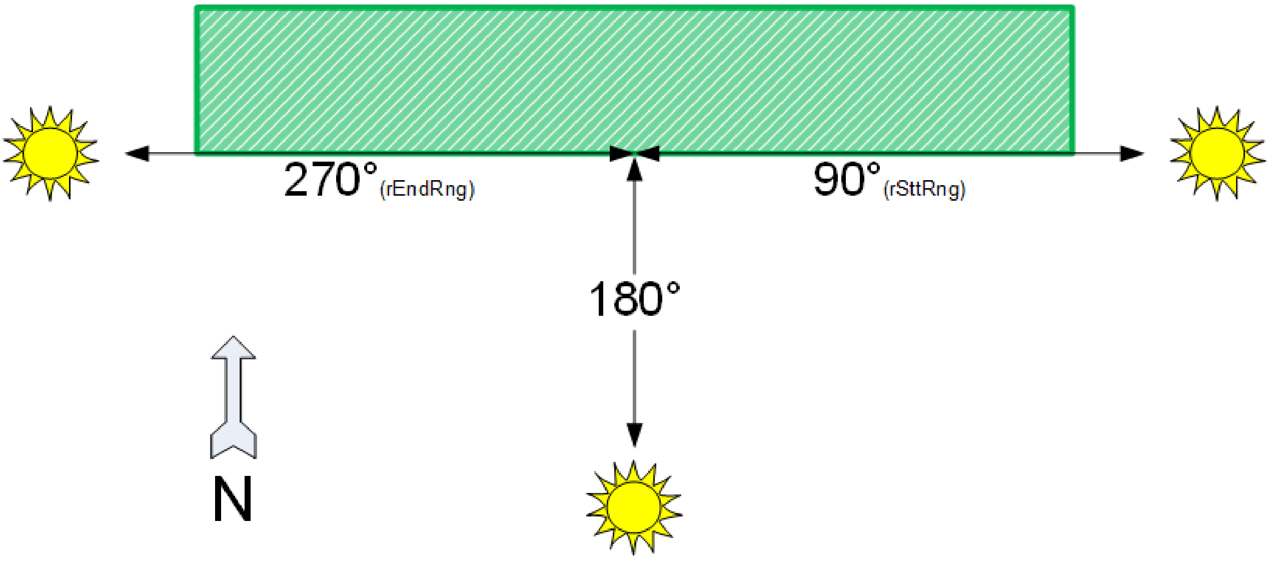
Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.24 FB_BARWithinRangeAzimuth

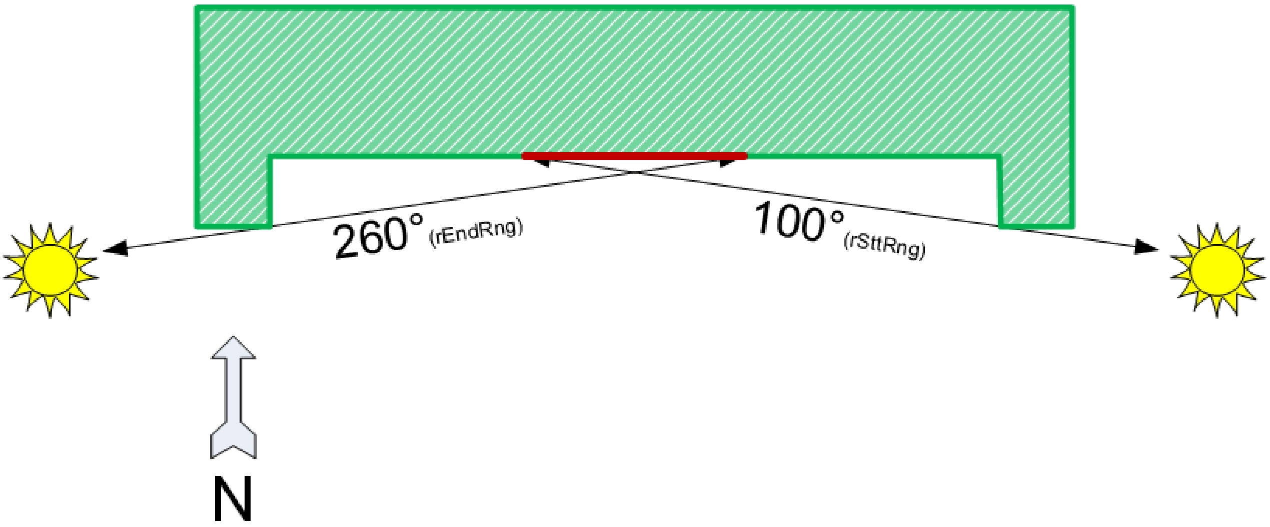


This function block checks whether the current azimuth angle (horizontal position of the sun) lies within the limits entered. As can be seen in the overview [▸ 233], the function block provides an additionally evaluation as to whether the sun shading of a window group should be activated. Therefore the observations in the remainder of the text always apply to one window group.

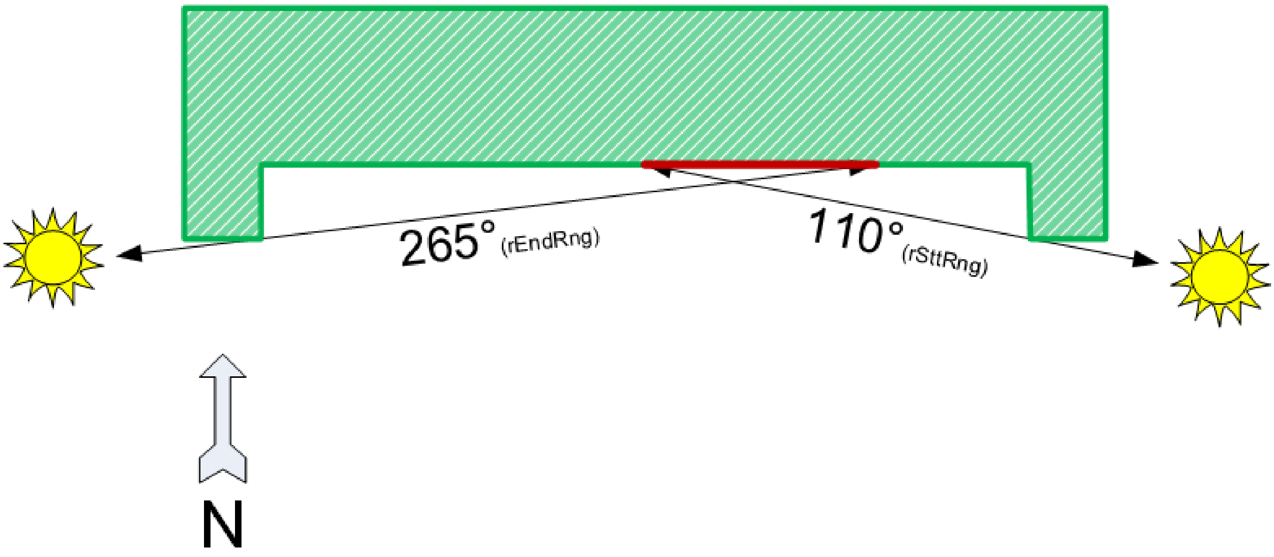
A smooth facade is always irradiated by the sun at an azimuth angle of *Facade orientation-90°* to Facade orientation+90°.



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 *lrBeginRange* may thereby be larger than the end *lrEndRange*; it is then regarded beyond 0°:

Example:

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 [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.

lrAzimuth: 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 [▶ 318].

VAR_IN_OUT

```
lrBeginRange : LREAL;
lrEndRange   : LREAL;
```

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

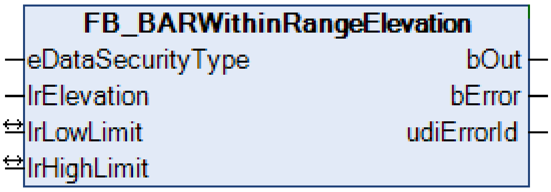
lrBeginRange: Beginning of the range in degrees.

lrEndRange: End of the range in degrees.

Requirements

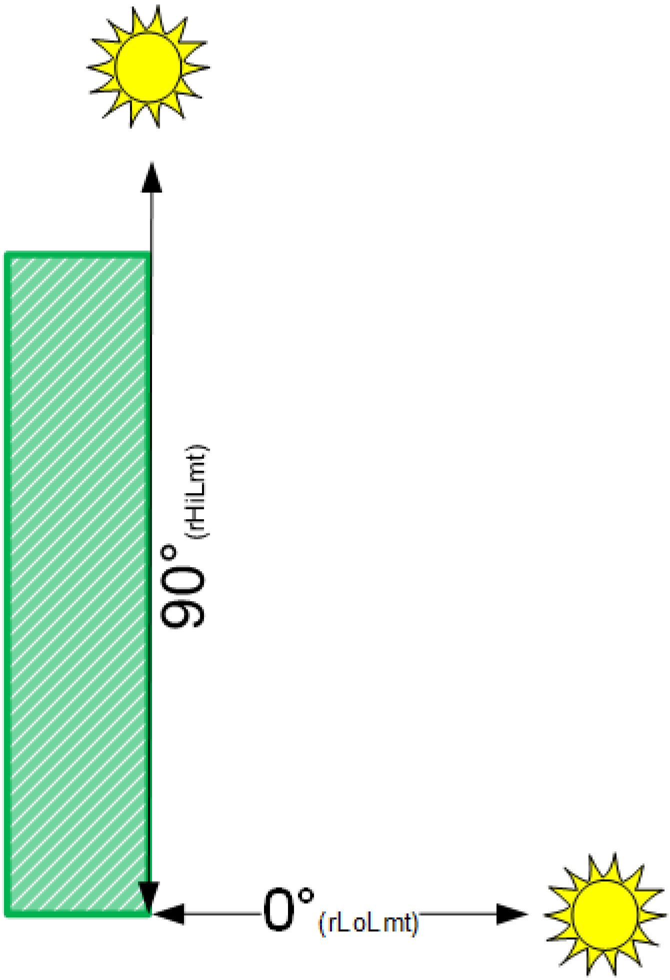
Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.25 FB_BARWithinRangeElevation

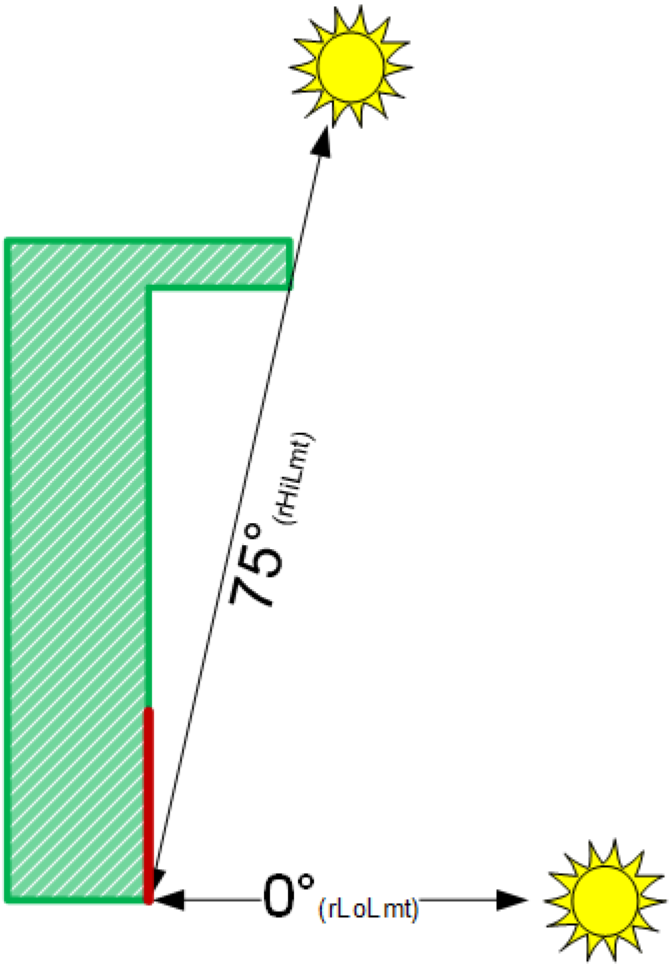


This function block checks whether the current angle of elevation (vertical position of the sun) lies within the limits entered. As can be seen in the overview [▶ 233], the function block provides an additionally evaluation as to whether the sun shading of a window group should be activated. Therefore the observations in the remainder of the text always apply to one window group.

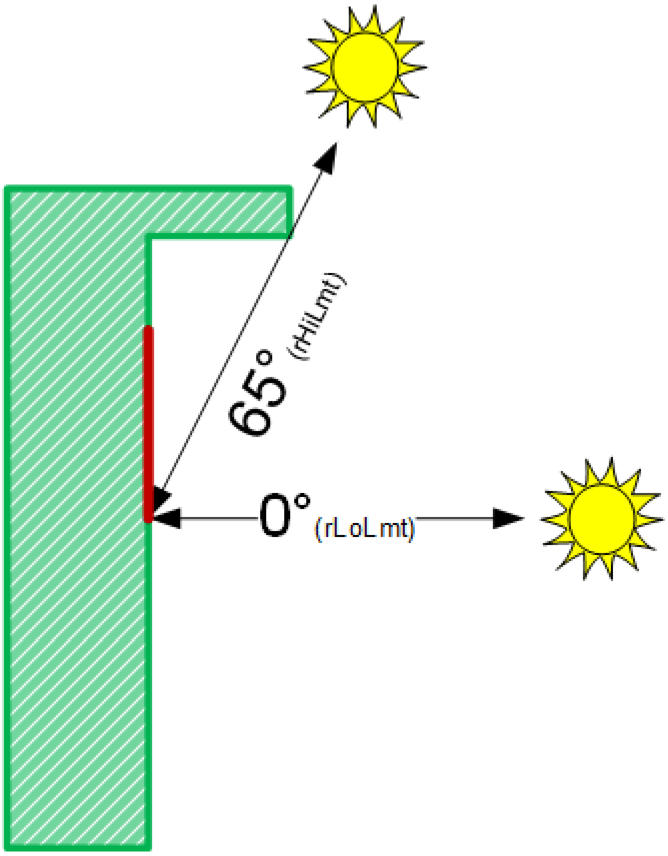
A normal vertical facade is irradiated by the sun at an angle of elevation of 0° to maximally 90°.



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, *lrLowLimit*, may thereby not be greater than or equal to the upper limit, *lrHighLimit*. 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 [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.

lrElevation: Current angle of elevation 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 [▸ 318].

VAR_IN_OUT

```
lrLowLimit   : LREAL;
lrHighLimit  : LREAL;
```

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

lrLowLimit: Lower limit value in degrees.

lrHighLimit: Upper limit value in degrees.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.26 Program example

5.1.5.4.26.1 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.

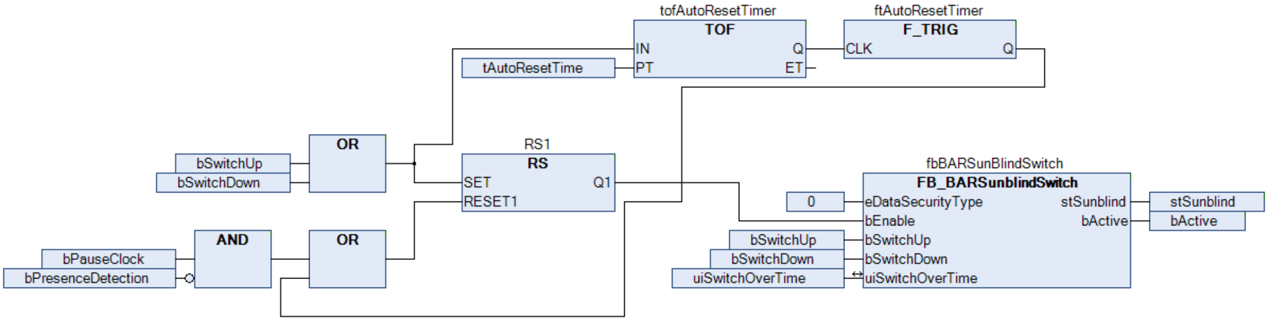
bPauseClock : Pause signal from the property, e.g. lunch break: 12:00 - 13:00. *bPauseClock* = TRUE: Pause.

bPresenceDetection : Occupied signal from the room in which the blinds are controlled. *bPresenceDetection***=** TRUE: Room is occupied.

uiSwitchOverTime : Time in milliseconds until the corresponding manual command latches with a continuous command input.

stSunblind : Positioning telegram, see ST_BARSunblind [▶ 504]. For further linking to a priority selection, see FB_BARSunblindPrioritySwitch [▶ 286].

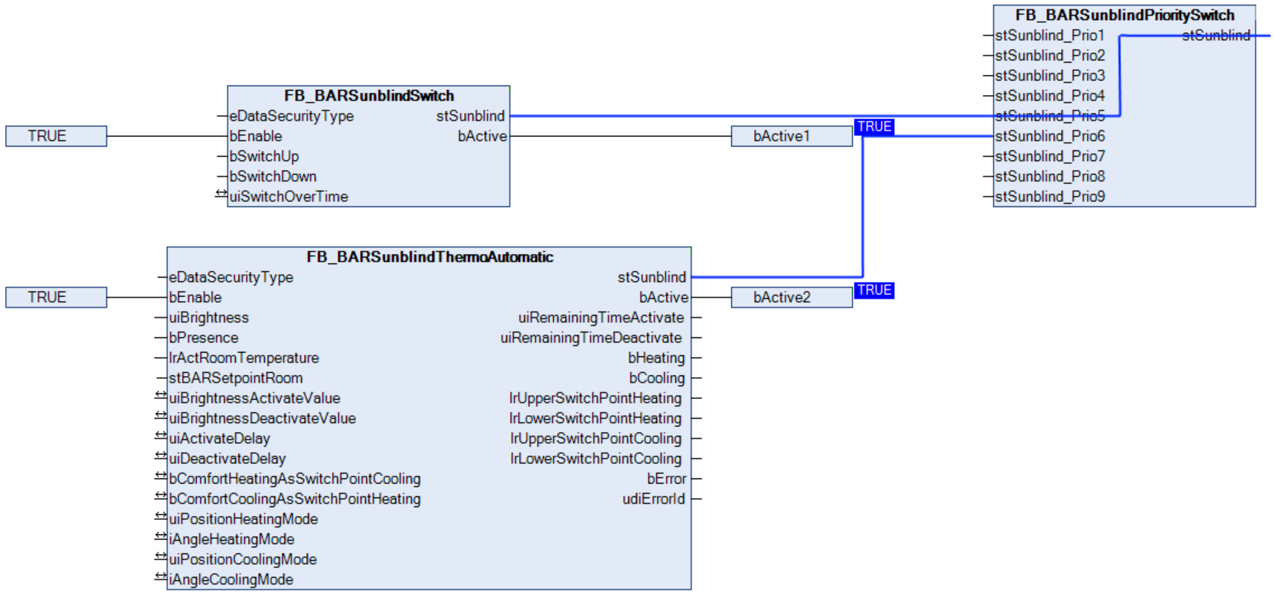
SwitchWithResetEnable:

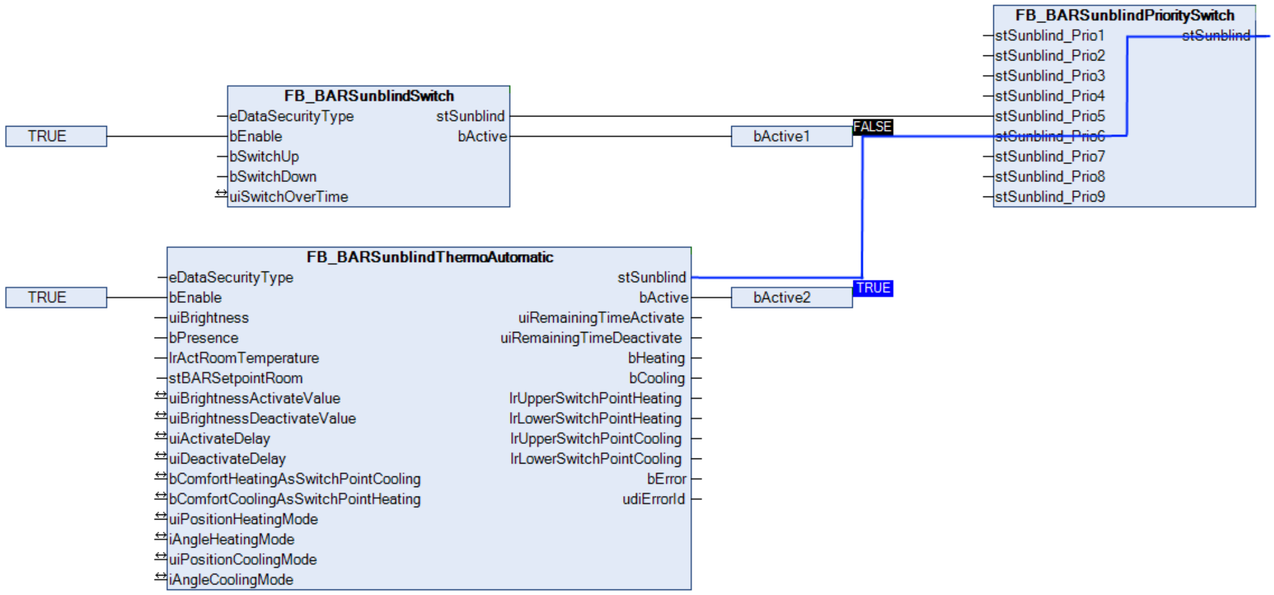


Functioning

If one of the blind pushbuttons *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 switch signal that has led to the activation can be passed on directly by the function block in the command telegram *stSunblind*.
At the same time a TRUE signal is applied to the input of the switch-off delay *tofAutoResetTimer* and its output *Q* is set directly 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 [▶ 286], then the telegram will be passed through there to the next priority:





Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.27 List descriptions

5.1.5.4.27.1 List of facade elements

The data of all windows (facade elements) per facade are saved within the program in a field of structure elements of the type ST_BARFacadeElement [▶ 502].
The declaration is global, since the administration function block FB_BARFacadeElementEntry [▶ 252] as well as the shading correction FB_BARShadingCorrection [▶ 264] / FB_BARShadingCorrectionSouth [▶ 267] access this field directly by 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
```

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.27.2 List of shading elements

The shading elements per facade are saved within the program in a field of structure elements of the type ST_BARShadingObject [▶ 503].
The declaration is global, since the administration function block FB_BARShadingObjectsEntry [▶ 270] as well as the shading correction FB_BARShadingCorrection [▶ 264] / FB_BARShadingCorrectionSouth [▶ 267] access this field directly by 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
```

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.5.4.27.3 Error Codes

udiErrorId (hex)	udiErrorId (dec)	Error
0x0	0	No error
0x8001	32769	FB_BARBlindPositionEntry [▶ 248]: Values for *lrSunElevation* are not ascending or unequal
0x8002	32770	FB_BARBlindPositionEntry [▶ 248]: Values for *lrSunElevation* are not within the valid range from 0°..90°
0x8003	32771	FB_BARBlindPositionEntry [▶ 248]: Values for *uiBlindPosition* are not within the valid range from 0%..100%
0x8004	32772	FB_BARSunProtectionEx [▶ 306]: The duration of the positioning interval is zero or exceeds 7200 s.
0x8005	32773	FB_BARSunProtectionEx [▶ 306]: The longitude entered is not within the valid range from -180°..180°.
0x8006	32774	FB_BARSunProtectionEx [▶ 306]: The latitude entered is not within the valid range from -90°..90°.
0x8007	32775	FB_BARSunProtectionEx [▶ 306]: The value for the slat spacing (*uiLouvreSpacing*) is greater than or equal to the value for the slat width (*uiLouvreWidth*).
0x8008	32776	FB_BARSunProtectionEx [▶ 306]: The "Values valid" bit (*bValid*) in the position table *stBlindPositionTable* is not set - invalid values, see FB_BARBlindPositionEntry [▶ 248].
0x8009	32777	FB_BARSunProtectionEx [▶ 306]: The value entered for the fixed shutter height (*uiFixedShutterHeight*) exceeds 100%
0x801A	32778	FB_BARSunProtectionEx [▶ 306]: The value entered for the window height is zero
0x801B	32779	FB_BARSunProtectionEx [▶ 306]: The value entered for the slat spacing is zero
0x801C	32780	FB_BARSunProtectionEx [▶ 306]: The value entered for the slat width is zero
0x801D	32781	FB_BARShadingObjectsEntry [▶ 270]: 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 [▶ 270]: The corners of the square are not on the same level.
0x801F	32783	FB_BARShadingObjectsEntry [▶ 270]: The z-component of P1 is less than zero. This corner would thus lie behind the facade.
0x8010	32784	FB_BARShadingObjectsEntry [▶ 270]: The z-component of P3 is less than zero. This corner would thus lie behind the facade.
0x8011	32785	FB_BARShadingObjectsEntry [▶ 270]: P1 is equal to P2. The object entered is thus not a rectangle.
0x8012	32786	FB_BARShadingObjectsEntry [▶ 270]: P1 is equal to P3. The object entered is thus not a rectangle.
0x8013	32787	FB_BARShadingObjectsEntry [▶ 270]: P1 is equal to P4. The object entered is thus not a rectangle.
0x8014	32788	FB_BARShadingObjectsEntry [▶ 270]: P2 is equal to P3. The object entered is thus not a rectangle.
0x8015	32789	FB_BARShadingObjectsEntry [▶ 270]: P2 is equal to P4. The object entered is thus not a rectangle.
0x8016	32790	FB_BARShadingObjectsEntry [▶ 270]: P3 is equal to P4. The object entered is thus not a rectangle.
0x8017	32791	FB_BARShadingObjectsEntry [▶ 270]: The radius entered is zero
0x8018	32792	FB_BARShadingObjectsEntry [▶ 270]: The z-component of the ball center is less than zero. This point would thus lie behind the facade.
0x8019	32793	FB_BARFacadeElementEntry [▶ 252]: 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 [▶ 252]: The X-component of the first corner (Corner1) is less than zero.
0x801B	32795	FB_BARFacadeElementEntry [▶ 252]: The Y-component of the first corner (Corner1) is less than zero.
0x801C	32796	FB_BARFacadeElementEntry [▶ 252]: The window width is less than or equal to zero.
0x801D	32797	FB_BARFacadeElementEntry [▶ 252]: The window height is less than or equal to zero.
0x801E	32798	FB_BARDelayedHysteresis [▶ 251]: The switch-on value *bOnValue* is smaller than the switch-off value *bOffValue*.
0x801F	32799	FB_BARWithinRangeAzimuth [▶ 309]: The beginning of the range *lrBeginRange* and the end of the range *lrEndRange* are the same.
0x8020	32800	FB_BARWithinRangeAzimuth [▶ 309]: One of the range limits entered is greater than 360° or less than 0°

udiErrorId (hex)	udiErrorId (dec)	Error
0x8021	32801	FB_BARWithinRangeAzimuth [▸ 309]: The range entered is greater than 180°.
0x8022	32802	FB_BARWithinRangeElevation [▸ 311]: The lower limit value *lrLowLimit* entered is greater than or equal to the upper limit value *lrHighLimit*.
0x8023	32803	FB_BARWithinRangeElevation [▸ 311]: The upper limit value *lrHighLimit* is greater than 90° or the lower limit value *lrLowLimit* is less than 0°.
0x8024	32804	FB_BARShadingObjectsEntry [▸ 270]: The number of the month when shading begins or when shading ends is zero or greater than 12.
0x8025	32805	FB_BARShadingObjectsEntry [▸ 270]: Index error! iId lies outside the permissible limits *1..iShadingObjects*. See list of shading elements [▸ 317].
0x8026	32806	FB_BARSunblindActuator [▸ 273]/FB_BARSunblindActuatorEx [▸ 279]: The shutter height entered is 0.
0x8027	32807	FB_BARSunblindActuator [▸ 273]/FB_BARSunblindActuatorEx [▸ 279]: The total "up" stroke time (*uiTotalTimeUp/udiTotalTimeUp*) or the total "down" stroke time (*uiTotalTimeDown/ udiTotalTimeDown*) respectively is 0 ms.
0x8028	30808	FB_BARSunblindActuator [▸ 273]/FB_BARSunblindActuatorEx [▸ 279]: the slat "Up" turning time (*uiTurningTimeUp*) or the slat "Down" turning time (*uiTurningTimeDown*) is 0 ms.
0x8029	32809	FB_BARSunblindActuator [▸ 273]/FB_BARSunblindActuatorEx [▸ 279]: The upper slat angle limit value (*iAngleLimitUp*) is smaller than or equal to the lower slat angle limit value (*iAngleLimitDown*)
0x802A	32810	FB_BARSunblindScene [▸ 289]: The selected scene number exceeds the maximum number of 20.
0x802B	32811	FB_BARSunblindThermoAutomatic [▸ 294]: The preset room temperature values are in the wrong order.
0x802C	32812	FB_BARSunblindThermoAutomatic [▸ 294]: The brightness activation limit value is smaller than or equal to the deactivation limit value.
0x802D	32813	FB_BARSunblindThermoAutomatic [▸ 294]: The blind position (height in %) of the heating mode is greater than or equal to that of the cooling mode.
0x802E	32814	FB_BARSunblindTwilightAutomatic [▸ 294]: The brightness limit value for the activation of the automatic twilight function is greater than or equal to the deactivation value.
0x802F	32815	FB_BARSunblindWeatherProtection [▸ 303]: The frost limit value is greater than 10 °C.
0x8030	32816	FB_BARSunblindWeatherProtection [▸ 303]: The limit value for the activation of the storm alarm is smaller than or equal to the deactivation value.
0x8031	32817	FB_BARSunblindWeatherProtection [▸ 303]: The input of wind force limit values of less than 0 is not permissible.
0x8032	32818	FB_BARShadingCorrection [▸ 264] / FB_BARShadingCorrectionSouth [▸ 267]: 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 [▸ 264] / FB_BARShadingCorrectionSouth [▸ 267]: A window element in the selected group is marked as invalid (bValid=FALSE).
0x8034	32820	FB_BARFacadeElementEntry [▸ 252]: Index error! iColumn or iRow lies outside the permissible limits *1..iColumnsPerFacade* or *1..iRowsPerFacade*. respectively. See list of facade elements [▸ 316].
0x8035	32821	FB_BARSunProtectionEx [▸ 306]: The façade inclination entered (lrFacadeAngle) is not within the valid range of -90°..90°.
0x8036	32822	FB_BARSunblindActuator [▸ 273]/FB_BARSunblindActuatorEx [▸ 279]: The time for extending the backlash (*uiBacklashTimeUp*) or (*uiBacklashTimeDown*) respectively is 0 ms.
0x8037	32823	FB_BARRollerBlind [▸ 287]: The entire "Up" travel time (*udiTotalTimeUp*) or the total "Down" travel time (*udiTotalTimeDown*) is 0 ms.
0x8038	32824	FB_BARSMISunblindActuator: The value for *wLouvreRange* may not be 0.
0x8039	32825	FB_BARSMISunblindActuator: The ratio *wLouvreRange* / (*iAngleLimitUp* + *iAngleLimitDown*) must be greater than 0.
0x803A.. 0x80FF	32826.. 33023	reserved for shading function blocks
0x8100	33024	FB_BARLightCircuit [▸ 220]: The operation mode *uiLightCtrlMode* is greater than 2. However, only 0, 1 or 2 is permitted.
0x8101	33025	FB_BARLightCircuitDim [▸ 222]: The operation mode *uiLightCtrlMode* is greater than 2. However, only 0, 1 or 2 is permitted.
0x8102	33026	FB_BARLightCircuitDim [▸ 222]: The lower limit value, *lrMinDimValue*, is less than zero or greater than 100%.

udiErrorId (hex)	udiErrorId (dec)	Error
0x8103	33027	FB_BARLightCircuitDim [▶ 222]: The upper limit value, *lrMaxDimValue*, is less than zero or greater than 100%.
0x8104	33028	FB_BARLightCircuitDim [▶ 222]: The lower limit value, *lrMinDimValue*, is greater than the upper limit value, *lrMaxDimValue*.
0x8105	33029	FB_BARLightCircuitDim [▶ 222]: The value for "Manual On", *lrManualDimValue*, is less than zero or greater than 100%.
0x8106	33030	FB_BARAutomaticLight [▶ 204]: The operation mode *uiLightCtrlMode* is greater than 2. However, only 0, 1 or 2 is permitted.
0x8107	33031	FB_BARAutomaticLight [▶ 204]: The value for "Manual On", *lrManualDimValue*, is less than zero or greater than 100%.
0x8108	33032	FB_BARAutomaticLight [▶ 204]: The light switch-on value, *lrOnDimValue*, is less than zero or greater than 100%.
0x8109	33033	FB_BARStairwellAutomatic [▶ 225]: The operation mode *uiLightCtrlMode* is greater than 2. However, only 0, 1 or 2 is permitted.
0x810A	33034	FB_BARTwilightAutomatic [▶ 228]: The operation mode *uiLightCtrlMode* is greater than 2. However, only 0, 1 or 2 is permitted.
0x810B	33035	FB_BARTwilightAutomatic [▶ 228]: The value for "Manual On", *lrManualDimValue*, is less than zero or greater than 100%.
0x810C	33036	FB_BARTwilightAutomatic [▶ 228]: The ON value for the automatic function, *lrDimOnValue*, is less than zero or greater than 100%.
0x810D	33037	FB_BARTwilightAutomatic [▶ 228]: The OFF value for the automatic function, *lrDimOffValue*, is less than zero or greater than 100%.
0x810E	33038	FB_BARTwilightAutomatic [▶ 228]: The switch-on threshold value, *uiSwitchOnValue*, is greater than or equal to the switch-off threshold value, *uiSwitchOffValue*.
0x810F	33039	FB_BARDaylightControl [▶ 213]: The operation mode *uiLightCtrlMode* is greater than 2. However, only 0, 1 or 2 is permitted.
0x8110	33040	FB_BARConstantLightControl [▶ 207]: The operation mode *uiLightCtrlMode* is greater than 2. However, only 0, 1 or 2 is permitted.
0x8111	33041	FB_BARConstantLightControl [▶ 207]: The lower limit value, *lrMinDimValue*, is less than zero or greater than 100%.
0x8112	33042	FB_BARConstantLightControl [▶ 207]: The upper limit value, *lrMaxDimValue*, is less than zero or greater than 100%.
0x8113	33043	FB_BARConstantLightControl [▶ 207]: The lower limit value, *lrMinDimValue*, is greater than the upper limit value, *lrMaxDimValue*.
0x8114	33044	FB_BARConstantLightControl [▶ 207]: The starting value of the controller, lrStartDimValue, is less than zero or greater than 100%.
0x8115	33045	FB_BARConstantLightControl [▶ 207]: The value for "Manual On", *lrManualDimValue*, is less than zero or greater than 100%.
0x8116.. 0x81FF	33046.. 33279	reserved for light function blocks
0x8200	33280	FB_BARPICtrl [▶ 198]: The execution cycle number *uiCtrlCycleCall* is zero
0x8201	33281	FB_BARPICtrl [▶ 198]: The proportional band *rXp* is zero.
0x8202	33282	FB_BARPICtrl [▶ 198]: The lower control value limit value *rYMin* is greater than or equal to the upper limit value *rYMax*.
0x8203	33283	FB_BARPICtrl [▶ 198]: The task cycle time *tTaskCycleTime* is set to 0.
0x8204	33284	FB_HVACOptimizedOn [▶ 423]: The outside temperature values *lrOutsideTemp[i]* of the pre-start function *stTempChangeFunction* are not in ascending order or 2 values are the same.
0x8205	33285	FB_HVACOptimizedOn [▶ 423]: One or more values of the room temperature change *lrRoomTempChange[i]* in the pre-start function *stTempChangeFunction* are higher than the upper limit value *lrMaxTempChange*.
0x8206	33286	FB_HVACOptimizedOn [▶ 423]: One or more values of the room temperature change *lrRoomTempChange[i]* in the pre-start function *stTempChangeFunction* are lower than the lower limit value *lrMinTempChange*.
0x8207	33287	FB_HVACOptimizedOn [▶ 423]: The control input *eCtrlFct* is set neither to heating nor to cooling mode (*eBARCtrlFct_Heating* or *eBARCtrlFct_Cooling*).
0x8208	33288	FB_HVACOptimizedOff [▶ 433]: The outside temperature values *lrOutsideTemp[i]* of the pre-start function *stTempChangeFunction* are not in ascending order or 2 values are the same.
0x8209	33289	FB_HVACOptimizedOff [▶ 433]: One or more values of the room temperature change *lrRoomTempChange[i]* in the pre-start function *stTempChangeFunction* are higher than the upper limit value *lrMaxTempChange*.

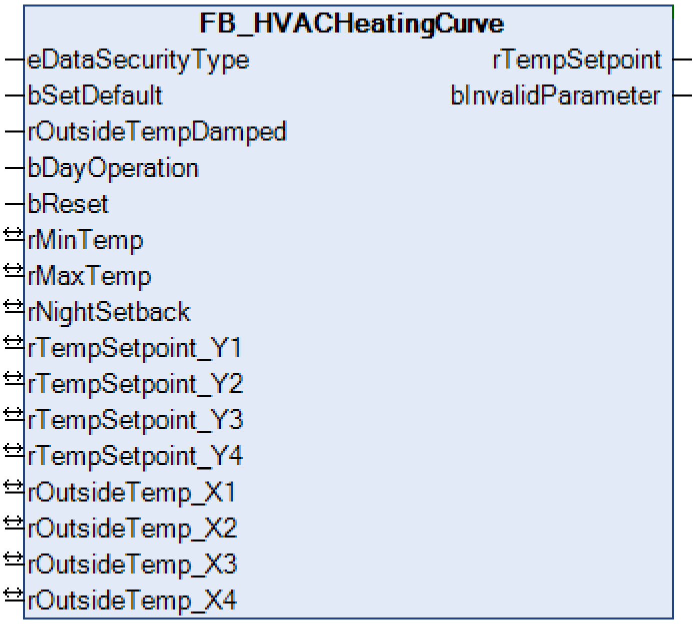
udiErrorId (hex)	udiErrorId (dec)	Error
0x820A	33290	FB_HVACOptimizedOff [▶ 433]: One or more values of the room temperature change *lrRoomTempChange[i]* in the pre-start function *stTempChangeFunction* are lower than the lower limit value *lrMinTempChange*.
0x820B	33291	FB_HVACOptimizedOff [▶ 433]: The control input *eCtrlFct* is set neither to heating nor to cooling mode (*eBARCtrlFct_Heating* or *eBARCtrlFct_Cooling*).
0x820C	33292	FB_HVACTempChangeFunctionEntry [▶ 443]: The outside temperature values *lrOutsideTemp[i]* of the pre-start function *stTempChangeFunction* are not in ascending order or 2 values are the same.
0x8C00.. 0x8CFF	35840.. 36095	See SMI error codes Subtract the value 0x0C00 from the error code to obtain the SMI-specific error code.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.6 HVAC Setpoint modules

5.1.6.1 FB_HVACHeatingCurve



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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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;
bInvalidParameter : BOOL;
```

rTempSetpoint: The calculated setpoint for the flow temperature (0°C..500°C).

bInvalidParameter: Indicates that there is an incorrect input parameter. *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.

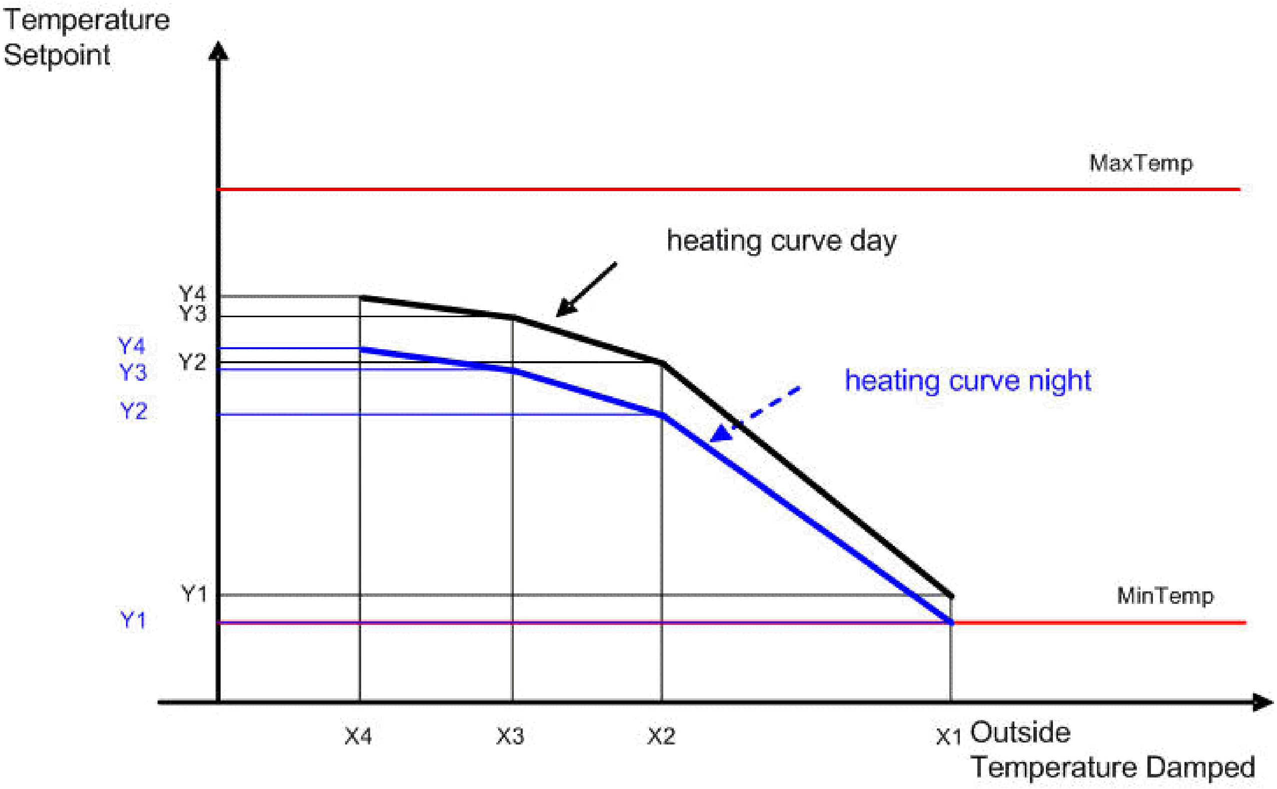
rTempSetpoint_Y1 / rOutsideTemp_X1: The course of point 1 of the heating curve is parameterized by this value pair. The variable is saved persistently. Preset to 20.

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 > X 2 > X3 > X4 and Y1 < Y2 < Y3 <Y4.

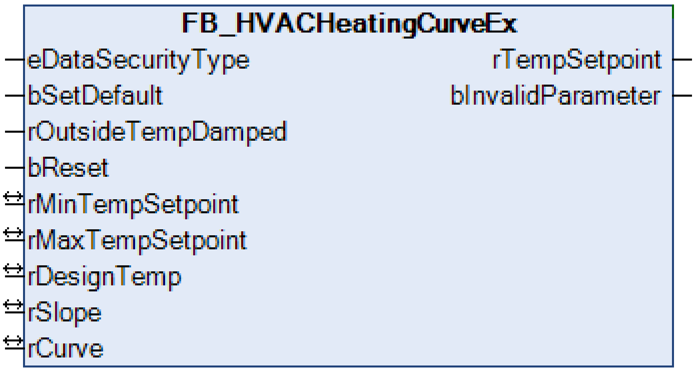
Furthermore the minimum value for the setpoint of the feed temperature must be <= *rTempSetpoint_Y1* and >= 0. The maximum value for the setpoint of the feed temperature must be >= *rTempSetpoint_Y4*.

If one of these conditions is not met, the variable *bInvalidParameter* will be set to TRUE and the default values of the VAR_IN_OUT variables will be adopted.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.6.2 FB_HVACHeatingCurveEx



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.

$$a = rSlope * e^{(b * \ln(z))}$$

$$b = 1/\ln(4) * \ln(1+rCurve) \qquad z, u < 0 \Longrightarrow z,u = 0.0001$$

$$rTempSetpoint = 20 + a * e^{((1-b) * \ln(u))}$$

z = 20 - rDesignTemp;

u = 20 - rOutsideTemp;

VAR_INPUT

```
eDataSecurityType     : E_HVACDataSecurityType;
bSetDefault           : BOOL;
rOutsideTempDamped    : REAL;
bDayOperation         : BOOL;
bReset                : BOOL;
```

eDataSecurityType: If eDataSecurityType [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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.

bReset: Acknowledge input in the event of a fault.

VAR_OUTPUT

```
rTempSetpoint     : REAL;
bInvalidParameter : BOOL;
```

rTempSetpoint: The calculated setpoint [°C] for the flow temperature.

bInvalidParameter: Indicates that there is an incorrect input parameter. *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.

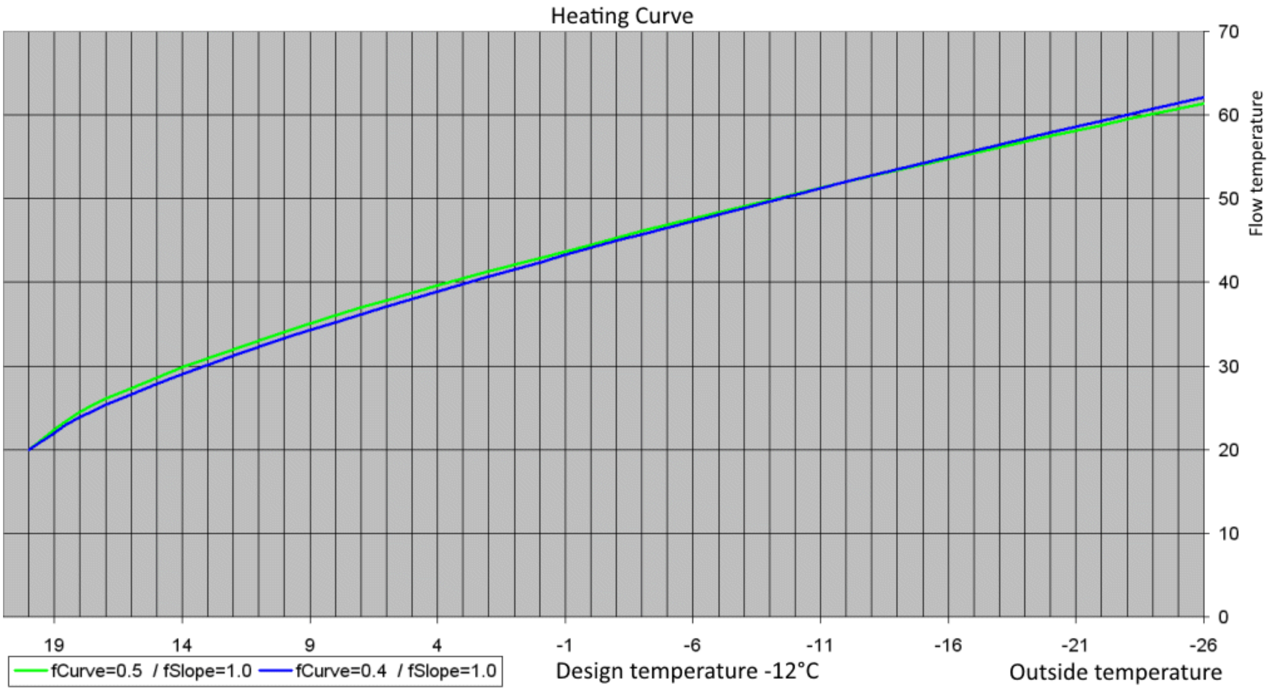
rMaxTempSetpoint: The maximum value for the setpoint of the flow temperature [°C] is defined by this variable. The variable is saved persistently. Preset to 90.

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.

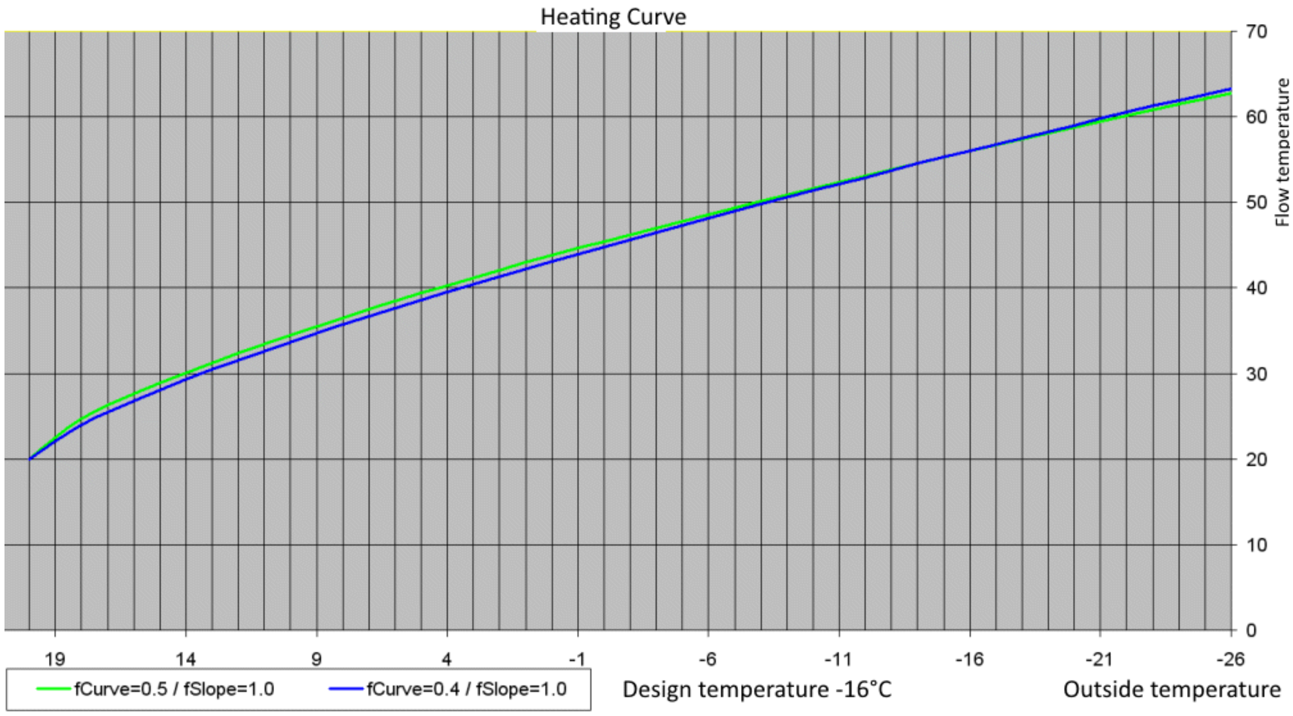
rCurve: Factor for the curve. The variable is saved persistently. Preset to 0.5.

Course of the heating characteristic curves



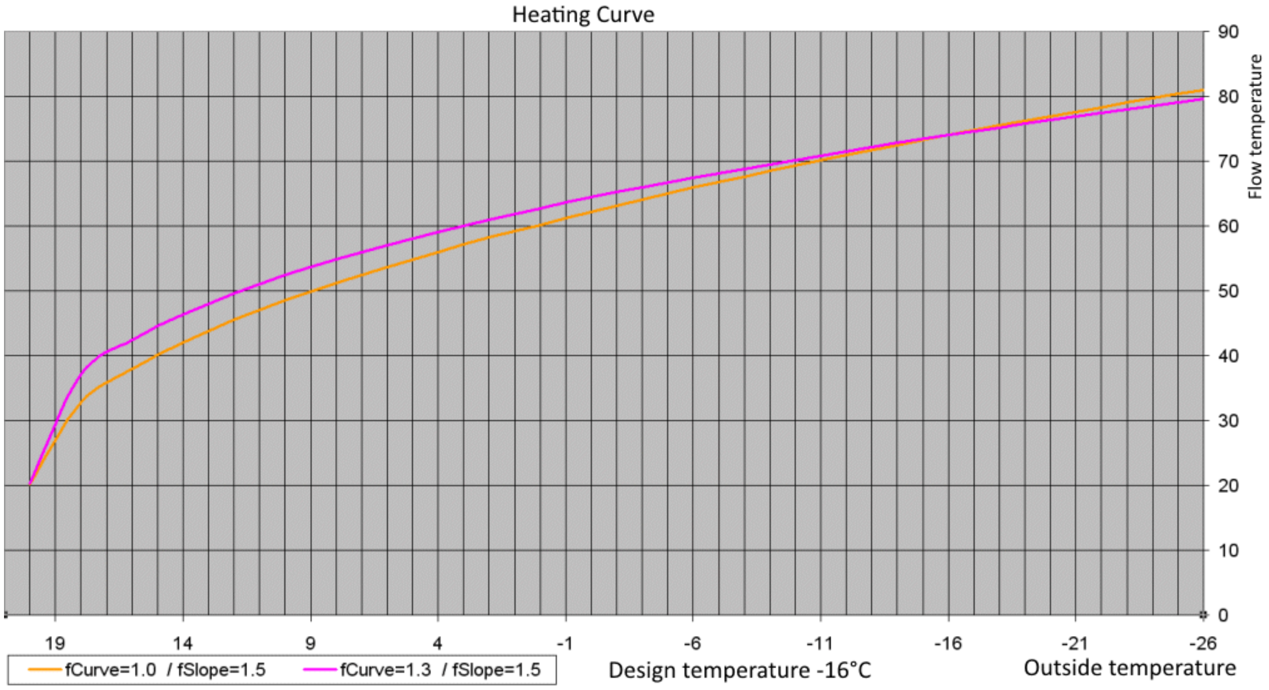
Flow temperature of 50 °C at approx. -9.2 °C

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



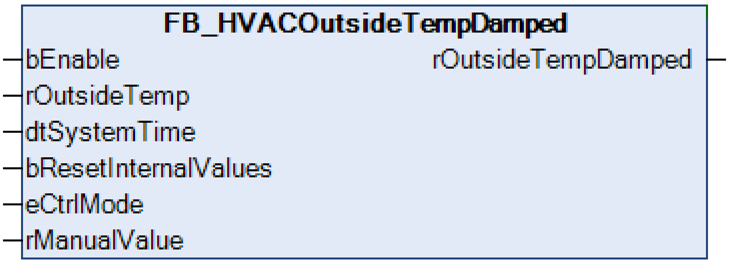
Flow temperature of 60 °C at approx. -0.2 °C

Flow temperature of 60 °C at approx. -3.0 °C

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.6.3 FB_HVACOutsideTempDamped



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 am, 2 pm and 9 pm. The measurement taken at 9 pm receives double the weighting.

VAR_INPUT

```
bEnable                 : BOOL;
rOutsideTemp            : REAL;
dtSystemTime            : DT;
bResetInternalValues    : 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 mode (see E_HVACCtrlMode [▶ 490]).

rManualValue: Manual value that is set at the output *rOutsideTempDamped* if *eCtrlMode* = *eHVACCtrlMode_Manual*.

VAR_OUTPUT

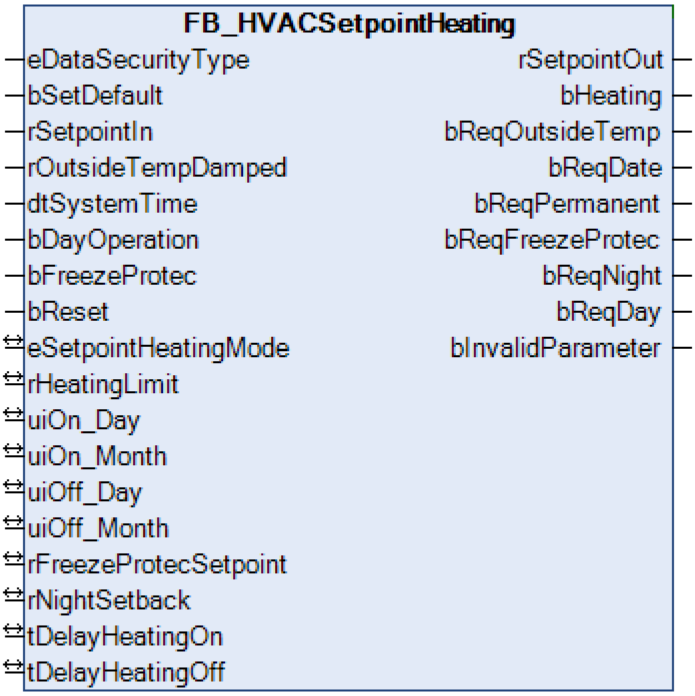
```
rOutsideTempDamped      : REAL;
```

rOutsideTempDamped: Damped outside temperature.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.6.4 FB_HVACSetpointHeating



By 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 supply air 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 the function block **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;
bFreezeProtec          : BOOL;
bReset                 : BOOL;
```

eDataSecurityType: If eDataSecurityType [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 [▶ 322].

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 [▶ 328] regarding this point.

dtSystemtime: This variable transfers the computer system time to the function block.

bDayOperation: The output variable of the 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;
```

rSetpointOut: Setpoint for the supply air 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 operating mode ‘heating period according to outside temperature’.

bReqDate: The heating circuit is in the operating mode ‘heating period according to date’.

bReqPermanent: The heating circuit is in the operating mode ‘heating circuit always on’.

bReqFreezeProtec: The heating circuit is in the operating mode ‘frost protection mode’.

bReqNight: The heating circuit is in the operating mode ‘night operation’.

bReqDay: The heating circuit is in the operating mode ‘day operation’.

bInvalidParameter: Indicates that there is an incorrect input parameter. *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;
rFreezeProtecSetpoint  : REAL;
rNightSetback          : REAL;
tDelayHeatingOn        : TIME;
tDelayHeatingOff       : TIME;
```

eSetpointHeatingMode: Enum that specifies the operation mode of the heating circuit (see E_HVACSetpointHeatingMode [▶ 494]). 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.

rFreezeProtecSetpoint: Setpoint for the heating circuit in frost protection mode (0°C..100°C). The variable is saved persistently. Preset to 8.

rNightSetback: *rNightSetback* specifies the value for the night setback in °C (0°C..100°C). *rNightSetback* is taken into account in the operation mode *eHVACSetpointHeatingMode_OnNight*. The variable is saved persistently. Preset to 10.

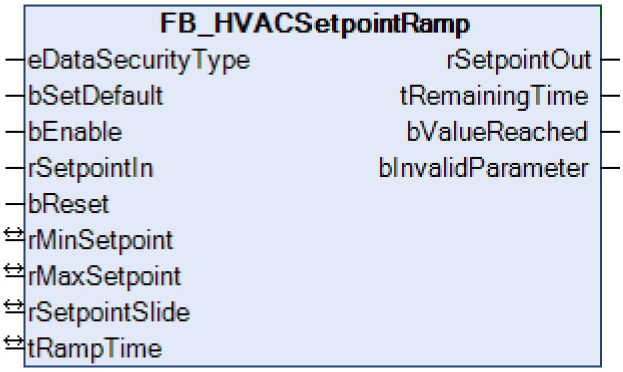
tDelayHeatingOn: Time delay for the setting of the output *bHeating* to TRUE after the damped outside temperature has undershot the heating limit or the date has been reached (0h..100h). The variable is saved persistently. Preset to 0.

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.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.6.5 FB_HVACSetpointRamp



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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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: Enables 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 starting value of the ramp is the last value before the change of *rSetpointIn* to the new target value; see fig. 1.1.
The value range for *rSetpointIn* is specified by the parameters *rMaxSetpoint* and *rMinSetpoint.*
If no correct variable value is present at *rSetpointIn*, then the last valid variable value, if one exists, will be taken. If no valid last value exists, operation continues internally with *rMinSetpoint*. *bInvalidParameter* will be set in the event of an incorrect variable value; the function block continues to work normally.

bReset: Acknowledge input in the event of a fault.

VAR_OUTPUT

```
rSetpointOut         : REAL;
tRemainingTime       : TIME;
bValueReached        : BOOL;
bInvalidParameter    : 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*.

bInvalidParameter: 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 block; see description of variables. After rectifying the incorrect parameter entry, the message *bInvalidParameter* must be acknowledged via *bReset*.

VAR_IN_OUT

```
rMinSetpoint     : REAL;
rMaxSetpoint     : REAL;
rSetpointSlide   : REAL;
tRampTime        : TIME;
```

rMinSetpoint/rMaxSetpoint: The value range for *rSetpointIn* is specified by the parameters *rMaxSetpoint* and *rMinSetpoint* (-10000..10000). *rMaxSetpoint* must be greater than *rMinSetpoint.*

If no correct variable value is present, then the last valid variable value, if one exists, will be taken. 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 function block continues to work normally. The variable is saved persistently. Preset to 10 or 40 respectively.

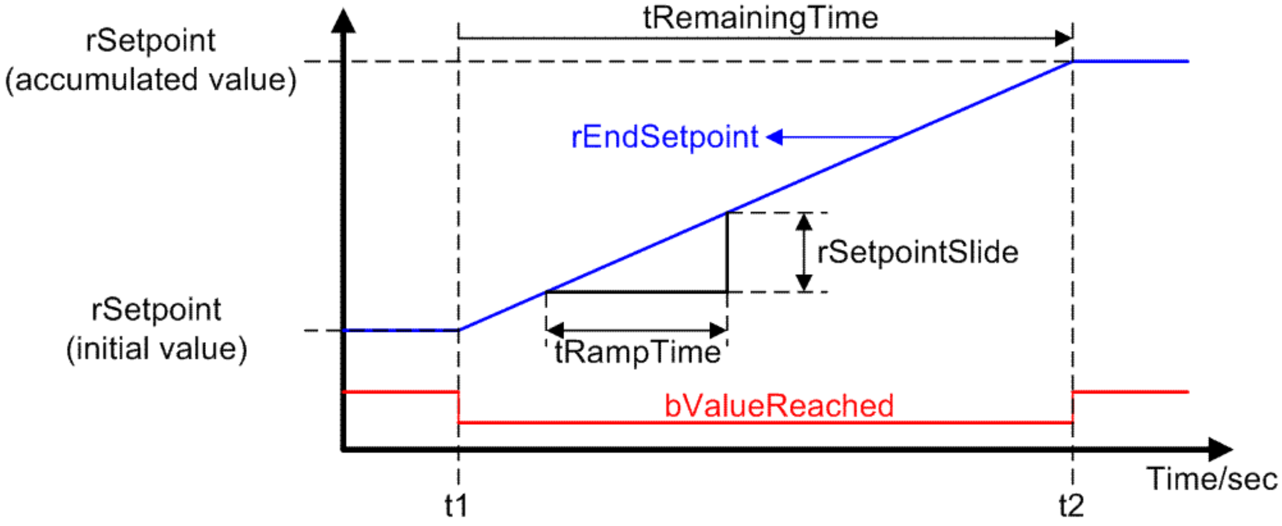
rSetpointSlide: Absolute temperature change in [Kelvin/*tRampTime*] with which the output is converted in sliding steps from a lower to a higher value or from a higher to a lower value, see fig. 1.1

If no correct variable value is present, then the last valid variable value, if one exists, will be taken. 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 function block continues to work 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 no correct variable value is present, then the last valid variable value, if one exists, will be taken. 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 function block continues to work normally. The variable is saved persistently. Preset to 3600 s.

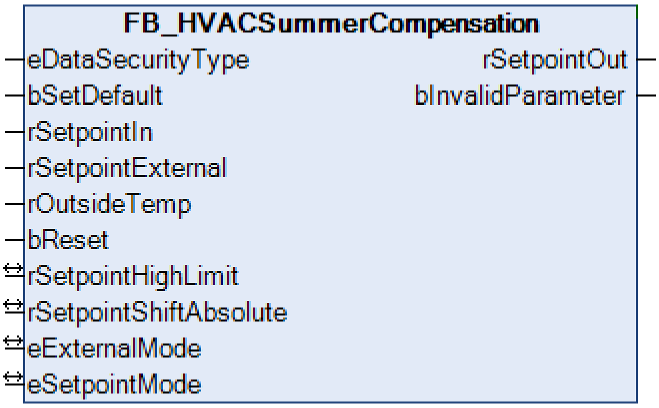
Fig. 1.1:



Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.6.6 FB_HVACSummerCompensation



This function block is a setpoint 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*: = 6K

outside temperature *rOutsideTemp*: = 28 °C

eExternalMode:= e*HVACExternalMode_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).

The set differential value between outside temperature and room temperature *rSetpointShiftAbsolute*: = 6K

outside temperature *rOutsideTemp*: = 28 °C

eExternalMode:= *eHVACExternalMode_On*

eSetpointMode:= *eHVACSetpointMode_DIN*

calculated room temperature setpoint *rSetpointOut*:= 20°C

Example III:

Room setpoint *rSetpointIn*: = 23 °C

external setpoint *rSetpointExternal*: = -3K (e.g. +/- 3K can be set via a potentiometer).

The set differential value between outside temperature and room temperature *rSetpointShiftAbsolute*: = 6K

outside temperature *rOutsideTemp*: = 34 °C

rSetpointHighLimit: = 35 °C

eExternalMode:= *eHVACExternalMode_On*

eSetpointMode:= *eHVACSetpointMode_DINLimited*

calculated room temperature setpoint *rSetpointOut*:= 31°C

Example IV:

Room setpoint *rSetpointIn*: = 24 °C

external setpoint *rSetpointExternal*: = -3K (e.g. +/- 3K can be set via a potentiometer).

The set differential value between outside temperature and room temperature *rSetpointShiftAbsolute*: = 6K

outside temperature r*OutsideTemp*:= 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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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: Setpoint for the room.

bInvalidParameter: Indicates that there is an incorrect input parameter. *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 external setpoint is activated/deactivated via the enum E_HVACExternalMode (see E_HVACExternalMode [▶ 491]).
eHVACExternalMode_Off means that the external setpoint set via the potentiometer on the control panel is deactivated.
eHVACExternaMode_On means that the external setpoint is activated, e.g. +/- 3 °C
eHVACExternalMode_ShiftAbsolut corresponds to an absolute external setpoint in °C

eSetpointMode: Enum that specifies the type of setpoint determination (see E_HVACSetpointMode [▶ 494]).

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.

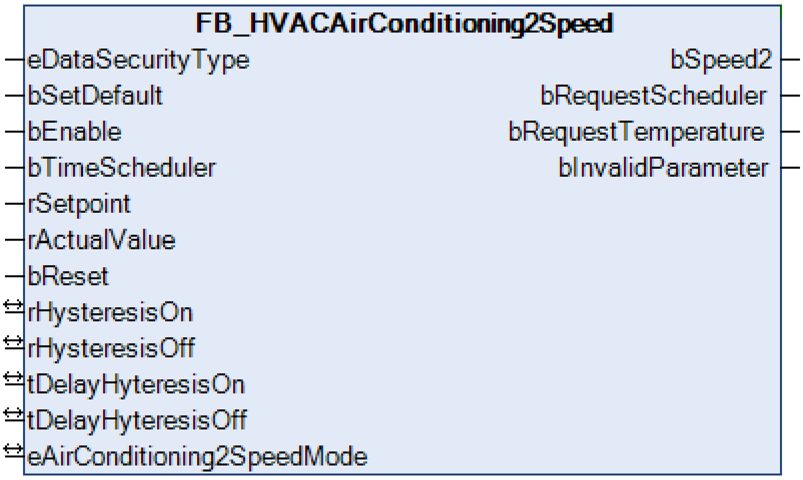
1. According to DIN (*eHVACSetpointMode_DIN*). The room temperature setpoint follows the outside temperature, guided by a difference.
2. Limited (*eHVACSetpointMode_DINLimited*). The room temperature setpoint follows the outside temperature, guided by a difference. However, it is limited. The limit value is *rSetpointHighLimit*.
3. Constant (*eHVACSetpointMode_ConstantValueBase*). The room temperature setpoint is specified as a constant value; the outside temperature does not exert any influence.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7 HVAC Special functions

5.1.7.1 FB_HVACAirConditioning2Speed



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 scheduler 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 release 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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 temperature

rActualValue: actual value of room air 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 speed.

bRequestScheduler: Request for the second fan speed 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 temperature, *rSetpoint*, and the actual value of the room or exhaust temperature, *rActualValue*, depending on the time delays *tDelayHysteresisOn*/*tDelayHysteresisOff*, the switch-on and release hystereses *rHysteresisOn*/*rHysteresisOff* and the setting of the operating 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 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;
tDelayHyteresisOn          : TIME;
tDelayHyteresisOff         : TIME;
eAirConditioning2SpeedMode : E_HVACAirConditioning2SpeedMode;
```

rHysteresisOn: Parameter 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 larger than *rHysteresisOff* if the operation mode *eAirConditioning2SpeedMode*=*eHVACAirConditioning2SpeedModeHeatingAndCooling*.

If no correct variable value is present, then the last valid variable value, if one exists, will be taken. 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 function block continues to work normally. The variable is saved persistently. Preset to 5.

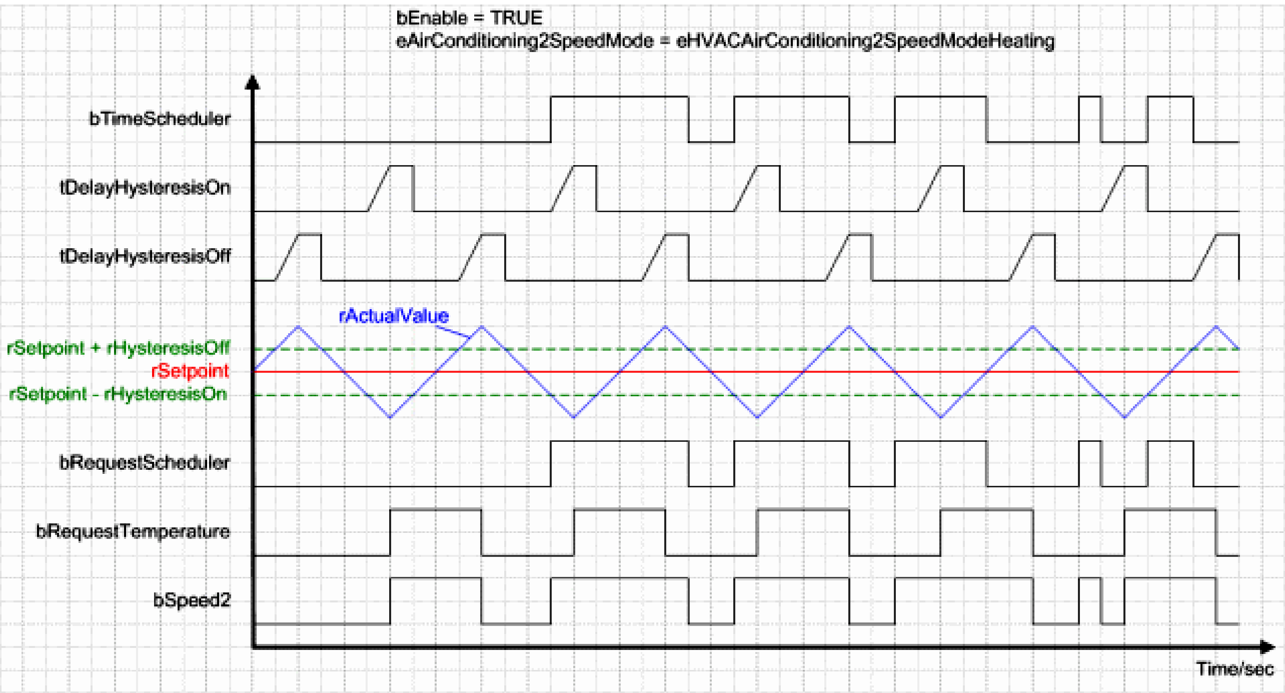
rHysteresisOff: Parameter to switch off the second fan stage via a hysteresis loop, load-dependent, via the output variable *bSpeed2* depending on the enum *eAirConditioning2SpeedMode (0..1000).*
If no correct variable value is present, then the last valid variable value, if one exists, will be taken. 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 function block continues to work 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.

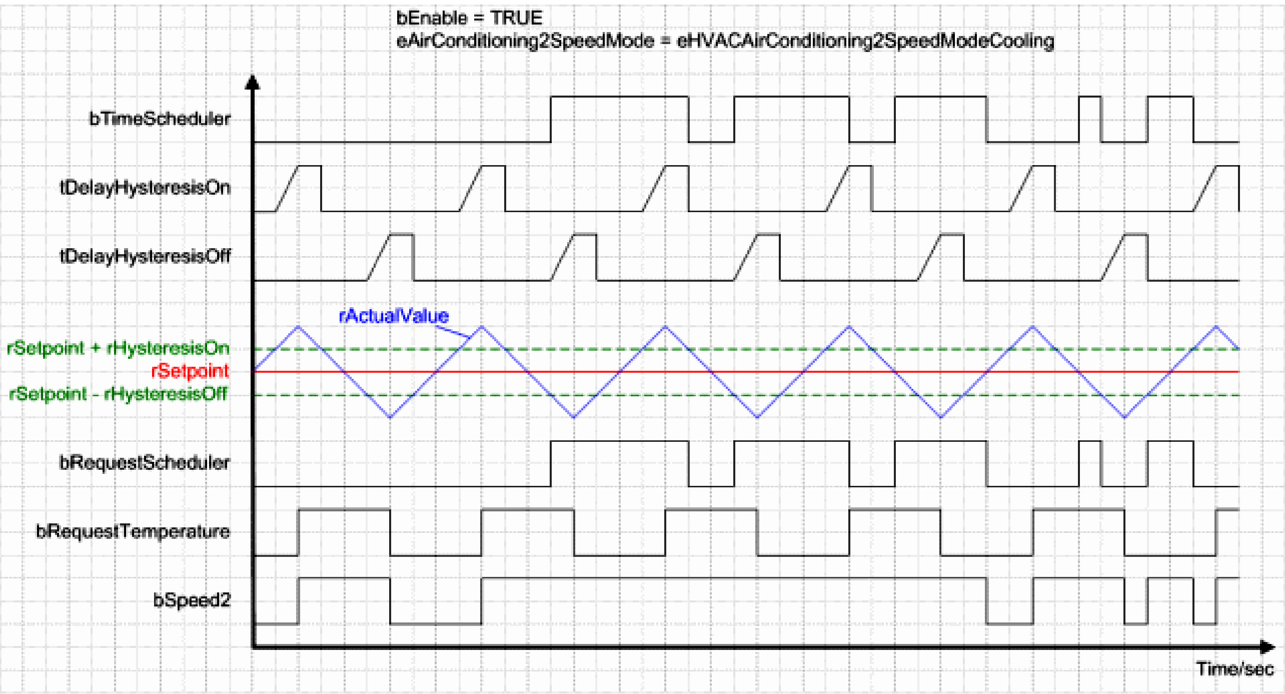
tDelayHysteresisOff: Switch-off delay of the output variable *bSpeed2* in the event of a load-dependent request for the second fan stage. The variable is saved persistently. Preset to 300 s.

eAirConditioning2SpeedMode: Enum that specifies the load-dependent operation mode for switching on the second fan stage via the output variable *bSpeed2* (see E_HVACAirConditioning2SpeedMode [▸ 489]).
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.

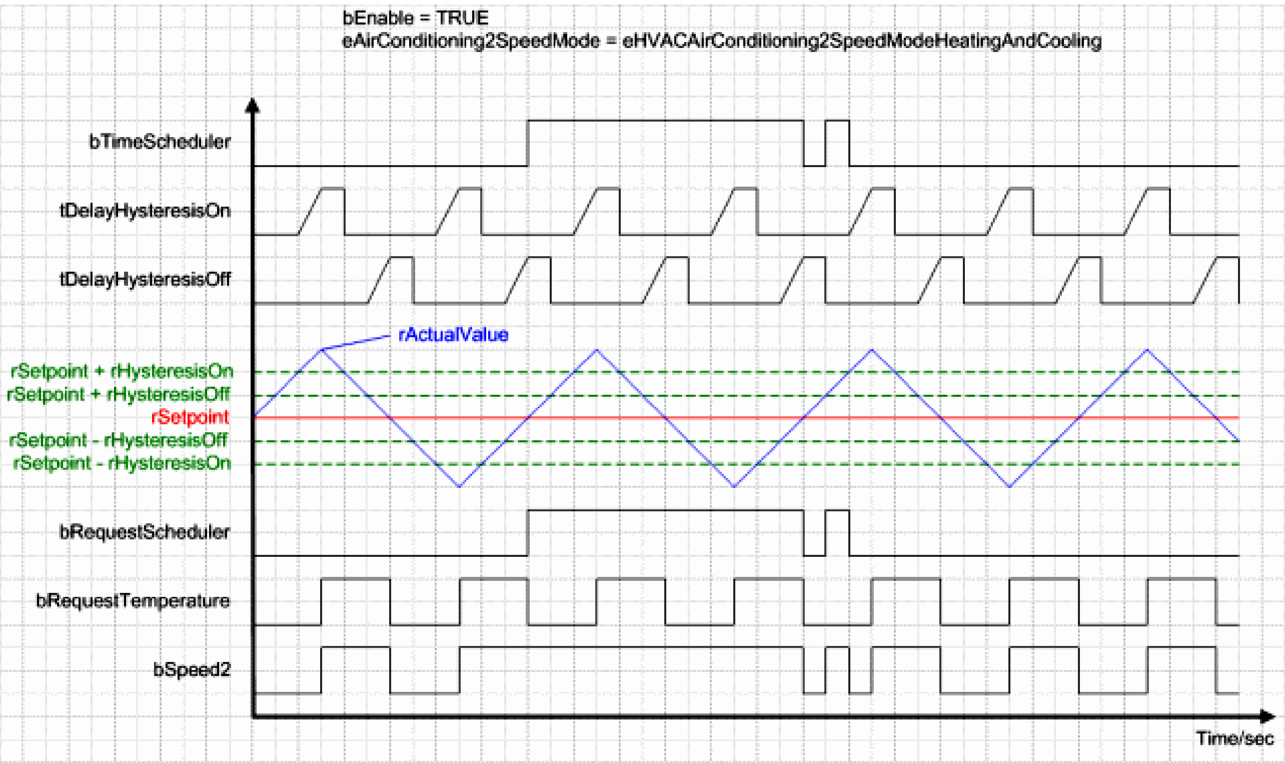
Operating mode eAirConditioning2SpeedMode = eHVACAirConditioning2SpeedMode_Heating



Operating mode eAirConditioning2SpeedMode = eHVACAirConditioning2SpeedMode_Cooling



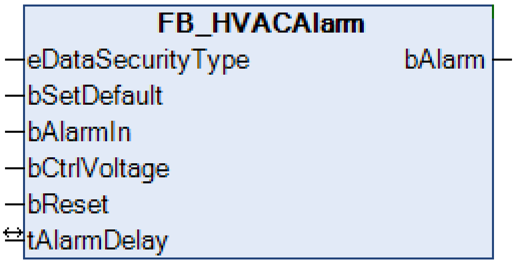
Operating mode eAirConditioning2SpeedMode = eHVACAirConditioning2SpeedMode_HeatingAndCooling



Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.2 FB_HVACAlarm



A TRUE on the input *bAlarmIn* indicates that an alarm signal is present. The alarm is routed through 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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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: Via the input *bCtrlVoltage* a check 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

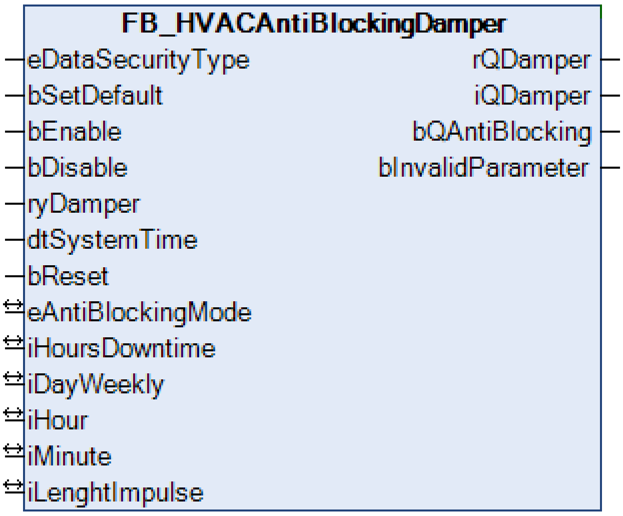
```
tAlarmDelay     : TIME;
```

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 500 s.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.3 FB_HVACAntiBlockingDamper



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
dtSystemTime         : DT;
bReset               : BOOL;
```

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 *AntiBlocking* function is enabled by 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 fed to the outputs *rQDamper* and *iQDamper* (0%..100%). 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 *bInvalidParameter*.

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 (see E_HVACAntiBlockingMode [▸ 489]).

eHVACAntiBlockingMode_Off:= 0: Off

eHVACAntiBlockingMode_Downtime:= 1: After the expiry of a standstill time in hours, the *Antiblocking*-pulse will be routed 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 (0..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;
iQDamper          : INT;
bQAntiBlocking    : BOOL;
bInvalidParameter: BOOL;
```

rQDamper: This output corresponds to the input value of the control value (0%..100%). 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 %.

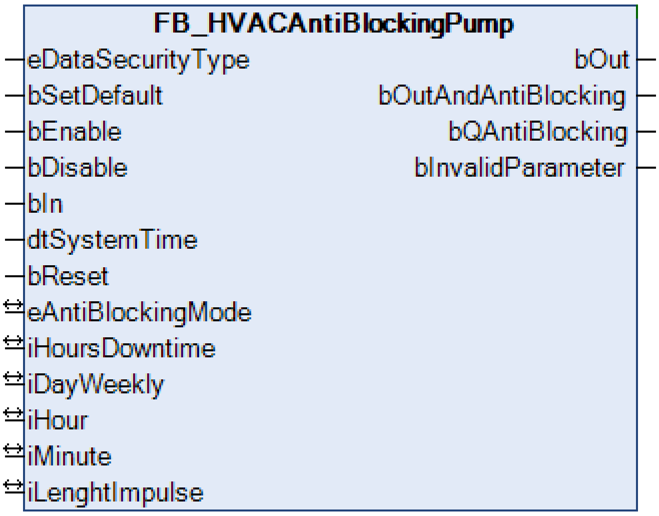
bQAntiBlocking: This output is independent of the switch-on signal and shows only the *Antiblocking-pulse*.

bInvalidParameter: An error occurred during the plausibility check. It is deleted again by *bReset*.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.4 FB_HVACAntiBlockingPump



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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 *AntiBlocking* function is enabled by 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 relayed to the output *bOut*.

dtSystemTime: System time

bReset: Acknowledge input in the event of a fault. Resets the flag *bInvalidParameter*.

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 (see E_HVACAntiBlockingMode [▶ 489]).

eHVACAntiBlockingMode_Off:=0 : Off

eHVACAntiBlockingMode_Downtime:=1 : After the expiry of a standstill time in hours, the *Antiblocking*-pulse will be routed 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..23h). 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*.

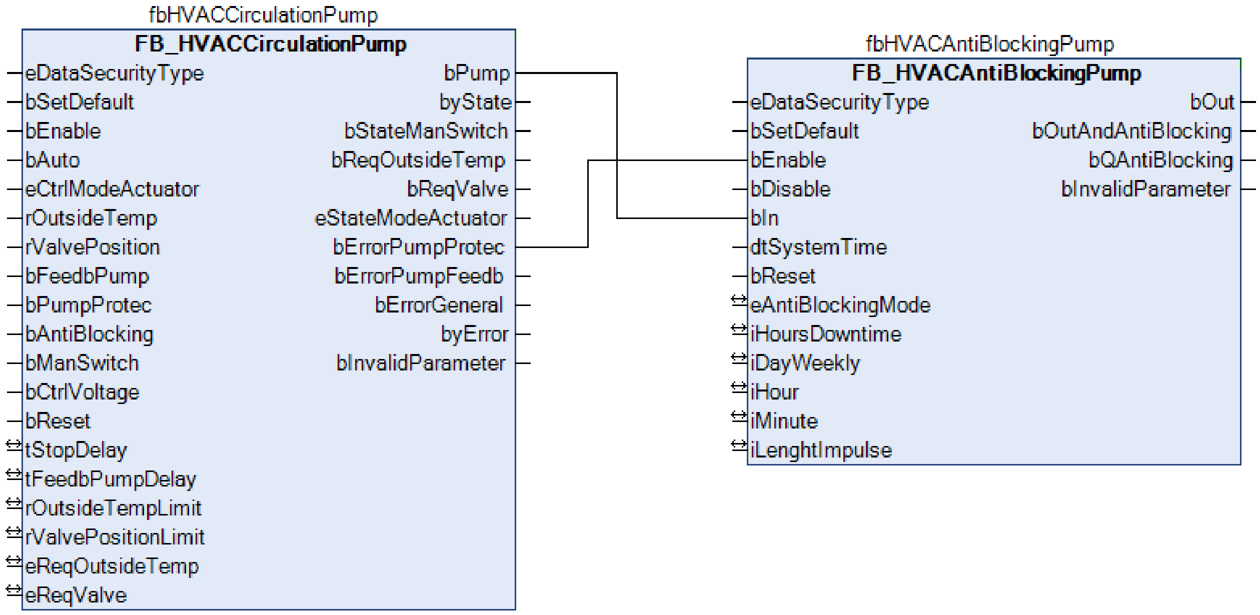
bOutAndAntiBlocking: The output can go TRUE if the request comes from the pump function block OR from the anti-blocking function. The pump is connected to this output.
This output corresponds to the switch-on condition and is dependent upon the state of the *Antiblocking-pulse* and the inputs *bEnable* and *bDisable*. If *bEnable* = FALSE, then 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 request comes from the anti-blocking 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*.

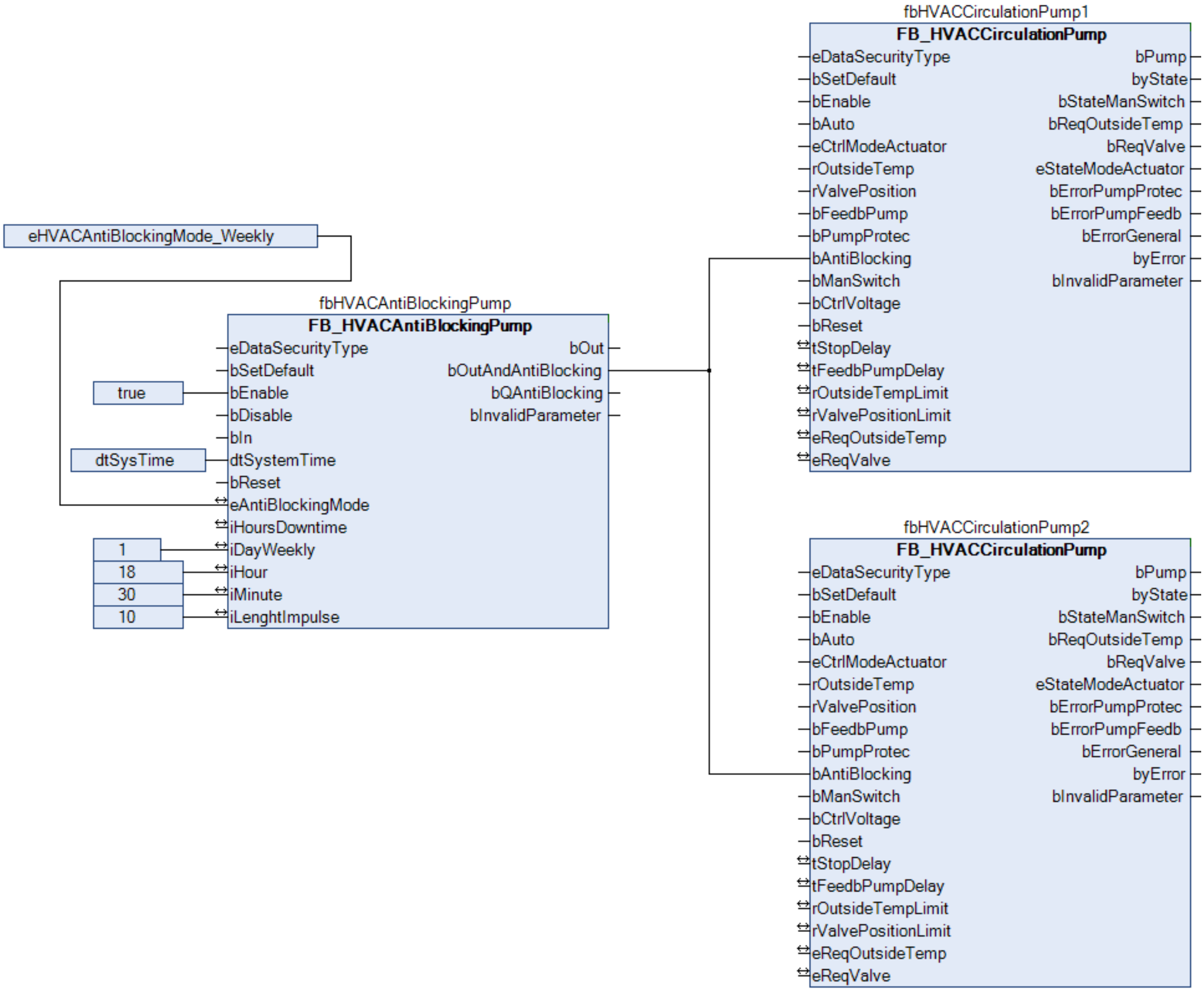
Example of a combination of a pump block and the FB_HVACAntiBlocking:

- 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:

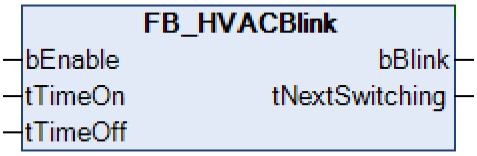
- Enable fbHVACAntiBlocking via *bEnable*.
- Select weekly mode (e.g. Monday at 6.30 pm for 10 secs).



Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.5 FB_HVACBlink



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: Enables the function block.

tTimeOn: Switch-on duration of the flash pulse, *bBlink* = TRUE.

tTimeOff: Switch-off duration of the flash pulse, *bBlink* = FALSE.

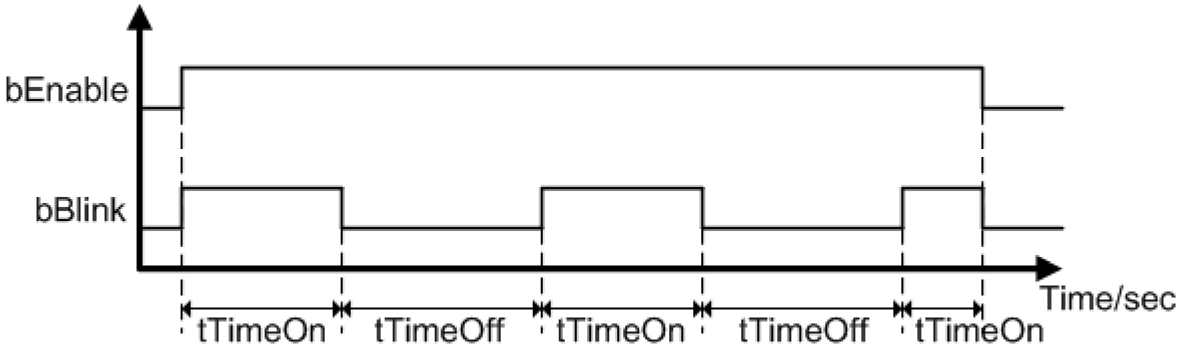
VAR_OUTPUT

```
bBlink           :BOOL;
tNextSwitching   :TIME;
```

bBlink: Flash output

tNextSwitching: Time until the next change of state of the output *bBlink.*

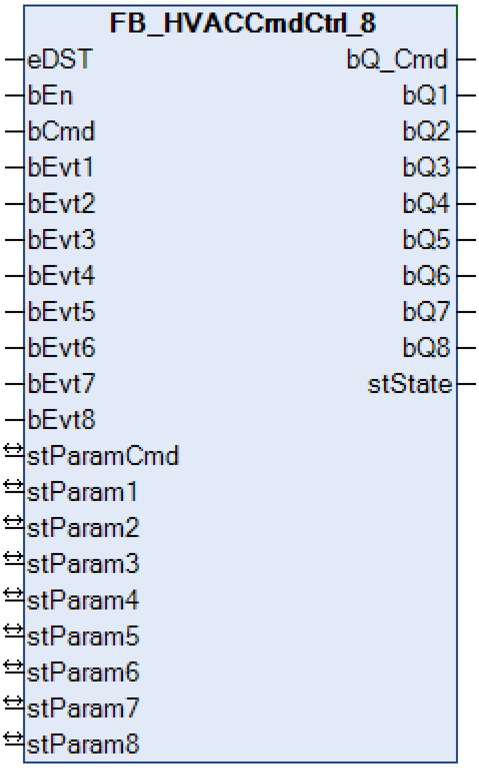
Behavior of the output value



Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.6 FB_HVACCmdCtrl_8



With this function block, individual units in a system can be sequentially switched on (*bQ_Cmd* > *bQ1* > *bQ2… bQ8*) or switched off (*bQ8* > *bQ7* > *bQ6… bQ_Cmd*) in a certain order. *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) AND Event(n) AND (DelayOn(n) = 0 AND (MinOff(n) = 0 AND *stState.udiStep* = n

Example for the output bQ_Cmd

bQ_Cmd = *bCmd* AND *stState.udiSecRT_DelayOn* = 0 AND *stState.udiSecRT_MinOff[0]* = 0 AND *stState.udiStep* = 0

Example for the output bQ2

bQ2 = *bQ1* AND *bEvt2* AND *stState.udiSecRT_DelayOn* = 0 AND *stState[2].udiSecRT_MinOff* = 0 AND *stState.udiStep* = 2

Switch-off condition if is TRUE and is FALSE: *bEn bCmd*

Q(n) = NOT Q(n + 1) AND (DelayOff(n) = 0 AND (MinOn(n) = 0 AND *stState.udiStep* = n

Example for the output bQ8

bQ8 = *stState.udiSecRT_DelayOff* = 0 AND *stState[7].udiSecRT_MinOn* = 0 AND *stState.udiStep* = 8

Example for the output bQ_Cmd

bQ_Cmd = NOT*bQ1* AND *stState.udiSecRT_DelayOff* = 0 AND *stState[0].udiSecRT_MinOn* = 0 AND *stState.udiStep* = 0

Current step

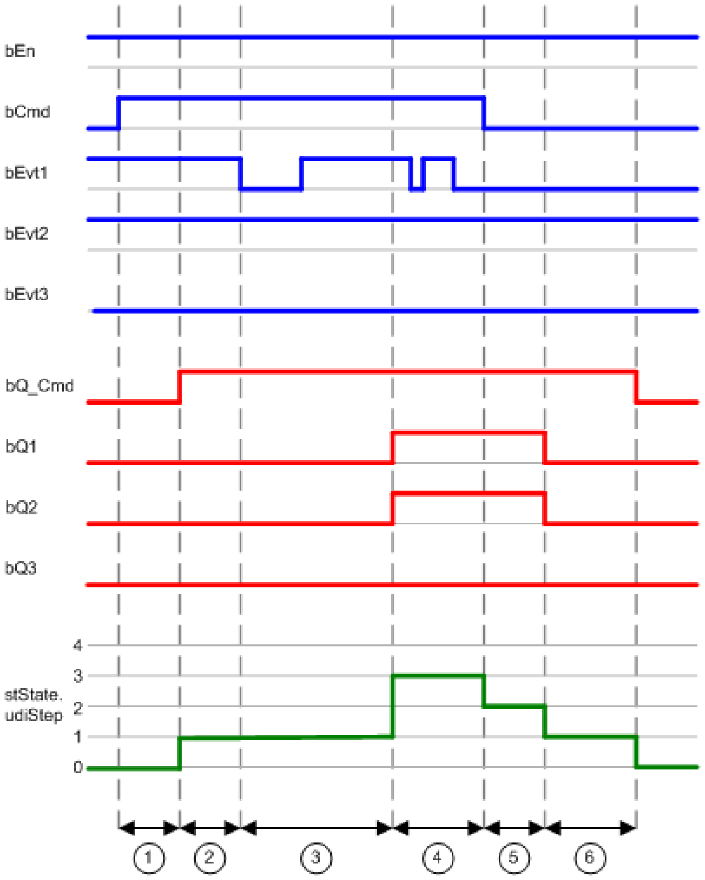
On the basis of the status variable *stState.udiStep* it is possible to see which step the function block is currently in.

Switch-on and switch-off sequence

If *bEn* and *bCmd* are TRUE the switch-on sequence is (*bQ_Cmd* > *bQ1* > *bQ2 … bQ8*). The switch-off sequence (*bQ8* > *bQ7* > *bQ6 … bQ_Cmd*) is active if *bEn* = TRUE and *bCmd* = 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* = 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.



① 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.

② 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.

③ 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.

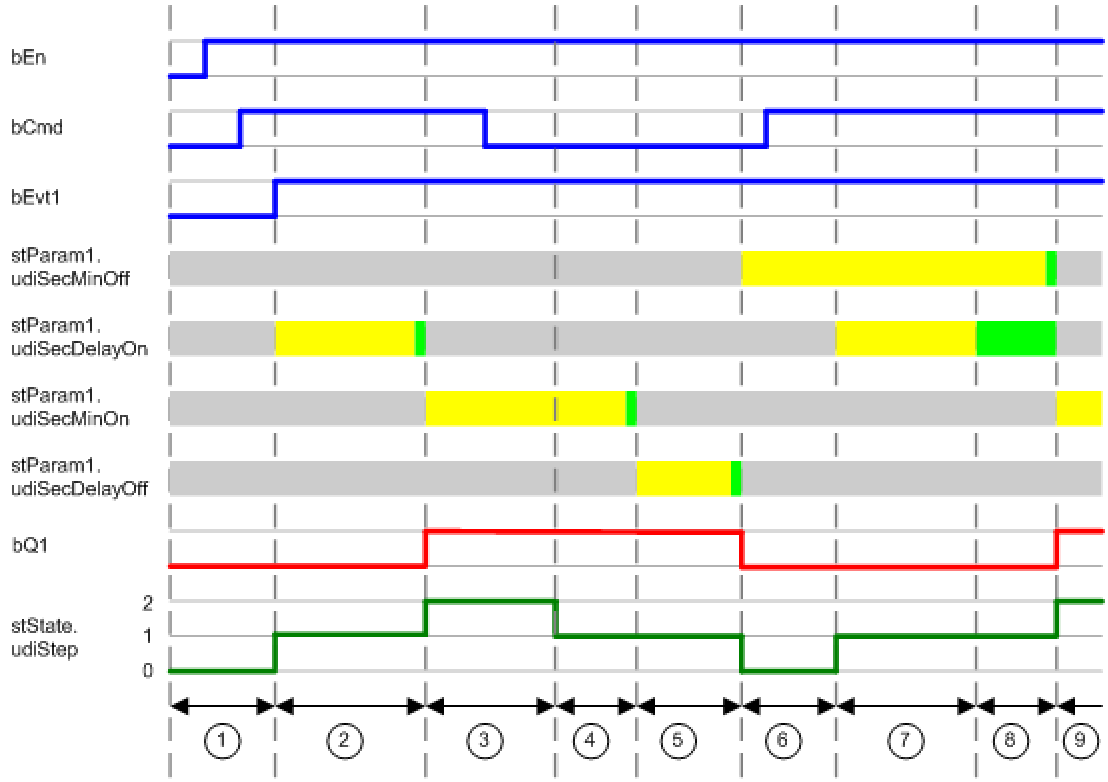
④ Da das Ereignis bEvt2 vor Freigabe der Stufe 2 TRUE war und für die Stufe 2 keine Einschaltverzögerungszeit eingestellt wurde, ändert sich sofort der Status von bQ2 auf TRUE. Im Zuge dessen ändert sich der Wert der Stufe (stState.udiStep) von 2 nach 3.

⑤ 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 Auschaltverzögerungszeit stState.udiSecRT_DelayOff ändert sich der Status von bQ2 nach FALSE und der Wert der Stufe von 2 nach 1.

⑥ 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*.



Zeitstatus	
Steht	▨
Aktiv	🟨
Abgelaufen	🟩

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.

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;
```

eDataSecurityType: If eDataSecurityType [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.

bEn: Enables 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*AND*bEvt2* AND *stState.udiSecRT_DelayOn* = 0 AND *stState.udiSecRT_MinOff[2]* = 0 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* AND *bEvt3*AND*stState.udiSecRT_DelayOn* = 0 AND *stState.udiSecRT_MinOff[3]* = 0 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* AND *bEvt4*AND*stState.udiSecRT_DelayOn* = 0 AND *stState.udiSecRT_MinOff[4]* = 0 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* AND *bEvt5*AND*stState.udiSecRT_DelayOn* = 0 AND *stState.udiSecRT_MinOff[5]* = 0 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* AND *bEvt6*AND*stState.udiSecRT_DelayOn* = 0 AND *stState.udiSecRT_MinOff[6]* = 0 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*AND*bEvt7* AND *stState.udiSecRT_DelayOn* = 0 AND *stState.udiSecRT_MinOff[7]* = 0 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* AND *bEvt8*AND*stState.udiSecRT_DelayOn* = 0 AND *stState.udiSecRT_MinOff[8]* = 0 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 sequence.

The input variable *bCmd* and the parameter structure *stParamCmd* belong to *bQ_Cmd*. *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*AND*stState*.*udiSecRT_DelayOn* = 0 AND*stState*.*udiSecRT_MinOff[0]* = 0 AND*stState*.*udiStep* = 0

Switch-off condition

bQ_Cmd = NOT*bQ1*AND*stState*.*udiSecRT_DelayOff* = 0 AND*stState*.*udiSecRT_MinOn[0]* = 0 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 sequence.

The event 1 *bEvt1* and the parameter structure *stParam1* belong to *bQ1*. *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*AND*stState*.*udiSecRT_DelayOn* = 0 AND*stState*.*udiSecRT_MinOff[1]* = 0 AND*stState*.*udiStep* = 1

Switch-off condition

bQ1 = NOT*bQ1*AND*stState*.*udiSecRT_DelayOff* = 0 AND*stState*.*udiSecRT_MinOn[1]* = 0 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 sequence.

The event 2 *bEvt2* and the parameter structure *stParam2* belong to *bQ2*. *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* AND *stState*.*udiSecRT_DelayOn* = 0 AND *stState*.*udiSecRT_MinOff[2]* = 0 AND *stState*.*udiStep* = 2

Switch-off condition

bQ2 = NOT*bQ2* AND *stState*.*udiSecRT_DelayOff* = 0 AND *stState*.*udiSecRT_MinOn[2]* = 0 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 sequence.

The event 3 *bEvt3* and the parameter structure *stParam3* belong to *bQ3*. *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* AND *stState*.*udiSecRT_DelayOn* = 0 AND *stState*.*udiSecRT_MinOff[3]* = 0 AND*stState*.*udiStep* = 3

Switch-off condition

bQ3 = NOT*bQ3* AND *stState.udiSecRT_DelayOff* = 0 AND *stState.udiSecRT_MinOn[3]* = 0 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 sequence.

The event 4 *bEvt4* and the parameter structure *stParam4* belong to *bQ4*. *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* AND *stState.udiSecRT_DelayOn* = 0 AND *stState.udiSecRT_MinOff[4]* = 0 AND *stState.udiStep* = 4

Switch-off condition

bQ4 = NOT*bQ4* AND *stState.udiSecRT_DelayOff* = 0 AND *stState.udiSecRT_MinOn[4]* = 0 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 sequence.

The event 5 *bEvt5* and the parameter structure *stParam5* belong to *bQ5*. *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* AND *stState.udiSecRT_DelayOn* = 0 AND *stState.udiSecRT_MinOff[5]* = 0 AND *stState.udiStep* = 5

Switch-off condition

bQ5 = NOT *bQ5* AND *stState.udiSecRT_DelayOff* = 0 AND *stState.udiSecRT_MinOn[5]* = 0 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 sequence.

The event 6 *bEvt6* and the parameter structure *stParam6* belong to *bQ6*. *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* AND *stState.udiSecRT_DelayOn* = 0 AND *stState.udiSecRT_MinOff[6]* = 0 AND *stState.udiStep* = 6

Switch-off condition

bQ6 = NOT*bQ6* AND *stState.udiSecRT_DelayOff* = 0 AND *stState.udiSecRT_MinOn[6]* = 0 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 sequence.

The event 7 *bEvt7* and the parameter structure *stParam7* belong to *bQ7*. *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* AND *stState.udiSecRT_DelayOn* = 0 AND *stState.udiSecRT_MinOff[7]* = 0 AND *stState.udiStep* = 7

Switch-off condition

bQ7 = NOT*bQ7* AND *stState.udiSecRT_DelayOff* = 0 AND *stState.udiSecRT_MinOn[7]* = 0 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 sequence.

The event 8 *bEvt8* and the parameter structure *stParam8* belong to *bQ8*. *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*AND*stState.udiSecRT_DelayOn* = 0 AND*stState.udiSecRT_MinOff[8]* = 0 AND*stState.udiStep* = 8

Switch-off condition

bQ8 = *stState.udiSecRT_DelayOff* = 0 AND *stState.udiSecRT_MinOn[8]* = 0 AND *stState.udiStep* = 8

stState: This structure indicates the remaining times of the switch-on or switch-off delay of the active step and the minimum switch-on or minimum switch-off duration of the outputs (see ST_HVACCmdCtrl_8State [▸ 496]).

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 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*]*. 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 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*]*. 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* (see ST_HVACCmdCtrl_8Param [▶ 496]). 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.

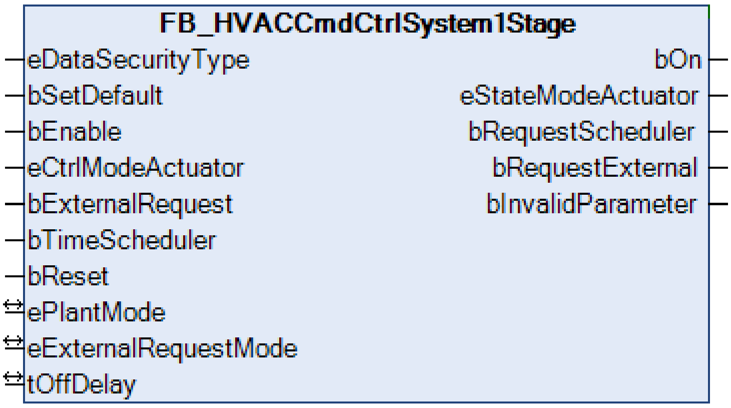
stParam7: The structure contains time variables that can be used to define a switch-on or switch-off delay as well as a minimum switch-on or minimum switch-off duration of the output *bQ7*. The variable is saved persistently.

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.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.7 FB_HVACCmdCtrlSystem1Stage



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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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* is also FALSE.

eCtrlModeActuator: Enum that specifies the system operation modes Manual, Auto and Off (see E_HVACActuatorMode [▶ 489]). If specified incorrectly, operation continues internally with the last valid operation mode. This is *eHVACActuatorMode_Auto_BMS* in the case of initial commissioning. *bInvalidParameter* is set in the case of an incorrect parameter specification.

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 (see E_HVACActuatorMode [▶ 489]).

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 system functions can be executed in automatic mode, depending on the input variables *bExternalRequest* and *bTimeScheduler* (see E_HVACPlantMode [▸ 492]).
ePlantMode = *eHVACPlantMode_TimeSchedulingOnly*: The system is switched on and off exclusively via the input variable *bTimeScheduler*.
ePlantMode = *eHVACPlantMode_TimeScheduling_And_ExternalRequest*: The system is switched on if the input variables *bTimeScheduler* AND *bExternalRequest* are TRUE.
ePlantMode = *eHVACPlantMode_TimeScheduling_Or_ExternalRequest*: The system is switched on via the input variable *bTimeScheduler* OR *bExternalRequest.*
ePlantMode = *eHVACPlantMode_ExternalRequestOnly*: The system is switched on and off exclusively via the input variable *bExternalRequest*.

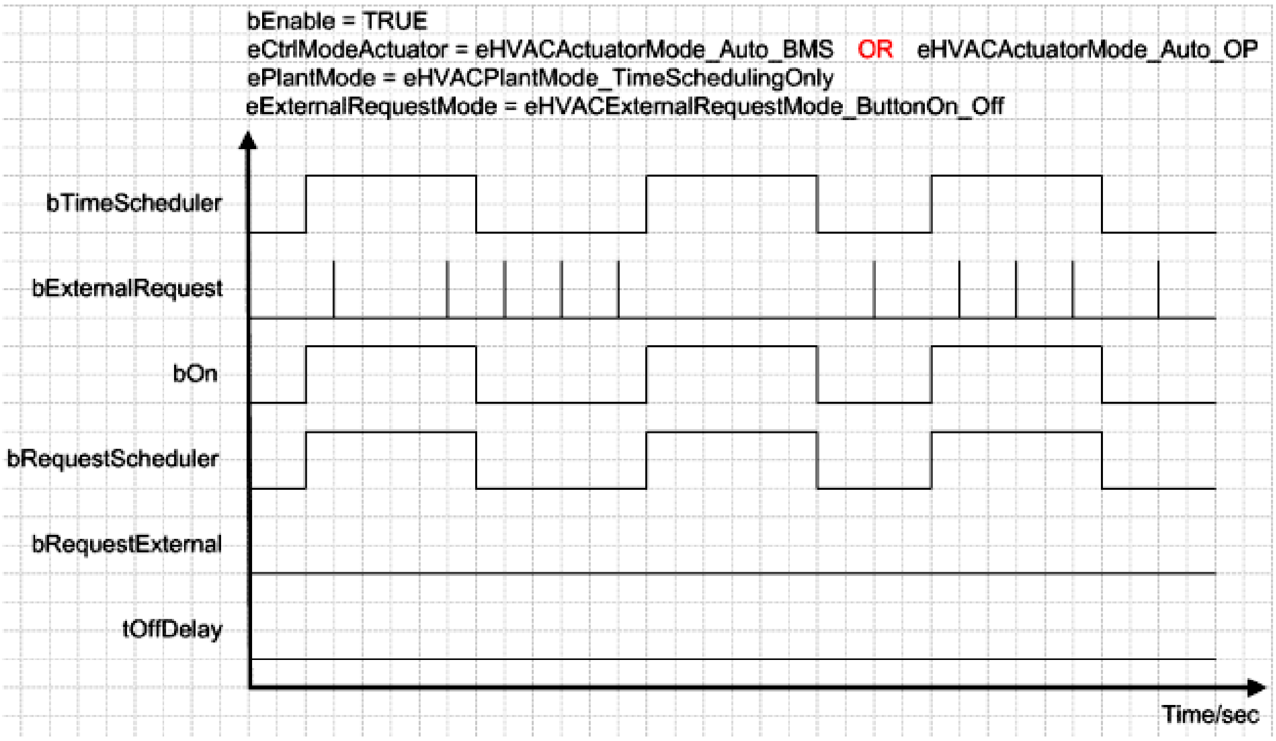
If no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification. The variable is saved persistently. Preset to 0.

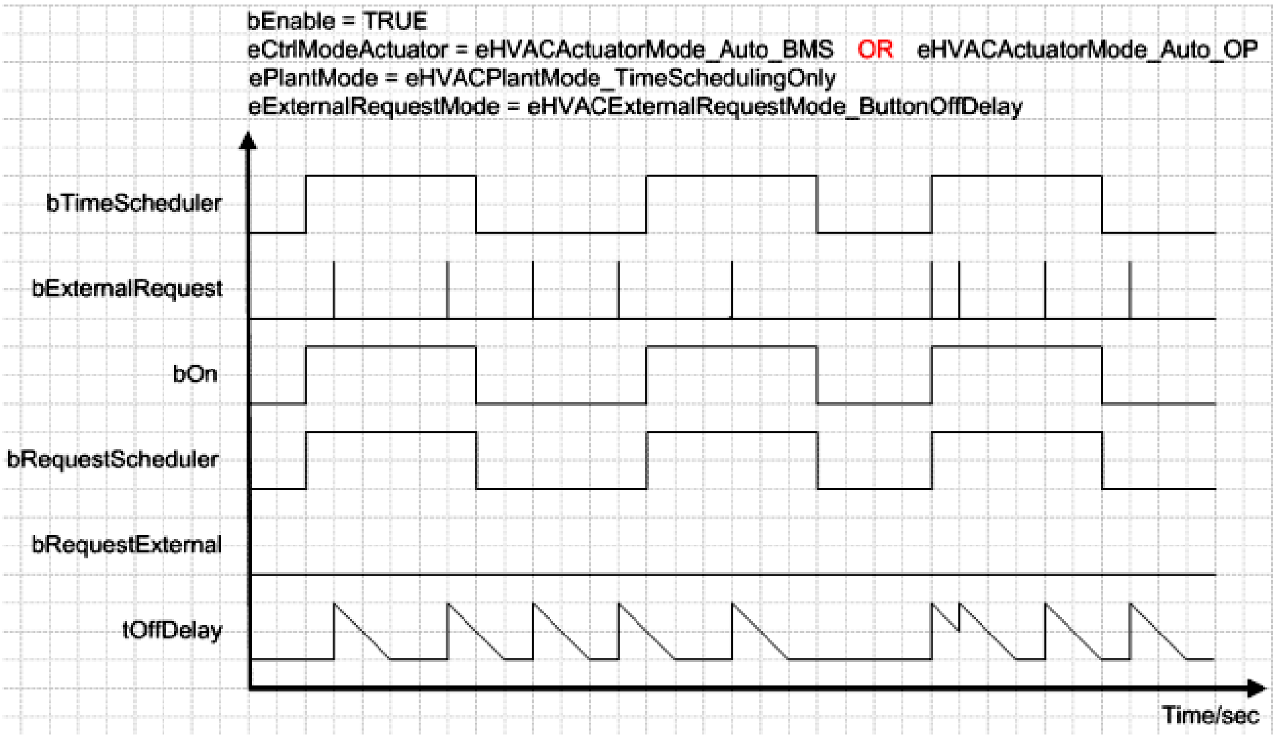
eExternalRequestMode: The enum *eExternalRequestMode* specifies the mode of operation of the input variable *bExternalRequest* in automatic mode, depending on the enum *ePlantMode* (see E_HVACExternalRequestMode [▸ 492]).
eExternalRequestMode = *eHVACExternalRequestMode_ButtonOn_Off*: The external request is set to TRUE following a rising edge on *bExternalRequest*. A further rising edge resets the request to FALSE.
eExternalRequestMode = *eHVACExternalRequestMode_ButtonOffDelay*: Following a rising edge on the input variable *bExternalRequest*, the external request extends or sets the utilization time of the system by the time setting of *tOffDelay*.
eExternalRequestMode = *eHVACExternalRequestMode_SwitchOn_Off*: The external request is active if *bExternalRequest* = TRUE. It is deactivated if *bExternalRequest* = FALSE.

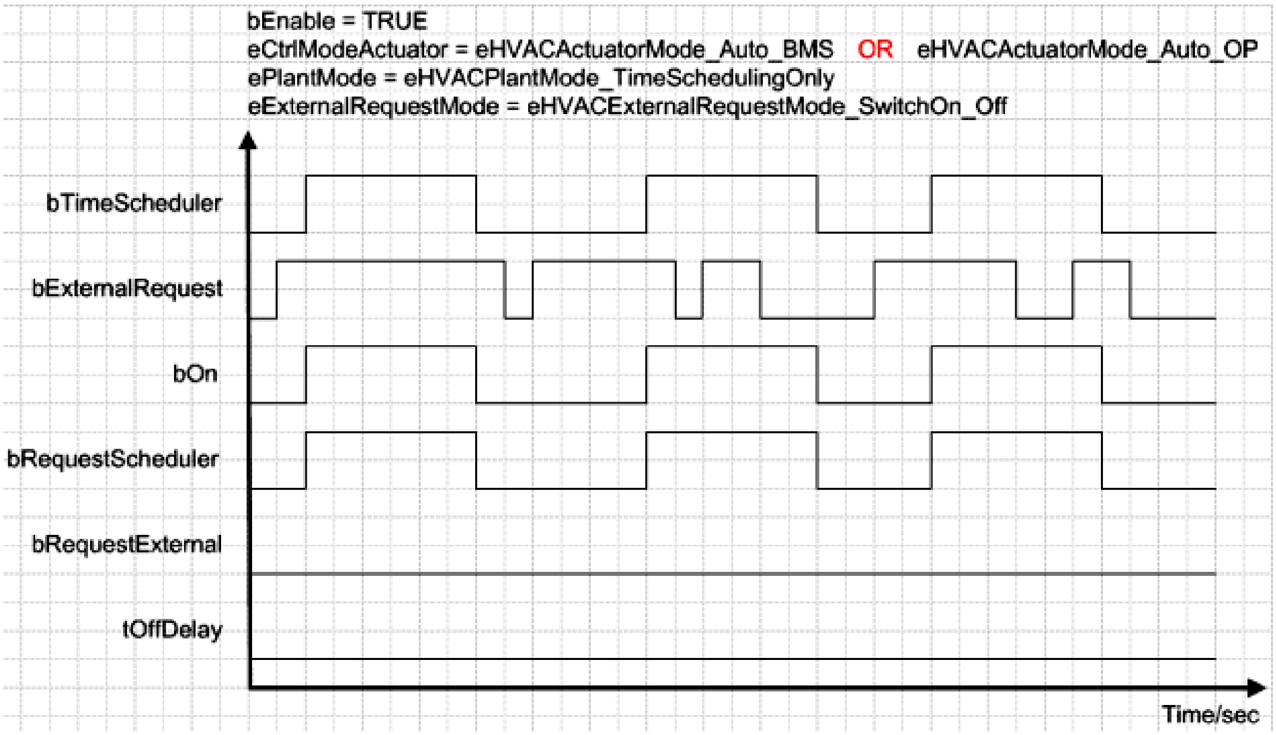
If an incorrect variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification. The variable is saved persistently. Preset to 0.

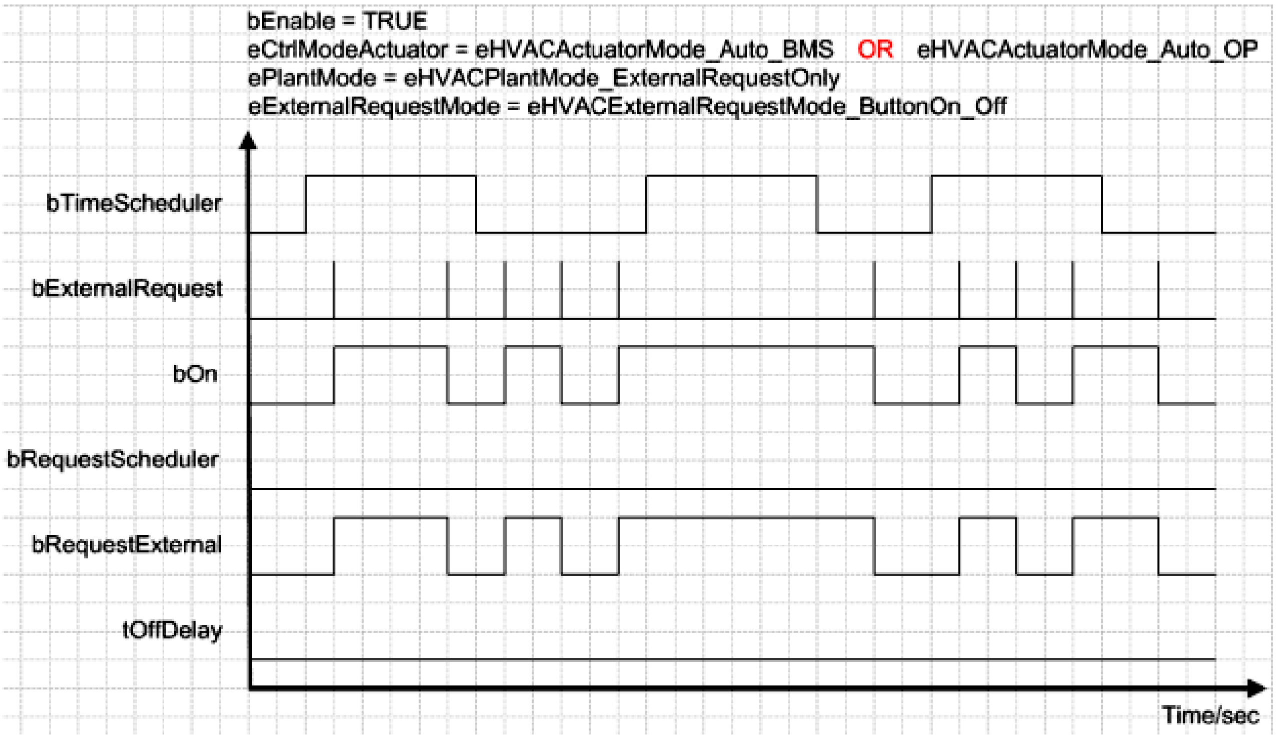
tOffDelay: Time value for the extension of the system utilization time. The extension of the utilization time can only be activated if *eModeExternalRequest* = 2. The variable is saved persistently. Preset to 30 min.

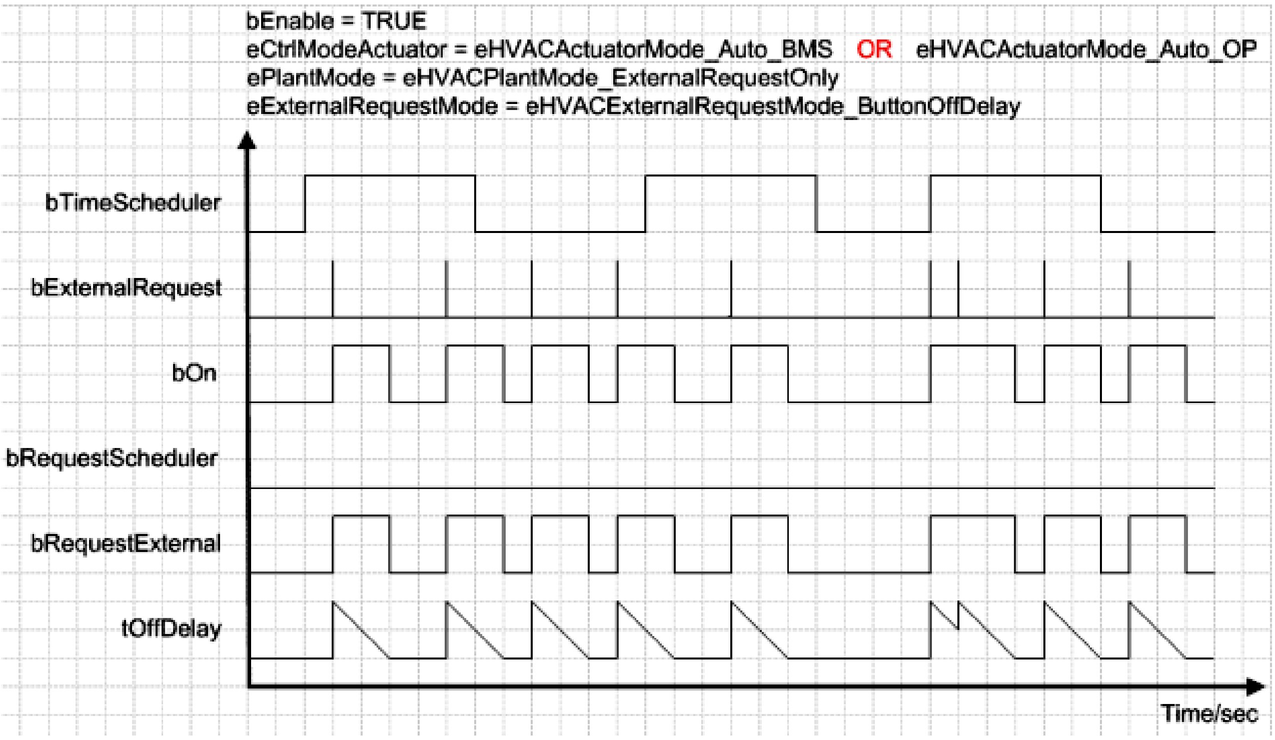
Behavior of the output value

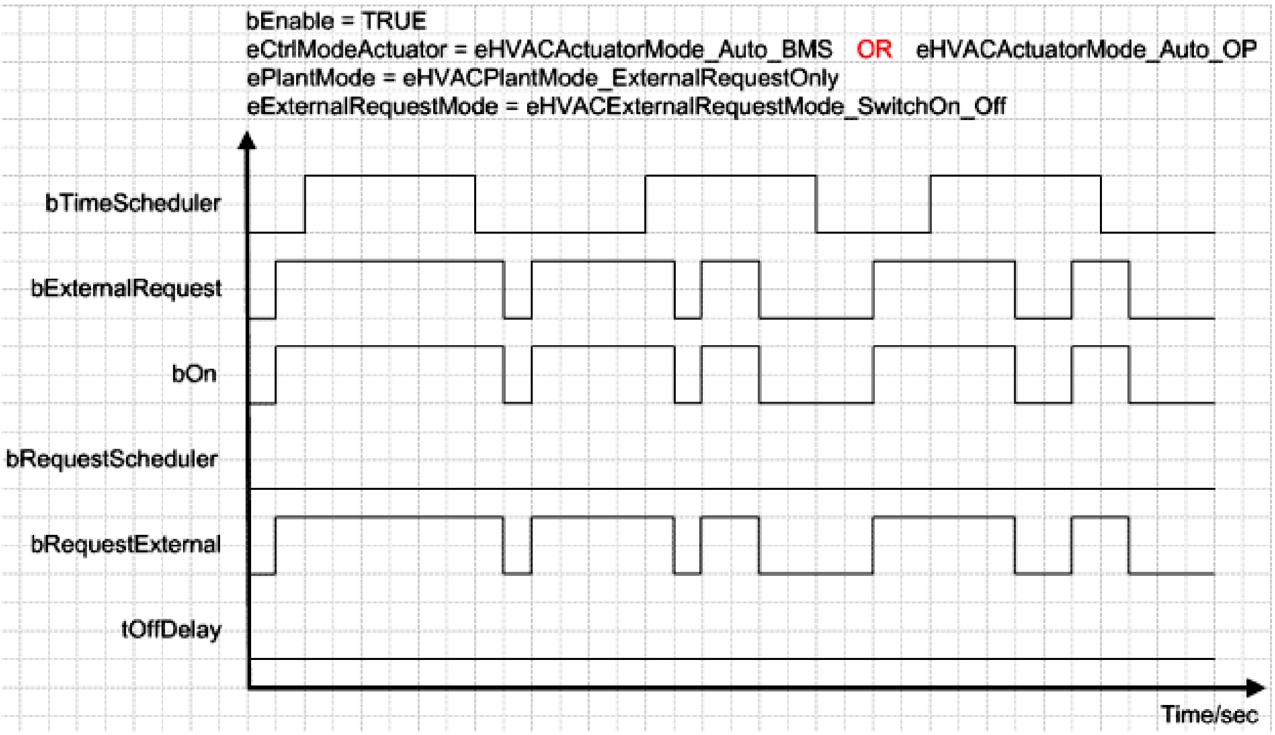


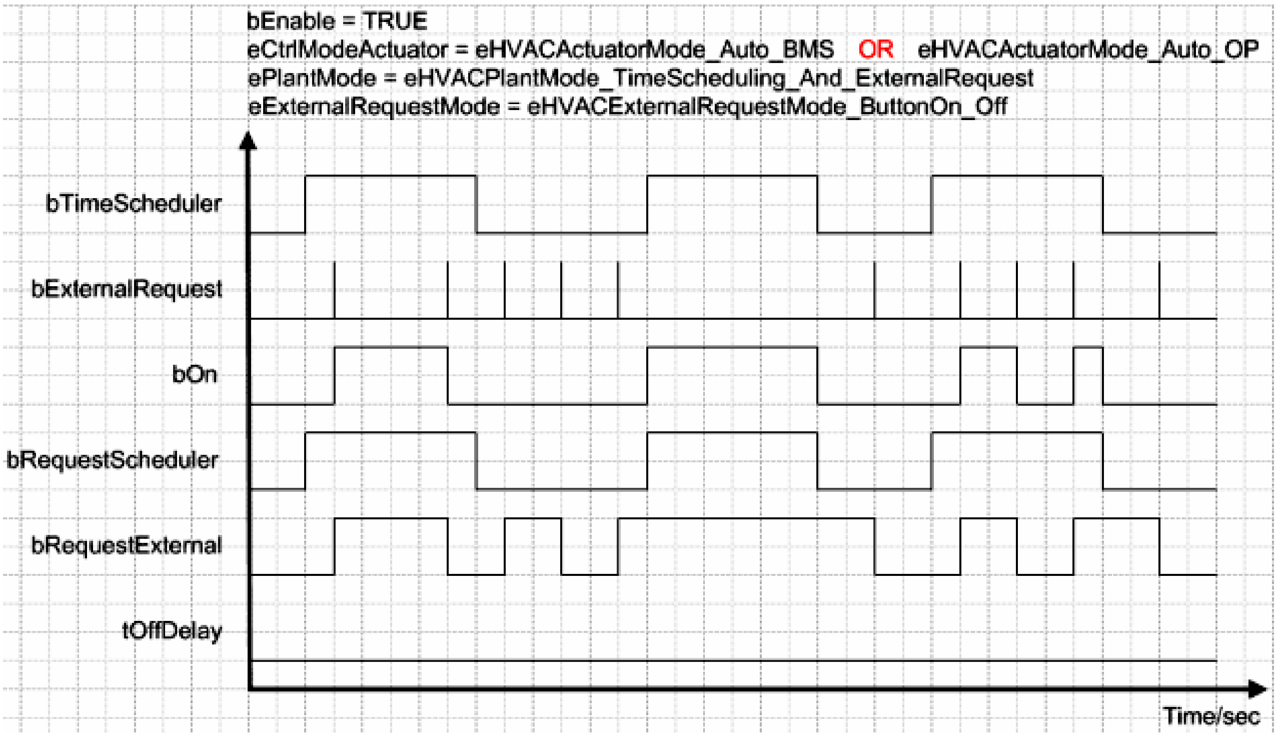


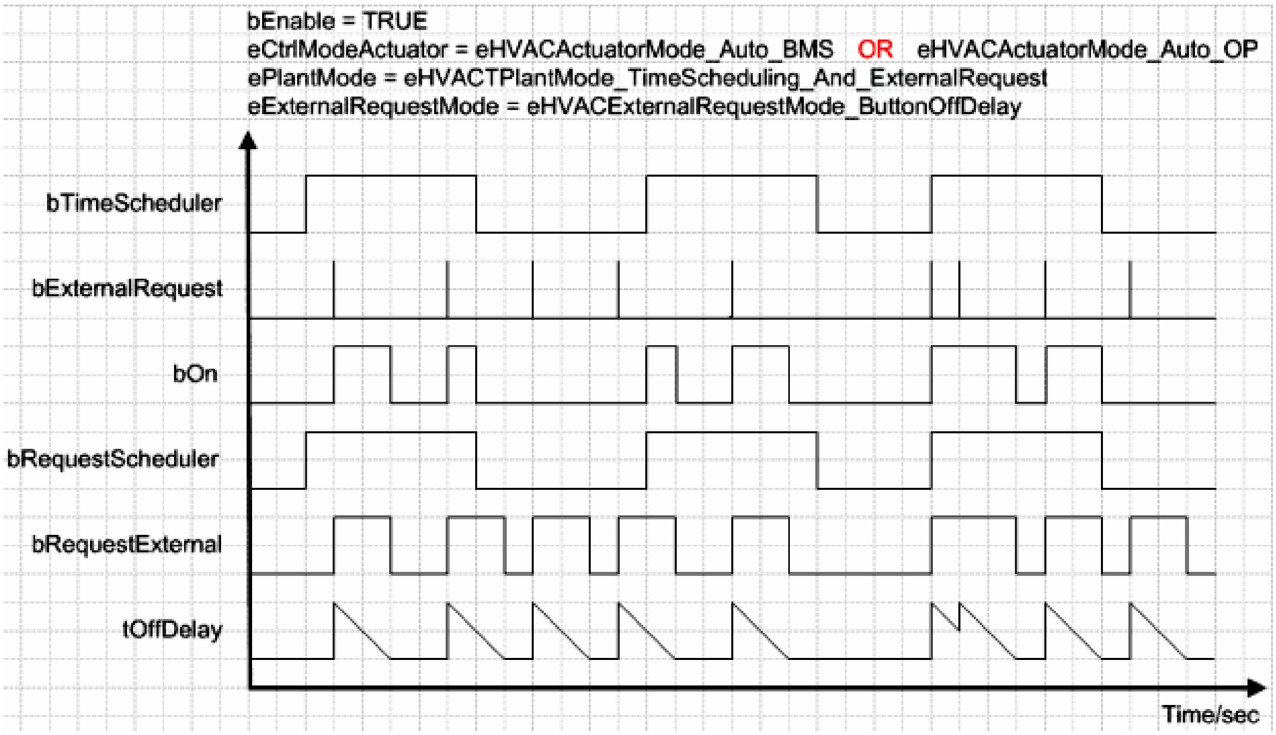

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeSchedulingOnly
eExternalRequestMode = eHVACExternalRequestMode_SwitchOn_Off
bTimeScheduler
bExternalRequest
bOn
bRequestScheduler
bRequestExternal
tOffDelay
Time/sec

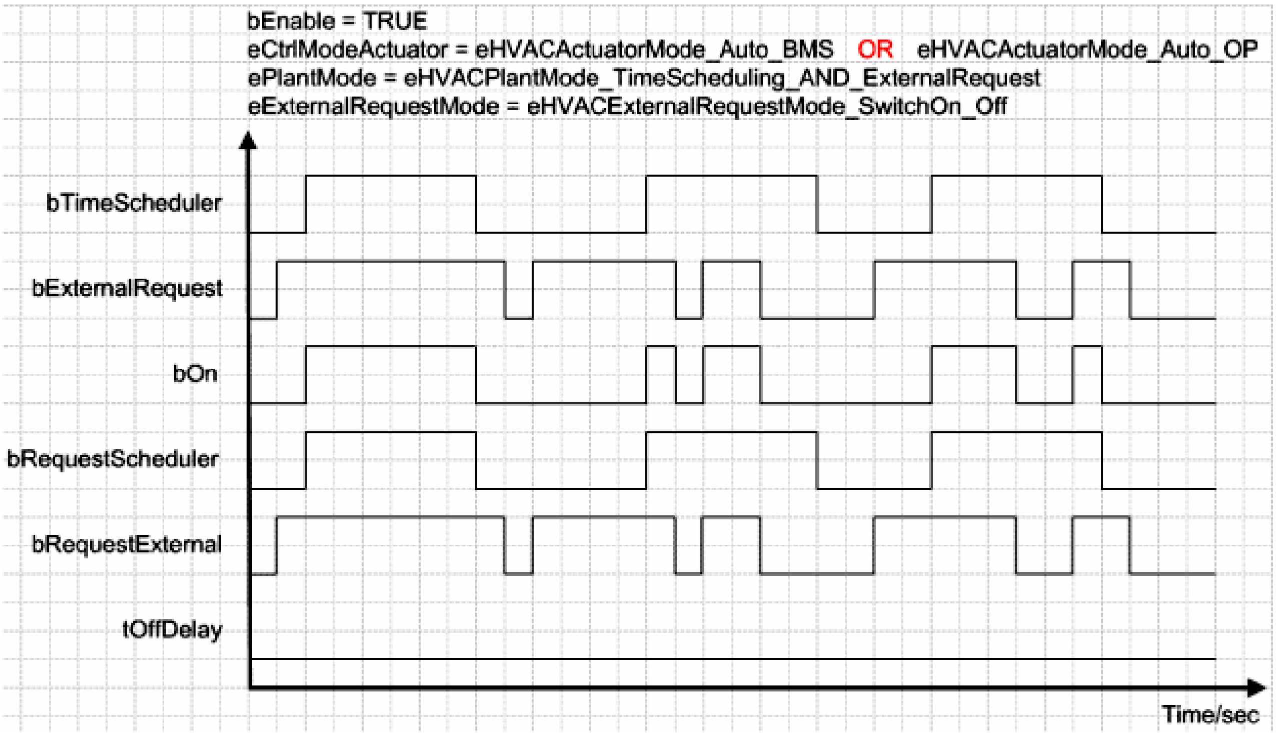

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_ExternalRequestOnly
eExternalRequestMode = eHVACExternalRequestMode_ButtonOn_Off
bTimeScheduler
bExternalRequest
bOn
bRequestScheduler
bRequestExternal
tOffDelay
Time/sec

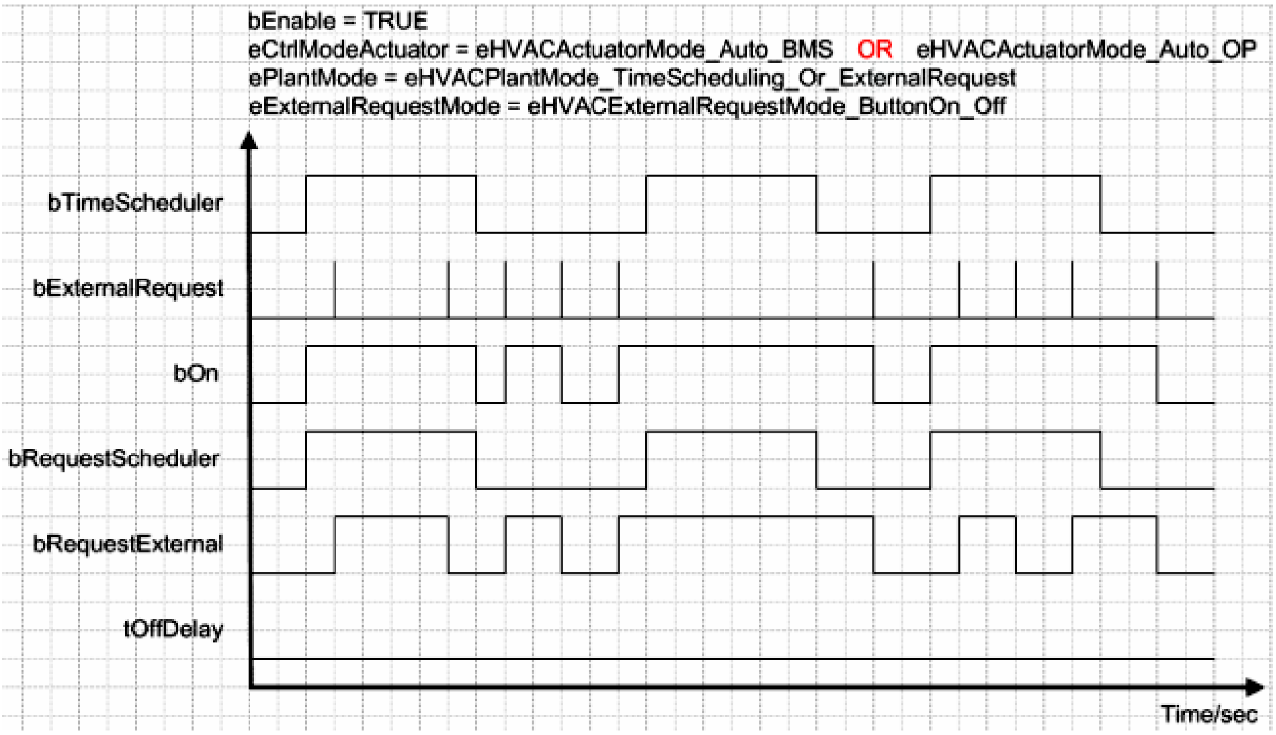

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_ExternalRequestOnly
eExternalRequestMode = eHVACExternalRequestMode_ButtonOffDelay
bTimeScheduler
bExternalRequest
bOn
bRequestScheduler
bRequestExternal
tOffDelay
Time/sec

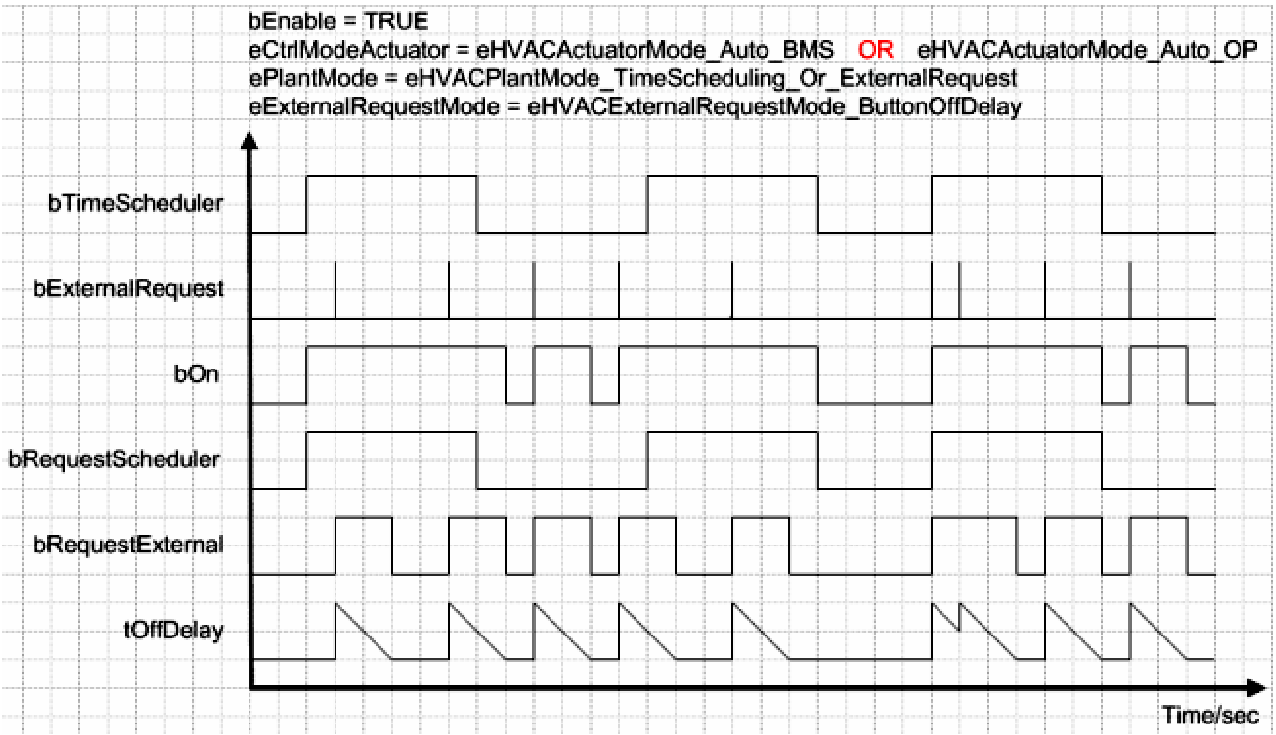

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_ExternalRequestOnly
eExternalRequestMode = eHVACExternalRequestMode_SwitchOn_Off
bTimeScheduler
bExternalRequest
bOn
bRequestScheduler
bRequestExternal
tOffDelay
Time/sec

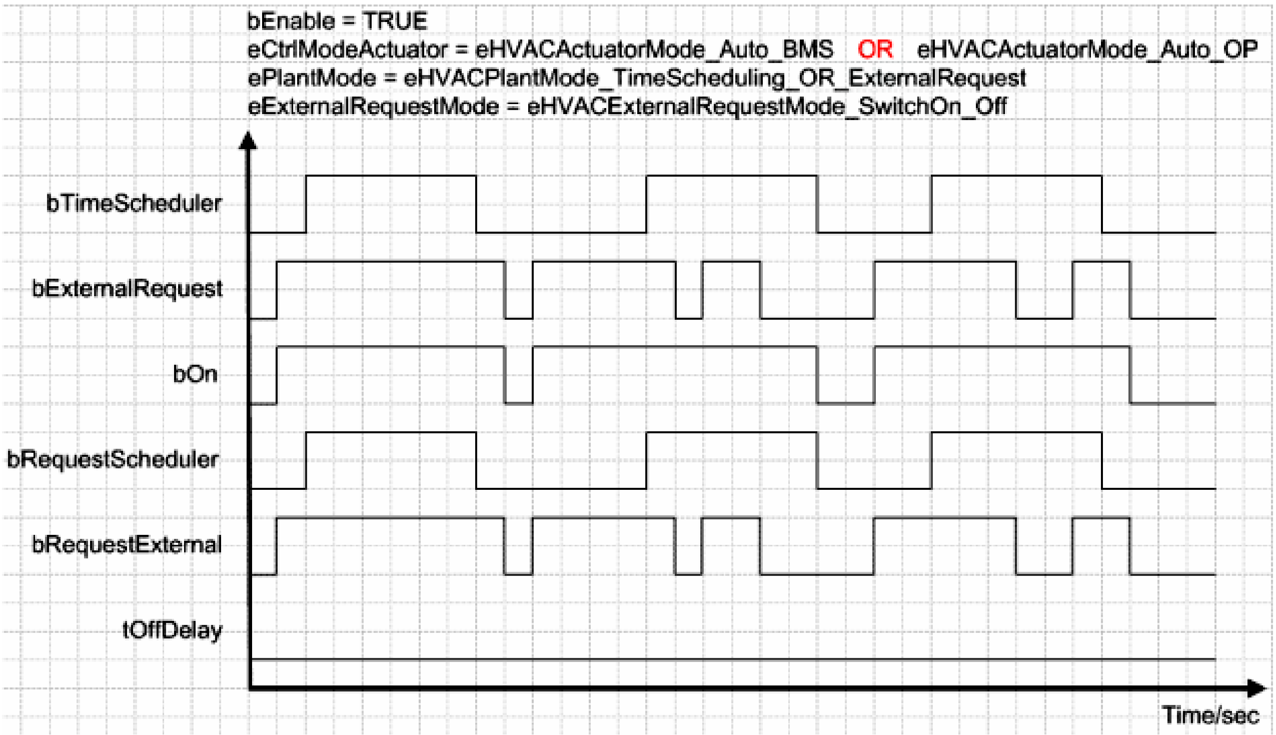

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeScheduling_And_ExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_ButtonOn_Off
bTimeScheduler
bExternalRequest
bOn
bRequestScheduler
bRequestExternal
tOffDelay
Time/sec


bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACTPlantMode_TimeScheduling_And_ExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_ButtonOffDelay
bTimeScheduler
bExternalRequest
bOn
bRequestScheduler
bRequestExternal
tOffDelay
Time/sec


bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeScheduling_AND_ExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_SwitchOn_Off
bTimeScheduler
bExternalRequest
bOn
bRequestScheduler
bRequestExternal
tOffDelay
Time/sec


bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeScheduling_Or_ExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_ButtonOn_Off
bTimeScheduler
bExternalRequest
bOn
bRequestScheduler
bRequestExternal
tOffDelay
Time/sec

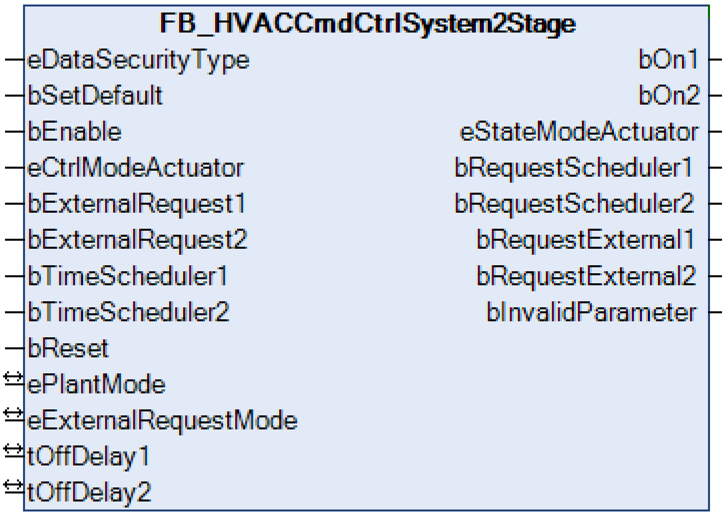




Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.8 FB_HVACCmdCtrlSystem2Stage



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 *FB_HVACCmdCtrlSystem2Stage* function block 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 [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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* are also FALSE.

eCtrlModeActuator: Enum that specifies the system operation modes Manual, Auto and Off (see E_HVACActuatorMode [▸ 489]). If specified incorrectly, operation continues internally with the last valid operation mode. This is *eHVACActuatorMode_Auto_BMS* in the case of initial commissioning. *bInvalidParameter* is set in the case of an incorrect parameter specification.

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: The system is enabled for stage 1 by this output variable.

bOn2: The system is enabled for stage 2 by this output variable. 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 (see E_HVACActuatorMode [▸ 489]).

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: With this enumeration variable various system functions can be executed in automatic operation mode, depending on the input variables *bExternalRequest1, bExternalRequest2, bTimeScheduler1* and *bTimeScheduler2* (see E_HVACPlantMode [▸ 492]).
ePlantMode = *eHVACPlantMode_TimeSchedulingOnly*: The system is switched to the respective stage exclusively via the input variables *bTimeScheduler1* or *bTimeScheduler2*.
ePlantMode = *eHVACPlantMode_TimeScheduling_And_ExternalRequest*: The system is switched to stage 1 if the input variables *bTimeScheduler1* AND *bExternalRequest1* are both TRUE. If the input variables *bTimeScheduler2* AND *bExternalRequest2* are both TRUE, the system is switched to stage 2.
ePlantMode = *eHVACPlantMode_TimeScheduling_Or_ExternalRequest*: The system is switched to the respective stage via the input variables *bExternalRequest1, bExternalRequest2 OR bTimeScheduler1, bTimeScheduler2*.
ePlantMode = *eHVACPlantMode_ExternalRequestOnly*: The system is switched to the respective stage exclusively via the input variable *bExternalRequest1* or *bExternalRequest2*.

If an incorrect variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

The variable is saved persistently. Preset to 0.

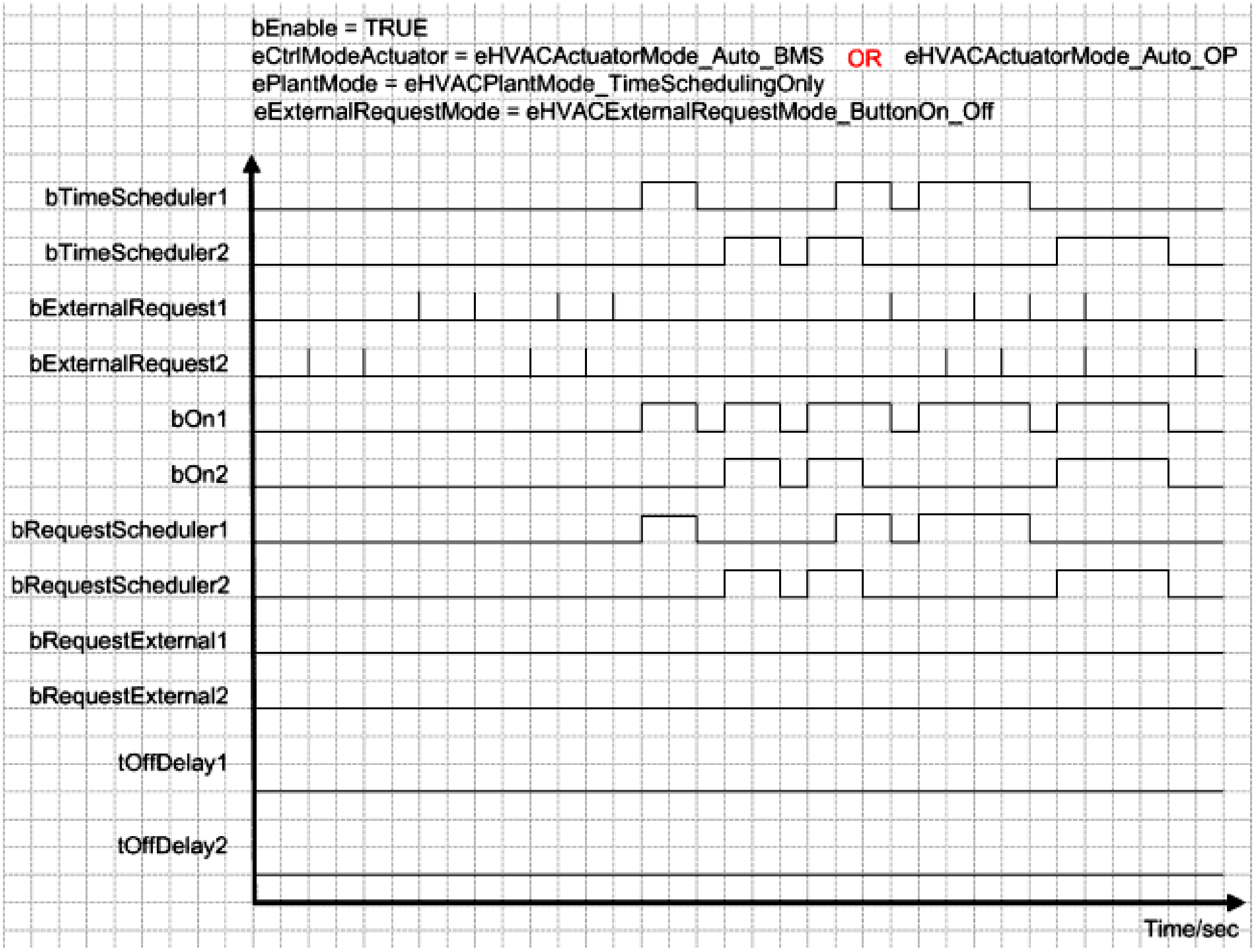
eExternalRequestMode: The enum *eExternalRequestMode* specifies the mode of operation of the input variables *bExternal Request1* and *bExternalRequest2* in automatic operation mode, depending on the enum *ePlantMode* (see E_HVACExternalRequestMode [▸ 492]).
eExternalRequestMode = *eHVACExternalRequestMode_ButtonOn_Off*: The external request is set to TRUE following a rising edge on *bExternalRequest1* or *bExternalRequest2*. A further rising edge resets the request to FALSE.
eExternalRequestMode = *eHVACExternalRequestMode_ButtonOffDelay*. Following a rising edge on the input variable *bExternalRequest1* or *bExternalRequest2*, the external request extends or sets the utilization time of the system in the respective stage by the time setting 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 no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

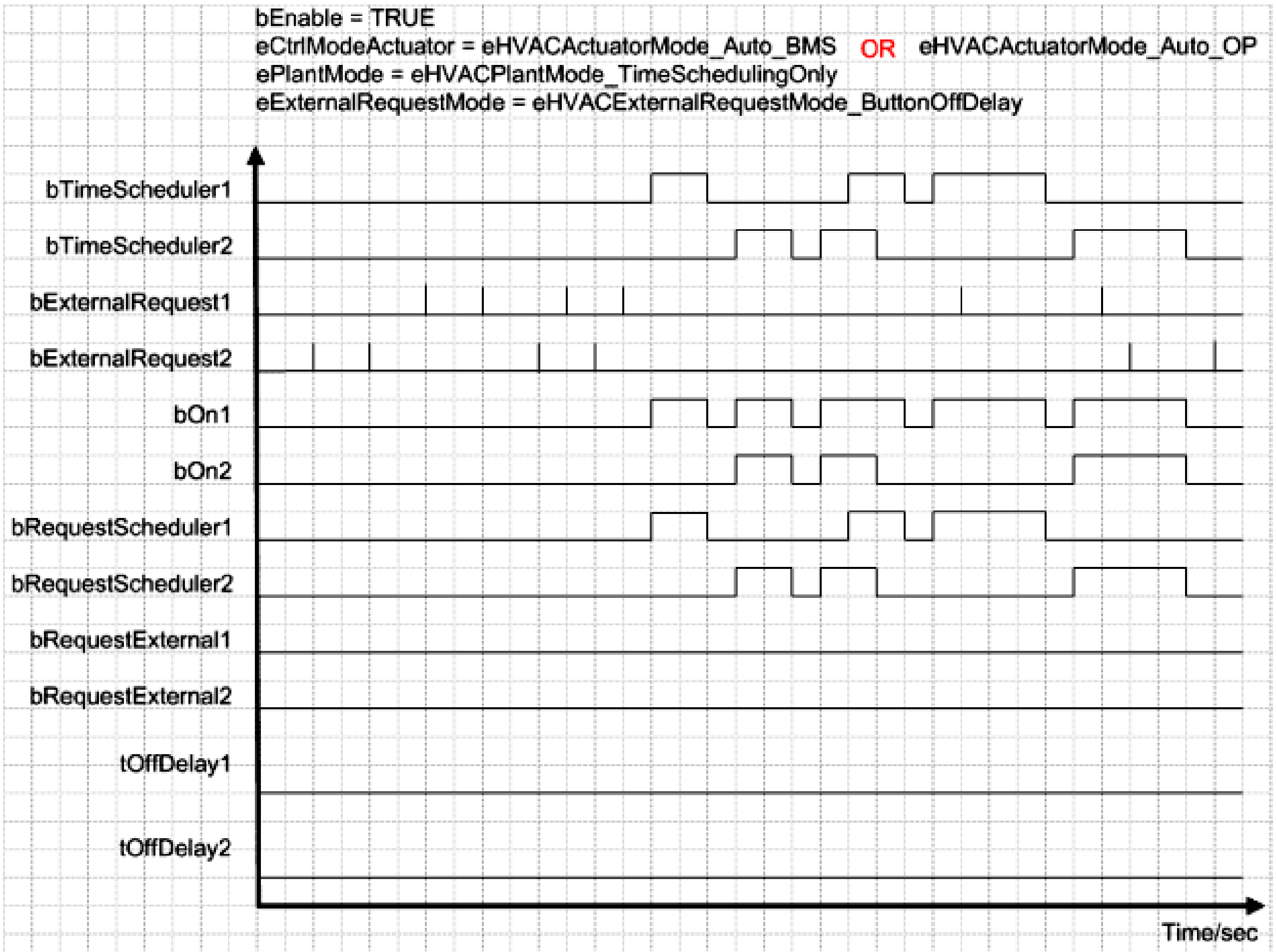
The variable is saved persistently. Preset to 0.

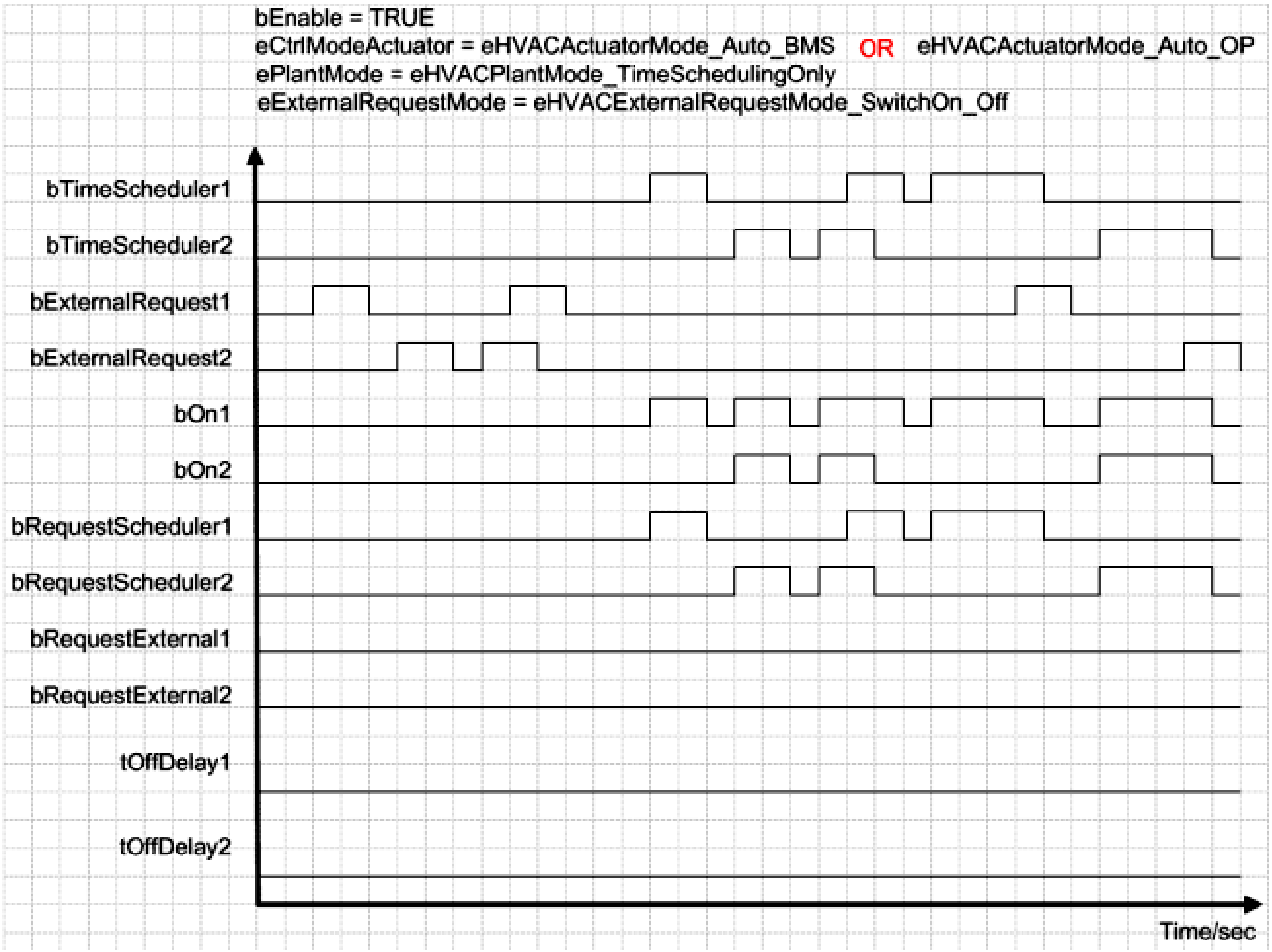
tOffDelay1: Time value for the extension of the system utilization time in stage 1. The extension of the utilization time can only be activated if *eModeExternalRequest* = 2. The variable is saved persistently. Preset to 30 min.

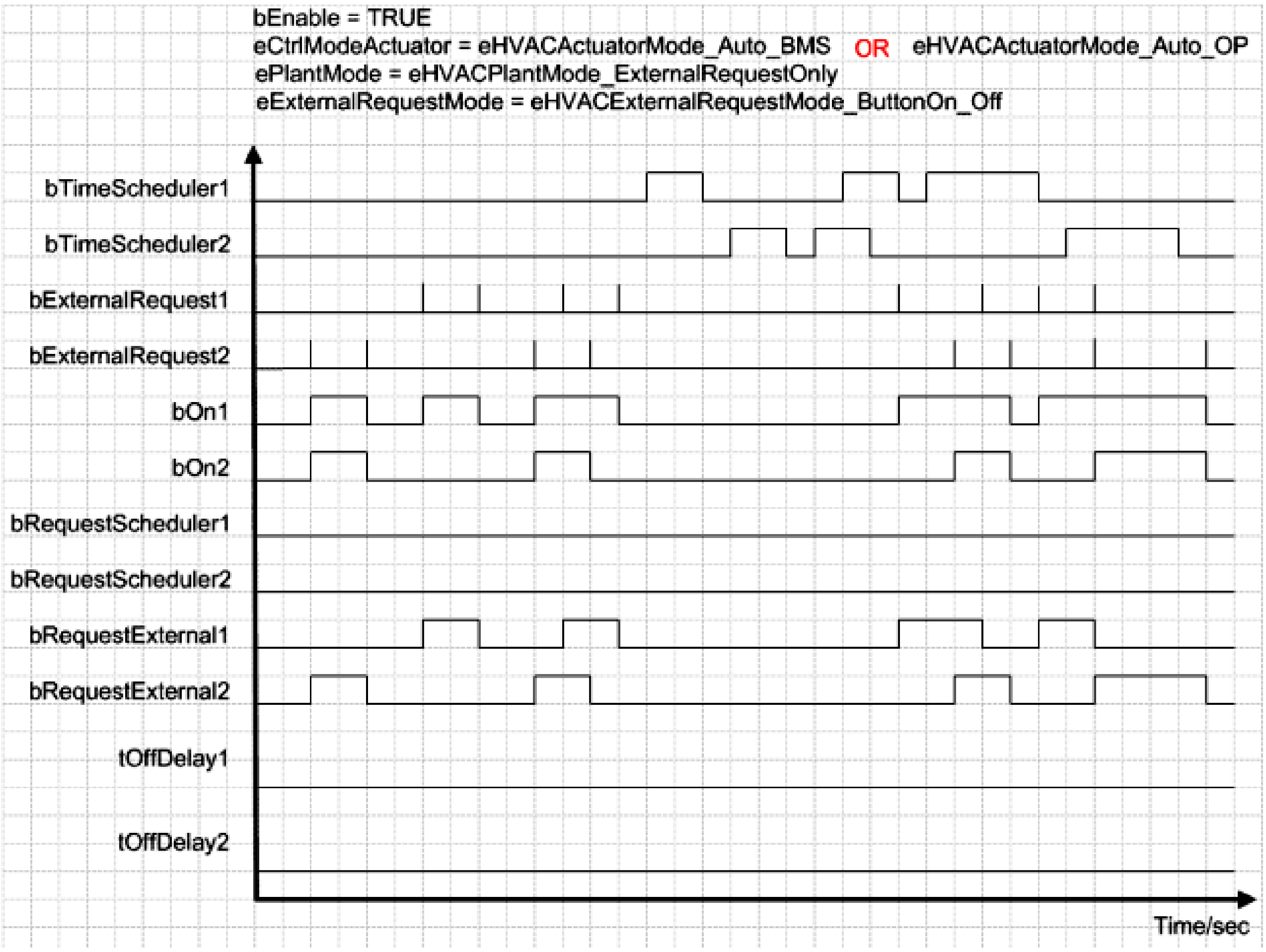
tOffDelay2: Time value for the extension of the system utilization time in stage 2. The extension of the utilization time can only be activated if *eModeExternalRequest* = 2. The variable is saved persistently. Preset to 30 min.

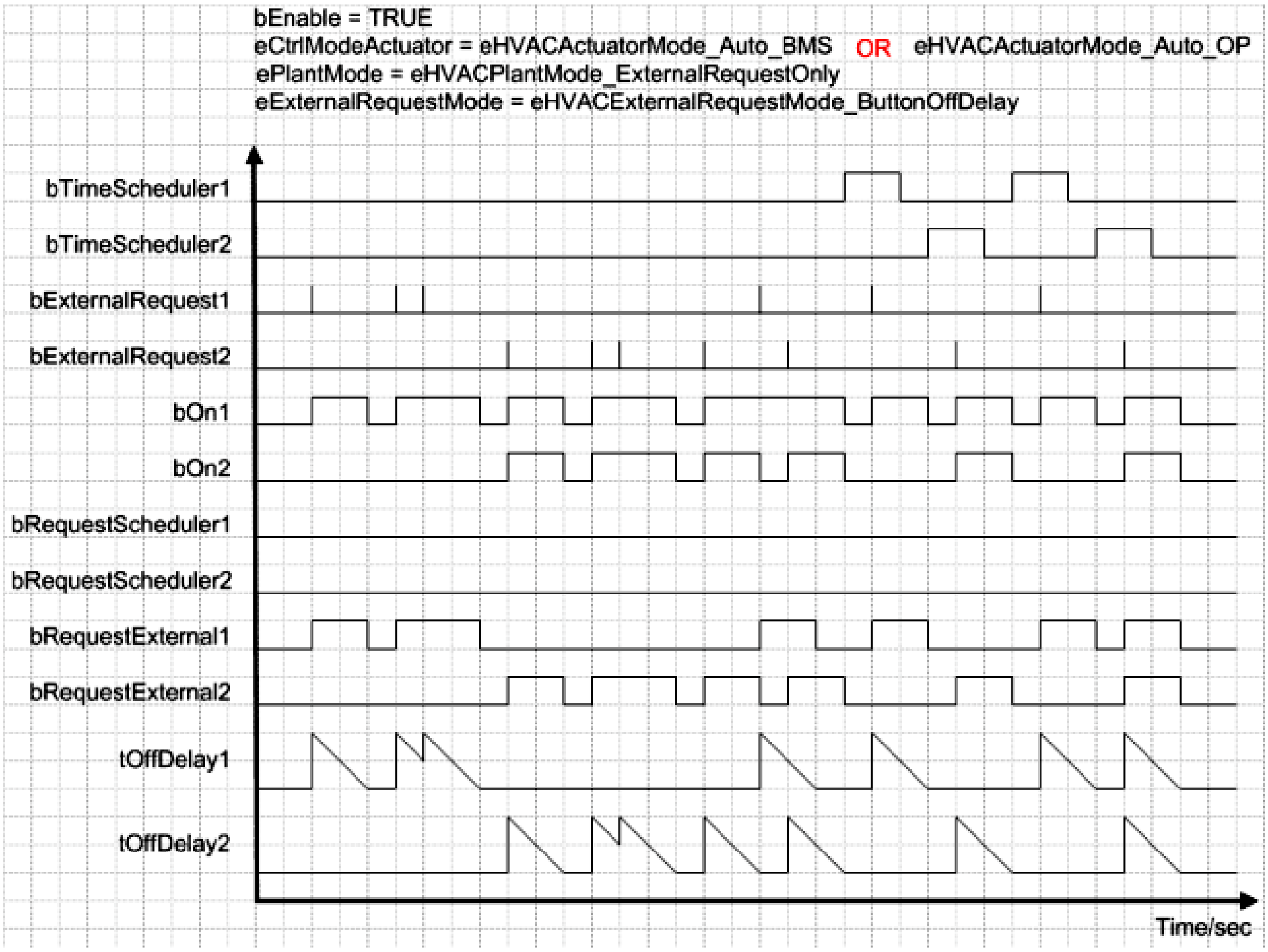
Behavior of the output value

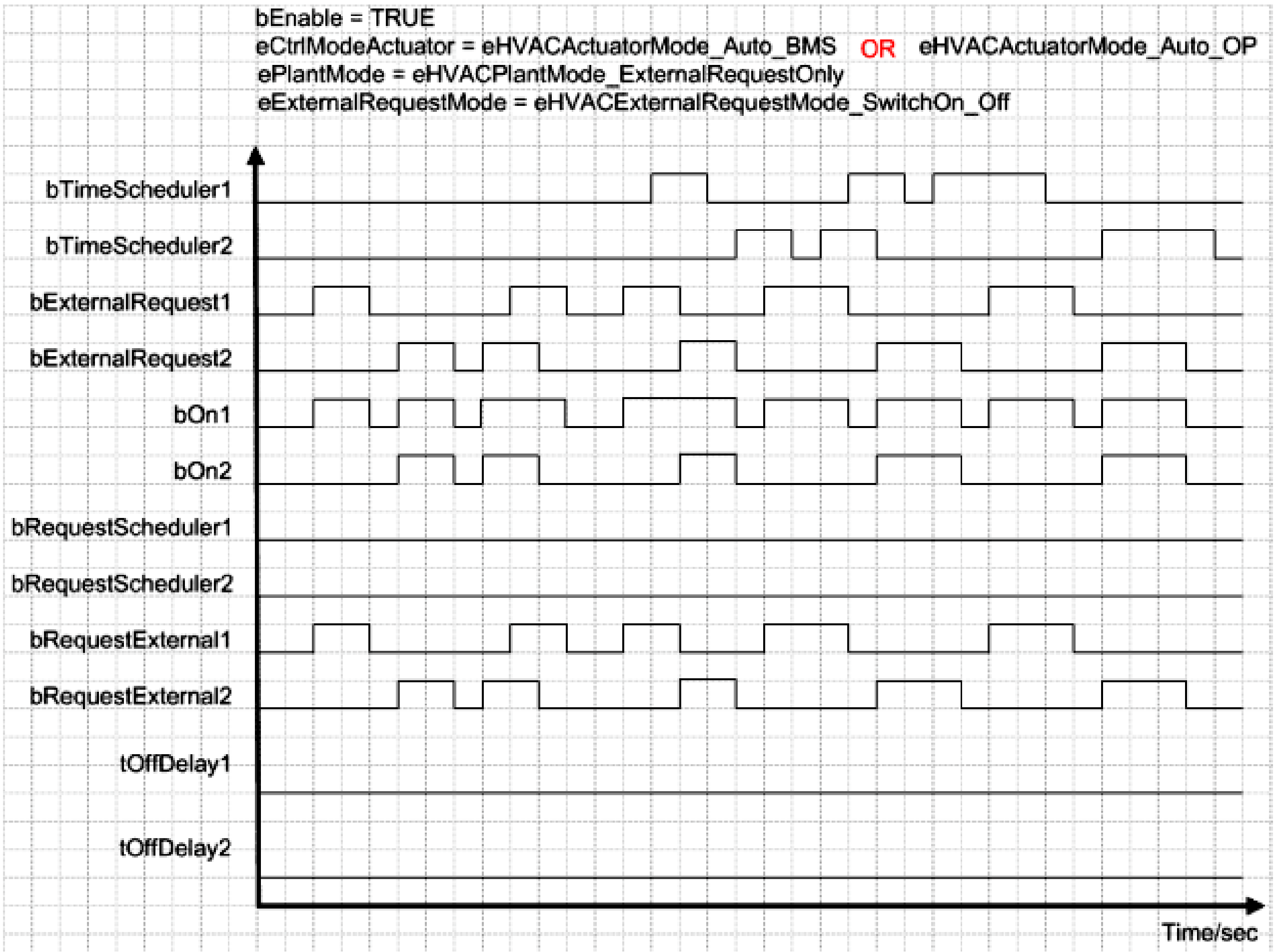


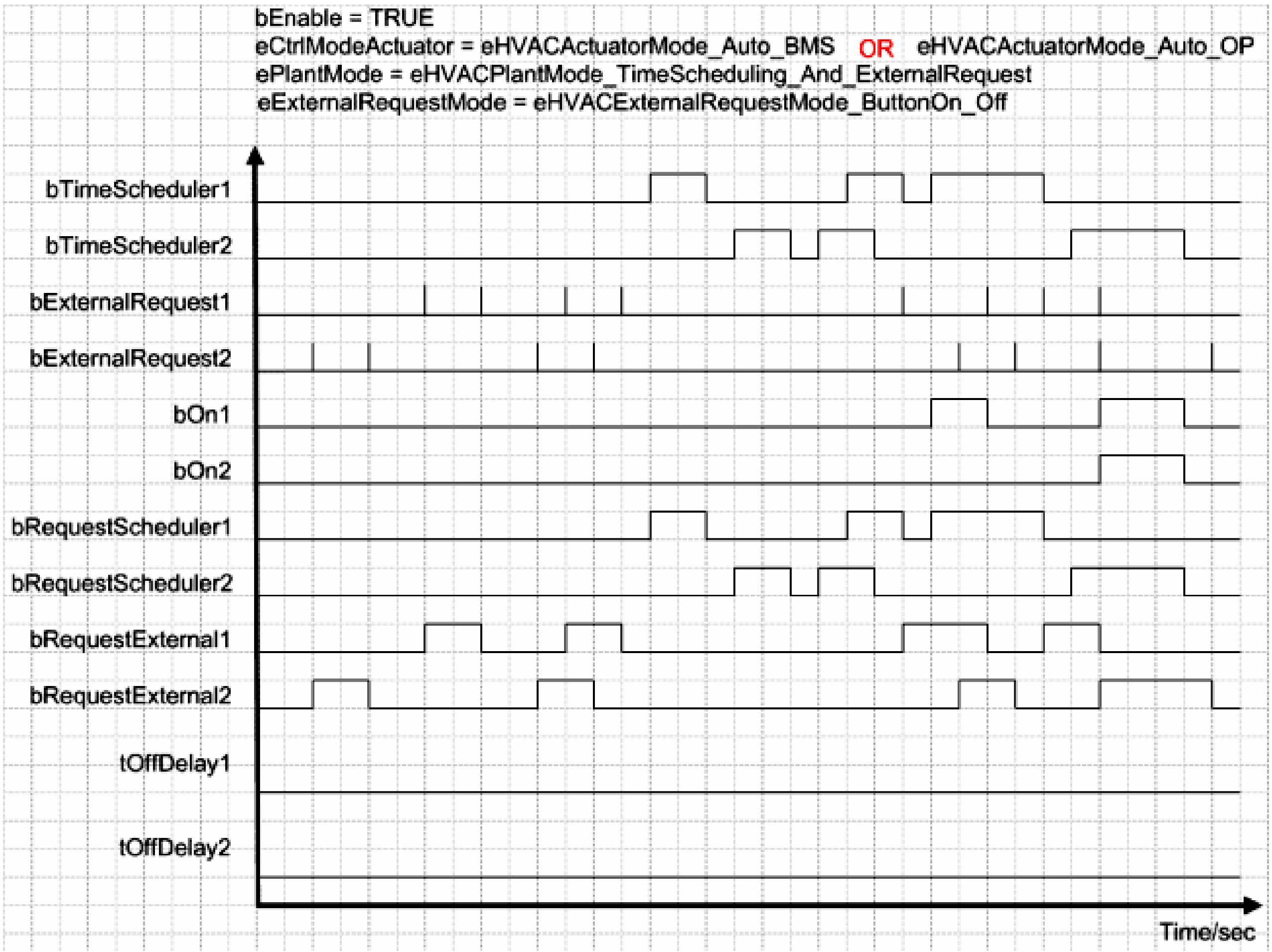

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeSchedulingOnly
eExternalRequestMode = eHVACExternalRequestMode_ButtonOffDelay
bTimeScheduler1
bTimeScheduler2
bExternalRequest1
bExternalRequest2
bOn1
bOn2
bRequestScheduler1
bRequestScheduler2
bRequestExternal1
bRequestExternal2
tOffDelay1
tOffDelay2
Time/sec

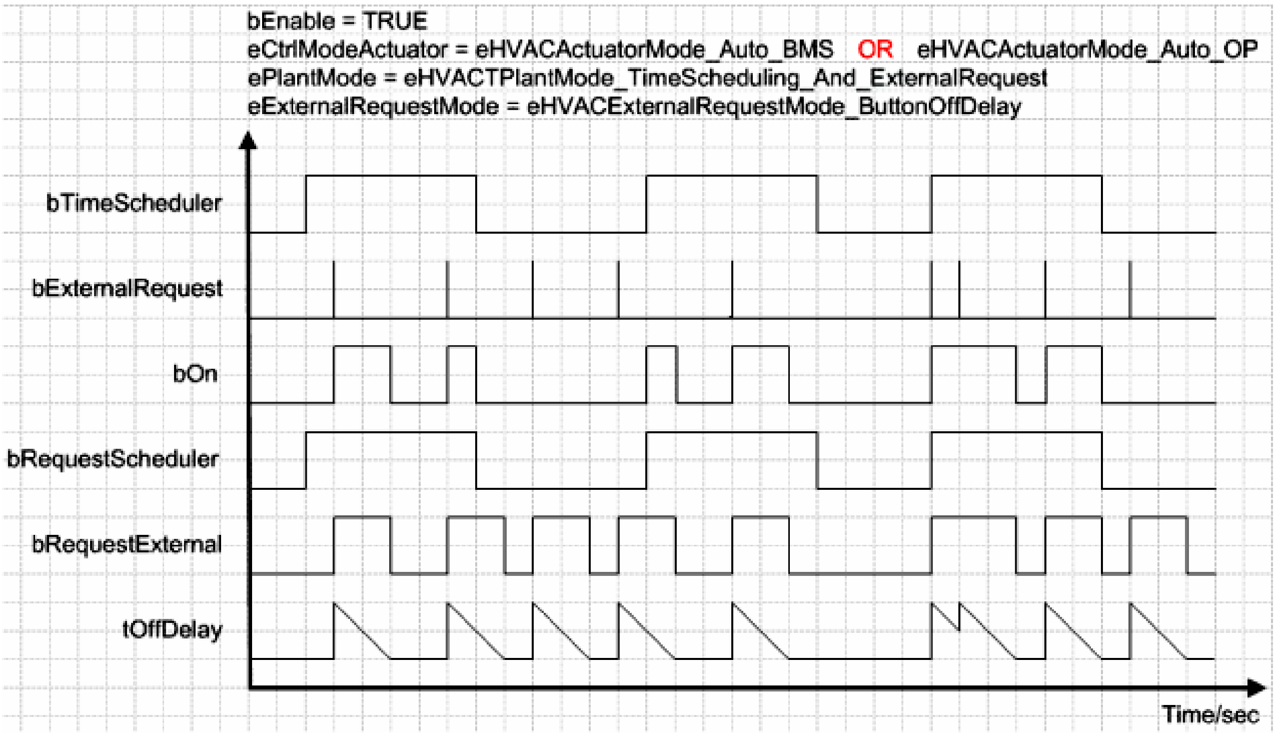

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeSchedulingOnly
eExternalRequestMode = eHVACExternalRequestMode_SwitchOn_Off
bTimeScheduler1
bTimeScheduler2
bExternalRequest1
bExternalRequest2
bOn1
bOn2
bRequestScheduler1
bRequestScheduler2
bRequestExternal1
bRequestExternal2
tOffDelay1
tOffDelay2
Time/sec

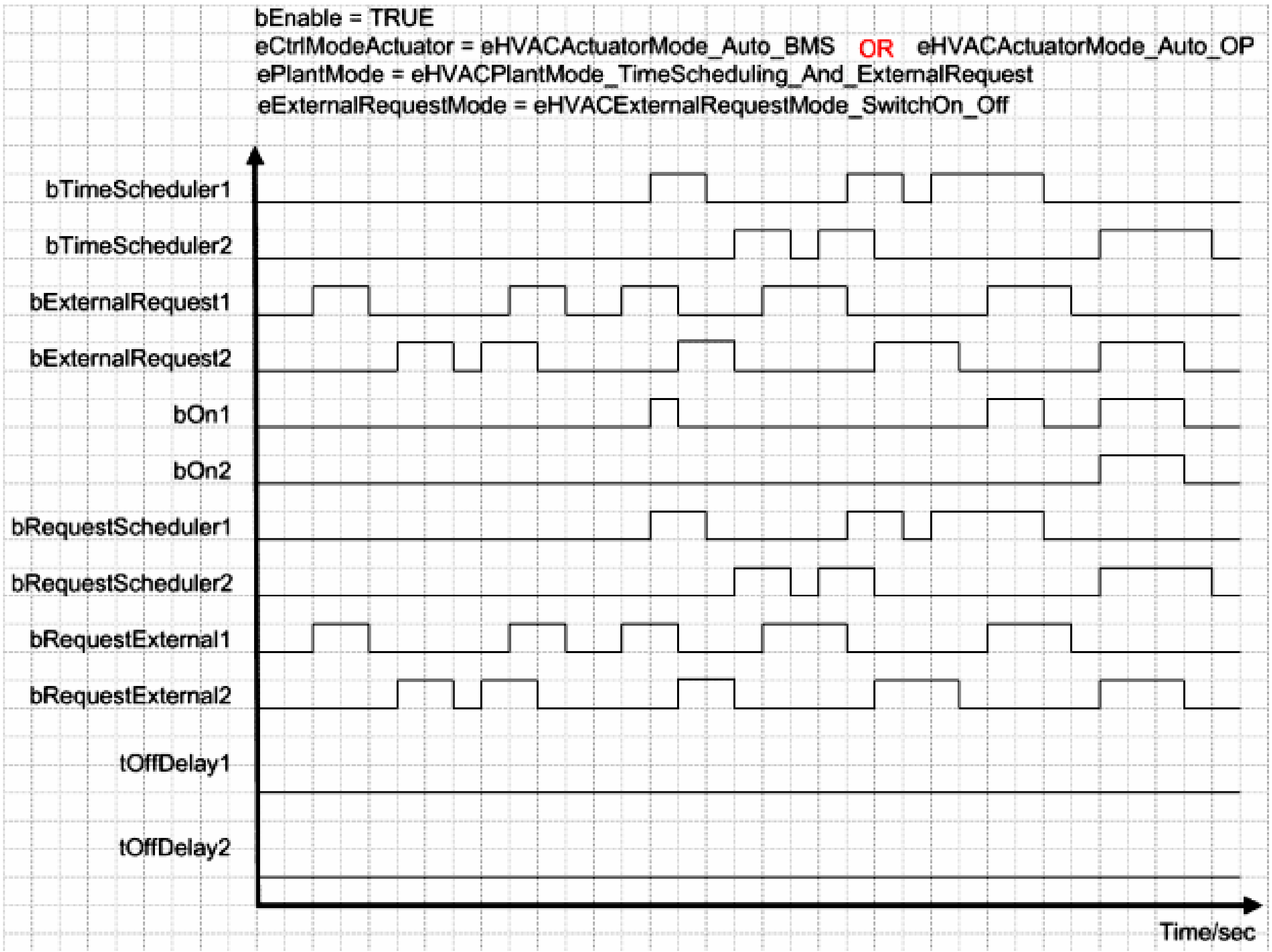

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_ExternalRequestOnly
eExternalRequestMode = eHVACExternalRequestMode_ButtonOn_Off
bTimeScheduler1
bTimeScheduler2
bExternalRequest1
bExternalRequest2
bOn1
bOn2
bRequestScheduler1
bRequestScheduler2
bRequestExternal1
bRequestExternal2
tOffDelay1
tOffDelay2
Time/sec

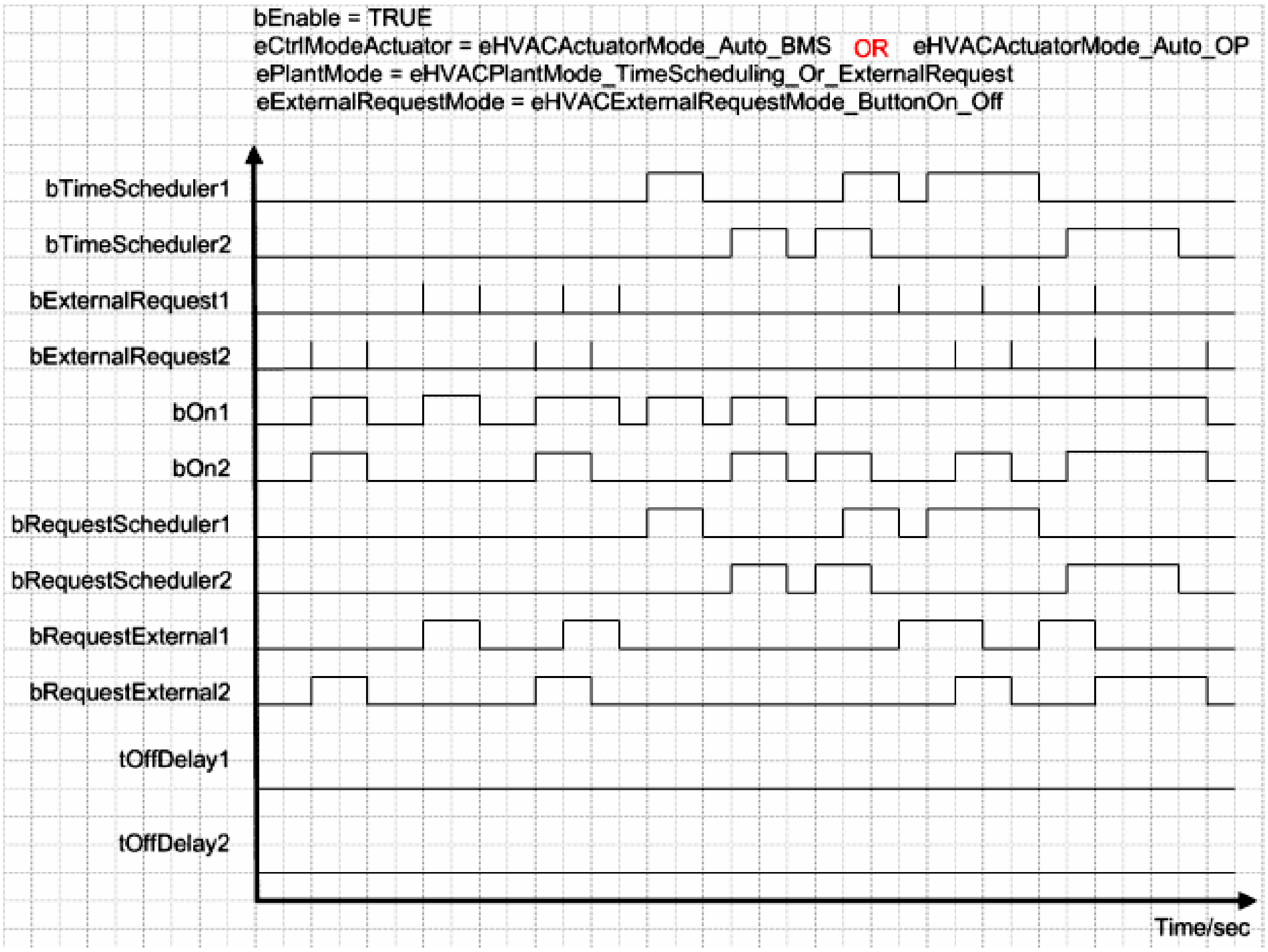

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_ExternalRequestOnly
eExternalRequestMode = eHVACExternalRequestMode_ButtonOffDelay
bTimeScheduler1
bTimeScheduler2
bExternalRequest1
bExternalRequest2
bOn1
bOn2
bRequestScheduler1
bRequestScheduler2
bRequestExternal1
bRequestExternal2
tOffDelay1
tOffDelay2
Time/sec

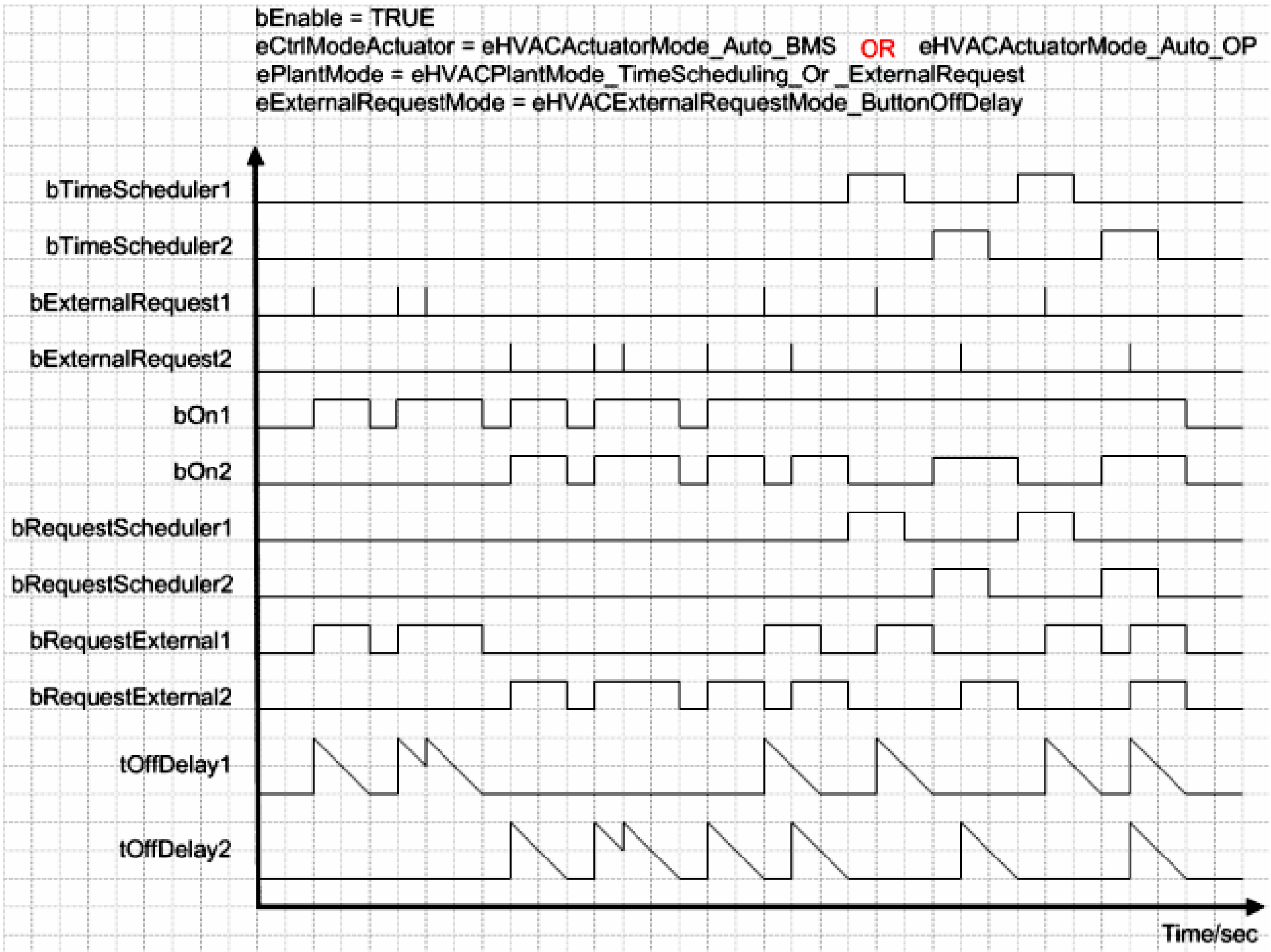

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_ExternalRequestOnly
eExternalRequestMode = eHVACExternalRequestMode_SwitchOn_Off
bTimeScheduler1
bTimeScheduler2
bExternalRequest1
bExternalRequest2
bOn1
bOn2
bRequestScheduler1
bRequestScheduler2
bRequestExternal1
bRequestExternal2
tOffDelay1
tOffDelay2
Time/sec

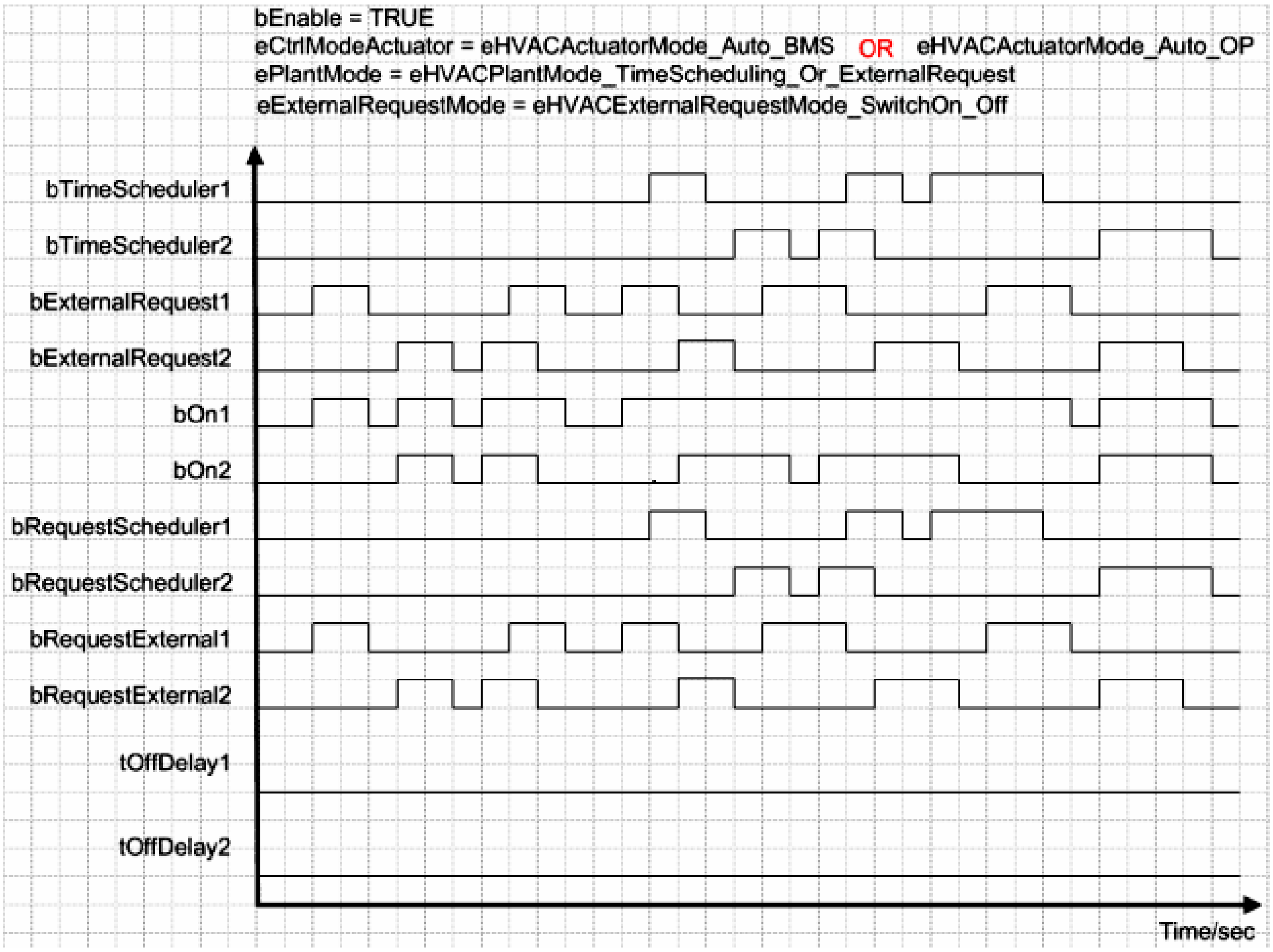

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeScheduling_And_ExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_ButtonOn_Off
bTimeScheduler1
bTimeScheduler2
bExternalRequest1
bExternalRequest2
bOn1
bOn2
bRequestScheduler1
bRequestScheduler2
bRequestExternal1
bRequestExternal2
tOffDelay1
tOffDelay2
Time/sec


bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACTPlantMode_TimeScheduling_And_ExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_ButtonOffDelay
bTimeScheduler
bExternalRequest
bOn
bRequestScheduler
bRequestExternal
tOffDelay
Time/sec


bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeScheduling_And_ExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_SwitchOn_Off
bTimeScheduler1
bTimeScheduler2
bExternalRequest1
bExternalRequest2
bOn1
bOn2
bRequestScheduler1
bRequestScheduler2
bRequestExternal1
bRequestExternal2
tOffDelay1
tOffDelay2
Time/sec


bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeScheduling_Or_ExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_ButtonOn_Off
bTimeScheduler1
bTimeScheduler2
bExternalRequest1
bExternalRequest2
bOn1
bOn2
bRequestScheduler1
bRequestScheduler2
bRequestExternal1
bRequestExternal2
tOffDelay1
tOffDelay2
Time/sec

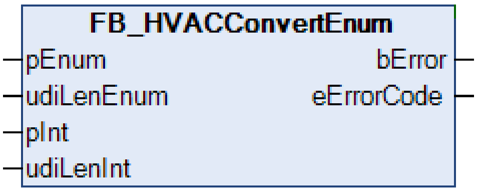

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeScheduling_Or _ExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_ButtonOffDelay
bTimeScheduler1
bTimeScheduler2
bExternalRequest1
bExternalRequest2
bOn1
bOn2
bRequestScheduler1
bRequestScheduler2
bRequestExternal1
bRequestExternal2
tOffDelay1
tOffDelay2
Time/sec



Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.9 FB_HVACConvertEnum



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 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;
```

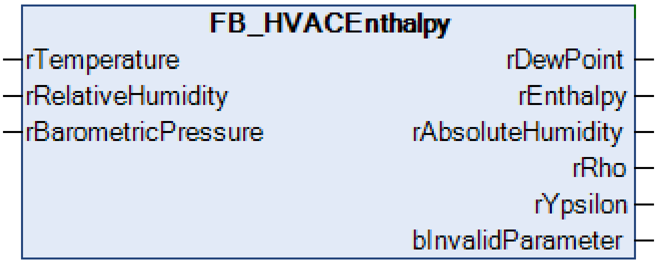
bError: This output indicates with a TRUE that there is an error. (The number of bytes of which the data types Integer or Enum consist is wrong.)

eErrorCode: Supplies the error number [▶ 491] when the output *bError* is set. The following errors can occur in this function block: *eHVACErrorCodes_Error_LEN_Int* (43), *eHVACErrorCodes_Error_LEN_Enum* (44) (see E_HVACErrorCodes [▶ 491]).

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.10 FB_HVACEnthalpy



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 pressure are required for calculating these parameters. The enthalpy is a measure for the energy of a thermodynamic system.

VAR_INPUT

```
rTemperature           : REAL;
rRelativeHumidity      : REAL;
rBarometricPressure    : REAL;
```

rTemperature: Input for the temperature value [°C].

rRelativeHumidity: Input for the relative humidity [%]. The value must equal or greater 1.

rBarometricPressure: Input for the air pressure [hPa].

VAR_OUTPUT

```
rDewPoint           : REAL;
rEnthalpy           : REAL;
rAbsoluteHumidity   : REAL;
rRho                : REAL;
rYpsilon            : REAL;
bInvalidParameter   : BOOL;
```

rDewPoint: Dew point [°C].

rEnthalpy: Enthalpy [kJ / kg].

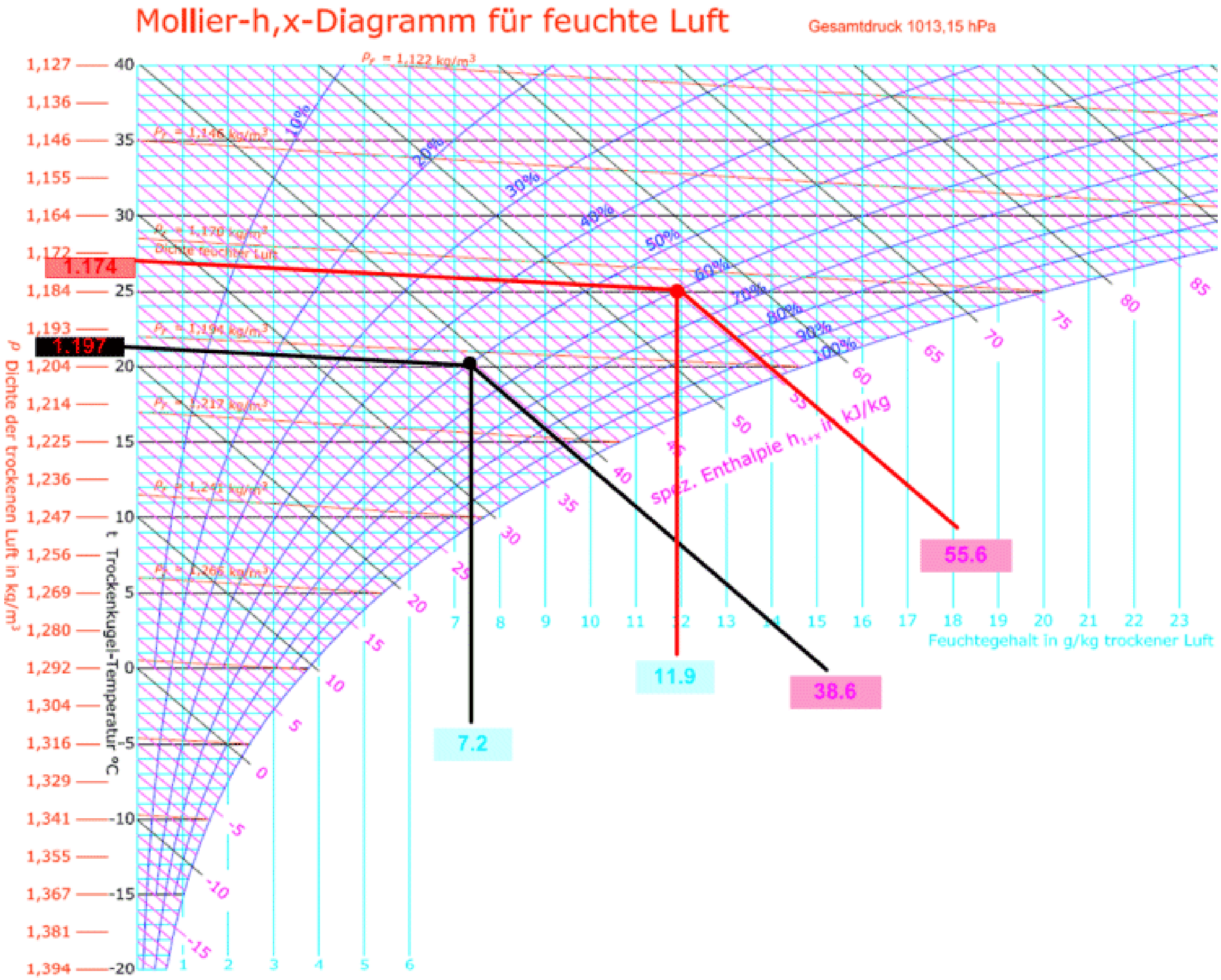
rAbsoluteHumidity: Absolute humidity [kg/kg]. The result must be multiplied by 1000 in order to obtain the absolute humidity in [g/kg].

rRho: Density of moist air, which specifies the mixture as a mass in relation to the volume. The unit is [kg/m³].

rYpsilon: Specific volume per 1 kg of dry air. The unit is [m³/kg] dry air.

bInvalidParameter: Set to TRUE if the input variable *rRelativeHumidity* or *rBarometricPressure* is smaller than or equal to zero.

H,x-diagram [www.air2000.de]:



Example 1:

rTemperature: = 20 °C
rRelativeHumidity:= 50%
rBarometricPressure:= 1013.15hPa

Results from the function block: rDewPoint:= 9.4°C
rEnthalpy:= 38.6kJ/kg
rAbsoluteHumidity:= 0.0072kg/kg, converted to g/kg ==> 7.2 g/kg

Example 2:

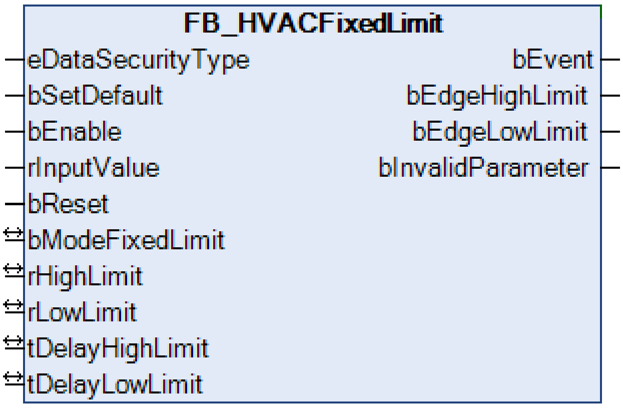
rTemperature: = 25 °C
rRelativeHumidity:= 60%
rBarometricPressure:= 1013.15hPa

Results from the function block: rDewPoint:= 17°C
rEnthalpy:= 55.6kJ/kg
rAbsoluteHumidity:= 0.0119kg/kg, converted to g/kg ==> 11.9 g/kg

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.11 FB_HVACFixedLimit



This function block represents a limit value 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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 the 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*.

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*.

bInvalidParameter: Indicates that an incorrect parameter is present at one of the variables *rHighLimit, rLowLimit, tDelayHighLimit* or *tDelayHighLimit*. An incorrect parameter entry does not lead to a standstill of the block; see description of variables. After rectifying the incorrect parameter entry, the message *bInvalidParameter* 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* undershoots 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 undershot 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 no correct variable value is present, then the last valid variable value, if one exists, will be taken. 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 function block continues to work normally.

The variable is saved persistently. Preset to 22.

rLowLimit: Lower limit value.
If no correct variable value is present, then the last valid variable value, if one exists, will be taken. 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 function block continues to work normally.

The variable is saved persistently. Preset to 18.

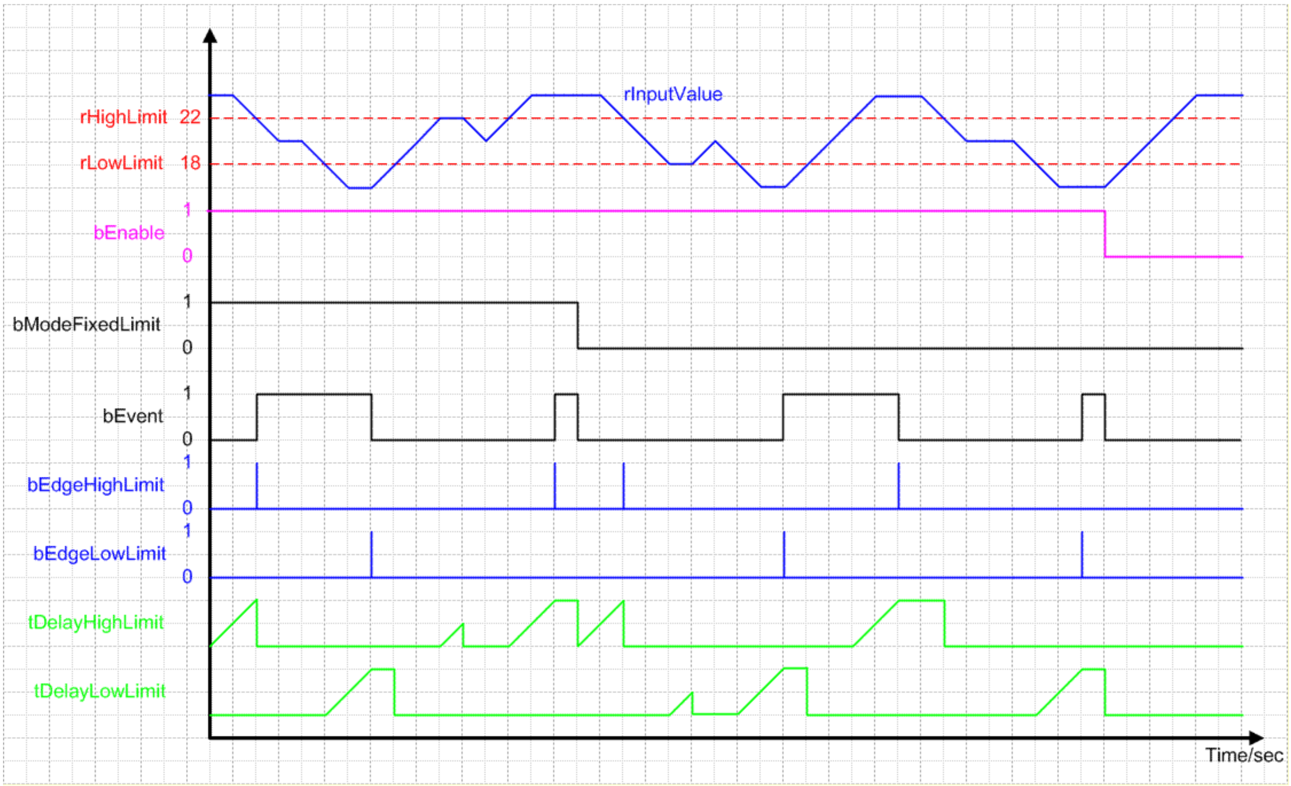
tDelayHighLimit: Switching delay [s] when the upper limit value is exceeded.
If no correct variable value is present, then the last valid variable value, if one exists, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* will be set in the event of an incorrect parameter entry; the function block continues to work normally.

The variable is saved persistently. Preset to 1 s.

tDelayLowLimit: Switching delay [s] when the lower limit value is undershot.
If no correct variable value is present, then the last valid variable value, if one exists, will be taken. 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 function block continues to work normally.

The variable is saved persistently. Preset to 1 s.

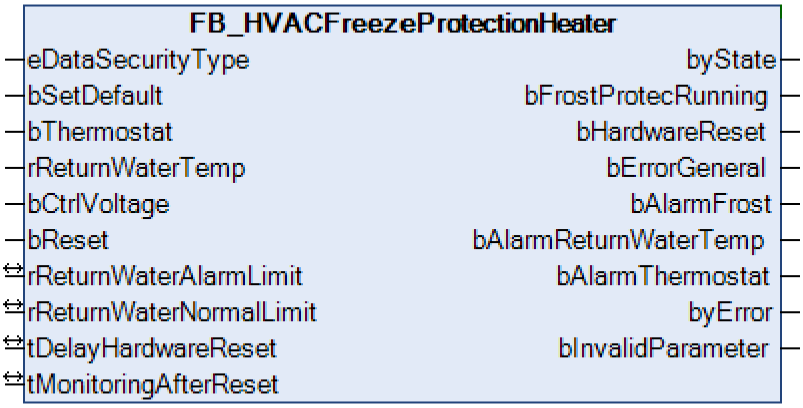
Behavior of the output value



Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.12 FB_HVACFreezeProtectionHeater



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 is activated if the return flow temperature of the air heater *rReturnWaterTemp* is lower than the limit value *rReturnWaterAlarmLimit* or if the input *bThermostat* is FALSE = error.
For each of these errors, the respective message *bAlarmReturnWaterTemp* and/or *bAlarmThermostat* at the output of the function block is 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 a fault 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.

Once the system has restarted following an automatic reset, the timer *tMonitoringAfterReset* is started.
If another frost warning occurs during 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 [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 monitor. *If control voltage is present bCtrlVoltage* = TRUE.

bReset: Input for acknowledging the fault.

VAR_OUTPUT

```
byState                : BYTE;
bFrostProtecRunning    : BOOL;
bHardwareReset         : BOOL;
bErrorGeneral          : BOOL;
bAlarmReturnWaterTemp  : BOOL;
bAlarmThermostat       : BOOL;
byError                : BYTE;
bInvalidParameter      : BOOL;
```

byState: Output of the state as a byte.

byState.0:= bFrostProtecRunning;
byState.1:= bHardwareReset;
byState.2:= bThermostat;
byState.7:= bCtrlVoltage;

bFrostProtecRunning: TRUE if the frost protection program is active.

bHardwareReset: Output that indicates with a 1 s pulse that the frost protection relay is self-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 there is an incorrect input parameter. *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.

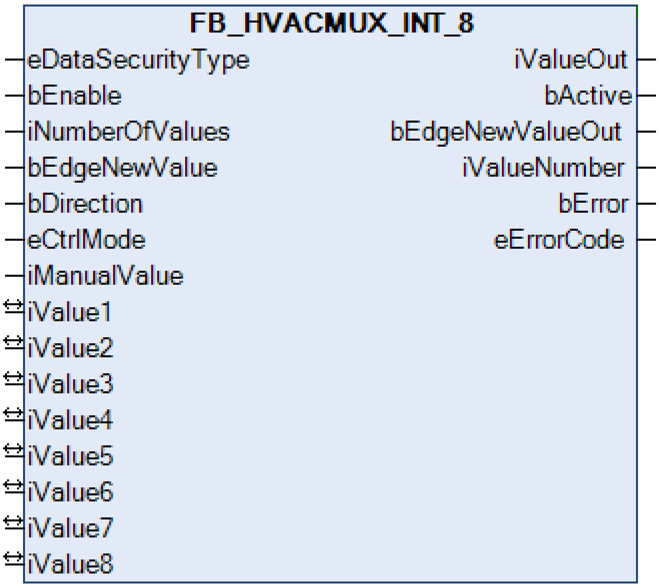
tDelayHardwareReset: Time that elapses, after input, between the input of the frost error and output of a reset pulse in order to return the system to normal operation. The variable is saved persistently.

tMonitoringAfterReset: Monitoring time after the first frost warning. The variable is saved persistently.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.13 FB_HVACMUX_INT_8



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 [▶ 490].
The following conditions must first be fulfilled for this:

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 *iValue***X**, 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* is incremented. 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* is decremented. 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 *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 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.

VAR_INPUT

```
eDataSecurityType  : E_HVACDataSecurityType;
bEnable            : BOOL;
iNumberOfValues    : INT;
bEdgeNewValue      : BOOL;
bDirection         : BOOL;
eCtrlMode          : E_HVACCtrlMode;
iManualValue       : INT;
```

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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* and *iValueNumber* are set constantly to 0.

iNumberOfValues: Indicates the number of variables *iValue1-8* that can be output via *iValueOut*. If *iNumberOfValues* = 8 is, then the output *iValueNumber* varies between 1 and 8. The variables *iValue1* to *iValue8* are then taken into account for the output *iValueOut* (see Application [▸ 385]).
iNumberOfValues determines the start behavior of the outputs *iValueOut* and *iValueNumber*, (see Application [▸ 385]).
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: *iValueNumber* is incremented or decremented if there is a rising edge on *bEdgeNewValue*.
If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *iNumberOfValues* > 0 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* is incremented. If *iValueNumber* = 6, then *iValueOut* = *iValue6*. If *iValueNumber* = *iNumberOfValues* and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = 1 and hence *iValueOut* = iValue1.
If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *iNumberOfValues* = **6** and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* is decremented. If *iValueNumber* = 5, then *iValueOut* = *iValue5*. If *iValueNumber* = 1 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues***(6)** and hence *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 from 1 to *iNumberOfValues* in ascending form and thus determines the output value *iValueOut* (see Application [▸ 385]).

bDirection = TRUE means that *iValueNumber* decrements from *iNumberOfValues* to 1 in descending form and thus determines the output value *iValueOut* (see Application [▸ 385]).
bDirection is not taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual.*

eCtrlMode: This enum specifies the operation mode of the function block (see E_HVACCtrlMode [▸ 490]).
If the operation mode *eCtrlMode* = *eHVACCtrlMode_Auto*, then the variable *iManualValue* is not taken into account.
If the operation mode *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 contents of the variable *iValue15* are then output via *iValueOut* (see Application [▸ 385]).
iManualValue is not taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual.* The value of *iValueNumber* is determined by *iManualValue*. *iManualValue* should lie within the range of 0 to 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 on *bEdgeNewValue*, then *iValueNumber* is incremented. If *iValueNumber* = 6, then *iValueOut* = *iValue6*. If *iValueNumber* = *iNumberOfValues* and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = 1 and hence *iValueOut* = iValue1.
If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *iNumberOfValues* = **7** and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* is decremented. If *iValueNumber* = 5, then *iValueOut* = *iValue5*. If *iValueNumber* = 1 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues***(7)** and hence *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* is as follows: if *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *iNumberOfValues* > 0, then *iValueNumber* = 1 and *iValueOut* = *iValue1.*

- *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *iNumberOfValues* = 7, then *iNumberOfValues* = 7 and *iValueOut* = *iValue7.*

- *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 13, then *iValueNumber* = 13 and *iValueOut* = *iValue13.*

- *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 0, then *iValueNumber* = 0 and *iValueOut* = *0.*

bActive: *bActive* goes 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 *iValue***X**, 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 there is an error and that there is a wrong parameter on the variable *iNumberOfValues*. *iValueOut* and *iValueNumber* are set constantly to 0 and the Enum *eErrorCode* indicate the error code. The message *bError* does not have to be acknowledged after rectification of the error.

eErrorCode: Supplies the error number [▶ 491] when the output *bError* is set. The following error can occur in this function block:

eHVACErrorCodes_InvalidParam_iNumberOfValues: There is an incorrect value on *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

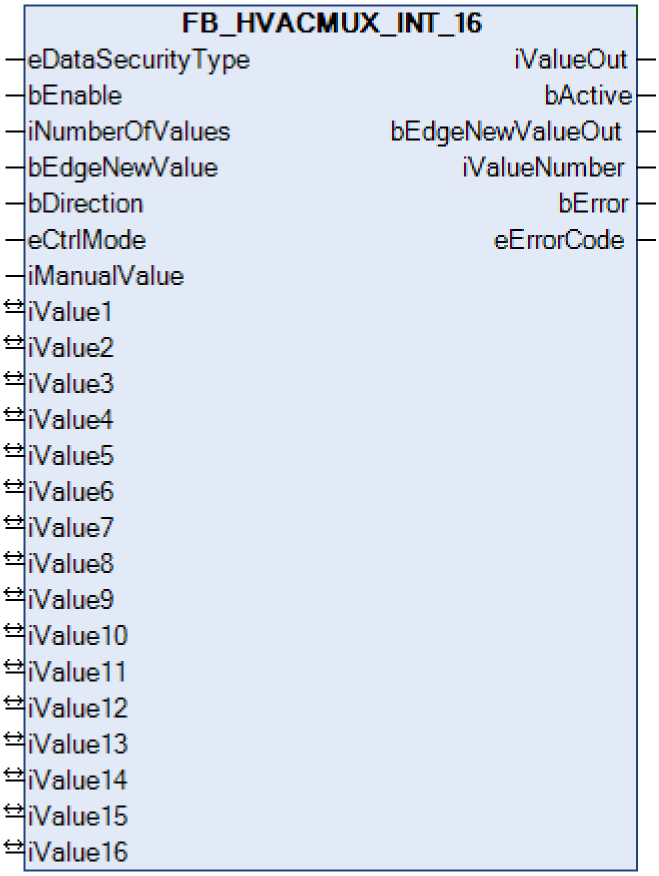
```
iValue1-8          : INT;
```

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 is, then the output *iValueNumber* varies between 1 and 8. The variables *iValue1* to *iValue8* are then taken into account for the output *iValueOut* if the operation mode *eCtrlMode* = *eHVACCtrlMode_Auto* (see Application [▶ 385]).

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.14 FB_HVACMUX_INT_16



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 [▶ 490].
The following conditions must first be fulfilled for this:

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 *iValue***X**, 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* is incremented. 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* is decremented. 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 *iValueOut* = *iValue1.*

- If *bDirection* = TRUE AND *iNumberOfValues* = 12, then *iNumberOfValues* = 12 and *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 *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 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.

VAR_INPUT

```
eDataSecurityType  : E_HVACDataSecurityType;
bEnable            : BOOL;
iNumberOfValues    : INT;
bEdgeNewValue      : BOOL;
bDirection         : BOOL;
eCtrlMode          : E_HVACCtrlMode;
iManualValue       : INT;
```

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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* and *iValueNumber* are set constantly to 0.

iNumberOfValues: Indicates the number of variables *iValue1-16* that can be output via *iValueOut*. If *iNumberOfValues* = 8 is, then the output *iValueNumber* varies between 1 and 8. The variables *iValue1* to *iValue8* are then taken into account for the output *iValueOut* (see Application [▸ 389]).
iNumberOfValues determines the start behavior of the outputs *iValueOut* and *iValueNumber.*
iNumberOfValues is not taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual* (see Application [▸ 389]).
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: *iValueNumber* is incremented or decremented if there is a rising edge on *bEdgeNewValue*.
If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *iNumberOfValues* > 0 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* is incremented. If *iValueNumber* = 6, then *iValueOut* = *iValue6*. If *iValueNumber* = *iNumberOfValues* and there

is a rising edge on *bEdgeNewValue*, then *iValueNumber* = 1 and hence *iValueOut* = iValue1.
If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *iNumberOfValues* = **12** and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* is decremented. If *iValueNumber* = 5, then *iValueOut* = *iValue5*. If *iValueNumber* = 1 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues*(**12)** and hence *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 from 1 to *iNumberOfValues* in ascending form and thus determines the output value *iValueOut* (see Application [▶ 389]).
bDirection = TRUE means that *iValueNumber* decrements from *iNumberOfValues* to 1 in descending form and thus determines the output value *iValueOut* (see Application [▶ 389]).
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 this Enum.
If the operation mode *eCtrlMode* = *eHVACCtrlMode_Auto,* then the variable *iManualValue* is not taken into account (see E_HVACCtrlMode [▶ 490]).
If the operation mode *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 contents of the variable *iValue15* are then output via *iValueOut* (see Application [▶ 389]).
iManualValue is not taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual.* The value of *iValueNumber* is determined by *iManualValue*. *iManualValue* should lie within the range of 0 to 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 *iValue1-16* 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 on *bEdgeNewValue*, then *iValueNumber* is incremented. If *iValueNumber* = 6, then *iValueOut* = *iValue6*. If *iValueNumber* = *iNumberOfValues* and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = 1 and hence *iValueOut* = iValue1.
If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *iNumberOfValues* = **12** and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* is decremented. If *iValueNumber* = 5, then *iValueOut* = *iValue5*. If *iValueNumber* = 1 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues*(**12)** and hence *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* is as follows: if *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *iNumberOfValues* > 0, then *iValueNumber* = 1 and *iValueOut* = *iValue1.*

- *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *iNumberOfValues* = 12, then *iNumberOfValues* = 12 and *iValueOut* = *iValue12*.

- *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 13, then *iValueNumber* = 13 and *iValueOut* = *iValue13.*

- *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 0, then *iValueNumber* = 0 and *iValueOut* = *0.*

bActive: *bActive* goes 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 *iValue***X**, 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 there is an error and that there is a wrong parameter on the variable *iNumberOfValues*. *iValueOut* and *iValueNumber* are set constantly to 0 and the Enum *eErrorCode* indicate the error code. The message *bError* does not have to be acknowledged after rectification of the error.

eErrorCode: Supplies the error number [▶ 491] when the output *bError* is set. The following error can occur in this function block:

eHVACErrorCodes_InvalidParam_iNumberOfValues: There is an incorrect value on *iNumberOfValues*. *iNumberOfValues* must not fall below 1 and must not exceed 8.

To access the enum error number in the PLC eErrorCode can assigned to a variable of data type WORD.
eHVACErrorCodes_InvalidParam_iNumberOfValues = 42

VAR_IN_OUT

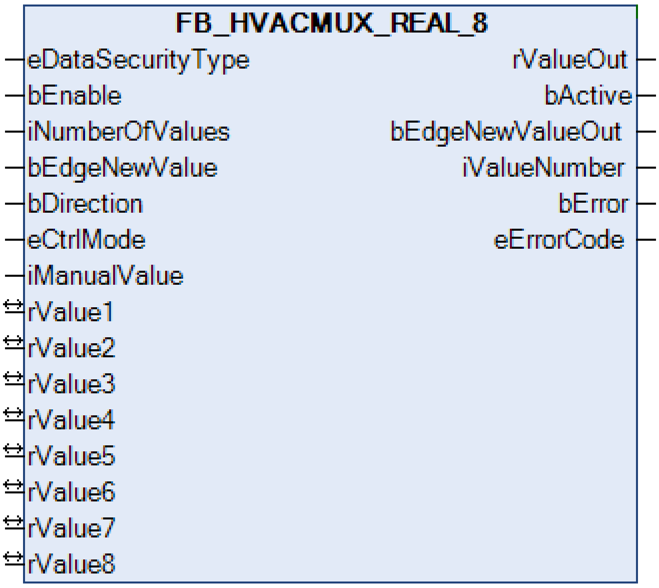
```
iValue1-16          : INT;
```

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 is, then the output *iValueNumber* varies between 1 and 8. The variables *iValue1* to *iValue8* are then taken into account for the output *iValueOut* if the operation mode *eCtrlMode* = *eHVACCtrlMode_Auto* (see Application [▶ 389]).

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.15 FB_HVACMUX_REAL_8



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 [▶ 490].
The following conditions must first be fulfilled for this:

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 *iValue***X**, 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* is incremented. If *iValueNumber* = 4, then *rValueOut* = *iValue4*. If *iValueNumber*= *iNumberOfValues***(7)** and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = 1 and hence *rValueOut* = *rValue1.*

- If *bDirection* = TRUE AND *iNumberOfValues* = **8** and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* is decremented. If *iValueNumber* = 3, then *rValueOut* = *rValue3*. If *iValueNumber*= 1 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues***(8)** and hence *rValueOut* = *rValue8.*

The start behavior of the outputs *iValueNumber* and *rValueOut* is as follows:

- If *bDirection* = FALSE AND *iNumberOfValues* > 0, then *iValueNumber* = 1 and *rValueOut* = *rValue1.*

- If *bDirection* = TRUE AND *iNumberOfValues* = 6, then *iNumberOfValues* = 6 and *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 rValue1-n if eDataSecurityType = eHVACDataSecurityType_Persistent. This would lead to premature wear of the storage medium of the controller. If the VAR_IN_OUT variables rValue1-n change frequently and are not to be stored persistently, then eDataSecurityType must be eDataSecurityType_Idle.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
iNumberOfValues   : INT;
bEdgeNewValue     : BOOL;
bDirection        : BOOL;
eCtrlMode         : E_HVACCtrlMode;
iManualValue      : INT;
```

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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* and *iValueNumber* are set constantly to 0.

iNumberOfValues: Indicates the number of variables *rValue1-8* that can be output via *rValueOut*. If *iNumberOfValues* = 8 is, then the output *iValueNumber* varies between 1 and 8. The variables *rValue1* to *rValue8* are then taken into account for the output *rValueOut* (see Application [▸ 393]).
iNumberOfValues determines the start behavior of the outputs *rValueOut* and *iValueNumber*, (see Application [▸ 393]).
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: *iValueNumber* is incremented or decremented if there is a rising edge on *bEdgeNewValue*.
If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *iNumberOfValues* > 0 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* is incremented. If *iValueNumber* = 6, then *rValueOut* = *rValue6*. If *iValueNumber*= *iNumberOfValues* and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = 1 and hence *rValueOut* = *rValue1*.
If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *iNumberOfValues* = **6** and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* is decremented. If *iValueNumber* = 5, then *rValueOut* = *rValue5*. If *iValueNumber*= 1 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues*(**6**) and hence *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 from 1 to *iNumberOfValues* in ascending form and thus determines the output value *rValueOut* (see Application [▸ 393]).

bDirection = TRUE means that *iValueNumber* decrements from *iNumberOfValues* to 1 in descending form and thus determines the output value *rValueOut* (see Application [▶ 393]).
bDirection is not taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual.*

eCtrlMode: This enum specifies the operation mode of the function block (see E_HVACCtrlMode [▶ 490]).
If the operation mode *eCtrlMode* = *eHVACCtrlMode_Auto*, then the variable *iManualValue* is not taken into account.
If the operation mode *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 contents of the variable *rValue15* are then output via *rValueOut* (see Application [▶ 393]).
iManualValue is not taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual.* The value of *iValueNumber* is determined by *iManualValue*. *iManualValue* should lie within the range of 0 to 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 on *bEdgeNewValue*, then *iValueNumber* is incremented. If *iValueNumber* = 6, then *rValueOut* = *rValue6*. If *iValueNumber*= *iNumberOfValues* and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = 1 and hence *rValueOut* = *rValue1*.
If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *iNumberOfValues* = **7** and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* is decremented. If *iValueNumber* = 5, then *rValueOut* = *rValue5*. If *iValueNumber*= 1 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues***(7)** and hence *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* is as follows: if *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *iNumberOfValues* > 0, then *iValueNumber* = 1 and *rValueOut* = *rValue1.*

- *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *iNumberOfValues* = 7, then *iNumberOfValues* = 7 and *rValueOut* = *rValue7.*

- *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 13, then *iValueNumber* = 13 and *rValueOut* = *rValue13.*

- *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 0, then *iValueNumber* = 0 and *rValueOut = 0.*

bActive: *bActive* goes 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 *rValue***X**, 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 there is an error and that there is a wrong parameter on the variable *iNumberOfValues*. *rValueOut* and *iValueNumber* are set constantly to 0 and the Enum *eErrorCode* indicate the error code. The message *bError* does not have to be acknowledged after rectification of the error.

eErrorCode: Supplies the error number [▶ 491] when the output *bError* is set. The following error can occur in this function block:

eHVACErrorCodes_InvalidParam_iNumberOfValues: There is an incorrect value on *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

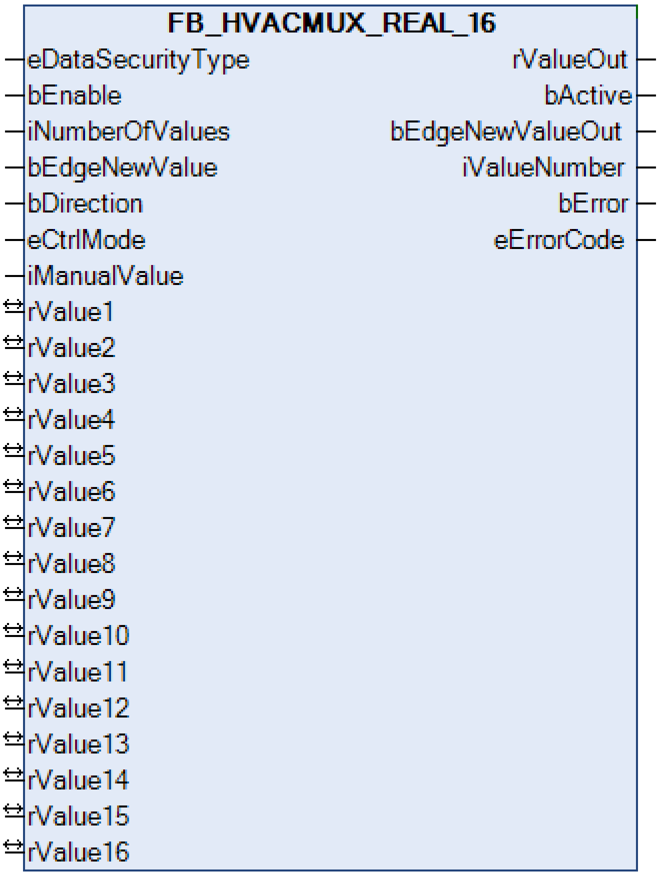
```
rValue1-8          : INT;
```

rValue1-8: The value of the output variable *rValueOut* is determined by the variables *rValue1* to *rValue8*. If *iValueNumber* = 6, then *rValueOut* = *iValue6*.
If *iNumberOfValues* = 8 is, then the output *iValueNumber* varies between 1 and 8. The variables *rValue1* to *rValue8* are then taken into account for the output *rValueOut* if the operation mode *eCtrlMode* = *eHVACCtrlMode_Auto* (see Application [▶ 393]). The variable is saved persistently.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.16 FB_HVACMUX_REAL_16



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 [▶ 490].
The following conditions must first be fulfilled for this:

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 *iValue***X**, 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 *rValueOut* = *rValue6*. If *iValueNumber*= *iNumberOfValues***(12)** and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = 1 and hence *rValueOut* = *rValue1.*

- If *bDirection* = TRUE AND *iNumberOfValues* = **8** and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* decrements. If *rValueNumber* = 5, then *rValueOut* = *rValue5*. If *iValueNumber*= 1 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues***(8)** and hence *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 rValue1-n if eDataSecurityType = eHVACDataSecurityType_Persistent. This would lead to premature wear of the storage medium of the controller. If the VAR_IN_OUT variables rValue1-n change frequently and are not to be stored persistently, then eDataSecurityType must be eDataSecurityType_Idle.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
iNumberOfValues   : INT;
bEdgeNewValue     : BOOL;
bDirection        : BOOL;
eCtrlMode         : E_HVACCtrlMode;
iManualValue      : INT;
```

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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* and *iValueNumber* are set constantly to 0.

iNumberOfValues: Indicates the number of variables *rValue1-16* that can be output via *rValueOut*. If *iNumberOfValues* = 8 is, then the output *iValueNumber* varies between 1 and 8. The variables *rValue1* to *rValue8* are then taken into account for the output *rValueOut* (see Application [▸ 397]).
iNumberOfValues determines the start behavior of the outputs *rValueOut* and *iValueNumber*, (see Application [▸ 397]).
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: *iValueNumber* is incremented or decremented if there is a rising edge on *bEdgeNewValue*.
If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *iNumberOfValues* > 0 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* is

incremented. If *iValueNumber* = 6, then *rValueOut* = *rValue6*. If *iValueNumber*= *iNumberOfValues* and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = 1 and hence *rValueOut* = *rValue1*.
If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *iNumberOfValues* = **12** and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* is decremented. If *iValueNumber* = 5, then *rValueOut* = *rValue5*. If *iValueNumber*= 1 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues***(12)** and hence *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 from 1 to *iNumberOfValues* in ascending form and thus determines the output value *rValueOut* (see Application [▶ 397]).
bDirection = TRUE means that *iValueNumber* decrements from *iNumberOfValues* to 1 in descending form and thus determines the output value *rValueOut* (see Application [▶ 397]).
bDirection is not taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual.*

eCtrlMode: This enum specifies the operation mode of the function block (see E_HVACCtrlMode [▶ 490]).
If the operation mode *eCtrlMode* = *eHVACCtrlMode_Auto*, then the variable *iManualValue* is not taken into account.
If the operation mode *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 contents of the variable *rValue15* are then output via *rValueOut* (see Application [▶ 397]).
iManualValue is not taken into account in the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual.* The value of *iValueNumber* is determined by *iManualValue*. *iManualValue* should lie within the range of 0 to 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 on *bEdgeNewValue*, then *iValueNumber* is incremented. If *iValueNumber* = 6, then *rValueOut* = *rValue6*. If *iValueNumber*= *iNumberOfValues* and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = 1 and hence *rValueOut* = *rValue1*.
If *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *iNumberOfValues* = **12** and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* is decremented. If *iValueNumber* = 5, then *rValueOut* = *rValue5*. If *iValueNumber*= 1 and there is a rising edge on *bEdgeNewValue*, then *iValueNumber* = *iNumberOfValues***(12)** and hence *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* is as follows: if *bEnable* = TRUE AND *bError* = FALSE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *iNumberOfValues* > 0, then *iValueNumber* = 1 and *rValueOut* = *rValue1.*

- *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *iNumberOfValues* = 12, then *iNumberOfValues* = 12 and *rValueOut* = *rValue12*.

- *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 13, then *iValueNumber* = 13 and *rValueOut* = *rValue13.*

- *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 0, then *iValueNumber* = 0 and *rValueOut* = *0.*

bActive: *bActive* goes 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 *rValue***X**, 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 there is an error and that there is a wrong parameter on the variable *iNumberOfValues*. *rValueOut* and *iValueNumber* are set constantly to 0 and the Enum *eErrorCode* indicate the error code. The message *bError* does not have to be acknowledged after rectification of the error.

eErrorCode: Supplies the error number [▸ 491] when the output *bError* is set. The following error can occur in this function block:

eHVACErrorCodes_InvalidParam_iNumberOfValues: There is an incorrect value on *iNumberOfValues*. *iNumberOfValues* must not fall below 1 and must not exceed 16.

To access the enum error number in the PLC eErrorCode can assigned to a variable of data type WORD.
eHVACErrorCodes_InvalidParam_iNumberOfValues = 42

VAR_IN_OUT

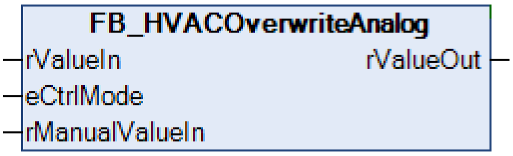
```
rValue1-16          : REAL;
```

rValue1-16: The value of the output variable *rValueOut* is determined by the variables *rValue1* to *rValue16*.
If *iValueNumber* = 6, then *rValueOut* = *iValue6*.
If *iNumberOfValues* = 8 is, then the output *iValueNumber* varies between 1 and 8. The variables *rValue1* to *rValue8* are then taken into account for the output *rValueOut* if the operation mode *eCtrlMode* = *eHVACCtrlMode_Auto* (see Application [▸ 397]). The variable is saved persistently.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.17 FB_HVACOverwriteAnalog



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 that is forwarded to the output *rValueOut* if *eCtlrMode* = *eHVACCtrlMode_Auto.*

eCtrlMode: The operation mode is selected via this enum (see E_HVACCtrlMode [▸ 490]). Manual or automatic operation mode.

rManualValueIn: Analog manual input value that is forwarded to the output *rValueOut* if *eCtrlMode* = *eHVACCtrlMode_Manual*.

VAR_OUTPUT

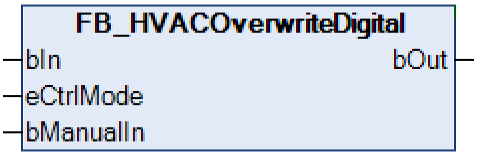
```
rValueOut             :REAL;
```

rValueOut: Analog output value.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.18 FB_HVACOverwriteDigital



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 the output *bOut* if *eCtlrMode* = *eHVACCtrlMode_Auto.*

eCtrlMode: The operation mode is selected via this enum (see E_HVACCtrlMode [▶ 490]). Manual or automatic operation mode.

bManualIn: Digital input variable whose state is forwarded to the output *bOut* if *eCtlrMode* = *eHVACCtrlMode_Manual*.

VAR_OUTPUT

```
bOut            :BOOL;
```

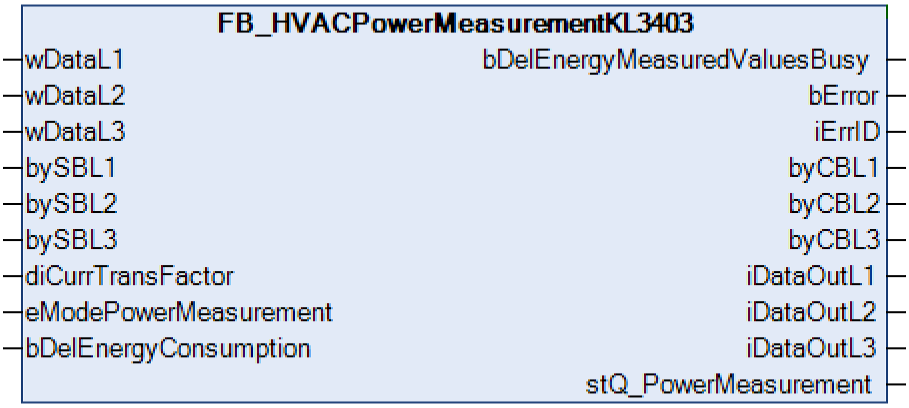
bOut: Digital output variable.

.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.19 FB_HVACPowerMeasurementKL3403



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 φ 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

```
STRUCT
  diIL1, diIL2, diIL3                : DINT;   A      Ieff  Auflösung: 0,1 A
  diUL1, diUL2, diUL3                : DINT;   V      Ueff  Auflösung: 0,1 V
  diPL1, diPL2, diPL3                : DINT;   kW     Pw  pro Phase Auflösung: 0,1 kW
  diPg                               : DINT;   kW     Pges Auflösung: 0,1 kW
  diCosPhiL1, diCosPhiL2, diCosPhiL3 : DINT;   cosPhi pro Phase Auflösung: 0,01
  diCosPhi                           : DINT;   cosPhiges Auflösung: 0,01
  diWL1, diWL2, diWL3                : DINT;   kWh    Energieverbrauch Auflösung: 1 kWh
  diWg                               : DINT;   kWh    Energieverbrauch Auflösung: 1 kWh
  diImaxL1, diImaxL2, diImaxL3       : DINT;   A      Imax Auflösung: 0,1 A
  diUmaxL1, diUmaxL2, diUmaxL3       : DINT;   V      Umax Auflösung: 0,1 V
  diPmaxL1, diPmaxL2, diPmaxL3       : DINT;   kW     Pmax Auflösung: 0,1 kW
  diSg                               : DINT;   kVA    Sges Auflösung: 0,1 kVA
  diQg                               : DINT;   kvar   Qges Auflösung: 0,1 kvar
  dummy                              : DINT;   Reserve, füllt die Struktur auf 120Bytes auf
END_STRUCT
```

VAR_INPUT

```
wDataL1, wDataL2, wDataL3   : WORD;
bySBL1, bySBL2, bySBL3      : BYTE;
diCurrTransFactor           : DINT;
eModePowerMeasurement       : E_HVACPowerMeasurementMode;
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 final measured 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 with KL3403-0000: A 400/1A transformer produces 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 = 400A.*

Example with KL3403-0010: A 400/5A transformer produces 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 = 400A.*

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 (see E_HVACPowerMeasurementMode [▸ 492]).

```
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 := 8[1])
);
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 have to be deleted from the internal EEPROM, no value is refreshed during this time and the variable *bDelEnergyMeasuredValuesBusy* indicates TRUE.

bError: If TRUE, an error has occurred in the register communication.

iErrID: Error ID in the register communication

0x100 Timeout error. The time permitted for execution has been exceeded.
0x200 Parameter error (e.g. an invalid register number).
0x300 The read value differs from the written value (write access to this register may not be enabled or have failed)

byCBLx: This output serves the selection of the desired input value and register communication in order to delete the energy consumption.

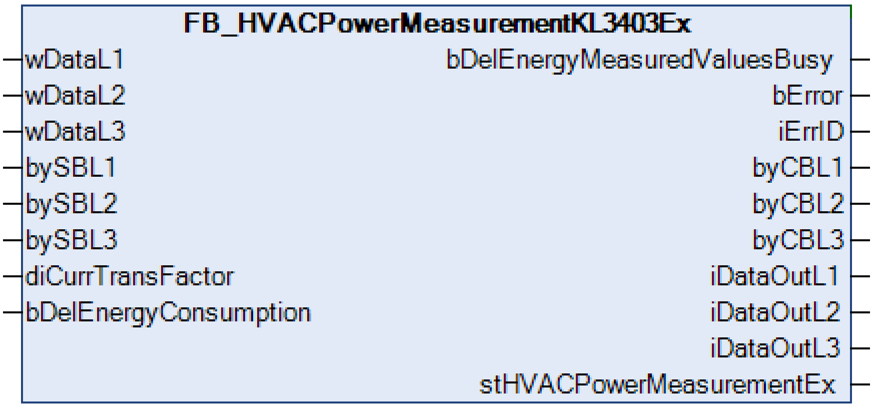
iDataOutLx: This output serves the register communication in order to delete the energy consumption.

stQ_PowerMeasurement: The results of the power measurement as an output data structure (see ST_HVACPowerMeasurement [▸ 497]).

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.20 FB_HVACPowerMeasurementKL3403Ex



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 φ 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

```
STRUCT
  fIL1, fIL2, fIL3                         : LREAL;   A         Ieff Strom (Effektivwert)
  fIg                                      : LREAL;   A         Gesamt Strom (Effektivwert)
  fUL1, fUL2, fUL3                         : LREAL;   V         Ueff Spannung (Effektivwert)
  fPL1, fPL2, fPL3                         : LREAL;   kW        P Wirkleistung Phase
  fPg                                      : LREAL;   kW        Pges Wirkleistung
  fCosPhiL1, fCosPhiL2, fCosPhiL3          : LREAL;   cosPhi    Leistungsfaktor pro Phase
  fCosPhi                                  : LREAL;   cosPhiges Leistungsfaktor
  fWL1, fWL2, fWL3                         : LREAL;   kWh       Energieverbrauch
  fWg                                      : LREAL;   kWh       Energieverbrauch
  fImaxL1, fImaxL2, fImaxL3                : LREAL;   A         Imax Spitzenwert des Stroms
  fUmaxL1, fUmaxL2, fUmaxL3                : LREAL;   V         Umax Spitzenwert der Spannung
  fPmaxL1, fPmaxL2, fPmaxL3                : LREAL;   kW        Pmax Spitzenwert der Wirkleistung

  fSg                                      : LREAL;   kVA       Sges Scheinleistung
  fQg                                      : LREAL;   kvar      Qges Blindleistung
  fFrequencyL1, fFrequencyL2, fFrequencyL3 : 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 final measured 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 with KL3403-0000: A 400/1A transformer produces 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 = 400A.*

Example with KL3403-0010: A 400/5A transformer produces 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 = 400A.*

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 time permitted for execution has been exceeded.
0x200 Parameter error (e.g. an invalid register number).
0x300 The read value differs from the written value (write access to this register may not be enabled or have failed)

byCBLx: This output serves the selection of the desired input value and register communication in order to delete the energy consumption.

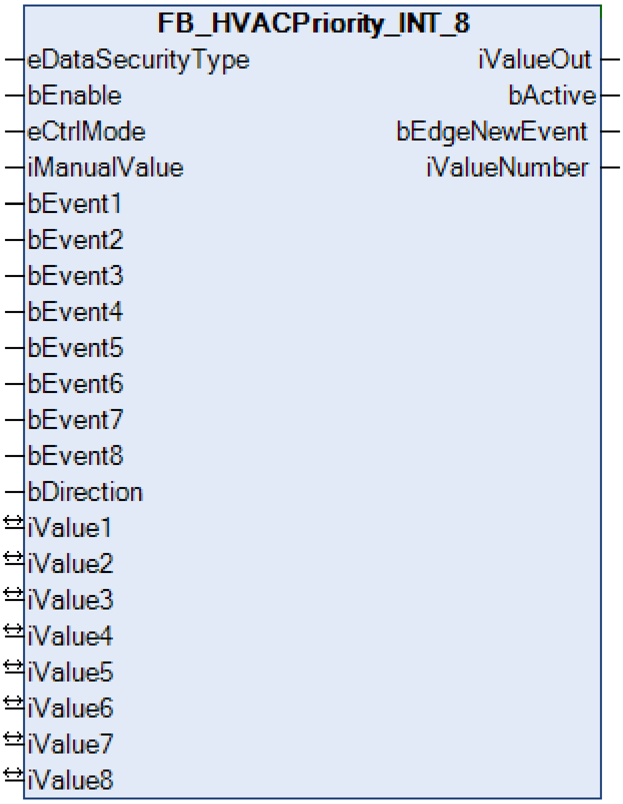
iDataOutLx: This output serves the register communication in order to delete the energy consumption.

stQ_PowerMeasurementEx: The results of the power measurement as an output data structure (see ST_HVACPowerMeasurementEx [▶ 498]).

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.21 FB_HVACPriority_INT_8



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 via the events occurring at the inputs *bEvent1-8*, *iValue1-8* and the control direction of the prioritization *bDirection*.
bDirection =FALSE means that the events have a higher priority in ascending form from 1 to 8. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent7* = TRUE (highest occurring event in Table 1 [▶ 406] column 1), then *iValueOut* has the value of *iValue7*.
bDirection = TRUE means that the events have a higher priority in descending form from 8 to 1. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in Table 2 [▶ 406] column 1), then *iValueOut* has the value of *iValue1*.

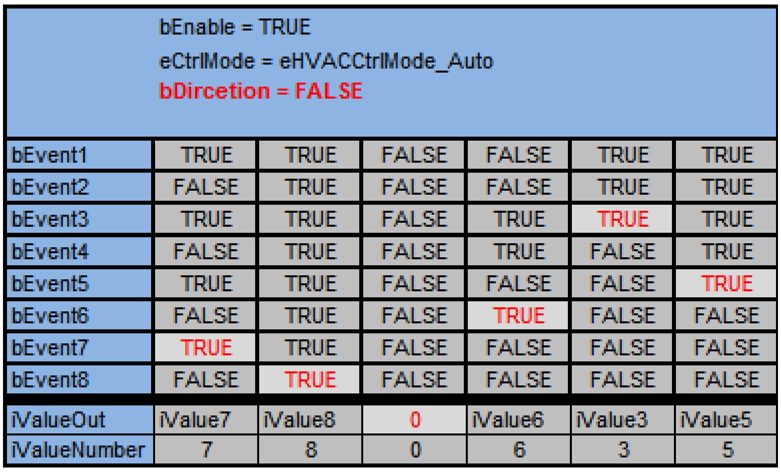
The function block can be used as a multiplexer if *eCtrlMode* = *eHVACCtrlMode_Manual*. The value of *iManualValue* refers to one of the VAR_IN_OUT variables *iValue1-8*, whose value is output via *iValueOut*. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 2, then *iValueOut* = *iValue2,* see Table 3 [▶ 407].

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 form from 1 to 8

It can be seen in Table 1 that *bDirection* = FALSE. This means that the events determine the output value *iValueOut* in ascending form 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*.

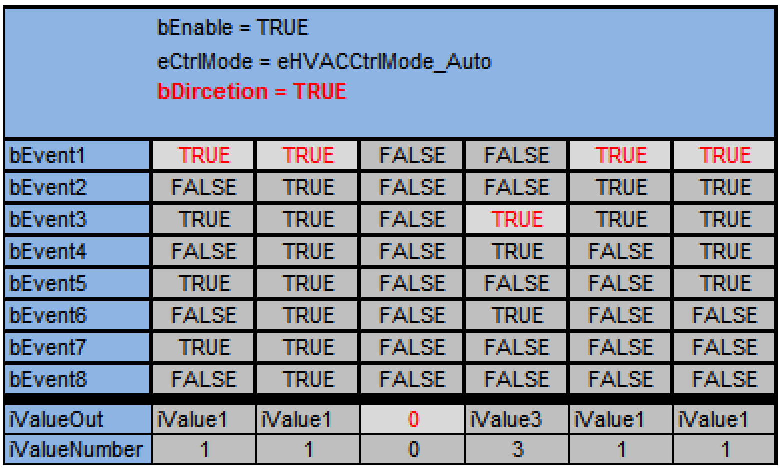


bEnable = TRUE
eCtrlMode = eHVACCtrlMode_Auto
bDircetion = FALSE

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 form from 8 to 1

It can be seen in Table 2 that *bDirection* = TRUE . This means that the events determine the output value *iValueOut* in descending form from 8 to 1. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in Table 2 [▶ 407] column 1), then *iValueOut* = *iValue1*.

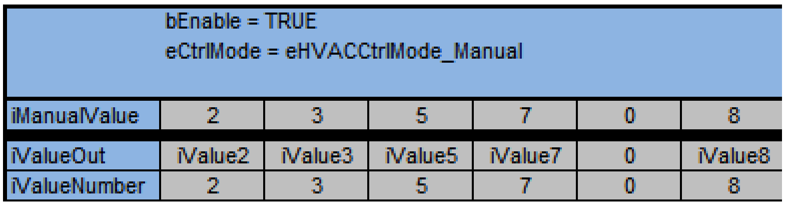


bEnable = TRUE
eCtrlMode = eHVACCtrlMode_Auto
bDircetion = TRUE

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

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

VAR_INPUT

```
eDataSecurityType  : E_HVACDataSecurityType;
bEnable            : BOOL;
bEvent1 - bEvent8  : BOOL;
eCtrlMode          : E_HVACCtrlMode;
iManualValue       : INT;           0..8
bDirection         : BOOL;
```

eDataSecurityType: If eDataSecurityType [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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: This enum specifies the operation mode of the function block (see E_HVACCtrlMode [▶ 490]).
If *eCtrlMode* = *eHVACCtrlMode_Auto*, then the function block represents a prioritization of events. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto*, then the output *iValueOut* is controlled via the events occurring at the inputs *bEvent1-8, iValue1-8* and the control direction of the prioritization *bDirection*, see Table 1 [▶ 406] and Table 2 [▶ 407]
If *eCtrlMode* = *eHVACCtrlMode_Manual*, then the function block represents a multiplexer. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual,* then the value of the output *iValueOut* is controlled via *iManualValue*. If *iManualProfile* = 5, then *iValueOut* = *iValue5*, see Table 3 [▶ 407]

iManualProfile: If the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual* (Multiplexer) AND *bEnable* = TRUE AND *bError* = FALSE*,* then the value of the output *iValueOut* is controlled via *iManualValue*. If *iManualProfile* = 5, then *iValueOut* = *iValue5*, see Table 3 [▶ 407]

bEvent1-8: The output *iValueOut* is controlled via the events occurring at the inputs *bEvent1-8*, *iValue1-8* and the control direction of the prioritization *bDirection*.
bDirection =FALSE means that the events determine the output value *iValueOut* in ascending form from 1 to 8. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent7* = TRUE (highest occurring event in Table 1 [▶ 406] column 1), then *iValueOut* = *iValue7*.
bDirection = *TRUE* means that the events determine the output value iValueOut in descending form from 8 to 1. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in Table 2 [▶ 406] column 1), then *iValueOut* = *iValue1*.
The variables *bEvent1-8* and *bDirection* are taken into account only if *eCtrlMode* = *eHVACCtrlMode_Auto*.

bDirection: The function block can be used to prioritize events if *eCtrlMode* = *eHVACCtrlMode_Auto*. The output *iValueOut* is controlled via the events occurring at the inputs *bEvent1-8*, *iValue1-8* and the control direction of the prioritization *bDirection*.
bDirection =FALSE means that the events determine the output value *iValueOut* in ascending form from 1 to 8. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent7* = TRUE (highest occurring event in Table 1 [▶ 406] column 1), then *iValueOut* = *iValue7*.
bDirection = TRUE means that the events determine the output value *iValueOut* in descending form from 8 to 1. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in Table 2 [▶ 406] column 1), then *iValueOut* = *iValue1*.
The variables *bEvent1-8* and *bDirection* are taken into account only 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 via the events occurring at the inputs *bEvent1-8*, *iValue1-8* and the control direction of the prioritization *bDirection*.

bDirection =FALSE means that the events determine the output value *iValueOut* in ascending form from 1 to 8. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent7* = TRUE (highest occurring event in Table 1 [▸ 406] column 1), then *iValueOut* = *iValue7*.
bDirection = TRUE means that the events determine the output value *iValueOut* in descending form from 8 to 1. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in Table 2 [▸ 406] 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 [▸ 407].

bActive: *bActive* goes TRUE if

1. *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* and one of the input variables *bEvent1-8* is 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)* for controlling the output *iValueOut* by means of *iValue1-8* has changed.

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 [▸ 406], Table 2 [▸ 407] and Table 3 [▸ 407]

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, then 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, ...* and *iValue8* to the event *bEvent8*

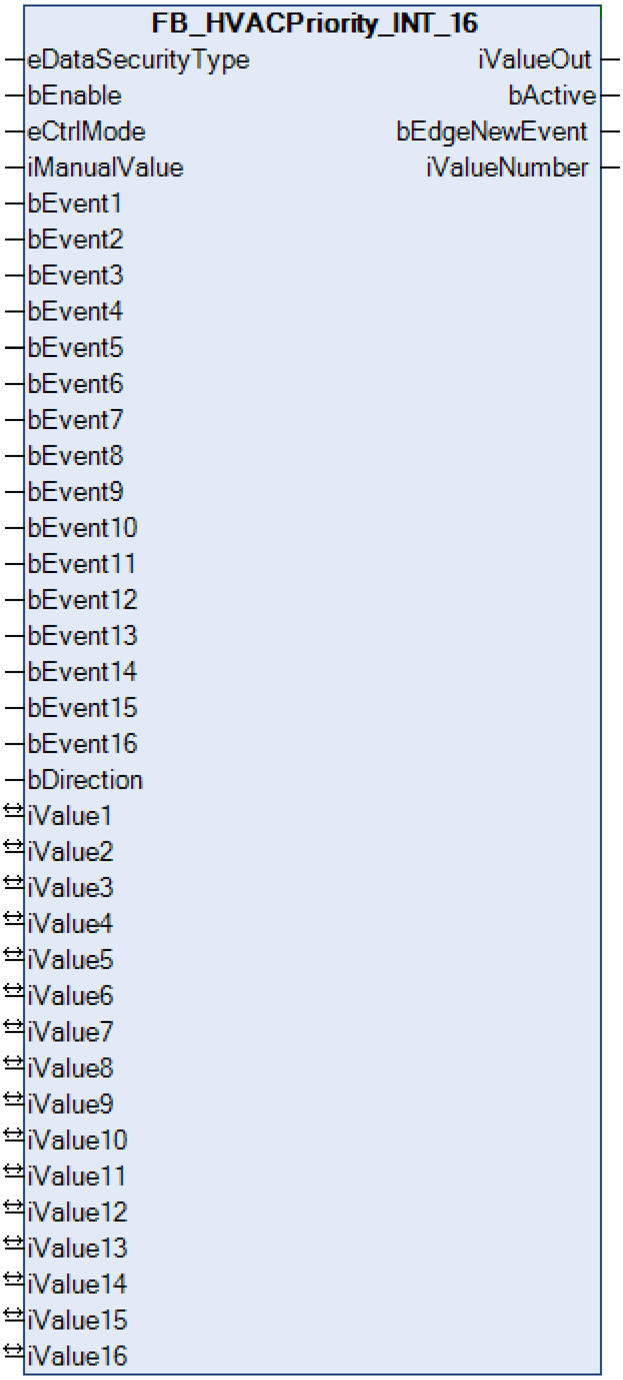
If the function block is used as a multiplexer, then each of the variables *iValue1-8* is assigned to the value of *iManualValue*. If *iManualValue* = 1, then *iValueOut* = *iValue1*. If *iManualValue* = 2, then *iValueOut* = *iValue2….* if *iManualValue* = 8, then *iValueOut* = *iValue8*.

The variable is saved persistently.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.22 FB_HVACPriority_INT_16



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 via the events occurring at the inputs *bEvent1-16*, *iValue1-16* and the control direction of the prioritization *bDirection*.
bDirection =FALSE means that the events have a higher priority in ascending form from 1 to 16. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent15* = TRUE (highest occurring event in Table 1 [▶ 411] column 1), then *iValueOut* has the value of *iValue15*.
bDirection = TRUE means that the events have a higher priority in descending form from 16 to 1. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in Table 2 [▶ 411] column 1), then *iValueOut* has the value of *iValue1*.

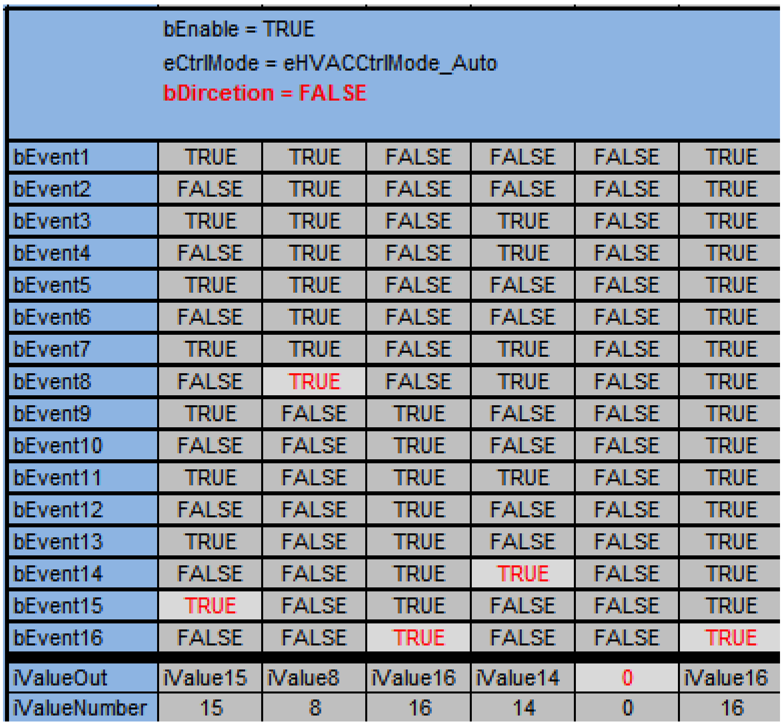
The function block can be used as a multiplexer if *eCtrlMode* = *eHVACCtrlMode_Manual*. The value of *iManualValue* refers to one of the VAR_IN_OUT variables *iValue1-16*, whose value is output via *iValueOut*. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 2, then *iValueOut* = *iValue2,* see Table 3 [▸ 412].

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 form from 1 to 16

It can be seen in Table 1 that *bDirection* = FALSE. This means that the events determine the output value *iValueOut* in ascending form 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*.

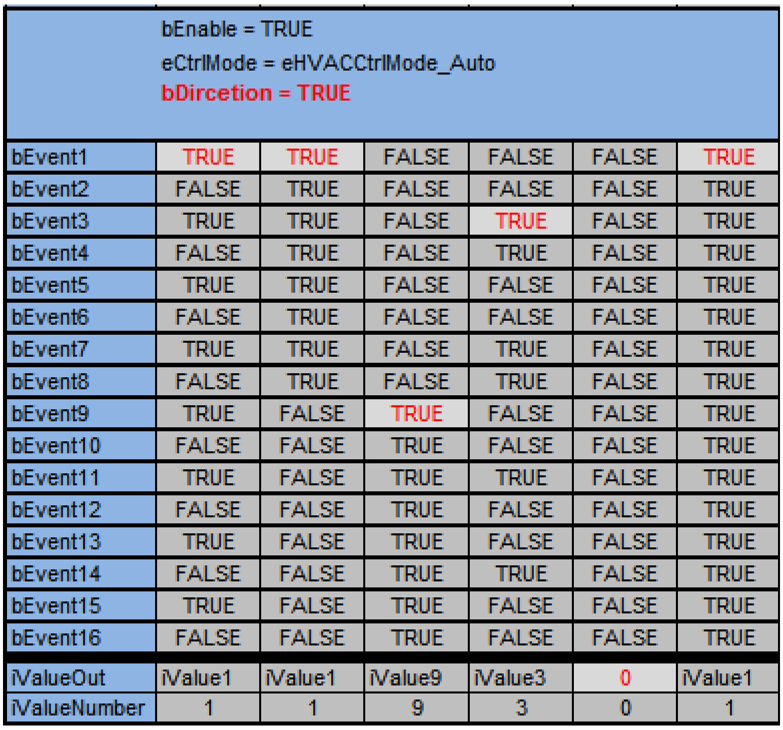


bEnable = TRUE
eCtrlMode = eHVACCtrlMode_Auto
bDircetion = FALSE

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 form from 16 to 1

It can be seen in Table 2 that *bDirection* = TRUE . This means that the events determine the output value *iValueOut* in descending form from 16 to 1. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in Table 2 [▸ 411] column 1), then *iValueOut* = *iValue1*.

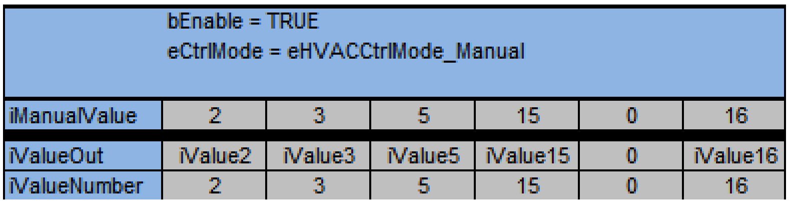


bEnable = TRUE
eCtrlMode = eHVACCtrlMode_Auto
bDircetion = TRUE

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

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

VAR_INPUT

```
eDataSecurityType  : E_HVACDataSecurityType;
bEnable            : BOOL;
bEvent1 - bEvent8  : BOOL;
eCtrlMode          : E_HVACCtrlMode;
iManualValue       : INT;           0..16
bDirection         : BOOL;
```

eDataSecurityType: If eDataSecurityType [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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: This enum specifies the operation mode of the function block (see E_HVACCtrlMode [▸ 490]). If *eCtrlMode* = *eHVACCtrlMode_Auto*, then the function block represents a prioritization of events. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto*, then the output *iValueOut* is controlled via the events occurring at the inputs *bEvent1-16, iValue1-16* and the control direction of the prioritization *bDirection*, see Table 1 [▸ 411] and Table 2 [▸ 411]
If *eCtrlMode* = *eHVACCtrlMode_Manual*, then the function block represents a multiplexer. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual,* then the value of the output *iValueOut* is controlled via *iManualValue*. If *iManualProfile* = 5, then *iValueOut* = *iValue5*, see Table 3 [▸ 412]

iManualProfile: If the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual* (Multiplexer) AND *bEnable* = TRUE AND *bError* = FALSE, then the value of the output *iValueOut* is controlled via *iManualValue*. If *iManualProfile* = 5, then *iValueOut* = *iValue5*, see Table 3 [▸ 412]

bEvent1-16: The output *iValueOut* is controlled via the events occurring at the inputs *bEvent1-16*, *iValue1-16* and the control direction of the prioritization *bDirection*.
bDirection =FALSE means that the events determine the output value *iValueOut* in ascending form from 1 to 16. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent15* = TRUE (highest occurring event in Table 1 [▸ 411] column 1), then *iValueOut* = *iValue15*.
bDirection = TRUE means that the events determine the output value *iValueOut* in descending form from 16 to 1. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in Table 2 [▸ 411] column 1), then *iValueOut* = *iValue1*.
The variables *bEvent1-16* and *bDirection* are taken into account only if *eCtrlMode* = *eHVACCtrlMode_Auto*.

bDirection: The function block can be used to prioritize events if *eCtrlMode* = *eHVACCtrlMode_Auto*. The output *iValueOut* is controlled via the events occurring at the inputs *bEvent1-16*, *iValue1-16* and the control direction of the prioritization *bDirection*.
bDirection =FALSE means that the events determine the output value *iValueOut* in ascending form from 1 to 16. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent15* = TRUE (highest occurring event in Table 1 [▸ 411] column 1), then *iValueOut* = *iValue15*.
bDirection = TRUE means that the events determine the output value *iValueOut* in descending form from 16 to 1. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in Table 2 [▸ 411] column 1), then *iValueOut* = *iValue1*.
The variables *bEvent1-16* and *bDirection* are taken into account only 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 via the events occurring at the inputs *bEvent1-16*, *iValue1-16* and the control direction of the prioritization *bDirection*.
bDirection =FALSE means that the events determine the output value *iValueOut* in ascending form from 1 to 16. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent15* = TRUE (highest occurring event in Table 1 [▸ 411] column 1), then *iValueOut* = *iValue15*.
bDirection = TRUE means that the events determine the output value *iValueOut* in descending form from 16 to 1. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = TRUE AND *bEvent1* = TRUE (lowest occurring event in Table 2 [▸ 411] 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 [▸ 412].

bActive: *bActive* goes TRUE if

1. *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* and one of the input variables *bEvent1-16* is 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)* for controlling the output *iValueOut* by means of *iValue1-16* has changed.

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 [▸ 411], Table 2 [▸ 411] and Table 3 [▸ 412]

VAR_IN_OUT

```
iValue1-16          : 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, then 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*

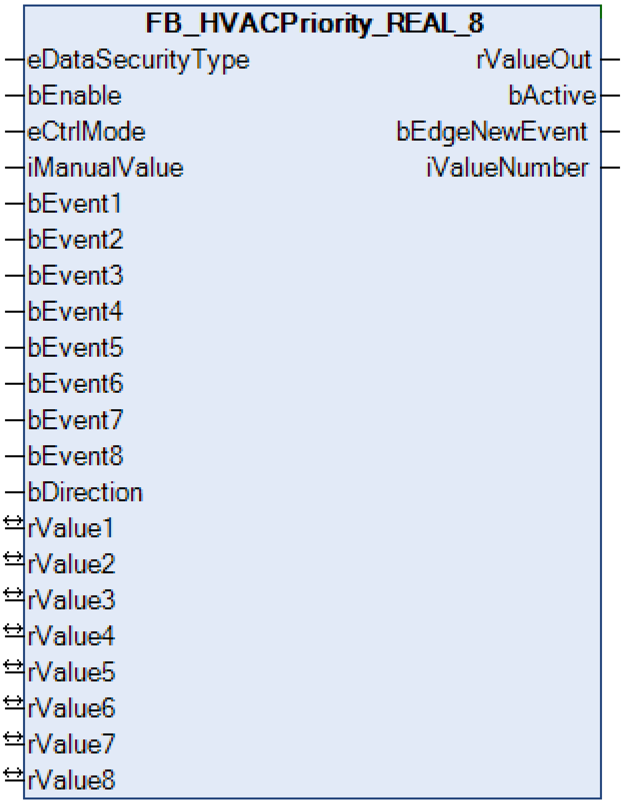
If the function block is used as a multiplexer, then each of the variables *iValue1-16* is assigned to the value of *iManualValue*. If *iManualValue* = 1, then *iValueOut* = *iValue1*. If *iManualValue* = 2, then *iValueOut* = *iValue2*…. if *iManualValue* = 16, then *iValueOut* = *iValue16*.

The variable is saved persistently.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.23 FB_HVACPriority_REAL_8



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 via the events occurring at the inputs *bEvent1-8*, *rValue1-8* and the control direction of the prioritization *bDirection*.
bDirection =FALSE means that the events have a higher priority in ascending form from 1 to 8. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent7* = TRUE (highest occurring event in Table 1 [▸ 415] column 1), then *rValueOut* has the value of *rValue7*.
bDirection = TRUE means that the events have a higher priority in descending form from 8 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 *rValueOut* has the value of *rValue1*.

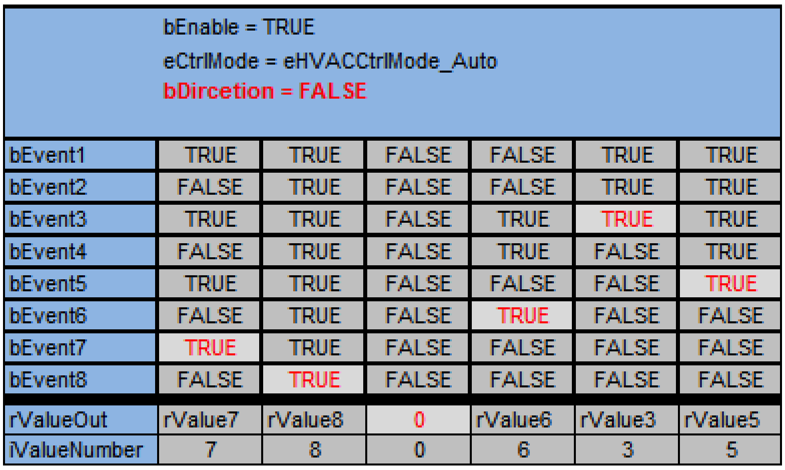
The function block can be used as a multiplexer if *eCtrlMode* = *eHVACCtrlMode_Manual*. The value of *iManualValue* refers to one of the VAR_IN_OUT variables *rValue1-8*, whose value is output via *rValueOut*. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 2, then *rValueOut* = *rValue2,* see Table 3 [▸ 416].

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 form from 1 to 8

It can be seen in Table 1 that *bDirection* = FALSE. This means that the events determine the output value *rValueOut* in ascending form 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*.

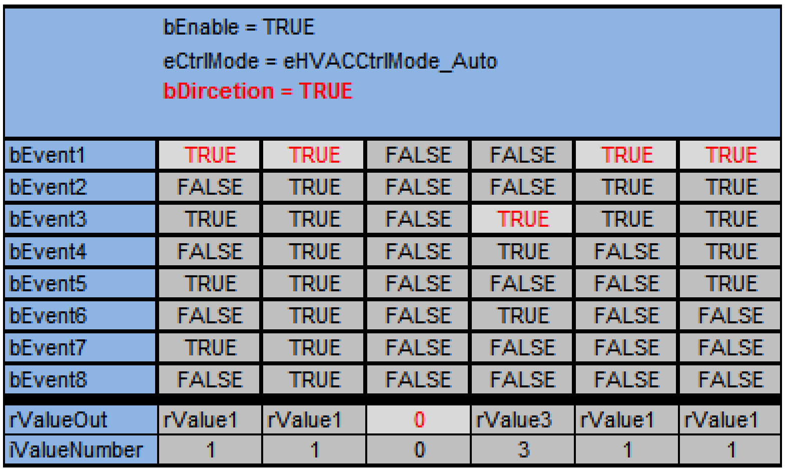


bEnable = TRUE
eCtrlMode = eHVACCtrlMode_Auto
bDircetion = FALSE

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 form from 8 to 1

It can be seen in Table 2 that *bDirection* = TRUE . This means that the events determine the output value *rValueOut* in descending form from 8 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 *rValueOut* = *rValue1*.

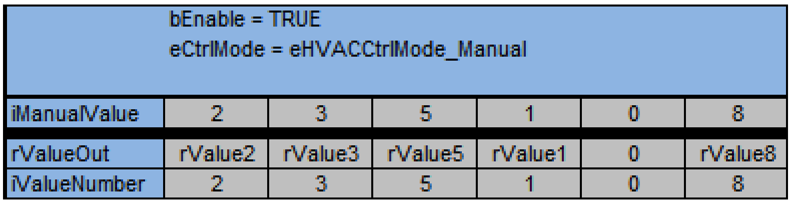


bEnable = TRUE
eCtrlMode = eHVACCtrlMode_Auto
bDircetion = TRUE

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

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

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
bEvent1 - bEvent8 : BOOL;
eCtrlMode         : E_HVACCtrlMode;
iManualValue      : INT;            0..8
bDirection        : BOOL;
```

eDataSecurityType: If eDataSecurityType [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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: This enum specifies the operation mode of the function block (see E_HVACCtrlMode [▶ 490]). If *eCtrlMode* = *eHVACCtrlMode_Auto*, then the function block represents a prioritization of events. If *bEnable* = *TRUE* AND *eCtrlMode* = *eHVACCtrlMode_Auto*, then the output *rValueOut* is controlled via the events occurring at the inputs *bEvent1-8, rValue1-8* and the control direction of the prioritization *bDirection*, see Table 1 [▶ 415] and Table 2 [▶ 415]

If *eCtrlMode* = *eHVACCtrlMode_Manual*, then the function block represents a multiplexer. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual,* then the value of the output *rValueOut* is controlled via *iManualValue*. If *iManualProfile* = 5, then *rValueOut* = *rValue5*, 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 *rValueOut* is controlled via *iManualValue*. If *iManualProfile* = 5, then *rValueOut* = *rValue5*, see Table 3 [▶ 416]

bEvent1-8: The output *rValueOut* is controlled via the events occurring at the inputs *bEvent1-8*, *rValue1-8* and the control direction of the prioritization *bDirection*.
bDirection =FALSE means that the events determine the output value *rValueOut* in ascending form from 1 to 8. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent7* = TRUE (highest occurring event in Table 1 [▶ 415] column 1), then *rValueOut* = *rValue7*.
bDirection = *TRUE* means that the events determine the output value *rValueOut* in descending form from 8 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 *rValueOut* = *rValue1*.
The variables *bEvent1-8* and *bDirection* are taken into account only if *eCtrlMode* = *eHVACCtrlMode_Auto*.

bDirection: The function block can be used to prioritize events if *eCtrlMode* = *eHVACCtrlMode_Auto*. The output *rValueOut* is controlled via the events occurring at the inputs *bEvent1-8*, *rValue1-8* and the control direction of the prioritization *bDirection*.
bDirection =FALSE means that the events determine the output value *rValueOut* in ascending form from 1 to 8. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent7* = TRUE (highest occurring event in Table 1 [▶ 415] column 1), then *rValueOut* = *rValue7*.
bDirection = *TRUE* means that the events determine the output value *rValueOut* in descending form from 8 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 *rValueOut* = *rValue1*.
The variables *bEvent1-8* and *bDirection* are taken into account only 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 via the events occurring at the inputs *bEvent1-8*, *rValue1-8* and the control direction of the prioritization *bDirection*.
bDirection =FALSE means that the events determine the output value *rValueOut* in ascending form from 1 to 8. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent7* = TRUE (highest occurring event in Table 1 [▶ 415] column 1), then *rValueOut* = *rValue7*.
bDirection = *TRUE* means that the events determine the output value *rValueOut* in descending form from 8 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 *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 [▶ 416].

bActive: *bActive* goes TRUE if

1. *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* and one of the input variables *bEvent1-8* is 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)* for controlling the output *rValueOut* by means of *rValue1-8* has changed.

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 [▶ 415], Table 2 [▶ 415] and Table 3 [▶ 416]

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, then 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, ...* and *rValue8* to the event *bEvent8*

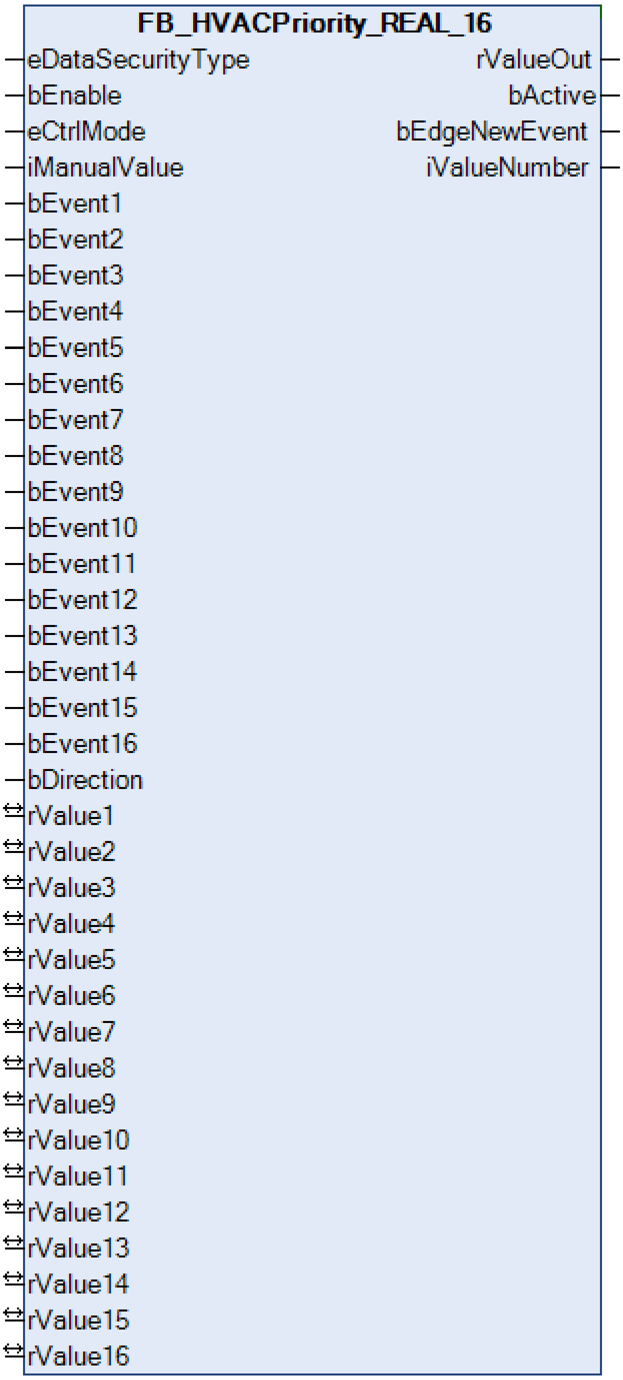
If the function block is used as a multiplexer, then each of the variables *rValue1-8* is assigned to the value of *iManualValue*. If *iManualValue* = 1, then *rValueOut* = *rValue1*. If *iManualValue* = 2, then *rValueOut* = *rValue2*…. If *iManualValue* = 8, then *rValueOut* = *rValue8*.

The variable is saved persistently.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.24 FB_HVACPriority_REAL_16



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 via the events occurring at the inputs *bEvent1-16*, *rValue1-16* and the control direction of the prioritization *bDirection*.
bDirection =FALSE means that the events have a higher priority in ascending form from 1 to 16. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent15* = TRUE (highest occurring event in Table 1 [▶ 420] column 1), then *rValueOut* has the value of *rValue15*.
bDirection = TRUE means that the events have a higher priority in descending form from 16 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 *rValueOut* has the value of *rValue1*.

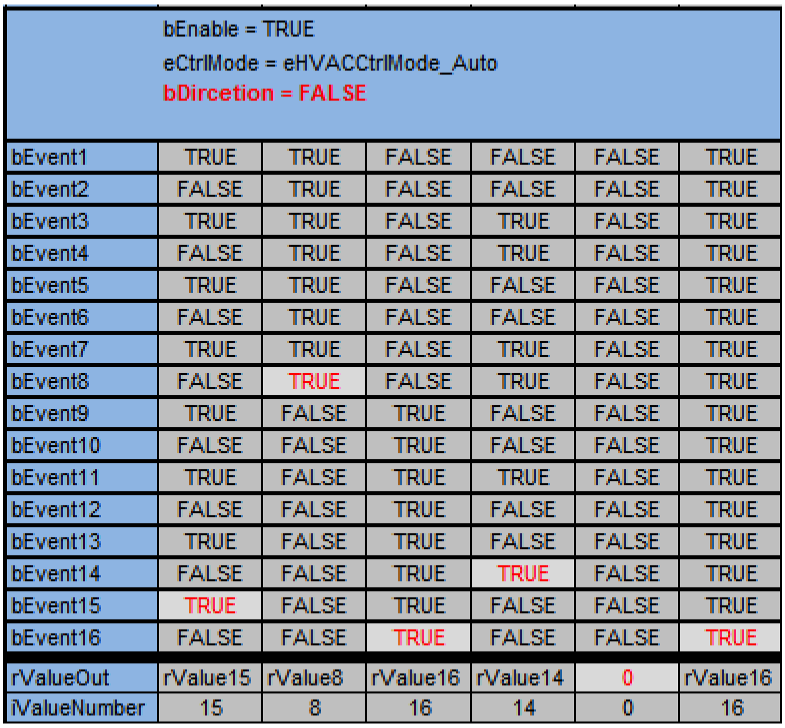
The function block can be used as a multiplexer if *eCtrlMode* = *eHVACCtrlMode_Manual*. The value of *iManualValue* refers to one of the VAR_IN_OUT variables *rValue1-16*, whose value is output via *rValueOut*. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual* AND *iManualValue* = 2, then *rValueOut* = *rValue2*, see Table 3 [▶ 421].

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 form from 1 to 16

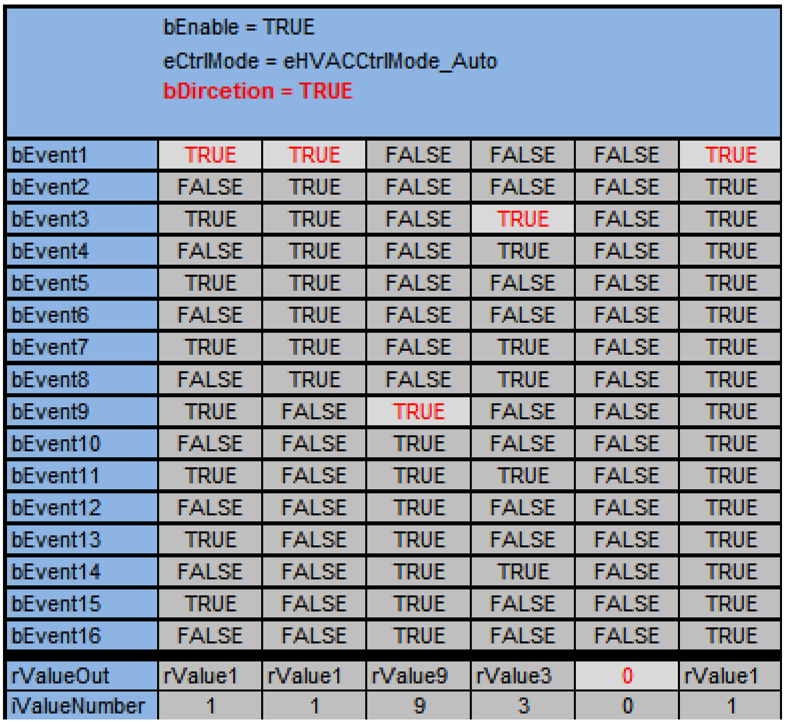
It can be seen in Table 1 that *bDirection* = FALSE. This means that the events determine the output value *rValueOut* in ascending form from 1 to 16. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent15* = TRUE (highest occurring event in column 1), then *rValueOut* = *rValue15*.



	bEnable = TRUE eCtrlMode = eHVACCtrlMode_Auto bDircetion = 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 form from 16 to 1

It can be seen in Table 2 that *bDirection* = TRUE . This means that the events determine the output value *rValueOut* in descending form from 16 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 *rValueOut* = *rValue1*.

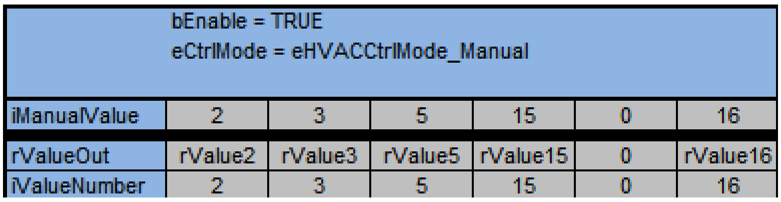


bEnable = TRUE
eCtrlMode = eHVACCtrlMode_Auto
bDircetion = TRUE

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

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

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
bEvent1 - bEvent8 : BOOL;
eCtrlMode         : E_HVACCtrlMode;
iManualValue      : INT;            0..16
bDirection        : BOOL;
```

eDataSecurityType: If eDataSecurityType [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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: This enum specifies the operation mode of the function block (see E_HVACCtrlMode [▶ 490]).
If *eCtrlMode* = *eHVACCtrlMode_Auto*, then the function block represents a prioritization of events. If *bEnable* = *TRUE* AND *eCtrlMode* = *eHVACCtrlMode_Auto*, then the output *rValueOut* is controlled via the events occurring at the inputs *bEvent1-16, rValue1-16* and the control direction of the prioritization *bDirection*, see Table 1 [▶ 420] and Table 2 [▶ 420]
If *eCtrlMode* = *eHVACCtrlMode_Manual*, then the function block represents a multiplexer. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Manual,* then the value of the output *rValueOut* is controlled via *iManualValue*. If *iManualProfile* = 5, then *rValueOut* = *rValue5*, see Table 3 [▶ 421]

iManualProfile: If the operation mode *eCtrlMode* = *eHVACCtrlMode_Manual* (Multiplexer) AND *bEnable* = TRUE AND *bError* = FALSE*,* then the value of the output *rValueOut* is controlled via *iManualValue*. If *iManualProfile* = 5, then *rValueOut* = *rValue5*, see Table 3 [▶ 421]

bEvent1-16: The output *rValueOut* is controlled via the events occurring at the inputs *bEvent1-16*, *rValue1-16* and the control direction of the prioritization *bDirection*.
bDirection =FALSE means that the events determine the output value *rValueOut* in ascending form from 1 to 16. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent15* = TRUE (highest occurring event in Table 1 [▶ 420] column 1), then *rValueOut* = *rValue15*.
bDirection = *TRUE* means that the events determine the output value *rValueOut* in descending form from 16 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 *rValueOut* = *rValue1*.
The variables *bEvent1-16* and *bDirection* are taken into account only if *eCtrlMode* = *eHVACCtrlMode_Auto*.

bDirection: The function block can be used to prioritize events if *eCtrlMode* = *eHVACCtrlMode_Auto*. The output *rValueOut* is controlled via the events occurring at the inputs *bEvent1-16*, *rValue1-16* and the control direction of the prioritization *bDirection*.
bDirection =FALSE means that the events determine the output value *rValueOut* in ascending form from 1 to 16. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent15* = TRUE (highest occurring event in Table 1 [▶ 420] column 1), then *rValueOut* = *rValue15*.
bDirection = TRUE means that the events determine the output value *rValueOut* in descending form from 16 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 *rValueOut* = *rValue1*.
The variables *bEvent1-16* and *bDirection* are taken into account only 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 via the events occurring at the inputs *bEvent1-16*, *rValue1-16* and the control direction of the prioritization *bDirection*.
bDirection =FALSE means that the events determine the output value *rValueOut* in ascending form from 1 to 16. If *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* AND *bDirection* = FALSE AND *bEvent15* = TRUE (highest occurring event in Table 1 [▶ 420] column 1), then *rValueOut* = *rValue15*.
bDirection = TRUE means that the events determine the output value *rValueOut* in descending form from 16 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 *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 [▶ 421].

bActive: *bActive* goes TRUE if

1. *bEnable* = TRUE AND *eCtrlMode* = *eHVACCtrlMode_Auto* and one of the input variables *bEvent1-16* is 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)* for controlling the output *rValueOut* by means of *rValue1-16* has changed.

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 [▶ 420], Table 2 [▶ 420] and Table 3 [▶ 421]

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, then 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, ...* and *rValue16* to the event *bEvent16*

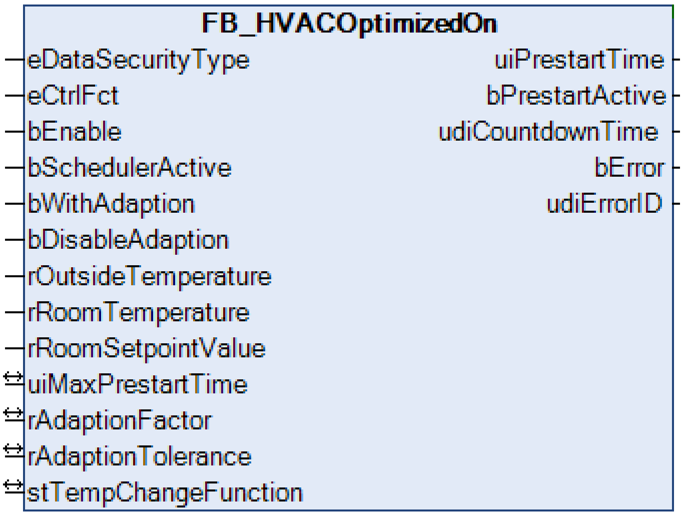
If the function block is used as a multiplexer, then each of the variables *rValue1-16* is assigned to the value of *iManualValue*. If *iManualValue* = 1, then *rValueOut* = *rValue1*. If *iManualValue* = 2, then *rValueOut* = *rValue2*…. If *iManualValue* = 16, then *rValueOut* = *rValue16*.

The variable is saved persistently.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.25 FB_HVACOptimizedOn



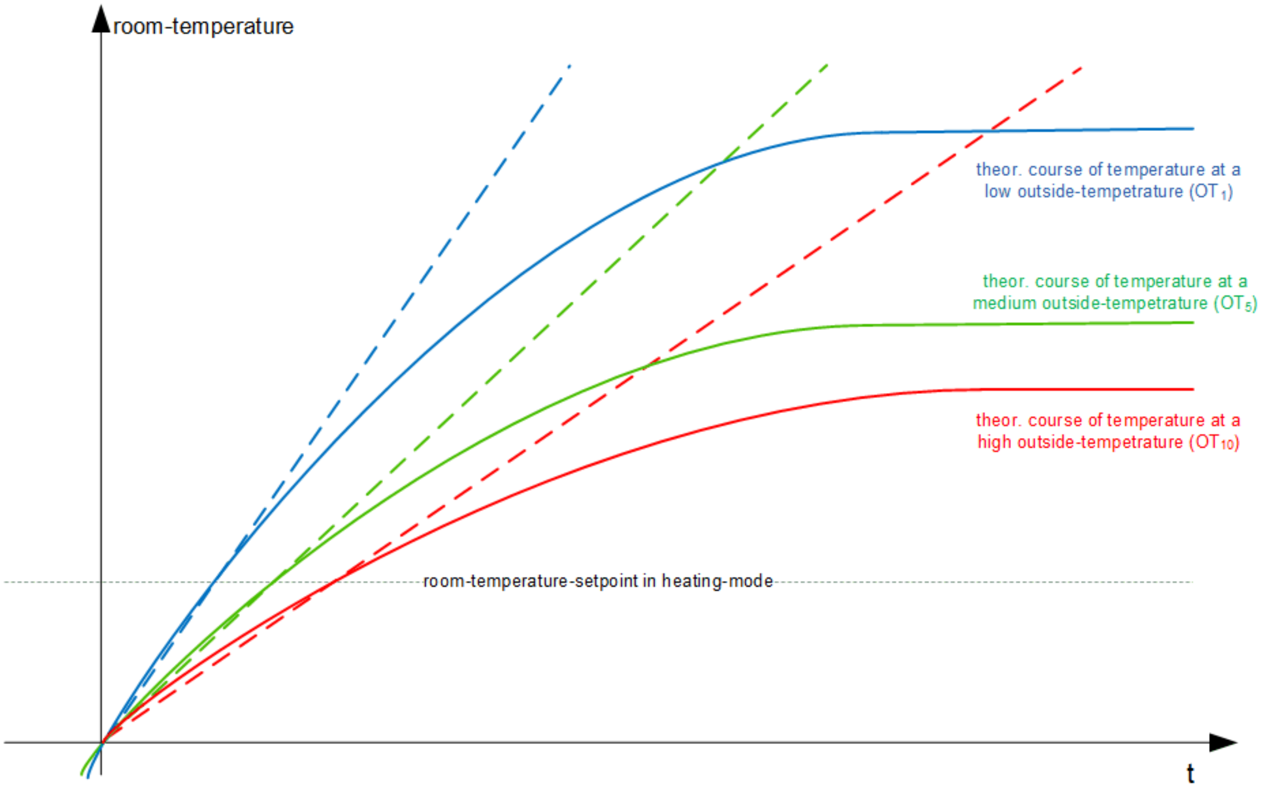
Function block for the optimized switch-on of boilers and air conditioning systems in conjunction with switching time function blocks.

The explanations in this document relate to the heating behavior. The function block is used in the same way in conjunction with cooling devices.

Calculation of the pre-start time

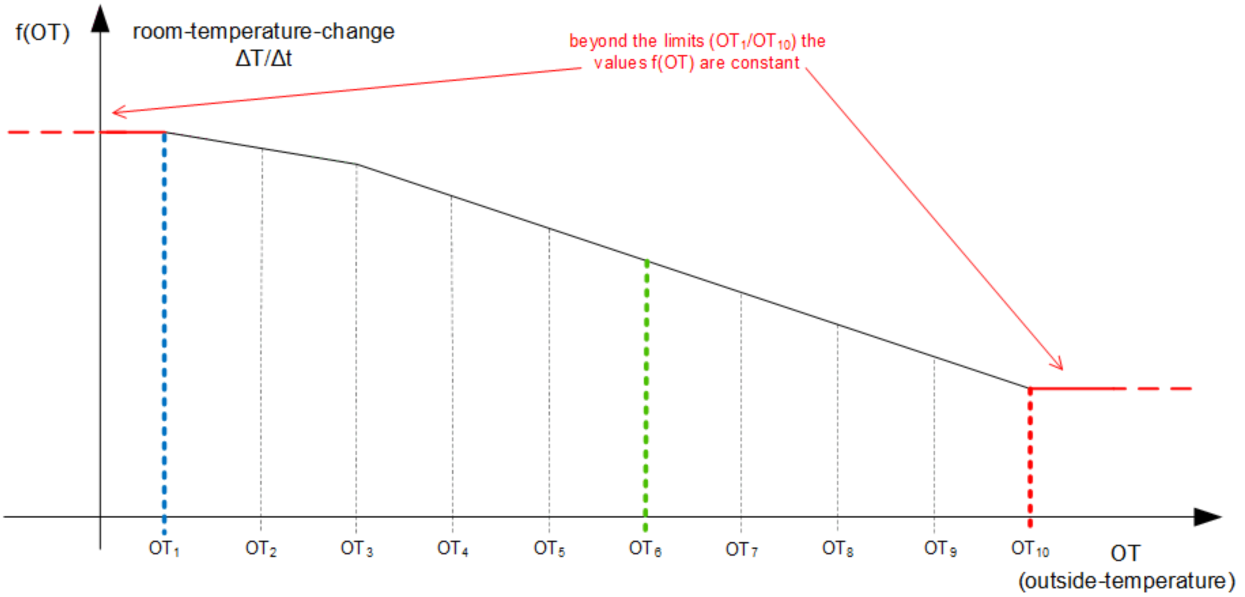
Company buildings and public facilities are generally unoccupied overnight and at weekends, so the heating boilers there often run only in standby mode. The goal of optimized switch-on is to start the boiler early enough for the building to have been heated sufficiently at a certain point in time. Conversely, 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 setpoint is linear. This results in a characteristic temperature change ΔT/Δt for each outside temperature, which is shown here by the dotted line.

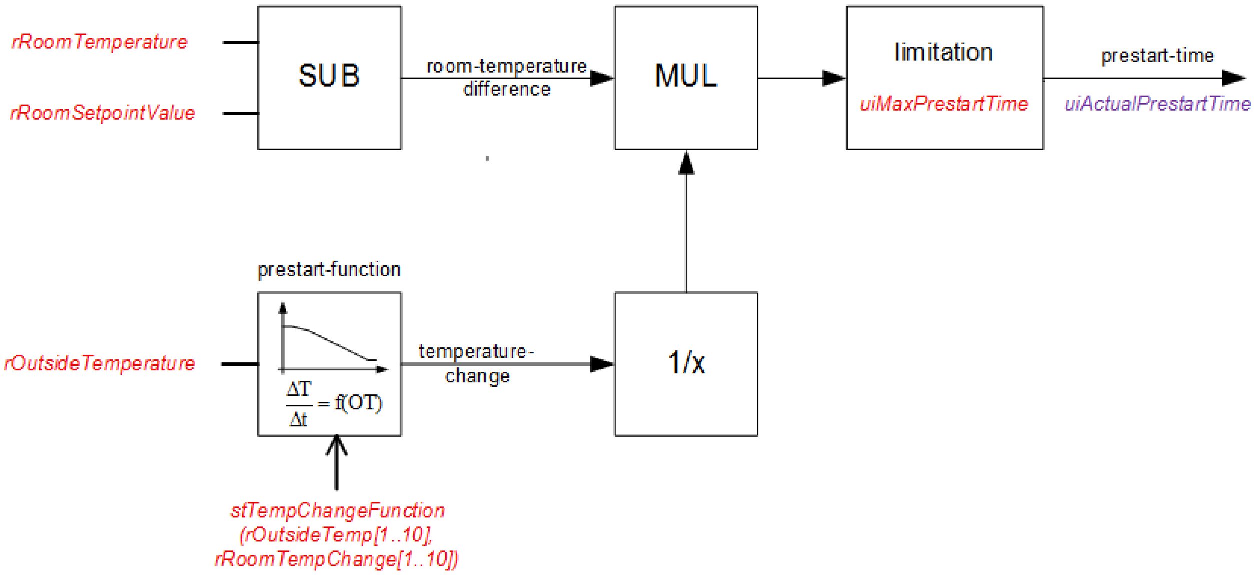
With the structure variable stTempChangeFunction [▸ 499], the function block *FB_HVACOptimizedOn* is based on a table in which the respectively expected room temperature change is assigned for 10 discrete outside temperatures. During the commissioning or the initial starting of the system, the pre-start 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 [▸ 443]. 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})

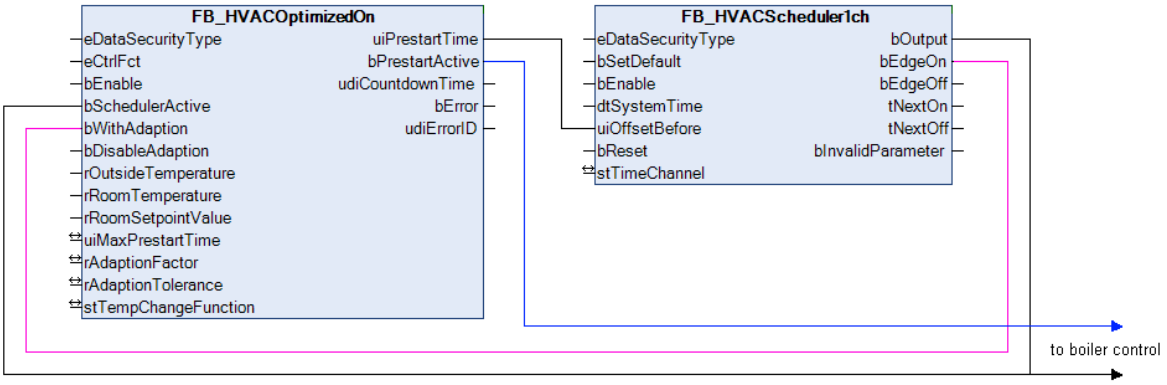
The pre-start time is then determined in accordance with the following sequence:



The input and output variables of the function block are shown here in red. The internal flag *uiActualPrestartTime*, which always contains the current pre-start time, is shown here in violet. The output *uiPrestartTime* corresponds to this flag, but is "frozen" during the pre-start phase. This is explained in more detail below.

Connection of Modules and Prestart Phase

Based on these specifications the module will calculate a prestart time and provide it to the time switch module. If the boiler should have warmed up the buildung at monday morning at 06:00 a.m. and the prestop module calculates at 05:00 a.m. a prestart time of 60 minutes, the boiler will be started immediately. The heating-up behaviour is watched in a reference room during the prestart time phase and the function of the temperature change is corrected.



The prestart module provides a prestart time to the time switch module. If this switches the output *bOutput*, the prestart module gets informed by its input *bSchedulerActive*. Then the prestart module starts an internal countdown with the previously output prestart time *uiPrestartTime* [min]. The countdown either runs 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. It can be used e. g. for the quickstart of the boiler.
The output *uiPrestartTime* follows continously the above described calculation - but when a countdown has been started it is kept constant, in order to inhibit a fluctuation in the outdoor temperature to specify a lower prestart time and switch off the boiler.

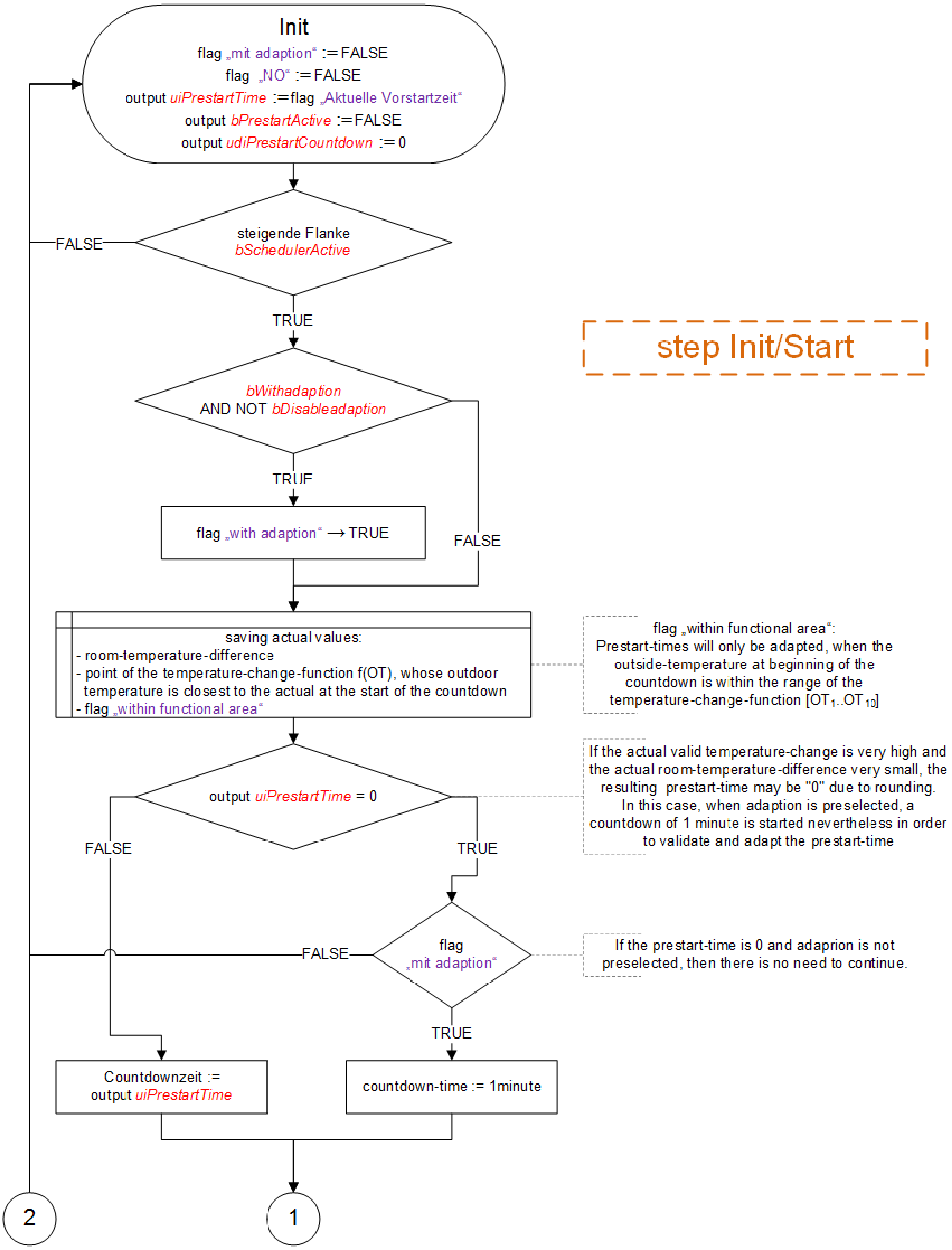
If the time-switch-module requests an adaption of the prestart-function (rising edge at bWithAdaption, red line) the prestart module will decide after the prestart phase, whether the calculated prestart time has been exact, too short or too long (within a tolerance) and will correct the temperature change. For this the prestart time of the point whose outdoor temperature is nearest to the actual (at start of the countdown) is corrected upward or downward. This process is called "Adaption [▶ 429]".

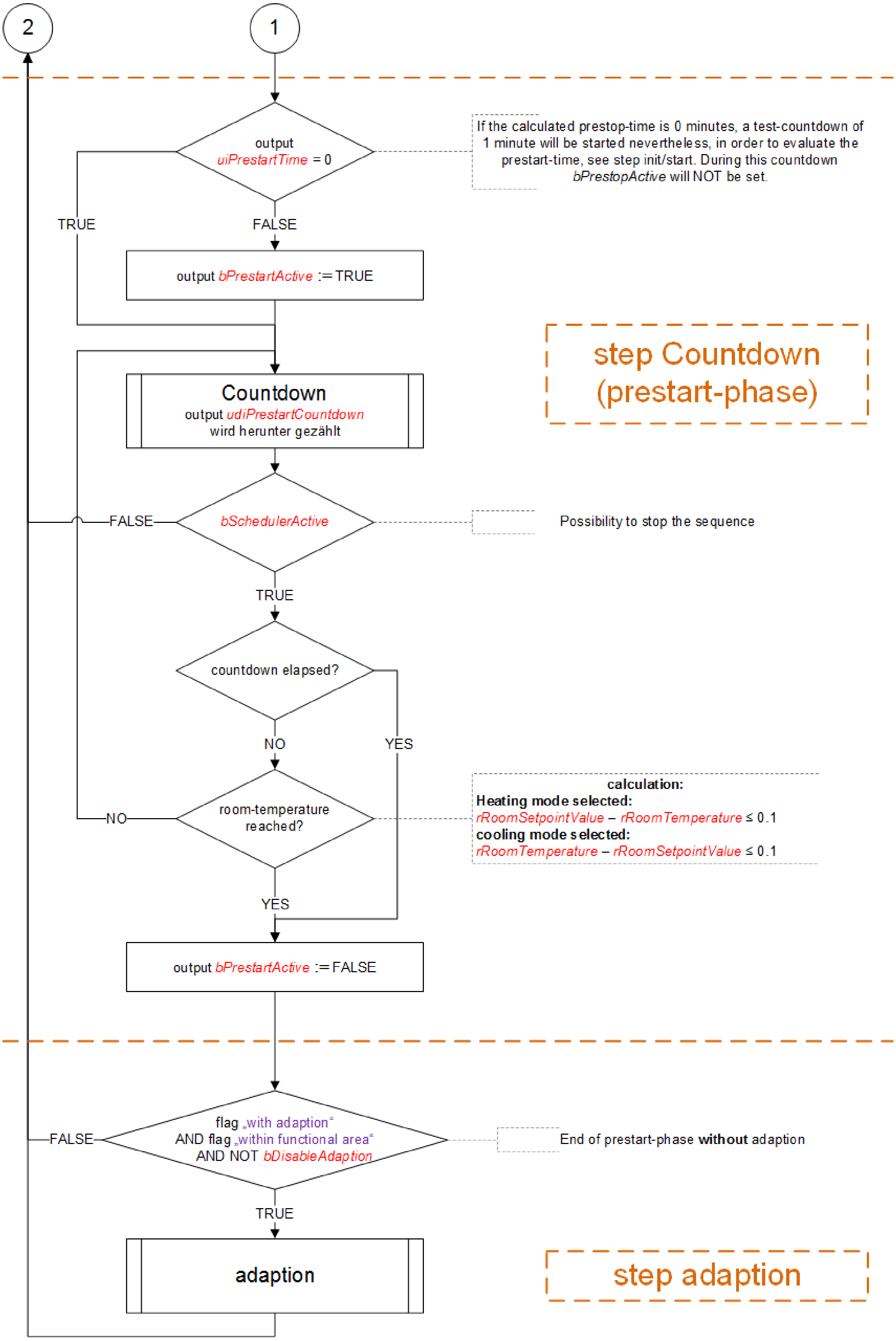
Note: Temperature changes are only corrected, when the outdoor temperature has been within the function range at the beginning of the prestart-phase, see "Adaption [▶ 429]".

When adaption is requested, the input *bWithAdaption* has to be set at the same time as the input *bSchedulerActive*. For *bWithApation* a simple trigger impulse is sufficient. When the input *bSchedulerActive* drops down, a started countdown is immediately aborted. The adaption can be disabled via the input *bDisableAdaption*. This option should be selected, when the temperature change function should not be changed any more after a sufficient number of adaptions.

The following program flow chart is to illustrate the behaviour, in red you see the input- and output variables of the module:

The internal flag "actual prestart time" is the always actual result of the calculation described obove.


Init
flag „mit adaption“ := FALSE
flag „NO“ := FALSE
output uiPrestartTime := flag „Aktuelle Vorstartzeit“
output bPrestartActive := FALSE
output udiPrestartCountdown := 0
FALSE
steigende Flanke bSchedulerActive
TRUE
step Init/Start
bWithadaption AND NOT bDisableadaption
TRUE
flag „with adaption“ → TRUE
FALSE
saving actual values:
- room-temperature-difference
- point of the temperature-change-function f(OT), whose outdoor temperature is closest to the actual at the start of the countdown
- flag „within functional area“
flag „within functional area“: Prestart-times will only be adapted, when the outside-temperature at beginning of the countdown is within the range of the temperature-change-function [OT1..OT10]
output uiPrestartTime = 0
FALSE
TRUE
If the actual valid temperature-change is very high and the actual room-temperature-difference very small, the resulting prestart-time may be "0" due to rounding. In this case, when adaption is preselected, a countdown of 1 minute is started nevertheless in order to validate and adapt the prestart-time
FALSE
flag „mit adaption“
If the prestart-time is 0 and adaprion is not preselected, then there is no need to continue.
TRUE
Countdownzeit := output uiPrestartTime
countdown-time := 1minute
2
1


2
1
output
uiPrestartTime = 0
If the calculated prestop-time is 0 minutes, a test-countdown of 1 minute will be started nevertheless, in order to evaluate the prestart-time, see step init/start. During this countdown bPrestopActive will NOT be set.
TRUE
FALSE
output bPrestartActive := TRUE
step Countdown (prestart-phase)
Countdown
output udiPrestartCountdown
wird herunter gezählt
FALSE
bSchedulerActive
Possibility to stop the sequence
TRUE
countdown elapsed?
NO
YES
calculation:
Heating mode selected:
rRoomSetpointValue – rRoomTemperature ≤ 0.1
cooling mode selected:
rRoomTemperature – rRoomSetpointValue ≤ 0.1
NO
room-temperature reached?
YES
output bPrestartActive := FALSE
FALSE
flag „with adaption"
AND flag „within functional area"
AND NOT bDisableAdaption
End of prestart-phase without adaption
TRUE
adaption
step adaption

The input *bEnable* (not described here), sets the module out of function if it is FALSE: the prestart time *uiPrestartTime* is set "0" and the prestart is not started resp. is immediately reset.

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 ΔT/Δt 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

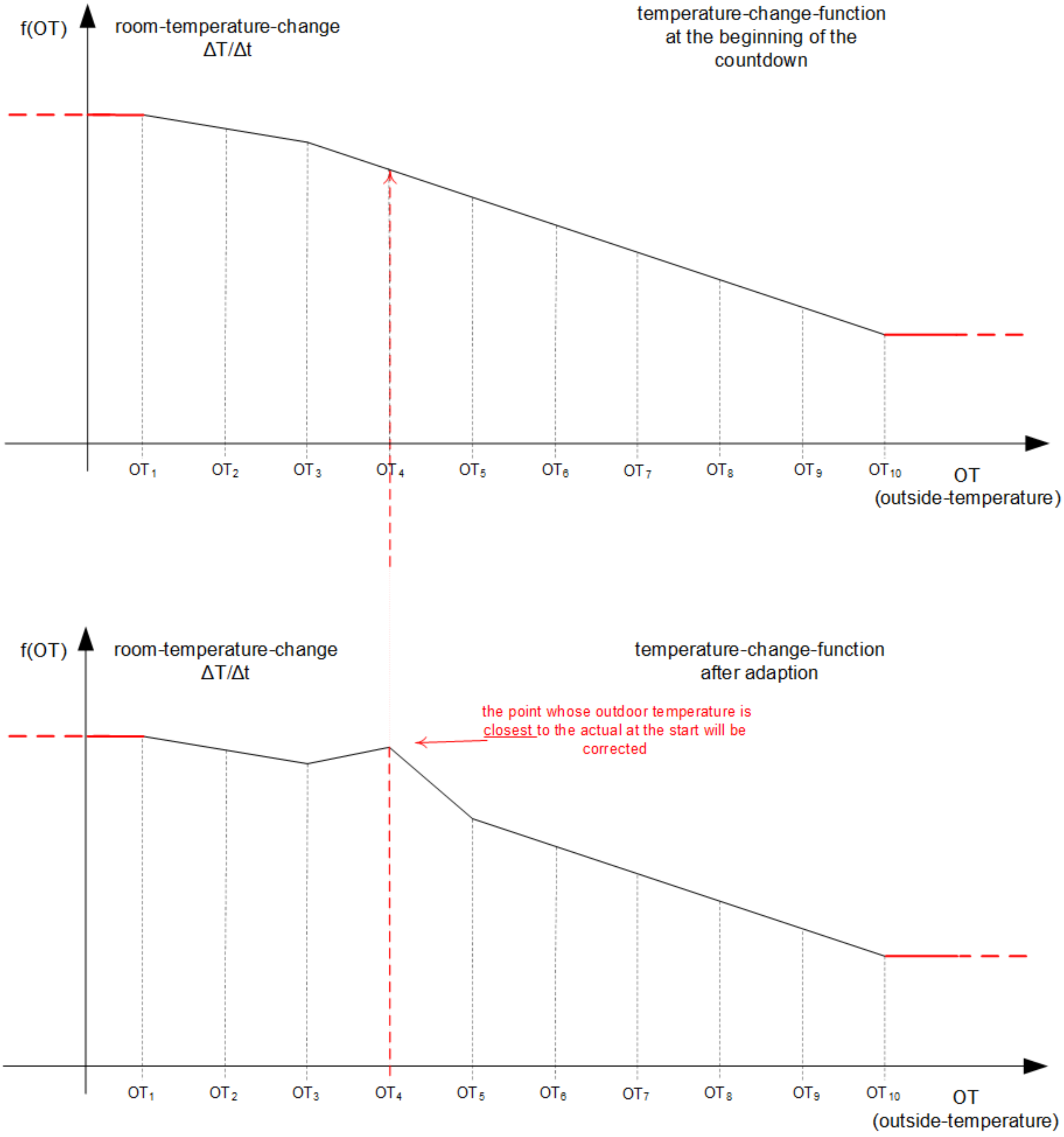
1. no change.
2. too much time was calculated – new value: "Room temperature difference at the start" / "Time elapsed so far"
3. too little time was calculated – new value: ("Room temperature difference at start" - "Room temperature difference at end") / "Countdown time"

The temperature change function is now corrected at the interpolation point whose outside temperature value was nearest to that at the beginning of the pre-stop phase. To this end the point concerned was saved before the start of the countdown, see program flowchart.

However, the previously calculated value ΔT/Δt 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*:

$$\mathrm{f(AT)}_{\mathrm{NEW}} := \frac{\mathrm{f(AT)}_{\mathrm{OLD}} \cdot \left(100 - rAdaptionFactor\right) \;+\; \left(\Delta T/\Delta t_{\mathrm{calculated}} \cdot rAdaptionFactor\right)}{100}$$

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-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* AT_1 to AT_{10}.

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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= eHVACDataSecurityType_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 (see E_BARCtrlFct [▸ 499]). 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 setpoint 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 [▸ 318].

VAR_IN_OUT

```
uiMaxPrestartTime      : UINT;
rAdaptionFactor        : REAL;
rAdaptionTolerance     : REAL;
stTempChangeFunction   : ST_HVACTempChangeFunction;
```

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

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 ΔT/Δt 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.

$$f(AT)_{NEW} := \frac{f(AT)_{OLD} \cdot \left(100 - rAdaptionFactor\right) \quad + \quad \left(\Delta T/\Delta t_{calculated} \cdot rAdaptionFactor\right)}{100}$$

With an adaptation factor of 100% the newly calculated value is adopted fully, while with 0% the old value is retained.

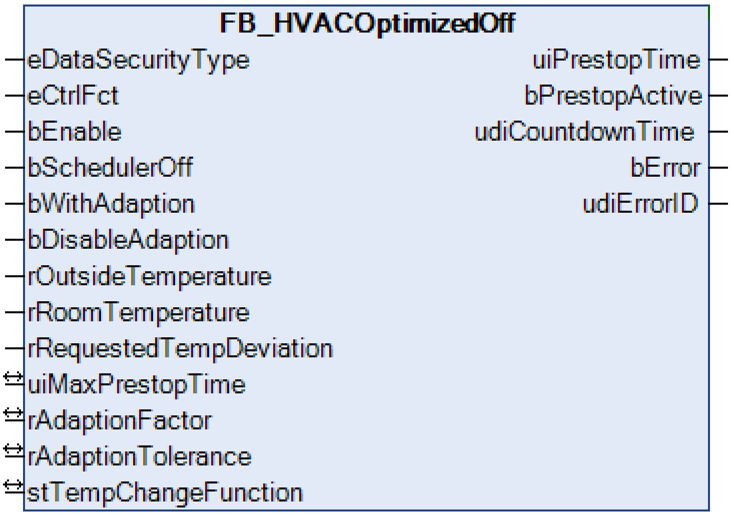
rAdaptionTolerance: If the countdown has expired and the room temperature setpoint has been reached with adaptation activated, no adaptation is carried out because the pre-start time is precisely right. The value *rAdaptionTolerance* defines a tolerance range: If the actual value lies within the range *rRoomSetpointValue* .. *rRoomSetpointValue* + *rAdaptionTolerance*, then this is regarded as reaching the setpoint.

stTempChangeFunction: Structure variable of the type ST_HVACTempChangeFunction [▶ 499] that contains the 10 value pairs (outside temperature, inside 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 [▶ 443].

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.26 FB_HVACOptimizedOff



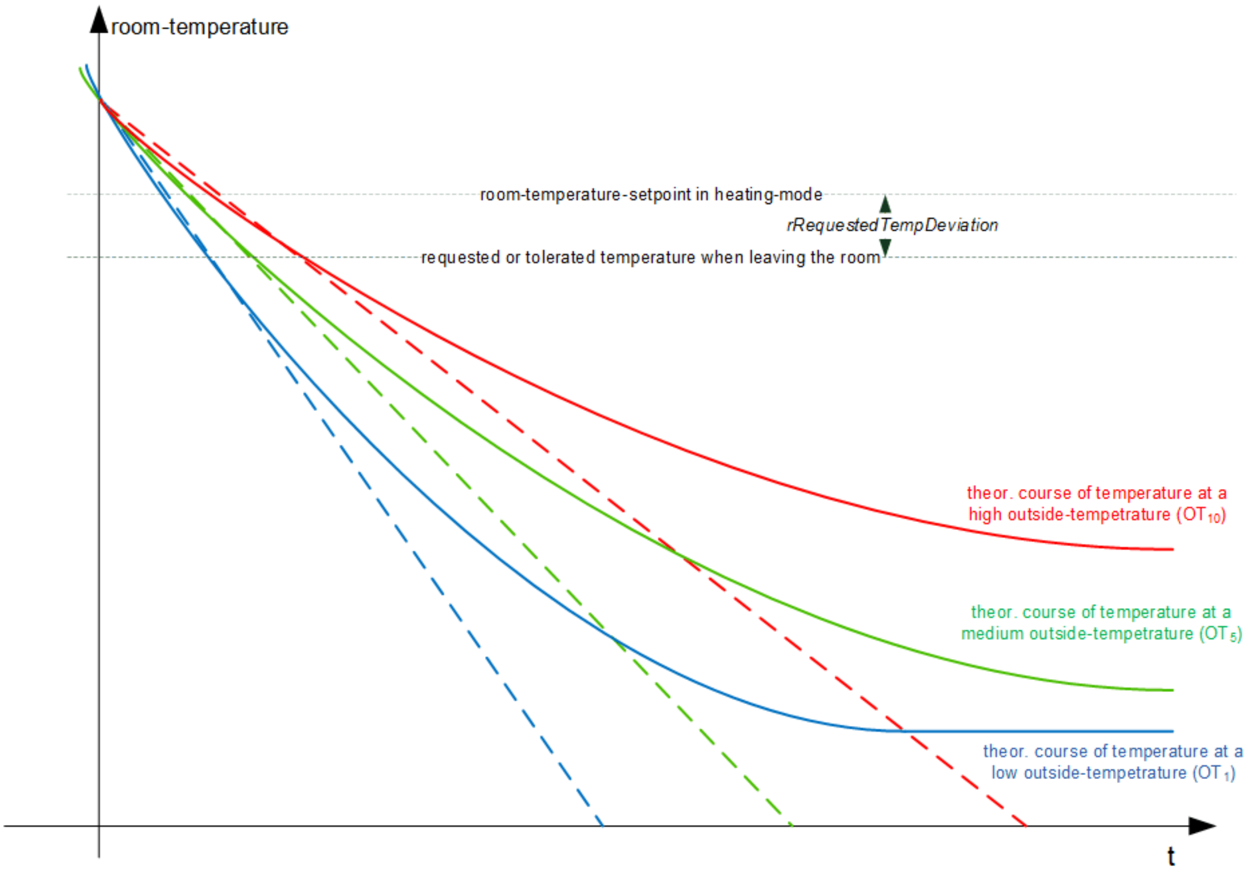
Function block for the optimized switch-off (pre-stop) of boilers and air conditioning systems in conjunction with switching time function blocks.

The explanations in this document relate to the heating behavior. The function block is used in the same way in conjunction with cooling devices.

Calculating the PrestopTime

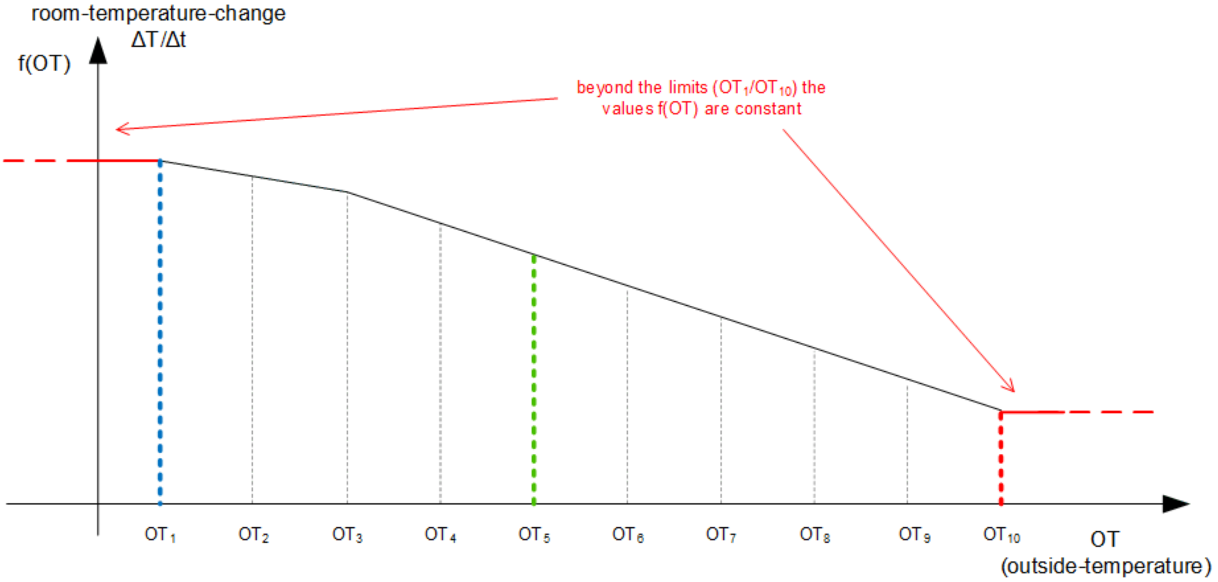
Buildings and parts of buildings which are used at a certain and projectable time, e. g. schools and conference rooms, don´t need to be heated for the whole using-time. Because the buildings save thermal energy, it is possible to shut off the heating earlier. The room temperature declines till the end of usage to a lower niveau which is hardly noticed. This is a desired declining to an objective temperature - the difference of declining is an input of the function block and is called *rRequestedTempDeviation*.

The flow temperature of the boiler refers to the outdoor temperature. Therefore there are different declining processes for different outdoor temperatures which meet basically an exponential function:



To determine the prestop time, the area between the functions of room temperature start value and room temperature target value is assumed linear. So for each outdoor temperatue there is a characteristic temperature change ΔT/Δt, which is illustrated here in a dotted line.

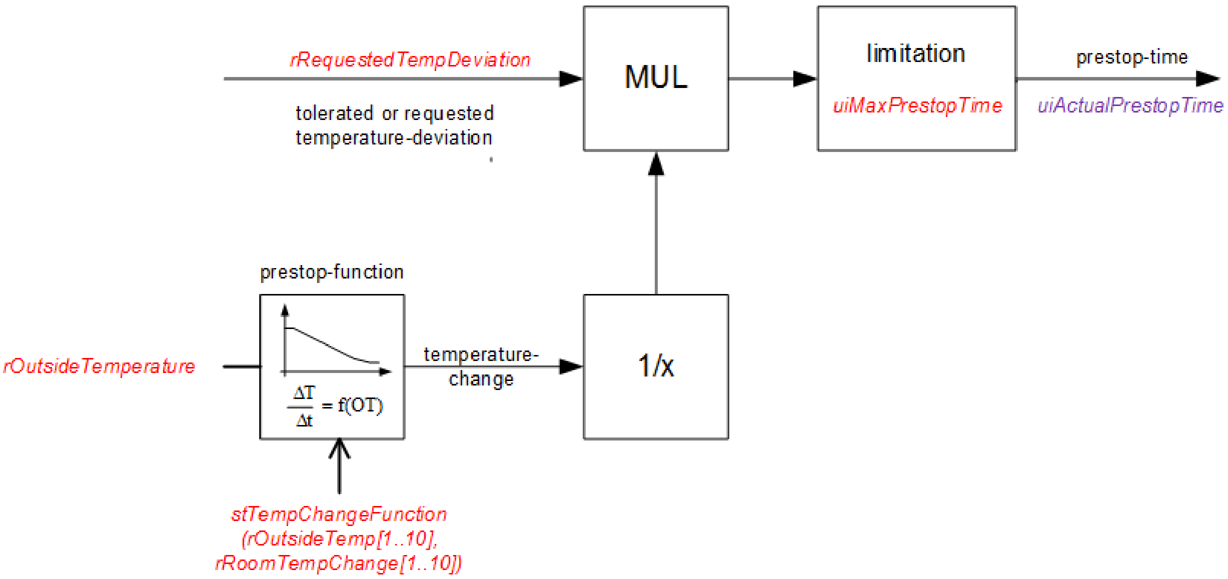
The function block FB_HVACOptimizedOff refers with the variable stTempChangeFunction [▶ 499] to a table, in which there are 10 outdoor temperatures with their corresponding room temperature changes. When commissioning or starting the system for the first time, the pre-start function must be roughly pre-defined, firstly to define the external temperature range and secondly to accelerate the adaptation process. This entry is made with the block FB_HVACTempChangeFunctionEntry [▶ 443]. With these values you can approximately determine the prestop temperature. Typically this temperature-change-function f(OT) has the following course: (the change is positive illustrated):



Function values inside these 10 points are determined via equation of line, values outside correspond to the function value f(OT_1) resp. f(OT_{10}):

- OT < OT_1: f(OT) = f(OT_1)
- OT > OT_{10}: f(OT) = f(OT_{10})

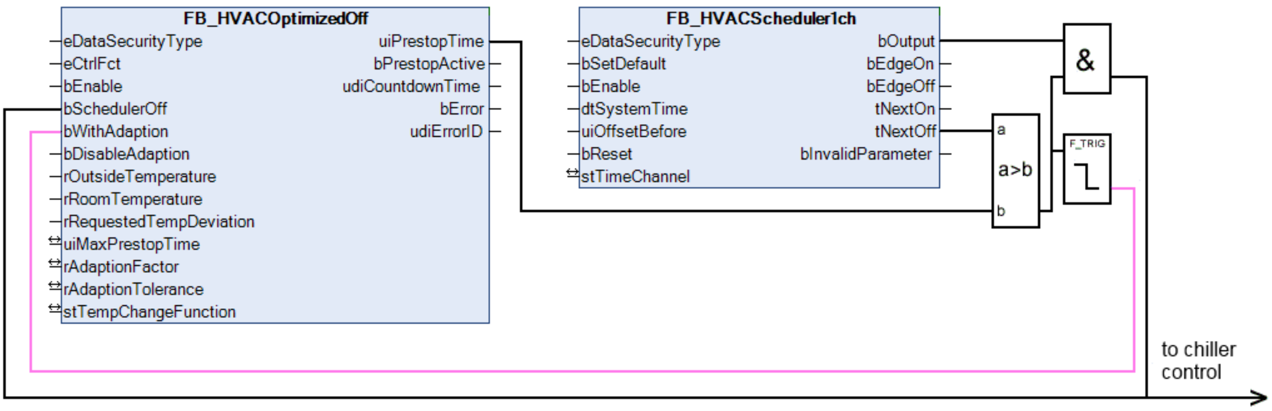
Thus the prestop time is calculated:



The input and output variables of the module are illustrated in red. The internal flag *uiActualPrestopTime* which holds always the actual prestop time is illustrated in violet. The output *uiPrestopTime* corresponds to this flag, but becomes "freezed" during the prestop phase. This will be explained in detail in the following chapter.

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* . The pre-stop function block then internally 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. The *bPrestopActive* output is set during the countdown.

The output *uiPrestopTime* continuously follows the calculation shown above, but is kept constant during a started countdown so that a fluctuation in the outside temperature does not specify a shorter pre-stop time and the boiler is suddenly switched on again.

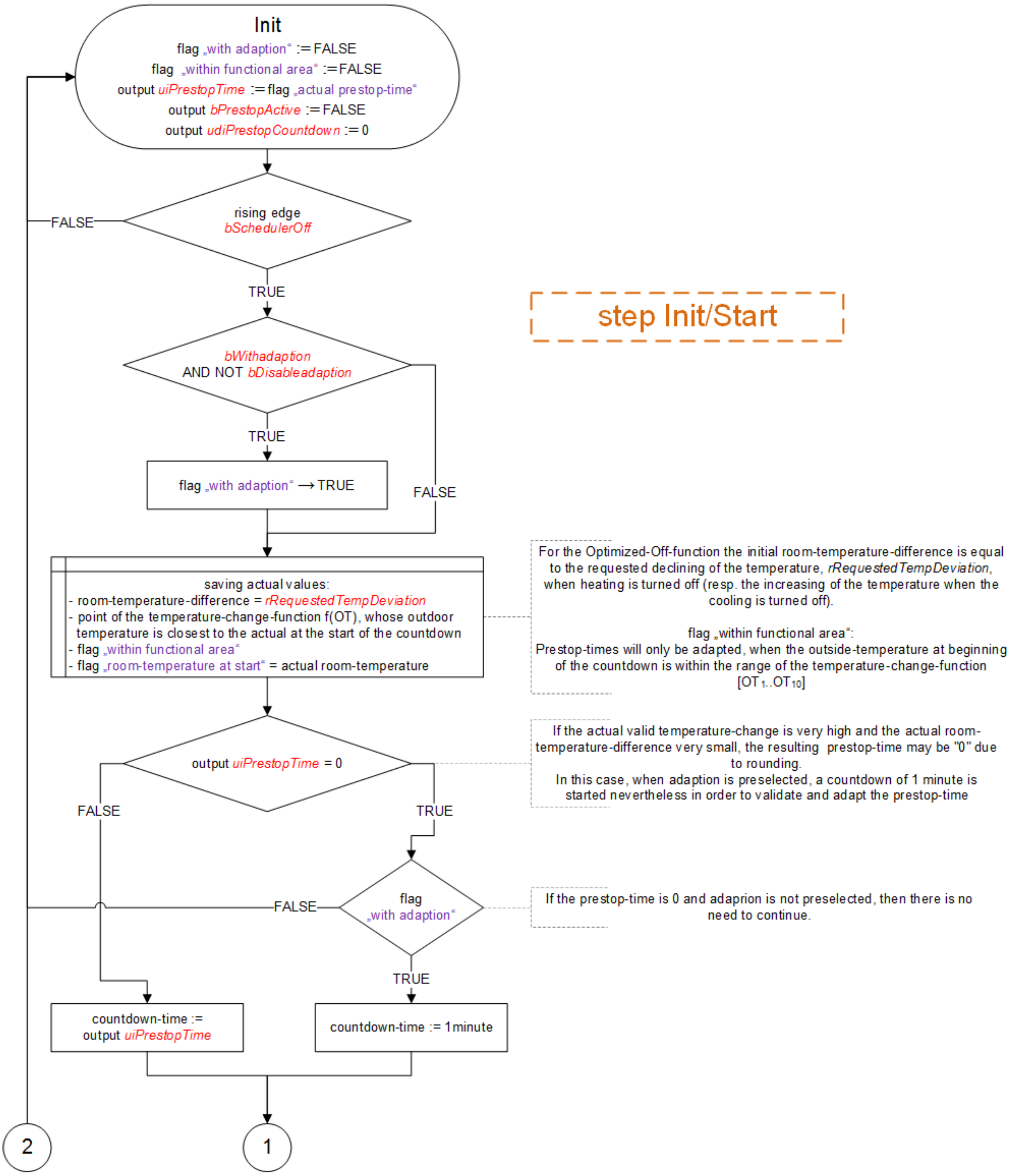
If, in the case of a pre-stop, the timer function block still sends the adaptation order as an edge (red line) to the optimization function block, then the latter will decide following the pre-stop phase whether the previously determined pre-stop time was precisely within a tolerance, or too short or 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 [▶ 439]".

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 [▶ 439]".

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.

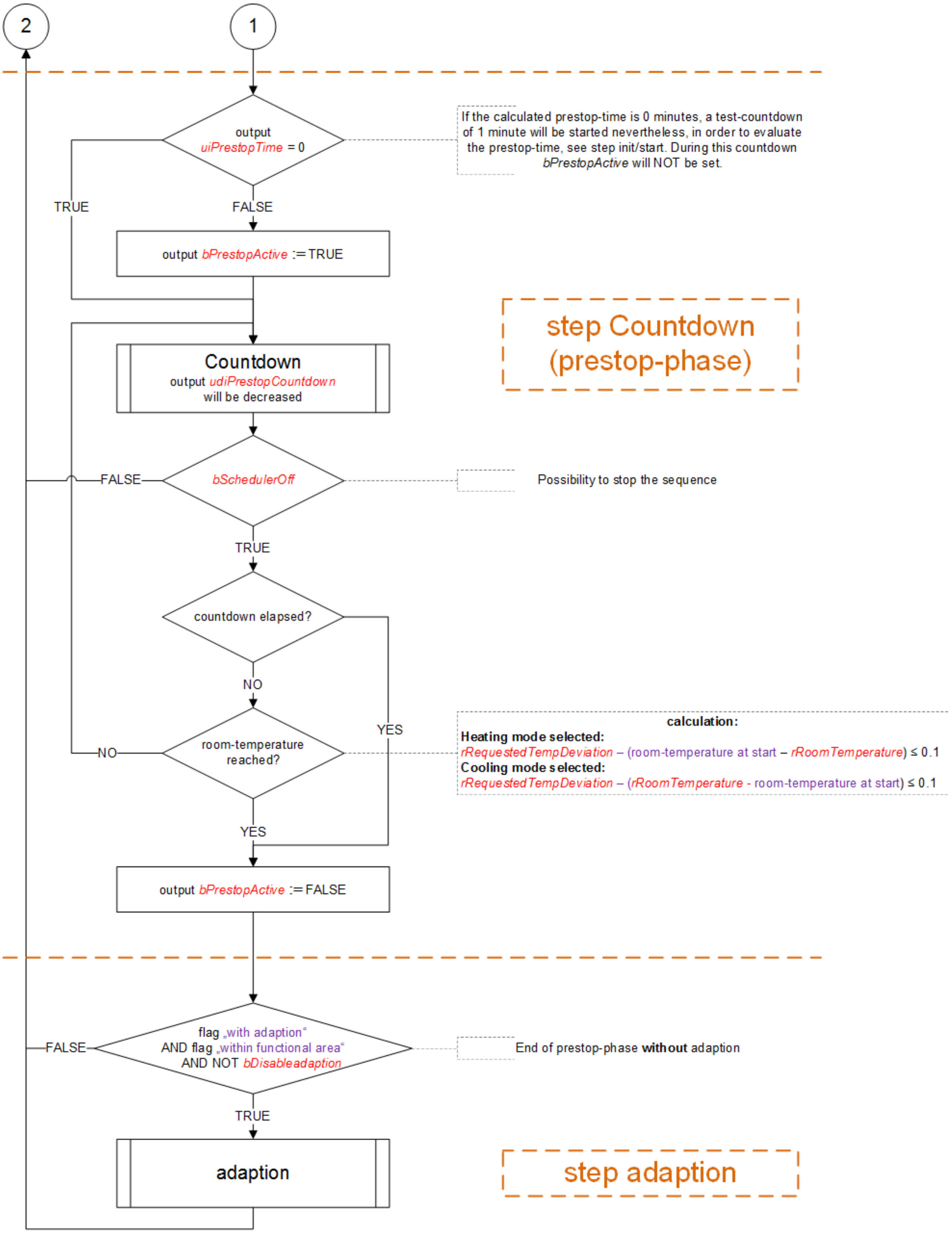

Init
flag „with adaption“ := FALSE
flag „within functional area“ := FALSE
output uiPrestopTime := flag „actual prestop-time“
output bPrestopActive := FALSE
output udiPrestopCountdown := 0
rising edge
bSchedulerOff
FALSE
TRUE
bWithadaption
AND NOT bDisableadaption
TRUE
FALSE
flag „with adaption“ ⟶ TRUE
saving actual values:
- room-temperature-difference = rRequestedTempDeviation
- point of the temperature-change-function f(OT), whose outdoor temperature is closest to the actual at the start of the countdown
- flag „within functional area“
- flag „room-temperature at start“ = actual room-temperature
output uiPrestopTime = 0
FALSE
TRUE
flag
„with adaption“
FALSE
TRUE
countdown-time := output uiPrestopTime
countdown-time := 1minute
2
1
step Init/Start

For the Optimized-Off-function the initial room-temperature-difference is equal to the requested declining of the temperature, *rRequestedTempDeviation*, when heating is turned off (resp. the increasing of the temperature when the cooling is turned off).

flag „within functional area“:
Prestop-times will only be adapted, when the outside-temperature at beginning of the countdown is within the range of the temperature-change-function [OT$_1$..OT$_{10}$]

If the actual valid temperature-change is very high and the actual room-temperature-difference very small, the resulting prestop-time may be "0" due to rounding.
In this case, when adaption is preselected, a countdown of 1 minute is started nevertheless in order to validate and adapt the prestop-time

If the prestop-time is 0 and adaprion is not preselected, then there is no need to continue.


2
1
output
uiPrestopTime = 0
TRUE
FALSE
If the calculated prestop-time is 0 minutes, a test-countdown of 1 minute will be started nevertheless, in order to evaluate the prestop-time, see step init/start. During this countdown bPrestopActive will NOT be set.
output bPrestopActive := TRUE
step Countdown (prestop-phase)
Countdown
output udiPrestopCountdown will be decreased
FALSE
bSchedulerOff
Possibility to stop the sequence
TRUE
countdown elapsed?
NO
YES
NO
room-temperature reached?
calculation:
Heating mode selected:
rRequestedTempDeviation – (room-temperature at start – rRoomTemperature) ≤ 0.1
Cooling mode selected:
rRequestedTempDeviation – (rRoomTemperature - room-temperature at start) ≤ 0.1
YES
output bPrestopActive := FALSE
FALSE
flag „with adaption“
AND flag „within functional area“
AND NOT bDisableadaption
End of prestop-phase without adaption
TRUE
adaption
step adaption

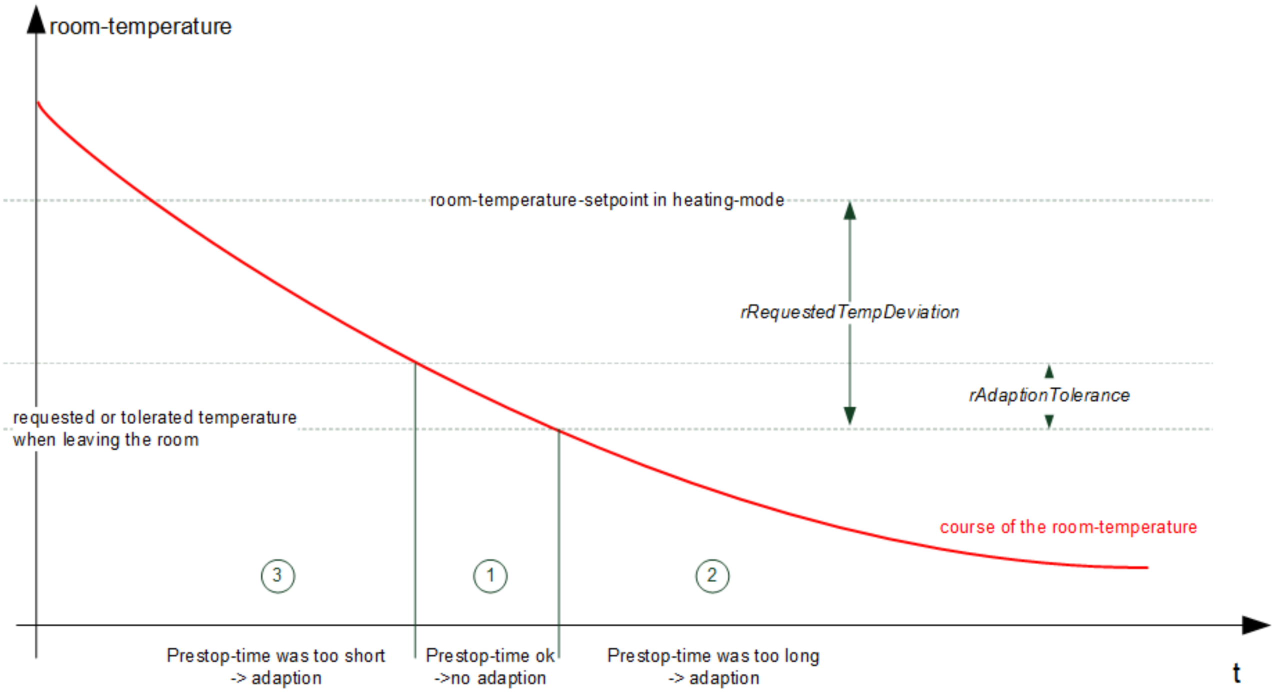
The input *bEnable*, whose mode of operation is not shown here, factually deactivates the function block when FALSE. "0" is output as the pre-stop time *uiPrestopTime* and the sequence shown above is not started or is immediately reset as the case may be.

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.


room-temperature
room-temperature-setpoint in heating-mode
rRequestedTempDeviation
rAdaptionTolerance
requested or tolerated temperature when leaving the room
course of the room-temperature
3
1
2
Prestop-time was too short -> adaption
Prestop-time ok ->no adaption
Prestop-time was too long -> adaption
t

The value ΔT/Δt 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

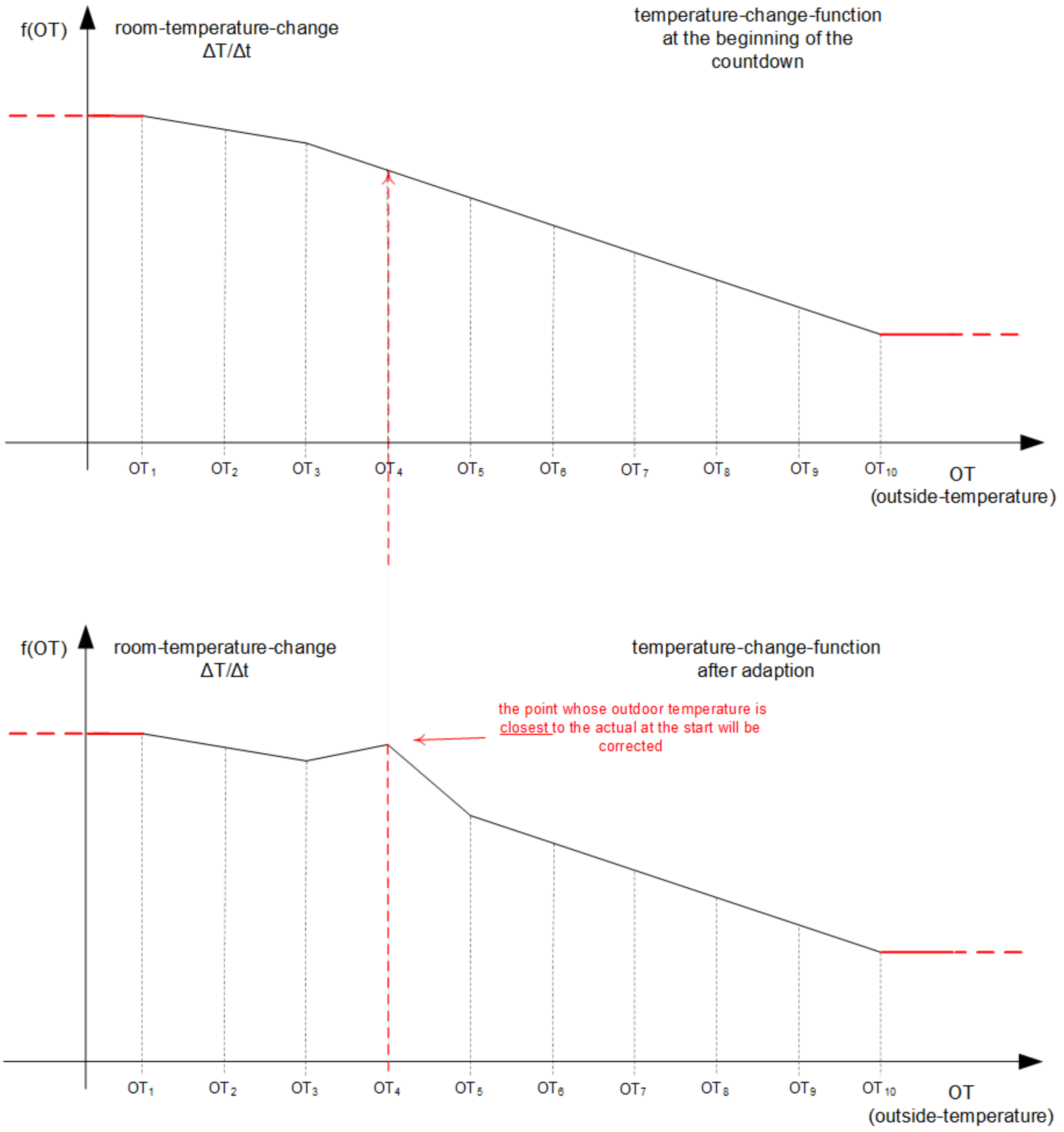
1. no change.
2. too much time was calculated – new value: "requested deviation *rRequestedDeviation*" / "time elapsed so far"
3. too little time was calculated – new value: ("requested deviation *rRequestedDeviation*" - "Deviation at the end") / "Countdown time"

The temperature change function is now corrected at the interpolation point whose outside temperature value was nearest to that at the beginning of the pre-stop phase. To this end the point concerned was saved before the start of the countdown, see program flowchart.

However, the previously calculated value ΔT/Δt 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}.

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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= eHVACDataSecurityType_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 (see E_BARCtrlFct [▶ 499]). 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 setpoint is reached.

bWithAdaption: If this input is set *simultaneously* with the input *bSchedulerOff*, an adaptation takes place after the countdown has elapsed. 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 pre-stop time elapsing, by the room temperature setpoint being reached 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 [▶ 318].

VAR_IN_OUT

```
uiMaxPrestopTime       : UINT;
rAdaptionFactor        : REAL;
rAdaptionTolerance     : REAL;
stTempChangeFunction   : ST_HVACTempChangeFunction;
```

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 to be defined as input variables, they would **not** be able to write a reference variable.
Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/5090272523/.zip.

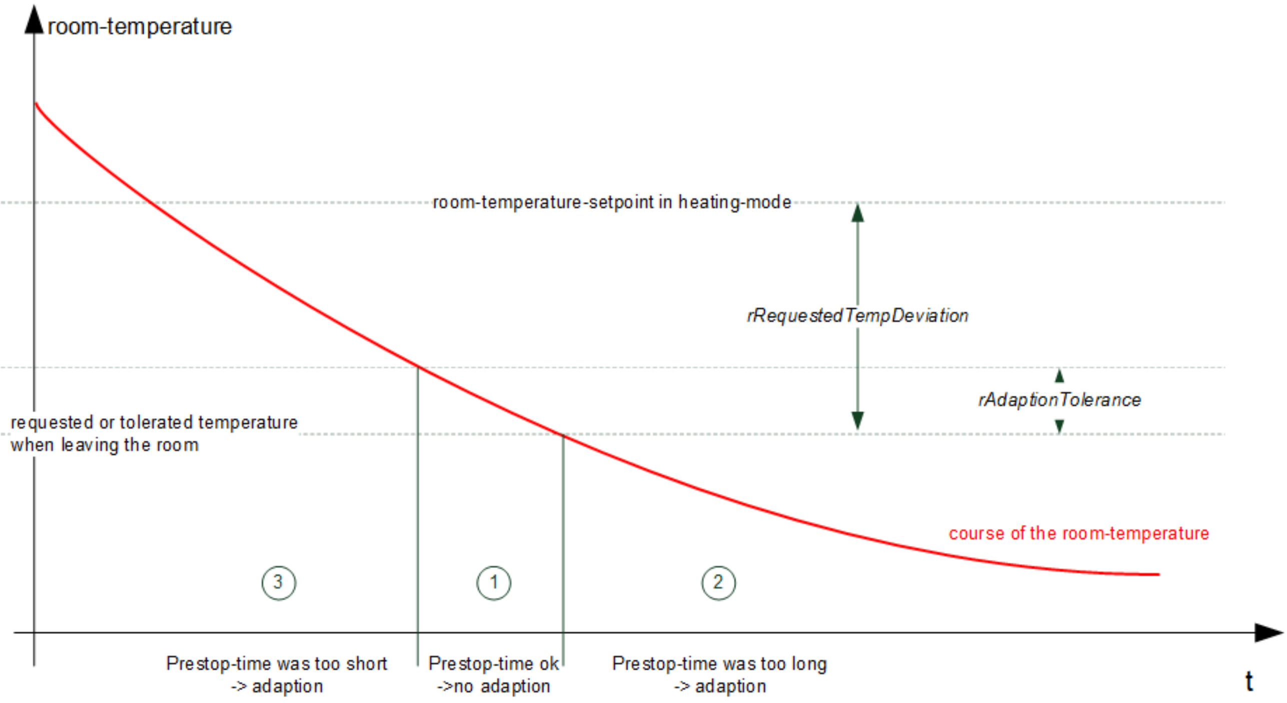
uiMaxPrestopTime: Due to the adaptation the pre-stop times are shortened and lengthened within the temperature change function. While they are logically automatically limited to 0 minutes in the downward direction, the upward limit in minutes can be defined by this input.

rAdaptionFactor: In the adaptation step the temperature change ΔT/Δt 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.

$$f(AT)_{NEW} := \frac{f(AT)_{OLD} \cdot (100 - rAdaptionFactor) \ + \ (\Delta T/\Delta t_{calculated} \cdot rAdaptionFactor)}{100}$$

With an adaptation factor of 100% the newly calculated value is adopted fully, while with 0% the old value is retained.

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:

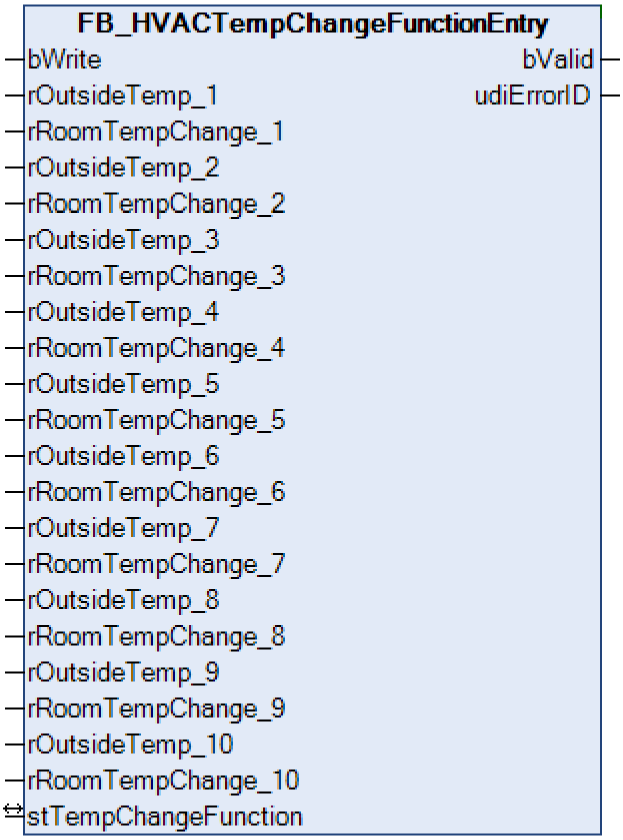

room-temperature
room-temperature-setpoint in heating-mode
rRequestedTempDeviation
rAdaptionTolerance
requested or tolerated temperature when leaving the room
course of the room-temperature
3
1
2
Prestop-time was too short -> adaption
Prestop-time ok ->no adaption
Prestop-time was too long -> adaption
t

stTempChangeFunction: Structure variable of the type ST_HVACTempChangeFunction [▶ 499] that contains the 10 value pairs (outside temperature, inside 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 [▶ 443].

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.27 FB_HVACTempChangeFunction



Function block for the input of the interpolation points of the pre-start function.

In order to keep the function blocks FB_HVACOptimizedOn [▶ 423] and FB_HVACOptimizedOff [▶ 433] well-arranged, they contain no entry of the individual value pairs for the temperature change function. The function blocks access the structure variable of the temperature change function (ST_HVACTempChangeFunction [▶ 499]) 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.

Note: 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 [▸ 318].

VAR_IN_OUT

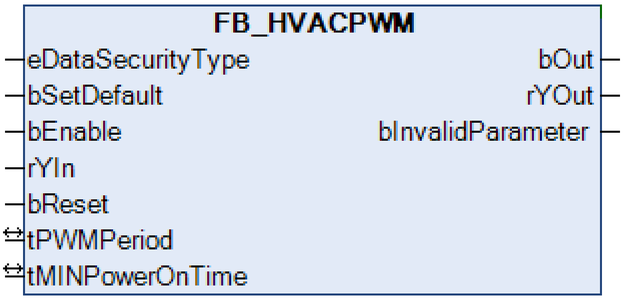
```
stTempChangeFunction   : ST_HVACTempChangeFunction;
```

stTempChangeFunction : structure variable of type ST_HVACTempChangeFunction [▸ 499], which contains the 10 pairs of values (outdoor temperature, room temperature change). These pairs of values, which have to be - sorted by rising outdoor temperature - entered in the field variable, define the 9 fragment lines of the temperature change function.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.28 FB_HVACPWM



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;
bReset             : BOOL;
```

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 variables of the function block (0%..100%).

bReset: Acknowledge input in the event of a fault.

VAR_OUTPUT

```
bOut                : BOOL;
rYOut               : REAL;
bInvalidParameter   : BOOL;
```

bOut: PWM signal.

rYOut: Output of the input variable of the function block (0%..100%).

bInvalidParameter: Indicates that there is an incorrect input parameter. *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.

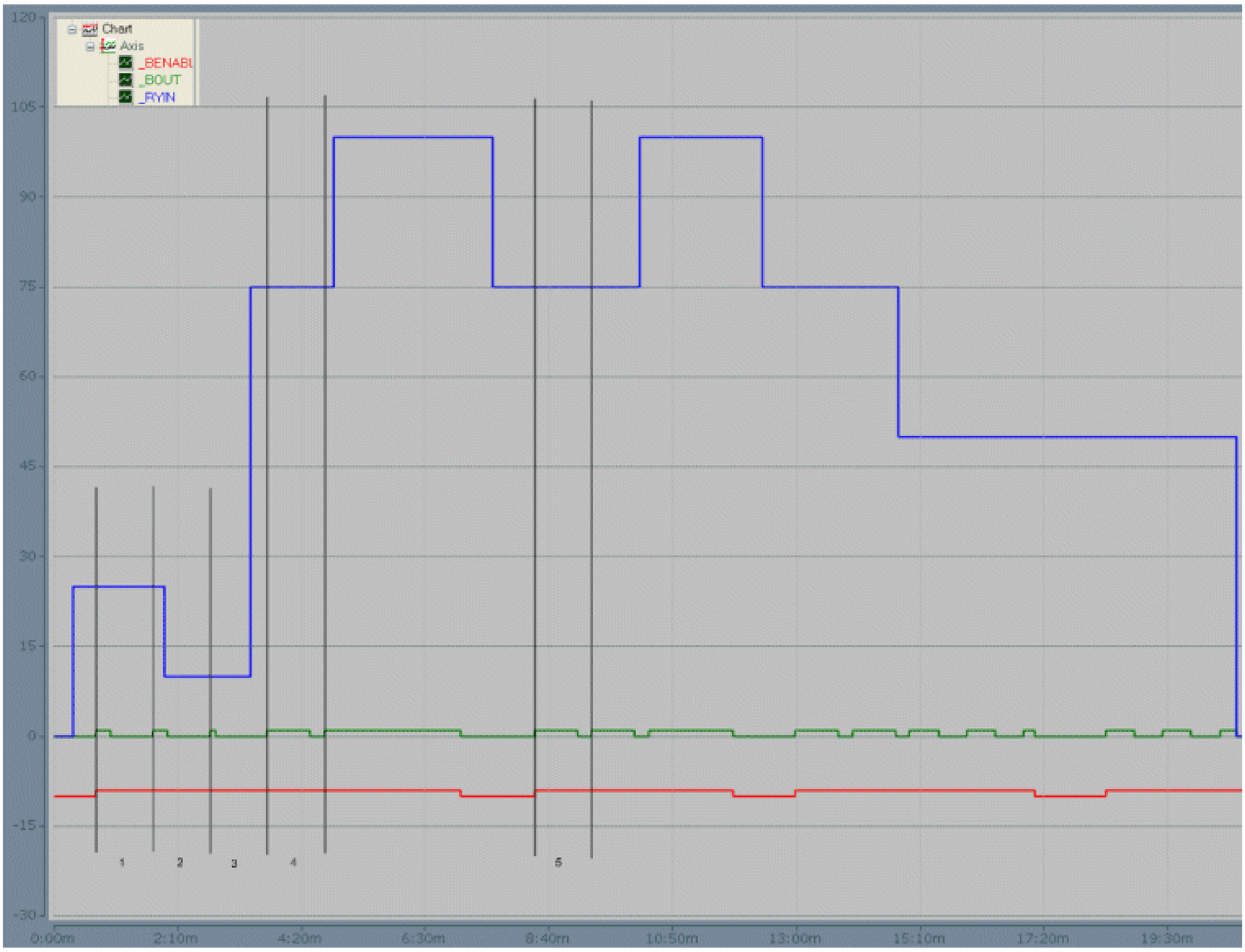
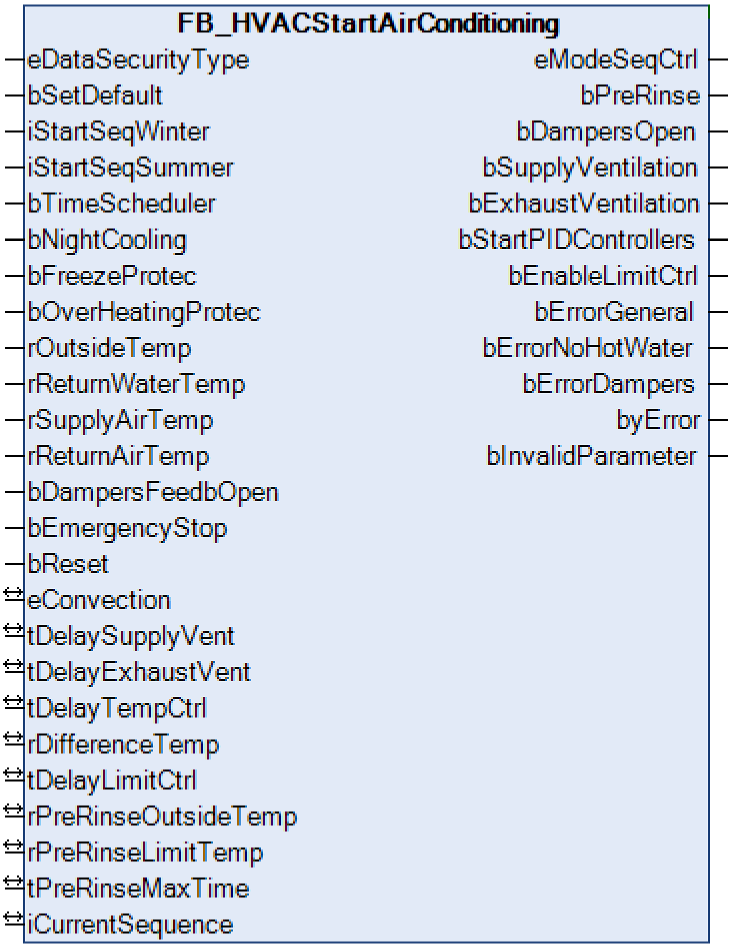


Figure: Scope2 recording for additional explanation of the mode of operation of the function block. The output (*bOut* = TRUE) is set on enabling the function block (*bEnable* = TRUE). See also sections 1 and 5. A change in the input variable during a running period is only taken into account in the next period. See also sections 2, 3 and 4.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.29 FB_HVACStartAirConditioning



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 with hot water before the outside and exhaust air dampers are opened. In air handling units 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 continues until the timer *tPreRinseMaxTime* elapses. **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 active when the system is switched off, the ramp-down times of frequency converters must also be taken into account!**
The output *bErrorDampers* becomes TRUE if no feedback is received from the damper drives within the set time. 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 elapsed, the output *bEnableLimitCtrl* is enabled and, hence, 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 applied to the output the same as that of *iStartSeqWinter*. 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 becomes active in the case of operation requests from the automatic programs, the scheduler activates, summer night cooling, support operation and overheating protection.
In order to activate the sequence controllers, the enumeration variable *eModeSeqCtrl* at the output of the function block is set 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, bFreezeProtec* and *bOverHeatingProtec*. 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 (bOverHeatingProtec) 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.

- Deactivate limit monitoring at the analog inputs
- Deactivate the controllers (reset *iCurrentSequence* to 0)
- Switch off the fans
- Close the dampers after the timer *tDelaySupplyVent* has elapsed.

VAR_INPUT

```
eDataSecurityType      : E_HVACDataSecurityType;
bSetDefault            : BOOL;
iStartSeqWinter        : INT;
iStartSeqSummer        : INT;
bTimeScheduler         : BOOL;
bNightCooling          : BOOL;
bFreezeProtec          : BOOL;
bOverHeatingProtec     : BOOL;
rOutsideTemp           : REAL;
rReturnWaterTemp       : REAL;
rSupplyAirTemp         : REAL;
```

```
rReturnAirTemp        : REAL;
bDampersFeedbOpen     : BOOL;
bEmergencyStop        : BOOL;
bReset                : BOOL;
```

eDataSecurityType: If eDataSecurityType [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 pre-heater 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 switch command.

bNightCooling: Input for the night cooling function.

bFreezeProtec: Input for the frost protection function.

bOverHeatingProtec: 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: A collection of all VAC system error messages that lead to the system switching off can be applied to this input.

bReset: Input for acknowledging the 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 (see E_HVACSequenceCtrlMode [▶ 493]).

bPreRinse: TRUE if the pre-rinsing procedure is active.

bDampersOpen: If TRUE, then the outside and exhaust air dampers are open.

bSupplyVentilation: Switches on the supply air fan if TRUE.

bExhaustVentilation: Switches on the exhaust air fan if TRUE.

bStartPIDControllers: Enables the controllers.

bEnableLimitCtrl: After the timer *tDelayLimitCtrl* has elapsed, the output *bEnableLimitCtrl* is enabled and, hence, 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 there is an incorrect input parameter. *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 (see E_HVACConvectionMode [▶ 491]). 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.

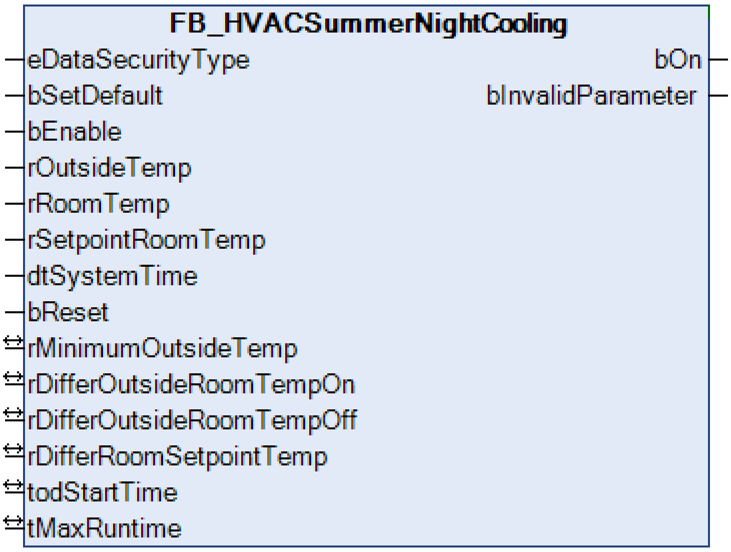
tPreRinseMaxTime: Maximum duration of the pre-rinsing procedure. The variable is saved persistently. Preset to 300 s.

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.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.30 FB_HVACSummerNightCooling



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 for activation of summer night cooling:
- *bEnable* = TRUE
- rOutsideTemp > (*rMinimumOutsideTemp* + 0.2K)
- ((rRoomTemp - rOutsideTemp) > rDifferOutsideRoomTempOn)
- ((rRoomTemp - rSetpointRoomTemp) > rDifferRoomSetpointTemp)
- the system time *dtSystemtime* must be within the timeframe *todStartTime* to 12 noon

For the deactivation of summer night cooling *bOn* = FALSE it is sufficient if one of the following condition is met:
- *bEnable* = FALSE
- (*rOutsideTemp* < (*rMinimumOutsideTemp* - 0.2K))
- ((*rRoomTemp* - *rOutsideTemp*) < *rDifferOutsideRoomTempOff*)
- the system time *dtSystemtime* lies outside the timeframe of *todStartTime* to 12 noon
- summer night cooling was active for the maximum time *tMaxRuntime* within the timeframe between *todStartTime* and 12 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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 of 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 summer night cooling is active.
The following conditions must be met for activation of summer night cooling:
bEnable = TRUE AND *rOutsideTemp* > (*rMinimumOutsideTemp* + 0.2K) AND ((rRoomTemp - rOutsideTemp) > rDifferOutsideRoomTempOn) AND
((rRoomTemp - rSetpointRoomTemp) > rDifferRoomSetpointTemp) AND the system time *dtSystemtime* must lie within the timeframe *todStartTime* to 12 noon.

For the deactivation of summer night cooling *bOn* = FALSE it is sufficient if one of the following condition is met:
bEnable = FALSE OR (*rOutsideTemp* < (*rMinimumOutsideTemp* - 0.2K)) OR ((*rRoomTemp* - *rOutsideTemp*) < *rDifferOutsideRoomTempOff*) OR the system time *dtSystemtime* lies outside the timeframe of *todStartTime* to 12 noon OR the summer night cooling had been active for the specified maximum time *tMaxRuntime* within the timeframe of *todStartTime* to 12 noon. Summer night cooling can be switched on several times within this timeframe.

bInvalidParameter: Indicates that there is an incorrect input parameter. *bInvalidParameter* 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 the specified range.
If an incorrect variable value is present, then the last valid variable value, if one, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

rDifferOutsideRoomTempOn: The difference *rRoomTemp* - *rOutsideTemp* must be greater than the value of *rDifferOutsideRoomTempOn* so that one of the conditions for activating the summer night cooling *bOn* = TRUE is satisfied (0..100).
((rRoomTemp - rOutsideTemp) > rDifferOutsideRoomTempOn)

rDifferOutsideRoomTempOn must be 0.4 K higher than *rDifferOutsideRoomTempOff.* Furthermore *rDifferOutsideRoomTempOn* must be within its specified range.
If an incorrect variable value is present, then the last valid variable value, if one, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

The variable is saved persistently. Preset to 5 °C.

rDifferOutsideRoomTempOff: The difference *rRoomTemp* - *rOutsideTemp* must be less than the absolute value of *rDifferOutsideRoomTempOff* for summer night cooling to be disabled (*bOn* = FALSE).
((rRoomTemp - rOutsideTemp) < rDifferOutsideRoomTempOff)

rDifferOutsideRoomTempOff must be 0.4 K lower than *rDifferOutsideRoomTempOn.* In addition, *rDifferOutsideRoomTempOff* must lie within its range (0..100).
If an incorrect variable value is present, then the last valid variable value, if one, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification. The variable is saved persistently. Preset to 2 °C.

rDifferRoomSetpointTemp: The room temperature must exceed the room temperature setpoint by this value to meet one of the conditions for activation of summer night cooling (*bOn* = TRUE).
((*rRoomTemp* - *rSetpointRoomTemp*) > *rDifferRoomSetpointTemp*)

rDifferRoomSetpointTemp must lie within its range (0..100).
If an incorrect variable value is present, then the last valid variable value, if one, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification.

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 specified range.
If an incorrect variable value is present, then the last valid variable value, if one, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification. The variable is saved persistently. Preset to 2 o'clock.

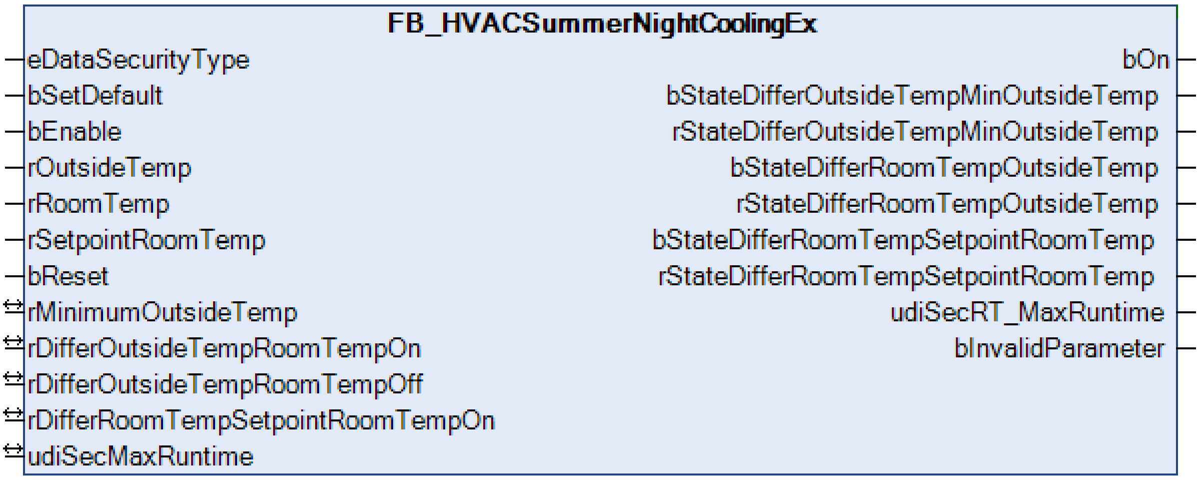
tMaxRuntime: Maximum runtime of the summer night cooling function within the timeframe between *todStartTime* and 12 noon (>0s). Summer night cooling can be switched on several times within this timeframe, but the total time cannot exceed *tMaxRuntime*.

tMaxRuntime must be greater than T#0s.
If an incorrect variable value is present, then the last valid variable value, if one, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification. The variable is saved persistently. Preset to 20 min.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.31 FB_HVACSummerNightCoolingEx


FB_HVACSummerNightCoolingEx
eDataSecurityType
bSetDefault
bEnable
rOutsideTemp
rRoomTemp
rSetpointRoomTemp
bReset
rMinimumOutsideTemp
rDifferOutsideTempRoomTempOn
rDifferOutsideTempRoomTempOff
rDifferRoomTempSetpointRoomTempOn
udiSecMaxRuntime
bOn
bStateDifferOutsideTempMinOutsideTemp
rStateDifferOutsideTempMinOutsideTemp
bStateDifferRoomTempOutsideTemp
rStateDifferRoomTempOutsideTemp
bStateDifferRoomTempSetpointRoomTemp
rStateDifferRoomTempSetpointRoomTemp
udiSecRT_MaxRuntime
bInvalidParameter

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.

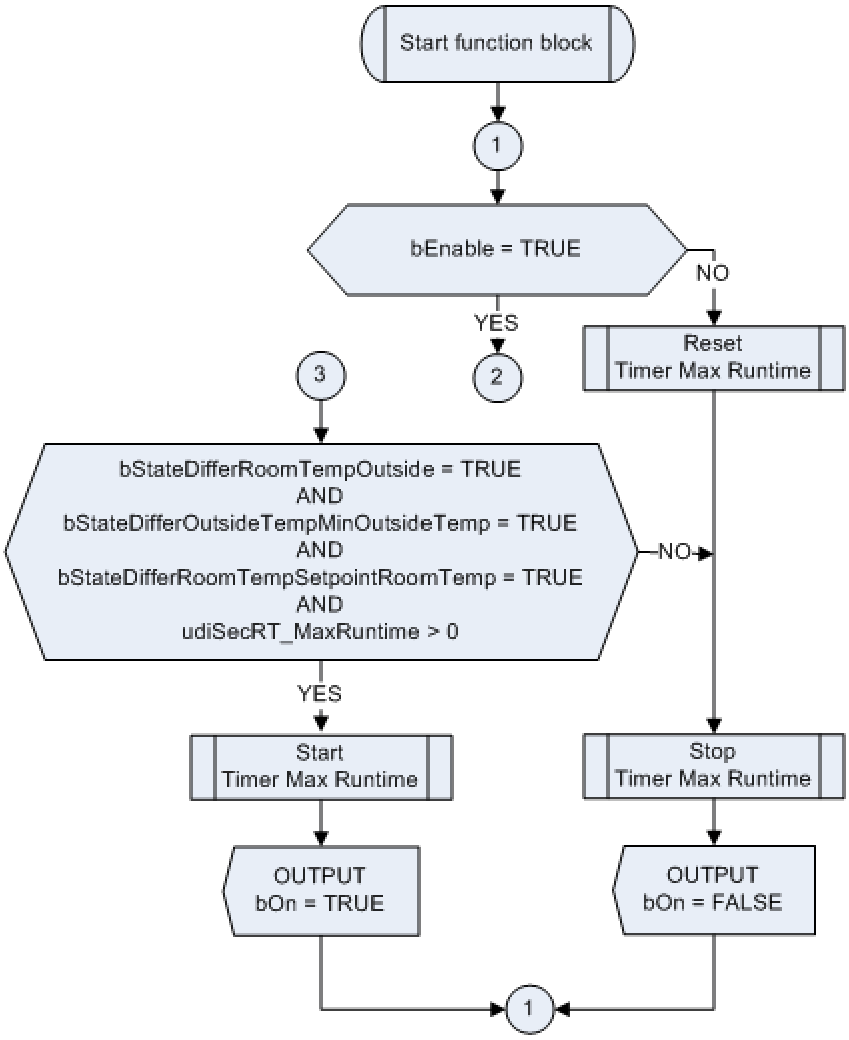
The start conditions for the summer night cooling are defined by parameterizing the **FB_HVACSummerNightCooling** function block. The function block can be used to open motor-driven windows or to switch air conditioning systems to summer night cooling mode outside their normal hours of operation. Summer night cooling mode is active if the output variable *bOn* is TRUE.

The following conditions must be met for activation of summer night cooling:
- *bEnable* = TRUE
- *rOutsideTemp* > (*rMinimumOutsideTemp* + 0.2K)*;* the value 0.2K is an internal constant of the function block.
- (rRoomTemp - rOutsideTemp) > rDifferOutsideTempRoomTempOn
-(rRoomTemp - rSetpointRoomTemp) > rDifferRoomTempSetpointRoomTempOn
- *udiSecRT_MaxRuntime* > 0

For the deactivation of summer night cooling *bOn* = FALSE it is sufficient if one of the following condition is met:
- *bEnable* = FALSE
- *rOutsideTemp* < (*rMinimumOutsideTemp* - 0.2K); the value 0.2K is an internal constant of the function block.
- (*rRoomTemp* - *rOutsideTemp*) < *rDifferOutsideTempRoomTempOff*
-*udiSecRT_MaxRuntime* = 0

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 lease one PLC cycle. The function block and the summer night cooling function can be enabled via a timer.

Program flowchart



VAR_INPUT

```
eDataSecurityType    : E_HVACDataSecurityType;
bSetDefault          : BOOL;
bEnable              : BOOL;
rOutsideTemp         : REAL;
rRoomTemp            : REAL;
rSetpointRoomTemp    : REAL;
bReset               : BOOL;
```

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▶ 491]:= 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*. Once the time *udiSecMaxRuntime* has elapsed the summer night cooling function can only be reactivated if *bEnable* is FALSE for at lease 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 of 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 for activation of summer night cooling:
- *bEnable* = TRUE
- *rOutsideTemp* > (*rMinimumOutsideTemp* + 0.2K)*;* the value 0.2K is an internal constant of the function block.
- (rRoomTemp - rOutsideTemp) > rDifferOutsideTempRoomTempOn
-(rRoomTemp - rSetpointRoomTemp) > rDifferRoomTempSetpointRoomTempOn
- *udiSecRT_MaxRuntime* > 0

For the deactivation of summer night cooling *bOn* = FALSE it is sufficient if one of the following condition is met:
- *bEnable* = FALSE
- *rOutsideTemp* < (*rMinimumOutsideTemp* - 0.2K); the value 0.2K is an internal constant of the function block.
- (*rRoomTemp* - *rOutsideTemp*) < *rDifferOutsideTempRoomTempOff*
-*udiSecRT_MaxRuntime* = 0

bStateDifferOutsideTempMinOutsideTemp: Status display. *bStateDifferOutsideTempMinOutsideTemp* is *TRUE* if *rOutsideTemp* > (rMinimumOutsideTemp + 0.2). *bStateDifferOutsideTempMinOutsideTemp* is *FALSE* if *rOutsideTemp* < (rMinimumOutsideTemp - 0.2).

rStateDifferOutsideTempMinOutsideTemp: Value of the difference *rOutsideTemp* - *rMinimumOutsideTemp*

bStateDifferRoomTempOutsideTemp: Status display. *bStateDifferRoomTempOutsideTemp* is TRUE if (*rRoomTemp* - *rOutsideTemp*) > *rDifferOutsideTempRoomTempOn*. *bStateDifferRoomTempOutsideTemp* is FALSE if (*rRoomTemp* - *rOutsideTemp*) < *rDifferOutsideTempRoomTempOff.*

rStateDifferRoomTempOutsideTemp: Value of the difference *rRoomTemp* - *rOutsideTemp*

bStateDifferRoomTempSetpointRoomTemp: Status display. *bStateDifferRoomTempSetpointRoomTemp* is TRUE if (*rRoomTemp* - *rSetpointRoomTemp*) > *rDifferRoomTempSetpointRoomTempOn*. *bStateDifferRoomTempSetpointRoomTemp* is FALSE if (*rRoomTemp* - *rSetpointRoomTemp*) <= *rDifferRoomTempSetpointRoomTempOn.*

rStateDifferRoomTempSetpointRoomTemp: Value of the difference *rRoomTemp* - *rSetpointRoomTemp*

udiSecRT_MaxRuntime: Remaining summer night cooling time. If *udiSecRT_MaxRuntime* = 0 the summer night cooling function can only be reactivated if *bEnable* is FALSE for at lease one PLC cycle

bInvalidParameter: Indicates that there is an incorrect input parameter. *bInvalidParameter* 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 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).

rMinimumOutsideTemp must be within the specified range.
If an incorrect variable value is present, then the last valid variable value, if one, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification. The variable is saved persistently. Preset to 10 °C.

rDifferOutsideTempRoomTempOn: The difference *rRoomTemp* - *rOutsideTemp* must be greater than the value of *rDifferOutsideRoomTempOn* so that one of the conditions for activating the summer night cooling *bOn* = TRUE is satisfied (0..10).
((rRoomTemp - rOutsideTemp) > rDifferOutsideRoomTempOn)

rDifferOutsideRoomTempOn must be 0.4 K higher than *rDifferOutsideRoomTempOff.* Furthermore *rDifferOutsideRoomTempOn* must be within its specified range.
If an incorrect variable value is present, then the last valid variable value, if one, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification. The variable is saved persistently. Preset to 5 °C.

rDifferOutsideTempRoomTempOff: The difference *rRoomTemp* - *rOutsideTemp* must be less than the amount of *rDifferOutsideTempRoomTempOff* for summer night cooling to be disabled (*bOn* = FALSE).
((rRoomTemp - rOutsideTemp) < rDifferOutsideRoomTempOff)

rDifferOutsideRoomTempOff must be 0.4 K lower than *rDifferOutsideRoomTempOn.* In addition, *rDifferOutsideRoomTempOff* must lie within its range (0..10).
If an incorrect variable value is present, then the last valid variable value, if one, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification. The variable is saved persistently. Preset to 2 °C.

rDifferRoomTempSetpointRoomTempOn: The room temperature must exceed the room temperature setpoint by this value to meet one of the conditions for activation of summer night cooling (*bOn* = TRUE) (0..10).
((*rRoomTemp* - *rSetpointRoomTemp*) > *rDifferRoomSetpointTemp*)

rDifferRoomSetpointTemp must lie within its range.
If an incorrect variable value is present, then the last valid variable value, if one, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification. The variable is saved persistently. Preset to 2 °C.

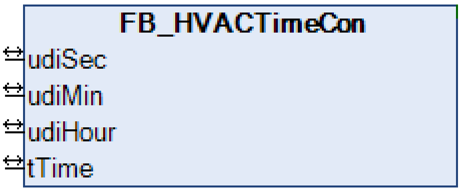
udiSecMaxRuntime: Maximum runtime of the summer night cooling function. Summer night cooling can be switched on and off several times within the time. The remaining summer night cooling time is indicated via the variable *udiSecRT_MaxRuntime*.

udiSecMaxRuntime must be greater than 0.
If an incorrect variable value is present, then the last valid variable value, if one, will be taken. If there is no valid last value, operation continues with the default value. *bInvalidParameter* is set in the case of an incorrect parameter specification. The variable is saved persistently. Preset to 180.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.32 FB_HVACTimeCon



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).

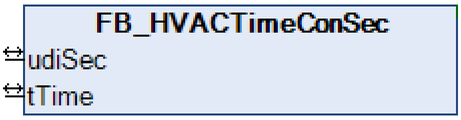
udiHour: Variable that displays the hours (0..1191).

tTime: Variable of the type Time.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.33 FB_HVACTimeConSec



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;
```

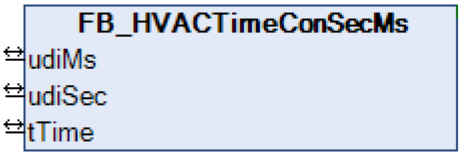
udiSec: Variable that displays the seconds (0..4294967)

tTime: Variable of the type Time.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.34 FB_HVACTimeConSecMs



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).

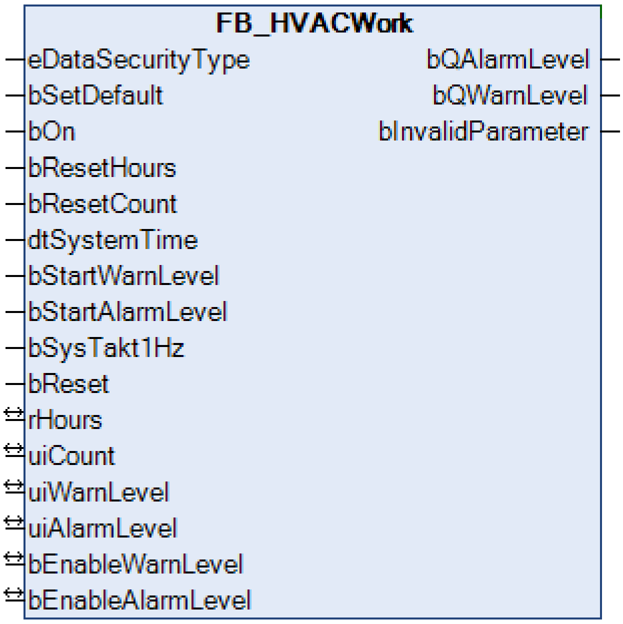
udiSec: Variable that displays the seconds (0..4294966).

tTime: Variable of the type Time.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.7.35 FB_HVACWork



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 sec. 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 rising 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 [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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: Operation signal; 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 hour 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 *bInvalidParameter*.

VAR_OUTPUT

```
bQAlarmLevel        : BOOL;
bQWarnLevel         : BOOL;
bInvalidParameter   : BOOL;
```

bQAlarmLevel: Operating hour counter has reached the alarm level.

bQWarnLevel: Operating hour counter has reached the warning level.

bInvalidParameter: An error occurred during the plausibility check. It is deleted again by *bReset*.

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.

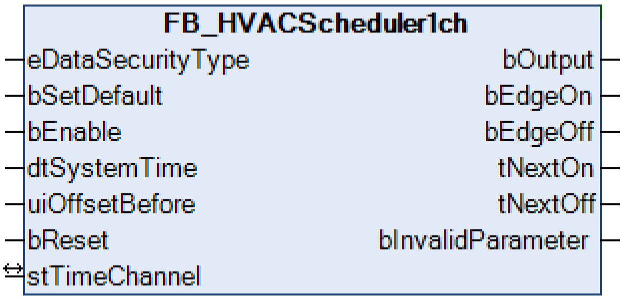
bEnableAlarmLevel: Enables the output *bQAlarmLevel*. The variable is saved persistently. Preset to FALSE.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.8 HVAC Time schedule

5.1.8.1 FB_HVACScheduler1ch



This function block is a weekly scheduler with one channel. Any desired combination of days of the week can be assigned to the channel. The 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 channel are 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 channel stTimeChannel remains active.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
dtSystemTime      : DT;
uiOffsetBefore    : UINT;
bReset            : BOOL;
```

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 variable *bEnable*. If *bEnable* = FALSE there is no switching at the output *bOutput*.

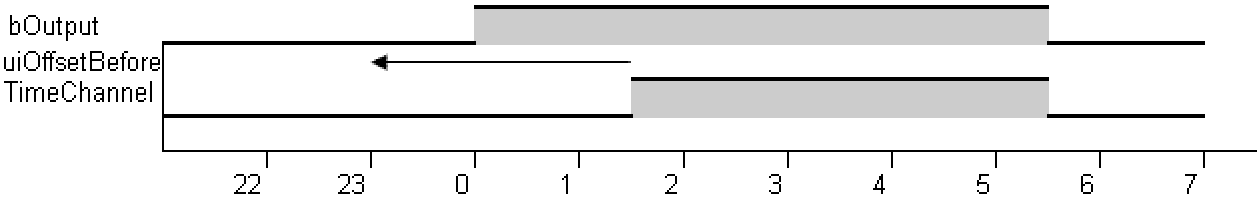
dtSystemTime: The variable *dtSystemTime* transfers the computer system time to the timer program.

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 *bInvalidParameter*

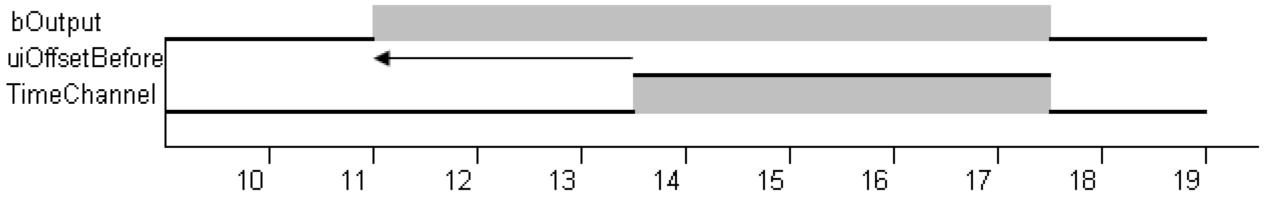
Example 1:

Switch-on period from 1:30 to 5:30; *uiOffsetBefor* has the value 150 min. The switch-on time is brought forward by only 90 minutes to 0:00.



Example 2:

Switch-on period from 13:30 to 17:30; uiOffsetBefor has the value 150 min. The switch-on time is brought forward by 150 minutes to 11:00.



VAR_OUTPUT

```
bOutput           : BOOL;
bEdgeOn           : BOOL;
bEdgeOff          : BOOL;
tNextOn           : TIME;
tNextOff          : TIME;
bInvalidParameter: BOOL;
```

bOutput: Is TRUE if one of the 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 program. Maximum value 10080 minutes (1 week).

tNextOff: Time until the next switch-off of the program. Maximum forecast until midnight.

bInvalidParameter: An error occurred during the plausibility check. It is deleted again by *bReset*.

VAR_IN_OUT

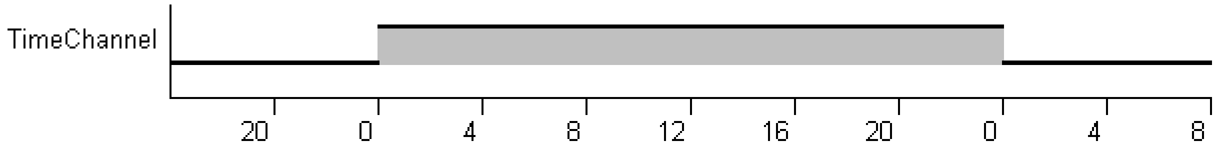
```
stTimeChannel      : ST_HVACTimeChannel;
```

stTimeChannel: With the variable *stTimeChannel* it is possible to generate one switch-on time per day of the week (see ST_HVACTimeChannel [▶ 499]).

Special notes regarding time input

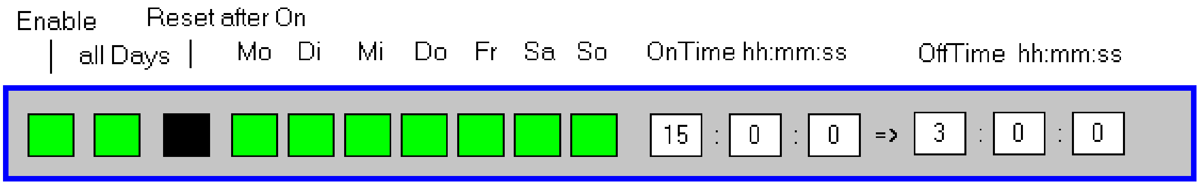
The following applies:

- 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:

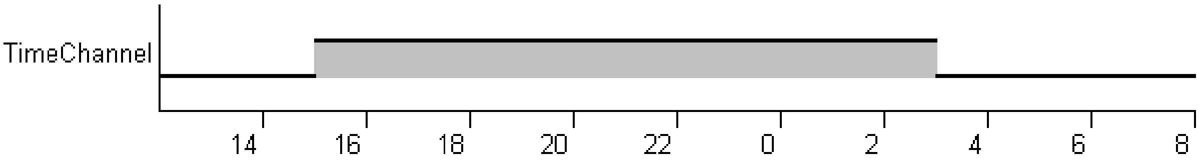
If the switch-on time is later than the switch-off time, the switch-on period is extended through midnight into the next day, regardless of whether the next day is selected.



Enable | all Days | Reset after On | 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).

Example:

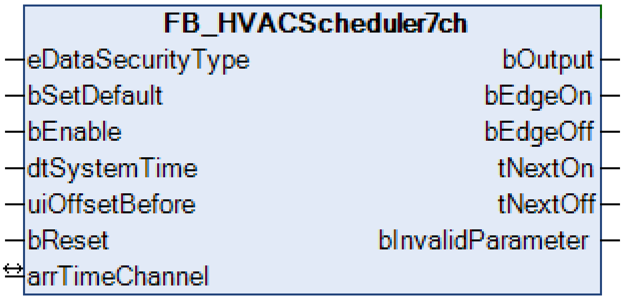
```
VAR_GLOBAL
    stTimeChannel : ST_HVACTimeChannel;
END_VAR
```

Within the weekly planner each channel is enabled via the variable *bEnable* (On/Off). If *bEnable* = FALSE, then the 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 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.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.8.2 FB_HVACScheduler7ch



This function block is a weekly scheduler with 7 channels. Any desired combination of days of the week can be assigned to any channel. The 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 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 channels arrTimeChannel remain active.

VAR_INPUT

```
eDataSecurityType  : E_HVACDataSecurityType;
bSetDefault        : BOOL;
bEnable            : BOOL;
dtSystemTime       : DT;
uiOffsetBefore     : UINT;
bReset             : BOOL;
```

eDataSecurityType: If eDataSecurityType [▸ 491]:= *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 [▸ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▸ 491]:= *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 [▸ 491]:= 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 variable *bEnable*. If *bEnable* = FALSE there is no switching at the output *bOutput*.

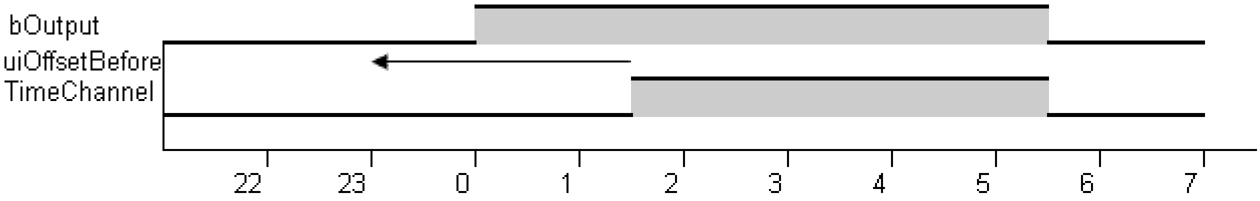
dtSystemTime: The variable *dtSystemTime* transfers the computer system time to the timer program.

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 *bInvalidParameter*.

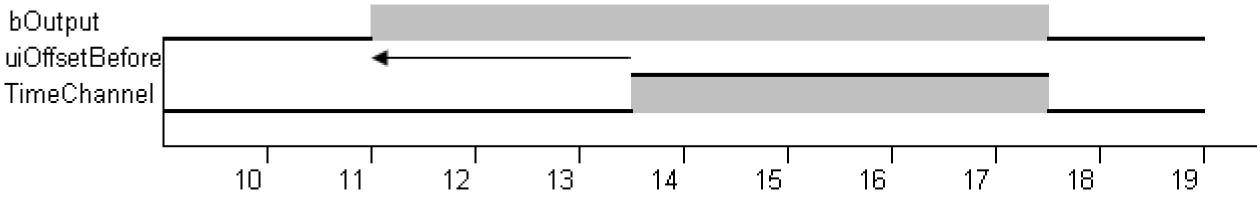
Example 1:

Switch-on period from 1:30 to 5:30; *uiOffsetBefore* has the value 150 min. The switch-on time is brought forward by only 90 minutes to 0:00.



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_OUTPUT

```
bOutput              : BOOL;
bEdgeOn              : BOOL;
bEdgeOff             : BOOL;
tNextOn              : TIME;
tNextOff             : TIME;
bInvalidParameter    : BOOL;
```

bOutput: Is TRUE if one of the 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 program. Maximum value 10080 minutes (1 week).

tNextOff: Time until the next switch-off of the program. Maximum forecast until midnight (1440 min).

bInvalidParameter: An error occurred during the plausibility check. It is deleted again by *bReset*.

VAR_IN_OUT

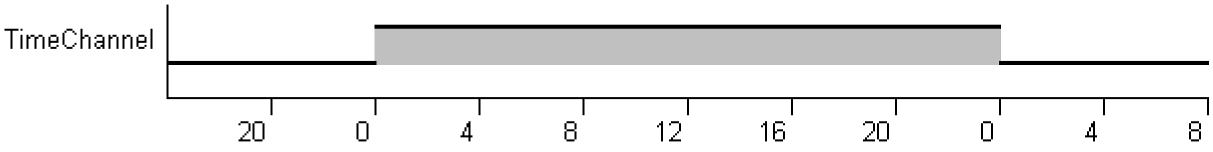
```
arrTimeChannel     : ARRAY[1..7] OF ST_HVACTimeChannel;
```

arrTimeChannel: The variable *arrTimeChannel* is an array with 7 declarations of the structure variable *ST_HVACTimeChannel (see* ST_HVACTimeChannel [▶ 499]*)*. It is possible to generate 1 switch-on time per day of the week with the 7 channels.

Special notes regarding time input

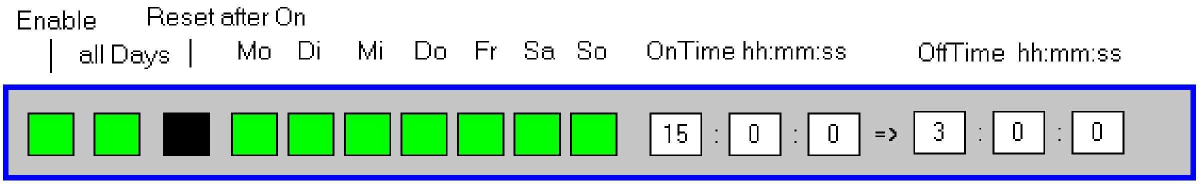
The following applies:

- 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.

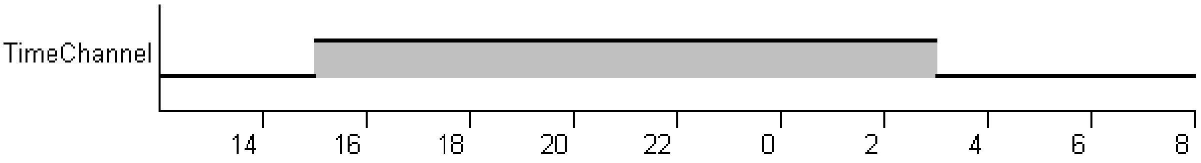


The following applies:

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.



This setting produces:



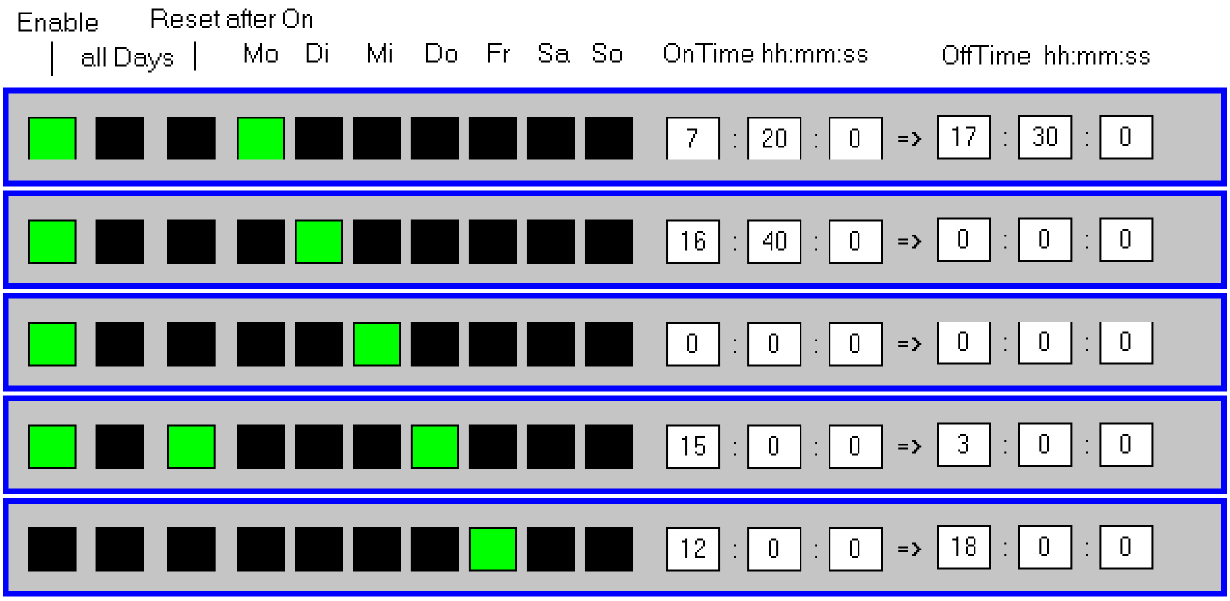
An array of the type *ST_HVACTimeChannel* (see ST_HVACTimeChannel [▶ 499]) must be generated for each instance of the function block *FB_HVACScheduler7ch.* The size of the array must be 7, since the function block processes a total of 7 weekly switch cycles and expects these as a reference (VAR_IN_OUT).

Example

```
VAR_GLOBAL
    arrTimeChannel : ARRAY[1..7] OF ST_HVACTimeChannel;
END_VAR
```

Within the weekly planner each channel is enabled via the variable *bEnable* (On/Off). If *bEnable* = FALSE, then the 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 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	all Days	Reset after On	Mo	Di	Mi	Do	Fr	Sa	So	OnTime hh:mm:ss		OffTime hh:mm:ss
🟩	■	■	🟩	■	■	■	■	■	■	7 : 20 : 0	=>	17 : 30 : 0
🟩	■	■	■	🟩	■	■	■	■	■	16 : 40 : 0	=>	0 : 0 : 0
🟩	■	■	■	■	🟩	■	■	■	■	0 : 0 : 0	=>	0 : 0 : 0
🟩	■	🟩	■	■	■	🟩	■	■	■	15 : 0 : 0	=>	3 : 0 : 0
■	■	■	■	■	■	■	🟩	■	■	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!)

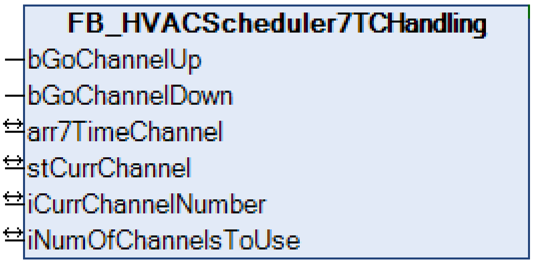
Thursday switched on from 15:00 until 03:00 Friday morning, non-recurring!

Friday channel not activated, always switched off!

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.8.3 FB_HVACScheduler7TCHandling



This function block can be used to select and modify an individual line from the data array of a weekly timer. In this way it is easily possible to edit the data sets with a visualization that only offers a low graphic resolution, without having to show all the data of the array.
The data set from *arr7TimeChannel* with index *iCurrChannelNumber* is copied to 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: With a positive edge the focus is placed on the next higher index in the data array *arr7TimeChannel*.

bGoChannelDown: With a positive edge the focus is placed on 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: Array of the data sets of the timer *FB_HVACScheduler7ch* (see ST_HVACTimeChannel [▸ 499]).

stCurrChannel: Contains the contents of the data set from the array *arr7TimeChannel*, which is selected via the variable *iCurrChannelNumber* (see ST_HVACTimeChannel [▸ 499]).

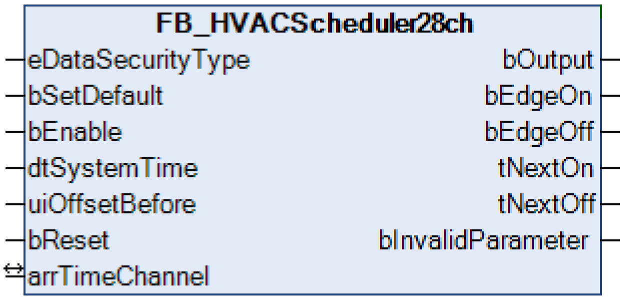
iCurrChannelNumber: Selection index for the array *arr7TimeChannel*.

iNumOfChannelsToUse: Maximum (7) permissible index for *iCurrChannelNumber* in case not all channels are needed.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.8.4 FB_HVACScheduler28ch



This function block is a weekly scheduler with 28 channels. Any desired combination of days of the week can be assigned to any channel. The 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 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 channels arrTimeChannel remain active.

VAR_INPUT

```
eDataSecurityType    : E_HVACDataSecurityType;
bSetDefault          : BOOL;
bEnable              : BOOL;
dtSystemTime         : DT;
uiOffsetBefore       : UINT;
bReset               : BOOL;
```

eDataSecurityType: If eDataSecurityType [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 variable *bEnable*. If *bEnable* = FALSE there is no switching at the output *bOutput*.

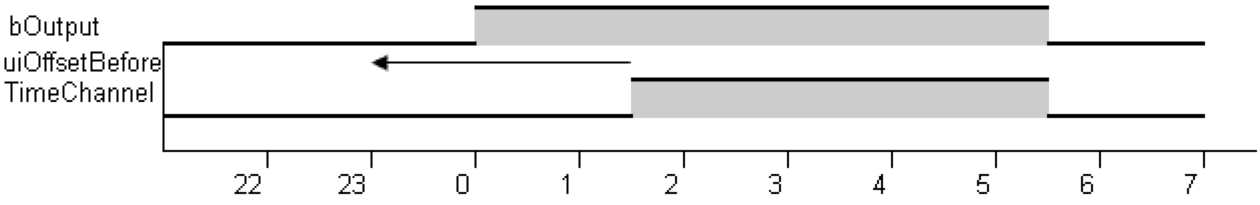
dtSystemTime: The variable *dtSystemTime* transfers the computer system time to the timer program.

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 *bInvalidParameter*.

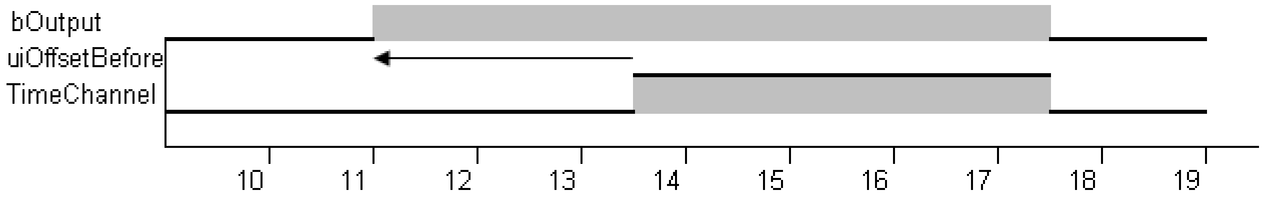
Example 1

Switch-on period from 1:30 to 5:30; *uiOffsetBefor* has the value 150 min. The switch-on time is brought forward by only 90 minutes to 0:00.



Example 2

Switch-on period from 13:30 to 17:30; *uiOffsetBefor* has the value 150 min. The switch-on time is brought forward by 150 minutes to 11:00.



VAR_OUTPUT

```
bOutput            : BOOL;
bEdgeOn            : BOOL;
bEdgeOff           : BOOL;
tNextOn            : TIME;
tNextOff           : TIME;
bInvalidParameter : BOOL;
```

bOutput: Is TRUE if one of the 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 program. Maximum value 10080 minutes (1 week).

tNextOff: Time until the next switch-off of the program. Maximum forecast until midnight. Maximum value 1440 minutes.

bInvalidParameter: An error occurred during the plausibility check. It is deleted again by *bReset*.

VAR_IN_OUT

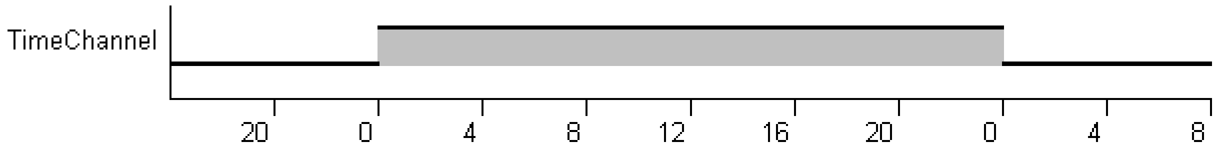
```
arrTimeChannel    : ARRAY[1..28] OF ST_HVACTimeChannel;
```

arrTimeChannel: The variable *arrTimeChannel* is an array with 28 declarations of the structure variable *ST_HVACTimeChannel (see* ST_HVACTimeChannel [▶ 499]). It is possible to generate up to 3 switch-on times per day of the week with the 28 channels.

Special notes regarding time input

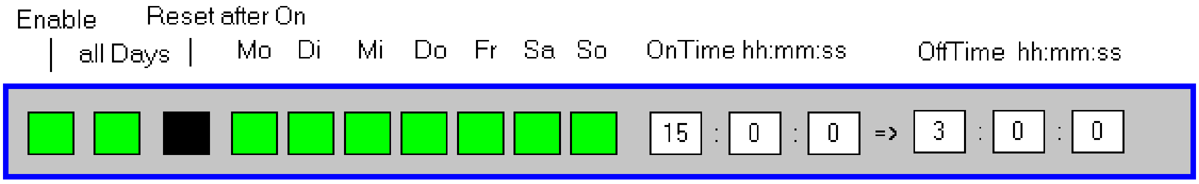
The following applies:

- 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.



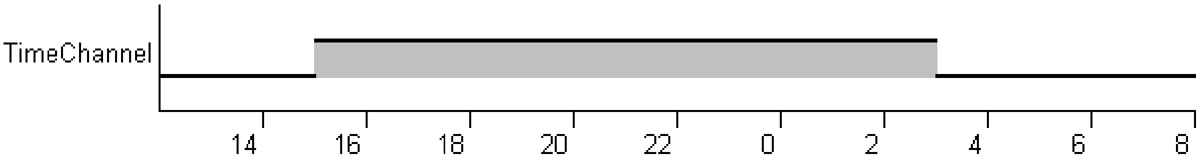
The following applies:

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	all Days	Reset after On	Mo	Di	Mi	Do	Fr	Sa	So	OnTime hh:mm:ss	OffTime hh:mm:ss
🟩	🟩	■	🟩	🟩	🟩	🟩	🟩	🟩	🟩	15 : 0 : 0 =>	3 : 0 : 0

This setting produces:



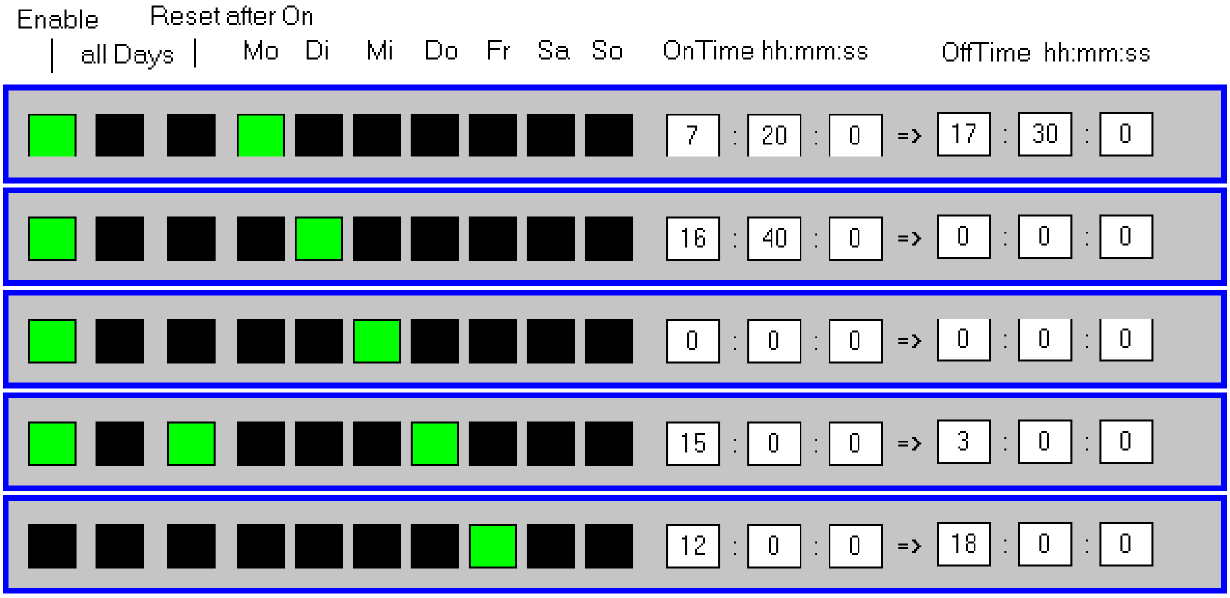
For each instance of the function block *FB_HVACScheduler28ch* it is necessary to create an array of type *ST_HVACTimeChannel*. The size of the array must be 28, since the function block processes a total of 28 weekly cycles and expects these as a reference (VAR_IN_OUT).

Example

```
VAR_GLOBAL
    arrTimeChannel : ARRAY[1..28] OF ST_HVACTimeChannel;
END_VAR
```

Within the weekly planner each time switch channel is enabled via the variable *bEnable* (On/Off). If *bEnable* = FALSE, then the time 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 time 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	all Days	Reset after On	Mo	Di	Mi	Do	Fr	Sa	So	OnTime hh:mm:ss				OffTime hh:mm:ss		
🟩	■	■	🟩	■	■	■	■	■	■	7	20	0	=>	17	30	0
🟩	■	■	■	🟩	■	■	■	■	■	16	40	0	=>	0	0	0
🟩	■	■	■	■	🟩	■	■	■	■	0	0	0	=>	0	0	0
🟩	■	🟩	■	■	■	🟩	■	■	■	15	0	0	=>	3	0	0
■	■	■	■	■	■	■	🟩	■	■	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!)

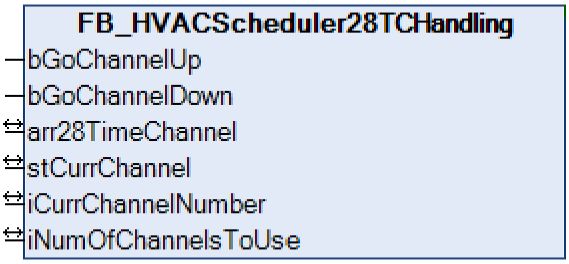
Thursday switched on from 15:00 until 03:00 Friday morning, non-recurring!

Friday Time switch channel not activated, always switched off!

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.8.5 FB_HVACScheduler28TCHandling



This function block can be used to select and modify an individual line from the data array of a weekly timer. In this way it is easily possible to edit the data sets with a visualization that only offers a low graphic resolution, without having to show all the data of the array.
The data set from *arr7TimeChannel* with index *iCurrChannelNumber* is copied to 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: With a positive edge the focus is placed on the next higher index in the data array *arr28TimeChannel*.

bGoChannelDown: With a positive edge the focus is placed on 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: Array of the data sets (28) of the timer *FB_HVACScheduler28CH*.

stCurrChannel: Contains the contents of the data set from the array *arr28TimeChannel*, which is selected via the variable *iCurrChannelNumber* (see ST_HVACTimeChannel [▶ 499]).

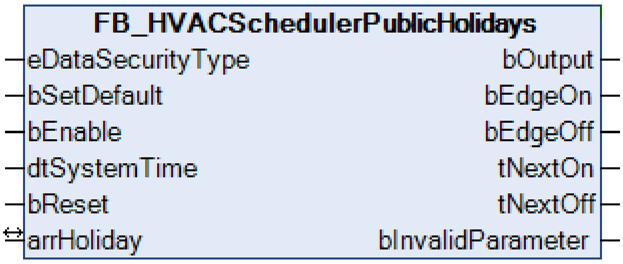
iCurrChannelNumber: Selection index for the array *arr28TimeChannel*.

iNumOfChannelsToUse: Maximum permissible index for *iCurrChannelNumber, if not all channels are used.*

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.8.6 FB_HVACSchedulerPublicHolidays



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 switching 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 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 [▶ 474], only daily switching is possible here.

If the input variable bEnable = FALSE, then the output bOutput = FALSE. The channels arrHoliday remain active.

VAR_INPUT

```
eDataSecurityType     : E_HVACDataSecurityType;
bSetDefault           : BOOL;
bEnable               : BOOL;
dtSystemTime          : DT;
bReset                : BOOL;
```

eDataSecurityType: If eDataSecurityType [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 variable *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_OUTPUT

```
bOutput             : BOOL;
bEdgeOn             : BOOL;
bEdgeOff            : BOOL;
tNextOn             : TIME;
tNextOff            : TIME;
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 (max. 50 days).

tNextOff: Time until the next switch-off of the timer program (max. 50 days).

bInvalidParameter: An error occurred during the plausibility check. It is deleted again by *bReset*.

VAR_IN_OUT

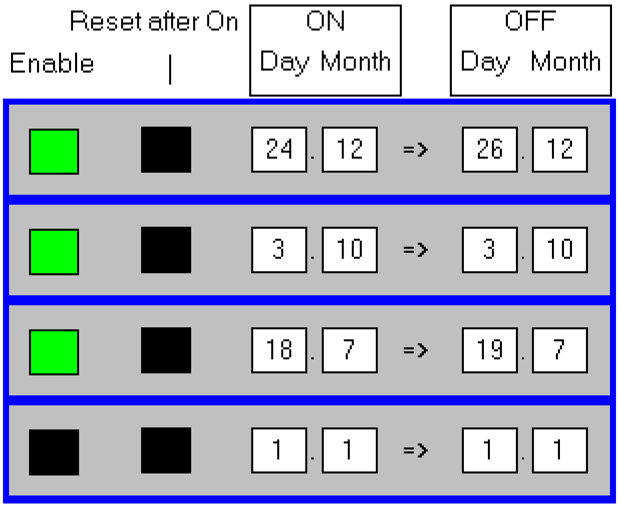
```
arrHoliday    : ARRAY[1..20] OF ST_HVACHoliday;
```

arrHoliday: The variable *arrHoliday* is an array with 20 declarations of the structure variable *ST_HVACHoliday (see* ST_HVACHoliday [▶ 496]*)*.

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).

Each channel is enabled with the variable *bEnable* (On/Off). If *bEnable* = FALSE, then the 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



Enable	Reset after On	ON Day	ON Month		OFF Day	OFF Month
🟩	■	24	12	=>	26	12
🟩	■	3	10	=>	3	10
🟩	■	18	7	=>	19	7
■	■	1	1	=>	1	1

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

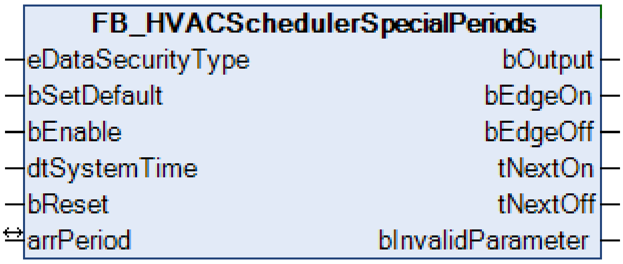
switched on from 18/07 00:00 until 19/07 23:59:59

1.1. until 1.1. channel not activated, always switched off!

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.8.7 FB_HVACSchedulerSpecialPeriods



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 switching 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 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 channels arrPeriod remain active.

VAR_INPUT

```
eDataSecurityType    : E_HVACDataSecurityType;
bSetDefault          : BOOL;
bEnable              : BOOL;
dtSystemTime         : DT;
bReset               : BOOL;
```

eDataSecurityType: If eDataSecurityType [▶ 491]:= *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 [▶ 480] 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/TF8000_TC3_HVAC/Resources/5090272523/.zip

If eDataSecurityType [▶ 491]:= *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 [▶ 491]:= 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 variable *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_OUTPUT

```
bOutput             : BOOL;
bEdgeOn             : BOOL;
bEdgeOff            : BOOL;
tNextOn             : TIME;
tNextOff            : TIME;
bInvalidParameter   : BOOL;
```

bOutput: Is TRUE if one of the 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 (max. 50 days).

tNextOff: Time until the next switch-off of the timer program (max. 50 days).

bInvalidParameter: An error occurred during the plausibility check. It is deleted again by *bReset*.

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* (see ST_HVACPeriod [▶ 497]).

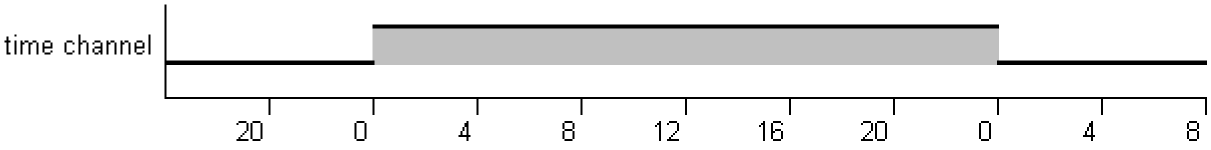
For each instance of the function block *FB_HVACSchedulerSpecialPeriods* it is necessary to create an array of type *ST_HVACPeriod*. The size of the array must be 20, since the function block processes a total of 20 time channels and expects these as a reference (VAR_IN_OUT).

Special notes regarding time input

The following applies:

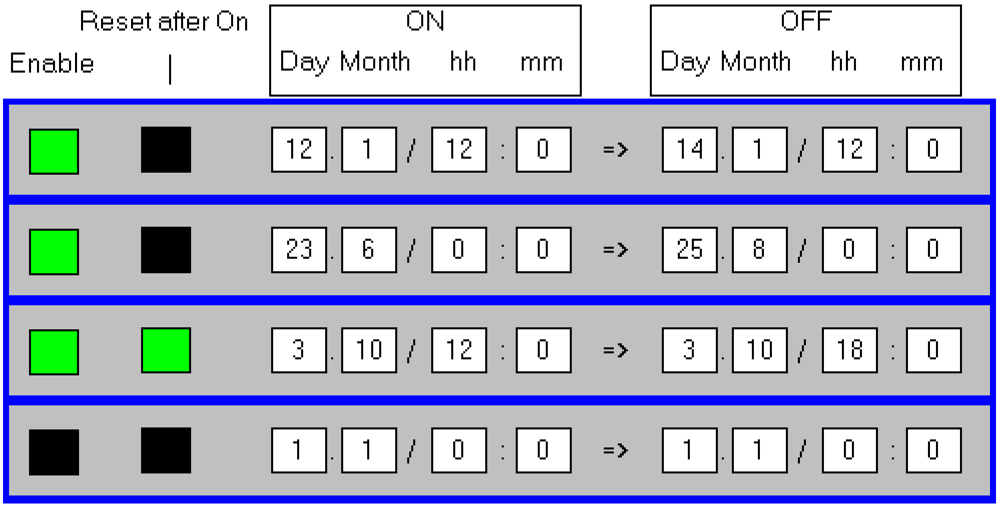
Switch-on time 00:00:00: Switch on begins at 0.00 on the selected day.

Switch-off time 00:00:00: Switch-off takes place at 23:59:59 on the selected day.



Each channel is enabled via the variable *bEnable* (On/Off). If *bEnable* = FALSE, then the 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
🟩	⬛	12	1	12	0	=>	14	1	12	0
🟩	⬛	23	6	0	0	=>	25	8	0	0
🟩	🟩	3	10	12	0	=>	3	10	18	0
⬛	⬛	1	1	0	0	=>	1	1	0	0

12/01 switched on from 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, one-shot!

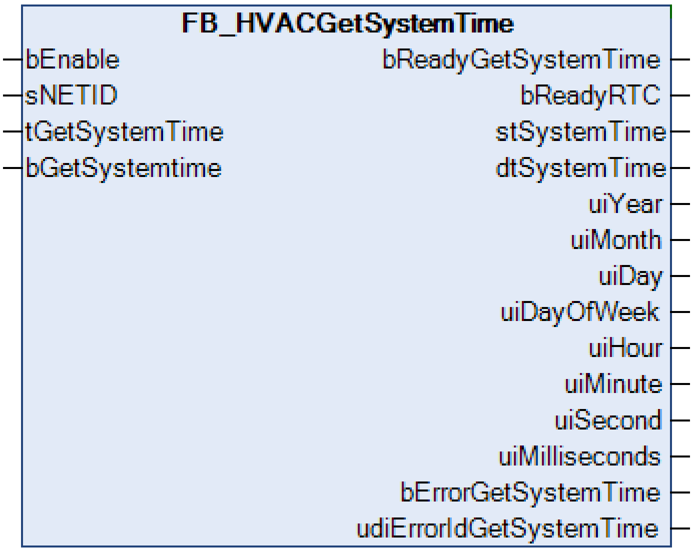
1.1. until 1.1. channel not activated, always switched off!

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.9 HVAC System

5.1.9.1 FB_HVACGetSystemTime



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.

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. For further information please refer to the Beckhoff Information System under: Beckhoff Information System > Industrial PC > Operating systems > CE > SNTP: Simple Network Time Protocol

VAR_INPUT

```
bEnable          : BOOL;
sNETID           : T_AmsNetId;
tGetSystemTime   : TIME;
bGetSystemTime   : BOOL;
```

bEnable: Enables the function block. If *bEnable* = TRUE, then the RTC clock is initialized with the NT system time.

sNETID: It is possible here to provide the AmsNetId of the TwinCAT computer whose NT system time is to be found. 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;

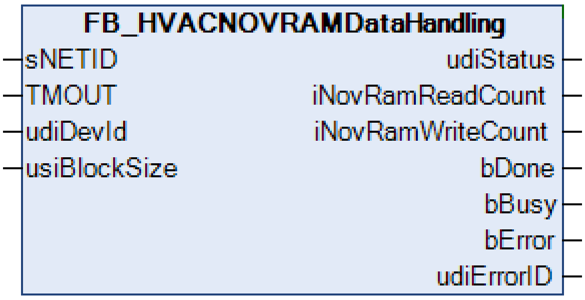
bErrorGetSystemTime: This output is set in the event that an ADS error occurs when transmitting the command. The output indicates with a TRUE that an error has occurred during initialization or synchronization with the NT system time. The function block continues to attempt to initialize or synchronize 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 (see ADS Return Codes) if a *bErrorGetSystemTime* output is set

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.9.2 FB_HVACNOVRAMDataHandling



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 is as follows if everything was correct when starting:
udiStatus:= 1;
iNovRamReadCount:= 1;
iNovRamWriteCount:= 0;
bDone:= TRUE;

In the case of ARM-based runtime systems, please make sure that the memory addresses of the PLC variables are divisible by 4!

The globally declared variables *g_dwHVACVarConfigStart* and *g_dwHVACVarConfigEnd* are located with 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.

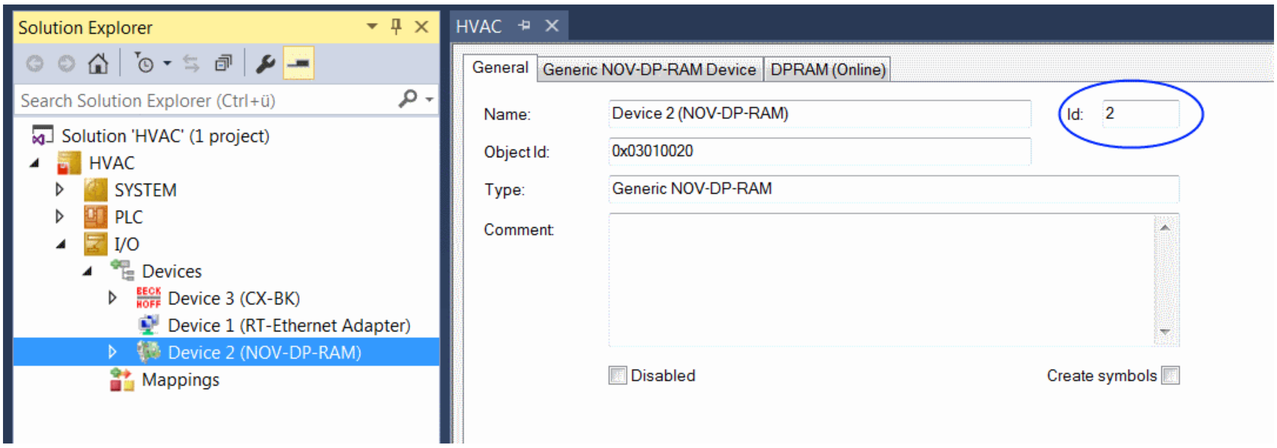
VAR_INPUT

```
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.


Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'HVAC' (1 project)
HVAC
SYSTEM
PLC
I/O
Devices
Device 3 (CX-BK)
Device 1 (RT-Ethernet Adapter)
Device 2 (NOV-DP-RAM)
Mappings
HVAC
General
Generic NOV-DP-RAM Device
DPRAM (Online)

Name: Device 2 (NOV-DP-RAM)
Id: 2
Object Id: 0x03010020
Type: Generic NOV-DP-RAM
Comment:
□ Disabled
Create symbols □

usiBlockSize: The block size to be accessed per read/write cycle is specified 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 state

= 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.

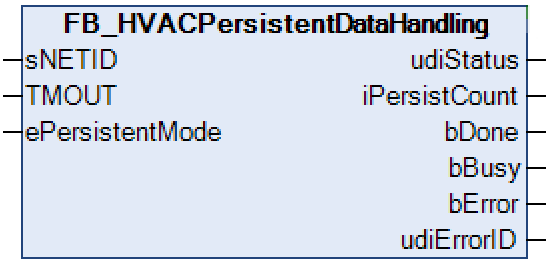
bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorID*.

udiErrorID: Contains the command-specific error code (see ADS Return Codes).

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.9.3 FB_HVACPersistentDataHandling



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 *Tc2_HVAC* are written to this file in case of a change. This file is saved in the ..\TwinCAT\3.1\Boot\PLC directory and is named as follows: Port_c.bootdata (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 parameter *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.

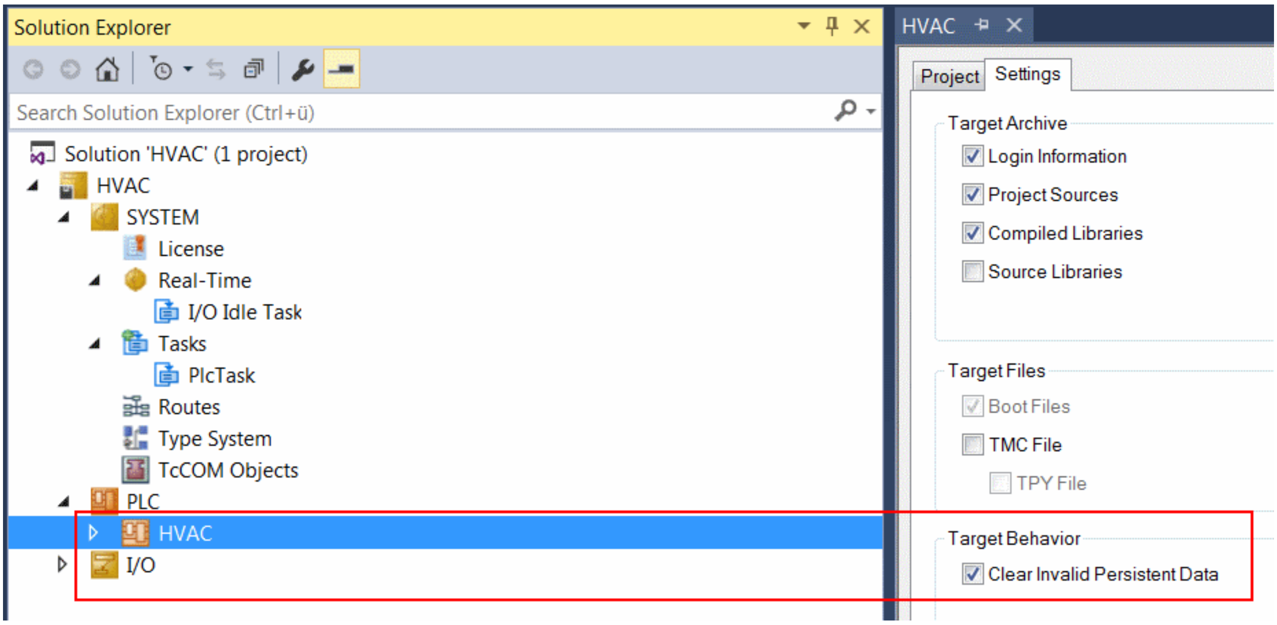
The behavior of the function block is as follows if everything was correct when starting:
udiStatus:= 1;
iPersistCount:= 1;
bDone:= FALSE;

If the result of the query of usiStatus is 1, this means that the persistent variables have been read from the memory. bDone only goes TRUE if the persistent variables have been written successfully.

Setting:

The following setting must be made in TwinCAT:

The option **Delete invalid persistent data** must be deactivated under PLC Configuration on the PLC Settings tab!!



Target Archive
☑ Login Information
☑ Project Sources
☑ Compiled Libraries
□ Source Libraries

Target Files
☑ Boot Files
□ TMC File
□ TPY File

Target Behavior
☑ Clear Invalid Persistent Data

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. Also see E_PersistentMode (see Tc2_Utilities library).

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.

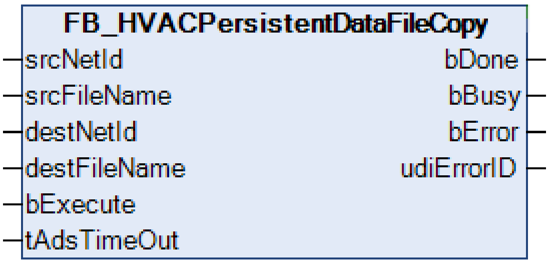
bError: This output is set to TRUE if an error occurs during the execution of a command.

udiErrorID: Contains the command-specific error code (see ADS Return Codes).

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.9.4 FB_HVACPersistentDataFileCopy



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. Network drives cannot be accessed. 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.

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 for the file to be opened. Note: 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 the 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 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 switched 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.

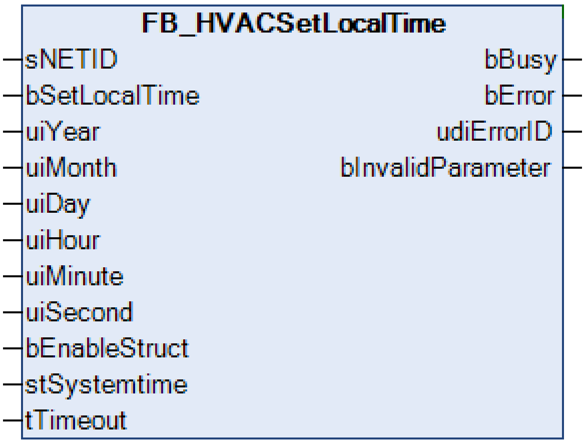
bError: This output is switched to TRUE as soon as an error occurs during the execution of a command.

udiErrorID: Contains the command-specific error code (see ADS Return Codes).

Requirements

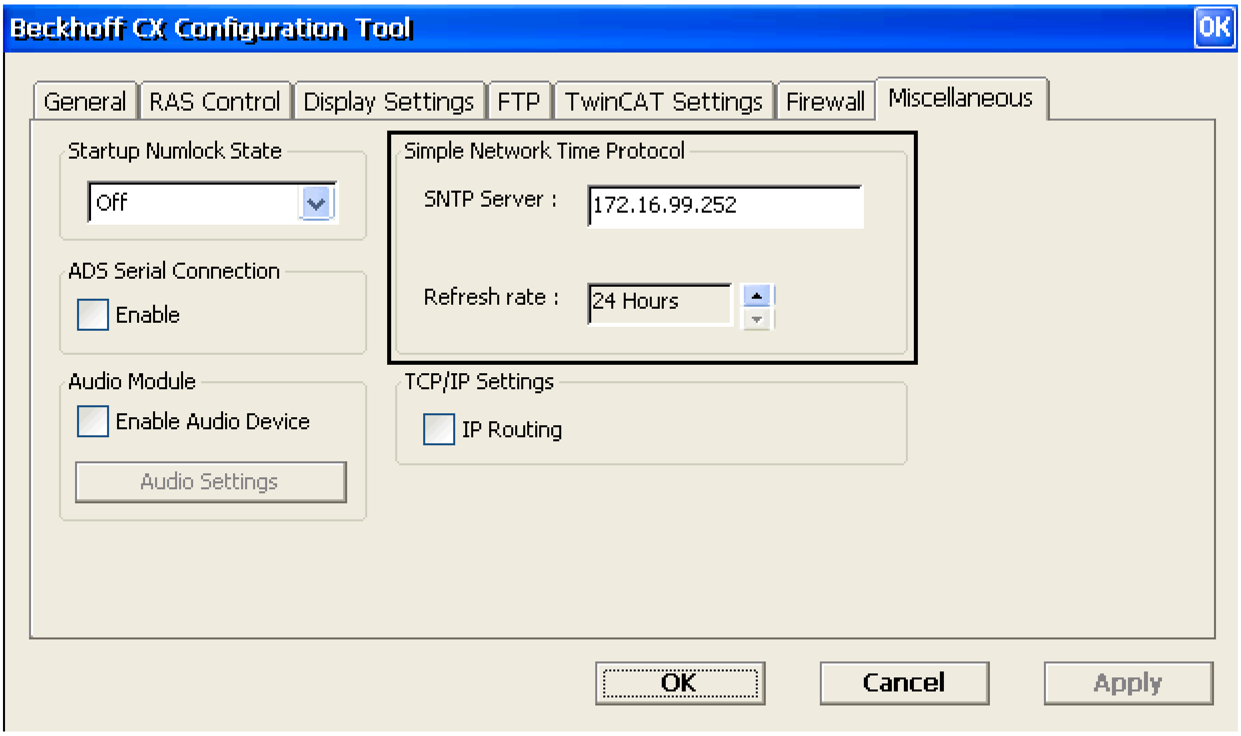
Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.9.5 FB_HVACSetLocalTime



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 via the structure *stSystemtime*, see *bEnableStruct.*
Internally, an instance of the function block *NT_SetLocalTime* from the TcUtilities library is called in the function block.

The local NT system time can also be synchronized with a reference time with the aid of the SNTP protocol. For further information please refer to the Beckhoff Information System under: Beckhoff Information System > Industrial PC > Operating systems > CE > SNTP: Simple Network Time Protocol



Startup Numlock State: Off

ADS Serial Connection: □ Enable

Audio Module: □ Enable Audio Device

Simple Network Time Protocol

SNTP Server : 172.16.99.252

Refresh rate : 24 Hours

TCP/IP Settings: □ IP Routing

VAR_INPUT

```
sNetId          : T_AmsNetId;
bSetLocalTime   : BOOL;
uiYear          : UINT;
uiMonth         : UINT;
uiDay           : UINT;
uiHour          : UINT;
uiMinute        : UINT;
uiSecond        : UINT;
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, *bInvalidParameter* = 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, *bInvalidParameter* = TRUE and the new local NT system time was not set.

uiDay: The 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, *bInvalidParameter* = 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, *bInvalidParameter* = 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, *bInvalidParameter* = 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, *bInvalidParameter* = 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 [s], 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: This output is set if an error occurs during the transfer of the NT system time. *bError* is reset by the activation of the function block via the input variable *bSetLocalTime*.

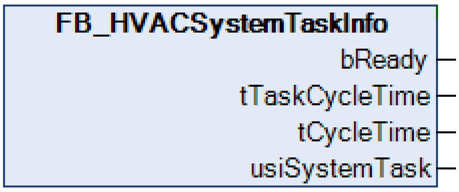
udiErrorId: Returns the ADS error code if *bError* is set (see ADS Return Codes)

bInvalidParameter: *bInvalidParameter* is TRUE if the ranges of the variables for the time of day and the date are not adhered to: *uiYear, uiMonth, uiDay, uiHour, uiMinute* and *uiSecond.* The same ranges apply to the time structure *stSystemtime* as to the input variables. If *bInvalidParameter* = TRUE, then the new local NT system time was not set. *bInvalidParameter* is reset by the activation of the function block via the input variable *bSetLocalTime.*

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.9.6 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*.

The *tTaskCycleTime* of the PLC program must not be longer than 100 ms, as otherwise the digital outputs will be released.
The reason for this is that the internal K-bus of the bus terminals runs synchronously to the PLC program, the bus will no longer be triggered early enough and the bus terminal watchdog is activated.

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.

Requirements

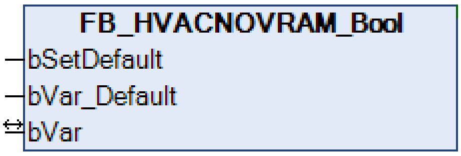
Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.10 BackupVar

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 NOVRAM function block
BOOL	FB_HVACNOVRAM_Bool [▶ 486]
BYTE	FB_HVACNOVRAM_Byte
DINT	FB_HVACNOVRAM_Dint
DWORD	FB_HVACNOVRAM_Dword
INT	FB_HVACNOVRAM_Int
LREAL	FB_HVACNOVRAM_Lreal
REAL	FB_HVACNOVRAM_Real
TIME	FB_HVACNOVRAM_Time
SINT	FB_HVACNOVRAM_Sint
UDINT	FB_HVACNOVRAM_Udint
UINT	FB_HVACNOVRAM_Uint
USINT	FB_HVACNOVRAM_Usint
WORD	FB_HVACNOVRAM_Word

5.1.10.1 FB_HVACNOVRAM_XX



The variables of various types (BOOL, BYTE, DINT, DWORD, INT, LREAL, REAL, SINT, TIME, UDINT, UINT, USINT, WORD) are saved fail-safe in the NOVRAM with the aid of the function blocks *FB_HVACNOVRAM_XX*.
By way of example the function block *FB_HVACNOVRAM_BOOL* is described here.

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

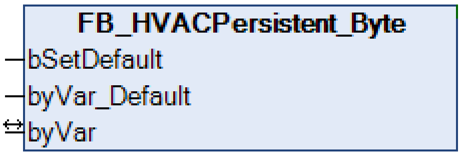
```
bVar  : BOOL;
```

bVar: Variable that is programmed fail-safe by the user.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.10.2 FB_HVACPersistent_XX



The variables of various types (BOOL, BYTE, DINT, DWORD, INT, LREAL, REAL, SINT, STRING, STRUCT, TIME, UDINT, UINT, USINT, WORD) are saved persistently with the aid of the function blocks *FB_HVACPersistent_XX*.
By way of example the function block *FB_HVACPersistent_BYTE* is described here.

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;
```

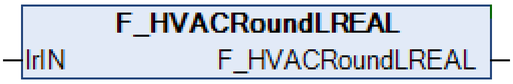
byVar: Variable that is programmed fail-safe by the user.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.11 Functions

5.1.11.1 F_RoundLREAL



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.

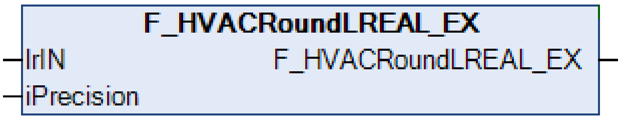
VAR_INPUT

```
lrIN                 : LREAL;
```

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.1.11.2 F_RoundLREAL_EX



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

```
Name                 : Type
```

```
lrIN                 : LREAL;
iPrecision           : INT;
```

lrIN: Floating point number that is to be formatted.

- If lrIN < 0.1 AND lrIN >= 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.

Requirements

Development environment	required library	required function
TwinCAT 3.1 build 4022.16 or higher	Tc2_HVAC V3.3.1.0	TF8000 | TC3 HVAC V1.0.0.0

5.2 DUTs

5.2.1 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 the management level.

..**OP**:.**O**perator **P**anel; local control of the actuator from a local control panel.

5.2.2 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.

5.2.3 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.

5.2.4 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**: **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.

5.2.5 E_HVACAirConditioning2SpeedMode

```
TYPE E_HVACAirConditioning2SpeedMode:
(
    eHVACAirConditioning2SpeedMode_Off                    := 0,
    eHVACAirConditioning2SpeedMode_Heating                := 1,
    eHVACAirConditioning2SpeedMode_Cooling                := 2,
    eHVACAirConditioning2SpeedMode_HeatingAndCooling      := 3
);
END_TYPE
```

5.2.6 E_HVACAnalogOutputMode

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

..**_BMS**: **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.

5.2.7 E_HVACAntiBlockingMode

```
TYPE E_HVACAntiBlockingMode:
(
    eHVACAntiBlockingMode_Off       := 0,
    eHVACAntiBlockingMode_Downtime  := 1,
    eHVACAntiBlockingMode_Weekly    := 2
);
END_TYPE
```

5.2.8 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
```

5.2.9 E_HVACCtrlMode

```
TYPE E_HVACCtrlMode :
(
    eHVACCtrlMode_Auto     := 0,
    eHVACCtrlMode_Manual   := 1
);
END_TYPE
```

5.2.10 E_HVACConvectionMode

```
TYPE E_HVACConvectionMode :
(
    eHVACConvectionMode_WithFan        := 0,
    eHVACConvectionMode_WithoutFan     := 1
);
END_TYPE
```

5.2.11 E_HVACDataSecurityType

```
TYPE E_HVACDataSecurityType:
(
    eHVACDataSecurityType_Persistent      := 0,
    eHVACDataSecurityType_Idle            := 1
);
END_TYPE
```

5.2.12 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,
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_rY_Max                    := 37,
eHVACErrorCodes_InvalidParam_rY_Min                    := 38,
eHVACErrorCodes_InvalidParam_iAggregate                := 39,
eHVACErrorCodes_InvalidParam_udiSecDelayUp             := 40,
eHVACErrorCodes_InvalidParam_udiSecDelayDown           := 41,
eHVACErrorCodes_InvalidParam_iNumberOfValues           := 42,
eHVACErrorCodes_Error_LEN_Int                          := 43,
eHVACErrorCodes_Error_LEN_Enum                         := 44
);
END_TYPE
```

5.2.13 E_HVACExternalMode

```
TYPE E_HVACExternalMode :
(
    eHVACExternalMode_Off                 := 0,
    eHVACExternalMode_On                  := 1,
    eHVACExternalMode_ShiftAbsolute      := 2
);
END_TYPE
```

5.2.14 E_HVACExternalRequestMode

```
TYPE E_HVACExternalRequestMode:
(
    eHVACExternalRequestMode_ButtonOn_Off          := 0,
    eHVACExternalRequestMode_ButtonOffDelay        := 1,
    eHVACExternalRequestMode_SwitchOn_Off          := 2
);
END_TYPE
```

5.2.15 E_HVACPlantMode

```
TYPE E_HVACPlantMode:
(
    eHVACPlantMode_TimeSchedulingOnly                      := 0,
    eHVACPlantMode_ExternalRequestOnly                     := 1,
    eHVACPlantMode_TimeScheduling_And_ExternalRequest      := 2,
    eHVACPlantMode_TimeScheduling_Or_ExternalRequest       := 3
);
END_TYPE
```

5.2.16 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
```

5.2.17 E_HVACReferencingMode

```
TYPE E_HVACReferencingMode :
(
    eHVACReferencingMode_Emulation          := 0,
    eHVACReferencingMode_AnalogFeedback     := 1
);
END_TYPE
```

5.2.18 E_HVACRegOutsideTemp

```
TYPE E_HVACRegOutsideTemp:
(
    eHVACReqOutsideTemp_NoRequest          := 0,
    eHVACReqOutsideTemp_OTLowerLimit       := 1,
    eHVACReqOutsideTemp_OTHigherLimit      := 2
);
END_TYPE
```

5.2.19 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*

5.2.20 E_HVACRegValve

```
TYPE E_HVACRegValve:
(
    eHVACReqValve_NoRequest                := 0,
    eHVACReqValve_OrValvePosHigherLimit    := 1,
    eHVACReqValve_AndValvePosHigherLimit   := 2
);
END_TYPE
```

5.2.21 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
```

5.2.22 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
```

5.2.23 E_HVACSetpointHeatingMode

```
TYPE E_HVACSetpointHeatingMode :
(
    eHVACSetpointHeatingMode_Off                := 0,
    eHVACSetpointHeatingMode_OnOutsideTemp      := 1,
    eHVACSetpointHeatingMode_OnDate             := 2,
    eHVACSetpointHeatingMode_OnPermanent        := 3,
    eHVACSetpointHeatingMode_OnFreezeProtec     := 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 operating 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 feed temperature setpoint calculated by the heating characteristic curve or the room setpoint module is routed through to the output *iSetpoint*. The output *bHeating* goes to TRUE and, hence, 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 according to date. The heating circuit is enabled from the date *iOn_Day / iOn_Month* until *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 permanently switched on.

eHVACSetpointHeatingMode_OnFreezeProtec: The heating circuit is in frost protection mode.

eHVACSetpointHeatingMode_OnNight: The heating circuit is in night operating mode.

eHVACSetpointHeatingMode_OnDay: The heating circuit is in day operating mode.

5.2.24 E_HVACSetpointMode

```
TYPE E_HVACSetpointMode :
(
    eHVACSetpointMode_DIN                   := 0,
    eHVACSetpointMode_DINLimited            := 1,
    eHVACSetpointMode_ConstantValueBase     := 2
);
END_TYPE
```

5.2.25 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.

5.2.26 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
```

5.2.27 E_HVACTemperatureSensorMode

```
TYPE E_HVACTemperatureSensorMode :
(
  eHVACTemperatureSensorMode_ReplacementValue               := 0,
  eHVACTemperatureSensorMode_LastValue                      := 1,
  eHVACTemperatureSensorMode_AutoResetReplacementValue      := 2,
  eHVACTemperatureSensorMode_AutoResetLastValue             := 3
);
END_TYPE
```

5.2.28 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
```

5.2.29 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.

5.2.30 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

5.2.31 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: State indicating which step the function block is in.

udiSecRT_MinOn[0.. g_iNumOfCmdCtrl_8 **]:** 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*]*.

udiSecRT_MinOff[0.. g_iNumOfCmdCtrl_8 **]:** 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*]*.

5.2.32 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
```

5.2.33 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 elelment is activated.

udiSecDelayLow: Delay time after whose expiry the I-transfer elelment is activated.

5.2.34 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
```

5.2.35 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
```

Structure via which the individual points of the XY coordinates are given their valency.

rX[1..g_iMaxNoOfScale_nPoint]: Array containing the valency of the individual points on the X axis. The number of points specified depends on *iNumberOfPoint*.

The following must be observed: rX[1] must be greater than rX[2], rX[2] must be greater than rX[n] OR rX[1] must be smaller than rX[2], rX[2] must be smaller than rX[n].

rY[1..g_iMaxNoOfScale_nPoint]: Array containing the valency of the individual points on the Y axis. The number of points specified depends on *iNumberOfPoint*.

iNumberOfPoint: Number of individual points of the XY coordinates.

The following must be observed: iNumberOfPoint must not be smaller than 2 OR larger than g_iMaxNoOfScale_nPoint(60).

5.2.36 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
```

5.2.37 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
```

5.2.38 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 [▶ 135] 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 continuous units. The variable value must not be smaller than *g_rY_Min* and must not exceed *g_rY_Max* 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 [▶ 135] is stopped. This variable is output via the structure *stAggregateX*.

rY_Min: Parameter specification for continuous units. The variable value must not be smaller than g_rY_Min and must not exceed g_rY_Max 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 [▶ 135] is stopped. This variable is output via the structure *stAggregateX*.

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 the FB_HVACI_CtrlStep [▶ 112]. This variable is output via the structure *stI_Ctrl*.

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 the FB_HVACI_CtrlStep [▶ 112]. 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 [▶ 112]. 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 [▶ 112]. 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*.

5.2.39 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
```

5.2.40 ST_HVACTempChangeFunction

Stucture of the node points for the temperature change function in the modules FB_HVACOptimizedOn [▸ 423] and FB_HVACOptimizedOff [▸ 433].

```
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.

5.2.41 HVAC room functions

5.2.41.1 E_BARCtrlFct

```
TYPE E_BARCtrlFct:
(
     eBARCtrlFct_Off          := 0,
     eBARCtrlFct_Heating      := 1,
     eBARCtrlFct_Cooling      := 2
);
END_TYPE
```

eBARCtrlFct_Off: Function selection OFF acitve.

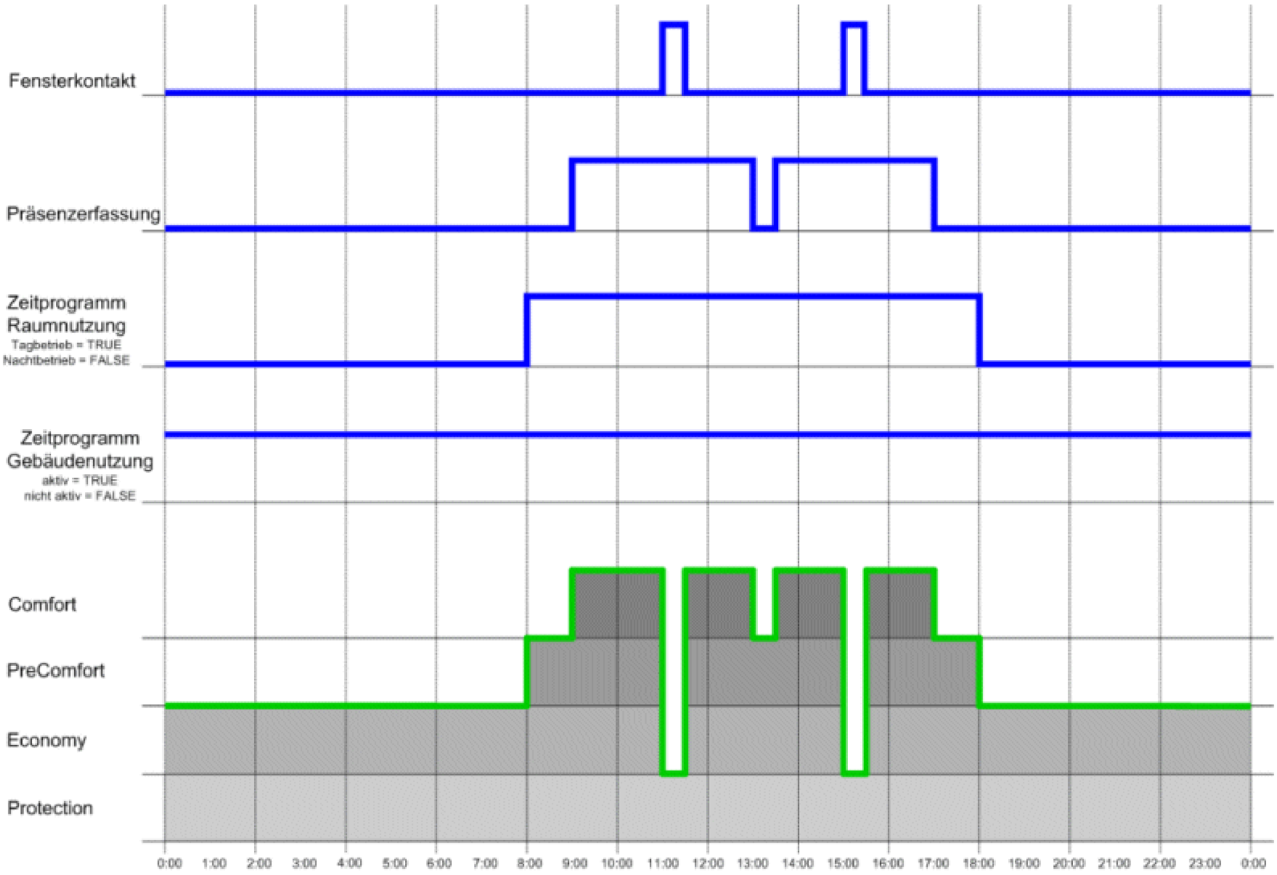
eBARCtrlFct_Heating: Function selection for heating operation active.

eBARCtrlFct_Cooling: Function selection for cooling operation active.

5.2.41.2 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, in this mode a switch to a higher enery level is possible if the upper energy level are already TRUE. In the mode the state of the inputs *bProtection*, *bEconomy*, *bPreComfort* and *bComfort* are evaluated for the selection of the energy level.



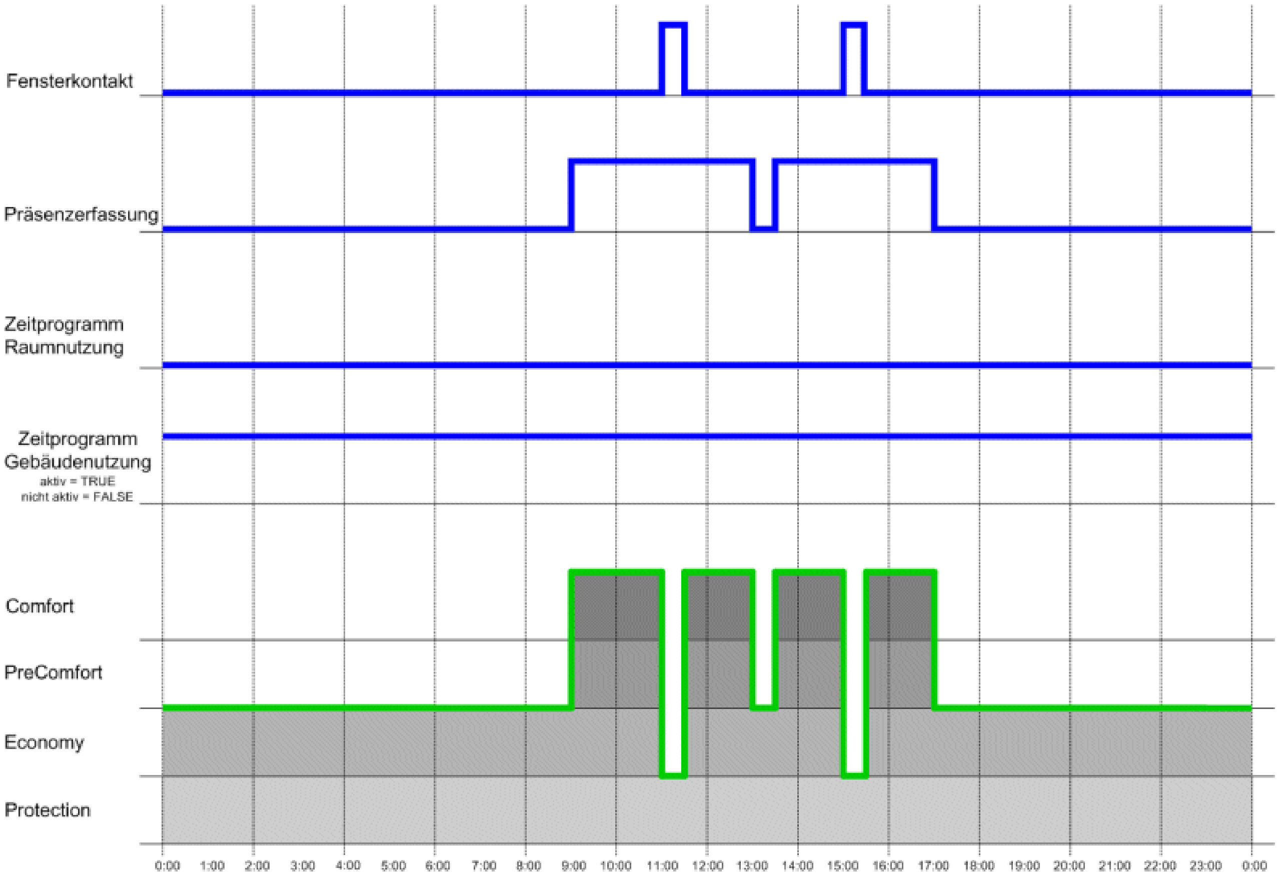
eBAREnergyLevel_Protection: In this mode which is manually operated by the building management system, the room/zone/range held in protection. This mode is only building protection mode.

eBAREnergyLevel_Economy: In this mode which is manually operated by the building management system, the room/zone/range held in Economy. This mode is set for longer periods of absence or at night.

eBAREnergyLevel_PreComfort: In this mode which is manually operated by the building management system, the room/zone/range held in PreComfort. This mode is set for shorter periods of absence

eBAREnergyLevel_Comfort: In this mode which is manually operated by the building management system, the room/zone/range held in the Comfort. This mode is set for periods of absence.

eBAREnergyLevel_AUTO_II: Automatic-mode-II means, in case of presence detection, this mode switches from energy level Protection, Economy or PreComfort directly to energy level Comfort.
See diagram



5.2.41.3 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 [▶ 306];

ePosModeTable: the height positioning takes place with the help of a table of 6 interpolation points, 4 of which are parameterisable. A blind position in relation to the position of the sun is then calculated from these points by linear interpolation. See FB_BARBlindPositionEntry [▶ 248] for further description.

ePosModeMaxIncidence: the positioning takes place with specification of the maximum desired incidence of light.

5.2.41.4 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.

5.2.41.5 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.

5.2.41.6 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 [▶ 306]. It is set to TRUE by the function block FB_BARBlindPositionEntry [▶ 248] if the data entered correspond to the validity criteria described.

5.2.41.7 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 as to whether this corner is in the shade: *bShaded*=TRUE: corner is in the shade.

5.2.41.8 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 the 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 [▸ 502].

bValid: plausibility of the data entered: *bValid*=TRUE: Data are plausible.

5.2.41.9 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;
  stBARSetpointRooom_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.

5.2.41.10 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 square.

lrMx .. lrMz: centre coordinates. Of importance only if the element is a ball.

lrRadius: radius of the ball. Of importance only if the element is a ball.

usiBeginMonth: beginning of the shading period (number of month).

usiEndMonth: end of the shading period (number of month).

eType: object type, see E_BARShadingObjectType [▸ 501].

bValid: plausibility of the data: bValid=TRUE: Data are plausible.

Remark about the shading period:

The monthly entries may not be 0 or greater than 12, otherwise all 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 (the following year).

5.2.41.11 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 louvre position in degrees.

bManUp: manual command: blind up.

bManDown: manual command: blind down.

bManualMode: TRUE: manual 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 [▸ 286]. The blind actuators FB_BARSunblindActuator [▸ 273] and FB_BARRollerBlind [▸ 287] ignore it.

5.2.41.12 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.

5.3 GVLs

5.3.1 Constants

Global constants

```
VAR_GLOBAL
  g_stHVACCycleTimeInterpretation    : ST_HVACCTRL_CYCLE_TIME_INTERPRETATION;
  g_dwHVACVarConfigStart        AT%M* : DWORD;
  g_dwHVACVarConfigEnd          AT%M* : DWORD;
  g_bHVACParamsChanged                : BOOL;
  g_bHVACBackupDataReadDone           : BOOL;
  g_bHVACNOVRAMDataReadDone           : BOOL;
  g_bHVACPersDataReadDone             : BOOL;
  g_bHVACPersParamsChanged            : BOOL;
  g_bHVACNOVRAMParamsChanged          : BOOL;
  g_bHVACNOVRAMUsed                   : BOOL;
  g_bHVACPersUsed                     : BOOL;
  g_bHVACPersWriteBusy                : BOOL;
END_VAR

VAR_GLOBAL CONSTANT
  g_iNumOfCmdCtrl_8                   : INT;
  rCloseToZero                        : REAL;
  lrCloseToZero                       : LREAL;
  uiMaxDataFileSize                  : UINT;
  g_udiMaxSec                        : UDINT;
END_VAR
```

g_bHVACParamsChanged: Triggers a backup.

g_bHVACBackupDataReadDone: Indicates that backup data have been read.

g_bHVACNOVRAMDataReadDone: Indicates that the NOVRAM has been read out.

g_bHVACPersDataReadDone: Indicates that the persistent data have been read out.

g_bHVACPersParamsChanged: Persistent data have changed, saving is taking place.

g_bHVACNOVRAMParamsChanged: Data in the NOVRAM have changed, saving is taking place.

g_bHVACNOVRAMUsed: NOVRAM has been written to.

g_bHVACPersUsed: Persistent data have been created.

g_bHVACPersWriteBusy: Working counter has been reset.

g_iNumOfCmdCtrl_8: (*FB_HVACCmdCtrl_8*) (preset to 8).

rCloseToZero: (Preset to 0.00000001).

lrCloseToZero: (Preset to 0.00000001).

uiMaxDataFileSize: Limitation of the data size to avoid overflows in FOR loops (preset to 65534).

g_udiMaxSec: (*FB_HVACI_CtrlStep / FB_HVACPowerRangeTable*) (preset to 4294967).

5.3.2 Parameter

Global parameters

```
VAR_GLOBAL CONSTANT
  iRowsPerFacade                : INT;
  iColumnsPerFacade             : INT;
  iShadingObjects               : INT;
  usiMaxSunblindScenes          : USINT;

  g_tHVACWriteBackupDataTime    : TIME;
  g_udiMaxNoOfBytesInStruct     : UDINT;
  g_udiMaxSizeOfString          : UDINT;
  g_iMaxNoOfScale_nPoint        : INT;
  g_iMaxNumberOfSteps           : INT;
  g_iMinNumberOfSteps           : INT;
  g_iMaxNumberOfProfiles        : INT;
  g_iMinNumberOfProfiles        : INT;
  g_iMaxNumberOfAggregates      : INT;
  g_iMinNumberOfAggregates      : INT;
  g_iAggregateMinNumberOfSteps  : INT;
  g_iAggregateMaxNumberOfSteps  : INT;
  g_iMinNumberOfSequences       : INT;
  g_iMaxNumberOfSequences       : INT;
END_VAR
```

iRowsPerFacade: Number of horizontally arranged shading elements on a floor (preset to 10).

iColumnsPerFacade: Number of vertically arranged shading elements on a facade (preset to 20).

iShadingObjects: Upper limit of the shading elements (preset to 20).

usiMaxSunblindScenes: Upper limit of the possible shading scenes (preset to 20).

g_tHVACWriteBackupDataTime: Time delay for writing the backup data (preset to 5 s).

g_udiMaxNoOfBytesInStruct: Maximum number of bytes in a structure (preset to 16).

g_udiMaxSizeOfString: Size in number of letters (as in the declaration of the string) + 1 byte for termination (preset to 256).

g_iMaxNoOfScale_nPoint: Maximum number of XY coordinates of the structure ST_HVACParameterScale_nPoint [▸ 497] (see FB_HVACScale_nPoint [▸ 87]) (preset to 60).

g_iMaxNumberOfSteps: (*FB_HVACI_CtrlStep/FB_HVACPowerRangeTable*) (preset to 32).

g_iMinNumberOfSteps: (*FB_HVACI_CtrlStep/FB_HVACPowerRangeTable*) (preset to 0).

g_iMaxNumberOfProfiles: (*FB_HVACPowerRangeTable*) (preset to 16).

g_iMinNumberOfProfiles: (*FB_HVACPowerRangeTable*) (preset to 1).

g_iMaxNumberOfAggregates: (*FB_HVACPowerRangeTable*) (preset to 6).

g_iMinNumberOfAggregates: (*FB_HVACPowerRangeTable*) (preset to 1).

g_iAggregateMinNumberOfSteps: (*FB_HVACPowerRangeTable*) (preset to 0).

g_iAggregateMaxNumberOfSteps: (*FB_HVACPowerRangeTable*) (preset to 6).

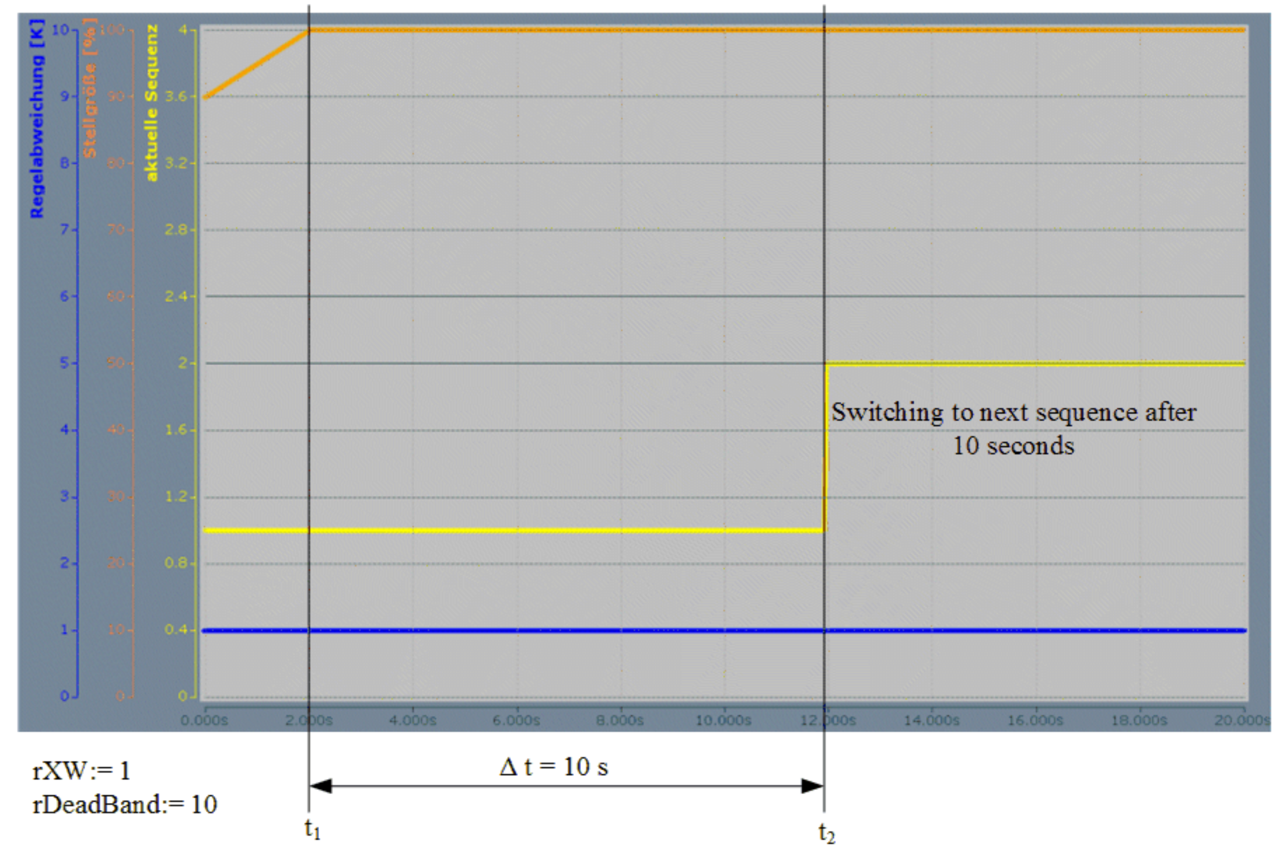
g_iMinNumberOfSequences: (*FB_HVAC2PointCtrlSequence*) (preset to 1).

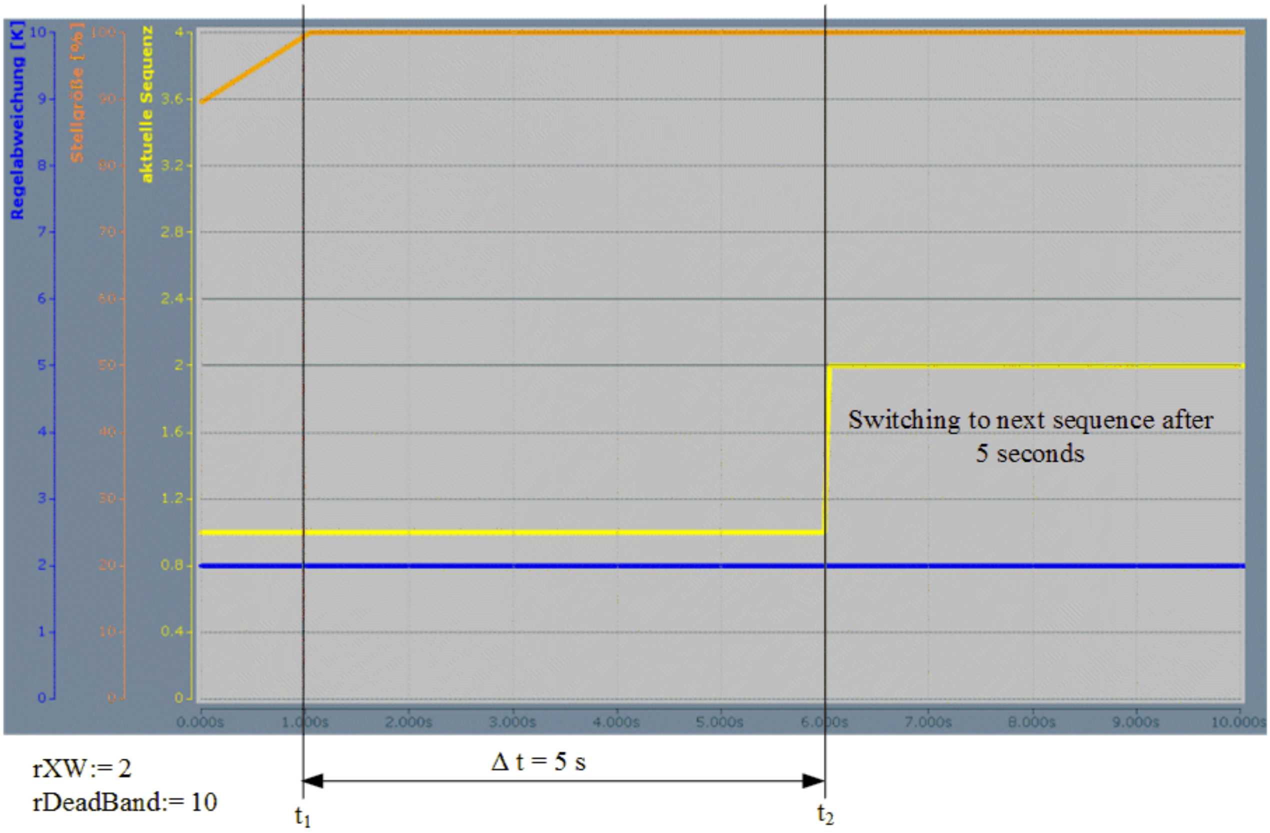
g_iMaxNumberOfSequences: (*FB_HVAC2PointCtrlSequence*) (preset to 16).

6 Appendix

6.1 Calculation of switching time when changing sequence

Example of the determination of the switching time from one sequence to the next.


Regelabweichung [K]
Stellgröße [%]
aktuelle Sequenz
10
9
8
7
6
5
4
3
2
1
0
100
90
80
70
60
50
40
30
20
10
0
4
3.6
3.2
2.8
2.4
2
1.6
1.2
0.8
0.4
0
0.000s
2.000s
4.000s
6.000s
8.000s
10.000s
12.000s
14.000s
16.000s
18.000s
20.000s
Switching to next sequence after 10 seconds
rXW:= 1
rDeadBand:= 10
Δ t = 10 s
t_1
t_2


Regelabweichung [K]
Stellgröße [%]
aktuelle Sequenz
10
9
8
7
6
5
4
3
2
1
0
100
90
80
70
60
50
40
30
20
10
0
4
3.6
3.2
2.8
2.4
2
1.6
1.2
0.8
0.4
0
0.000s
1.000s
2.000s
3.000s
4.000s
5.000s
6.000s
7.000s
8.000s
9.000s
10.000s
Switching to next sequence after 5 seconds
rXW:= 2
rDeadBand:= 10
Δ t = 5 s
t_1
t_2

6.2 Table of sequence controller operating modes

Operating modes

A further special feature of the sequence controller is its control by the Enum E_HVACSequenceCtrlMode [▶ 493]

Not only is the controller enabled by the Enum E_HVACSequenceCtrlMode [▶ 493], 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 [▶ 493] 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=ZuluftMax.Temp	Enable rY=ZuluftMin.Temp	Enable rY=ZuluftMin.Temp
FB_HVACPIDPreHeating Pre-heater	disabled 0 %	Enable	disabled 0 %	Enable 0 % - 100 %	disabled 0 %	disabled 0 %
FB_HVACPIDReHeating Re-heater	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 flaps	disabled 0 %	Enable	max. outside air rate 100 %	0 % air circulation only	0 % air circulation only	max. outside air rate 100 %

6.3 Support and Service

Beckhoff and their partners around the world offer comprehensive support and service, making available fast and competent assistance with all questions related to Beckhoff products and system solutions.

Download finder

Our download finder contains all the files that we offer you for downloading. You will find application reports, technical documentation, technical drawings, configuration files and much more.

The downloads are available in various formats.

Beckhoff's branch offices and representatives

Please contact your Beckhoff branch office or representative for local support and service on Beckhoff products!

The addresses of Beckhoff's branch offices and representatives round the world can be found on our internet page: www.beckhoff.com

You will also find further documentation for Beckhoff components there.

Beckhoff Support

Support offers you comprehensive technical assistance, helping you not only with the application of individual Beckhoff products, but also with other, wide-ranging services:

- support
- design, programming and commissioning of complex automation systems
- and extensive training program for Beckhoff system components

Hotline:	+49 5246 963-157
e-mail:	support@beckhoff.com

Beckhoff Service

The Beckhoff Service Center supports you in all matters of after-sales service:

- on-site service
- repair service
- spare parts service
- hotline service

Hotline:	+49 5246 963-460
e-mail:	service@beckhoff.com

Beckhoff Headquarters

Beckhoff Automation GmbH & Co. KG

Huelshorstweg 20
33415 Verl
Germany

Phone:	+49 5246 963-0
e-mail:	info@beckhoff.com
web:	www.beckhoff.com

6.4 ADS Return Codes

Grouping of error codes:
Global error codes: ADS Return Codes [▶ 511]... (0x9811_0000 ...)
Router error codes: ADS Return Codes [▶ 511]... (0x9811_0500 ...)
General ADS errors: ADS Return Codes [▶ 512]... (0x9811_0700 ...)
RTime error codes: ADS Return Codes [▶ 514]... (0x9811_1000 ...)

Global error codes

Hex	Dec	HRESULT	Name	Description
0x0	0	0x98110000	ERR_NOERROR	No error.
0x1	1	0x98110001	ERR_INTERNAL	Internal error.
0x2	2	0x98110002	ERR_NORTIME	No real time.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Allocation locked – memory error.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Mailbox full – the ADS message could not be sent. Reducing the number of ADS messages per cycle will help.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Wrong HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Target port not found – ADS server is not started or is not reachable.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Target computer not found – AMS route was not found.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unknown command ID.
0x9	9	0x98110009	ERR_BADTASKID	Invalid task ID.
0xA	10	0x9811000A	ERR_NOIO	No IO.
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	Unknown AMS command.
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 error.
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	Port not connected.
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	Invalid AMS length.
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	Invalid AMS Net ID.
0x10	16	0x98110010	ERR_LOWINSTLEVEL	Installation level is too low –TwinCAT 2 license error.
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	No debugging available.
0x12	18	0x98110012	ERR_PORTDISABLED	Port disabled – TwinCAT system service not started.
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	Port already connected.
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 error.
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync error.
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	No index map for AMS Sync available.
0x18	24	0x98110018	ERR_INVALIDAMSPORT	Invalid AMS port.
0x19	25	0x98110019	ERR_NOMEMORY	No memory.
0x1A	26	0x9811001A	ERR_TCPSEND	TCP send error.
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	Host unreachable.
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	Invalid AMS fragment.
0x1D	29	0x9811001D	ERR_TLSSEND	TLS send error – secure ADS connection failed.
0x1E	30	0x9811001E	ERR_ACCESSDENIED	Access denied – secure ADS access denied.

Router error codes

Hex	Dec	HRESULT	Name	Description
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	Locked memory cannot be allocated.
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	The router memory size could not be changed.
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	The mailbox has reached the maximum number of possible messages.
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	The Debug mailbox has reached the maximum number of possible messages.
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	The port type is unknown.
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	The router is not initialized.
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	The port number is already assigned.
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	The port is not registered.
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	The maximum number of ports has been reached.
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	The port is invalid.
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	The router is not active.
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	The mailbox has reached the maximum number for fragmented messages.
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	A fragment timeout has occurred.
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	The port is removed.

General ADS error codes

Hex	Dec	HRESULT	Name	Description
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	General device error.
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	Service is not supported by the server.
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	Invalid index group.
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	Invalid index offset.
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	Reading or writing not permitted.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parameter size not correct.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Invalid data values.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Device is not ready to operate.
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Device is busy.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Invalid operating system context. This can result from use of ADS blocks in different tasks. It may be possible to resolve this through multitasking synchronization in the PLC.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Insufficient memory.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Invalid parameter values.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Not found (files, ...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax error in file or command.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objects do not match.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Object already exists.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol not found.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Invalid symbol version. This can occur due to an online change. Create a new handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Device (server) is in invalid state.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode not supported.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification handle is invalid.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification client not registered.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	No further handle available.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Notification size too large.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Device not initialized.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Device has a timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface query failed.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Wrong interface requested.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class ID is invalid.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object ID is invalid.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Request pending.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Request is aborted.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal warning.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Invalid array index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol not active.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Access denied.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Missing license.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	License expired.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	License exceeded.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Invalid license.
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	License problem: System ID is invalid.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	License not limited in time.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Licensing problem: time in the future.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	License period too long.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception at system startup.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	License file read twice.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Invalid signature.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Invalid certificate.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public key not known from OEM.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	License not valid for this system ID.
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo license prohibited.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Invalid function ID.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Outside the valid range.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Invalid alignment.
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Invalid platform level.

Hex	Dec	HRESULT	Name	Description
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Context – forward to passive level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Context – forward to dispatch level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Context – forward to real time.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Client error.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Service contains an invalid parameter.
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling list is empty.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var connection already in use.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	The called ID is already in use.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout has occurred – the remote terminal is not responding in the specified ADS timeout. The route setting of the remote terminal may be configured incorrectly.
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Error in Win32 subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Invalid client timeout value.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port not open.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	No AMS address.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Internal error in Ads sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Hash table overflow.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Key not found in the table.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	No symbols in the cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Invalid response received.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port is locked.
0x756	1878	0x98110756	ADSERR_CLIENT_REQUESTCANCELLED	The request was cancelled.

RTime error codes

Hex	Dec	HRESULT	Name	Description
0x1000	4096	0x98111000	RTERR_INTERNAL	Internal error in the real-time system.
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	Timer value is not valid.
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	Task pointer has the invalid value 0 (zero).
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	Stack pointer has the invalid value 0 (zero).
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	The request task priority is already assigned.
0x1005	4101	0x98111005	RTERR_NOMORETCB	No free TCB (Task Control Block) available. The maximum number of TCBs is 64.
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	No free semaphores available. The maximum number of semaphores is 64.
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	No free space available in the queue. The maximum number of positions in the queue is 64.
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	An external synchronization interrupt is already applied.
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	No external sync interrupt applied.
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	Application of the external synchronization interrupt has failed.
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	Call of a service function in the wrong context
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel VT-x extension is not supported.
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel VT-x extension is not enabled in the BIOS.
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Missing function in Intel VT-x extension.
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Activation of Intel VT-x fails.

Specific positive HRESULT Return Codes:

HRESULT	Name	Description
0x0000_0000	S_OK	No error.
0x0000_0001	S_FALSE	No error. Example: successful processing, but with a negative or incomplete result.
0x0000_0203	S_PENDING	No error. Example: successful processing, but no result is available yet.
0x0000_0256	S_WATCHDOG_TIMEOUT	No error. Example: successful processing, but a timeout occurred.

TCP Winsock error codes

Hex	Dec	Name	Description
0x274C	10060	WSAETIMEDOUT	A connection timeout has occurred - error while establishing the connection, because the remote terminal did not respond properly after a certain period of time, or the established connection could not be maintained because the connected host did not respond.
0x274D	10061	WSAECONNREFUSED	Connection refused - no connection could be established because the target computer has explicitly rejected it. This error usually results from an attempt to connect to a service that is inactive on the external host, that is, a service for which no server application is running.
0x2751	10065	WSAEHOSTUNREACH	No route to host - a socket operation referred to an unavailable host.
More Winsock error codes: Win32 error codes			

More Information:
www.beckhoff.com/tf8000

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