# Table of contents

1 Foreword .......................................................................................................................... 5  
   1.1 Notes on the documentation .................................................................................. 5  
   1.2 Safety instructions ............................................................................................... 6  
   1.3 Notes on information security .............................................................................. 7  
2 Introduction ..................................................................................................................... 8  
   2.1 Target groups ....................................................................................................... 8  
   2.2 User requirement profile ..................................................................................... 8  
   2.3 General technical characteristics ....................................................................... 9  
      2.3.1 Integration into TwinCAT ............................................................................. 9  
      2.3.2 Hardware requirements ............................................................................. 9  
      2.3.3 Remanent data .......................................................................................... 9  
      2.3.4 Default values .......................................................................................... 9  
      2.3.5 Value range monitoring ............................................................................ 10  
3 General Information ...................................................................................................... 11  
4 Integration in TwinCAT ................................................................................................. 12  
   4.1 System requirements ........................................................................................... 12  
   4.2 Installation .......................................................................................................... 12  
   4.3 Licensing ............................................................................................................ 15  
5 Programming ................................................................................................................ 18  
   5.1 POU`s ................................................................................................................. 18  
      5.1.1 Overview .................................................................................................... 18  
      5.1.2 HVAC Actuators ....................................................................................... 25  
      5.1.3 HVAC Analog modules ............................................................................. 67  
      5.1.4 HVAC Controller ..................................................................................... 109  
      5.1.5 HVAC Room Functions ........................................................................... 184  
      5.1.6 HVAC Setpoint modules ........................................................................ 319  
      5.1.7 HVAC Special functions .......................................................................... 333  
      5.1.8 HVAC Time schedule ............................................................................... 459  
      5.1.9 HVAC System ............................................................................................ 476  
      5.1.10 BackupVar .............................................................................................. 484  
      5.1.11 Functions ................................................................................................. 486  
   5.2 DUTs ...................................................................................................................... 487  
      5.2.1 E_HVAC2PointActuatorMode .................................................................... 487  
      5.2.2 E_HVAC2PointCtrlMode .......................................................................... 487  
      5.2.3 E_HVAC3PointActuatorMode ................................................................ 487  
      5.2.4 E_HVACActuatorMode ............................................................................. 488  
      5.2.5 E_HVACAirConditioning2SpeedMode .................................................... 488  
      5.2.6 E_HVACAnalogOutputMode .................................................................. 488  
      5.2.7 E_HVACAntiBlockingMode .................................................................... 488  
      5.2.8 E_HVACBusTerminal_KL32xx ................................................................ 489  
      5.2.9 E_HVACCtrlMode ..................................................................................... 490  
      5.2.10 E_HVACCmConvMode .......................................................................... 490  
      5.2.11 E_HVACDataSecurityType .................................................................... 490
5.2.12 E_HVACErrorCodes ................................................................. 490
5.2.13 E_HVACExternalMode .......................................................... 491
5.2.14 E_HVACExternalRequestMode ........................................... 491
5.2.15 E_HVACPlantMode .............................................................. 491
5.2.16 E_HVACPowerMeasurementMode ..................................... 491
5.2.17 E_HVACReferencingMode .................................................. 491
5.2.18 E_HVACRegOutsideTemp ................................................... 492
5.2.19 E_HVACRegPump ............................................................... 492
5.2.20 E_HVACRegValve ............................................................... 492
5.2.21 E_HVACSensorType .......................................................... 492
5.2.22 E_HVACSequenceCtrlMode .............................................. 493
5.2.23 E_HVACSetpointHeatingMode .......................................... 493
5.2.24 E_HVACSetpointMode ....................................................... 494
5.2.25 E_HVACState ................................................................. 494
5.2.26 E_HVACTemperatureCurve ............................................. 494
5.2.27 E_HVACTemperatureSensorMode .................................... 494
5.2.28 ST_HVAC2PointCtrlSequence ........................................ 494
5.2.29 ST_HVACAggregate ......................................................... 495
5.2.30 ST_HVACCmdCtrl_8Param ............................................. 495
5.2.31 ST_HVACCmdCtrl_8State ................................................ 495
5.2.32 ST_HVACHoliday ............................................................. 496
5.2.33 ST_HVACICtrl ................................................................. 496
5.2.34 ST_HVACPeriod ............................................................. 496
5.2.35 ST_HVACParameterScale_nPoint .................................... 496
5.2.36 ST_HVACPowerMeasurement ......................................... 497
5.2.37 ST_HVACPowerMeasurementEx ..................................... 497
5.2.38 ST_HVACPowerRangeTable ........................................ 497
5.2.39 ST_HVACTimeChannel ................................................... 498
5.2.40 ST_HVACTempChangeFunction ..................................... 499
5.2.41 HVAC room functions ..................................................... 499
5.3 GVLs .................................................................. 504
5.3.1 Constants ................................................................. 504
5.3.2 Parameter ............................................................... 505
6 Appendix ................................................................ 507
6.1 Calculation of switching time when changing sequence ............ 507
6.2 Table of sequence controller operating modes ....................... 509
6.3 Support and Service ....................................................... 509
6.4 ADS Return Codes .......................................................... 511
1 Foreword

1.1 Notes on the documentation

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

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

Disclaimer

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

Trademarks

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

Patent Pending

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

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

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.
The reproduction, distribution and utilization of this document as well as the communication of its contents to others without express authorization are prohibited.
Offenders will be held liable for the payment of damages. All rights reserved in the event of the grant of a patent, utility model or design.
1.2 Safety instructions

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Description of symbols

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

<table>
<thead>
<tr>
<th>▶ DANGER</th>
<th>Serious risk of injury!</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>▶ WARNING</th>
<th>Risk of injury!</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>▶ CAUTION</th>
<th>Personal injuries!</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Damage to the environment or devices</td>
</tr>
<tr>
<td>Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.</td>
</tr>
</tbody>
</table>

Tip or pointer

This symbol indicates information that contributes to better understanding.
1.3 Notes on information security

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

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

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

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

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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>TwinCAT PLC function to be implemented</th>
<th>TwinCAT Standard PLC libraries to be implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>PC or CX</td>
<td>Tc2_HVAC</td>
<td>Tc2_Coupler; Tc2_IoFunctions; Tc2_Math; Tc2_Standard; Tc2_System; Tc2_Utilsities</td>
</tr>
</tbody>
</table>

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_HVACPersistentDataHandling. 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.

**NOTE**

**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 \texttt{bInvalidParameter}. The variable \texttt{bInvalidParameter} can be reset by a rising edge at the input variable \texttt{bReset}.
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

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

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.

I accept the terms in the license agreement

I do not accept the terms in the license agreement

InstallShield
3. Enter your user data.

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

6. Confirm the dialog with **Yes**.
7. Select Finish to exit the setup.

☞ 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](image)

   - 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").

   ![Manage Licenses](image)

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

   - In the tabular overview of licenses, the previously selected license is displayed with the status "missing".

   ![Order Information](image)
7. Click **7-Day Trial License**... to activate the 7-day trial license.

![License management interface](image)

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

![Security code input](image)

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_HVAC2PointActuator</td>
<td>Control of two-point dampers or valves</td>
</tr>
<tr>
<td>FB_HVAC3PointActuator</td>
<td>Control of three-point dampers or valves</td>
</tr>
<tr>
<td>FB_HVACCirculationPump</td>
<td>Control of pumps</td>
</tr>
<tr>
<td>FB_HVACCirculationPumpEx</td>
<td>Control of pumps; the control of the output bPump has been changed compared to FB_HVACCirculationPump</td>
</tr>
<tr>
<td>FB_HVACMotor1Speed</td>
<td>Control of single-speed drives</td>
</tr>
<tr>
<td>FB_HVACMotor2Speed</td>
<td>Control of two-speed drives</td>
</tr>
<tr>
<td>FB_HVACMotor3Speed</td>
<td>Control of three-speed drives</td>
</tr>
<tr>
<td>FB_HVACRedundancyCtrl</td>
<td>Control of up to eight actuators depending on the runtime; uses an internal timer for determining the runtime</td>
</tr>
<tr>
<td>FB_HVACRedundancyCtrlEx</td>
<td>Control of up to eight actuators depending on the runtime; the runtimes of the actuators must be created externally as hour values.</td>
</tr>
</tbody>
</table>

**HVAC analog modules**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_HVACAnalogInput</td>
<td>Acquisition of analog input signals</td>
</tr>
<tr>
<td>FB_HVACAnalogOutput</td>
<td>Control of analog actuators</td>
</tr>
<tr>
<td>FB_HVACAnalogOutputEx</td>
<td>Control of analog actuators with integrated scaling function</td>
</tr>
<tr>
<td>FB_HVACAnalogTo3Point</td>
<td>Conversion of an analog control signal into a three-point step signal</td>
</tr>
<tr>
<td>FB_HVACConfigureKL32xx</td>
<td>Parameterization of the connected sensor type on an input channel from the PLC</td>
</tr>
<tr>
<td>FB_HVACScale</td>
<td>Scaling function block</td>
</tr>
<tr>
<td>FB_HVACScale_nPoint</td>
<td>Simulation of characteristic curves in the PLC</td>
</tr>
<tr>
<td>FB_HVACScaleXX</td>
<td>Mapping of characteristic curves with 2, 4 or 7 interpolation points.</td>
</tr>
<tr>
<td>FB_HVACTemperatureCurve</td>
<td>Different temperature characteristic curves that can be mapped are stored inside the FB.</td>
</tr>
<tr>
<td>FB_HVACTemperatureSensor</td>
<td>Acquisition of temperature values in 1/10 °C, tailored to the KL320x Bus Terminals</td>
</tr>
<tr>
<td>FB_HVACTemperatureSensorEx</td>
<td>Acquisition of temperature values in 1/10°C, tailored to the KL320x Bus Terminals; without 2\textsuperscript{nd} order filter, but with a smoothing function</td>
</tr>
<tr>
<td>FB_HVACTemperatureSensorEx2</td>
<td>Acquisition of temperature values in 1/10°C or 1/100°C, tailored to the KL320x Bus Terminals; without 2\textsuperscript{nd} order filter, but with a smoothing function</td>
</tr>
</tbody>
</table>
## HVAC controller

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_HVAC2PointCtrl [109]</td>
<td>2-point controller</td>
</tr>
<tr>
<td>FB_HVAC1_CtrlStep [112]</td>
<td>I-transfer element for step control in conjunction with FB_HVACPowerRangeTable</td>
</tr>
<tr>
<td>FB_HVAC1_CtrlStepEx [119]</td>
<td>I-transfer element for step control in conjunction with FB_HVACPowerRangeTable</td>
</tr>
<tr>
<td>FB_HVACPIDCtrl [127]</td>
<td>PID controller</td>
</tr>
<tr>
<td>FB_HVACPIDCtrl_Ex [130]</td>
<td>PID controller (extended)</td>
</tr>
<tr>
<td>FB_HVACPowerRangeTable [135]</td>
<td>Power range table for the sequential control of power generators such as boilers or refrigerating machines</td>
</tr>
<tr>
<td>FB_HVAC2PointCtrlSequence [149]</td>
<td>2-point sequence controller</td>
</tr>
<tr>
<td>FB_HVACBasicSequenceCtrl [156]</td>
<td>general sequence controller</td>
</tr>
<tr>
<td>FB_HVACMasterSequenceCtrl [159]</td>
<td>Master controller in a VAC system</td>
</tr>
<tr>
<td>FB_HVACPIDCooling [162]</td>
<td>PID cooling controller</td>
</tr>
<tr>
<td>FB_HVACPIDDehumidify [165]</td>
<td>PID dehumidification controller</td>
</tr>
<tr>
<td>FB_HVACPIDEnergyRecovery [168]</td>
<td>PID heat recovery controller</td>
</tr>
<tr>
<td>FB_HVACPIDHumidify [171]</td>
<td>PID humidification controller</td>
</tr>
<tr>
<td>FB_HVACPIDMixedAir [175]</td>
<td>PID mixed air chamber controller</td>
</tr>
<tr>
<td>FB_HVACPIDPreHeating [178]</td>
<td>PID pre-heating controller</td>
</tr>
<tr>
<td>FB_HVACPIDReHeating [181]</td>
<td>PID reheating controller</td>
</tr>
</tbody>
</table>

## Air conditioning room function

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_BAREnergyLevel [184]</td>
<td>This function block is for the adaptation of the supply of energy for the use of the building.</td>
</tr>
<tr>
<td>FB_BARFanCoil [186]</td>
<td>This function block maps a 3-speed fan with the corresponding switching hysteresis.</td>
</tr>
<tr>
<td>FB_BARFctSelection [189]</td>
<td>This function block is for enabling room heating or room cooling.</td>
</tr>
<tr>
<td>FB_BARSetpointRoom [192]</td>
<td>This function block assigns a setpoint for cooling operation and another for heating operation to each of the four energy levels.</td>
</tr>
</tbody>
</table>

## Room function controller

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_BARPICtrl [197]</td>
<td>simple PI Controller</td>
</tr>
</tbody>
</table>
### Room functions; lighting

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_BARAutomaticLight</td>
<td>Occupancy-controlled automatic light with switch-off delay</td>
</tr>
<tr>
<td>FB_BARConstantLightControl</td>
<td>Constant light controller with continuous output of analog values</td>
</tr>
<tr>
<td>FB_BARDaylightControl</td>
<td>Daylight control without dimming procedures</td>
</tr>
<tr>
<td>FB_BARLightActuator</td>
<td>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.</td>
</tr>
<tr>
<td>FB_BARLightCircuit</td>
<td>Simple light switching circuit without dimming function</td>
</tr>
<tr>
<td>FB_BARLightCircuitDim</td>
<td>Simple light switching circuit with dimming function</td>
</tr>
<tr>
<td>FB_BARStairwellAutomatic</td>
<td>Stairwell lighting with preliminary warning sequence</td>
</tr>
<tr>
<td>FB_BARTwilightAutomatic</td>
<td>Twilight switching</td>
</tr>
</tbody>
</table>

### Sun protection room function

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_BARBlindPositionEntry</td>
<td>Sun protection function: Input of blind positions.</td>
</tr>
<tr>
<td>FB_BARDelayedHysteresis</td>
<td>External brightness check with delay and hysteresis</td>
</tr>
<tr>
<td>FB_BARFacadeElementEntry</td>
<td>Shading correction: Input of facade elements per function block.</td>
</tr>
<tr>
<td>FB_BARReadFacadeElementList</td>
<td>Shading correction: Input of facade elements via data list (csv).</td>
</tr>
<tr>
<td>FB_BARReadShadingObjectsList</td>
<td>Shading correction: Input of shading objects via data list (csv).</td>
</tr>
<tr>
<td>FB_BARShadingCorrection</td>
<td>Shading correction function block (FB_BarShadingCorrectionSouth applies to the southern hemisphere)</td>
</tr>
<tr>
<td>FB_BARShadingCorrectionSouth</td>
<td></td>
</tr>
<tr>
<td>FB_BARShadingObjectsEntry</td>
<td>Shading correction: Input of shading objects per function block.</td>
</tr>
<tr>
<td>FB_BARSunblindActuator</td>
<td>Blind actuator</td>
</tr>
<tr>
<td>FB_BARSunblindActuatorEx</td>
<td></td>
</tr>
<tr>
<td>FB_BARSunblindEvent</td>
<td>Output of a specified blind position and angle in percent</td>
</tr>
<tr>
<td>FB_BARSunblindPrioritySwitch</td>
<td>Priority controller</td>
</tr>
<tr>
<td>FB_BARRollerBlind</td>
<td>Roller shutter actuator</td>
</tr>
<tr>
<td>FB_BARSunblindScene</td>
<td>Manual operation with scene selection and programming</td>
</tr>
<tr>
<td>FB_BARSunblindSwitch</td>
<td>Manual operation</td>
</tr>
<tr>
<td>FB_BARSunblindThermoAutomatic</td>
<td>Thermo-automatic function</td>
</tr>
<tr>
<td>FB_BARSunblindTwilightAutomatic</td>
<td>Automatic twilight function</td>
</tr>
<tr>
<td>FB_BARSunblindWeatherProtection</td>
<td>Weather protection function</td>
</tr>
<tr>
<td>FB_BARSunProtectionEx</td>
<td>Sun protection function, see Overview of automatic sun protection (shading correction)</td>
</tr>
<tr>
<td>FB_BARWithinRangeAzimuth</td>
<td>Verification of valid sun position and sun direction range (azimuth angle)</td>
</tr>
<tr>
<td>FB_BARWithinRangeElevation</td>
<td>Verification of valid sun position and sun elevation range (elevation angle)</td>
</tr>
</tbody>
</table>
## HVAC Setpoint modules

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_HVACHeatingCurve [p.319]</td>
<td>Calculation the supply temperature depending on the outside temperature with four base</td>
</tr>
<tr>
<td>FB_HVACHeatingCurveEx [p.322]</td>
<td>Calculation the supply temperature depending on the outside temperature</td>
</tr>
<tr>
<td>FB_HVACOutsideTempDamped [p.325]</td>
<td>Calculation the damped outside temperature</td>
</tr>
<tr>
<td>FB_HVACSetpointHeating [p.326]</td>
<td>Control an heating circuit with different operating modes</td>
</tr>
<tr>
<td>FB_HVACSetpointRamp [p.328]</td>
<td>Moving setpoint ramp</td>
</tr>
<tr>
<td>FB_HVACSummerCompensation [p.330]</td>
<td>summer compensation</td>
</tr>
</tbody>
</table>
HVAC special functions
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_HVACAirConditioning2Speed [333]</td>
<td>Control of air handling units with two-speed fans</td>
</tr>
<tr>
<td>FB_HVACAlarm [337]</td>
<td>Alarm function block</td>
</tr>
<tr>
<td>FB_HVACAntiBlockingDamper [338]</td>
<td>Blocking protection for damper drives</td>
</tr>
<tr>
<td>FB_HVACAntiBlockingPump [340]</td>
<td>Blocking protection for pumps</td>
</tr>
<tr>
<td>FB_HVACBlink [343]</td>
<td>Flash sequence</td>
</tr>
<tr>
<td>FB_HVACCmdCtrl_8 [344]</td>
<td>With this function block, individual units in a system can be sequentially switched on or off in a certain order. <strong>FB_HVACCmdCtrl_8</strong> can be used as the starting function block of a ventilation system.</td>
</tr>
<tr>
<td>FB_HVACCmdCtrlSystem1Stage [353]</td>
<td>Single-stage system switch</td>
</tr>
<tr>
<td>FB_HVACCmdCtrlSystem2Stage [362]</td>
<td>Two-stage system switch</td>
</tr>
<tr>
<td>FB_HVACConvertEnum [375]</td>
<td>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.</td>
</tr>
<tr>
<td>FB_HVACEnthalpy [376]</td>
<td>Determination of the dew point, the specific enthalpy and the absolute humidity.</td>
</tr>
<tr>
<td>FB_HVACFixedLimit [378]</td>
<td>Limit switch</td>
</tr>
<tr>
<td>FB_HVACFreezeProtectionHeater [380]</td>
<td>Frost protection monitoring</td>
</tr>
<tr>
<td>FB_HVACMux8 [59]</td>
<td>evaluates the FIFO memory of <strong>FB_HVACRedundancyCtrlEx</strong></td>
</tr>
<tr>
<td>FB_HVACMUX_INT_16 [387]</td>
<td>Function block contains two different types of multiplexer</td>
</tr>
<tr>
<td>FB_HVACMUX_INT_8 [383]</td>
<td>Function block contains two different types of multiplexer</td>
</tr>
<tr>
<td>FB_HVACMUX_REAL_16 [395]</td>
<td>Function block contains two different types of multiplexer</td>
</tr>
<tr>
<td>FB_HVACMUX_REAL_8 [391]</td>
<td>Function block contains two different types of multiplexer</td>
</tr>
<tr>
<td>FB_HVACOverwriteAnalog [398]</td>
<td>analog override in manual operation</td>
</tr>
<tr>
<td>FB_HVACOverwriteDigital [399]</td>
<td>digital override in manual operation</td>
</tr>
<tr>
<td>FB_HVACPowerMeasurementKL3403 [400]</td>
<td>Acquisition of the measured/calculated values of the three-phase power measurement terminal (KL3403)</td>
</tr>
<tr>
<td>FB_HVACPowerMeasurementKL3403Ex [402]</td>
<td>In comparison with the <strong>FB_HVACPowerMeasurementKL3403</strong>, the results are output in LREAL format. The output was extended by the frequencies of the three phases.</td>
</tr>
<tr>
<td>FB_HVACPriority_INT_16 [408]</td>
<td>This function block can be used to prioritize events or as a multiplexer</td>
</tr>
<tr>
<td>FB_HVACPriority_INT_8 [404]</td>
<td>This function block can be used to prioritize events or as a multiplexer</td>
</tr>
<tr>
<td>FB_HVACPriority_REAL_16 [417]</td>
<td>This function block can be used to prioritize events or as a multiplexer</td>
</tr>
<tr>
<td>FB_HVACPriority_REAL_8 [412]</td>
<td>This function block can be used to prioritize events or as a multiplexer</td>
</tr>
<tr>
<td>FB_HVACOptimizedOff [431]</td>
<td>Function block for the optimized switch-off (pre-stop) of boilers and air conditioning systems in conjunction with switching time function blocks.</td>
</tr>
<tr>
<td>FB_HVACOptimizedOn [421]</td>
<td>Function block for the optimized switch-on of boilers and air conditioning systems in conjunction with switching time function blocks.</td>
</tr>
<tr>
<td>FB_HVACTempChangeFunctionEntry [441]</td>
<td>Function block for the input of the interpolation points of the pre-start function</td>
</tr>
<tr>
<td>FB_HVACPWM [442]</td>
<td>PWM function block</td>
</tr>
<tr>
<td>FB_HVACStartAirConditioning [445]</td>
<td>Start program of an air handling unit</td>
</tr>
<tr>
<td>FB_HVACSummerNightCooling [449]</td>
<td>Summer night cooling</td>
</tr>
</tbody>
</table>
### HVAC Time schedule

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_HVACScheduler1ch</td>
<td>Weekly time switch with 1 time switch channel</td>
</tr>
<tr>
<td>FB_HVACScheduler7ch</td>
<td>Weekly time switch with 7 time switch channel</td>
</tr>
<tr>
<td>FB_HVACScheduler7TCHandling</td>
<td>this FB can be used to select and modify an individual line from the data array of a weekly timer</td>
</tr>
<tr>
<td>FB_HVACScheduler28ch</td>
<td>Weekly time switch with 28 time switch channel</td>
</tr>
<tr>
<td>FB_HVACScheduler28TCHandling</td>
<td>this FB can be used to select and modify an individual line from the data array of a weekly timer</td>
</tr>
<tr>
<td>FB_HVACSchedulerSpecialPeriods</td>
<td>Yearly scheduler with day, month and exactly time</td>
</tr>
<tr>
<td>FB_HVACSchedulerPublicHolidays</td>
<td>Yearly scheduler with day and month</td>
</tr>
</tbody>
</table>

### HVAC system

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_HVACGetSystemTime</td>
<td>Implementation of an internal clock in the TwinCAT PLC</td>
</tr>
<tr>
<td>FB_HVACNOVRAMDataHandling</td>
<td>Data are written fail-safe to the NOVRAM</td>
</tr>
<tr>
<td>FB_HVACPersistentDataHandling</td>
<td>Data are written fail-safe to a file</td>
</tr>
<tr>
<td>FB_HVACPersistentDataFileCopy</td>
<td>Copying of binary files between local or remote TwinCAT PC and the local TwinCAT PC</td>
</tr>
<tr>
<td>FB_HVACSetLocalTime</td>
<td>Sets the local Windows system time and the date of a TwinCAT system</td>
</tr>
<tr>
<td>FB_HVACSystemTaskInfo</td>
<td>Determination of system variables of the task</td>
</tr>
</tbody>
</table>

### HVAC backup function blocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_HVACNOVRAM</td>
<td>FBs for standard data types</td>
</tr>
<tr>
<td>FB_HVACPersistent</td>
<td>FBs for standard data types</td>
</tr>
</tbody>
</table>

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDataSecurityType</td>
<td>E_HVACDataSecurityType;</td>
</tr>
<tr>
<td>bSetDefault</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bEnable</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bIn</td>
<td>BOOL;</td>
</tr>
<tr>
<td>eCtrlModeActuator</td>
<td>E_HVAC2PointActuatorMode;</td>
</tr>
<tr>
<td>bManSwitch</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bLimitSwitchClose</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bLimitSwitchOpen</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bCtrlVoltage</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bReset</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bEnableLimitSwitch</td>
<td>BOOL;</td>
</tr>
<tr>
<td>eDataSecurityType: If eDataSecurityType := eHVACDataSecurityType_Persistent, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally. A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted the saved data are automatically read back from the flash into the RAM. Application example: <a href="https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip">https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip</a> If eDataSecurityType := eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.</td>
<td></td>
</tr>
</tbody>
</table>

NOTE

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

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

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

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).
**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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bOut</code></td>
<td>BOOL</td>
</tr>
<tr>
<td><code>byState</code></td>
<td>BYTE</td>
</tr>
<tr>
<td><code>eStateModeActuator</code></td>
<td>E_HVAC2PointActuatorMode</td>
</tr>
<tr>
<td><code>bErrorLimitSwitch</code></td>
<td>BOOL</td>
</tr>
<tr>
<td><code>bInvalidParameter</code></td>
<td>BOOL</td>
</tr>
</tbody>
</table>

**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` [487]).

**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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bEnableLimitSwitch</code></td>
<td>BOOL</td>
</tr>
<tr>
<td><code>tStrokeTime</code></td>
<td>TIME</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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 function block.

VAR_INPUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDataSecurityType</td>
<td>E_HVACDataSecurityType</td>
</tr>
<tr>
<td>bSetDefault</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEnable</td>
<td>BOOL</td>
</tr>
<tr>
<td>bAutoOpen</td>
<td>BOOL</td>
</tr>
<tr>
<td>bAutoClose</td>
<td>BOOL</td>
</tr>
<tr>
<td>eCtrlModeActuator</td>
<td>E_HVAC3PointActuatorMode</td>
</tr>
<tr>
<td>bManSwitch</td>
<td>BOOL</td>
</tr>
<tr>
<td>rFeedb</td>
<td>REAL</td>
</tr>
<tr>
<td>bCtrlVoltage</td>
<td>BOOL</td>
</tr>
<tr>
<td>bReset</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

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

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

**NOTE**

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

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

**bEnable**: The function block is enabled by the PLC program with the input variable `bEnable`. The three-point actuator always remains fully closed as long as the function block is not enabled. Output `bClose` is permanently TRUE.
bAutoOpen / bAutoClose: In automatic mode the three-point actuator is controlled by the input variables bAutoClose and bAutoOpen.

eCtrlModeActuator: Enum that defines the operation mode (see E_HVAC3PointActuatorMode [487]).

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bOpen</td>
<td>BOOL</td>
</tr>
<tr>
<td>bClose</td>
<td>BOOL</td>
</tr>
<tr>
<td>byState</td>
<td>BYTE</td>
</tr>
<tr>
<td>eStateModeActuator</td>
<td>E_HVAC3PointActuatorMode</td>
</tr>
<tr>
<td>bErrorFeedb</td>
<td>BOOL</td>
</tr>
<tr>
<td>bInvalidParameter</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

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 [487]).

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEnableFeedbCtrl</td>
<td>BOOL</td>
</tr>
<tr>
<td>rMinValueFeedb</td>
<td>REAL</td>
</tr>
<tr>
<td>rMaxValueFeedb</td>
<td>REAL</td>
</tr>
<tr>
<td>rHysteresisFeedb</td>
<td>REAL</td>
</tr>
<tr>
<td>tStrokeTime</td>
<td>TIME</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

### 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
- `bFeedbDelay`: BOOL
- `bOutsideTempLimit`: BOOL
- `bValvePositionLimit`: BOOL
- `bReqOutsideTemp`: BOOL
- `bReqValve`: BOOL
eDataSecurityType: If eDataSecurityType := eHVACDataSecurityType_Persistent, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instance 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/zip/5090272523.zip

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

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

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

bEnable: The function block is enabled if bEnable = TRUE.

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

eCtrlModeActuator: Enum that defines the operation mode (see E_HVACActuatorMode).

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 [488]).

bErrorPumpProtec: Pump error.

bErrorPumpFeedb: If the input bFeedbPump does not go TRUE within the time of tFeedbPumpDelay (tFeedbPumpDelay must be > #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

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT

VAR_IN_OUT
**NOTE**

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_HVACReqValve[.]`). The variable is saved persistently.

The table below shows a summary of all possible combinations:

<table>
<thead>
<tr>
<th><code>eReqOutsideTemp</code></th>
<th><code>eReqValve</code></th>
<th>Function</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTHigherLimit</td>
<td>NoRequest</td>
<td>outsidetemp higher limit</td>
<td>Heating circuit, air heater</td>
</tr>
<tr>
<td>OTHigherLimit</td>
<td>OrValvePosHigherLimit</td>
<td>outsidetemp OR valve higher limit</td>
<td></td>
</tr>
<tr>
<td>OTHigherLimit</td>
<td>AndValvePosHigherLimit</td>
<td>outsidetemp AND valve higher limit</td>
<td></td>
</tr>
<tr>
<td>NoRequest</td>
<td>NoRequest</td>
<td>no request</td>
<td>Primary pump</td>
</tr>
<tr>
<td>NoRequest</td>
<td>OrValvePosHigherLimit</td>
<td>valve higher limit</td>
<td></td>
</tr>
<tr>
<td>NoRequest</td>
<td>AndValvePosHigherLimit</td>
<td>not valid</td>
<td></td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
5.1.2.4 FB_HVACirculationPumpEx

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_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

double eDataSecurityType : E_HVACDataSecurityType;
bool bSetDefault : BOOL;
bool bEnable : BOOL;
bool bExternal : BOOL;
E_HVACActuatorMode eCtrlModeActuator : E_HVACActuatorMode;
double rOutsideTemp : REAL;
double rValvePosition : REAL;
bool bFeedPump : BOOL;
bool bPumpProtec : BOOL;
bool bManSwitch : BOOL;
bool bCtrlVoltage : BOOL;
bool bReset : BOOL;

eDataSecurityType: If eDataSecurityType [490] == 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 [479] 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/zip/5090272523.zip

If eDataSecurityType[490] = eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

NOTE
A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType[490] = 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_HVACirculationPumpEx: 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[488]).

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 rValvePositionLimitOn 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 `bErrorFeedbPump`. 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;
- `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 FB HVACCirculationPumpEx [33].

The output `bPump` switches itself off, after the switch-on conditions are no longer fulfilled, depending on the Enum `E_HVACRegPump` [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` := `bReset`;
  - `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 the diagrams 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 [488]).

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 \texttt{bPump} = TRUE, then the input \texttt{bFeedbPump} must be TRUE within the time delay \texttt{udiSecFeedbPumpDelay} and must remain so until \texttt{bPump} = FALSE. Otherwise this error is indicated via the output variable \texttt{bErrorFeedb} (0s..4294967s). \texttt{udiSecFeedbPumpDelay} is active only in the operation mode \texttt{eHVACActuatorMode\_Auto\_BMS OR eHVACActuatorMode\_Auto\_OP} and only if the time \texttt{udiSecFeedbPumpDelay} is greater than 0. The input is given in seconds. The variable is saved persistently. Preset to 0.

\texttt{rOutsideTempHighLimit}: Value above which the pump is switched on or off depending on the outside temperature \texttt{rOutsideTemp} and the enum \texttt{eReqPump}, see \texttt{FB\_HVACCirculationPumpEx\_33} (-60°C..60°C). \texttt{rOutsideTempHighLimit} is active only in the operation mode \texttt{eHVACActuatorMode\_Auto\_BMS OR eHVACActuatorMode\_Auto\_OP}. If an incorrect variable value is present on \texttt{rOutsideTempHighLimit}, the last valid variable value is taken. \texttt{bInvalidParameter} is set in the case of an incorrect parameter specification. The variable is saved persistently. Preset to 4.

\texttt{rOutsideTempLowLimit}: Value below which the pump is switched on or off depending on the outside temperature \texttt{rOutsideTemp} and the enum \texttt{eReqPump}, see \texttt{FB\_HVACCirculationPumpEx\_33} (-60°C..60°C). \texttt{rOutsideTempLowLimit} is active only in the operation mode \texttt{eHVACActuatorMode\_Auto\_BMS OR eHVACActuatorMode\_Auto\_OP}. If an incorrect variable value is present on \texttt{rOutsideTempLowLimit}, the last valid variable value is taken. \texttt{bInvalidParameter} is set in the case of an incorrect parameter specification. The variable is saved persistently. Preset to 1.

\texttt{rValvePositionLimitOn}: Threshold value for the position \texttt{rValvePosition} of a control valve associated with a pump, above which the pump should switch on automatically, e.g. heater pump, see \texttt{FB\_HVACCirculationPumpEx\_33} (0%..100%). \texttt{rValvePositionLimitOn} is active only in the operation mode \texttt{eHVACActuatorMode\_Auto\_BMS OR eHVACActuatorMode\_Auto\_OP}. \texttt{rValvePositionLimitOn} may not be smaller than \texttt{rValvePositionLimitOff}. Otherwise the last valid variable value is taken and \texttt{bInvalidParameter} is set. If an incorrect variable value is present on \texttt{rValvePositionLimit}, the last valid variable value is taken. \texttt{bInvalidParameter} is set in the case of an incorrect parameter specification. The variable is saved persistently. Preset to 5.

\texttt{rValvePositionLimitOff}: Threshold value for the position \texttt{rValvePosition} of a control valve associated with a pump, below which the pump should switch off automatically, e.g. heater pump, see \texttt{FB\_HVACCirculationPumpEx\_33} (0%..100%). \texttt{rValvePositionLimitOff} is active only in the operation mode \texttt{eHVACActuatorMode\_Auto\_BMS OR eHVACActuatorMode\_Auto\_OP}. \texttt{rValvePositionLimitOff} may not be larger than \texttt{rValvePositionLimitOn}. Otherwise the last valid variable value is taken and \texttt{bInvalidParameter} is set. If an incorrect variable value is present on \texttt{rValvePositionLimit}, the last valid variable value is taken. \texttt{bInvalidParameter} is set in the case of an incorrect parameter specification. The variable is saved persistently. Preset to 1.

\texttt{eReqPump}: Using the Enum \texttt{eReqPump\_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 \texttt{rOutsideTempLowLimit} or exceeds the limit value \texttt{rOutsideTempHighLimit}.

- depending on the position \texttt{rValvePosition} of the valve associated with the pump, the pump can be switched on when the threshold value \texttt{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 \texttt{bPump}:

\texttt{eHVACReqPump\_No}: There is no request on the part of the Enum to control the pump

\texttt{eHVACReqPump\_OT\_LL}: The outside temperature (OT = \texttt{rOutsideTemp}) must be smaller than \texttt{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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDataSecurityType</td>
<td>E_HVACDataSecurityType;</td>
</tr>
<tr>
<td>bSetDefault</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bEnable</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bAuto</td>
<td>BOOL;</td>
</tr>
<tr>
<td>eCtrlModeActuator</td>
<td>E_HVACActuatorMode;</td>
</tr>
<tr>
<td>bMotorProtec</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bFeedbContactor</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bFeedbProcess</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bRepairSwitch</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bManSwitch</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bCtrlVoltage</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bReset</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>

**eDataSecurityType:** If `eDataSecurityType [490] := 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 [479]` 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/zip/5090272523.zip

If `eDataSecurityType [490] := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

**NOTE**

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType [490] := 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 [488]`). 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 \( b_{\text{FeedbProcess}} \). The process feedback is present if the input \( b_{\text{FeedbProcess}} = \text{TRUE} \) (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 \( t_{\text{FeedbProcessTimer}} \), the drive switches off and indicates a fault at the output \( b_{\text{ErrorFeedbProcess}} \). The motor can only be restarted if the fault has been rectified and acknowledged at the input \( b_{\text{Reset}} \). 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 \( b_{\text{FeedbProcess}} \) can be delayed by the time \( t_{\text{DelayFeedbProcess}} \).

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

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

**bCtrlVoltage**: The state of the control voltage is monitored with the input \( b_{\text{CtrlVoltage}} \). The motor can only be switched on if \( b_{\text{CtrlVoltage}} = \text{TRUE} \) (quiescent current principle). If the control voltage is switched off, \( b_{\text{CtrlVoltage}} = \text{FALSE} \) and the output \( b_{\text{Motor}} \) 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**

\[
\begin{align*}
\text{bMotor} & : \text{BOOL}; \\
\text{byState} & : \text{BYTE}; \\
\text{bStateAuto} & : \text{BOOL}; \\
\text{bStateRepairSwitch} & : \text{BOOL}; \\
\text{bStateManSwitch} & : \text{BOOL}; \\
\text{eStateModeActuator} & : \text{HVACActuatorMode}; \\
\text{bErrorGeneral} & : \text{BOOL}; \\
\text{byError} & : \text{BYTE}; \\
\text{bErrorMotorprotec} & : \text{BOOL}; \\
\text{bErrorFeedbContactor} & : \text{BOOL}; \\
\text{bErrorFeedbProcess} & : \text{BOOL}; \\
\text{bInvalidParameter} & : \text{BOOL}; \\
\end{align*}
\]

**bMotor**: Output variable for controlling a single-speed motor.

**byState**: Status byte indicating the operating state of the function block.

\[
\begin{align*}
\text{byState.0} & := \text{bEnable}; \\
\text{byState.1} & := \text{bMotor}; \\
\text{byState.2} & := \text{bStateAuto}; \\
\text{byState.5} & := \text{bStateRepairSwitch}; \\
\text{byState.6} & := \text{bStateManSwitch}; \\
\text{byState.7} & := \text{bCtrlVoltage}; \\
\end{align*}
\]

**bStateAuto**: Indicates the state for automatic preselection if the operation mode \( \text{eCtrlModeActuator} \) is either \( \text{HVACActuatorMode\_Auto\_BMS} \) or \( \text{HVACActuatorMode\_Auto\_OP} \) and stage 1 was activated via the input variable \( b_{\text{Auto}} \).

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

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

**eStateModeActuator**: Enum via which the state of the operation mode of the motor is fed back to the controller (see \( \text{HVACActuatorMode} \)).

**bErrorGeneral**: The error message \( b_{\text{ErrorGeneral}} \) becomes TRUE as soon as one of the error messages \( b_{\text{ErrorMotorprotec}}, b_{\text{ErrorFeedbContactor}} \) or \( b_{\text{ErrorFeedbProcess}} \) is TRUE. The output \( b_{\text{Motor}} \) is then set to FALSE and is only enabled again when the fault has been rectified and acknowledged via \( b_{\text{Reset}} \).
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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>tStartDelay</td>
<td>TIME</td>
</tr>
<tr>
<td>tStopDelay</td>
<td>TIME</td>
</tr>
<tr>
<td>tDelayFeedbContactor</td>
<td>TIME</td>
</tr>
<tr>
<td>tFeedbProcessTimer</td>
<td>TIME</td>
</tr>
<tr>
<td>tDelayFeedbProcess</td>
<td>TIME</td>
</tr>
</tbody>
</table>

**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.
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 \( t_{\text{DelaySpeed2toSpeed1}} \).

**VAR_INPUT**

<table>
<thead>
<tr>
<th>eDataSecurityType</th>
<th>E_HVACDataSecurityType;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bSetDefault</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bEnable</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bAutoSpeed1</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bAutoSpeed2</td>
<td>BOOL;</td>
</tr>
<tr>
<td>eCtrlModeActuator</td>
<td>E_HVACActuatorMode;</td>
</tr>
<tr>
<td>bMotorProtecSpeed1</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bMotorProtecSpeed2</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bFeedbContactorSpeed1</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bFeedbContactorSpeed2</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bFeedbProcess</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bRepairSwitch</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bManSwitch</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bCtrlVoltage</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bReset</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>
| eDataSecurityType: If eDataSecurityType \( \neq \) E_HVACDataSecurityType_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 \( \neq \) FB3_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.
A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted the saved data are automatically read back from the flash in the RAM.

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

If eDataSecurityType[490] := eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

**NOTE**
A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType[490] := 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[488]). 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.

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bMotorSpeed1</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bMotorSpeed2</td>
<td>BOOL;</td>
</tr>
<tr>
<td>byState</td>
<td>BYTE;</td>
</tr>
<tr>
<td>bStateAutoSpeed1</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bStateAutoSpeed2</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bStateRepairSwitch</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bStateManSwitch</td>
<td>BOOL;</td>
</tr>
<tr>
<td>eStateModeActuator</td>
<td>E_HVACActuatorMode;</td>
</tr>
<tr>
<td>bErrorGeneral</td>
<td>BOOL;</td>
</tr>
<tr>
<td>byError</td>
<td>BYTE;</td>
</tr>
<tr>
<td>bErrorMotorprotecSpeed1</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bErrorMotorprotecSpeed2</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bErrorFeedbContactorSpeed1</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bErrorFeedbContactorSpeed2</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bErrorFeedbProcess</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bInvalidParameter</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>

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

bMotorSpeed2: Output variable for controlling speed 2 of the two-speed drive.
byState: Status byte indicating the operating state of the function block
byState.0 := bEnable;
byState.1 := bMotorSpeed1;
byState.2 := bMotorSpeed2;
byState.3 := bStateAutoSpeed1;
byState.4 := bStateAutoSpeed2;
byState.5 := bStateRepairSwitch;
byState.6 := bStateManSwitch;
byState.7 := bCtrlVoltage;

bStateAutoSpeed1: Indicates the state for automatic preselection 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 [P 488]).

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 bFeedbProcess
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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>tStartDelay</td>
<td>TIME;</td>
</tr>
<tr>
<td>tStopDelay</td>
<td>TIME;</td>
</tr>
<tr>
<td>tDelayFeedbContactorSpeed1</td>
<td>TIME;</td>
</tr>
<tr>
<td>tDelayFeedbContactorSpeed2</td>
<td>TIME;</td>
</tr>
<tr>
<td>tFeedbProcessTimer</td>
<td>TIME;</td>
</tr>
<tr>
<td>tDelayFeedbProcess</td>
<td>TIME;</td>
</tr>
<tr>
<td>tDelaySpeed1ToSpeed2</td>
<td>TIME;</td>
</tr>
<tr>
<td>tDelaySpeed2ToSpeed1</td>
<td>TIME;</td>
</tr>
</tbody>
</table>

**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. $b\text{InvalidParameter}$ 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 $b\text{MotorSpeed1}$ and $b\text{MotorSpeed2}$ 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. $b\text{InvalidParameter}$ 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 $b\text{MotorSpeed1}$ and $b\text{MotorSpeed2}$ are FALSE for the time $t\text{DelaySpeed2ToSpeed1}$ 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. $b\text{InvalidParameter}$ is set in the case of an incorrect parameter specification.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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[490] := 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[479]` 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/zip/5090272523.zip](https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip)

If `eDataSecurityType[490] := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

<table>
<thead>
<tr>
<th>NOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if <code>eDataSecurityType[490] := eHVACDataSecurityType_Persistent</code>. It would lead to early wear of the flash memory.</td>
</tr>
</tbody>
</table>

**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[488]` 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[488]` 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[488]` 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[488]`). 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.

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.

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.

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;
bErrorMotorprotectSpeed1 : BOOL;
bErrorMotorprotectSpeed2 : BOOL;
bErrorMotorprotectSpeed3 : BOOL;
bErrorFeedbackContactorSpeed1 : BOOL;
bErrorFeedbackContactorSpeed2 : BOOL;
bErrorFeedbackContactorSpeed3 : BOOL;
bErrorFeedbackProcess : 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 [488] 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 [488] 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 [488] 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.

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.

bErrorFeedbContactorSpeed2: Power section feedback error, see input variable bFeedbContactorSpeed2.

bErrorFeedbContactorSpeed3: Power section feedback error, see input variable bFeedbContactorSpeed3.

bErrorFeedbProcess: Process feedback error, see input variable bFeedbackProcess.

blInvalidParameter: Indicates that an incorrect parameter is present at one of the variables eCtrlModeActuator, tStartDelay, tStopDelay, tDelayFeedbContactorSpeed1, tDelayFeedbContactorSpeed2, 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 blInvalidParameter must be acknowledged via bReset.

VAR_IN_OUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tStartDelay</td>
<td>TIME;</td>
</tr>
<tr>
<td>tStopDelay</td>
<td>TIME;</td>
</tr>
<tr>
<td>tDelayFeedbContactorSpeed1</td>
<td>TIME;</td>
</tr>
<tr>
<td>tDelayFeedbContactorSpeed2</td>
<td>TIME;</td>
</tr>
</tbody>
</table>
tStartDelay: The start-up of the motor after enabling and switching on via the operation mode of the motor is delayed by the time $t_{\text{StartDelay}}$ [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. $b_{\text{InvalidParameter}}$ is set in the case of an incorrect parameter specification.

$t_{\text{StopDelay}}$: The switching off of the motor in the respective switch-on stage, either by switching the enable $b_{\text{Enable}}$ to FALSE, or by switching off via the operation mode $e_{\text{CtrlModeActuator}}$, or in automatic operation mode by switching the input variables $b_{\text{AutoSpeed1}}$, $b_{\text{AutoSpeed2}}$ and $b_{\text{AutoSpeed3}}$ to FALSE, is delayed by the time $t_{\text{StopDelay}}$ [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. $b_{\text{InvalidParameter}}$ is set in the case of an incorrect parameter specification.

$t_{\text{DelayFeedbContactorSpeed1}}$: Time delay of the feedback of the power section after switching on the motor. If this time has elapsed and $b_{\text{FeedbContactorSpeed1}} = \text{FALSE}$, then this is fed back to the controller via the error message $b_{\text{ErrorFeedbContactorSpeed1}}$ (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. $b_{\text{InvalidParameter}}$ is set in the case of an incorrect parameter specification.

$t_{\text{DelayFeedbContactorSpeed2}}$: Time delay of the feedback of the power section after switching on the motor. If this time has elapsed and $b_{\text{FeedbContactorSpeed2}} = \text{FALSE}$, then this is fed back to the controller via the error message $b_{\text{ErrorFeedbContactorSpeed2}}$ (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. $b_{\text{InvalidParameter}}$ is set in the case of an incorrect parameter specification.

$t_{\text{DelayFeedbContactorSpeed3}}$: Time delay of the feedback of the power section after switching on the motor. If this time has elapsed and $b_{\text{FeedbContactorSpeed3}} = \text{FALSE}$, then this is fed back to the controller via the error message $b_{\text{ErrorFeedbContactorSpeed3}}$ (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. $b_{\text{InvalidParameter}}$ is set in the case of an incorrect parameter specification.

$t_{\text{FeedbackProcessTimer}}$: Time delay [s] of the process feedback $b_{\text{FeedbProcess}}$ during the motor start-up phase. If this time has elapsed and $b_{\text{FeedbProcess}} = \text{FALSE}$, then this is fed back to the controller via the error message $b_{\text{ErrorFeedbProcess}}$ (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. $b_{\text{InvalidParameter}}$ is set in the case of an incorrect parameter specification.

$t_{\text{DelayFeedbProcess}}$: In order to avoid undesired switching off of the drive during operation due to the process monitoring $b_{\text{FeedbProcess}}$, e.g. in the event of short-term pressure fluctuations, the response of the input $b_{\text{FeedbProcess}}$ can be delayed by the time $t_{\text{DelayFeedbProcess}}$ [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. $b_{\text{InvalidParameter}}$ is set in the case of an incorrect parameter specification.

$t_{\text{DelaySpeed1}}$: 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 *tDelaySpeed3ToSpeed2* 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

5.1.2.8 **FB_HVACMux8**

This function block evaluates the FIFO memory from *FB_HVACRedundancyCtrlEx* [165]. 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 rln1 is now output on rQ2 and rln2 on rQ3 in FB_HVACMux8.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>arrFiFo</th>
<th>: Array[1..8] of INT;</th>
</tr>
</thead>
<tbody>
<tr>
<td>rIn1 - rIn8</td>
<td>: REAL;</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>rQ1 - rQ8</th>
<th>: REAL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bErr</td>
<td>: BOOL;</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

**5.1.2.9 FB_HVACMux8Ex**

This function block evaluates the FIFO memory from FB_HVACRedundancyCtrlEx. 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:

- **rln1** on the output **rQ4**
- **rln2** on the output **rQ3**
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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.22 or higher</td>
<td>Tc2_HVAC V3.3.4.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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

- bln1 on the output bQ4
- bln2 on the output bQ3
- bln3 on the output bQ1
- bln4 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.22 or higher</td>
<td>Tc2_HVAC V3.3.4.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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 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 ▶490]= 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 ▶479] 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/zip/5090272523.zip

If eDataSecurityType ▶490]= eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

**NOTE**

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType ▶490]= 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.
**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bQ1 - bQ8</td>
<td>BOOL</td>
</tr>
<tr>
<td>eErrorCode</td>
<td>E_HVACErrorCodes</td>
</tr>
<tr>
<td>bErrorGeneral</td>
<td>BOOL</td>
</tr>
<tr>
<td>bInvalidParameter</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

**bQ1 - 8**: Actuator feedback.

**eErrorCode**: Indicates which actuator has not returned an operating feedback within the prespecified time interval (see E_HVACErrorCodes [490]). 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**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>rHours</td>
<td>REAL</td>
</tr>
<tr>
<td>uiCount</td>
<td>UINT</td>
</tr>
<tr>
<td>tOverlap1Actuator</td>
<td>TIME</td>
</tr>
<tr>
<td>uiMaxOnLevel</td>
<td>UINT</td>
</tr>
<tr>
<td>tNoFeedbActuator</td>
<td>TIME</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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` [page 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**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDataSecurityType</td>
<td>E_HVACDataSecurityType;</td>
</tr>
<tr>
<td>bSetDefault</td>
<td>BOOL;</td>
</tr>
<tr>
<td>iCountCtrl</td>
<td>INT;</td>
</tr>
<tr>
<td>bEn1 – bEn8</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bFeedback1 – bFeedback8</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bForceChange</td>
<td>BOOL;</td>
</tr>
<tr>
<td>rHours1–rHours8</td>
<td>REAL;</td>
</tr>
</tbody>
</table>
eDataSecurityType: If eDataSecurityType := eHVACDataSecurityType_Persistent, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

<table>
<thead>
<tr>
<th>NOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType := eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.</td>
</tr>
</tbody>
</table>

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bQ1 - bQ8</td>
<td>BOOL</td>
</tr>
<tr>
<td>arrFiFo</td>
<td>Array[1..8] of INT</td>
</tr>
<tr>
<td>eErrorCode</td>
<td>E_HVACErrorCodes</td>
</tr>
<tr>
<td>bErrorGeneral</td>
<td>BOOL</td>
</tr>
<tr>
<td>bInvalidParameter</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

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). The recognition of this group of errors is activated by a time greater than 0 in the variable tNoFeedbActuator.

<table>
<thead>
<tr>
<th>eHVACErrorCodes</th>
<th>Error NoFeedbackActuator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error_NoFeedbackActuator1 := 15,</td>
<td></td>
</tr>
<tr>
<td>Error_NoFeedbackActuator2 := 16,</td>
<td></td>
</tr>
<tr>
<td>Error_NoFeedbackActuator3 := 17,</td>
<td></td>
</tr>
<tr>
<td>Error_NoFeedbackActuator4 := 18,</td>
<td></td>
</tr>
<tr>
<td>Error_NoFeedbackActuator5 := 19,</td>
<td></td>
</tr>
<tr>
<td>Error_NoFeedbackActuator6 := 20,</td>
<td></td>
</tr>
<tr>
<td>Error_NoFeedbackActuator7 := 21,</td>
<td></td>
</tr>
<tr>
<td>Error_NoFeedbackActuator8 := 22,</td>
<td></td>
</tr>
</tbody>
</table>

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**: 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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

```plaintext
VAR_INPUT
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault : BOOL;
iRawValue : INT;
tTaskCycleTime : TIME;
tCtrlCycleTime : TIME;
bEnableLimitCtrl : BOOL;
bCtrlVoltage : BOOL;
bReset : BOOL;
```
eDataSecurityType: If eDataSecurityType := eHVACDataSecurityType_Persistent, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

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

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

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

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

tCtrlCycleTime: Cycle time with which the function block is processed. This must be greater than or equal to 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 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).

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;
orOffset : 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 impossibly 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 impossibly 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.

orOffset: 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 + orOffset

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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;
- `rRangeHigh` :
- `rRangeLow` :
- `rDirection` :
- `bEnableFeedbCtrl` :
- `rHysteresisFeedbCtrl` :
- `bDelayFeedbCtrl` :

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

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

**NOTE**

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

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

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

**rSetpoint**: The setpoint for the analog output is transferred to the function block with the variable `rSetpoint`.
bCtrlVoltage: The control voltage is present if the bCtrlVoltage variable is TRUE. The feedback control is deactivated in the event of failure of the control voltage so that no false alarms occur.

eCtrlModeAnalogOutput: Enum that defines the operation mode (see E_HVACAnalogOutputMode [488]).

```
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 [488]).

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;
'hysteresisFeedbCtrl': 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,
\[\Rightarrow rY = 50\]
\[\Rightarrow rY\_Terminal = 16384\]

rSetpoint = 200, rRangeLow = 0, rRangeHigh = 200,
\[\Rightarrow rY = 100\]
\[\Rightarrow rY\_Terminal = 32767\]

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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

#### 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: \(y = m \times x + b\).
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)

\[
\begin{align*}
  rX1 &= 0 / \ iY1 &= 6553 \\
  rX2 &= 100 / \ iY2 &= 32767
\end{align*}
\]

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 [490] = 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 [479]` 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/zip/5090272523.zip

If eDataSecurityType[490] := eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

NOTE
A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType[490] := 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 [488]).

```plaintext
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**

```plaintext
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 [488]).

**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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
This function block serves to convert an analog control signal into a three-point step signal. Hence three-point dampers or valves can be controlled by a controller with a continuous control signal. The function block \textit{FB_HVACAnalogTo3Point} works with or without a continuous positional feedback signal from the drive.
Example: 3-Point-Actuator without feedback and limit switches

![Diagram](image)

**VAR_INPUT**

```plaintext
VAR_INPUT

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

**VAR_OUTPUT**

```plaintext
VAR_OUTPUT

eDataSecurityType : E_HVACDataSecurityType;
bOpen : BOOL;
bClose : BOOL;
bErrorGeneral : BOOL;
```

**VAR_IN_OUT**

```plaintext
VAR_IN_OUT

eDataSecurityType : E_HVACDataSecurityType;
```

**eDataSecurityType**: If `eDataSecurityType[490] := 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[479]` 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/zip/5090272523.zip

If `eDataSecurityType[490] := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.
A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType := eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.

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

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

**rSetpoint:** The setpoint for the position of the damper or the valve is transferred to the function block with the variable rSetpoint. The 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.
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[491] = eHVACReferencingMode_AnalogFeedback, then rFeedb is fed back to the controller via the output variable rPosition. If rFeedb > rMaxSetpoint, then rPosition = rMaxSetpoint and blnValidParameter is set. If rFeedb < rMinSetpoint, then rPosition = rMinSetpoint and blnValidParameter 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 := blnValidParameter
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.

blnInvalidParameter: Indicates that an incorrect parameter is present at one of the variables rMinSetpoint, rMaxSetpoint, rThreshold, tTolerance, 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 blnInvalidParameter 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. blnInvalidParameter 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 \( rTolerance \).

\( rSetpoint - rPosition \geq rThreshold: bOpen \) is activated

\( rPosition - rSetpoint \geq rThreshold: bClose \) 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. \( tPauseMin \) 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. \( tPauseMin \) preset to 500 ms.

**eReferencingMode**: Enum that specifies the referencing mode of the function block (see \( E_HVACReferencingMode \) [491]).

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. \( tPauseMin \) preset to 0.
Table 1: Example

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tInterval</td>
<td>120 seconds</td>
</tr>
<tr>
<td>tPauseMin</td>
<td>10 seconds</td>
</tr>
<tr>
<td>tPauseMax</td>
<td>60 seconds</td>
</tr>
<tr>
<td>rSetpoint</td>
<td>100%</td>
</tr>
<tr>
<td>rPosition</td>
<td>50%</td>
</tr>
<tr>
<td>tPauseInterval / bClose OR bOpen</td>
<td>35 seconds/85 seconds</td>
</tr>
<tr>
<td>rSetpoint</td>
<td>50%</td>
</tr>
<tr>
<td>rPosition</td>
<td>25%</td>
</tr>
<tr>
<td>tPauseInterval / bClose OR bOpen</td>
<td>47,5 seconds/72,5 seconds</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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`, `bSetSensor` 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 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` or `bScanSensor`. If scanning or setting has been concluded with a rising edge on `bSetSensor` or `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` or `bScanSensor`. If scanning or setting has been concluded with a rising edge on `bSetSensor` or `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 [492] 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 [492].

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 [492].

eSensorType: Enum with which the sensor type for the respective channel is specified for setting via bSetSensor (see E_HVACSensorType [492]).

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 [492]).

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])

<table>
<thead>
<tr>
<th>Sensor type</th>
<th>KL3201/2/4-0000</th>
<th>KL3201/2/4-0010</th>
<th>KL3201/2/4-0012</th>
<th>KL3201/2/4-0014</th>
<th>KL3201/2/4-0016</th>
<th>KL3201/2/4-0020</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Raw value</td>
<td>Raw value</td>
<td>Raw value</td>
<td>Raw value</td>
<td>Raw value</td>
<td>Raw value</td>
</tr>
<tr>
<td>PT100</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
</tr>
<tr>
<td>PT200</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
</tr>
<tr>
<td>PT500</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
</tr>
<tr>
<td>PT1000</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
</tr>
<tr>
<td>Ni100</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
</tr>
<tr>
<td>Ni120</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
</tr>
<tr>
<td>Ni100(0,DIN)</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
<td>x 1/10°C x</td>
</tr>
<tr>
<td>Ni100(0,Tk5000,LS)</td>
<td>not supported</td>
<td>not supported</td>
<td>not supported</td>
<td>not supported</td>
<td>not supported</td>
<td>not supported</td>
</tr>
<tr>
<td>Resist ance measurement 10 ... 1200 Ω</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Resist ance measurement 10 ... 5000 Ω</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Resist ance measurement 10 ... 10000 Ω</td>
<td>not supported</td>
<td>not supported</td>
<td>not supported</td>
<td>not supported</td>
<td>not supported</td>
<td>not supported</td>
</tr>
</tbody>
</table>
### Sensor type

<table>
<thead>
<tr>
<th>Sensor type</th>
<th>KL3201/2/4-0 023</th>
<th>KL3201/2/4-0 025</th>
<th>KL3201/2/4-0 029</th>
<th>KL3201/2/4-0 031</th>
<th>KL3228</th>
<th>KL3208-0010</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT100</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>not supported</td>
</tr>
<tr>
<td>PT200</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>not supported</td>
</tr>
<tr>
<td>PT500</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>not supported</td>
</tr>
<tr>
<td>PT1000</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Ni100</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>not supported</td>
</tr>
<tr>
<td>Ni120</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>not supported</td>
</tr>
<tr>
<td>Ni100 (DIN)</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Ni100 (Tk5000,LS)</td>
<td>not supported</td>
<td>not supported</td>
<td>x</td>
<td>1/10°C</td>
<td>1/10°C</td>
<td>x</td>
</tr>
<tr>
<td>Resistance measurement 10 ... 1200 Ω</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>not supported</td>
<td>not supported</td>
</tr>
<tr>
<td>Resistance measurement 10 ... 5000 Ω</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>not supported</td>
<td>not supported</td>
</tr>
<tr>
<td>Resistance measurement 10 ... 10000 Ω</td>
<td>not supported</td>
<td>not supported</td>
<td>not supported</td>
<td>not supported</td>
<td>not supported</td>
<td>not supported</td>
</tr>
</tbody>
</table>

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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 \[490\] = 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 \[479\] 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/zip/5090272523.zip

If eDataSecurityType \[490\] = eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

NOTE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType \[490\] = 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:

![FBD diagram]

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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 \( \text{stParameterScale\_nP} \) [496].

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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>rX</td>
<td>REAL</td>
</tr>
<tr>
<td>stParameterScale_nPoint</td>
<td>ST_HVACParameterScale_nPoint;</td>
</tr>
</tbody>
</table>

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

stParameterScale_nPoint: Structure via which the individual points of the XY coordinates are given their valency (see ST_HVACParameterScale_nPoint). 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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>rY</td>
<td>REAL</td>
</tr>
<tr>
<td>stStateParameterScale_nPoint</td>
<td>ST_HVACParameterScale_nPoint;</td>
</tr>
<tr>
<td>bErrorNumberOfPoint</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bErrorParameter</td>
<td>BOOL;</td>
</tr>
<tr>
<td>uiErrorPositionX_Parameter</td>
<td>INT;</td>
</tr>
</tbody>
</table>

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

stStateParameterScale_nPoint: State of the structure stParameterScale_nPoint. If bErrorNumberOfPoint or bErrorParameter = TRUE, 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).

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 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 [496] (see Parameter [505]).

Formulas for the linear equation, two-point form:

\[ m = \frac{Y_2 - Y_1}{X_2 - X_1} \]
\[ m = \frac{\text{stParameterScale}_nPoint.rY[2] - \text{stParameterScale}_nPoint.rY[1]}{\text{stParameterScale}_nPoint.rX[2] - \text{stParameterScale}_nPoint.rX[1]} \]
\[ Y = Y_1 + m \cdot (X - X_1) \]
\[ rY = rY[1] + m \cdot (rX - \text{stParameterScale}_nPoint.rX[1]) \]

Example curve 1 with calculation

\[ m = \text{rM} = \text{slope} \]
\[ X = 1199.8 \]
\[ X_2 = \text{stParameterScale}_nPoint.rX[2] = 1209 \]
\[ X_3 = \text{stParameterScale}_nPoint.rX[3] = 1163 \]
\[ Y_2 = \text{stParameterScale}_nPoint.rY[2] = 20 \]
\[ Y_3 = \text{stParameterScale}_nPoint.rY[3] = 21 \]
\[ m = \frac{(Y_3 - Y_2)}{(X_3 - X_2)} \]
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

IF \( rX < rX[1] \) THEN
\( rY := rY[1] \);
END_IF

\( rY \)
\( rY[1] \)
\( rY[2] \)
\( rY[5] \)
\( rY[6] \)
\( rX[1] \)
\( rX[2] \)
\( rX[3] \)
\( rX[4] \)
\( rX[5] \)
\( rX[6] \)
\( rX \)

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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 `IrX` and output as the calculated `Y` value at the output `IrY`. Different slopes of the characteristic curve can be specified in sections by specifying various interpolation points. The range is limited internally to the values `IrY01` and `IrY07` with the aid of the input `bLmt`.

**VAR_INPUT**

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

- **IrX**: 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**: Y 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 `IrY01` and `IrY0n`, depending on the maximum number of interpolation points.

**VAR_OUTPUT**

```plaintext
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 [490]` will be displayed.
- **udiErrorArgument**: Indication of which interpolation point has been entered incorrectly.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
5.1.3.9 \textbf{FB_HVACTemperatureCurve}

A temperature curve stored internally in the function block is selected from Tables 1 and 2 via the Enum \texttt{eTemperatureCurve \[494\]} and output via the structure \texttt{stParameterScale\_nPoint}. Characteristic curves can then be reproduced with this structure in conjunction with the function block \texttt{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 \texttt{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.

\textbf{VAR\_INPUT}

\begin{verbatim}
VAR_INPUT
eTemperatureCurve : E_HVACTemperatureCurve;
\end{verbatim}

\texttt{eTemperatureCurve} is an Enum via which a temperature curve stored internally in the function block can be selected from Tables 1 and 2 (see \texttt{E_HVACTemperatureCurve \[494\]}).

\textbf{VAR\_OUTPUT}

\begin{verbatim}
VAR_OUTPUT
stParameterScale\_nPoint : ST_HVACParameterScale\_nPoint;
\end{verbatim}

\texttt{stParameterScale\_nPoint} is a structure containing the number and valences of the points of the XY coordinates (see \texttt{ST_HVACParameterScale\_nPoint \[496\]}). \texttt{stParameterScale\_nPoint} contains the curves stored in Tables 1 and 2, depending on the specification via the Enum \texttt{eTemperatureCurve}.

Example:

\begin{verbatim}
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;
\end{verbatim}
Specified temperature curves, selectable via the Enum eTemperatureCurve
<table>
<thead>
<tr>
<th>°C</th>
<th>Ω</th>
<th>Ω</th>
<th>Ω</th>
<th>Ω</th>
<th>Ω</th>
</tr>
</thead>
<tbody>
<tr>
<td>-50</td>
<td>32886</td>
<td></td>
<td></td>
<td>743</td>
<td>803</td>
</tr>
<tr>
<td>-40</td>
<td>18641</td>
<td></td>
<td></td>
<td>791</td>
<td>843</td>
</tr>
<tr>
<td>-30</td>
<td>11130</td>
<td>21695</td>
<td>27886</td>
<td>53093</td>
<td>842</td>
</tr>
<tr>
<td>-20</td>
<td>6777</td>
<td>12987</td>
<td>16502</td>
<td>29125</td>
<td>893</td>
</tr>
<tr>
<td>-15</td>
<td>5341</td>
<td>10153</td>
<td>12844</td>
<td>21887</td>
<td>920</td>
</tr>
<tr>
<td>-10</td>
<td>4247</td>
<td>8011</td>
<td>10070</td>
<td>16599</td>
<td>946</td>
</tr>
<tr>
<td>-5</td>
<td>3390</td>
<td>6347</td>
<td>8134</td>
<td>12698</td>
<td>973</td>
</tr>
<tr>
<td>0</td>
<td>2728</td>
<td>5071</td>
<td>6452</td>
<td>9795</td>
<td>1000</td>
</tr>
<tr>
<td>1</td>
<td>2613</td>
<td>4851</td>
<td>6164</td>
<td>9309</td>
<td>1004.4</td>
</tr>
<tr>
<td>2</td>
<td>2503</td>
<td>4640</td>
<td>5891</td>
<td>8849</td>
<td>1008.9</td>
</tr>
<tr>
<td>3</td>
<td>2399</td>
<td>4441</td>
<td>5631</td>
<td>8415</td>
<td>1013.3</td>
</tr>
<tr>
<td>4</td>
<td>2300</td>
<td>4252</td>
<td>5384</td>
<td>8005</td>
<td>1017.8</td>
</tr>
<tr>
<td>5</td>
<td>2205</td>
<td>4071</td>
<td>5150</td>
<td>7617</td>
<td>1020</td>
</tr>
<tr>
<td>6</td>
<td>2115</td>
<td>3899</td>
<td>4927</td>
<td>7251</td>
<td>1026.7</td>
</tr>
<tr>
<td>7</td>
<td>2030</td>
<td>3738</td>
<td>4715</td>
<td>6905</td>
<td>1031.2</td>
</tr>
<tr>
<td>8</td>
<td>1948</td>
<td>3582</td>
<td>4513</td>
<td>6575</td>
<td>1035.8</td>
</tr>
<tr>
<td>9</td>
<td>1870</td>
<td>3434</td>
<td>4321</td>
<td>6265</td>
<td>1040.3</td>
</tr>
<tr>
<td>10</td>
<td>1796</td>
<td>3294</td>
<td>4138</td>
<td>5971</td>
<td>1056</td>
</tr>
<tr>
<td>11</td>
<td>1724</td>
<td>3158</td>
<td>3964</td>
<td>5691</td>
<td>1049.3</td>
</tr>
<tr>
<td>12</td>
<td>1656</td>
<td>3029</td>
<td>3797</td>
<td>5427</td>
<td>1053.9</td>
</tr>
<tr>
<td>13</td>
<td>1590</td>
<td>2905</td>
<td>3639</td>
<td>5177</td>
<td>1058.4</td>
</tr>
<tr>
<td>14</td>
<td>1528</td>
<td>2788</td>
<td>3488</td>
<td>4938</td>
<td>1063</td>
</tr>
<tr>
<td>15</td>
<td>1469</td>
<td>2677</td>
<td>3345</td>
<td>4713</td>
<td>1084</td>
</tr>
<tr>
<td>16</td>
<td>1412</td>
<td>2570</td>
<td>3207</td>
<td>4500</td>
<td>1072.2</td>
</tr>
<tr>
<td>17</td>
<td>1358</td>
<td>2468</td>
<td>3076</td>
<td>4298</td>
<td>1076.8</td>
</tr>
<tr>
<td>18</td>
<td>1306</td>
<td>2371</td>
<td>2952</td>
<td>4104</td>
<td>1081.4</td>
</tr>
<tr>
<td>19</td>
<td>1256</td>
<td>2277</td>
<td>2832</td>
<td>3922</td>
<td>1086</td>
</tr>
<tr>
<td>20</td>
<td>1209</td>
<td>2189</td>
<td>2719</td>
<td>3747</td>
<td>1112</td>
</tr>
<tr>
<td>21</td>
<td>1163</td>
<td>2103</td>
<td>2610</td>
<td>3582</td>
<td>1095.3</td>
</tr>
<tr>
<td>22</td>
<td>1120</td>
<td>2023</td>
<td>2506</td>
<td>3426</td>
<td>1100</td>
</tr>
<tr>
<td>23</td>
<td>1078</td>
<td>1945</td>
<td>2407</td>
<td>3277</td>
<td>1104.6</td>
</tr>
<tr>
<td>24</td>
<td>1038</td>
<td>1871</td>
<td>2289</td>
<td>3135</td>
<td>1109.3</td>
</tr>
<tr>
<td>25</td>
<td>1000</td>
<td>1800</td>
<td>2200</td>
<td>3000</td>
<td>1142</td>
</tr>
<tr>
<td>26</td>
<td>963</td>
<td>1732</td>
<td>2115</td>
<td>2872</td>
<td>1120</td>
</tr>
<tr>
<td>27</td>
<td>928</td>
<td>1667</td>
<td>2034</td>
<td>2750</td>
<td>1123.4</td>
</tr>
<tr>
<td>28</td>
<td>894</td>
<td>1604</td>
<td>1957</td>
<td>2634</td>
<td>1128.1</td>
</tr>
<tr>
<td>29</td>
<td>862</td>
<td>1545</td>
<td>1883</td>
<td>2522</td>
<td>1132.9</td>
</tr>
<tr>
<td>30</td>
<td>831</td>
<td>1488</td>
<td>1812</td>
<td>2417</td>
<td>1171</td>
</tr>
<tr>
<td>°C</td>
<td>Ω</td>
<td>Ω</td>
<td>Ω</td>
<td>Ω</td>
<td>Ω</td>
</tr>
<tr>
<td>----</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>35</td>
<td>694</td>
<td>1235</td>
<td>1500</td>
<td>1960</td>
<td>1200</td>
</tr>
<tr>
<td>40</td>
<td>583</td>
<td>1031</td>
<td>1248</td>
<td>1597</td>
<td>1230</td>
</tr>
<tr>
<td>45</td>
<td>491</td>
<td>865</td>
<td>1043</td>
<td>1310</td>
<td>1261</td>
</tr>
<tr>
<td>50</td>
<td>416</td>
<td>729</td>
<td>876</td>
<td>1081</td>
<td>1291</td>
</tr>
<tr>
<td>55</td>
<td>354</td>
<td>616</td>
<td>738</td>
<td>896</td>
<td>1322</td>
</tr>
<tr>
<td>60</td>
<td>302</td>
<td>524</td>
<td>626</td>
<td>746</td>
<td>1353</td>
</tr>
<tr>
<td>65</td>
<td>259</td>
<td>447</td>
<td>532</td>
<td>625</td>
<td>1385</td>
</tr>
<tr>
<td>70</td>
<td>223</td>
<td>383</td>
<td>454</td>
<td>526</td>
<td>1417</td>
</tr>
<tr>
<td>75</td>
<td>192</td>
<td>329</td>
<td>390</td>
<td>444</td>
<td>1450</td>
</tr>
<tr>
<td>80</td>
<td>167</td>
<td>284</td>
<td>335</td>
<td>346</td>
<td>1483</td>
</tr>
<tr>
<td>85</td>
<td>145</td>
<td>246</td>
<td>289</td>
<td>321</td>
<td>1516</td>
</tr>
<tr>
<td>90</td>
<td>127</td>
<td>214</td>
<td>251</td>
<td>275</td>
<td>1549</td>
</tr>
<tr>
<td>95</td>
<td>111</td>
<td>187</td>
<td>218</td>
<td>236</td>
<td>1584</td>
</tr>
<tr>
<td>100</td>
<td>97</td>
<td>163</td>
<td>190</td>
<td>204</td>
<td>1618</td>
</tr>
<tr>
<td>105</td>
<td>88</td>
<td>143</td>
<td>167</td>
<td>176</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>76</td>
<td>126</td>
<td>146</td>
<td>138</td>
<td>1688</td>
</tr>
<tr>
<td>115</td>
<td>111</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>99</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>88</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>135</td>
<td>62</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>2066</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>2148</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>170</td>
<td>1832</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>1868</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>1905</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>1941</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>210</td>
<td>1977</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>220</td>
<td>2049</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>230</td>
<td>2085</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>240</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>°C</td>
<td>S+S Sensor type 1K3 A1 NTC 1 kOhm</td>
<td>S+S Sensor type 1.8K3 A1 NTC 1.8 kOhm</td>
<td>S+S Sensor type 2.2K3 A1 NTC 2.2 kOhm</td>
<td>S+S Sensor type 3.3K3 A1 NTC 3 kOhm</td>
<td>S+S Sensor type Ni1000 DIN</td>
</tr>
<tr>
<td>----</td>
<td>---------------------------------</td>
<td>---------------------------------</td>
<td>---------------------------------</td>
<td>---------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>320</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>330</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>340</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>350</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>360</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>370</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>380</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>390</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>400</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>°C</th>
<th>Ω</th>
<th>Ω</th>
<th>Ω</th>
<th>Ω</th>
<th>Ω</th>
<th>Ω</th>
</tr>
</thead>
<tbody>
<tr>
<td>320</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2191</td>
</tr>
<tr>
<td>330</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2227</td>
</tr>
<tr>
<td>340</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2262</td>
</tr>
<tr>
<td>350</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2297</td>
</tr>
<tr>
<td>360</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2332</td>
</tr>
<tr>
<td>370</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2367</td>
</tr>
<tr>
<td>380</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2401</td>
</tr>
<tr>
<td>390</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2436</td>
</tr>
<tr>
<td>400</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2470</td>
</tr>
</tbody>
</table>
Specified temperature curves, selectable via the Enum eTemperatureCurve

<table>
<thead>
<tr>
<th>°C</th>
<th>Ø</th>
<th>°C</th>
<th>Ø</th>
</tr>
</thead>
<tbody>
<tr>
<td>-50</td>
<td>790.88</td>
<td>-40</td>
<td>830.83</td>
</tr>
<tr>
<td>-30</td>
<td>24500</td>
<td>-20</td>
<td>871.69</td>
</tr>
<tr>
<td>-20</td>
<td>14000</td>
<td>-10</td>
<td>913.48</td>
</tr>
<tr>
<td>0</td>
<td>5200</td>
<td>0</td>
<td>956.24</td>
</tr>
<tr>
<td>10</td>
<td>3330</td>
<td>10</td>
<td>1000</td>
</tr>
<tr>
<td>20</td>
<td>2200</td>
<td>20</td>
<td>1090.65</td>
</tr>
<tr>
<td>25</td>
<td>1800</td>
<td>25</td>
<td>1113.99</td>
</tr>
<tr>
<td>30</td>
<td>1480</td>
<td>30</td>
<td>1137.61</td>
</tr>
<tr>
<td>40</td>
<td>1040</td>
<td>40</td>
<td>1185.71</td>
</tr>
<tr>
<td>50</td>
<td>740</td>
<td>50</td>
<td>1234.97</td>
</tr>
<tr>
<td>60</td>
<td>540</td>
<td>60</td>
<td>1285.44</td>
</tr>
<tr>
<td>70</td>
<td>402</td>
<td>70</td>
<td>1337.14</td>
</tr>
<tr>
<td>80</td>
<td>306</td>
<td>80</td>
<td>1390.12</td>
</tr>
<tr>
<td>90</td>
<td>240</td>
<td>90</td>
<td>1444.39</td>
</tr>
<tr>
<td>100</td>
<td>187</td>
<td>100</td>
<td>1500</td>
</tr>
<tr>
<td>110</td>
<td>149</td>
<td>110</td>
<td>1556.98</td>
</tr>
<tr>
<td>120</td>
<td>118</td>
<td>120</td>
<td>1615.36</td>
</tr>
<tr>
<td>130</td>
<td>95</td>
<td>130</td>
<td>1675.18</td>
</tr>
<tr>
<td>140</td>
<td>77</td>
<td>140</td>
<td>1736.47</td>
</tr>
<tr>
<td>150</td>
<td>64</td>
<td>150</td>
<td>1799.26</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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;
- `rPresentValue`:
- `bErrorGeneral`:
- `byError`:
- `bErrorShortCircuit`:
- `bErrorBrokenSensor`:
- `bErrorHighLimit`:
- `bErrorLowLimit`:
- `bErrorCycleTime`:
- `blInvalidParameter`:
- `eTemperatureSensorMode`:

**NOTE**

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType [490]` = 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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>(rPresentValue)</td>
<td>REAL</td>
</tr>
<tr>
<td>(bErrorGeneral)</td>
<td>BOOL</td>
</tr>
<tr>
<td>(byError)</td>
<td>BYTE</td>
</tr>
<tr>
<td>(bErrorShortCircuit)</td>
<td>BOOL</td>
</tr>
<tr>
<td>(bErrorBrokenSensor)</td>
<td>BOOL</td>
</tr>
<tr>
<td>(bErrorHighLimit)</td>
<td>BOOL</td>
</tr>
<tr>
<td>(bErrorLowLimit)</td>
<td>BOOL</td>
</tr>
<tr>
<td>(bErrorCycleTime)</td>
<td>BOOL</td>
</tr>
<tr>
<td>(bInvalidParameter)</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

**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 \(rPresentValue \geq rHighLimit\). The warning that the upper limit value has been exceeded can only be acknowledged if \(rPresentValue \leq rHighLimit - 1.0\) for a period of 5 seconds.

**bErrorLowLimit**: Warning: lower limit value undershot; goes TRUE if \(rPresentValue \leq rLowLimit\). The warning that the lower limit value has been undershot can only be acknowledged if \(rPresentValue \geq rLowLimit + 1.0\) for a 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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>(rOffset)</td>
<td>REAL</td>
</tr>
<tr>
<td>(rHighLimit)</td>
<td>REAL</td>
</tr>
<tr>
<td>(rLowLimit)</td>
<td>REAL</td>
</tr>
</tbody>
</table>
rOffset: Temperature compensation in Kelvin (-50..+50), \( r_{\text{PresentValue}} = (i_{\text{RawValue}} / 10.0) + r_{\text{Offset}} \). 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 \( r_{\text{PresentValue}} >= r_{\text{HighLimit}} \). 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 \( r_{\text{PresentValue}} <= r_{\text{LowLimit}} \). 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 \( r_{\text{PresentValue}} \) 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 \( r_{\text{PresentValue}} \). 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 [494]). The variable is saved persistently. Preset to 3. eTemperatureSensorMode = eHVACTemperatureSensorMode_ReplacementValue: If bErrorShortCircuit or bErrorBrokenSensor = TRUE, then \( r_{\text{PresentValue}} = r_{\text{ReplacementValue}} \). 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 \( r_{\text{PresentValue}} \). 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 \( r_{\text{PresentValue}} = r_{\text{ReplacementValue}} \). 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 \( r_{\text{PresentValue}} \). 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

### 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` 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 `bError`.

### VAR_INPUT

- `eDataSecurityType` : `E_HVACDataSecurityType`
- `bSetDefault` : `BOOL`
- `bEnable` : `BOOL`
- `iRawValue` : `INT`
- `byStatusRawValue` : `BYTE`
- `bEnableLimitCtrl` : `BOOL`
- `bReset` : `BOOL`
eDataSecurityType: If eDataSecurityType \[490] = 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 \[479] 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/zip/5090272523.zip

If eDataSecurityType \[490] = eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

NOTE
A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType \[490] = 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.

iVarValue: 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 \[494], errors are acknowledged either with bReset or automatically.

VAR_OUTPUT

<table>
<thead>
<tr>
<th>Var Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>rPresentValue</td>
<td>REAL</td>
</tr>
<tr>
<td>bErrorGeneral</td>
<td>BOOL</td>
</tr>
<tr>
<td>byError</td>
<td>BYTE</td>
</tr>
<tr>
<td>bErrorSensor</td>
<td>BOOL</td>
</tr>
<tr>
<td>bErrorHighLimit</td>
<td>BOOL</td>
</tr>
<tr>
<td>bErrorLowLimit</td>
<td>BOOL</td>
</tr>
<tr>
<td>bInvalidParameter</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

rPresentValue: Temperature output variable with one decimal place. The value for rPresentValue is calculated and output according to the following formula:

\[
rPresentValue = ((iRawValue / 10 + rOffset) - rPresentValueOld) / rSmoothFactor + rPresentValueOld;
\]

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 \[494].

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 \[494] and is enabled once the error has been rectified and, depending on the operation mode eTemperatureSensorMode \[494], 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`.\(^494\).

**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`.\(^494\).

**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`.\(^494\). 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`, 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`.\(^494\). 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`.\(^494\).

### VAR_IN_OUT

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rOffset</code></td>
<td>REAL;</td>
</tr>
<tr>
<td><code>rHighLimit</code></td>
<td>REAL;</td>
</tr>
<tr>
<td><code>rLowLimit</code></td>
<td>REAL;</td>
</tr>
<tr>
<td><code>rReplacementValue</code></td>
<td>REAL;</td>
</tr>
<tr>
<td><code>rSmoothFactor</code></td>
<td>REAL;</td>
</tr>
<tr>
<td><code>eTemperatureSensorMode</code></td>
<td>E_HVACTemperatureSensorMode;</td>
</tr>
</tbody>
</table>

**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 (\(\geq 1\)) 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 [4.494]).

- **eHVACTemperatureSensorMode_FaultValue**: 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 \(bReset\).
- **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 \(bReset\).
- **eHVACTemperatureSensorMode_AutoResetReplacementValue**: If \(bErrorShortCircuit\) or \(bErrorBrokenSensor\) = TRUE, then \(rPresentValue = rReplacementValue\). The function block acknowledges itself automatically following correction of the error.
- **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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

### 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:

\[
i_{\text{ConversionFactor}} = 0: \text{rPresentValue} := (i_{\text{RawValue}} / 10 + \text{rOffset}) - \text{rPresentValueOld} / \text{rSmoothFactor} + \text{rPresentValueOld};
\]

\[
i_{\text{ConversionFactor}} = 1: \text{rPresentValue} := (i_{\text{RawValue}} / 100 + \text{rOffset}) - \text{rPresentValueOld} / \text{rSmoothFactor} + \text{rPresentValueOld};
\]

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDataSecurityType</td>
<td>E_HVACDataSecurityType;</td>
</tr>
<tr>
<td>bSetDefault</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bEnable</td>
<td>BOOL;</td>
</tr>
<tr>
<td>iRawValue</td>
<td>INT;</td>
</tr>
<tr>
<td>byStatusRawValue</td>
<td>BYTE;</td>
</tr>
<tr>
<td>iConversionFactor</td>
<td>INT;</td>
</tr>
<tr>
<td>bEnableLimitCtrl</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bReset</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE**

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType [490] = 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

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

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

bEnable: The controller is enabled via a TRUE and is then active.

rW: The setpoint is transferred with the 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 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 = 0
- bDirection = FALSE
- tDelayOn = 4s
- tDelayOff = 6s

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

### 5.1.4.2 FB_HVACI_CtrlStep

- eDataSecurityType
- bSetDefault
- bEnable
- rW
- rX
- eCtrlMode
- iManualStep
- bStepUp
- bStepDown
- bReset
- iNumberOfStepInProfile
- bBlock
- bDirection
- rHysteresisHigh
- rHysteresisLow
- udiSecDelayHigh
- udiSecDelayLow
- udiSecTlHigh
- udiSecTlLow
- IntegralHigh
- IntegralLow
- iStep
- rL_Cmd
- rE
- rW_High
- rW_Low
- b_rW_High
- b_rW_Low
- udiSecRT_DelayHigh
- udiSecRT_DelayLow
- bError
eErrorCode
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 \( rW_{\text{High}} / rW_{\text{Low}} \). Subsequently the integral starts. If the upper or lower limit value \( (\text{rIntegralHigh} / \text{rIntegralLow}) \) is reached at the output \( rI_{\text{Ctrl}} \), then a timing element is started upon whose expiry \( (\text{udiSecRT_DelayHigh} / \text{udiSecRT_DelayLow}) \) the output \( i\text{Step} \) is incremented or decremented; see Program flowchart[115].

An integration time setting of \( \text{udiSecTiHigh}/\text{udiSecTiLow} = 60 \) 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 \( rI_{\text{Ctrl}} \) has the value 5 after one minute. Example: \( b\text{Direction} = \text{FALSE}; \text{udiSecTiLow} = 60; rE = 2; \text{rIntegralLow} = 10 \)

For the remaining control deviation of 2 and the lower integral limit of 10, \( rl_{\text{Ctrl}} = \text{rIntegralLow} \) after 5 minutes. The consequence of this is that a timing element with the delay time \( \text{udiSecDelayLow} \) is started. After its expiry \( (\text{udiSecRT_DelayHigh} / \text{udiSecRT_DelayLow}) \), the output \( i\text{Step} \) is incremented by 1 and the I-transfer element and the time delays are reset.

\[ i\text{Step} = i\text{Step} + 1 \text{ if } (rX < rW_{\text{Low}} \text{ AND } i\text{Step} < i\text{NumberOfStepInProfile} \text{ AND NOT } b\text{Direction}) \]
\[ \text{OR} \]
\[ (rX >= rW_{\text{High}} \text{ AND } i\text{Step} < i\text{NumberOfStepInProfile} \text{ AND NOT } b\text{Direction}) \text{ OR } (i\text{Step} > 0 \text{ AND } b\text{Direction}) \]

The I-transfer element and the internal time delays \( (\text{udiSecDelayLow}, \text{udiSecDelayHigh}) \) are reset after each change of \( i\text{Step} \).

Transfer function of the I-transfer element \( (I_{\text{Ctrl}}) \)

\[ G(s) = \frac{1}{T1 * s} \]
Program flowchart

Behavior of the different variables
The figure shows when each variable is used and how.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDataSecurityType</td>
<td>E_HVACDataSecurityType;</td>
</tr>
<tr>
<td>bSetDefault</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bEnable</td>
<td>BOOL;</td>
</tr>
<tr>
<td>rW</td>
<td>REAL;</td>
</tr>
<tr>
<td>rX</td>
<td>REAL;</td>
</tr>
<tr>
<td>eCtrlMode</td>
<td>E_HVACCtrlMode;</td>
</tr>
<tr>
<td>iManualStep</td>
<td>INT;</td>
</tr>
<tr>
<td>bStepUp</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bStepDown</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bDirection</td>
<td>BOOL;</td>
</tr>
<tr>
<td>iNumberOfStepInProfile</td>
<td>INT;</td>
</tr>
<tr>
<td>bBlock</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bReset</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>

**eDataSecurityType:** If eDataSecurityType [490] = 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 [479] 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/zip/5090272523.zip

If eDataSecurityType [490] = eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

**NOTE**

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType [490] = 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 of *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.

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>iStep</em> : INT;</td>
</tr>
<tr>
<td><em>rI_Ctrl</em> : REAL;</td>
</tr>
<tr>
<td><em>rE</em> : REAL;</td>
</tr>
<tr>
<td><em>rW_High</em> : REAL;</td>
</tr>
<tr>
<td><em>rW_Low</em> : REAL;</td>
</tr>
<tr>
<td><em>b_rW_High</em> : BOOL;</td>
</tr>
<tr>
<td><em>b_rW_Low</em> : BOOL;</td>
</tr>
<tr>
<td><em>udiSecRT_DelayHigh</em> : UDINT;</td>
</tr>
<tr>
<td><em>udiSecRT_DelayLow</em> : UDINT;</td>
</tr>
<tr>
<td><em>bError</em> : BOOL;</td>
</tr>
<tr>
<td><em>eErrorCode</em> : E_HVACErrorCodes;</td>
</tr>
</tbody>
</table>

*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: \( rE = rW - rX \)

\( rW_{\text{High}} \): \( rW_{\text{High}} := rW + rHysteresisHigh \) – 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 [113] or Behavior of different variables [115].

\( rW_{\text{Low}} \): \( rW_{\text{Low}} := rW - rHysteresisLow \) – 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 [113] or Behavior of different variables [115].

\( b_{rW_{\text{High}}} \): \( b_{rW_{\text{High}}} \) goes TRUE if \( rX > rW_{\text{High}} \).

\( b_{rW_{\text{Low}}} \): \( b_{rW_{\text{Low}}} \) goes TRUE if \( rX < rW_{\text{Low}} \).

\( udiSecRT_{\text{DelayHigh}} \): Remaining time of the delay time \( udiSecDelayHigh \).

\( udiSecRT_{\text{DelayLow}} \): 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 HVACErrorCodes [490]). The following error can occur in this function block: HVACErrorCodes_InvalidParam_NumberOfStepInProfile.

To access the enum error number in the PLC \( eErrorCode [490] \) can assigned to a variable of data type WORD. HVACErrorCodes_Error_iNumberOfStepInProfile= 32

**VAR_IN_OUT**

- \( rHysteresisHigh \) : REAL;
- \( rHysteresisLow \) : REAL;
- \( udiSecDelayHigh \) : UDINT;
- \( udiSecDelayLow \) : UDINT;
- \( udiSecTiHigh \) : UDINT;
- \( udiSecTiLow \) : UDINT;
- \( rIntegralHigh \) : REAL;
- \( rIntegralLow \) : REAL;

\( rHysteresisHigh \): Positive value of the upper limit of the control deviation; see Functional description [113] or Behavior of different variables [115]. \( rW_{\text{High}} := rW + rHysteresisHigh \). The variable is saved persistently. Preset to 5.

\( rHysteresisLow \): Positive value of the lower limit of the control deviation; see Functional description [113] or Behavior of different variables [115]. \( rW_{\text{Low}} := rW - rHysteresisLow \). 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 [113] 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 [113] 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 [113] 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 [113] 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.
Programming

**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](#) or [Behavior of different variables](#). 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](#) or [Behavior of different variables](#). The variable is saved persistently. Preset to 15.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

#### 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 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, 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](#).
An integration time setting of $u_{diSecTiHigh}/u_{diSecTiLow} = 60$ 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 $udiSecDelayHigh / udiSecDelayLow$ are activated. After the expiry of the delay times $udiSecRT_DelayHigh / udiSecRT_DelayLow$, the steps of the power step sequence are controlled as follows:

\[
iStep = iStep + 1 \ if
\]

\[
(\ rX < rW_{Low} \ AND \ iStep < iNumberOfStepInProfile \ AND \ NOT \ bDirection)
\]

OR

\[
(\ rX >= rW_{High} \ AND \ iStep < iNumberOfStepInProfile \ AND \ bDirection)
\]

\[
iStep = iStep - 1 \ if
\]

\[
(\ rX < rW_{Low} \ AND \ iStep > 0 \ AND \ bDirection)
\]

OR

\[
(\ rX >= rW_{High} \ AND \ iStep > 0 \ AND \ NOT \ bDirection)
\]

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 ($I_{Ctrl}$)**

\[
G(s) = \frac{1}{T_I \ \ast \ s}
\]
Program flowchart

Temperature °C

rHysteresisHigh
rW
rHysteresisLow

Integral

rIntegralHigh
rIntegralLow

Timer

udlSecET_DelayHigh
udlSecET_DelayLow

iStep

Program flowchart
Behaviour of the different variables

The figure shows when each variable is used and how.

VAR_INPUT

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

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

- Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

**NOTE**

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

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

- **bEnable**: The function block is enabled via TRUE. If bEnable = FALSE, then iStep is set constantly to 0. The check of the variable iNumberOfStepInProfile remains active. If an error occurs, it is displayed with bError = TRUE and can be acknowledged with bReset once the fault has been corrected.

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

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

If bEnable = TRUE AND bError = FALSE AND eCtrlMode = eHVACCtrlMode_Auto AND bBlock = FALSE, then the I-transfer element and the internal timing elements are enabled for the control of the output iStep.

If bEnable = TRUE AND bError = FALSE AND eCtrlMode = eHVACCtrlMode_Auto AND bBlock = TRUE, then the I-transfer element and the internal timing elements are blocked and iStep = iNumberOfStepInProfile.

bReset: Input for acknowledgement of the faults once they have been corrected. Internally the system responds to a rising edge.

VAR_OUTPUT

iStep : INT;
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 \( b_{\text{Enable}} = \text{TRUE}, \ b_{\text{Error}} = \text{FALSE}, \ e_{\text{CtrlMode}} = \text{eHVACCtrlMode}_{-}\text{Auto}, \ b_{\text{Block}} = \text{FALSE} \) AND \( i_{\text{NumberOfStepInProfile}} > 0 \), the output of the I-transfer element \( r_{\text{I}\_\text{Ctrl}} = r_{\text{IntegralHigh}} \) or \( r_{\text{IntegralLow}} \) and one of the delay times \( t_{\text{RemainingTimeDelayHigh}} \) or \( t_{\text{RemainingTimeDelayLow}} = \text{T\#0s} \), then the output \( i_{\text{Step}} \) is controlled as follows:

\[
i_{\text{Step}} = i_{\text{Step}} + 1 \text{ if } (r_X < r_{W\_\text{Low}} \text{ AND } i_{\text{Step}} < i_{\text{NumberOfStepInProfile}} \text{ AND NOT } b_{\text{Direction}}) \text{ OR } (r_X >= r_{W\_\text{High}} \text{ AND } i_{\text{Step}} < i_{\text{NumberOfStepInProfile}} \text{ AND } b_{\text{Direction}}).
\]

\[
i_{\text{Step}} = i_{\text{Step}} - 1 \text{ if } (r_X < r_{W\_\text{Low}} \text{ AND } i_{\text{Step}} > 0 \text{ AND } b_{\text{Direction}}) \text{ OR } (r_X >= r_{W\_\text{High}} \text{ AND } i_{\text{Step}} > 0 \text{ AND NOT } b_{\text{Direction}}), \text{ see Program flowchart [» 121]}
\]

If \( b_{\text{Enable}} = \text{TRUE} \) AND \( b_{\text{Error}} = \text{FALSE} \) AND \( e_{\text{CtrlMode}} = \text{eHVACCtrlMode}_{-}\text{Auto} \) AND \( b_{\text{Block}} = \text{TRUE} \), then the I-transfer element and the internal timing elements are blocked and \( i_{\text{Step}} = i_{\text{NumberOfStepInProfile}} \).

\( r_{\text{I}\_\text{Ctrl}} \): Output of the I-transfer element.

If \( r_{\text{I}\_\text{Ctrl}} >= r_{\text{IntegralHigh}} \) or \( r_{\text{IntegralLow}} \) and \( t_{\text{RemainingTimeDelayHigh}} \) or \( t_{\text{RemainingTimeDelayLow}} = \text{T\#0s} \), then the number of steps is incremented or decremented by 1 via \( i_{\text{Step}} \). 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].

\( r_{E} \): Control deviation with which the internal I-transfer element works: \( r_{E} = r_{W} - r_X \)

\( r_{W\_\text{High}} \): \( r_{W\_\text{High}} := r_{W} + r_{\text{HysteresisHigh}} \) – upper setpoint limit which, when exceeded by \( r_X \), causes the I-transfer element to be activated first, followed by the timing element of the delay time \( udi\text{SecDelayHigh} \); see Functional description [» 119] or Behavior of different variables [» 123].

\( r_{W\_\text{Low}} \): \( r_{W\_\text{Low}} := r_{W} - r_{\text{HysteresisLow}} \) – lower setpoint limit which, when undershot by \( r_X \), causes the I-transfer element to be activated first, followed by the timing element of the delay time \( udi\text{SecDelayLow} \); see Functional description [» 119] or Behavior of different variables [» 123].

\( b_{r_{W\_\text{High}}} \): \( b_{r_{W\_\text{High}}} \) goes \text{TRUE} if \( r_X > r_{W\_\text{High}} \).

\( b_{r_{W\_\text{Low}}} \): \( b_{r_{W\_\text{Low}}} \) goes \text{TRUE} if \( r_X < r_{W\_\text{Low}} \).

\( udi\text{SecRT\_DelayHigh} \): Remaining time of the delay time \( udi\text{SecDelayHigh} \).

\( udi\text{SecRT\_DelayLow} \): Remaining time of the delay time \( udi\text{SecDelayLow} \).

\( b_{\text{Error}} \): This output signals with a \text{TRUE} that there is an error and that there is a wrong parameter on the variable \( i_{\text{NumberOfStepInProfile}} \). \( i_{\text{Step}} \) is set constantly to 0 and the Enum \( e_{\text{ErrorCode}} \) indicates the error code. Once the fault has been corrected the \( b_{\text{Error}} \) message must be acknowledged with \( b_{\text{Reset}} \).

\( e_{\text{ErrorCode}} \): Returns the error number if a \( b_{\text{Error}} \) output is set (see \( \text{E\_HVACErrorCodes} \) [» 490]). The following error can occur in this function block: \( \text{eHVACErrorCodes\_InvalidParam\_NumberOfStepInProfile= 32} \)

To access the enum error number in the PLC \( e_{\text{ErrorCode}} \) [» 490] can assigned to a variable of data type \text{WORD}. \( e_{\text{HVACErrorCodes\_Error\_iNumberOfStepInProfile}} \)

---

**VAR\_IN\_OUT**

\( r_{\text{HysteresisHigh}} \) : \text{REAL};
\( r_{\text{HysteresisLow}} \) : \text{REAL};
\( udi\text{SecDelayHigh} \) : \text{UDINT};
\( udi\text{SecDelayLow} \) : \text{UDINT};
\( udi\text{SecTiHigh} \) : \text{UDINT};
\( udi\text{SecTiLow} \) : \text{UDINT};
\( r_{\text{IntegralHigh}} \) : \text{REAL};
\( r_{\text{IntegralLow}} \) : \text{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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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;
- rTaskCycleTime : TIME;
- rCtrlCycleTime : TIME;
- eCtrlMode : E_HVACCtrlMode;
- rYManual : REAL;
- rInitialValue : REAL;
- bResetController : BOOL;
- bReset : BOOL;

**eDataSecurityType:** If eDataSecurityType [490] := 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 [479] 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/zip/5090272523.zip

If eDataSecurityType [490] := eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.
A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType` := eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.

**NOTE**

**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`).

**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.

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

**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`).

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

**eErrorCode:** Contains the command-specific error code (see `E_HVACErrorCodes`).

**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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
5.1.4.5 **FB_HVACPIDCtrl_Ex**

- **eDataSecurityType**
- **bSetDefault**
- **bEnable**
- **rW**
- **rX**
- **tTaskCycleTime**
- **uiCycleCell**
- **bSetSyncValue**
- **rDeadRange**
- **rSyncValue**
- **bDirection**
- **rKp**
- **rTi**
- **rTv**
- **rTd**
- **rYMin**
- **rYMax**
- **tMinToMax**
- **tMaxToMin**
- **bMaxLimit**
- **bMinLimit**
- **bDecLimit**
- **blncLimit**
- **bActive**
- **bARWactive**
- **b2Ptactive**
- **eErrorCode**

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 following scheme:

<table>
<thead>
<tr>
<th>rY</th>
<th>0,0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rXW</td>
<td>0,0</td>
</tr>
<tr>
<td>bMaxLimit</td>
<td>FALSE</td>
</tr>
<tr>
<td>bMinLimit</td>
<td>FALSE</td>
</tr>
<tr>
<td>bDecLimit</td>
<td>FALSE</td>
</tr>
<tr>
<td>blncLimit</td>
<td>FALSE</td>
</tr>
<tr>
<td>bActive</td>
<td>FALSE</td>
</tr>
<tr>
<td>bARWactive</td>
<td>FALSE</td>
</tr>
<tr>
<td>b2Ptactive</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

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 respective 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:

<table>
<thead>
<tr>
<th>bDirection</th>
<th>( rXW ) (control deviation)</th>
<th>Control direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>( lrX-lrW ) (actual value setpoint)</td>
<td>direct (cooling)</td>
</tr>
<tr>
<td>FALSE</td>
<td>( lrW-lrX ) (setpoint actual value)</td>
<td>indirect (heating)</td>
</tr>
</tbody>
</table>

**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 $I_r$, $K_p$, $T_n$, $T_v$ and $T_d$ are set to 0.0 or t#0s, then the controller has by definition a 2-point behavior.

The dead range $r\text{DeadRange}$ 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 $r\text{DeadRange}$, while switching to the minimum value always takes place when the control deviation $rXW$ is smaller than the negative half of the hysteresis value $r\text{DeadRange}$:

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

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

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

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

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

bEnable: Controller activation. At the moment of activation the controller reacts directly to the control deviation without internal synchronization to a value.

rW: Setpoint

rX: Actual value

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

uiCycleCall: Call cycle of the function block as a multiple of the cycle time. A zero entry is automatically 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)

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 \( \text{E_HVACErrorCodes[490]} \)).

**VAR_IN_OUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>rDeadRange</td>
<td>REAL</td>
</tr>
<tr>
<td>rSyncValue</td>
<td>REAL</td>
</tr>
<tr>
<td>bDirection</td>
<td>BOOL</td>
</tr>
<tr>
<td>rKp</td>
<td>REAL</td>
</tr>
<tr>
<td>tTi</td>
<td>TIME</td>
</tr>
<tr>
<td>tTv</td>
<td>TIME</td>
</tr>
<tr>
<td>tTd</td>
<td>TIME</td>
</tr>
<tr>
<td>rYMin</td>
<td>REAL</td>
</tr>
<tr>
<td>rYMax</td>
<td>REAL</td>
</tr>
<tr>
<td>tMinToMax</td>
<td>TIME</td>
</tr>
<tr>
<td>tMaxToMin</td>
<td>TIME</td>
</tr>
</tbody>
</table>

**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: $t_{\text{MaxToMin}}$ in seconds in reference to a change from $rY_{\text{Max}}$ to $rY_{\text{Min}}$. The variable is saved persistently. Preset to 0 s.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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 $i_{\text{Step}}$.

All information or parameters relevant to system control are summarized in the power range table ARRAY $\text{arrPowerRangeTable}$. The power range table is a two-dimensional array with the structure $\text{ST_HVACPowerRange}$. 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 $i_{\text{Profile}}$. The field range 0 to g_iMaxNumberOfSteps represents the lines. This range is called the step and is addressed via the input variable $i_{\text{Step}}$.

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 $\text{ST_HVACPowerRange}$ needed for system control. The parameters indicated here are transferred via the output structures $\text{stI_Ctrl}$ and $\text{stAggregate1-6}$ 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 $i_{\text{NumberOfStepInProfile}}$. The number depends on the entries in the power range table and on the specification in the selected profile $si_{\text{CurrentProfile}}$. The selected profile is checked internally from step 1 up to the step in which all variables of the structure $\text{ST_HVACPowerRange}$ have the value 0. $i_{\text{NumberOfStepInProfile}}$ is always intended only for the specified profile. $i_{\text{NumberOfStepInProfile}}$ can thus be used to limit the steps in a profile for other function blocks such as for the input $i_{\text{NumberOfStepInProfile}}$ of FB_HVACI_CtrlStep [112], which could control the individual steps of the FB_HVACPowerRangeTable via its output $i_{\text{Step}}$. The step 0 is not taken into account when evaluating the number of steps via $i_{\text{NumberOfStepInProfile}}$, 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 $\text{stAggregate1-6}$ 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 $\text{stI_Ctrl}$. 
### 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` 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.
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[497]` (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;
   stI_Ctrl.rIntegralLow := 10;
   stI_Ctrl.udiSecDelayLow := 300;
   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[497]` (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[2,1].iAggregateStep;
   stAggregate1.rY_Max := arrPowerRangeTable[2,1].rY_Max;
   stAggregate1.rY_Min := arrPowerRangeTable[2,1].rY_Min;
   stAggregate1.bBlock := arrPowerRangeTable[2,1].bBlock;
   stAggregate2.iAggregateStep := arrPowerRangeTable[2,2].iAggregateStep;
   stAggregate2.rY_Max := arrPowerRangeTable[2,2].rY_Max;
   stAggregate2.rY_Min := arrPowerRangeTable[2,2].rY_Min;
   stAggregate2.bBlock := arrPowerRangeTable[2,2].bBlock;
   stI_Ctrl.rIntegralHigh := arrPowerRangeTable[2,2].rIntegralHigh;
   stI_Ctrl.rIntegralLow := arrPowerRangeTable[2,2].rIntegralLow;
   stI_Ctrl.udiSecDelayHigh := arrPowerRangeTable[2,2].udiSecDelayHigh;
   stI_Ctrl.udiSecDelayLow := arrPowerRangeTable[2,2].udiSecDelayLow;
   stI_Ctrl.rIntegralLow := 10;
   stI_Ctrl.udiSecDelayLow := 10;
   stI_Ctrl.udiSecDelayHigh := 300;
   stI_Ctrl.udiSecDelayLow := 300;
   Contents of the output structures `stAggregate1`, `stAggregate2` and `stI_Ctrl`. The variables of the output structures `stAggregate3-6` have the value 0.
The column \textbf{iCurrentProfile = 3} represents the third profile of the power range table \textit{arrPowerRange}. In this profile the output \textit{iNumberOfStepInProfile} has the value 3, because no variable value in the structure \textit{ST_HVACPowerRange [497]} (marked red) is greater than 0 in the line \textit{iCurrentStep = 4}. If one assumes that the current power step \textit{iCurrentStep} has the value 3, then the variable values are output as follows on the output structures:

\begin{verbatim}
arrPowerRangeTable[iCurrentProfile,iCurrentStep];
\end{verbatim}

Contents of the output structures \textit{stAggregate3}, \textit{stAggregate4}, \textit{stAggregate5} and \textit{stl_Ctrl}. The variables of the output structures \textit{stAggregate1}, \textit{stAggregate2} and \textit{stAggregate6} have the value 0.

\begin{verbatim}
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;
\end{verbatim}

Output structure \textit{stl_Ctrl} adjusted to the example
Output structures stAggregate1-6 adjusted to the example

<table>
<thead>
<tr>
<th>iCurrentStep</th>
<th>stAggregate1</th>
<th>stAggregate2</th>
<th>stAggregate3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 \[\textbf{[490]}\] = 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 \[\textbf{[479]}\] 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/zip/5090272523.zip](https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip)

If eDataSecurityType \[\textbf{[490]}\] = eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

### NOTE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType \[\textbf{[490]}\] = 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[P-IS], udiSecDelayHigh, and arrPowerRangeTable[P-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 \[\textbf{[490]}\]). 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 \[\textbf{[490]}\]). 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_iMinNumberOfProfiles` 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` 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` 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` via the output structure `stI_Ctrl`.

**iCurrentProfile**: Indicates the current profile with which the function block from the power range table `arrPowerRangeTable` 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` is working. The most current step is always the highest step viewed from 1 upwards. 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_HVAC_Ctrl[496]** for the function block **FB_HVAC_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.rY_Min**
- **stl_Ctrl.rY_Max**
- **stl_Ctrl.iAggregateStep**
- **stl_Ctrl.bBlock**

... and similar for **stAggregate2**.

The values parameterized in the power range table are then output via the output structure **arrPowerRangeTable**. If the value of the variable **iCurrentProfile = 2** 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**.

If the value of the variable **arrPowerRangeTable [8, 5]**.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**:

... and similar for **stAggregate2**.

- **stAggregate1.rY_Max**
- **stAggregate1.rY_Min**
- **stAggregate1.iAggregateStep**
- **stAggregate1.bBlock**

... and similar for **stAggregate2**.

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**:

... and similar for **stAggregate2**.

- **stAggregate1.rY_Max**
- **stAggregate1.rY_Min**
- **stAggregate1.iAggregateStep**
- **stAggregate1.bBlock**

... and similar for **stAggregate2**.

If **iCurrentStep = 3** and **arrPowerRangeTable [8, 3]**.iAggregate = 2, then this means that the following values are output via the output structures **stAggregate1** and **stAggregate2**:

... and similar for **stAggregate2**.

- **stAggregate1.rY_Max**
- **stAggregate1.rY_Min**
- **stAggregate1.iAggregateStep**
- **stAggregate1.bBlock**

... and similar for **stAggregate2**.
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 [490]).

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 [497] 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 or equal to 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_iNumberOfStepInProfile: Error when checking the specified 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 [490] can assigned to a variable of data type WORD.

eHVACErrorCodes_InvalidParam_iStep = 31, eHVACErrorCodes_InvalidParam_iNumberOfStepInProfile = 32, eHVACErrorCodes_InvalidParam_iNumberOfAggregates = 33, eHVACErrorCodes_InvalidParam_iAggregateSteps = 34, eHVACErrorCodes_InvalidParam_iNumberOfProfiles = 35, eHVACErrorCodes_InvalidParam_iProfile = 36, eHVACErrorCodes_InvalidParam_iAggregate = 39, eHVACErrorCodes_InvalidParam_udiSecDelayUp = 40, eHVACErrorCodes_InvalidParam_udiSecDelayDown = 41

iErrorPosArrayProfile: The variable iErrorPosArrayProfile 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
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:

```c
bError = TRUE;
eErrorCode = eHVACErrorCodes_InvalidParam_iNumberOfStepInProfile;
iErrorPosArrayProfile = 2;
```

No variable value of the structure `ST_HVACPowerRange [497]` 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:

```c
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:

```c
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 [497]`. 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 [497]` 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 [497]` 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 [497]

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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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.

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 \textit{iNumberOfSequences} in the function blocks. In addition each function block is given its own number. In the sequence \textit{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 \textit{FB_HVACPIDEnergyRecovery} 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 \textit{FB_HVACPIDMixedAir}. The parameter \textit{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 \textit{eHVACSequenceCtrlMode} to 4. As a result the pre-heater or the sequence controller \textit{FB_HVACPIDPreHeating} respectively is activated. If the control variable is too high, the sequence is reversed until the cooler is activated by means of \textit{FB_HVACPIDCooling} 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 \textit{rDeadRange}.

### Table of operation modes

A further special feature of the sequence controller is its control by the Enum \textit{E_HVACSequenceCtrlMode}. Not only is the controller enabled by the Enum \textit{E_HVACSequenceCtrlMode}, 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 \textit{E_HVACSequenceCtrlMode} as illustrated in the table.

<table>
<thead>
<tr>
<th>Value of: \textit{E_HVACSequenceCtrlMode}</th>
<th>0 (Stop)</th>
<th>1 (On)</th>
<th>2 (night cooling)</th>
<th>3 (support operation)</th>
<th>4 (overheating protection)</th>
<th>5 (night cooling and overheating protection)</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{FB_HVACMasterSequenceCtrl} Master controller</td>
<td>disabled 0 %</td>
<td>Enable</td>
<td>disabled 0 %</td>
<td>Enable (rY=\text{Supply air max. temp.})</td>
<td>Enable (rY=\text{Supply air min. temp.})</td>
<td>Enable (rY=\text{Supply air min. temp.})</td>
</tr>
<tr>
<td>\textit{FB_HVACPIDPreHeating} Preheater</td>
<td>disabled 0 %</td>
<td>Enable</td>
<td>disabled 0 %</td>
<td>Enable 0 % - 100 %</td>
<td>disabled 0 %</td>
<td>disabled 0 %</td>
</tr>
<tr>
<td>\textit{FB_HVACPIDReHeating} Reheater</td>
<td>disabled 0 %</td>
<td>Enable</td>
<td>disabled 0 %</td>
<td>disabled 0 %</td>
<td>disabled 0 %</td>
<td>disabled 0 %</td>
</tr>
<tr>
<td>\textit{FB_HVACPIDCooling} Cooler</td>
<td>disabled 0 %</td>
<td>Enable</td>
<td>disabled 0 %</td>
<td>disabled 0 %</td>
<td>Enable 0 % - 100 %</td>
<td>Enable 0 % - 100 %</td>
</tr>
<tr>
<td>\textit{FB_HVACPIDEnergyRecovery} Heat recovery</td>
<td>disabled 0 %</td>
<td>Enable</td>
<td>disabled 0 %</td>
<td>disabled 0 %</td>
<td>disabled 0 %</td>
<td>disabled 0 %</td>
</tr>
<tr>
<td>\textit{FB_HVACPIDMixedAir} Mixed air dampers</td>
<td>disabled 0 %</td>
<td>Enable</td>
<td>max. outside air rate 100 %</td>
<td>0 % air circulation only</td>
<td>0 % air circulation only</td>
<td>max. outside air rate 100 %</td>
</tr>
</tbody>
</table>
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. It can be used for cascades of vessels and in sequences of refrigerating machines or recooling plants for controlling the power stages. For each power stage an instance of the function block is used.

FB_HVAC2PointCtrlSequence can be used in a control sequence with other sequence controllers from TcHVAC.lib.

Depending on the application several actuators (stages) are used for reaching a controlled variable, which operate in a so-called control sequence. In the control sequence shown below four stages are enabled via the respective output \( bOut = \text{TRUE of } FB_{-}HVAC2PointCtrlSequence \). 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 \( bOut = \text{TRUE or FALSE, depending on the control direction.} \)

Transfer function of the internal I-transfer element

\[
G(s) = \frac{1}{T_i \cdot s}
\]

Through an internal AND link of the I-transfer element consisting of \( (tCtrl_{-}I\_Ti\_HighLimit, tCtrl_{-}I\_Ti\_LowLimit, rCtrl_{-}I\_HighLimit, rCtrl_{-}I\_LowLimit, st2PointCtrlSequence.rCtrl_{-}I\_Out) \) and the delay times \( tDelayIncreaseSequence/tDelayDecreaseSequence \) switching in the control sequence to the right or left is controlled via \( iCurrentSequence \). The active sequence controller is indicated via \( bActiveCtrl = \text{TRUE (iCurrentSequence = iMyNumberInSequence).} \)

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**

iNumberOfSequences = 4

![Diagram showing the behavior of four FB_HVAC2PointCtrlSequence in a control sequence](image)

- rX > (rN + rW_HighLimit):
  - iCurrentSequence = iCurrentSequence - 1

- rX < (rN - rW_LowLimit):
  - iCurrentSequence = iCurrentSequence + 1
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

VAR_INPUT

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

NOTE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType[490]= 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[479] 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/zip/5090272523.zip

If eDataSecurityType[490]= eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

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 iTi_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). 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**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bOut</td>
<td>Output of the 2-point sequence controller.</td>
</tr>
<tr>
<td>bActiveCtrl</td>
<td></td>
</tr>
<tr>
<td><em>rW_Max</em></td>
<td></td>
</tr>
<tr>
<td><em>rW_Min</em></td>
<td></td>
</tr>
<tr>
<td>rE</td>
<td></td>
</tr>
<tr>
<td>e2PointCtrlState</td>
<td></td>
</tr>
<tr>
<td>st2PointCtrlSequence</td>
<td></td>
</tr>
<tr>
<td>bError</td>
<td></td>
</tr>
<tr>
<td>eErrorCode</td>
<td></td>
</tr>
</tbody>
</table>

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(487)).

st2PointCtrlSequence: The structure indicates various states, inputs and outputs of the function block (see ST_HVAC2PointCtrlSequence(494)). 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: \( \text{st2PointCtrlSequence.rW\_Max} := rW + rW\_HighLimit \) – 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 [\textbf{149}] in this document.

st2PointCtrlSequence.rW_Min: \( \text{st2PointCtrlSequence.rW\_Min} := rW - rW\_LowLimit \) – 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 [\textbf{149}] 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. \( \text{st2PointCtrlSequence.rCtrl\_I\_HighLimit} = rCtrl\_I\_HighLimit \)

st2PointCtrlSequence.rCtrl_I_LowLimit: Lower limit value at which the integration of the internal I-transfer element is stopped. \( \text{st2PointCtrlSequence.rCtrl\_I\_Low} = rCtrl\_I\_LowLimit \times (-1) \)

st2PointCtrlSequence.rCtrl_I_Out: Output of the internal I-transfer element. If \( \text{st2PointCtrlSequence.rCtrl\_I\_Out} = \text{st2PointCtrlSequence.rCtrl\_I\_HighLimit} \) OR \( \text{st2PointCtrlSequence.rCtrl\_I\_LowLimit} \) AND either \( \text{st2PointCtrlSequence.tRemainingTimeIncreaseSequence} \) OR \( \text{st2PointCtrlSequence.tRemainingTimeDecreaseSequence} = \text{T\#0s} \) AND \( b\_Active = \text{TRUE} \), 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 \text{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 \( b\_Reset \). The \( eErrorCode \) enum indicates the error number. If \( bError \) is \text{TRUE}, output \( bOut \) becomes \text{FALSE}.

\[
\text{If } bEnable = \text{FALSE OR } bError = \text{TRUE OR (e2PointCtrlMode = eHVAC2PointCtrlMode\_On\_BMS OR eHVAC2PointCtrlMode\_On\_OP OR eHVAC2PointCtrlMode\_Off\_BMS OR eHVAC2PointCtrlMode\_Off\_OP), then the automatic switchover of the controller from the active controller to the neighboring controller on the left or right in the control sequence remains in operation. The switching mode is therefore always active. The active function block is displayed with } bActiveCtrl = \text{TRUE. Depending on the control deviation } rE \text{ up- or down-switching of the sequence via } iCurrentSequence \text{ is directly executed. If } rX >= rW \text{, then } iCurrentSequence = iCurrentSequence - 1. \text{ If } rX < rW \text{, then } iCurrentSequence = iCurrentSequence + 1.
\]

\( eErrorCode \): returns the error number if an output \( bError \) is set (E_HVACErrorCodes [\textbf{490}]

\( E\_HVACErrorCodes \)). 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 [149] can assigned to a variable of data type WORD.
eHVACErrorCodes_Error_iNumberOfSequences = 27 eHVACErrorCodes_Error_iMyNumberInSequence = 28 eHVACErrorCodes_Error_iCurrentSequences = 29

VAR_IN_OUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>rW_HighLimit</td>
<td>REAL;</td>
</tr>
<tr>
<td>rW_LowLimit</td>
<td>REAL;</td>
</tr>
<tr>
<td>tDelayIncreaseSequence</td>
<td>TIME;</td>
</tr>
<tr>
<td>tDelayDecreaseSequence</td>
<td>TIME;</td>
</tr>
<tr>
<td>tCtrl_I_Ti_HighLimit</td>
<td>TIME;</td>
</tr>
<tr>
<td>tCtrl_I_Ti_LowLimit</td>
<td>TIME;</td>
</tr>
<tr>
<td>rCtrl_I_HighLimit</td>
<td>REAL;</td>
</tr>
<tr>
<td>rCtrl_I_LowLimit</td>
<td>REAL;</td>
</tr>
<tr>
<td>iCurrentSequence</td>
<td>INT;</td>
</tr>
</tbody>
</table>

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 [149] 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 [149] 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 [149] 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 [149] 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 [149] 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 [149] 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 \( st2PointCtrlSequence.tRemainingTimeDecreaseSequence = T\#0s \) 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, \( i_{\text{CurrentSequence}} \) 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. \( i_{\text{CurrentSequence}} \) must be written for one PLC cycle and must be > 0 and \( \leq i_{\text{NumberOfSequences}} \).

If \( \text{bEnable} = \text{FALSE} \) OR \( \text{bError} = \text{TRUE} \) OR \( (\text{e2PointCtrlMode} = \text{eHVAC2PointCtrlMode\_On\_BMS} \) OR \( \text{eHVAC2PointCtrlMode\_On\_OP} \) OR \( \text{eHVAC2PointCtrlMode\_Off\_BMS} \) OR \( \text{eHVAC2PointCtrlMode\_Off\_OP} \)), 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 \( \text{bActiveCtrl} = \text{TRUE} \). Depending on the control deviation \( rE \) up- or down-switching of the sequence via \( i_{\text{CurrentSequence}} \) is directly executed. If \( rX \geq rW \), then \( i_{\text{CurrentSequence}} = i_{\text{CurrentSequence}} - 1 \). If \( rX < rW \), then \( i_{\text{CurrentSequence}} = i_{\text{CurrentSequence}} + 1 \).

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

#### 5.1.4.7.2 FB\_HVACBasicSequenceCtrl

Unlike the sequence controllers FB\_HVACPIDCooling [162], FB\_HVACDehumidify [165], FB\_HVACEnergyRecovery [168], FB\_HVACPIDHumidify [171], FB\_HVACPIDMixedAir [175], FB\_HVACPIDPreHeating [178] and FB\_HVACPIDReHeating [181], 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` := eHVACDataSecurityType_Persistent, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

**NOTE**

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

- **bSetDefault**: If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.
- **bEnable**: Input variable for enabling the controller. The controller is 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`).
- **rYManual**: Manual value.
- **rInitialValue**: The restart behavior of the controller is influenced by `rInitialValue`.
- **bResetController**: A positive edge on the input `bResetController` resets the PID controller.
- **bReset**: Acknowledge input in the event of a fault.
**VAR_OUTPUT**

- **rY**: 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` [490]).
- **bInvalidParameter**: TRUE if an error has occurred during the plausibility check. The message must be acknowledged with `bReset`.

**VAR_IN_OUT**

- **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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

#### 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

```plaintext
VAR_INPUT
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault : BOOL;
eModeSeqCtrl : E_HVACSequenceCtrlMode;
```
**eDataSecurityType**: If `eDataSecurityType[490] := 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[479]` 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/zip/5090272523.zip

If `eDataSecurityType[490] := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

### NOTE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType[490] := 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;
- `bAWactive` : 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 \( b\text{MaxLimit} \) is TRUE if the output \( rY \) has reached the value \( rY\text{Max} \).

bMinLimit: The output \( b\text{MinLimit} \) is TRUE if the output \( rY \) has reached the value \( rY\text{Min} \).

bActive: \( b\text{Active} \) is TRUE if the controller is active and enabled.

bARWactive: \( b\text{ARWactive} \) is TRUE if the integral component of the controller has reached the lower or upper control variable limit.

eState: State of the controller (see \( \text{HVACState} \) [\text{494}]).

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

eErrorCode: Contains the command-specific error code (see \( \text{HVACErrorCode} \) [\text{490}]).

bInvalidParameter: TRUE if an error has occurred during the plausibility check. The message must be acknowledged with \( b\text{Reset} \).

VAR_IN_OUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>rDeadRange</td>
<td>REAL</td>
</tr>
<tr>
<td>rInitialValue</td>
<td>REAL</td>
</tr>
<tr>
<td>rKp</td>
<td>REAL</td>
</tr>
<tr>
<td>tTi</td>
<td>TIME</td>
</tr>
<tr>
<td>tTv</td>
<td>TIME</td>
</tr>
<tr>
<td>tTd</td>
<td>TIME</td>
</tr>
<tr>
<td>rYMin</td>
<td>REAL</td>
</tr>
<tr>
<td>rYMax</td>
<td>REAL</td>
</tr>
</tbody>
</table>

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 \( r\text{InitialValue} \) (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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDataSecurityType</td>
<td>E_HVACDataSecurityType;</td>
</tr>
<tr>
<td>bSetDefault</td>
<td>BOOL;</td>
</tr>
<tr>
<td>eModeSeqCtrl</td>
<td>E_HVACSequenceCtrlMode;</td>
</tr>
<tr>
<td>iNumberOfSequences</td>
<td>INT;</td>
</tr>
<tr>
<td>iMyNumberInSequence</td>
<td>INT;</td>
</tr>
<tr>
<td>rW</td>
<td>REAL;</td>
</tr>
<tr>
<td>rX</td>
<td>REAL;</td>
</tr>
<tr>
<td>tTaskCycleTime</td>
<td>TIME;</td>
</tr>
<tr>
<td>tCtrlCycleTime</td>
<td>TIME;</td>
</tr>
<tr>
<td>eCtrlMode</td>
<td>E_HVACCtrlMode;</td>
</tr>
<tr>
<td>rYManual</td>
<td>REAL;</td>
</tr>
<tr>
<td>bResetController</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bReset</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>

**eDataSecurityType**: If `eDataSecurityType` \[490\] = `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` \[479\] 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/zip/5090272523.zip

If `eDataSecurityType` \[490\] = `eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.
A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType` := `eHVACDataSecurityType_Persistent`. It would lead to early wear of the flash memory.

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

**eModeSeqCtrl:** Among other things, control is enabled via this enumeration (see `E_HVACSequenceCtrlMode`). In addition, the operation mode of the VAC system is transmitted to the controller blocks in the sequence. See also Table operation modes.

**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`). Manual or automatic operation mode.

**rYManual:** Manual value that is set at the output `rY` if `eCtrlMode = eHVACCtlMode_Manual`.

**bResetController:** A positive edge on the input `bResetController` resets the PID controller.

**bReset:** Acknowledge input in the event of a fault.

### VAR_OUTPUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rY</code></td>
<td>REAL</td>
</tr>
<tr>
<td><code>rXW</code></td>
<td>REAL</td>
</tr>
<tr>
<td><code>bMaxLimit</code></td>
<td>BOOL</td>
</tr>
<tr>
<td><code>bMinLimit</code></td>
<td>BOOL</td>
</tr>
<tr>
<td><code>bActive</code></td>
<td>BOOL</td>
</tr>
<tr>
<td><code>bARWactive</code></td>
<td>BOOL</td>
</tr>
<tr>
<td><code>eState</code></td>
<td><code>E_HVACState</code></td>
</tr>
<tr>
<td><code>bError</code></td>
<td>BOOL</td>
</tr>
<tr>
<td><code>eErrorCode</code></td>
<td><code>E_HVACErrorCodes</code></td>
</tr>
<tr>
<td><code>bInvalidParameter</code></td>
<td>BOOL</td>
</tr>
</tbody>
</table>

- **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`).
- **bError:** This output indicates with a TRUE that there is an error.
- **eErrorCode:** `E_HVACErrorCodes` (see `E_HVACErrorCodes`).
bInInvalidParameter: TRUE if an error has occurred during the plausibility check. The message must be acknowledged with bReset.

**VAR_IN_OUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>rDeadRange</td>
<td>REAL;</td>
</tr>
<tr>
<td>rInitialValue</td>
<td>REAL;</td>
</tr>
<tr>
<td>rKp</td>
<td>REAL;</td>
</tr>
<tr>
<td>tTi</td>
<td>TIME;</td>
</tr>
<tr>
<td>tTv</td>
<td>TIME;</td>
</tr>
<tr>
<td>tTd</td>
<td>TIME;</td>
</tr>
<tr>
<td>rDeadBand</td>
<td>REAL;</td>
</tr>
<tr>
<td>rYMin</td>
<td>REAL;</td>
</tr>
<tr>
<td>rYMax</td>
<td>REAL;</td>
</tr>
<tr>
<td>iCurrentSequence</td>
<td>INT;</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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 and FB_HVACPIDDehumidify, act on the cooler valve. First of all the control signal \( rY \) is relayed by FB_HVACPIDCooling 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 := eHVACDataSecurityType_Persistent, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.
A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted the saved data are automatically read back from the flash into the RAM.

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

If eDataSecurityType [490] := eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

<table>
<thead>
<tr>
<th>NOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType [490] := eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>rY, rXW        : REAL;</td>
</tr>
<tr>
<td>bMaxLimit      : BOOL;</td>
</tr>
<tr>
<td>bMinLimit      : BOOL;</td>
</tr>
<tr>
<td>bActive        : BOOL;</td>
</tr>
<tr>
<td>bARWactive     : BOOL;</td>
</tr>
<tr>
<td>eState         : E_HVACState;</td>
</tr>
<tr>
<td>bError         : BOOL;</td>
</tr>
<tr>
<td>eErrorCode     : E_HVACErrorCodes;</td>
</tr>
<tr>
<td>bInvalidParameter : BOOL;</td>
</tr>
</tbody>
</table>

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 $E_{HVACErrorCodes}$[490]).

bInvalidParameter: TRUE if an error has occurred during the plausibility check. The message must be acknowledged with $bReset$.

**VAR_IN_OUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>rDeadRange</td>
<td>REAL</td>
</tr>
<tr>
<td>rInitialValue</td>
<td>REAL</td>
</tr>
<tr>
<td>rKp</td>
<td>REAL</td>
</tr>
<tr>
<td>tTi</td>
<td>TIME</td>
</tr>
<tr>
<td>tTv</td>
<td>TIME</td>
</tr>
<tr>
<td>tTd</td>
<td>TIME</td>
</tr>
<tr>
<td>rDeadBand</td>
<td>REAL</td>
</tr>
<tr>
<td>rYMin</td>
<td>REAL</td>
</tr>
<tr>
<td>rYMax</td>
<td>REAL</td>
</tr>
<tr>
<td>iCurrentSequence</td>
<td>INT</td>
</tr>
</tbody>
</table>

$rDeadRange$: 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

#### 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;
**eDataSecurityType:** If \( eDataSecurityType[490] = eHVACDataSecurityType_Persistent \), 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 \( FB_{HVACPersistentDataHandling}[479] \) 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/zip/5090272523.zip

If \( eDataSecurityType[490] = eHVACDataSecurityType_Idle \) the persistently declared variables are not saved in a fail-safe manner.

**NOTE**

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if \( eDataSecurityType[490] = eHVACDataSecurityType_Persistent \). 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_INPUT**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>rYManual</td>
<td>REAL</td>
</tr>
<tr>
<td>bResetController</td>
<td>BOOL</td>
</tr>
<tr>
<td>bReset</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

**VAR_OUTPUT**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>rY</td>
<td>REAL</td>
</tr>
<tr>
<td>rXW</td>
<td>REAL</td>
</tr>
<tr>
<td>bMaxLimit</td>
<td>BOOL</td>
</tr>
<tr>
<td>bMinLimit</td>
<td>BOOL</td>
</tr>
<tr>
<td>bActive</td>
<td>BOOL</td>
</tr>
<tr>
<td>bARWactive</td>
<td>BOOL</td>
</tr>
<tr>
<td>eState</td>
<td>E_HVACState</td>
</tr>
</tbody>
</table>
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** [490]).

**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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

---

### 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 including a humidification controller and a dehumidification controller.
This function block is a special application of a PID controller. The humidity controller is only enabled if the function block input `bHumidistat` is TRUE.

**VAR_INPUT**

- `eDataSecurityType` : E_HVACDataSecurityType;
- `bSetDefault` : BOOL;
- `eModeSeqCtrl` : E_HVACSequenceCtrlMode;
- `iNumberOfSequences` : INT;
- `iMyNumberInSequence` : INT;
- `rW` : REAL;
- `rX` : REAL;
- `tTaskCycleTime` : TIME;
- `tCtrlCycleTime` : TIME;
- `bHumidistat` : BOOL;
- `eCtrlMode` : E_HVACCtrlMode;
- `rYManual` : REAL;
- `bResetController` : BOOL;
- `bReset` : BOOL;

**eDataSecurityType:** If `eDataSecurityType[490]` := 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[479]` 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/zip/5090272523.zip

If `eDataSecurityType[490]` := eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

---

**NOTE**

A cyclically changing variable must never be linked with the `IN_OUT` variable of a function block, if `eDataSecurityType[490]` := 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_HVACErrCodes;
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_HVACErrCodes (see E_HVACErrCodes [490]).

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;
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.

The restart behavior of the controller is influenced by $rInitialValue$. The variable is saved persistently. Preset to 0.

The proportional factor amplification. The variable is saved persistently. Preset to 0.

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.

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.

Damping time. The variable is saved persistently. Preset to 0 s.

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.

Lower limit of the working range of the controller. The variable is saved persistently. Preset to 0.

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

Number of the active controller from the sequence.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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 \( r_{\text{OutsideTemp}} \). The variable \( r_{\text{ReturnAirTemp}} \) is for the exhaust air or room temperature. The \textit{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 [449].

The maximum and minimum outside air rates can be set with the parameters \( rY_{\text{Max}} \) and \( rY_{\text{Min}} \).

\textbf{VAR\_INPUT}

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDataSecurityType</td>
<td>E_HVACDataSecurityType;</td>
</tr>
<tr>
<td>bSetDefault</td>
<td>BOOL;</td>
</tr>
<tr>
<td>eModeSeqCtrl</td>
<td>E_HVACSequenceCtrlMode;</td>
</tr>
<tr>
<td>iNumberOfSequences</td>
<td>INT;</td>
</tr>
<tr>
<td>iMyNumberInSequence</td>
<td>INT;</td>
</tr>
<tr>
<td>rW</td>
<td>REAL;</td>
</tr>
<tr>
<td>rX</td>
<td>REAL;</td>
</tr>
<tr>
<td>tTaskCycleTime</td>
<td>TIME;</td>
</tr>
<tr>
<td>tCtrlCycleTime</td>
<td>TIME;</td>
</tr>
<tr>
<td>rOutsideTemp</td>
<td>REAL;</td>
</tr>
<tr>
<td>rReturnAirTemp</td>
<td>REAL;</td>
</tr>
<tr>
<td>eCtrlMode</td>
<td>E_HVACCtrlMode;</td>
</tr>
</tbody>
</table>
eDataSecurityType: If eDataSecurityType [490] := 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 [479] 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/zip/5090272523.zip

If eDataSecurityType [490] := eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

NOTE
A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType [490] := 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_HVACCtlMode [490]). Manual or automatic operation mode.

rYManual: Manual value that is set at the output rY if eCtrlMode = eHVACCtlMode_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 [490]).

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 t(Ti) time is set, the faster the controller corrects. The control loop becomes unstable if the time is too short. Larger t(Ti)-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 \textit{rDeadband} (See Example \hyperlink{Example507}{507}). The variable is saved persistently. Preset to 0.

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

\textbf{iCurrentSequence:} Number of the active controller from the sequence.

\section*{Requirements}

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

\subsection*{5.1.4.7.9 \textit{FB\textunderscore 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 \textit{rYDehumidify}. As soon as the signal of \textit{rYDehumidify} is greater than zero, the preheater is disabled and the value of \textit{iCurrentSequence} is incremented by one. As a result of this the reheater \textit{FB\textunderscore HVACPIDReHeating} is activated in place of the preheater. If the value of \textit{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).

\begin{verbatim}
VAR_INPUT
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault : BOOL;
eModeSeqCtrl : E_HVACSequenceCtrlMode;
iNumberOfSequences : INT;
iMyNumberInSequence : INT;
rW : REAL;
\end{verbatim}
rX : REAL;
tTaskCycleTime : TIME;
tCtrlCycleTime : TIME;
rYDehumidify : REAL;
eCtrlMode : E_HVACCtrlMode;
rYManual : REAL;
rYDehumidify : REAL;
bResetController : BOOL;
bReset : BOOL;

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

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

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

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

eModeSeqCtrl : Among other things, control is enabled via this enum (see `E_HVACSequenceCtrlMode`). In addition, the operation mode of the VAC system is transmitted to the controller function block in the sequence. See also Table operation modes.

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 : In the case of an VAC system with dehumidification, the control value of the dehumidification controller `FB_HVACPIDDehumidify` 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`). 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**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rY</td>
<td>Control signal output of the PID controller.</td>
</tr>
<tr>
<td>rXW</td>
<td>Control deviation.</td>
</tr>
<tr>
<td>bMaxLimit</td>
<td>The output <code>bMaxLimit</code> is TRUE if the output <code>rY</code> has reached the value <code>rYMax</code>.</td>
</tr>
<tr>
<td>bMinLimit</td>
<td>The output <code>bMinLimit</code> is TRUE if the output <code>rY</code> has reached the value <code>rYMin</code>.</td>
</tr>
<tr>
<td>bActive</td>
<td><code>bActive</code> is TRUE if the controller is active and enabled.</td>
</tr>
<tr>
<td>bARWactive</td>
<td><code>bARWactive</code> is TRUE if the integral component of the controller has reached the lower or upper control variable limit.</td>
</tr>
<tr>
<td>eState</td>
<td>State of the controller (see <code>E_HVACState</code>).</td>
</tr>
<tr>
<td>bError</td>
<td>This output indicates with a TRUE that there is an error.</td>
</tr>
<tr>
<td>eErrorCode</td>
<td><code>E_HVACErrorCodes</code> (see <code>E_HVACErrorCodes</code>).</td>
</tr>
<tr>
<td>bInvalidParameter</td>
<td>TRUE if an error has occurred during the plausibility check. The message must be acknowledged with <code>bReset</code>.</td>
</tr>
</tbody>
</table>

**VAR_IN_OUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rDeadRange</td>
<td>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 <code>rY</code>. 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 <code>rY</code> 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.</td>
</tr>
<tr>
<td>rKp</td>
<td>Proportional factor amplification. The variable is saved persistently. Preset to 0.</td>
</tr>
<tr>
<td>tTi</td>
<td>Integration time. The I-component corrects the residual control deviation following correction of the P-component. The smaller the <code>tTi</code> time is set, the faster the controller corrects. The control loop becomes unstable if the time is too short. Larger <code>tTi</code>-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.</td>
</tr>
<tr>
<td>tTv</td>
<td>Rate time. The larger <code>tTv</code> 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 <code>tTv</code>. The variable is saved persistently. Preset to 0 s.</td>
</tr>
<tr>
<td>tTd</td>
<td>Damping time. The variable is saved persistently. Preset to 0 s.</td>
</tr>
<tr>
<td>rDeadBand</td>
<td>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</td>
</tr>
</tbody>
</table>
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 $r_{\text{Deadband}}$ (See Example 507). The variable is saved persistently. Preset to 0.

$r_{\text{YMin}}$: Lower limit of the working range of the controller. The variable is saved persistently. Preset to 0.

$r_{\text{YMax}}$: Upper limit of the working range of the controller. The variable is saved persistently. Preset to 100.

$i_{\text{CurrentSequence}}$: Number of the active controller from the sequence.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDataSecurityType</td>
<td>E_HVACDataSecurityType;</td>
</tr>
<tr>
<td>bSetDefault</td>
<td>BOOL;</td>
</tr>
<tr>
<td>eModeSeqCtrl</td>
<td>E_HVACSequenceCtrlMode;</td>
</tr>
<tr>
<td>iNumberOfSequences</td>
<td>INT;</td>
</tr>
<tr>
<td>iMyNumberInSequence</td>
<td>INT;</td>
</tr>
<tr>
<td>rW</td>
<td>REAL;</td>
</tr>
<tr>
<td>rX</td>
<td>REAL;</td>
</tr>
<tr>
<td>tTaskCycleTime</td>
<td>TIME;</td>
</tr>
<tr>
<td>tCtrlCycleTime</td>
<td>TIME;</td>
</tr>
<tr>
<td>rYDehumidify</td>
<td>REAL;</td>
</tr>
<tr>
<td>eCtrlMode</td>
<td>E_HVACCtrlMode;</td>
</tr>
</tbody>
</table>
**eDataSecurityType**: If \( e\text{DataSecurityType} \) \( \Rightarrow e\text{HVACDataSecurityType\_Persistent} \), 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 \( \text{FB\_HVACPersistentDataHandling} \) must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000\_TC3\_HVAC/Resources/zip/5090272523.zip

If \( e\text{DataSecurityType} \) \( \Rightarrow e\text{HVACDataSecurityType\_Idle} \) the persistently declared variables are not saved in a fail-safe manner.

### NOTE

A cyclically changing variable must never be linked with the IN\_OUT variable of a function block, if \( e\text{DataSecurityType} \) \( \Rightarrow e\text{HVACDataSecurityType\_Persistent} \). 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 \( \text{E\_HVACSequenceCtrlMode} \)). In addition, the operation mode of the VAC system is transmitted to the controller blocks in the sequence. See also Table operation modes.

**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 \( t\text{CtrlCycleTime} \) 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**: Constant value.

**eCtrlMode**: The operation mode is selected via this enum (see \( \text{E\_HVACCtrlMode} \)). Manual or automatic operation mode.

**rYManual**: Manual value that is set at the output \( rY \) if \( e\text{CtrlMode} = e\text{HVACCtrlMode\_Manual} \).

**bResetController**: A positive edge on the input \( b\text{ResetController} \) resets the PID controller.

**bReset**: Acknowledge input in the event of a fault.

**VAR\_OUTPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>rY</td>
<td>REAL</td>
</tr>
<tr>
<td>rW</td>
<td>REAL</td>
</tr>
<tr>
<td>bMaxLimit</td>
<td>BOOL</td>
</tr>
<tr>
<td>bMinLimit</td>
<td>BOOL</td>
</tr>
<tr>
<td>bActive</td>
<td>BOOL</td>
</tr>
<tr>
<td>bARWactive</td>
<td>BOOL</td>
</tr>
<tr>
<td>eState</td>
<td>E_HVACState</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
</tr>
<tr>
<td>eErrorCode</td>
<td>E_HVACErrorCodes</td>
</tr>
<tr>
<td>bInvalidParameter</td>
<td>BOOL</td>
</tr>
</tbody>
</table>
**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 [490]).

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

### VAR_IN_OUT

```
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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

### 5.1.5 HVAC Room Functions

#### 5.1.5.1 Air conditioning

**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 [490] := 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 [479]` 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/zip/5090272523.zip](https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip)

If `eDataSecurityType [490] := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

**NOTE**

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType [490] := 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:** 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:** 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

### 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 [490] := 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 [479]` 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/zip/5090272523.zip](https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip)

If `eDataSecurityType [490] := eHVACDataSecurityType_Idle`, the persistently declared variables are not saved in a fail-safe manner.
NOTE
A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType := eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.

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

bEnable: The function block is activated if the variable bEnable is TRUE. No fan stage is activated with a FALSE.

eCtrlFct: This input is connected to the output eCtrlFct 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:= blnInvalidParameter

udiSecRT_MINPowerOnTime: Indicates the remaining time of the minimum switch-on time.

udiRT_TimePeriod: Indicates the remaining time of the manual override.

blnInvalidParameter: Indicates that there is an incorrect input parameter. blnInvalidParameter 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 blnInvalidParameter is set to TRUE. The variable is saved persistently. Preset to 3.

<table>
<thead>
<tr>
<th>Cooling</th>
<th>Heating</th>
<th>Valence</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1 (= fan controller active in heating mode)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2 (= fan controller active in cooling mode)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3 (= fan controller active in heating mode and cooling mode)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>bEnable</th>
<th>bPipeSystem</th>
<th>bFeedb-HeatMedium</th>
<th>bFeedb-CoolMedium</th>
<th>interim result</th>
<th>bDewPoint</th>
<th>eCtrlFct</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>OFF</td>
<td>TRUE / FALSE</td>
<td>OFF</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Heating</td>
<td>TRUE / FALSE</td>
<td>Heating</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Cooling</td>
<td>TRUE</td>
<td>OFF</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Heating</td>
<td>TRUE / FALSE</td>
<td>Heating</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Heating</td>
<td>TRUE / FALSE</td>
<td>Heating</td>
</tr>
</tbody>
</table>

In the 4 pipe system

<table>
<thead>
<tr>
<th>bEnable</th>
<th>bPipeSystem</th>
<th>T_Room &lt;=</th>
<th>T_Room &gt;</th>
<th>interim result</th>
<th>bDewPoint</th>
<th>eCtrlFct</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>OFF</td>
<td>TRUE / FALSE</td>
<td>OFF</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Cooling</td>
<td>TRUE</td>
<td>OFF</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Heating</td>
<td>TRUE / FALSE</td>
<td>Heating</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Heating</td>
<td>TRUE / FALSE</td>
<td>Heating</td>
</tr>
</tbody>
</table>
**VAR_INPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDataSecurityType</td>
<td>E_HVACDataSecurityType;</td>
</tr>
<tr>
<td>bSetDefault</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bEnable</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bPipeSystem</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bFeedbHeatMedium</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bFeedbCoolMedium</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bDewPoint</td>
<td>BOOL;</td>
</tr>
<tr>
<td>rRoomTemp</td>
<td>REAL;</td>
</tr>
<tr>
<td>stSetpoint</td>
<td>ST_BARSetpointRoom;</td>
</tr>
</tbody>
</table>

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

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

**NOTE**

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

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

**bEnable:** TRUE enables the function block. If FALSE the function block is disabled and `eCtrlFct := eHVACCtrlFct_Off`.

**bPipeSystem:** FALSE means that a 2 pipe system is present. TRUE means that a 4 pipe system is present.

**bFeedbHeatMedium:** Signal from the power generation or distribution that heating medium is available.

**bFeedbCoolMedium:** Signal from the power generation or distribution that cooling medium is available.

**bDewPoint:** The dew point sensor is connected to this input. If this is triggered the cooling control function is deactivated and `eCtrlFct := eHVACCtrlFct_Off` is set.

**rRoomTemp:** This input variable transfers the current room temperature to the function block.

**stSetpoint:** STRUCTURE containing the setpoints for the individual energy levels (see `ST_BARSetpointRoom`).

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>eCtrlFct</td>
<td>E_BARCtrlFct;</td>
</tr>
<tr>
<td>uiReqHeating</td>
<td>UINT;</td>
</tr>
<tr>
<td>uiReqCooling</td>
<td>UINT;</td>
</tr>
<tr>
<td>udiRT_ChangeOverDelay</td>
<td>UDINT;</td>
</tr>
</tbody>
</table>

**eCtrlFct:** This output contains the current control function (see `E_BARCtrlFct`).

**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**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uOnChangeOverDelay</td>
<td>UINT;</td>
</tr>
</tbody>
</table>
**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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

#### 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 [490] = 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 [479] 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/zip/5090272523.zip
If `eDataSecurityType[490] := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

**NOTE**

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType[490] := 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:

<table>
<thead>
<tr>
<th>Energy levels</th>
<th>Protection-Heating</th>
<th>Economy-Heating</th>
<th>Pre-ComfortHeating</th>
<th>ComfortHeating</th>
<th>ComfortCooling</th>
<th>Pre-ComfortCooling</th>
<th>Economy-Cooling</th>
<th>Protection-Cooling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base setpoint value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[°C]</td>
<td>12</td>
<td>15</td>
<td>19</td>
<td>21</td>
<td>24</td>
<td>28</td>
<td>35</td>
<td>40</td>
</tr>
<tr>
<td>Setpoint ShiftHeat</td>
<td>+3</td>
<td>-</td>
<td>-</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>at [K]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resulting setpoint</td>
<td>12</td>
<td>15</td>
<td>19</td>
<td>24</td>
<td>27</td>
<td>31</td>
<td>35</td>
<td>40</td>
</tr>
</tbody>
</table>

A lowering of the ComfortHeating setpoint affects only the ComfortHeating and PreComfortHeating setpoints.

Example:

<table>
<thead>
<tr>
<th>Energy levels</th>
<th>Protection-Heating</th>
<th>Economy-Heating</th>
<th>Pre-ComfortHeating</th>
<th>ComfortHeating</th>
<th>ComfortCooling</th>
<th>Pre-ComfortCooling</th>
<th>Economy-Cooling</th>
<th>Protection-Cooling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base setpoint value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[°C]</td>
<td>12</td>
<td>15</td>
<td>19</td>
<td>21</td>
<td>24</td>
<td>28</td>
<td>35</td>
<td>40</td>
</tr>
<tr>
<td>Setpoint ShiftHeat</td>
<td>-3</td>
<td>-</td>
<td>-</td>
<td>-3</td>
<td>-3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>at [K]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resulting setpoint</td>
<td>12</td>
<td>15</td>
<td>16</td>
<td>18</td>
<td>24</td>
<td>28</td>
<td>35</td>
<td>40</td>
</tr>
</tbody>
</table>

**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:
A lowering of the ComfortCooling setpoint only affects the ComfortCooling. The PreComfortCooling is not changed.

Example:

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:
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:

<table>
<thead>
<tr>
<th>Energy levels</th>
<th>Protection-Heating</th>
<th>EconomyHeating</th>
<th>PreComfortHeating</th>
<th>ComfortHeating</th>
<th>ComfortCooling</th>
<th>PreComfortCooling</th>
<th>EconomyCooling</th>
<th>ProtectionCooling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base setpoint value [°C]</td>
<td>12</td>
<td>15</td>
<td>19</td>
<td>21</td>
<td>24</td>
<td>28</td>
<td>35</td>
<td>40</td>
</tr>
<tr>
<td>Setpoint ShiftUser [K]</td>
<td>+3</td>
<td>-</td>
<td>-</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Resulting setpoint [°C]</td>
<td>12</td>
<td>15</td>
<td>19</td>
<td>24</td>
<td>27</td>
<td>31</td>
<td>35</td>
<td>40</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Energy levels</th>
<th>Protection-Heating</th>
<th>EconomyHeating</th>
<th>PreComfortHeating</th>
<th>ComfortHeating</th>
<th>ComfortCooling</th>
<th>PreComfortCooling</th>
<th>EconomyCooling</th>
<th>ProtectionCooling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base setpoint value [°C]</td>
<td>12</td>
<td>15</td>
<td>19</td>
<td>21</td>
<td>24</td>
<td>28</td>
<td>35</td>
<td>40</td>
</tr>
<tr>
<td>Setpoint ShiftUser [K]</td>
<td>-3</td>
<td>-</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Resulting setpoint [°C]</td>
<td>12</td>
<td>15</td>
<td>16</td>
<td>18</td>
<td>21</td>
<td>28</td>
<td>35</td>
<td>40</td>
</tr>
</tbody>
</table>

bReset: Acknowledge input in case of a fault or an incorrect parameter.

VAR OUTPUT

stSetpoint : ST_BARSersetpointRoom;
bInvalidParameter : BOOL;

stSetpoint: Structure containing the setpoints for all the energy levels (see ST_BARSersetpointRoom [ 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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;
rXp : REAL;
rYMin : REAL;
rYMax : REAL;
rSyncValue : REAL;
**eDataSecurityType**: If \texttt{eDataSecurityType} := \texttt{eHVACDataSecurityType\_Persistent}, the persistent \texttt{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 \texttt{FB\_HVACPersistentDataHandling} must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000\_TC3\_HVAC/Resources/zip/5090272523.zip

If \texttt{eDataSecurityType} := \texttt{eHVACDataSecurityType\_Idle} the persistently declared variables are not saved in a fail-safe manner.

<table>
<thead>
<tr>
<th>NOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if \texttt{eDataSecurityType} := \texttt{eHVACDataSecurityType_Persistent}. It would lead to early wear of the flash memory.</td>
</tr>
</tbody>
</table>

\texttt{bSetDefault}: If the variable is TRUE, the default values of the \texttt{VAR\_IN\_OUT} variables are adopted.

\texttt{bEnable}: Controller activation

\texttt{rW}: Setpoint.

\texttt{rX}: Actual value.

\texttt{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.

\texttt{uiCtrlCycleCall} : Call cycle of the function block as a multiple of the cycle time. A zero entry is automatically evaluated as \texttt{uiCtrlCycleCall} =1.

Example: \texttt{tTaskCycleTime} = 20ms, \texttt{uiCtrlCycleCall} =10 -> The control algorithm is called every 200 ms. Thus the outputs are also updated only every 200 ms.

\texttt{bSync}: A rising edge at this input sets the (internal) I-component such that \texttt{rSyncValue} is output at the control value output. If the I-component is deactivated by \texttt{tTn}=0 ms, however, then this command is ignored.

**VAR\_OUTPUT**

\texttt{rY} \hspace{1em} \texttt{REAL};
\texttt{rE} \hspace{1em} \texttt{REAL};
\texttt{bARW} \hspace{1em} \texttt{BOOL};
\texttt{bMaxLimit} \hspace{1em} \texttt{BOOL};
\texttt{bMinLimit} \hspace{1em} \texttt{BOOL};
\texttt{bError} \hspace{1em} \texttt{BOOL};
\texttt{udiErrorId} \hspace{1em} \texttt{UDINT};

\texttt{rY}: Control value.

\texttt{rE}: Control deviation (calculation dependent on control direction [\ref{control_direction}])

\texttt{rEMin}: lower control deviation limit value, which results from the input proportional band.

\texttt{rEMax}: upper control deviation limit value, which results from the input proportional band.

\texttt{bARW}: Anti-Reset-Windup function is active.

\texttt{bMaxLimit}: The control value has reached its upper limit value.

\texttt{bMinLimit}: The control value has reached its lower limit value.

\texttt{bError}: This output is switched to TRUE if the parameters entered are erroneous.

\texttt{udiErrorId}: Contains the error code if the values entered should be erroneous. See Error codes [\ref{error_codes}].
bDirection: The control direction [201] 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 Kp*E due to the proportional component and then grows by a further E in each interval Tn.

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

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 Kp 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 Kp 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 Kp factor.
The equation for this is:
• Control value = control deviation x gain factor → \( Y = E \times K_p \)
This relationship also applies to the changes in the control deviation and the control value:
• Control value change = control deviation change x gain factor → \( \Delta Y = \Delta E \times K_p \)
Referenced to the minimum and maximum value of the control value, \( Y_{\min} \) and \( Y_{\max} \):
• \( Y_{\max} - Y_{\min} = (E(Y_{\max}) - E(Y_{\min})) \times K_p \)
This difference, \( E(Y_{\max}) - E(Y_{\min}) \), is called the proportional band (Xp). Transposed, the equation is then:
• \( K_p = \frac{Y_{\max} - Y_{\min}}{X_p} \)
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 $K_p$ factor. However, the control deviation limit values $E(Y_{max}) - E(Y_{min})$ are shifted:

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.

<table>
<thead>
<tr>
<th>bDirection</th>
<th>rE (control-deviation)</th>
<th>control-direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>IrX-IrW (actual value-setpoint)</td>
<td>direct (cooling)</td>
</tr>
<tr>
<td>FALSE</td>
<td>IrW-IrX (setpoint-actual value)</td>
<td>reverse (heating)</td>
</tr>
</tbody>
</table>
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 $T_n=0$ 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 [315]. The Module is also stopped, if the input $bEnable$ is not set. In both cases the outputs are set as follows:

<table>
<thead>
<tr>
<th>$rY$</th>
<th>0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$rE$</td>
<td>0.0</td>
</tr>
<tr>
<td>$bARW$</td>
<td>FALSE</td>
</tr>
<tr>
<td>$bMaxLimit$</td>
<td>FALSE</td>
</tr>
<tr>
<td>$bMinLimit$</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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 [219] - simple light switching circuit without dimming function.

FB_BARLightCircuitDim [221] - simple light switching circuit with dimming function.


FB_BARStairwellAutomatic [224] - stairwell lighting with early warning sequence.


FB_BARDaylightControl [212] - daylight controller without dimming processes
FB_BARConstantLightControl - constant light control with continuous output of analog values.

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

FB_BARLightActuator - 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**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDataSecurityType</td>
<td>E_HVACDataSecurityType;</td>
</tr>
<tr>
<td>uiLightCtrlMode</td>
<td>UINT;</td>
</tr>
<tr>
<td>bSwitch</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>lrDimValue</td>
<td>LREAL;</td>
</tr>
<tr>
<td>uiRemainingHoldTime</td>
<td>UINT;</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL;</td>
</tr>
<tr>
<td>udiErrorId</td>
<td>UDINT;</td>
</tr>
</tbody>
</table>

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

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

Application example: [https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip](https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip)

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

**NOTE**

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

**uiLightCtrlMode:** Operation mode.

- 0: Automatic mode; the automatic light circuit is active and reacts to the input bSwitch.
- 1: Manual On; the automatic light circuit is inactive - the output lrDimValue is set constantly to lrManualDimValue.
- 2: Manual Off; the automatic light circuit is inactive - the output lrDimValue is set constantly to 0.0.

**bSwitch:** A rising edge switches the light on in automatic mode (uiLightCtrlMode=0), a falling edge starts the holding time generator. This input has no function in manual operation mode (uiLightCtrlMode=1 or 2).
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 [315].

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/zip/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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

Documents about this

example_persistent.zip (Resources/zip/5090272523.zip)
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 \( b_{\text{Presence}} \). Optionally the constant light control can also be operated by a pushbutton. A short pulse on the input \( b_{\text{Toggle}} \) 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 \( b_{\text{CentralOn}} \). A central command to switch off can be connected to the input \( b_{\text{CentralOff}} \).

Actuating the pushbutton on \( b_{\text{Toggle}} \) for a longer period manually changes the setpoint value for the room brightness. Upon a falling edge on \( b_{\text{Toggle}} \) 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 \( b_{\text{Presence}} \), a rising edge on \( b_{\text{CentralOn}} \) or a pulse on \( b_{\text{Toggle}} \), the last manually selected setpoint value or the value of \( u_{\text{SetpointValue}} \) is adopted, depending on the setting of the parameter \( b_{\text{InitialMode}} \).

In automatic mode the lighting is switched on only if the room brightness lies below an adjustable hysteresis value \( u_{\text{TargetRange}} \). As the external brightness increases, the constant light controller reduces the artificial light portion until a minimum load control value \( Ir_{\text{MinDimValue}} \) lies at the output of the controller \( Ir_{\text{DimValue}} \). Subsequently, the controller switches the lighting off following a time delay with the timer \( u_{\text{OffDelay}} \).

If the external brightness decreases and the light should be turned on again, it can be delayed by \( u_{\text{OnDelay}} \) [sec].

In order to avoid unpleasant visible changes in the brightness, the rate of change of the control signal is retarded with the parameter \( u_{\text{ControlRampTime}} \).

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 \( b_{\text{CentralOn}}, b_{\text{CentralOff}} \) and \( b_{\text{Toggle}} \).

The function block knows three different modes, which can be set via the input \( u_{\text{LightCtrlMode}} \):
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.
Programming

Switch-on and switch-off delay, minimum output value

1. Initial situation: the room-brightness is well-regulated.
   The outside-brightness decreases and thus the room-brightness. Because the room-brightness is still within the target range, the control will not react.

2. The outside-brightness decreases further and the room-brightness leaves the target range. The control starts its ramp – due to the lower slope of the ramp, the control only succeeds in damping the loss of room-brightness.

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

4. The target-range is reached again.

5. The target-range is reached again.
6. The outside-brightness rises and thus the room-brightness. Because the room-brightness is still within the target-range, the control will not react.

7. The outside-brightness continues to rise and the measured room-brightness leaves the target-range. The control starts its ramp – due to the lower slope of the ramp, the control only succeeds in damping the gain of room-brightness.

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

9. The target-range is reached again. Because the outside-brightness is now lower than in section 1, the electrical dim-value is higher.
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 $\text{uiOnDelay}/\text{uiOffDelay}$ can be defined in the function block around a lower limit value $\text{lrMinDimValue}$. If the internally determined output value sinks below this minimum value, then the output remains at this minimum for the time $\text{uiOffDelay}$ [s]. Only after that is 0.0 output at the output $\text{lrDimValue}$. 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 $\text{uiOnDelay}$ [s] and then to the value $\text{lrMinDimValue}$. 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 $\text{lrDimValue}$ is appropriate for the lighting conditions.

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 $\text{bSwitch}$ (longer than $\text{uiSwitchOverTime}$ 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 $\text{bSwitch}$ resets to FALSE, then the brightness value now measured on $\text{uiBrightness}$ is adopted as the new setpoint value to which control is to take place.

Manual operation

The inputs $\text{bSwitch}$, $\text{bCentralOn}$ and $\text{bCentralOff}$ have no function in the manual operation modes: if $\text{uiLightCtrlMode}=1$ the output value $\text{lrDimValue}$ is set constantly to $\text{lrManualDimValue}$ and if $\text{uiLightCtrlMode}=2$ 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[490] := 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[479] 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/zip/5090272523.zip

If eDataSecurityType[490] := eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

NOTE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType[490] := 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 [315].

<table>
<thead>
<tr>
<th>VAR_IN_OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>uiSwitchOverTime : UINT;</td>
</tr>
<tr>
<td>uiAdjustRampTime : UINT;</td>
</tr>
<tr>
<td>bPresenceOnActive : BOOL;</td>
</tr>
<tr>
<td>bInitialMode : BOOL;</td>
</tr>
<tr>
<td>uiTargetRange : UINT;</td>
</tr>
<tr>
<td>uiControlRampTime : UINT;</td>
</tr>
<tr>
<td>uiOnDelay : UINT;</td>
</tr>
<tr>
<td>uiOffDelay : UINT;</td>
</tr>
<tr>
<td>lrMinDimValue : LREAL;</td>
</tr>
<tr>
<td>lrMaxDimValue : LREAL;</td>
</tr>
<tr>
<td>lrStartDimValue : LREAL;</td>
</tr>
<tr>
<td>lrManualDimValue : LREAL;</td>
</tr>
</tbody>
</table>

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/zip/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 [206].

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).
Daylight switch. Unlike the constant light control \[ \text{206} \], 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 \text{uiLightCtrlMode}:

- Automatic mode
- Manual On mode
- Manual Off mode

**Automatic mode**

If the brightness in the room, \text{uiBrightness}, falls below the value \text{uiSwitchOnValue} [lux] for the time \text{uiOnDelay} [s], then the light is switched on. There is now a delay, \text{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 \text{uiSwitchOffOffset}. If an increase in the external brightness causes the room brightness to exceed this determined re-switch-off value for the duration of \text{uiOffDelay} [s], then the lighting is switched off again.
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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDataSecurityType</td>
<td>E_HVACDataSecurityType;</td>
</tr>
<tr>
<td>uiLightCtrlMode</td>
<td>UINT</td>
</tr>
<tr>
<td>bSwitch</td>
<td>BOOL</td>
</tr>
<tr>
<td>bCentralOn</td>
<td>BOOL</td>
</tr>
<tr>
<td>bCentralOff</td>
<td>BOOL</td>
</tr>
<tr>
<td>uiBrightness</td>
<td>UINT</td>
</tr>
</tbody>
</table>

**eDataSecurityType:** If eDataSecurityType \[490\] = 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 \[479\] 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/zip/5090272523.zip](https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip)

If eDataSecurityType \[490\] = eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

**NOTE**

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType \[490\] = 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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>lrDimValue</td>
<td>LREAL</td>
</tr>
<tr>
<td>bOn</td>
<td>BOOL</td>
</tr>
<tr>
<td>uiRemainingOnDelay</td>
<td>UINT</td>
</tr>
<tr>
<td>uiRemainingOffDelay</td>
<td>UINT</td>
</tr>
<tr>
<td>uiPreOnValue</td>
<td>UINT</td>
</tr>
<tr>
<td>uiLightGain</td>
<td>UINT</td>
</tr>
<tr>
<td>uiSwitchOffValue</td>
<td>UINT</td>
</tr>
<tr>
<td>bControlActive</td>
<td>BOOL</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
</tr>
<tr>
<td>udiErrorId</td>
<td>UDINT</td>
</tr>
</tbody>
</table>

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 [315].

VAR_IN_OUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uiSwitchOnValue</td>
<td>UINT</td>
</tr>
<tr>
<td>uiSwitchOffOffset</td>
<td>UINT</td>
</tr>
<tr>
<td>uiLightOnTime</td>
<td>UINT</td>
</tr>
<tr>
<td>uiOnDelay</td>
<td>UINT</td>
</tr>
<tr>
<td>uiOffDelay</td>
<td>UINT</td>
</tr>
</tbody>
</table>

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/zip/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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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 [221], 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 \texttt{FB\_BARLightCircuitDim} also contains a ramp function for presetting the target value, the light actuator function described here is called the "secondary dimtime" - \texttt{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 \texttt{FB\_BARLightCircuitDim} will change its output value \texttt{lrDimValue} from 0% to 100% in 10 s. However, the function block \texttt{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 \(\[221\] \) 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 \(\[221\] \)). 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 \(eDataSecurityType[490]\) \(=\) 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 \(\[479\] \) 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/zip/5090272523.zip](https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip)

If \(eDataSecurityType[490]\) \(=\) eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

**NOTE**

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if \(eDataSecurityType[490]\) \(=\) eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.

**IrTargetValue**: 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 \(IrTargetValue\). 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: lrDimValue=0.0 => bOn=FALSE - lrDimValue>0.0 => bOn=TRUE.

bSceneMode: The function block presently outputs a scene value and not the value lrTargetValue on the input.

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

udiErrorld: Contains the error code if the values entered should be erroneous. See Error codes [315].

VAR_IN_OUT

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

uiSecDimTime: Ramp time in seconds. This is the time that the light actuator function requires in order to regulate from 0..100%.

arrLightScenes: Table of saved light values.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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**

\[
\begin{align*}
uiLightCtrlMode & : \text{UINT}; \\
bSwitch & : \text{BOOL}; \\
bCentralOn & : \text{BOOL}; \\
bCentralOff & : \text{BOOL};
\end{align*}
\]

- **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**

\[
\begin{align*}
lrDimValue & : \text{LREAL}; \\
bLight & : \text{BOOL}; \\
bError & : \text{BOOL}; \\
udiErrorId & : \text{UDINT};
\end{align*}
\]

- **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 \([ \text{315}]\).

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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.
In the manual operation modes the output value $lr\text{DimValueOut}$ is set constantly to the value $lr\text{ManualDimValue}$ if $ui\text{LightCtrlMode}=1$ and constantly to the value 0.0 if $ui\text{LightCtrlMode}=2$.

**VAR_INPUT**

- eDataSecurityType : E_HVACDataSecurityType;
- uiLightCtrlMode : UINT;
- bSwitch : BOOL;
- bCentralOn : BOOL;
- bCentralOff : BOOL;

**eDataSecurityType**: If `eDataSecurityType`\[490\] = `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`\[479\] 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/zip/5090272523.zip

If eDataSecurityType [490] := eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

**NOTE**

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType [490] := eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.

---

**uiLightCtrlMode**: Operation mode.

- 0: Automatic mode; the output value IrDimmValue can be influenced by the command inputs bSwitch, bCentralOn and bCentralOff
- 1: Manual On mode; the value IrManualDimValue is output without a delay at the output IrDimValueOut, all other command inputs are ineffective.
- 2: Manual Off mode; the value 0.0 is output without a delay at the output IrDimValueOut, all other command inputs are ineffective.

**bSwitch**: A short TRUE signal that is shorter than uiSwitchOverTime in milliseconds switches the output IrDimValue. If the output IrDimValue is initially larger than 0.0, then it is switched to 0 and the previous value is saved. If on the other hand it is 0.0, then the output is set to the previously internally saved value. This saved value is set to the maximum value in the switch-on state of the program, see below.

A long TRUE signal that is longer than uiSwitchOverTime in milliseconds starts the dimming process, wherein the dimming direction changes if two long signals follow one another on the input bSwitch.

**bCentralOn**: Switches the output IrDimValueOut to the previously saved (at the last switch-off) dimming value, see bSwitch.

**bCentralOff**: Switches the output IrDimValueOut to 0.0.

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

---

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>lrDimValue</td>
<td>LREAL</td>
</tr>
<tr>
<td>bLight</td>
<td>BOOL</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
</tr>
<tr>
<td>udiErrorId</td>
<td>UDINT</td>
</tr>
</tbody>
</table>

**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 [315].

---

**VAR_IN_OUT**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uiSwitchOverTime</td>
<td>UINT</td>
</tr>
<tr>
<td>uiDimRampTime</td>
<td>UINT</td>
</tr>
<tr>
<td>uiCycleDelay</td>
<td>UINT</td>
</tr>
<tr>
<td>lrMinDimValue</td>
<td>LREAL</td>
</tr>
<tr>
<td>lrMaxDimValue</td>
<td>LREAL</td>
</tr>
<tr>
<td>lrManualDimValue</td>
<td>LREAL</td>
</tr>
</tbody>
</table>

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/zip/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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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 \textit{bSwitch} input has no function in the manual modes: the output value \textit{bOn} is set constantly to TRUE if \textit{uiLightCtrlMode}=1 and constantly to FALSE if \textit{uiLightCtrlMode}=2.

\begin{itemize}
  \item \textbf{Manual mode}
  \begin{itemize}
    \item Changing to manual mode resets any lighting sequence that had started up to that point.
  \end{itemize}
\end{itemize}

\textbf{VAR_INPUT}

\begin{verbatim}
VAR_INPUT
eDataSecurityType   : E_HVACDataSecurityType;
uiLightCtrlMode      : UINT;
bSwitch              : BOOL;
\end{verbatim}

\textbf{eDataSecurityType}: If \textbf{eDataSecurityType}[\textbullet\ 490] = \textit{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 \textit{FB\_HVACPersistentDataHandling}[\textbullet\ 479] 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/zip/5090272523.zip

If \textbf{eDataSecurityType}[\textbullet\ 490] = \textit{eHVACDataSecurityType\_Idle} the persistently declared variables are not saved in a fail-safe manner.
NOTE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType[490] = 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 [315].

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/zip/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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
5.1.5.3.8  FB_BARTTwilightAutomatic

Automatic twilight function

The function block knows three different modes, which can be set via the input \textit{uiLightCtrlMode}:

- Automatic mode
- Manual On mode
- Manual Off mode

\textbf{Automatic mode}

In automatic mode a positive edge on \textit{bSwitch} activates the automatic twilight function, if it had been inactive beforehand. A further edge on \textit{bSwitch} deactivates the automatic twilight function again. The automatic function is explicitly switched on and off with \textit{bCentralOn} and \textit{bCentralOff}, independent of the previous state of the automatic function. If the automatic twilight function is activated, then the function block switches the output \textit{lrDimValue} to the switch-on value \textit{lrDimValueOn} when the brightness falls below a switch-on threshold value \textit{uiSwitchOnValue} for an entered delay time \textit{uiOnDelay}. Conversely, if a switch-off threshold value \textit{uiSwitchOffValue} is exceeded for an entered time delay \textit{uiOffDelay}, then the output is switched to the value \textit{lrDimValueOff}. In the inactive state the light output value \textit{lrDimValue} is set to 0.0.

\textbf{Manual operation}

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

\textbf{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.

\textbf{VAR_INPUT}

\begin{verbatim}
VAR_INPUT
  eDataSecurityType: E_HVACDataSecurityType;
  uiLightCtrlMode   : UINT;
  bSwitch           : BOOL;
  bCentralOn        : BOOL;
  bCentralOff       : BOOL;
  uiBrightness      : UINT;
\end{verbatim}
**eDataSecurityType**: If `eDataSecurityType[490]` := `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[479]` 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/zip/5090272523.zip

If `eDataSecurityType[490]` := `eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

**NOTE**
A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType[490]` := `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**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>lrDimValue</td>
<td>LREAL;</td>
</tr>
<tr>
<td>uiRemainingOnDelay</td>
<td>UINT;</td>
</tr>
<tr>
<td>uiRemainingOffDelay</td>
<td>UINT;</td>
</tr>
<tr>
<td>bControlActive</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL;</td>
</tr>
<tr>
<td>udiErrorId</td>
<td>UDINT;</td>
</tr>
</tbody>
</table>

**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 [315].
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**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uiSwitchOnValue</td>
<td>UINT</td>
</tr>
<tr>
<td>uiSwitchOffValue</td>
<td>UINT</td>
</tr>
<tr>
<td>uiOnDelay</td>
<td>UINT</td>
</tr>
<tr>
<td>uiOffDelay</td>
<td>UINT</td>
</tr>
<tr>
<td>lrDimOnValue</td>
<td>LREAL</td>
</tr>
<tr>
<td>lrDimOffValue</td>
<td>LREAL</td>
</tr>
<tr>
<td>lrManualDimValue</td>
<td>LREAL</td>
</tr>
</tbody>
</table>

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/zip/5090272523.zip](https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

**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

**Priority controller**
- FB: BARSunblindPrioritySwitch
  - Prio1-input
  - output

**Weather protection**
- FB: BARSunblindWeatherProtection
  - Positioning telegram
  - ST: BARSunblind

**Maintenance position**
- FB: BARSunblindEvent
  - Positioning telegram
  - ST: BARSunblind

**Manual operation**
- FB: BARSunblindSwitch
  - or
- Manual operation with scene selection and programming
  - FB: BARSunblindScene
  - Positioning telegram
  - ST: BARSunblind

**Automatic twilight function**
- FB: BARSunblindTwilightAutomatic
  - Positioning telegram
  - ST: BARSunblind

**Thermo-automatic function**
- FB: BARSunblindThermoAutomatic
  - Positioning telegram
  - ST: BARSunblind

**Automatic sun protection**
- FB: BARSunblindProtectionEx
  - Overview of automatic sun protection (shading correction)
  - Positioning telegram
  - ST: BARSunblind

**Parking position**
- FB: BARSunblindEvent
  - Positioning telegram
  - ST: BARSunblind

**Blind-actuator**
- FB: BARSunblindActuator
5.1.5.4.2 Overview sun protection

Overview of automatic sun protection

- **Entry of shading objects**
  - FB_BARShadingObjectsEntry
  - FB_BARReadShadingObjectsList

- **Entry of facade elements**
  - FB_BARFacadeElementEntry
  - FB_BARReadFacadeElementList

- **External brightness check with delay and hysteresis**
  - FB_BARDelayedHysteresis

- **Check of valid position of the sun**
  - Sun direction range (azimuth angle)
    - FB_BARWithinRangeAzimuth
  - Sun height range (angle of elevation)
    - FB_BARWithinRangeElevation

- **Shading correction of a window group**
  - FB_BARShadingCorrection
  - shading objects: facade-elements
  - dGroupNotShaded

- **Automatic sun protection**
  - FB_BARSunProtectionEx
  - Enable positioning-telegramm
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:

- Louvre adjustment
- Height adjustment

Louvre adjustment

A louvered blind that can be adjusted offers the option of intelligent sun protection here. The position of the louvres 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 utilised 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 louvre angle:

- the current height of the sun (angle of elevation)
- the position of the sun, i.e. the azimuth angle
- the facade orientation
- the width of the louvres
- the spacing of the louvres

The effective angle of elevation is calculated from the first three of the parameters listed above:

If the facade orientation and the position of the sun (azimuth) are equal, then the effective angle is equal to the current angle of elevation.

However, if the sunlight falls at an angle onto the facade as seen from the sun direction, the effective angle is larger for the same angle of elevation.

The following three pictures illustrate the relationship between the effective angle of elevation and the dimensions of the blind and how the resulting louvre angle λ changes:
Lamellenwinkel

Louvre at an angle of $\lambda = 0$

Louvre-positioning at in the morning/evening

$\lambda = $ louvre-angle – in this drawing $\lambda < 0$
$\varepsilon = $ height of the sun (elevation)

Louvre-positioning at noon

$\lambda = $ louvre-angle – in this drawing: $\lambda > 0$
$\varepsilon = $ height of the sun (elevation)
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:
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:

\[ \epsilon = \text{height of the sun (Elevation)} \]
facade inclination: $\varphi = 0^\circ$

facade inclination: $\varphi < 0^\circ$

facade inclination: $\varphi > 0^\circ$
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:
<table>
<thead>
<tr>
<th>Line of sight</th>
<th>Facade orientation</th>
</tr>
</thead>
<tbody>
<tr>
<td>North</td>
<td>β=0°</td>
</tr>
<tr>
<td>East</td>
<td>β=90°</td>
</tr>
<tr>
<td>South</td>
<td>β=180°</td>
</tr>
<tr>
<td>West</td>
<td>β=270°</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Line of sight</th>
<th>Facade orientation</th>
</tr>
</thead>
<tbody>
<tr>
<td>South</td>
<td>β=0°</td>
</tr>
<tr>
<td>East</td>
<td>β=90°</td>
</tr>
<tr>
<td>North</td>
<td>β=180°</td>
</tr>
<tr>
<td>West</td>
<td>β=270°</td>
</tr>
</tbody>
</table>

All further explanations in this chapter are referring to the characteristics of the northern hemisphere. For the subsequent parameterization of the shading-correction (FB_BARShadingCorrection [261] / FB_BARShadingCorrectionSouth [264]) 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 P₀ 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 $0^\circ$ at sunrise (horizontal incidence of light) and can reach maximally $90^\circ$, 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.
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 $S_4$; $S_2$ and $S_3$ need not be considered if they are not higher.
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**

![Diagram of shaded building and ball elements](image)

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

This function block serves for the input of interpolation points for the function block FB_BARSunProtectionEx \[\text{[p.303]}\], if this should be operated in the height positioning mode with the help of a table, see E_BARPosMode \[\text{[p.501]}\].
In addition to the operation modes "Fixed shutter height" and "Maximum light incidence", the function block FB_BARSunProtectionEx \[303\] 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 \[303\], 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 \[315\] 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 \[303\] thus increases to 6, see ST_BARBlindPositionTable \[502\].

The interpolation of the values takes place in the glare protection block.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDataSecurityType</td>
<td>E_HVACDataSecurityType;</td>
</tr>
<tr>
<td>bSetDefault</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>
eDataSecurityType: If eDataSecurityType:= eHVACDataSecurityType_Persistent, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

NOTE

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

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

VAR_OUTPUT

bValid : BOOL;
uiErrorId : UDINT;
stBlindPositionTable : ST_BARBlindPositionTable;

bValid: this output will be TRUE as long as the entries correspond to the criteria listed above.

uiErrorId: contains the error code, if the entries should not correspond to the criteria listed above. See Error codes.

stBlindPositionTable : Transfer structure of the interpolation points, see ST_BARBlindPositionTable.

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/zip/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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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[490] = 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[479] 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/zip/5090272523.zip

If eDataSecurityType[490] = eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

**NOTE**

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType[490] = 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[315].
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/zip/5090275223.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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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 [314]. 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 [238].

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:

![Diagram of a window with corners labeled C1, C2, C3, C4 and an annotation Cn = corner n]

All data in meters!

\[
\begin{align*}
\text{lrCorner2X} &= \text{lrCorner1X} \\
\text{lrCorner2Y} &= \text{lrCorner1Y} + \text{lrWindowHeight} \text{ (window height)} \\
\text{lrCorner3X} &= \text{lrCorner1X} + \text{lrWindowWidth} \text{ (window width)} \\
\text{lrCorner3Y} &= \text{lrCorner2Y} \\
\text{lrCorner4X} &= \text{lrCorner1X} + \text{lrWindowWidth} \text{ (window width)} \\
\text{lrCorner4Y} &= \text{lrCorner1Y}
\end{align*}
\]

The function block is used in three steps:

- Read
- Change
- Write

Read

With the entries in \textit{iColumn} and \textit{iRow} the corresponding element is selected from the list \textit{arrFacadeElement[iColumn,iRow]}. A rising edge on \textit{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 [315] 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 [261] / FB_BARShadingCorrectionSouth [264]` and prevents incorrect calculations there.

Error (bValid=FALSE)

The function block `FB_BARShadingCorrection [261] / FB_BARShadingCorrectionSouth [264]`, 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 [490] := 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 [479] 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/zip/5090272523.zip

If eDataSecurityType [490] := eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

NOTE
A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType [490] := 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 : UDIINT;

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 [315].

**VAR_IN_OUT**

arrFacadeElement : ARRAY[1..iColumnsPerFacade, 1..iRowsPerFacade] OF ST_BARFacadeElement;

arrFacadeElement: List of facade elements [314].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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 [314]. 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](#) or ADS_Error codes.

**uiAmountOfSetsRead**: Number of data sets read

**VAR_IN_OUT**

```plaintext
arrFacadeElement : ARRAY[1..iColumnsPerFacade, 1..iRowsPerFacade] OF ST_BARFacadeElement;
```

**arrFacadeElement**: List of facade elements.

---

**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](#). 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` ([ ]), does not accept elements with the group entry '0'.

---
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

```plaintext
PROGRAM ReadFacadeElements

VAR
  bInit : BOOL;
  bRead : R_TRIG;
  arrFacadeElement : ARRAY[1..iColumnsPerFacade, 1..iRowsPerFacade] OF ST_BAFac;
  fbReadFacadeElementList : FB_BARReadFacadeElementList;
  bBusy : BOOL;
  bError : BOOL;
  udiErrorId : UDINT;
  uiAmountOfSetsRead : 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:
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 [p. 267] and was therefore provided in greater detail with an error number, which is explained in greater detail under error codes [p. 315]. 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

#### 5.1.5.4.9 FB_BARReadShadingObjectsList

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 [p. 267] and was therefore provided in greater detail with an error number, which is explained in greater detail under error codes [p. 315]. 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

#### 5.1.5.4.9 FB_BARReadShadingObjectsList

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 [p. 267] and was therefore provided in greater detail with an error number, which is explained in greater detail under error codes [p. 315]. 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

#### 5.1.5.4.9 FB_BARReadShadingObjectsList

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 [p. 267] and was therefore provided in greater detail with an error number, which is explained in greater detail under error codes [p. 315]. 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

#### 5.1.5.4.9 FB_BARReadShadingObjectsList

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 [p. 267] and was therefore provided in greater detail with an error number, which is explained in greater detail under error codes [p. 315]. 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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 [314]. 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 [315] 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 [314].

**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 [267].

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.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>Description</td>
<td>Type</td>
<td>Begin</td>
<td>End</td>
<td>P1x/Mx</td>
<td>P1y/My</td>
<td>P1z/Mz</td>
<td>P2y/R</td>
<td>P3x</td>
<td>P3y</td>
<td>P3z</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0 - Tetragon (Month)</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>-94.75</td>
<td>0</td>
<td>36.06</td>
<td>11</td>
<td>-70</td>
<td>71</td>
<td>11</td>
<td>68</td>
</tr>
<tr>
<td>2</td>
<td>1 - Globe</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>-23.33</td>
<td>0</td>
<td>9.9</td>
<td>10.5</td>
<td>-3.54</td>
<td>10.5</td>
<td>22</td>
<td>66</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>62.23</td>
<td>0</td>
<td>0</td>
<td>14.47</td>
<td>62.23</td>
<td>14.47</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>46</td>
<td>0</td>
<td>13</td>
<td>14.47</td>
<td>46</td>
<td>14.47</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>46</td>
<td>0</td>
<td>13</td>
<td>14.47</td>
<td>46</td>
<td>14.47</td>
<td>38</td>
<td>89</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>46</td>
<td>0</td>
<td>13</td>
<td>14.47</td>
<td>46</td>
<td>14.47</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>46</td>
<td>0</td>
<td>13</td>
<td>14.47</td>
<td>46</td>
<td>14.47</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>46</td>
<td>0</td>
<td>13</td>
<td>14.47</td>
<td>46</td>
<td>14.47</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>46</td>
<td>0</td>
<td>13</td>
<td>14.47</td>
<td>46</td>
<td>14.47</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>46</td>
<td>0</td>
<td>13</td>
<td>14.47</td>
<td>46</td>
<td>14.47</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>46</td>
<td>0</td>
<td>13</td>
<td>14.47</td>
<td>46</td>
<td>14.47</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>46</td>
<td>0</td>
<td>13</td>
<td>14.47</td>
<td>46</td>
<td>14.47</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>46</td>
<td>0</td>
<td>13</td>
<td>14.47</td>
<td>46</td>
<td>14.47</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>46</td>
<td>0</td>
<td>13</td>
<td>14.47</td>
<td>46</td>
<td>14.47</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

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 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:
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 [267] and was therefore provided in greater detail with an error number, which is explained in greater detail under error codes [315]. 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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 [264].

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 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 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):

<table>
<thead>
<tr>
<th>Line of sight</th>
<th>Façade orientation</th>
</tr>
</thead>
<tbody>
<tr>
<td>North</td>
<td>β=0°</td>
</tr>
<tr>
<td>East</td>
<td>β=90°</td>
</tr>
<tr>
<td>South</td>
<td>β=180°</td>
</tr>
<tr>
<td>West</td>
<td>β=270°</td>
</tr>
</tbody>
</table>

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 \(\Rightarrow e\)HVACDataSecurityType\_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 \(\Rightarrow 479\) 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/zip/5090272523.zip

If eDataSecurityType \(\Rightarrow 490\) = eHVACDataSecurityType\_Idle the persistently declared variables are not saved in a fail-safe manner.

<table>
<thead>
<tr>
<th>NOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType (\Rightarrow 490) = eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.</td>
</tr>
</tbody>
</table>

**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**

```plaintext
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 \(\Rightarrow 315\).

**VAR\_IN\_OUT**

```plaintext
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/zip/5090272523.zip

- **arrFacadeElement:** List of facade elements \(\Rightarrow 314\).
- **arrShadingObject:** List of shading objects \(\Rightarrow 314\).
- **lrFacadeOrientation:** Facade orientation, see illustration above.
- **usiGroupId:** Window group regarded. The group 0 is reserved here for unused window elements, see FB\_BARFacadeElementEntry \(\Rightarrow 249\). 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

5.1.5.4.11 FB_BARShadingCorrectionSouth

<table>
<thead>
<tr>
<th>Function block for the shading evaluation of a window group on a facade.</th>
</tr>
</thead>
<tbody>
<tr>
<td>This function block is valid only for the southern hemisphere. The valid function block for the northern hemisphere is FB_BARShadingCorrection.</td>
</tr>
<tr>
<td>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.</td>
</tr>
<tr>
<td>The function block thereby accesses two lists, which are to be defined:</td>
</tr>
<tr>
<td>1. The data of the elements (window) of the facade in which the group to be regarded is located. This list of facade elements is accessed via the IN-OUT variable arrFacadeElement, which itself is globally defined.</td>
</tr>
<tr>
<td>2. 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 is likewise globally defined. The IN-OUT variable arrShadingObject accesses it directly.</td>
</tr>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

In the southern hemisphere, the following applies for the facade orientation (looking out of the window):

<table>
<thead>
<tr>
<th>Line of sight</th>
<th>Facade orientation</th>
</tr>
</thead>
<tbody>
<tr>
<td>South</td>
<td>β=0°</td>
</tr>
<tr>
<td>East</td>
<td>β=90°</td>
</tr>
<tr>
<td>North</td>
<td>β=180°</td>
</tr>
<tr>
<td>West</td>
<td>β=270°</td>
</tr>
</tbody>
</table>

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 \texttt{bFacadeSunlit} is set to FALSE.

**VAR_INPUT**

\begin{align*}
\text{eDataSecurityType} & : \text{E\_HVACDataSecurityType;} \\
\text{stTimeAct} & : \text{TIME\_STRUCT;} \\
\text{lrAzimuth} & : \text{LREAL;} \\
\text{lrElevation} & : \text{LREAL;} \\
\end{align*}

\textbf{eDataSecurityType:} If \texttt{eDataSecurityType} \[490\] : \text{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 \texttt{FB\_HVACPersistentDataHandling} \[479\] 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/zip/5090272523.zip](https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip)

If \texttt{eDataSecurityType} \[490\] : \text{eHVACDataSecurityType\_Idle} the persistently declared variables are not saved in a fail-safe manner.

\textbf{NOTE}

A cyclically changing variable must never be linked with the IN\_OUT variable of a function block, if \texttt{eDataSecurityType} \[490\] : \text{eHVACDataSecurityType\_Persistent}. It would lead to early wear of the flash memory.

\textbf{stTimeAct:} Entry of the current time in GMT (Greenwich Mean Time).

\textbf{lrAzimuth:} Sun direction in degrees at the time of observation.

\textbf{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 [315].

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/zip/5090272523.zip.

arrFacadeElement: List of facade elements [314].
arrShadingObject: List of shading objects [314].
lrFacadeOrientation: Facade orientation, see illustration above.
usiGroupId: Window group regarded. The group 0 is reserved here for unused window elements, see FB_BARFacadeElementEntry [249]. 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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 [314]. 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 [238].

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 and lrP4z are output values arises from the fact that they are formed from the input parameters:

\[
\begin{align*}
(lrP2x &= lrP1x; \quad lrP2z = lrP1z; \\
(lrP4x &= lrP3x; \quad lrP4y = lrP1y; \quad lrP4z = lrP3z; \\
\end{align*}
\]

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 be 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 \( [315] \) 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 \( FB\_BARShadingCorrection \) \( [261] \) / \( FB\_BARShadingCorrectionSouth \) \( [264] \) 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>iId</td>
<td>INT</td>
</tr>
<tr>
<td>bRead</td>
<td>BOOL</td>
</tr>
<tr>
<td>bWrite</td>
<td>BOOL</td>
</tr>
<tr>
<td>lrP1x</td>
<td>LREAL</td>
</tr>
<tr>
<td>lrP1y</td>
<td>LREAL</td>
</tr>
<tr>
<td>lrP1z</td>
<td>LREAL</td>
</tr>
<tr>
<td>lrP2y</td>
<td>LREAL</td>
</tr>
<tr>
<td>lrP3x</td>
<td>LREAL</td>
</tr>
<tr>
<td>lrP3y</td>
<td>LREAL</td>
</tr>
<tr>
<td>lrP3z</td>
<td>LREAL</td>
</tr>
<tr>
<td>lrMx</td>
<td>LREAL</td>
</tr>
<tr>
<td>lrMy</td>
<td>LREAL</td>
</tr>
<tr>
<td>lrMz</td>
<td>LREAL</td>
</tr>
<tr>
<td>lrRadius</td>
<td>LREAL</td>
</tr>
<tr>
<td>usiBeginMonth</td>
<td>USINT</td>
</tr>
<tr>
<td>usiEndMonth</td>
<td>USINT</td>
</tr>
<tr>
<td>eType</td>
<td>E_BARShadingObjectType</td>
</tr>
</tbody>
</table>

**iId**: Index of the selected element. This refers to the selection of a field element of the array saved in the IN-OUT variable \( arrShadingObject \). \( iId \) \textbf{may not be zero!} This results from the field definition \( [267] \).

**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.
IrP1z: Z-coordinate of point 1 of the shading element (square) in meters.
IrP2y: Y-coordinate of point 2 of the shading element (square) in meters.
IrP3x: X-coordinate of point 3 of the shading element (square) in meters.
IrP3y: Y-coordinate of point 3 of the shading element (square) in meters.
IrP3z: Z-coordinate of point 3 of the shading element (square) in meters.
IrMx: X-coordinate of the center of the shading element (ball) in meters.
IrMy: Y-coordinate of the center of the shading element (ball) in meters.
IrMz: Z-coordinate of the center of the shading element (ball) in meters.
IrRadius: 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
1rP2x : LREAL;
1rP2z : LREAL;
1rP4x : LREAL;
1rP4y : LREAL;
1rP4z : LREAL;
bValid : BOOL;
udiErrorId : UDINT;

IrP2x: determined X-coordinate of point 2 of the shading element (square) in metres. See "Note" above.
IrP2z: determined Z-coordinate of point 2 of the shading element (square) in metres. See "Note" above.
IrP4x: determined X-coordinate of point 4 of the shading element (square) in metres. See "Note" above.
IrP4y: determined Y-coordinate of point 4 of the shading element (square) in metres. See "Note" above.
IrP4z: 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 [315].

VAR_IN_OUT
arrShadingObject : ARRAY[1..iShadingObjects] OF ST_BARShadingObject;
arrShadingObject : List of shading objects [314]. Values are persistently saved.
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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 [289].

The total stroke time in milliseconds is limited here by the UINT format to 65535 ms. FB_BARSunblindActuatorEx [276] 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].

```plaintext
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 lowest 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 $\lambda$ 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. 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;

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

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

NOTE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType := eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.
**bEnable**: Enable input for the function block. As long as this input is TRUE, the actuator function block accepts and executes commands as described above. A FALSE signal on this input resets the control outputs **bBlindUp** and **bBlindDown** and the function block remains in a state of rest.

**stSunblind**: Positioning telegram, see [ST_BARSunblind](#).

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBlindUp</td>
<td>Control output for blind up.</td>
</tr>
<tr>
<td>bBlindDown</td>
<td>Control output for blind down.</td>
</tr>
<tr>
<td>uiActPosition</td>
<td>Current position in percent.</td>
</tr>
<tr>
<td>uiActAngle</td>
<td>Current louver angle in degrees.</td>
</tr>
<tr>
<td>bReferencing</td>
<td>The blind is in the reference-process, which means, that the output <strong>bBlindUp</strong> will be set for the complete drive-up time + 5 s. Only the manual command &quot;down&quot; can move the blind in the opposite direction and terminate the reference-mode.</td>
</tr>
<tr>
<td>bBusy</td>
<td>A positioning or a referencing procedure is in progress.</td>
</tr>
<tr>
<td>bError</td>
<td>This output is switched to TRUE if the parameters entered are erroneous.</td>
</tr>
<tr>
<td>udiErrorId</td>
<td>Contains the error code if the values entered should be erroneous. See <a href="#">Error codes</a>.</td>
</tr>
</tbody>
</table>

**VAR_IN_OUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uiBacklashTimeUp</td>
<td>Time to traverse the backlash in the upward direction in ms.</td>
</tr>
<tr>
<td>uiBacklashTimeDown</td>
<td>Time to traverse the backlash in the downward direction in ms.</td>
</tr>
<tr>
<td>uiTurningTimeUp</td>
<td>Time for turning the slats in the upward direction in ms.</td>
</tr>
<tr>
<td>uiTurningTimeDown</td>
<td>Time for turning the slats in the downward direction in ms.</td>
</tr>
<tr>
<td>uiTotalTimeUp</td>
<td>Complete time for driving up in ms.</td>
</tr>
<tr>
<td>uiTotalTimeDown</td>
<td>Complete time for driving down in ms.</td>
</tr>
<tr>
<td>uiBlindHeight</td>
<td>Shutter height in mm.</td>
</tr>
<tr>
<td>iAngleLimitUp</td>
<td>Highest position of the slats in degrees.</td>
</tr>
</tbody>
</table>

This position is reached once the blind has moved to the top position.
The slat angle $\lambda$, 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 $\lambda$, as defined above, is then typically less than zero.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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 [289].

As opposed to the function block FB_BARSunblindActuator [270], 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].

```plaintext
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 lamellae.
3) The lamellae 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 $\lambda$ 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

<table>
<thead>
<tr>
<th>E_HVACDataSecurityType;</th>
<th>BOOL;</th>
<th>ST_BARSunblind;</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>eDataSecurityType</code></td>
<td><code>bEnable</code></td>
<td><code>stSunblind</code></td>
</tr>
</tbody>
</table>

`eDataSecurityType`: If `eDataSecurityType` [490] := `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` [479] must be instanced once in the main program, which is called cyclically. Otherwise the instantiated FB is not released internally.

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

If `eDataSecurityType` [490] := `eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

NOTE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType` [490] := `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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBlindUp</td>
<td>BOOL</td>
</tr>
<tr>
<td>bBlindDown</td>
<td>BOOL</td>
</tr>
<tr>
<td>uiActPosition</td>
<td>UINT</td>
</tr>
<tr>
<td>uiActAngle</td>
<td>UINT</td>
</tr>
<tr>
<td>bReferencing</td>
<td>BOOL</td>
</tr>
<tr>
<td>bBusy</td>
<td>BOOL</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
</tr>
<tr>
<td>udiErrorId</td>
<td>UDINT</td>
</tr>
</tbody>
</table>

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 [315].

VAR_IN_OUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uiBacklashTimeUp</td>
<td>UINT</td>
</tr>
<tr>
<td>uiBacklashTimeDown</td>
<td>UINT</td>
</tr>
<tr>
<td>uiTurningTimeUp</td>
<td>UINT</td>
</tr>
<tr>
<td>uiTurningTimeDown</td>
<td>UINT</td>
</tr>
<tr>
<td>udiTotalTimeUp</td>
<td>UDINT</td>
</tr>
<tr>
<td>udiTotalTimeDown</td>
<td>UDINT</td>
</tr>
<tr>
<td>uiBlindHeight</td>
<td>UINT</td>
</tr>
<tr>
<td>iAngleLimitUp</td>
<td>INT</td>
</tr>
<tr>
<td>iAngleLimitDown</td>
<td>INT</td>
</tr>
</tbody>
</table>

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/zip/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 $\lambda$, as defined above, is then typically greater than zero.

\[ \text{iAngleLimitDown: Lowest position of the slats in degrees.} \]

This position is reached once the blind has moved to the bottom position.

The slat angle $\lambda$, as defined above, is then typically less than zero.

\[ \text{Inside} \quad \text{Outside} \]

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

#### 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 \([\text{490}]\) := 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 \([\text{479}]\) 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/zip/5090272523.zip

If eDataSecurityType \([\text{490}]\) := eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

NOTE
A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType \([\text{490}]\) := 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 \([\text{504}]\). 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 \([\text{504}]\) and serves merely to indicate whether the function block is sending an active telegram.

stSunblind : Output structure of the blind positions, see ST_BARSunblind \([\text{504}]\)

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/zip/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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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 [270] or the roller shutter function block FB_BARRollerblind [284] 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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 [289].

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 \[\text{490}\] = 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 \[\text{479}\] 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/zip/5090272523.zip

If eDataSecurityType \[\text{490}\] = eHVACDataSecurityType\_Idle the persistently declared variables are not saved in a fail-safe manner.

NOTE
A cyclically changing variable must never be linked with the IN\_OUT variable of a function block, if eDataSecurityType \[\text{490}\] = 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 \[\text{504}\].

VAR\_OUTPUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBlindUp</td>
<td>BOOL</td>
</tr>
<tr>
<td>bBlindDown</td>
<td>BOOL</td>
</tr>
<tr>
<td>uiActPosition</td>
<td>UINT</td>
</tr>
<tr>
<td>bReferencing</td>
<td>BOOL</td>
</tr>
<tr>
<td>bBusy</td>
<td>BOOL</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
</tr>
<tr>
<td>udiErrorId</td>
<td>UDINT</td>
</tr>
</tbody>
</table>

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 \[\text{315}\].

VAR\_IN\_OUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>udiTotalTimeUp</td>
<td>UDINT</td>
</tr>
<tr>
<td>udiTotalTimeDown</td>
<td>UDINT</td>
</tr>
</tbody>
</table>

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/zip/5090272523.zip.

udiTotalTimeUp: Complete time for driving up in ms.

udiTotalTimeDown: Complete time for driving down in ms.
### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

#### 5.1.5.4.18 FB_BARSunblindScene

This function block represents an extension of the manual operation FB_BARSunblindSwitch [289] by a scene memory and a call function. The blind actuator FB_BARSunblindActuator [270] or the roller shutter actuator FB_BARRollerblind [284] 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].**

```plaintext
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 [270] and FB_BARRollerblind [284] 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 [289], the scene selection function block can be connected to the blind function block either directly or via an upstream priority controller FB_BARSunblindPrioritySwitch [283]. 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;
iSetpointPosition : UINT;
iSetpointAngle : INT;

eDataSecurityType: If $\text{eDataSecurityType}_\{490\} = \text{eHVACDataSecurityType\_Persistent}$, 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 $\text{FB\_HVACPersistentDataHandling}_\{479\}$ 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/zip/5090272523.zip

If $\text{eDataSecurityType}_\{490\} = \text{eHVACDataSecurityType\_Idle}$ the persistently declared variables are not saved in a fail-safe manner.

NOTE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if $\text{eDataSecurityType}_\{490\} = \text{eHVACDataSecurityType\_Persistent}$. 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 [270]` or `FB_BARRollerblind [284]` for the reference blind/roller shutter in order to be able to save a position that was previously driven manually.

**iSetpointAngle**: dto. Slat angle in degrees.

### VAR_OUTPUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stSunblind</code></td>
<td><code>ST_BARSunblind</code></td>
</tr>
<tr>
<td><code>bActive</code></td>
<td><code>BOOL</code></td>
</tr>
<tr>
<td><code>uiPositionActScene</code></td>
<td><code>UINT</code></td>
</tr>
<tr>
<td><code>iAnglePositionActScene</code></td>
<td><code>INT</code></td>
</tr>
<tr>
<td><code>bError</code></td>
<td><code>BOOL</code></td>
</tr>
<tr>
<td><code>udiErrorId</code></td>
<td><code>UDINT</code></td>
</tr>
</tbody>
</table>

- `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.
- `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 [315].

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 [230]) 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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uiSwitchOverTime</code></td>
<td><code>UINT</code></td>
</tr>
<tr>
<td><code>arrBlindScenes</code></td>
<td><code>ARRAY[0..20] OF ST_BARSunblindScene</code></td>
</tr>
</tbody>
</table>

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_TCS_HVAC/Resources/zip/5090272523.zip](https://infosys.beckhoff.com/content/1033/TF8000_TCS_HVAC/Resources/zip/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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

#### 5.1.5.4.19 FB_BARSunblindSwitch

With the help of this function block the blind controller `FB_BARSunblindActuator`[270] or the roller shutter controller `FB_BARRollerBlind`[284] 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`[270] and `FB_BARRollerBlind`[284] 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`[283]. The connection is made via the positioning telegram `stSunblind`[504].

**Use of a priority controller:**
VAR_INPUT

eDataSecurityType : E_HVACDataSecurityType;
bEnable : BOOL;
bSwitchUp : BOOL;
bSwitchDown : BOOL;

NOTE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType := eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

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 - 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**

<table>
<thead>
<tr>
<th>stSunblind</th>
<th>ST_BARSunblind</th>
</tr>
</thead>
<tbody>
<tr>
<td>bActive</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

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/zip/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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

**5.1.5.4.20 FB_BARSunblindThermoAutomatic**

This function block controls the blind in relation to the temperature inside the room if the room is unoccupied. The objective is to block radiant heat when cooling and to increase the input of heat by opening the blinds when heating.
The 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 \( uibrightness\text{ActivateValue} \) 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 (\( bPresence = \text{TRUE} \)) or the external brightness value \( uibrightness \) falls below the value \( uibrightness\text{DeactivateValue} \) 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:

<table>
<thead>
<tr>
<th>( bComfortCoolingAsSwitchPointHeating ) = FALSE</th>
<th>( bComfortCoolingAsSwitchPointHeating ) = TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>upper limit value</td>
<td>( stBARSetpointRoom_PreComfortCool )</td>
</tr>
<tr>
<td>lower limit value</td>
<td>( stBARSetpointRoom_ComfortHeat )</td>
</tr>
</tbody>
</table>
Diagram

**example 1:** bComfortCoolingAsSwitchPointHeating = FALSE

**example 2:** bComfortCoolingAsSwitchPointHeating = TRUE

---

**Initial situation:**

The thermo-automatic is not active yet. Another function is controlling the blind.

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.

The temperature falls below the comfort-heating-value. The thermo-automatic switches over to the heating-mode and the blind will be driven to heating-position.

The temperature rises above the comfort-heating-value. The position of the blind will not be changed.

Only after reaching the upper limit (example 1: PreComfortCool and example 2: ComfortCool) the blind will be driven to the cooling-position.

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:

<table>
<thead>
<tr>
<th></th>
<th>bComfortHeatingAsSwitchPointCooling = FALSE</th>
<th>bComfortHeatingAsSwitchPointCooling = TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>upper limit value</td>
<td><code>stBARSetpointRoom_ComfortCool</code></td>
<td><code>stBARSetpointRoom_ComfortCool</code></td>
</tr>
<tr>
<td>lower limit value</td>
<td><code>stBARSetpointRoom_PreComfortHeat</code></td>
<td><code>stBARSetpointRoom_ComfortHeat</code></td>
</tr>
</tbody>
</table>
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;

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;

VAR_INPUT

eDataSecurityType: If eDataSecurityType [490] := 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 [479] 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/zip/5090272523.zip

If eDataSecurityType [490] := eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

NOTE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType [490] := 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 if 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 [291], which contains the information about the temperature switching points for the air conditioning of the room, see above.

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.

IrUpperSwitchPointHeating: Indicates the selected upper switch point in heating mode at which the blind is driven to the position uiPositionCoolingMode for cooling. This is either stBARSetpointRoom_ComfortCool or stBARSetpointRoom_PreComfortCool, depending on the preselection (see diagrams above).

IrLowerSwitchPointHeating: Indicates the lower switch point in heating mode at which the blind is driven to the position uiPositionHeatingMode for heating. This is permanently set as the value stBARSetpointRoom_ComfortHeat (see diagrams above).

IrUpperSwitchPointCooling: Indicates the 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).

IrLowerSwitchPointCooling: Indicates the selected lower switch point in cooling mode at which the blind is driven to the position uiPositionHeatingMode for heating. This is either stBARSetpointRoom_ComfortHeat or stBARSetpointRoom_PreComfortHeat, depending on the preselection (see diagrams above).

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

udiErrorId: Contains the error code if the values entered should be erroneous. See Error codes [315].

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 [230]) 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

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uiBrightnessActivateValue</td>
<td>UINT; (<em>lux</em>)</td>
</tr>
<tr>
<td>uiBrightnessDeactivateValue</td>
<td>UINT; (<em>lux</em>)</td>
</tr>
<tr>
<td>uiActivateDelay</td>
<td>UINT; (<em>sec</em>)</td>
</tr>
<tr>
<td>uiDeactivateDelay</td>
<td>UINT; (<em>sec</em>)</td>
</tr>
<tr>
<td>bComfortHeatingAsSwitchPointCooling</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bComfortCoolingAsSwitchPointHeating</td>
<td>BOOL;</td>
</tr>
<tr>
<td>uiPositionHeatingMode</td>
<td>UINT;</td>
</tr>
<tr>
<td>iAngleHeatingMode</td>
<td>INT;</td>
</tr>
<tr>
<td>uiPositionCoolingMode</td>
<td>UINT;</td>
</tr>
<tr>
<td>iAngleCoolingMode</td>
<td>INT;</td>
</tr>
</tbody>
</table>

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/zip/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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

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

**bEnable:** The function block has no function if this input is FALSE. 0 is output for both position and angle in the positioning telegram `stSunblind` 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**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>stSunblind</td>
<td>ST_BARSunblind</td>
</tr>
<tr>
<td>bActive</td>
<td>BOOL</td>
</tr>
<tr>
<td>uiRemainingTimeActivate</td>
<td>UINT</td>
</tr>
<tr>
<td>uiRemainingTimeDeactivate</td>
<td>UINT</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
</tr>
<tr>
<td>udiErrorId</td>
<td>UDINT</td>
</tr>
</tbody>
</table>

**NOTE**

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

**bActive:** Corresponds to the Boolean value `bActive` in the blind telegram `ST_BARSunblind` 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.

**iPositionActiveMode:** 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**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uisActivateValue</td>
<td>UINT</td>
</tr>
<tr>
<td>uiDeactivateValue</td>
<td>UINT</td>
</tr>
<tr>
<td>uiActivateDelay</td>
<td>UINT</td>
</tr>
<tr>
<td>uiDeactivateDelay</td>
<td>UINT</td>
</tr>
<tr>
<td>uiPositionActiveMode</td>
<td>UINT</td>
</tr>
<tr>
<td>iAngleActiveMode</td>
<td>INT</td>
</tr>
</tbody>
</table>
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/zip/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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

5.1.5.4.22 FB_BARSunblindWeatherProtection

The weather protection has the highest priority in the blind controller (see overview [230]) 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 \textit{uiPositionProtection} (height
position in percent) and \textit{iAngleProtection} (slat angle in degrees).

**VAR_INPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDataSecurityType</td>
<td>E_HVACDataSecurityType;</td>
</tr>
<tr>
<td>bEnable</td>
<td>BOOL;</td>
</tr>
<tr>
<td>lrWindSpeed</td>
<td>LREAL;</td>
</tr>
<tr>
<td>lrOutsideTemperature</td>
<td>LREAL;</td>
</tr>
<tr>
<td>bRainSensor</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>

\textbf{eDataSecurityType}: If \texttt{eDataSecurityType}[\texttt{490}] = \texttt{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 \texttt{FB\_HVACPersistentDataHandling}[\texttt{479}] 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/zip/5090272523.zip

If \texttt{eDataSecurityType}[\texttt{490}] = \texttt{eHVACDataSecurityType\_Idle} the persistently declared variables are not
saved in a fail-safe manner.

**NOTE**

A cyclically changing variable must never be linked with the IN\_OUT variable of a function block,
if \texttt{eDataSecurityType}[\texttt{490}] = \texttt{eHVACDataSecurityType\_Persistent}. It would lead to early wear of the flash
memory.

\textbf{bEnable}: The function block has no function if this input is FALSE. 0 is output for both position and angle in
the positioning telegram \texttt{ST\_Sunblind}[\texttt{504}] and \texttt{bActive} is FALSE. This means that another function takes
over control of the blind via the priority controller.

\textbf{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.

\textbf{lrOutsideTemperature}: Outside temperature in degrees Celsius.

\textbf{bRainSensor}: Input for a rain sensor.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>stSunblind</td>
<td>ST_BARSunblind;</td>
</tr>
<tr>
<td>blceAlarm</td>
<td>BOOL;</td>
</tr>
<tr>
<td>udiRemainingTimeIceAlarm</td>
<td>UDINT;</td>
</tr>
<tr>
<td>bStormAlarm</td>
<td>BOOL;</td>
</tr>
<tr>
<td>uiRemainingTimeStormDetect</td>
<td>UINT;</td>
</tr>
<tr>
<td>uiRemainingTimeStormAlarm</td>
<td>UINT;</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL;</td>
</tr>
<tr>
<td>udiErrorId</td>
<td>UDINT;</td>
</tr>
</tbody>
</table>

\textbf{stSunblind}: Output structure of the blind positions, see \texttt{ST\_BARSunblind}[\texttt{504}].

\textbf{bActive}: Corresponds to the Boolean value \texttt{bActive} in the blind telegram \texttt{ST\_BARSunblind}[\texttt{504}] and
serves merely to indicate whether the function block is sending an active telegram.

\textbf{blceAlarm}: Indicates the icing up alarm.

\textbf{udiRemainingTimeIceAlarm}: in the case of impending icing up (blceAlarm=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 (blceAlarm=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 [315].

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 [230]) 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

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>lrFrostTemperature</td>
<td>LREAL</td>
</tr>
<tr>
<td>lrWindSpeedStormUpcoming</td>
<td>LREAL</td>
</tr>
<tr>
<td>lrWindSpeedStormAbating</td>
<td>LREAL</td>
</tr>
<tr>
<td>uiDeiceTime</td>
<td>UINT</td>
</tr>
<tr>
<td>uiDelayStormUpcoming</td>
<td>UINT</td>
</tr>
<tr>
<td>uiDelayStormAbating</td>
<td>UINT</td>
</tr>
<tr>
<td>uiPositionProtection</td>
<td>UINT</td>
</tr>
<tr>
<td>iAngleProtection</td>
<td>INT</td>
</tr>
</tbody>
</table>

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/zip/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.

uiDelayStormAbating: 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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 [245]
  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) [232].

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

VAR_IN_OUT

VAR_OUTPUT

NOTE

If an error should occur, then this automatic function is deactivated and position and angle are set to 0. This means that if a priority controller is in use, another function with a lower priority (see Overview [230]) automatically takes over control of the blind. In the case of a direct connection, conversely, the blind will drive to position/angle 0.
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/zip/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:

<table>
<thead>
<tr>
<th>Line of sight</th>
<th>Facade orientation</th>
</tr>
</thead>
<tbody>
<tr>
<td>North</td>
<td>β=0°</td>
</tr>
<tr>
<td>East</td>
<td>β=90°</td>
</tr>
<tr>
<td>South</td>
<td>β=180°</td>
</tr>
<tr>
<td>West</td>
<td>β=270°</td>
</tr>
</tbody>
</table>

The following applies for the southern hemisphere:

<table>
<thead>
<tr>
<th>Line of sight</th>
<th>Facade orientation</th>
</tr>
</thead>
<tbody>
<tr>
<td>South</td>
<td>β=0°</td>
</tr>
<tr>
<td>East</td>
<td>β=90°</td>
</tr>
<tr>
<td>North</td>
<td>β=180°</td>
</tr>
<tr>
<td>West</td>
<td>β=270°</td>
</tr>
</tbody>
</table>

**lrFacadeAngle**: Facade inclination in degrees. See facade inclination [236].

**uiLouvreWidth**: Width of the slats in mm, see sketch [233].

**uiLouvreSpacing**: Slat spacing in mm, see sketch [233].

**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 [235]). 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
<tr>
<td>higher</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 [232], 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 \( \text{lrBeginRange} \) may thereby be larger than the end \( \text{lrEndRange} \); it is then regarded beyond 0°:

**Example:**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>lrAzimuth</td>
<td>10.0°</td>
</tr>
<tr>
<td>lrBeginRange</td>
<td>280.0°</td>
</tr>
<tr>
<td>lrEndRange</td>
<td>20.0°</td>
</tr>
<tr>
<td>bOut</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

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**

```plaintext
VAR_INPUT
eDataSecurityType : E_HVACDataSecurityType;
lrAzimuth : LREAL;
```
**eDataSecurityType**: If eDataSecurityType [490] = 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 [479] 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/zip/5090272523.zip

If eDataSecurityType [490] = eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

<table>
<thead>
<tr>
<th>NOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType [490] = eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.</td>
</tr>
</tbody>
</table>

**lrAzimuth**: Current azimuth angle.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bOut : BOOL;</td>
</tr>
<tr>
<td>bError : BOOL;</td>
</tr>
<tr>
<td>udiErrorId : UDINT;</td>
</tr>
</tbody>
</table>

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 [315].

**VAR_IN_OUT**

<table>
<thead>
<tr>
<th>VAR_IN_OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>lrBeginRange : LREAL;</td>
</tr>
<tr>
<td>lrEndRange : LREAL;</td>
</tr>
</tbody>
</table>

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/zip/5090272523.zip.

lrBeginRange: Beginning of the range in degrees.

lrEndRange: End of the range in degrees.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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 [232], 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, \textit{lrLowLimit}, may thereby not be greater than or equal to the upper limit, \textit{lrHighLimit}. Such entries result in an error on the output \textit{bError} – the test output \textit{bOut} is then additionally set to FALSE.

### VAR_INPUT

eDataSecurityType : E_HPPDataSecurityType;

\textbf{eDataSecurityType:} If \textit{eDataSecurityType} := \textit{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 \textit{FB\_HVACPersistentDataHandling} must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

If \textit{eDataSecurityType} := \textit{eHVACDataSecurityType\_Idle} the persistently declared variables are not saved in a fail-safe manner.

### VAR_OUTPUT

\textit{bOut}: binary delayed output of the threshold switch

\textit{bError}: this output is switched to TRUE if the parameters entered are erroneous.

\textit{udiErrorId}: contains the error code if the values entered should be erroneous. See \textbf{Error codes}. 

### VAR_IN_OUT

\textit{lrLowLimit} : LREAL;

\textit{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/zip/5090272523.zip.

\textit{lrLowLimit}: Lower limit value in degrees.

\textit{lrHighLimit}: Upper limit value in degrees.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
5.1.5.4.26 Program example

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 [283].

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 which 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 [283], then the telegram will be passed through there to the next priority.
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
5.1.5.4.27  List descriptions

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.

The declaration is global, since the administration function block FB_BARFacadeElementEntry as well as the shading correction FB_BARShadingCorrection / FB_BARShadingCorrectionSouth access this field directly by input/output variable:

```plaintext
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:

```plaintext
VAR_GLOBAL CONSTANT
  iRowsPerFacade    : INT :=10;
  iColumnsPerFacade : INT :=20;
END_VAR
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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.

The declaration is global, since the administration function block FB_BARShadingObjectsEntry as well as the shading correction FB_BARShadingCorrection / FB_BARShadingCorrectionSouth access this field directly by input/output variable:

```plaintext
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:

```plaintext
VAR_GLOBAL CONSTANT
  iShadingObjects : INT := 20;
END_VAR
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
Error Codes
0x8001 32769 FB_BARBlindPositionEntry[245]: Values for lrSunElevation are not ascending or unequal
0x8002 32770 FB_BARBlindPositionEntry[245]: Values for lrSunElevation are not within the valid range from 0°..90°
0x8003 32771 FB_BARBlindPositionEntry[245]: Values for ulBlindPosition are not within the valid range from 0%..100%
0x8004 32772 FB_BARSunProtectionEx[303]: The duration of the positioning interval is zero or exceeds 7200 s.
0x8005 32773 FB_BARSunProtectionEx[303]: The longitude entered is not within the valid range from -180°..180°.
0x8006 32774 FB_BARSunProtectionEx[303]: The latitude entered is not within the valid range from -90°..90°.
0x8007 32775 FB_BARSunProtectionEx[303]: The value for the slat spacing (uiLouvreSpacing) is greater than or equal to the value for the slat width (uiLouvreWidth).
0x8008 32776 FB_BARSunProtectionEx[303]: The "Values valid" bit (bValid) in the position table stBlindPositionTable is not set - invalid values, see FB_BARBlindPositionEntry[245].
0x8009 32777 FB_BARSunProtectionEx[303]: The value entered for the fixed shutter height (uiFixedShutterHeight) exceeds 100%.
0x801A 32778 FB_BARSunProtectionEx[303]: The value entered for the window height is zero
0x801B 32779 FB_BARSunProtectionEx[303]: The value entered for the slat spacing is zero
0x801C 32780 FB_BARSunProtectionEx[303]: The value entered for the slat width is zero
0x801D 32781 FB_BARShadingObjectsEntry[267]: 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[267]: The corners of the square are not on the same level.
0x801F 32783 FB_BARShadingObjectsEntry[267]: The z-component of P1 is less than zero. This corner would thus lie behind the facade.
0x8010 32784 FB_BARShadingObjectsEntry[267]: The z-component of P3 is less than zero. This corner would thus lie behind the facade.
0x8011 32785 FB_BARShadingObjectsEntry[267]: P1 is equal to P2. The object entered is thus not a rectangle.
0x8012 32786 FB_BARShadingObjectsEntry[267]: P1 is equal to P3. The object entered is thus not a rectangle.
0x8013 32787 FB_BARShadingObjectsEntry[267]: P1 is equal to P4. The object entered is thus not a rectangle.
0x8014 32788 FB_BARShadingObjectsEntry[267]: P2 is equal to P3. The object entered is thus not a rectangle.
0x8015 32789 FB_BARShadingObjectsEntry[267]: P2 is equal to P4. The object entered is thus not a rectangle.
0x8016 32790 FB_BARShadingObjectsEntry[267]: P3 is equal to P4. The object entered is thus not a rectangle.
0x8017 32791 FB_BARShadingObjectsEntry[267]: The radius entered is zero
0x8018 32792 FB_BARShadingObjectsEntry[267]: The z-component of the ball center is less than zero. This point would thus lie behind the facade.
0x8019 32793 FB_BARFacadeElementEntry[249]: The group index is 0, but at the same time another entry of the facade element is 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[249]: The X-component of the first corner (Corner1) is less than zero.
0x801B 32795 FB_BARFacadeElementEntry[249]: The Y-component of the first corner (Corner1) is less than zero.
0x801C 32796 FB_BARFacadeElementEntry[249]: The window width is less than or equal to zero.
0x801D 32797 FB_BARFacadeElementEntry[249]: The window height is less than or equal to zero.
0x801E 32798 FB_BARDelayedHysteresis[248]: The switch-on value bOnValue is smaller than the switch-off value bOffValue.
0x801F 32799 FB_BARWithinRangeAzimuth[306]: The beginning of the range lrBeginRange and the end of the range lrEndRange are the same.
0x8020 32800 FB_BARWithinRangeAzimuth[306]: One of the range limits entered is greater than 360° or less than 0°.
<table>
<thead>
<tr>
<th>udiErrorId (hex)</th>
<th>udiErrorId (dec)</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8021</td>
<td>32801</td>
<td>FB_BARWithinRangeAzimuth [306]: The range entered is greater than 180°.</td>
</tr>
<tr>
<td>0x8022</td>
<td>32802</td>
<td>FB_BARWithinRangeElevation [309]: The lower limit value lrLowLimit entered is greater than or equal to the upper limit value lrHighLimit.</td>
</tr>
<tr>
<td>0x8023</td>
<td>32803</td>
<td>FB_BARWithinRangeElevation [309]: The upper limit value lrHighLimit is greater than 90° or the lower limit value lrLowLimit is less than 0°.</td>
</tr>
<tr>
<td>0x8024</td>
<td>32804</td>
<td>FB_BARSunblindActuatorEntry [267]: The number of the month when shading begins or when shading ends is zero or greater than 12.</td>
</tr>
<tr>
<td>0x8025</td>
<td>32805</td>
<td>FB_BARSunblindActuatorEntry [267]: Index error! Id lies outside the permissible limits 1..iShadingObjects. See list of shading elements [314].</td>
</tr>
<tr>
<td>0x8026</td>
<td>32806</td>
<td>FB_BARSunblindActuator [270]/FB_BARSunblindActuatorEx [276]: The shutter height entered is 0.</td>
</tr>
<tr>
<td>0x8027</td>
<td>32807</td>
<td>FB_BARSunblindActuator [270]/FB_BARSunblindActuatorEx [276]: The total &quot;up&quot; stroke time (uiTotalTimeUp/udiTotalTimeUp) or the total &quot;down&quot; stroke time (uiTotalTimeDown/udiTotalTimeDown) respectively is 0 ms.</td>
</tr>
<tr>
<td>0x8028</td>
<td>30808</td>
<td>FB_BARSunblindActuator [270]/FB_BARSunblindActuatorEx [276]: The shutter &quot;Up&quot; turning time (uiTurningTimeUp) or the shutter &quot;Down&quot; turning time (uiTurningTimeDown) is 0 ms.</td>
</tr>
<tr>
<td>0x8029</td>
<td>32809</td>
<td>FB_BARSunblindActuator [270]/FB_BARSunblindActuatorEx [276]: The upper slat angle limit value (AngleLimitUp) is smaller than or equal to the lower slat angle limit value (AngleLimitDown)</td>
</tr>
<tr>
<td>0x802A</td>
<td>32810</td>
<td>FB_BARSunblindScene [286]: The selected scene number exceeds the maximum number of 20.</td>
</tr>
<tr>
<td>0x802B</td>
<td>32811</td>
<td>FB_BARSunblindThermoAutomatic [291]: The preset room temperature values are in the wrong order.</td>
</tr>
<tr>
<td>0x802C</td>
<td>32812</td>
<td>FB_BARSunblindThermoAutomatic [291]: The brightness activation limit value is smaller than or equal to the deactivation limit value.</td>
</tr>
<tr>
<td>0x802D</td>
<td>32813</td>
<td>FB_BARSunblindThermoAutomatic [291]: The blind position (height in %) of the heating mode is greater than or equal to that of the cooling mode.</td>
</tr>
<tr>
<td>0x802E</td>
<td>32814</td>
<td>FB_BARSunblindTwilightAutomatic [291]: The brightness limit value for the activation of the automatic twilight function is greater than or equal to the deactivation value.</td>
</tr>
<tr>
<td>0x802F</td>
<td>32815</td>
<td>FB_BARSunblindWeatherProtection [300]: The frost limit value is greater than 10 °C.</td>
</tr>
<tr>
<td>0x8030</td>
<td>32816</td>
<td>FB_BARSunblindWeatherProtection [300]: The limit value for the activation of the storm alarm is smaller than or equal to the deactivation value.</td>
</tr>
<tr>
<td>0x8031</td>
<td>32817</td>
<td>FB_BARSunblindWeatherProtection [300]: The input of wind force limit values of less than 0 is not permissible.</td>
</tr>
<tr>
<td>0x8032</td>
<td>32818</td>
<td>FB_BARSunblindCorrection [261]/FB_BARSunblindCorrectionSouth [264]: 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.</td>
</tr>
<tr>
<td>0x8033</td>
<td>32819</td>
<td>FB_BARSunblindCorrection [261]/FB_BARSunblindCorrectionSouth [264]: A window element in the selected group is marked as invalid (bValid=FALSE).</td>
</tr>
<tr>
<td>0x8034</td>
<td>32820</td>
<td>FB_FacadeElementEntry [249]: Index error! iColumn or iRow lies outside the permissible limits 1..iColumnsPerFacade or 1..iRowsPerFacade, respectively. See list of facade elements [314].</td>
</tr>
<tr>
<td>0x8035</td>
<td>32821</td>
<td>FB_BARSunblindProtectionEx [303]: The façade inclination entered (lrFacadeAngle) is not within the valid range of -90°..90°.</td>
</tr>
<tr>
<td>0x8036</td>
<td>32822</td>
<td>FB_BARSunblindActuator [270]/FB_BARSunblindActuatorEx [276]: The time for extending the backlash (uiBacklashTimeUp) or (uiBacklashTimeDown) respectively is 0 ms.</td>
</tr>
<tr>
<td>0x8037</td>
<td>32823</td>
<td>FB_BARRollerBlind [284]: The entire &quot;Up&quot; travel time (uiTotalTimeUp) or the total &quot;Down&quot; travel time (uiTotalTimeDown) is 0 ms.</td>
</tr>
<tr>
<td>0x8038</td>
<td>32824</td>
<td>FB_BARSMSunblindActuator: The value for wLouvreRange may not be 0.</td>
</tr>
<tr>
<td>0x8039</td>
<td>32825</td>
<td>FB_BARSMSunblindActuator: The ratio wLouvreRange / (AngleLimitUp + AngleLimitDown) must be greater than 0.</td>
</tr>
<tr>
<td>0x803A_ 0x80FF</td>
<td>32826..33023</td>
<td>reserved for shading function blocks</td>
</tr>
<tr>
<td>0x8100</td>
<td>33024</td>
<td>FB_BARLightCircuit [219]: The operation mode uiLightCtrlMode is greater than 2. However, only 0, 1 or 2 is permitted.</td>
</tr>
<tr>
<td>0x8101</td>
<td>33025</td>
<td>FB_BARLightCircuitDim [221]: The operation mode uiLightCtrlMode is greater than 2. However, only 0, 1 or 2 is permitted.</td>
</tr>
<tr>
<td>0x8102</td>
<td>33026</td>
<td>FB_BARLightCircuitDim [221]: The lower limit value, lrMinDimValue, is less than zero or greater than 100%.</td>
</tr>
</tbody>
</table>
Programming

<table>
<thead>
<tr>
<th>udiErrorId</th>
<th>udiErrorId</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8103</td>
<td>33027</td>
<td>FB_BARLightCircuitDim: The upper limit value, lrMaxDimValue, is less than zero or greater than 100%.</td>
</tr>
<tr>
<td>0x8104</td>
<td>33028</td>
<td>FB_BARLightCircuitDim: The lower limit value, lrMinDimValue, is greater than the upper limit value, lrMaxDimValue.</td>
</tr>
<tr>
<td>0x8105</td>
<td>33029</td>
<td>FB_BARLightCircuitDim: The value for &quot;Manual On&quot;, lrManualDimValue, is less than zero or greater than 100%.</td>
</tr>
<tr>
<td>0x8106</td>
<td>33030</td>
<td>FB_BARAutomaticLight: The operation mode uiLightCtrlMode is greater than 2. However, only 0, 1 or 2 is permitted.</td>
</tr>
<tr>
<td>0x8107</td>
<td>33031</td>
<td>FB_BARAutomaticLight: The value for &quot;Manual On&quot;, lrManualDimValue, is less than zero or greater than 100%.</td>
</tr>
<tr>
<td>0x8108</td>
<td>33032</td>
<td>FB_BARAutomaticLight: The light switch-on value, lrOnDimValue, is less than zero or greater than 100%.</td>
</tr>
<tr>
<td>0x8109</td>
<td>33033</td>
<td>FB_BARStraightwelleAutomatic: The operation mode uiLightCtrlMode is greater than 2. However, only 0, 1 or 2 is permitted.</td>
</tr>
<tr>
<td>0x810A</td>
<td>33034</td>
<td>FB_BARTwilightAutomatic: The operation mode uiLightCtrlMode is greater than 2. However, only 0, 1 or 2 is permitted.</td>
</tr>
<tr>
<td>0x810B</td>
<td>33035</td>
<td>FB_BARTwilightAutomatic: The value for &quot;Manual On&quot;, lrManualDimValue, is less than zero or greater than 100%.</td>
</tr>
<tr>
<td>0x810C</td>
<td>33036</td>
<td>FB_BARTwilightAutomatic: The ON value for the automatic function, lrDimOnValue, is less than zero or greater than 100%.</td>
</tr>
<tr>
<td>0x810D</td>
<td>33037</td>
<td>FB_BARTwilightAutomatic: The OFF value for the automatic function, lrDimOffValue, is less than zero or greater than 100%.</td>
</tr>
<tr>
<td>0x810E</td>
<td>33038</td>
<td>FB_BARTwilightAutomatic: The switch-on threshold value, uiSwitchOnValue, is greater than equal to the switch-off threshold value, uiSwitchOffValue.</td>
</tr>
<tr>
<td>0x810F</td>
<td>33039</td>
<td>FB_BARDaylightControl: The operation mode uiLightCtrlMode is greater than 2. However, only 0, 1 or 2 is permitted.</td>
</tr>
<tr>
<td>0x8110</td>
<td>33040</td>
<td>FB_BARConstantLightControl: The operation mode uiLightCtrlMode is greater than 2. However, only 0, 1 or 2 is permitted.</td>
</tr>
<tr>
<td>0x8111</td>
<td>33041</td>
<td>FB_BARConstantLightControl: The lower limit value, lrMinDimValue, is less than zero or greater than 100%.</td>
</tr>
<tr>
<td>0x8112</td>
<td>33042</td>
<td>FB_BARConstantLightControl: The upper limit value, lrMaxDimValue, is less than zero or greater than 100%.</td>
</tr>
<tr>
<td>0x8113</td>
<td>33043</td>
<td>FB_BARConstantLightControl: The lower limit value, lrMinDimValue, is greater than the upper limit value, lrMaxDimValue.</td>
</tr>
<tr>
<td>0x8114</td>
<td>33044</td>
<td>FB_BARConstantLightControl: The starting value of the controller, lrStartDimValue, is less than zero or greater than 100%.</td>
</tr>
<tr>
<td>0x8115</td>
<td>33045</td>
<td>FB_BARConstantLightControl: The value for &quot;Manual On&quot;, lrManualDimValue, is less than zero or greater than 100%.</td>
</tr>
<tr>
<td>0x8116...</td>
<td>33046...</td>
<td>reserved for light function blocks</td>
</tr>
<tr>
<td>0x81FF</td>
<td>33279</td>
<td></td>
</tr>
</tbody>
</table>

| 0x8200     | 33280      | FB_BARPiCtrl: The execution cycle number uiCtrlCycleCall is zero |
| 0x8201     | 33281      | FB_BARPiCtrl: The proportional band rXp is zero. |
| 0x8202     | 33282      | FB_BARPiCtrl: The lower control value limit value rYMin is greater than or equal to the upper limit value rYMax. |
| 0x8203     | 33283      | FB_BARPiCtrl: The task cycle time iTaskCycleTime is set to 0. |
| 0x8204     | 33284      | FB_HVACOptimizedOn: The outside temperature values lrOutsideTemp[i] of the pre-start function atTempChangeFunction are not in ascending order or 2 values are the same. |
| 0x8205     | 33285      | FB_HVACOptimizedOn: One or more values of the room temperature change lr-RoomTempChange[i] in the pre-start function atTempChangeFunction are higher than the upper limit value lrMaxTempChange. |
| 0x8206     | 33286      | FB_HVACOptimizedOn: One or more values of the room temperature change lr-RoomTempChange[i] in the pre-start function atTempChangeFunction are lower than the lower limit value lrMinTempChange. |
| 0x8207     | 33287      | FB_HVACOptimizedOff: The control input eCtrlFct is set neither to heating nor to cooling mode (eBARCtrlFct_Cooling). |
| 0x8208     | 33288      | FB_HVACOptimizedOff: The outside temperature values lrOutsideTemp[i] of the pre-start function atTempChangeFunction are not in ascending order or 2 values are the same. |
| 0x8209     | 33289      | FB_HVACOptimizedOff: One or more values of the room temperature change lr-RoomTempChange[i] in the pre-start function atTempChangeFunction are higher than the upper limit value lrMaxTempChange. |
udiErrorId (hex) | udiErrorId (dec) | Error
--- | --- | ---
0x820A | 33290 | **FB_HVACOptimizedOff** [431]: One or more values of the room temperature change `lr-RoomTempChange[i]` in the pre-start function `stTempChangeFunction` are lower than the lower limit value `lrMinTempChange`.
0x820B | 33291 | **FB_HVACOptimizedOff** [431]: The control input `eCtrlFct` is set neither to heating nor to cooling mode (`eBARCtrlFct_Heating` or `eBARCtrlFct_Cooling`).
0x820C | 33292 | **FB_HVACTempChangeFunctionEntry** [441]: 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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

### 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**

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

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

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

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

rOutsideTempDamped: This input variable transfers the current damped outside temperature to the function block.

bDayOperation: TRUE = day operation, FALSE = night operation.

bReset: Acknowledge input in the event of a fault.

VAR_OUTPUT

rTempSetpoint : REAL;
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**

![Temperature Setpoint graph](image)

**Conditions**

The following applies to the input of the values: X₁ > X₂ > X₃ > X₄ and Y₁ < Y₂ < Y₃ < Y₄.

Furthermore the minimum value for the setpoint of the feed temperature must be \( \leq r\text{TempSetpoint}_Y1 \) and \( \geq 0 \). The maximum value for the setpoint of the feed temperature must be \( \geq r\text{TempSetpoint}_Y4 \).

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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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.

\[
a = r \text{Slope} \times e^{(2 \times \ln(2))}
\]

\[
b = 1/\ln(4) \times \ln(1 + r \text{Curve})
\]

\[
z, u < 0 \implies z,u = 0.0001
\]

\[
r \text{TempSetpoint} = 20 + a \times e^{(1-b) \times \ln(u)}
\]

\[
z = 20 - r \text{DesignTemp};
\]

\[
u = 20 - r \text{OutsideTemp};
\]

**VAR_INPUT**

- `eDataSecurityType` : `E_HVACDataSecurityType`;
- `bSetDefault` : `BOOL`;
- `rOutsideTempDamped` : `REAL`;
- `bDayOperation` : `BOOL`;
- `bReset` : `BOOL`;

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

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

Application example: [https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip](https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip)

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

**NOTE**

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

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

**rOutsideTempDamped**: This input variable transfers the current damped outside temperature to the function block.

**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**

![Heating Curve](image)

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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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**: The function block is enabled by the PLC program with the input variable \texttt{bEnable}. If \texttt{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 \texttt{E_HVACCtrlMode} [490]).
- **rManualValue**: Manual value that is set at the output \texttt{rOutsideTempDamped} if \texttt{eCtrlMode} = \texttt{eHVACCtrlMode\_Manual}.

**VAR_OUTPUT**

- **rOutsideTempDamped**: Damped outside temperature.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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 \texttt{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 \texttt{FB_HVACHeatingCurve}. The reason for this is that the value for the night setback is taken into account in \texttt{FB_HVACHeatingCurve}.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{eDataSecurityType}</td>
<td>\texttt{E_HVACDataSecurityType};</td>
</tr>
<tr>
<td>\texttt{bSetDefault}</td>
<td>\texttt{BOOL};</td>
</tr>
<tr>
<td>\texttt{bEnable}</td>
<td>\texttt{BOOL};</td>
</tr>
<tr>
<td>\texttt{rSetpointIn}</td>
<td>\texttt{REAL};</td>
</tr>
<tr>
<td>\texttt{rOutsideTempDamped}</td>
<td>\texttt{REAL};</td>
</tr>
<tr>
<td>\texttt{dtSystemTime}</td>
<td>\texttt{DT};</td>
</tr>
<tr>
<td>\texttt{bDayOperation}</td>
<td>\texttt{BOOL};</td>
</tr>
<tr>
<td>\texttt{bFreezeProtect}</td>
<td>\texttt{BOOL};</td>
</tr>
<tr>
<td>\texttt{bReset}</td>
<td>\texttt{BOOL};</td>
</tr>
</tbody>
</table>

\texttt{eDataSecurityType}: If \texttt{eDataSecurityType} \texttt{| 490|} = \texttt{eHVACDataSecurityType\_Persistent}, the persistent \texttt{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 \texttt{FB_HVACPersistentDataHandling} \texttt{| 479|} 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/zip/5090272523.zip

If \texttt{eDataSecurityType} \texttt{| 490|} = \texttt{eHVACDataSecurityType\_Idle} the persistently declared variables are not saved in a fail-safe manner.

**NOTE**

A cyclically changing variable must never be linked with the \texttt{IN\_OUT} variable of a function block, if \texttt{eDataSecurityType} \texttt{| 490|} = \texttt{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.\[319]\.

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\[325]\ 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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>rSetpointOut</td>
<td>REAL;</td>
</tr>
<tr>
<td>bHeating</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bReqOutsideTemp</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bReqDate</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bReqPermanent</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bReqFreezeProtec</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bReqNight</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bReqDay</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bInvalidParameter</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>eSetpointHeatingMode</td>
<td>E_HVACSetpointHeatingMode;</td>
</tr>
<tr>
<td>rHeatingLimit</td>
<td>REAL;</td>
</tr>
<tr>
<td>uiOn_Day</td>
<td>UINT;</td>
</tr>
<tr>
<td>uiOn_Month</td>
<td>UINT;</td>
</tr>
<tr>
<td>uiOff_Day</td>
<td>UINT;</td>
</tr>
<tr>
<td>uiOff_Month</td>
<td>UINT;</td>
</tr>
<tr>
<td>rFreezeProtecSetpoint</td>
<td>REAL;</td>
</tr>
<tr>
<td>rNightSetback</td>
<td>REAL;</td>
</tr>
<tr>
<td>tDelayHeatingOn</td>
<td>TIME;</td>
</tr>
<tr>
<td>tDelayHeatingOff</td>
<td>TIME;</td>
</tr>
</tbody>
</table>
eSetpointHeatingMode: Enum that specifies the operation mode of the heating circuit (see E_HVACSetpointHeatingMode [493]). 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 1.

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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

#### 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**

<table>
<thead>
<tr>
<th>eDataSecurityType</th>
<th>: E_HVACDataSecurityType;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bSetDefault</td>
<td>: BOOL;</td>
</tr>
<tr>
<td>bEnable</td>
<td>: BOOL;</td>
</tr>
<tr>
<td>rSetpointIn</td>
<td>: REAL;</td>
</tr>
<tr>
<td>bReset</td>
<td>: BOOL;</td>
</tr>
</tbody>
</table>
eDataSecurityType: If eDataSecurityType := eHVACDataSecurityType_Persistent, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

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

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

bEnable: 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. \textit{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.

\textbf{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. \textit{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.

\textbf{tRampTime}: Ramp time (1s..24h) during which the final setpoint is reached in sliding steps, depending on the absolute temperature change \textit{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. \textit{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.

\textbf{Fig. 1.1:}

![Diagram of rSetpointSlide and tRampTime](image)

\textbf{Requirements}

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
<tr>
<td>higher</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\textbf{5.1.6.6 FB_HVACSummerCompensation}

```
FB_HVACSummerCompensation

-eDataSecurityType 
-bSetDefault 
-rSetpointIn 
-rSetpointExternal 
-rOutsideTemp 
-bReset 
-rSetpointHighLimit 
-rSetpointShiftAbsolute 
-eExternalMode 
-eSetpointMode
```

330 Version: 1.1 TF8000
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 \( r_{\text{SetpointShiftAbsolute}} \), then the room temperature setpoint \( r_{\text{SetpointOut}} \) 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 \( e_{\text{SetpointMode}} := e_{\text{HVACSsetpointMode\_DINLimited}} \).

**Example I:**

Room setpoint \( r_{\text{SetpointIn}} \) = 21 °C

external setpoint \( r_{\text{SetpointExternal}} \) = 2K (e.g. +/- 3K can be set via a potentiometer, but are not taken into account because \( e_{\text{ExternalMode}} := e_{\text{HVACEExternalMode\_Off}} \))

set differential value between outside temperature and room temperature \( r_{\text{SetpointShiftAbsolute}} \) = 6K

outside temperature \( r_{\text{OutsideTemp}} \) = 28 °C

\( e_{\text{ExternalMode}} := e_{\text{HVACEExternalMode\_Off}} \)

\( e_{\text{SetpointMode}} := e_{\text{HVACSsetpointMode\_DIN}} \)

calculated room temperature setpoint \( r_{\text{SetpointOut}} \) = 22

**Example II:**

Room setpoint \( r_{\text{SetpointIn}} \) = 20 °C

external setpoint \( r_{\text{SetpointExternal}} \) = -2K (e.g. +/- 3K can be set via a potentiometer).

The set differential value between outside temperature and room temperature \( r_{\text{SetpointShiftAbsolute}} \) = 6K

outside temperature \( r_{\text{OutsideTemp}} \) = 28 °C

\( e_{\text{ExternalMode}} := e_{\text{HVACEExternalMode\_On}} \)

\( e_{\text{SetpointMode}} := e_{\text{HVACSsetpointMode\_DIN}} \)

calculated room temperature setpoint \( r_{\text{SetpointOut}} \) = 20°C

**Example III:**

Room setpoint \( r_{\text{SetpointIn}} \) = 23 °C

external setpoint \( r_{\text{SetpointExternal}} \) = -3K (e.g. +/- 3K can be set via a potentiometer).

The set differential value between outside temperature and room temperature \( r_{\text{SetpointShiftAbsolute}} \) = 6K

outside temperature \( r_{\text{OutsideTemp}} \) = 34 °C

\( r_{\text{SetpointHighLimit}} \) = 35 °C

\( e_{\text{ExternalMode}} := e_{\text{HVACEExternalMode\_On}} \)

\( e_{\text{SetpointMode}} := e_{\text{HVACSsetpointMode\_DINLimited}} \)

calculated room temperature setpoint \( r_{\text{SetpointOut}} \) = 31°C

**Example IV:**

Room setpoint \( r_{\text{SetpointIn}} \) = 24 °C

external setpoint \( r_{\text{SetpointExternal}} \) = -3K (e.g. +/- 3K can be set via a potentiometer).

The set differential value between outside temperature and room temperature \( r_{\text{SetpointShiftAbsolute}} \) = 6K
outside temperature \( r_{\text{OutsideTemp}} \): 36°C

\( r_{\text{SetpointHighLimit}} \): 30 °C

e\text{ExternalMode}:= \text{eHVACExternalMode\_On}

e\text{SetpointMode}:= \text{eHVACSetpointMode\_DINLimited}

calculated room temperature setpoint: = 31 °C, but is limited to 30 °C on account of \( r_{\text{SetpointHighLimit}} \) being set to 30 °C ==> \( r_{\text{SetpointOut}} \): = 30 °C

**VAR\_INPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( e\text{DataSecurityType} )</td>
<td>( \text{E_HVACDataSecurityType} )</td>
</tr>
<tr>
<td>( b\text{SetDefault} )</td>
<td>( \text{BOOL} )</td>
</tr>
<tr>
<td>( r\text{SetpointIn} )</td>
<td>( \text{REAL} )</td>
</tr>
<tr>
<td>( r\text{SetpointExternal} )</td>
<td>( \text{REAL} )</td>
</tr>
<tr>
<td>( r\text{OutsideTemp} )</td>
<td>( \text{REAL} )</td>
</tr>
<tr>
<td>( b\text{Reset} )</td>
<td>( \text{BOOL} )</td>
</tr>
</tbody>
</table>

\( e\text{DataSecurityType} \): If \( e\text{DataSecurityType} \):= \text{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 \( \text{FB\_HVACPersistentDataHandling} \) must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000\_TC3\_HVAC/Resources/zip/5090272523.zip

If \( e\text{DataSecurityType} \):= \text{eHVACDataSecurityType\_Idle} the persistently declared variables are not saved in a fail-safe manner.

**NOTE**

A cyclically changing variable must never be linked with the IN\_OUT variable of a function block, if \( e\text{DataSecurityType} \):= \text{eHVACDataSecurityType\_Persistent}. It would lead to early wear of the flash memory.

\( b\text{SetDefault} \): If the variable is TRUE, the default values of the VAR\_IN\_OUT variables are adopted.

\( r\text{SetpointIn} \): The setpoint for the room is applied to this input.

\( r\text{SetpointExternal} \): The setpoint adjustment or correction, e.g. from a potentiometer, is applied to this input.

\( r\text{OutsideTemp} \): Input for the outside temperature.

\( b\text{Reset} \): Acknowledge input in the event of a fault.

**VAR\_OUTPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r\text{SetpointOut} )</td>
<td>( \text{REAL} )</td>
</tr>
<tr>
<td>( b\text{InvalidParameter} )</td>
<td>( \text{BOOL} )</td>
</tr>
</tbody>
</table>

\( r\text{SetpointOut} \): Setpoint for the room.

\( b\text{InvalidParameter} \): Indicates that there is an incorrect input parameter. \( b\text{InvalidParameter} \) must be acknowledged with \( b\text{Reset} \).

**VAR\_IN\_OUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r\text{SetpointHighLimit} )</td>
<td>( \text{REAL} )</td>
</tr>
<tr>
<td>( r\text{SetpointShiftAbsolute} )</td>
<td>( \text{REAL} )</td>
</tr>
<tr>
<td>( e\text{ExternalMode} )</td>
<td>( \text{E_HVACExternalMode} )</td>
</tr>
<tr>
<td>( e\text{SetpointMode} )</td>
<td>( \text{E_HVACSetpointMode} )</td>
</tr>
</tbody>
</table>

\( r\text{SetpointHighLimit} \): 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.
- **eHVACExternalMode_On** means that the external setpoint is activated, e.g. +/- 3 °C
- **eHVACExternalMode_ShiftAbsolute** 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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

**5.1.7 HVAC Special functions**

**5.1.7.1 FB_HVACAirConditioning2Speed**

![FB_HVACAirConditioning2Speed](image)

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDataSecurityType</td>
<td>ENUM</td>
</tr>
<tr>
<td>bSetDefault</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEnable</td>
<td>BOOL</td>
</tr>
<tr>
<td>bTimeScheduler</td>
<td>BOOL</td>
</tr>
<tr>
<td>rSetpoint</td>
<td>REAL</td>
</tr>
<tr>
<td>rActualValue</td>
<td>REAL</td>
</tr>
<tr>
<td>bReset</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

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

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

NOTE

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

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

bEnable: The function block is enabled by the PLC program with the input variable bEnable = TRUE. The output variable bSpeed2 is FALSE if the function block is not enabled.

bTimeScheduler: Request by a timer program for the second fan stage. If bEnable and bTimeScheduler are TRUE, then bSpeed2 and bRequestScheduler are also TRUE.

rSetpoint: Setpoint of room or exhaust 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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bSpeed2</td>
<td>BOOL</td>
</tr>
<tr>
<td>bRequestScheduler</td>
<td>BOOL</td>
</tr>
<tr>
<td>bRequestTemperature</td>
<td>BOOL</td>
</tr>
<tr>
<td>bInvalidParameter</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

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;
tDelayHysteresisOn : TIME;
tDelayHysteresisOff : 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 [488]). 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`

```
\begin{array}{cc}
\text{bEnable} &= \text{TRUE} \\
\text{eAirConditioning2SpeedMode} &= \text{eHVACAirConditioning2SpeedMode\_Cooling} \\
\end{array}
```

Operating mode `eAirConditioning2SpeedMode = eHVACAirConditioning2SpeedMode\_HeatingAndCooling`

```
\begin{array}{cc}
\text{bEnable} &= \text{TRUE} \\
\text{eAirConditioning2SpeedMode} &= \text{eHVACAirConditioning2SpeedMode\_HeatingAndCooling} \\
\end{array}
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
5.1.7.2  FB_HVACAlarm

A TRUE on the input \( b_{\text{AlarmIn}} \) indicates that an alarm signal is present. The alarm is routed through to the output \( b_{\text{Alarm}} \) only if \( b_{\text{AlarmIn}} \) is present for longer than the preset time \( t_{\text{AlarmDelay}} \). Furthermore, the input \( b_{\text{CtrlVoltage}} \) must be TRUE in order for an alarm to be announced. Output \( b_{\text{Alarm}} \) remains set to TRUE until \( b_{\text{AlarmIn}} = \text{FALSE} \) and the alarm is acknowledged by a positive edge on \( b_{\text{Reset}} \).

**VAR_INPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( e_{\text{DataSecurityType}} )</td>
<td>( \text{E_HVACDataSecurityType} )</td>
</tr>
<tr>
<td>( b_{\text{SetDefault}} )</td>
<td>( \text{BOOL} )</td>
</tr>
<tr>
<td>( b_{\text{AlarmIn}} )</td>
<td>( \text{BOOL} )</td>
</tr>
<tr>
<td>( b_{\text{CtrlVoltage}} )</td>
<td>( \text{BOOL} )</td>
</tr>
<tr>
<td>( b_{\text{Reset}} )</td>
<td>( \text{BOOL} )</td>
</tr>
</tbody>
</table>

**eDataSecurityType:** If \( e_{\text{DataSecurityType}} = \text{eHVACDataSecurityType\_Persistent} \), the persistent \( \text{VAR\_IN\_OUT} \) variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

If \( e_{\text{DataSecurityType}} = \text{eHVACDataSecurityType\_Idle} \) the persistently declared variables are not saved in a fail-safe manner.

**NOTE**

A cyclically changing variable must never be linked with the IN\_OUT variable of a function block, if \( e_{\text{DataSecurityType}} = \text{eHVACDataSecurityType\_Persistent} \). It would lead to early wear of the flash memory.

**bSetDefault:** If the variable is TRUE, the default values of the \( \text{VAR\_IN\_OUT} \) variables are adopted.

**bAlarmIn:** A TRUE on the input \( b_{\text{AlarmIn}} \) indicates that an alarm signal is present.

**bCtrlVoltage:** Via the input \( b_{\text{CtrlVoltage}} \) a check is made of whether control voltage is present.

**bReset:** Acknowledge input.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_{\text{Alarm}} )</td>
<td>( \text{BOOL} )</td>
</tr>
</tbody>
</table>

**bAlarm:** TRUE if an alarm is announced.

**VAR_IN_OUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{\text{AlarmDelay}} )</td>
<td>( \text{TIME} )</td>
</tr>
</tbody>
</table>

**tAlarmDelay:** Delay after the alarm is first signaled if, during the preset time, \( b_{\text{AlarmIn}} \) was constantly present (0s..500s). The variable is saved persistently. Preset to 500 s.
### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

#### 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;
- **rQDamper**: REAL;
- **dtSystemTime**: DT;
- **bReset**: BOOL;

**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.

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

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

**NOTE**

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

<table>
<thead>
<tr>
<th>eAntiBlockingMode</th>
<th>iHoursDowntime</th>
<th>iDayWeekly</th>
<th>iHour</th>
<th>iMinute</th>
<th>iLengthImpulse</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_HVACAntiBlockingMode;</td>
<td>INT;</td>
<td>INT;</td>
<td>INT;</td>
<td>INT;</td>
<td>INT;</td>
</tr>
</tbody>
</table>

**eHVACAntiBlockingMode**: Enumeration value that specifies the type of Antiblocking method (see E_HVACAntiBlockingMode [488]).

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**

<table>
<thead>
<tr>
<th>rQDamper</th>
<th>iQDamper</th>
<th>bQAntiBlocking</th>
<th>bInvalidParameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL;</td>
<td>INT;</td>
<td>BOOL;</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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**

```plaintext
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault : BOOL;
bEnable : BOOL;
bDisable : BOOL;
bIn : BOOL;
dtSystemTime : DT;
bReset : BOOL;
```

- **VAR_INPUT**
  - *eDataSecurityType*: If `eDataSecurityType [490] := 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 [479]` 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/zip/5090272523.zip](https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip)

  If `eDataSecurityType [490] := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

**NOTE**

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType [490] := 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;

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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

5.1.7.5 FB_HVACBlink

The function block delivers a flash sequence at the output \texttt{bBlink} that is dependent on the two adjustable times \texttt{tTimeOn} and \texttt{tTimeOff}.

**VAR_INPUT**

\begin{verbatim}
VAR_INPUT
bEnable           :REAL;
tTimeOn           :TIME;
tTimeOff          :TIME;
\end{verbatim}

- \texttt{bEnable}: Enables the function block.
- \texttt{tTimeOn}: Switch-on duration of the flash pulse, \texttt{bBlink} = TRUE.
- \texttt{tTimeOff}: Switch-off duration of the flash pulse, \texttt{bBlink} = FALSE.

**VAR_OUTPUT**

\begin{verbatim}
VAR_OUTPUT
bBlink            :BOOL;
tNextSwitching    :TIME;
\end{verbatim}

- \texttt{bBlink}: Flash output
- \texttt{tNextSwitching}: Time until the next change of state of the output \texttt{bBlink}.

**Behavior of the output value**

![Graph showing the behavior of the output value](image)

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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 \texttt{stState.udiStep} it is possible to see which step the function block is currently in.

Switch-on and switch-off sequence

If \texttt{bEn} and \texttt{bCmd} are TRUE the switch-on sequence is (\texttt{bQ_Cmd} > \texttt{bQ1} > \texttt{bQ2} \ldots \texttt{bQ8}). The switch-off sequence (\texttt{bQ8} > \texttt{bQ7} > \texttt{bQ6} \ldots \texttt{bQ_Cmd}) is active if \texttt{bEn} = TRUE and \texttt{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 (\texttt{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 \texttt{bCmd} = TRUE, then the function block starts in step 0 with the switch-on conditions of \texttt{bQ_Cmd}. For the other outputs the minimum switch-on or switch-off times are active until their expiry.
The diagram shows the time responses of step 1 or the output bQ1.
Frageze des Funktionsbausteins mit bEn = TRUE. Start der Einschaltfolge bCmd = TRUE.

Der Status der Stufe (stState.udlStep) ändert sich von 0 nach 1. Dadurch wird die Einschaltverzögerungszeit der Stufe 1 (stParam1.udlSecDelayOn) aktiviert.

Die Einschaltverzögerung der Stufe 1 ist abgelaufen. Da die Mindestausschalzeit nicht aktiv ist und das Ereignis bEvt1 TRUE ist, wird der Ausgang bQ1 TRUE. Die Mindesteinschalzeit (stParam1.udlSecDelayOff) des Ausgangs bQ1 wird aktiviert. Der Status der Stufe (stState.udlStep) ändert sich von 1 nach 2.

Die Stufe 1 (stState.udlStep) ist wieder aktiv.

Nach Ablauf der Mindesteinschalzeit (stParam1.udlSecDelayOn) wird die Ausschalverzögerung (stParam1.udlSecDelayOff) der Stufe 1 aktiviert.

Nach Ablauf der Ausschalverzögerungszeit (stParam1.udlSecDelayOff) ändert sich die Status von bQ1 auf FALSE und der Stufe (stState.udlStep) auf 0. Die Mindestausschalzeit (stParam1.udlSecDelayOff) des Ausgangs bQ1 wird aktiviert.

Der Status der Stufe (stState.udlStep) ändert sich von 0 nach 1. Die Einschaltverzögerungszeit der Stufe 1 (stParam1.udlSecDelayOn) wird aktiviert.

Die Einschaltverzögerungszeit der Stufe 1 (stParam1.udlSecDelayOn) ist abgelaufen. Der Ausgang bQ1 wird nicht eingeschaltet, weil die Mindestausschalzeit (stParam1.udlSecDelayOff) noch nicht abgelaufen ist.

Der Ausgang bQ1 wird eingeschaltet, weil die Mindestausschalzeit (stParam1.udlSecDelayOff) abgelaufen ist. Der Status der Stufe (stState.udlStep) 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 \[490\] := 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 \[479\] 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/zip/5090272523.zip

If eDataSecurityType \[490\] := eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

NOTE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType \[490\] := 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:

\[
bQ1 = bQ_Cmd\text{AND}bEvt1\text{AND}stState.udiSecRT_DelayOn = 0 \text{AND} stState.udiSecRT_MinOff[1] = 0 \text{AND} stState.udiStep = 1
\]

The event bEvt1 has no effect on the switch-off condition of bQ1.

bEvt2: The event bEvt2 counts as one of the switch-on conditions of bQ2. In the switch-on phase of bQ2, the switch-on delay of the output is active only if bEvt2 is TRUE.

Switch-on condition:

\[
bQ2 = bQ1\text{AND}bEvt2 \text{AND stState.udiSecRT_DelayOn = 0 AND stState.udiSecRT_MinOff[2] = 0 AND stState.udiStep = 2}
\]

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:

\[ bQ_3 = bQ_2 \text{ AND } \text{bEvt3 AND stState.udiSecRT_DelayOn} = 0 \text{ AND } \text{stState.udiSecRT_MinOff[3]} = 0 \text{ AND } \text{stState.udiStep} = 3 \]

The event \text{bEvt3} has no effect on the switch-off condition of \text{bQ3}.

\text{bEvt4}: The event \text{bEvt4} counts as one of the switch-on conditions of \text{bQ4}. In the switch-on phase of \text{bQ4}, the switch-on delay of the output is active only if \text{bEvt4} is TRUE.

Switch-on condition:

\[ bQ_4 = bQ_3 \text{ AND } \text{bEvt4 AND stState.udiSecRT_DelayOn} = 0 \text{ AND } \text{stState.udiSecRT_MinOff[4]} = 0 \text{ AND } \text{stState.udiStep} = 4 \]

The event \text{bEvt4} has no effect on the switch-off condition of \text{bQ4}.

\text{bEvt5}: The event \text{bEvt5} counts as one of the switch-on conditions of \text{bQ5}. In the switch-on phase of \text{bQ5}, the switch-on delay of the output is active only if \text{bEvt5} is TRUE.

Switch-on condition:

\[ bQ_5 = bQ_4 \text{ AND } \text{bEvt5 AND stState.udiSecRT_DelayOn} = 0 \text{ AND } \text{stState.udiSecRT_MinOff[5]} = 0 \text{ AND } \text{stState.udiStep} = 5 \]

The event \text{bEvt5} has no effect on the switch-off condition of \text{bQ5}.

\text{bEvt6}: The event \text{bEvt6} counts as one of the switch-on conditions of \text{bQ6}. In the switch-on phase of \text{bQ6}, the switch-on delay of the output is active only if \text{bEvt6} is TRUE.

Switch-on condition:

\[ bQ_6 = bQ_5 \text{ AND } \text{bEvt6 AND stState.udiSecRT_DelayOn} = 0 \text{ AND } \text{stState.udiSecRT_MinOff[6]} = 0 \text{ AND } \text{stState.udiStep} = 6 \]

The event \text{bEvt6} has no effect on the switch-off condition of \text{bQ6}.

\text{bEvt7}: The event \text{bEvt7} counts as one of the switch-on conditions of \text{bQ7}. In the switch-on phase of \text{bQ7}, the switch-on delay of the output is active only if \text{bEvt7} is TRUE.

Switch-on condition:

\[ bQ_7 = bQ_6 \text{ AND } \text{bEvt7 AND stState.udiSecRT_DelayOn} = 0 \text{ AND } \text{stState.udiSecRT_MinOff[7]} = 0 \text{ AND } \text{stState.udiStep} = 7 \]

The event \text{bEvt7} has no effect on the switch-off condition of \text{bQ7}.

\text{bEvt8}: The event \text{bEvt8} counts as one of the switch-on conditions of \text{bQ8}. In the switch-on phase of \text{bQ8}, the switch-on delay of the output is active only if \text{bEvt8} is TRUE.

Switch-on condition:

\[ bQ_8 = bQ_7 \text{ AND } \text{bEvt8 AND stState.udiSecRT_DelayOn} = 0 \text{ AND } \text{stState.udiSecRT_MinOff[8]} = 0 \text{ AND } \text{stState.udiStep} = 8 \]

The event \text{bEvt8} has no effect on the switch-off condition of \text{bQ8}.

\textbf{VAR_OUTPUT}

\begin{verbatim}
VAR_OUTPUT
bQ_Cmd       : BOOL;
bQ1          : BOOL;
bQ2          : BOOL;
bQ3          : BOOL;
bQ4          : BOOL;
bQ5          : BOOL;
bQ6          : BOOL;
bQ7          : BOOL;
bQ8          : BOOL;
stState      : ST_HVACmdCtrl_8State;
\end{verbatim}
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

\[ bQ7 = \text{NOT} bQ7 \text{ AND } \text{stState.udiSecRT\_DelayOff} = 0 \text{ AND } \text{stState.udiSecRT\_MinOn}[7] = 0 \text{ AND } \text{stState.udiStep} = 7 \]

**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 \( \text{stParam8} \) 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 \( \text{stState} \).

Switch-on condition

\[ bQ8 = bEvt8 \text{ AND } \text{stState.udiSecRT\_DelayOn} = 0 \text{ AND } \text{stState.udiSecRT\_MinOff}[8] = 0 \text{ AND } \text{stState.udiStep} = 8 \]

Switch-off condition

\[ bQ8 = \text{stState.udiSecRT\_DelayOff} = 0 \text{ AND } \text{stState.udiSecRT\_MinOn}[8] = 0 \text{ AND } \text{stState.udiStep} = 8 \]

**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 \( \text{ST\_HVACCmdCtrl\_8State} \) [495]).

\( \text{stState.udiSecRT\_DelayOn} \): In the switch-on phase of the outputs (\( bCmd = \text{TRUE} \)), the remaining time of the switch-on delay of the current step \( \text{stState.udiStep} \) is indicated by \( \text{stState.udiSecRT\_DelayOn} \).

Example: If \( \text{stState.udiStep} = 5 \), then the function block is in step 5. The time variable of the parameter structure \( \text{stParam5.udiSecDelayOn} \) is used for the switch-on delay of the output \( bQ5 \) and the remaining time is indicated by \( \text{udiSecRT\_DelayOn} \).

\( \text{stState.udiSecRT\_DelayOff} \): In the switch-off phase of the outputs (\( bCmd = \text{FALSE} \)) the remaining time of the switch-off delay of the current step \( \text{stState.udiStep} \) is indicated by \( \text{udiSecRT\_DelayOff} \).

Example: If \( \text{stState.udiStep} = 0 \), then the function block is in step 0. The time variable \( \text{stParamCmd.udiSecDelayOff} \) is used for the switch-off delay of the output \( bQ Cmd \) and the remaining time is indicated by \( \text{udiSecRT\_DelayOff} \).

**stState.udiStep:** State indicating which step the function block is in.

Examples:

- If \( \text{stState.udiStep} = 0 \), then the function block is in the switch-on or switch-off sequence of the output \( bQ\_Cmd \).
- If \( \text{stState.udiStep} = 1 \), then the function block is in the switch-on or switch-off sequence of the output \( bQ1 \).
- If \( \text{stState.udiStep} = 8 \), then the function block is in the switch-on or switch-off sequence of the output \( bQ8 \)

\( \text{stState.udiSecRT\_MinOn}[0] \): \( .. \ g\_iNumOfCmdCtrl\_8 \) The remaining time of the minimum switch-on duration of the outputs is indicated in the one-dimensional field (Table)

\( \text{stState.udiSecRT\_MinOn}[0..g\_iNumOfCmdCtrl\_8] \). Following the expiry of the minimum switch-on duration of an output, the output can be switched off in the current step \( \text{stState.udiStep} \).

\( \text{stState.udiSecRT\_MinOff}[0] \): \( .. \ g\_iNumOfCmdCtrl\_8 \) The remaining time of the minimum switch-off duration of the outputs is indicated in the one-dimensional field (Table)

\( \text{stState.udiSecRT\_MinOff}[0..g\_iNumOfCmdCtrl\_8] \). Following the expiry of the minimum switch-off duration of an output, the output can be switched on in the current step \( \text{stState.udiStep} \).

**VAR_IN_OUT**

\( \text{stParamCmd} \) : \( \text{ST\_HVACCmdCtrl\_8Param} \);
\( \text{stParam1} \) : \( \text{ST\_HVACCmdCtrl\_8Param} \);
\( \text{stParam2} \) : \( \text{ST\_HVACCmdCtrl\_8Param} \);
\( \text{stParam3} \) : \( \text{ST\_HVACCmdCtrl\_8Param} \);
\( \text{stParam4} \) : \( \text{ST\_HVACCmdCtrl\_8Param} \);
\( \text{stParam5} \) : \( \text{ST\_HVACCmdCtrl\_8Param} \);
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 \( bQ_{\text{CMD}} \) (see \( \text{ST}_{-}\text{HVACCmdCtrl\_8Param} \)). The variable is saved persistently.

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 \( bQ_{1} \). The variable is saved persistently.

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 \( bQ_{2} \). The variable is saved persistently.

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 \( bQ_{3} \). The variable is saved persistently.

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 \( bQ_{4} \). The variable is saved persistently.

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 \( bQ_{5} \). The variable is saved persistently.

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 \( bQ_{6} \). The variable is saved persistently.

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 \( bQ_{7} \). The variable is saved persistently.

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 \( bQ_{8} \). The variable is saved persistently.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDataSecurityType</td>
<td>E_HVACDataSecurityType;</td>
</tr>
<tr>
<td>bSetDefault</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bEnable</td>
<td>BOOL;</td>
</tr>
<tr>
<td>eCtrlModeActuator</td>
<td>E_HVACActuatorMode;</td>
</tr>
<tr>
<td>bExternalRequest</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bTimeScheduler</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bReset</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>

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

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

Application example: [https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip](https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip)

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

**NOTE**

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

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

**bEnable:** The function block is enabled if `bEnable` = TRUE. If `bEnable` = FALSE, then `bOn` is also FALSE.

**eCtrlModeActuator:** Enum that specifies the system operation modes Manual, Auto and Off (see `E_HVACActuatorMode`). 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**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bOn</td>
<td>BOOL;</td>
</tr>
<tr>
<td>eStateModeActuator</td>
<td>E_HVACActuatorMode;</td>
</tr>
<tr>
<td>bRequestScheduler</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bRequestExternal</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bInvalidParameter</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>

**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`).

**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 \textit{bExternalRequest}.

bInvalidParameter: Indicates that an incorrect parameter is present at one of the variables \textit{eCtrlModeActuator}, \textit{ePlantMode} or \textit{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 \textit{bInvalidParameter} must be acknowledged via \textit{bReset}.

VAR IN OUT

\begin{verbatim}
ePlantMode : E_HVACPlantMode;
eExternalRequestMode : E_HVACExternalRequestMode;
tOffDelay : TIME;
\end{verbatim}

\textbf{ePlantMode}: With this enumeration variable various system functions can be executed in automatic mode, depending on the input variables \textit{bExternalRequest} and \textit{bTimeScheduler} (see \textit{E_HVACPlantMode} \cite{491}).
\textit{ePlantMode} = \textit{eHVACPlantMode\_TimeSchedulingOnly}: The system is switched on and off exclusively via the input variable \textit{bTimeScheduler}.
\textit{ePlantMode} = \textit{eHVACPlantMode\_TimeScheduling\_And\_ExternalRequest}: The system is switched on if the input variables \textit{bTimeScheduler AND bExternalRequest} are TRUE.
\textit{ePlantMode} = \textit{eHVACPlantMode\_TimeScheduling\_Or\_ExternalRequest}: The system is switched on via the input variable \textit{bTimeScheduler OR bExternalRequest}.
\textit{ePlantMode} = \textit{eHVACPlantMode\_ExternalRequestOnly}: The system is switched on and off exclusively via the input variable \textit{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. \textit{bInvalidParameter} is set in the case of an incorrect parameter specification. The variable is saved persistently. Preset to 0.

\textbf{eExternalRequestMode}: The enum \textit{eExternalRequestMode} specifies the mode of operation of the input variable \textit{bExternalRequest} in automatic mode, depending on the enum \textit{ePlantMode} (see \textit{E_HVACExternalRequestMode} \cite{491}).
\textit{eExternalRequestMode} = \textit{eHVACExternalRequestMode\_ButtonOn\_Off}: The external request is set to TRUE following a rising edge on \textit{bExternalRequest}. A further rising edge resets the request to FALSE.
\textit{eExternalRequestMode} = \textit{eHVACExternalRequestMode\_ButtonOffDelay}: Following a rising edge on the input variable \textit{bExternalRequest}, the external request extends or sets the utilization time of the system by the time setting of \textit{tOffDelay}.
\textit{eExternalRequestMode} = \textit{eHVACExternalRequestMode\_SwitchOn\_Off}: The external request is active if \textit{bExternalRequest} = TRUE. It is deactivated if \textit{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. \textit{bInvalidParameter} is set in the case of an incorrect parameter specification. The variable is saved persistently. Preset to 0.

\textbf{tOffDelay}: Time value for the extension of the system utilization time. The extension of the utilization time can only be activated if \textit{eModeExternalRequest} = 2. The variable is saved persistently. Preset to 30 min.
Behavior of the output value

\[ b_{\text{Enable}} = \text{TRUE} \]
\[ e_{\text{CtrlModeActuator}} = \text{eHVACActuatorMode_Auto_BMS} \text{ OR } \text{eHVACActuatorMode_Auto_OP} \]
\[ e_{\text{PlantMode}} = \text{eHVACPlantMode_TimeSchedulingOnly} \]
\[ e_{\text{ExternalRequestMode}} = \text{eHVACExternalRequestMode_ButtonOn_Off} \]

---

\[ b_{\text{TimeScheduler}} \]
\[ b_{\text{ExternalRequest}} \]
\[ b_{\text{On}} \]
\[ b_{\text{RequestScheduler}} \]
\[ b_{\text{RequestExternal}} \]
\[ t_{\text{OffDelay}} \]

---

\[ b_{\text{Enable}} = \text{TRUE} \]
\[ e_{\text{CtrlModeActuator}} = \text{eHVACActuatorMode_Auto_BMS} \text{ OR } \text{eHVACActuatorMode_Auto_OP} \]
\[ e_{\text{PlantMode}} = \text{eHVACPlantMode_TimeSchedulingOnly} \]
\[ e_{\text{ExternalRequestMode}} = \text{eHVACExternalRequestMode_ButtonOffDelay} \]

---

\[ b_{\text{TimeScheduler}} \]
\[ b_{\text{ExternalRequest}} \]
\[ b_{\text{On}} \]
\[ b_{\text{RequestScheduler}} \]
\[ b_{\text{RequestExternal}} \]
\[ t_{\text{OffDelay}} \]
Programming

bEnable = TRUE

eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP

ePlantMode = eHVACPlantMode_TimeSchedulingOnly

eExternalRequestMode = eHVACEExternalRequestMode_SwitchOn_Off

bTimeScheduler

bExternalRequest

bOn

bRequestScheduler

bRequestExternal

tOffDelay

Time/sec

bEnable = TRUE

eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP

ePlantMode = eHVACPlantMode_ExternalRequestOnly

eExternalRequestMode = eHVACEExternalRequestMode_ButtonOn_Off

bTimeScheduler

bExternalRequest

bOn

bRequestScheduler

bRequestExternal

tOffDelay

Time/sec
bEnable = TRUE

eCtrlModeActuator = eHVACActuatorMode_Auto_BMS  OR  eHVACActuatorMode_Auto_OP

ePlantMode = eHVACPlantMode_ExternalRequestOnly

eExternalRequestMode = eHVACEnternalRequestMode_ButtonOffDelay

Time/sec

bEnable = TRUE

eCtrlModeActuator = eHVACActuatorMode_Auto_BMS  OR  eHVACActuatorMode_Auto_OP

ePlantMode = eHVACPlantMode_ExternalRequestOnly

eExternalRequestMode = eHVACEnternalRequestMode_SwitchOn_Off

Time/sec
bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeScheduling_AndExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_ButtonOffDelay

bTimeScheduler
bExternalRequest
bOn
bRequestScheduler
bRequestExternal
lOffDelay

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeScheduling_AndExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_ButtonOffDelay

bTimeScheduler
bExternalRequest
bOn
bRequestScheduler
bRequestExternal
lOffDelay
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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` = `eHVACDataSecurityType_Persistent`, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block `FB_HVACPersistentDataHandling` must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

**NOTE**

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

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

*bEnable*: The function block is enabled if `bEnable` = TRUE. If `bEnable` = FALSE, then `bOn1` and `bOn2` are also FALSE.
eCtrlModeActuator: Enum that specifies the system operation modes Manual, Auto and Off (see E_HVACActuatorMode [488]). 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 [488]).

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 [491]).

ePlantMode = eHVACPlantMode_TimeSchedulingOnly: The system is switched to the respective stage exclusively via the input variables bTimeScheduler1 or bTimeScheduler2.

bTimeScheduler1 and bTimeScheduler2 (see E_HVACPlantMode [491]).

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 $b_{\text{ExternalRequest1}}$, $b_{\text{ExternalRequest2}}$ OR $b_{\text{TimeScheduler1}}$, $b_{\text{TimeScheduler2}}$.

ePlantMode = eHVACPlantMode_ExternalRequestOnly: The system is switched to the respective stage exclusively via the input variable $b_{\text{ExternalRequest1}}$ or $b_{\text{ExternalRequest2}}$.

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. $b_{\text{InvalidParameter}}$ 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 $b_{\text{External Request1}}$ and $b_{\text{ExternalRequest2}}$ in automatic operation mode, depending on the enum ePlantMode (see E_HVACExternalRequestMode [491]).

eExternalRequestMode = eHVACExternalRequestMode_ButtonOn_Off: The external request is set to TRUE following a rising edge on $b_{\text{ExternalRequest1}}$ or $b_{\text{ExternalRequest2}}$. A further rising edge resets the request to FALSE.

eExternalRequestMode = eHVACExternalRequestMode_ButtonOffDelay. Following a rising edge on the input variable $b_{\text{ExternalRequest1}}$ or $b_{\text{ExternalRequest2}}$, the external request extends or sets the utilization time of the system in the respective stage by the time setting of $t_{\text{OffDelay1}}$ or $t_{\text{OffDelay2}}$.

eExternalRequestMode = eHVACExternalRequestMode_SwitchOn_Off: The external request is active if $b_{\text{ExternalRequest1}}$ or $b_{\text{ExternalRequest2}}$ = TRUE. It is deactivated if $b_{\text{ExternalRequest1}}$ or $b_{\text{ExternalRequest2}}$ = 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. $b_{\text{InvalidParameter}}$ 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

\[
\begin{align*}
b\text{Enable} &= \text{TRUE} \\
e\text{CtrlModeActuator} &= e\text{HVAC} \text{ActuatorMode} \text{Auto_BMS OR eHVAC} \text{ActuatorMode} \text{Auto_OP} \\
e\text{PlantMode} &= e\text{HVAC} \text{PlantMode} \text{TimeSchedulingOnly} \\
e\text{ExternalRequestMode} &= e\text{HVAC} \text{ExternalRequestMode} \text{ButtonOn_Off}
\end{align*}
\]
bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS  OR  eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeschedulingOnly
eExternalRequestMode = eHVACExternalRequestMode_ButtonOffDelay

Time/sec
Programming

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEnable = TRUE</td>
<td></td>
</tr>
<tr>
<td>eCtrlModeActuator = eHVACActuatorMode_Auto_BMS</td>
<td>OR eHVACActuatorMode_Auto_OP</td>
</tr>
<tr>
<td>ePlantMode = eHVACPlantMode_ExternalRequestOnly</td>
<td></td>
</tr>
<tr>
<td>eExternalRequestMode = eHVACExternalRequestMode_ButtonOn_Off</td>
<td></td>
</tr>
</tbody>
</table>

![Graph with time axis and various boolean functions](image)

- bTimeScheduler1
- bTimeScheduler2
- bExternalRequest1
- bExternalRequest2
- bOn1
- bOn2
- bRequestScheduler1
- bRequestScheduler2
- bRequestExternal1
- bRequestExternal2
- lOffDelay1
- lOffDelay2

Time/sec
bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS  OR  eHVACActuatorMode_Auto_OP
ePlntMode = eHVACPlntMode_ExternalRequestOnly
eExternalRequestMode = eHVACExternalRequestMode_ButtonOfDelay

Time/sec
bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_ExternalRequestOnly
eExternalRequestMode = eHVACExternalRequestMode_SwitchOn_Off

Time/sec

bTimeScheduler1
bTimeScheduler2
bExternalRequest1
bExternalRequest2
bOn1
bOn2
bRequestScheduler1
bRequestScheduler2
bRequestExternal1
bRequestExternal2
tOffDelay1
tOffDelay2
bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeScheduling_Or_ExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_ButtonOffDelay

Time/sec
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>bError</th>
<th>BOOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>eErrorCode</td>
<td>E_HVACErrorCodes;</td>
</tr>
</tbody>
</table>

**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 [490] 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 [490]).

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>rTemperature</th>
<th>REAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>rRelativeHumidity</td>
<td>REAL</td>
</tr>
<tr>
<td>rBarometricPressure</td>
<td>REAL</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>rDewPoint</th>
<th>REAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>rEnthalpy</td>
<td>REAL</td>
</tr>
<tr>
<td>rAbsoluteHumidity</td>
<td>REAL</td>
</tr>
<tr>
<td>rRho</td>
<td>REAL</td>
</tr>
<tr>
<td>rYpsilon</td>
<td>REAL</td>
</tr>
<tr>
<td>bInvalidParameter</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

## 5.1.7.11 FB_HVACFixedLimit

This function block represents a limit value switch, whose output signal is switched depending on the time delays \( t_{\text{DelayHighLimit}} \) / \( t_{\text{DelayLowLimit}} \), the mode \( b_{\text{ModeFixedLimit}} \), the limit values \( r_{\text{HighLimit}} \) / \( r_{\text{LowLimit}} \) and the input signal \( r_{\text{InputValue}} \).

### VAR_INPUT

<table>
<thead>
<tr>
<th>eDataSecurityType</th>
<th>bEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td>bSetDefault</td>
<td>bEdgeHighLimit</td>
</tr>
<tr>
<td>bEnable</td>
<td>bEdgeLowLimit</td>
</tr>
<tr>
<td>rInputValue</td>
<td>bInvalidParameter</td>
</tr>
<tr>
<td>bReset</td>
<td></td>
</tr>
<tr>
<td>bModeFixedLimit</td>
<td></td>
</tr>
<tr>
<td>rHighLimit</td>
<td></td>
</tr>
<tr>
<td>rLowLimit</td>
<td></td>
</tr>
<tr>
<td>tDelayHighLimit</td>
<td></td>
</tr>
<tr>
<td>tDelayLowLimit</td>
<td></td>
</tr>
</tbody>
</table>

- **VAR_INPUT**

  ```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault : BOOL;
bEnable : BOOL;
rInputValue : REAL;
bReset : BOOL;
```

- **eDataSecurityType**: If \( e_{\text{DataSecurityType}} \) := \( e_{\text{HVACDataSecurityType_Persistent}} \), 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 \( FB_{\text{HVACPersistentDataHandling}} \) must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

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

  Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

- If \( e_{\text{DataSecurityType}} \) := \( e_{\text{HVACDataSecurityType_Idle}} \) the persistently declared variables are not saved in a fail-safe manner.

### NOTE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if \( e_{\text{DataSecurityType}} \) := \( e_{\text{HVACDataSecurityType_Persistent}} \). 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 \( b_{\text{Enable}} = \text{TRUE} \).
- **rInputValue**: Input signal
- **bReset**: Input for acknowledging the incorrect parameter entry via a rising edge.
**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>bEvent</th>
<th>BOOL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEdgeHighLimit</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bEdgeLowLimit</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bInvalidParameter</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>

**bEvent**: Output signal, which is switched depending on the time delays \(t_{\text{DelayHighLimit}}\) / \(t_{\text{DelayLowLimit}}\), the mode \(b_{\text{ModeFixedLimit}}\), the input signal \(r_{\text{InputValue}}\) and the limit values \(r_{\text{HighLimit}}\) / \(r_{\text{LowLimit}}\).

**bEdgeHighLimit**: Is TRUE for one PLC cycle after the input signal \(r_{\text{InputValue}}\) has exceeded the variable \(r_{\text{HighLimit}}\) for the duration of \(t_{\text{DelayHighLimit}}\).

**bEdgeLowLimit**: Is TRUE for one PLC cycle after the input signal \(r_{\text{InputValue}}\) has undershot the variable \(r_{\text{LowLimit}}\) for the duration of \(t_{\text{DelayLowLimit}}\).

**bInvalidParameter**: Indicates that an incorrect parameter is present at one of the variables \(r_{\text{HighLimit}}, r_{\text{LowLimit}}, t_{\text{DelayHighLimit}}\) or \(t_{\text{DelayLowLimit}}\). 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 \(b_{\text{InvalidParameter}}\) must be acknowledged via \(b_{\text{Reset}}\).

**VAR_IN_OUT**

<table>
<thead>
<tr>
<th>bModeFixedLimit</th>
<th>BOOL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>rHighLimit</td>
<td>REAL;</td>
</tr>
<tr>
<td>rLowLimit</td>
<td>REAL;</td>
</tr>
<tr>
<td>tDelayHighLimit</td>
<td>TIME;</td>
</tr>
<tr>
<td>tDelayLowLimit</td>
<td>TIME;</td>
</tr>
</tbody>
</table>

**bModeFixedLimit**: If \(b_{\text{ModeFixedLimit}} = \text{TRUE}\) then the output variable \(b_{\text{Event}}\) becomes TRUE if the input signal \(r_{\text{InputValue}}\) has exceeded the variable \(r_{\text{HighLimit}}\) for the duration of \(t_{\text{DelayHighLimit}}\). If \(b_{\text{Event}} = \text{TRUE}\), then \(b_{\text{EdgeHighLimit}}\) is set for one PLC cycle. If the input signal \(r_{\text{InputValue}}\) undershoots the variable \(r_{\text{LowLimit}}\) for the duration of \(t_{\text{DelayLowLimit}}\), then \(b_{\text{Event}}\) becomes FALSE and \(b_{\text{EdgeLowLimit}}\) is set for one PLC cycle.

If \(b_{\text{ModeFixedLimit}} = \text{FALSE}\) then the output variable \(b_{\text{Event}}\) becomes TRUE if the input signal \(r_{\text{InputValue}}\) has undershot the variable \(r_{\text{LowLimit}}\) for the duration of \(t_{\text{DelayLowLimit}}\). If \(b_{\text{Event}} = \text{TRUE}\), then \(b_{\text{EdgeLowLimit}}\) is set for one PLC cycle. If the input signal \(r_{\text{InputValue}}\) exceeds the variable \(r_{\text{HighLimit}}\) for the duration of \(t_{\text{DelayHighLimit}}\), then \(b_{\text{Event}}\) becomes FALSE and \(b_{\text{EdgeHighLimit}}\) 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. \(b_{\text{InvalidParameter}}\) 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. \(b_{\text{InvalidParameter}}\) 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. \(b_{\text{InvalidParameter}}\) 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. \(b_{\text{InvalidParameter}}\) 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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

- `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`: Output of the state as a byte.

---

**NOTE**

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

**Application example:** [https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip](https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip)

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

```
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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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:

\[bEnable = TRUE \text{ AND } bError = FALSE\]

If `eCtrlMode = eHVACCtrlMode_Auto`, the variable `bDirection` determines whether the output variable `iValueNumber` is incremented or decremented when using the input `bEdgeNewValue`. If `bDirection = FALSE` it is incremented, if `bDirection = TRUE` it is decremented.

If `eCtrlMode = eHVACCtrlMode_Manual`, the output `iValueNumber` is determined by the variable `iManualValue`. The variables `iNumberOfValues`, `bEdgeNewValue` and `bDirection` are not taken into account in this operation mode.

The output `iValueNumber` always indicates the number of the variable `iValueX`, whose content is output in the output variable `iValueOut`.

**Examples of `eCtrlMode = eHVACCtrlMode_Auto`**

- If `bDirection = FALSE \text{ 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 \text{ 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 \text{ AND } iNumberOfValues > 0`, then `iValueNumber = 1` and `iValueOut = iValue1`.

- If `bDirection = TRUE \text{ 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 \(i\text{ManualValue} = 13\), then \(i\text{ValueNumber} = 13\) and \(i\text{ValueOut} = i\text{Value13}\).
- If \(i\text{ManualValue} = 0\), then \(i\text{ValueNumber} = 0\) and \(i\text{ValueOut} = 0\).

**NOTE**

A frequently changing variable may not be applied to the VAR_IN_OUT variables \(i\text{Value1}-8\) if \(e\text{DataSecurityType} = e\text{HVACDataSecurityType_Persistent}\). This would lead to premature wear of the storage medium of the controller. If the VAR_IN_OUT variables \(i\text{Value1}-8\) change frequently and are not to be stored persistently, then \(e\text{DataSecurityType}\) must be \(e\text{DataSecurityType_Idle}\).

**VAR_INPUT**

<table>
<thead>
<tr>
<th>eDataSecurityType : E_HVACDataSecurityType;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEnable : BOOL;</td>
</tr>
<tr>
<td>iNumberOfValues : INT;</td>
</tr>
<tr>
<td>bEdgeNewValue : BOOL;</td>
</tr>
<tr>
<td>bDirection : BOOL;</td>
</tr>
<tr>
<td>eCtrlMode : E_HVACCtrlMode;</td>
</tr>
<tr>
<td>iManualValue : INT;</td>
</tr>
</tbody>
</table>

\(e\text{DataSecurityType}\): If \(e\text{DataSecurityType} = e\text{HVACDataSecurityType_Persistent}\), 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 \(FB:\text{HVACPersistentDataHandling}\) must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

If \(e\text{DataSecurityType} = e\text{HVACDataSecurityType_Idle}\) the persistently declared variables are not saved in a fail-safe manner.

**NOTE**

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if \(e\text{DataSecurityType} = e\text{HVACDataSecurityType_Persistent}\). It would lead to early wear of the flash memory.

\(b\text{Enable}\): The function block is enabled via TRUE. If \(b\text{Enable} = FALSE\), then \(i\text{ValueOut}\) and \(i\text{ValueNumber}\) are set constantly to 0.

\(i\text{NumberOfValues}\): Indicates the number of variables \(i\text{Value1}-8\) that can be output via \(i\text{ValueOut}\). If \(i\text{NumberOfValues} = 8\) is set, then the output \(i\text{ValueNumber}\) varies between 1 and 8. The variables \(i\text{Value1}\) to \(i\text{Value8}\) are then taken into account for the output \(i\text{ValueOut}\) (see \(FB:\text{HVACMUX_INT_8}\)).

\(i\text{NumberOfValues}\) determines the start behavior of the outputs \(i\text{ValueOut}\) and \(i\text{ValueNumber}\), (see \(FB:\text{HVACMUX_INT_8}\)).

\(i\text{NumberOfValues}\) is not taken into account in the operation mode \(e\text{CtrlMode} = e\text{HVACCtrlMode_Manual}\). \(i\text{NumberOfValues}\) must not fall below 1 and must not exceed 8. Otherwise an error is indicated by \(b\text{Error} = TRUE\) and the execution of the function block is stopped.

\(b\text{EdgeNewValue}\): \(i\text{ValueNumber}\) is incremented or decremented if there is a rising edge on \(b\text{EdgeNewValue}\).

If \(b\text{Enable} = TRUE\) AND \(b\text{Error} = FALSE\) AND \(e\text{CtrlMode} = e\text{HVACCtrlMode_Auto}\) AND \(b\text{Direction} = FALSE\) AND \(i\text{NumberOfValues} > 0\) and there is a rising edge on \(b\text{EdgeNewValue}\), then \(i\text{ValueNumber}\) is incremented. If \(i\text{ValueNumber} = 6\), then \(i\text{ValueOut} = i\text{Value6}\). If \(i\text{ValueNumber} = i\text{NumberOfValues}\) and there is a rising edge on \(b\text{EdgeNewValue}\), then \(i\text{ValueNumber} = 1\) and hence \(i\text{ValueOut} = i\text{Value1}\). If \(b\text{Enable} = TRUE\) AND \(b\text{Error} = FALSE\) AND \(e\text{CtrlMode} = e\text{HVACCtrlMode_Auto}\) AND \(b\text{Direction} = TRUE\) AND \(i\text{NumberOfValues} = 6\) and there is a rising edge on \(b\text{EdgeNewValue}\), then \(i\text{ValueNumber}\) is decremented. If \(i\text{ValueNumber} = 5\), then \(i\text{ValueOut} = i\text{Value5}\). If \(i\text{ValueNumber} = 1\) and there is a rising edge on \(b\text{EdgeNewValue}\), then \(i\text{ValueNumber} = i\text{NumberOfValues}(6)\) and hence \(i\text{ValueOut} = i\text{Value6}\).

\(b\text{EdgeNewValue}\) is not taken into account in the operation mode \(e\text{CtrlMode} = e\text{HVACCtrlMode_Manual}\).
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 FB_HVACMUX_INT_8 [383]). bDirection = TRUE means that iValueNumber decrements from iNumberOfValues to 1 in descending form and thus determines the output value iValueOut (see FB_HVACMUX_INT_8 [383]). 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 variables 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 FB_HVACMUX_INT_8 [383]). 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 iValueNumber = 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 `iValueX`, whose content is output in the output variable `iValueOut`. If `eCtrlMode` = `eHVACCtrlMode_Auto`, the variable `bDirection` determines whether the output variable `iValueNumber` is incremented or decremented when using the input `bEdgeNewValue`. If `bDirection` = FALSE it is incremented, if `bDirection` = TRUE it is decremented. If `eCtrlMode` = `eHVACCtrlMode_Manual` the value of `iValueNumber` is determined by `iManualValue`.

**bError**: The output signals with a TRUE that 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 [490] 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 `FB_HVACMUX_INT_8` [383]).

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
This function block contains two different types of multiplexers, one each for the operation modes \textit{eHVACCtrlMode\_Auto} or \textit{eHVACCtrlMode\_Manual}. This selection is made with the help of the Enum \texttt{eCtrlMode} \texttt{[490]}. The following conditions must first be fulfilled for this:

\begin{align*}
bEnable &= \text{TRUE AND } bError = \text{FALSE} \\
\end{align*}

If \texttt{eCtrlMode = eHVACCtrlMode\_Auto}, the variable \texttt{bDirection} determines whether the output variable \texttt{iValueNumber} is incremented or decremented when using the input \texttt{bEdgeNewValue}. If \texttt{bDirection = FALSE} it is incremented, if \texttt{bDirection = TRUE} it is decremented.

If \texttt{eCtrlMode = eHVACCtrlMode\_Manual}, the output \texttt{iValueNumber} is determined by the variable \texttt{iManualValue}. The variables \texttt{iNumberOfValues}, \texttt{bEdgeNewValue} and \texttt{bDirection} are not taken into account in this operation mode.

The output \texttt{iValueNumber} always indicates the number of the variable \texttt{iValueX}, whose content is output in the output variable \texttt{iValueOut}.

\textbf{Examples of \texttt{eCtrlMode = eHVACCtrlMode\_Auto}}

- If \texttt{bDirection = FALSE} AND \texttt{iNumberOfValues = 12} and there is a rising edge on \texttt{bEdgeNewValue}, then \texttt{iValueNumber} is incremented. If \texttt{iValueNumber = 6}, then \texttt{iValueOut = iValue6}. If \texttt{iValueNumber = iNumberOfValues(12)} and there is a rising edge on \texttt{bEdgeNewValue}, then \texttt{iValueNumber = 1} and hence \texttt{iValueOut = iValue1}.

- If \texttt{bDirection = TRUE} AND \texttt{iNumberOfValues = 8} and there is a rising edge on \texttt{bEdgeNewValue}, then \texttt{iValueNumber} is decremented. If \texttt{iValueNumber = 5}, then \texttt{iValueOut = iValue5}. If \texttt{iValueNumber = 1} and there is a rising edge on \texttt{bEdgeNewValue}, then \texttt{iValueNumber = iNumberOfValues(8)} and hence \texttt{iValueOut = iValue8}.

The start behavior of the outputs \texttt{iValueNumber} and \texttt{iValueOut} is as follows:

- If \texttt{bDirection = FALSE} AND \texttt{iNumberOfValues > 0}, then \texttt{iValueNumber = 1} and \texttt{iValueOut = iValue1}. 

\begin{itemize}
  \item \texttt{eDateSecurityType}
  \item \texttt{bEnable}
  \item \texttt{bActive}
  \item \texttt{bError}
  \item \texttt{bDirection}
  \item \texttt{bEnable}
  \item \texttt{bActive}
  \item \texttt{bError}
  \item \texttt{bDirection}
  \item \texttt{bEnable}
  \item \texttt{bActive}
  \item \texttt{bError}
  \item \texttt{bDirection}
  \item \texttt{bEnable}
  \item \texttt{bActive}
  \item \texttt{bError}
  \item \texttt{bDirection}
  \item \texttt{bEnable}
  \item \texttt{bActive}
  \item \texttt{bError}
  \item \texttt{bDirection}
  \item \texttt{bEnable}
  \item \texttt{bActive}
  \item \texttt{bError}
  \item \texttt{bDirection}
  \item \texttt{bEnable}
  \item \texttt{bActive}
  \item \texttt{bError}
  \item \texttt{bDirection}
- If $bDirection = \text{TRUE}$ 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 $iValueOut = 0$.

### NOTE

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

<table>
<thead>
<tr>
<th>eDataSecurityType</th>
<th>E_HVACDataSecurityType</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEnable</td>
<td>BOOL</td>
</tr>
<tr>
<td>iNumberOfValues</td>
<td>INT</td>
</tr>
<tr>
<td>bEdgeNewValue</td>
<td>BOOL</td>
</tr>
<tr>
<td>bDirection</td>
<td>BOOL</td>
</tr>
<tr>
<td>eCtrlMode</td>
<td>E_HVACCtrlMode</td>
</tr>
<tr>
<td>iManualValue</td>
<td>INT</td>
</tr>
</tbody>
</table>

**eDataSecurityType:** If $eDataSecurityType \rightarrow eHVACDataSecurityType\_Persistent$, 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 \rightarrow FB\_HVACMUX\_INT_16 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/zip/5090272523.zip

If $eDataSecurityType \rightarrow eHVACDataSecurityType\_Idle$ the persistently declared variables are not saved in a fail-safe manner.

### NOTE

A cyclically changing variable must never be linked with the IN\_OUT variable of a function block, if $eDataSecurityType \rightarrow eHVACDataSecurityType\_Persistent$. It would lead to early wear of the flash memory.

**bEnable:** The function block is enabled via TRUE. If $bEnable = \text{FALSE}$, 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 FB\_HVACMUX\_INT\_16 \rightarrow FB\_HVACMUX\_INT\_16).

$iNumberOfValues$ determines the start behavior of the outputs $iValueOut$ and $iValueNumber$.

$iNumberOfValues$ is not taken into account in the operation mode $eCtrlMode = eHVACCtrlMode\_Manual$ (see FB\_HVACMUX\_INT\_16 \rightarrow FB\_HVACMUX\_INT\_16).

$iNumberOfValues$ must not fall below 1 and must not exceed 16. Otherwise an error is indicated by $bError = \text{TRUE}$ and the execution of the function block is stopped.

**bEdgeNewValue:** $iValueNumber$ is incremented or decremented if there is a rising edge on $bEdgeNewValue$.

If $bEnable = \text{TRUE}$ AND $bError = \text{FALSE}$ AND $eCtrlMode = eHVACCtrlMode\_Auto$ AND $bDirection = \text{FALSE}$ AND $iNumberOfValues > 0$ and there is a rising edge on $bEdgeNewValue$, then $iValueNumber$ is
incremented. If $i\text{ValueNumber} = 6$, then $i\text{ValueOut} = i\text{Value6}$. If $i\text{ValueNumber} = i\text{NumberOfValues}$ and there is a rising edge on $b\text{EdgeNewValue}$, then $i\text{ValueNumber} = 1$ and hence $i\text{ValueOut} = i\text{Value1}$.

If $b\text{Enable} = \text{TRUE}$ AND $b\text{Error} = \text{FALSE}$ AND $e\text{CtrlMode} = e\text{HVACCtrlMode_Auto}$ AND $b\text{Direction} = \text{TRUE}$ AND $i\text{NumberOfValues} = 12$ and there is a rising edge on $b\text{EdgeNewValue}$, then $i\text{ValueNumber}$ is decremented. If $i\text{ValueNumber} = 5$, then $i\text{ValueOut} = i\text{Value5}$. If $i\text{ValueNumber} = 1$ and there is a rising edge on $b\text{EdgeNewValue}$, then $i\text{ValueNumber} = i\text{NumberOfValues}(12)$ and hence $i\text{ValueOut} = i\text{Value6}$.

$b\text{EdgeNewValue}$ is not taken into account in the operation mode $e\text{CtrlMode} = e\text{HVACCtrlMode_Manual}$.

$b\text{Direction}$: $b\text{Direction}$ determines the control direction of the function block.

$b\text{Direction} = \text{FALSE}$ means that $i\text{ValueNumber}$ increments from 1 to $i\text{NumberOfValues}$ in ascending form and thus determines the output value $i\text{ValueOut}$ (see $\text{FB_HVACMUX_INT_16}[\text{16}][387]$).

$b\text{Direction} = \text{TRUE}$ means that $i\text{ValueNumber}$ decrements from $i\text{NumberOfValues}$ to 1 in descending form and thus determines the output value $i\text{ValueOut}$ (see $\text{FB_HVACMUX_INT_16}[\text{16}][387]$).

$b\text{Direction}$ is not taken into account in the operation mode $e\text{CtrlMode} = e\text{HVACCtrlMode_Manual}$.

$e\text{CtrlMode}$: The operation mode of the function block is specified by means of this Enum. If the operation mode $e\text{CtrlMode} = e\text{HVACCtrlMode_Auto}$, then the variable $i\text{ManualValue}$ is not taken into account (see $E\_\text{HVACCtrlMode}[490]$).

If the operation mode $e\text{CtrlMode} = e\text{HVACCtrlMode_Manual}$, then the variables $i\text{NumberOfValues}$, $b\text{EdgeNewValue}$ and $b\text{Direction}$ are not taken into account.

$i\text{ManualValue}$: If the operation mode $e\text{CtrlMode} = e\text{HVACCtrlMode_Manual}$ AND $b\text{Enable} = \text{TRUE}$ AND $b\text{Error} = \text{FALSE}$ AND $i\text{ManualValue} = 15$, then $i\text{ValueNumber} = 15$. The contents of the variable $i\text{Value15}$ are then output via $i\text{ValueOut}$ (see $\text{FB_HVACMUX_INT_16}[\text{16}][387]$).

$i\text{ManualValue}$ is not taken into account in the operation mode $e\text{CtrlMode} = e\text{HVACCtrlMode_Manual}$. The value of $i\text{ValueNumber}$ is determined by $i\text{ManualValue}$. $i\text{ManualValue}$ should lie within the range of 0 to 16. If $i\text{ManualValue}$ falls below the value 0, then 0 is output via $i\text{ValueNumber}$. If $i\text{ManualValue}$ exceeds the value 16, then 16 is output via $i\text{ValueNumber}$. If $i\text{ManualValue} = 0$, then $i\text{ValueNumber} = 0$ and $i\text{ValueOut} = 0$.

**VAR OUTPUT**

$i\text{ValueOut}$: The value of the variable $i\text{Value1-16}$ is output via $i\text{ValueOut}$.

If $b\text{Enable} = \text{TRUE}$ AND $b\text{Error} = \text{FALSE}$ AND $e\text{CtrlMode} = e\text{HVACCtrlMode_Auto}$ AND $b\text{Direction} = \text{FALSE}$ AND $i\text{NumberOfValues} > 0$ and there is a rising edge on $b\text{EdgeNewValue}$, then $i\text{ValueNumber}$ is incremented. If $i\text{ValueNumber} = 6$, then $i\text{ValueOut} = i\text{Value6}$. If $i\text{ValueNumber} = i\text{NumberOfValues}$ and there is a rising edge on $b\text{EdgeNewValue}$, then $i\text{ValueNumber} = 1$ and hence $i\text{ValueOut} = i\text{Value1}$.

If $b\text{Enable} = \text{TRUE}$ AND $b\text{Error} = \text{FALSE}$ AND $e\text{CtrlMode} = e\text{HVACCtrlMode_Auto}$ AND $b\text{Direction} = \text{TRUE}$ AND $i\text{NumberOfValues} = 12$ and there is a rising edge on $b\text{EdgeNewValue}$, then $i\text{ValueNumber}$ is decremented. If $i\text{ValueNumber} = 5$, then $i\text{ValueOut} = i\text{Value5}$. If $i\text{ValueNumber} = 1$ and there is a rising edge on $b\text{EdgeNewValue}$, then $i\text{ValueNumber} = i\text{NumberOfValues}(12)$ and hence $i\text{ValueOut} = i\text{Value7}$.

If $b\text{Enable} = \text{TRUE}$ AND $b\text{Error} = \text{FALSE}$ AND $e\text{CtrlMode} = e\text{HVACCtrlMode_Manual}$, then the value of $i\text{ValueNumber}$ is determined by $i\text{ManualValue}$. If $i\text{ManualValue} = 7$ then $i\text{ValueNumber} = 7$, which then means that $i\text{ValueOut} = i\text{Value7}$.

The start behavior of the output $i\text{ValueOut}$ is as follows: if $b\text{Enable} = \text{TRUE}$ AND $b\text{Error} = \text{FALSE}$ AND $e\text{CtrlMode} = e\text{HVACCtrlMode_Auto}$ AND $b\text{Direction} = \text{FALSE}$ AND $i\text{NumberOfValues} > 0$, then $i\text{ValueNumber} = 1$ and $i\text{ValueOut} = i\text{Value1}$.

- $e\text{CtrlMode} = e\text{HVACCtrlMode_Auto}$ AND $b\text{Direction} = \text{TRUE}$ AND $i\text{NumberOfValues} = 12$, then $i\text{ValueNumber} = 12$ and $i\text{ValueOut} = i\text{Value12}$.
- $e\text{CtrlMode} = e\text{HVACCtrlMode_Manual}$ AND $i\text{ManualValue} = 13$, then $i\text{ValueNumber} = 13$ and $i\text{ValueOut} = i\text{Value13}$.
- $e\text{CtrlMode} = e\text{HVACCtrlMode_Manual}$ AND $i\text{ManualValue} = 0$, then $i\text{ValueNumber} = 0$ and $i\text{ValueOut} = 0$.

$b\text{Active}$: $b\text{Active}$ goes TRUE if

1. $b\text{Enable} = \text{TRUE}$ AND $b\text{Error} = \text{FALSE}$ AND $e\text{CtrlMode} = e\text{HVACCtrlMode_Auto}$. 


2. \(bEnable = \text{TRUE AND } bError = \text{FALSE AND } eCtrlMode = eHVACCtrlMode\_Manual\) AND \(iManualValue > 0\).

**bEdgeNewValueOut**: Is \(\text{TRUE for one PLC cycle if } bEnable = \text{TRUE AND } bError = \text{FALSE AND } iValueNumber\) changes its value.

**iValueNumber**: The output \(iValueNumber\) always indicates the number of the variable \(iValueX\), whose content is output in the output variable \(iValueOut\). If \(eCtrlMode = eHVACCtrlMode\_Auto\), the variable \(bDirection\) determines whether the output variable \(iValueNumber\) is incremented or decremented when using the input \(bEdgeNewValue\). If \(bDirection = \text{FALSE}\) it is incremented, if \(bDirection = \text{TRUE}\) it is decremented. If \(eCtrlMode = eHVACCtrlMode\_Manual\) the value of \(iValueNumber\) is determined by \(iManualValue\).

**bError**: The output signals with a \(\text{TRUE}\) 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 \textit{error number [490]} 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 \texttt{WORD}.

\[
eHVACErrorCodes\_InvalidParam\_iNumberOfValues = 42
\]

**VAR_IN_OUT**

\[
iValue1-16 : \text{INT};
\]

**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 \texttt{FB\_HVACMUX\_INT\_16 [387]}).

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
This function block contains two different types of multiplexers, one each for the operation modes \textit{eHVACCtrlMode\_Auto} or \textit{eHVACCtrlMode\_Manual}. This selection is made with the help of the Enum \textit{eCtrlMode}.\footnote{490}

The following conditions must first be fulfilled for this:

\begin{itemize}
  \item \textit{bEnable} = TRUE AND \textit{bError} = FALSE
\end{itemize}

If \textit{eCtrlMode} = \textit{eHVACCtrlMode\_Auto}, the variable \textit{bDirection} determines whether the output variable \textit{iValueNumber} is incremented or decremented when using the input \textit{bEdgeNewValue}. If \textit{bDirection} = FALSE it is incremented, if \textit{bDirection} = TRUE it is decremented.

If \textit{eCtrlMode} = \textit{eHVACCtrlMode\_Manual}, the output \textit{iValueNumber} is determined by the variable \textit{iManualValue}. The variables \textit{iNumberOfValues}, \textit{bEdgeNewValue} and \textit{bDirection} are not taken into account in this operation mode.

The output \textit{iValueNumber} always indicates the number of the variable \textit{iValueX}, whose content is output in the output variable \textit{iValueOut}.

\textbf{Examples of \textit{eCtrlMode} = \textit{eHVACCtrlMode\_Auto}}

- If \textit{bDirection} = FALSE AND \textit{iNumberOfValues} = 7 and there is a rising edge on \textit{bEdgeNewValue}, then \textit{iValueNumber} is incremented. If \textit{iValueNumber} = 4, then \textit{rValueOut} = \textit{iValue4}. If \textit{iValueNumber} = \textit{iNumberOfValues}(7) and there is a rising edge on \textit{bEdgeNewValue}, then \textit{iValueNumber} = 1 and hence \textit{rValueOut} = \textit{rValue1}.

- If \textit{bDirection} = TRUE AND \textit{iNumberOfValues} = 8 and there is a rising edge on \textit{bEdgeNewValue}, then \textit{iValueNumber} is decremented. If \textit{iValueNumber} = 3, then \textit{rValueOut} = \textit{rValue3}. If \textit{iValueNumber} = 1 and there is a rising edge on \textit{bEdgeNewValue}, then \textit{iValueNumber} = \textit{iNumberOfValues}(8) and hence \textit{rValueOut} = \textit{rValue8}.

The start behavior of the outputs \textit{iValueNumber} and \textit{rValueOut} is as follows:

- If \textit{bDirection} = FALSE AND \textit{iNumberOfValues} > 0, then \textit{iValueNumber} = 1 and \textit{rValueOut} = \textit{rValue1}.
- If \textit{bDirection} = TRUE AND \textit{iNumberOfValues} = 6, then \textit{iNumberOfValues} = 6 and \textit{rValueOut} = \textit{rValue6}.

\textbf{Examples of \textit{eCtrlMode} = \textit{eHVACCtrlMode\_Manual}}

- The value of \textit{iValueNumber} is determined by \textit{iManualValue}. If \textit{iManualValue} = 7 then \textit{iValueNumber} = 7, which then means that \textit{rValueOut} = \textit{rValue7}.

The start behavior of the outputs \textit{iValueNumber} and \textit{rValueOut} is as follows:
- If \( i\text{ManualValue} = 13 \), then \( i\text{ValueNumber} = 13 \) and \( r\text{ValueOut} = r\text{Value13} \).
- If \( i\text{ManualValue} = 0 \), then \( i\text{ValueNumber} = 0 \) and \( r\text{ValueOut} = 0 \).

### NOTE

A frequently changing variable may not be applied to the VAR_IN_OUT variables \( r\text{Value1-n} \) if \( e\text{DataSecurityType} = e\text{HVACDataSecurityType_Persistent} \). This would lead to premature wear of the storage medium of the controller. If the VAR_IN_OUT variables \( r\text{Value1-n} \) change frequently and are not to be stored persistently, then \( e\text{DataSecurityType} \) must be \( e\text{DataSecurityType_Idle} \).

### VAR_INPUT

- **eDataSecurityType**: \( E\_\text{HVACDataSecurityType} \)
- **bEnable**: BOOL
- **iNumberOfValues**: INT
- **bEdgeNewValue**: BOOL
- **bDirection**: BOOL
- **eCtrlMode**: \( E\_\text{HVACCtrlMode} \)
- **iManualValue**: INT

\( e\text{DataSecurityType} \): If \( e\text{DataSecurityType} = e\text{HVACDataSecurityType_Persistent} \), 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 \( FB\_\text{HVACPersistentDataHandling} \) must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

If \( e\text{DataSecurityType} = e\text{HVACDataSecurityType_Idle} \) the persistently declared variables are not saved in a fail-safe manner.

### NOTE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if \( e\text{DataSecurityType} = e\text{HVACDataSecurityType_Persistent} \). It would lead to early wear of the flash memory.

- **bEnable**: The function block is enabled via TRUE. If \( b\text{Enable} = \text{FALSE} \), then \( r\text{ValueOut} \) and \( i\text{ValueNumber} \) are set constantly to 0.

  - **iNumberOfValues**: Indicates the number of variables \( r\text{Value1-8} \) that can be output via \( r\text{ValueOut} \). If \( i\text{NumberOfValues} = 8 \) is, then the output \( i\text{ValueNumber} \) varies between 1 and 8. The variables \( r\text{Value1} \) to \( r\text{Value8} \) are then taken into account for the output \( r\text{ValueOut} \) (see \( FB\_\text{HVACMUX_REAL_8} \)).

  - **iNumberOfValues** determines the start behavior of the outputs \( r\text{ValueOut} \) and \( i\text{ValueNumber} \), (see \( FB\_\text{HVACMUX_REAL_8} \)).

  - **iNumberOfValues** is not taken into account in the operation mode \( e\text{CtrlMode} = e\text{HVACCtrlMode_Manual} \).

  - **iNumberOfValues** must not fall below 1 and must not exceed 8. Otherwise an error is indicated by \( b\text{Error} = \text{TRUE} \) and the execution of the function block is stopped.

- **bEdgeNewValue**: \( i\text{ValueNumber} \) is incremented or decremented if there is a rising edge on \( b\text{EdgeNewValue} \).

  - If \( b\text{Enable} = \text{TRUE} \) AND \( b\text{Error} = \text{FALSE} \) AND \( e\text{CtrlMode} = e\text{HVACCtrlMode_Auto} \) AND \( b\text{Direction} = \text{FALSE} \) AND \( i\text{NumberOfValues} > 0 \) and there is a rising edge on \( b\text{EdgeNewValue} \), then \( i\text{ValueNumber} \) is incremented. If \( i\text{ValueNumber} = 6 \), then \( r\text{ValueOut} = r\text{Value6} \). If \( i\text{ValueNumber} = i\text{NumberOfValues} \) and there is a rising edge on \( b\text{EdgeNewValue} \), then \( i\text{ValueNumber} = 1 \) and hence \( r\text{ValueOut} = r\text{Value1} \). If \( b\text{Enable} = \text{TRUE} \) AND \( b\text{Error} = \text{FALSE} \) AND \( e\text{CtrlMode} = e\text{HVACCtrlMode_Auto} \) AND \( b\text{Direction} = \text{TRUE} \) AND \( i\text{NumberOfValues} = 6 \) and there is a rising edge on \( b\text{EdgeNewValue} \), then \( i\text{ValueNumber} \) is decremented.

  - If \( i\text{ValueNumber} = 5 \), then \( r\text{ValueOut} = r\text{Value5} \). If \( i\text{ValueNumber} = 1 \) and there is a rising edge on \( b\text{EdgeNewValue} \), then \( i\text{ValueNumber} = i\text{NumberOfValues}(6) \) and hence \( r\text{ValueOut} = r\text{Value6} \).

  - \( b\text{EdgeNewValue} \) is not taken into account in the operation mode \( e\text{CtrlMode} = e\text{HVACCtrlMode_Manual} \).
**bDirection**: The function block determines the control direction of the function block.

- If **bDirection** = TRUE, **iValueNumber** increments from 1 to **iNumberOfValues** in ascending form and thus determines the output value **rValueOut** (see **FB_HVACMUX_REAL_8** [391]).
- If **bDirection** = FALSE, **iValueNumber** decrements from **iNumberOfValues** to 1 in descending form and thus determines the output value **rValueOut** (see **FB_HVACMUX_REAL_8** [391]).

**eCtrlMode** 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_Manual**, then the variable **iManualValue** is not taken into account.

If the operation mode **eCtrlMode** = **eHVACCtrlMode_Auto**, 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 **rValueOut** are then output via **rValueOut** (see **FB_HVACMUX_REAL_8** [391]). **iManualValue** is not taken into account in the operation mode **eCtrlMode** = **eHVACCtrlMode_Manual**. The value of **iValueNumber** is determined by **iManualValue**. **ManualValue** 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**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>rValueOut</td>
<td>INT</td>
</tr>
<tr>
<td>bActive</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEdgeNewValueOut</td>
<td>BOOL</td>
</tr>
<tr>
<td>iValueNumber</td>
<td>INT</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
</tr>
<tr>
<td>eErrorCode</td>
<td>E_HVACErrorCode</td>
</tr>
</tbody>
</table>

**rValueOut**: The value of the variable **rValueOut** 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** and **iValueNumber** is incremented. If **iValueNumber** = 6, then **rValueOut** = **rValue6**. And 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 **iValueNumber** = 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 `rValueX`, whose content is output in the output variable `rValueOut`. If `eCtrlMode = eHVACCtrlMode_Auto`, the variable `bDirection` determines whether the output variable `iValueNumber` is incremented or decremented when using the input `bEdgeNewValue`. If `bDirection = FALSE` it is incremented, if `bDirection = TRUE` it is decremented. If `eCtrlMode = eHVACCtrlMode_M 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 [490] 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 FB_HVACMUX_REAL_8 [391]). The variable is saved persistently.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
This function block contains two different types of multiplexers, one each for the operation modes `eHVACCtrlMode_Auto` or `eHVACCtrlMode_Manual`. This selection is made with the help of the Enum `eCtrlMode`. The following conditions must first be fulfilled for this:

\[
\text{bEnable} = \text{TRUE AND bError} = \text{FALSE}
\]

If `eCtrlMode = eHVACCtrlMode_Auto`, the variable `bDirection` determines whether the output variable `iValueNumber` is incremented or decremented when using the input `bEdgeNewValue`. If `bDirection = FALSE` it is incremented, if `bDirection = TRUE` it is decremented.

If `eCtrlMode = eHVACCtrlMode_Manual`, the output `iValueNumber` is determined by the variable `iManualValue`. The variables `iNumberOfValues`, `bEdgeNewValue` and `bDirection` are not taken into account in this operation mode.

The output `iValueNumber` always indicates the number of the variable `iValueX`, whose content is output in the output variable `rValueOut`.

**Examples of `eCtrlMode = eHVACCtrlMode_Auto`**

- If `bDirection = FALSE AND iNumberOfValues = 12` and there is a rising edge on `bEdgeNewValue`, then `iValueNumber` increments. If `iValueNumber = 6`, then `rValueOut = rValue6`. If `iValueNumber = iNumberOfValues(12)` and there is a rising edge on `bEdgeNewValue`, then `iValueNumber = 1` and hence `rValueOut = rValue1`.

- If `bDirection = TRUE AND iNumberOfValues = 8` and there is a rising edge on `bEdgeNewValue`, then `iValueNumber` decrements. If `iValueNumber = 5`, then `rValueOut = rValue5`. If `iValueNumber = 1` and there is a rising edge on `bEdgeNewValue`, then `iValueNumber = iNumberOfValues(8)` and 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 $rValueOut = 0$.

**NOTE**

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

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

**NOTE**

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

$bEnable$: The function block is enabled via TRUE. If $bEnable = FALSE$, then $rValueOut$ 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 $FB_HVACMUX_REAL_16$).

$iNumberOfValues$ determines the start behavior of the outputs $rValueOut$ and $iValueNumber$, (see $FB_HVACMUX_REAL_16$).

$iNumberOfValues$ 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 FB_HVACMUX_REAL_16 [395]).

bDirection = TRUE means that iValueNumber decrements from iNumberOfValues to 1 in descending form and thus determines the output value rValueOut (see FB_HVACMUX_REAL_16 [395]).

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 FB_HVACMUX_REAL_16 [395]).

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. \(bEnable = \text{TRUE} \text{ AND } bError = \text{FALSE} \text{ AND } eCtrlMode = eHVACCtrlMode\_Auto.\)

2. \(bEnable = \text{TRUE} \text{ AND } bError = \text{FALSE} \text{ AND } eCtrlMode = eHVACCtrlMode\_Manual \text{ AND } iManualValue > 0.\)

\(bEdgeNewValueOut: \) Is TRUE for one PLC cycle if \(bEnable = \text{TRUE} \text{ AND } bError = \text{FALSE} \text{ AND } iValueNumber\) changes its value.

\(iValueNumber: \) The output \(iValueNumber\) always indicates the number of the variable \(rValueX\), whose content is output in the output variable \(rValueOut\).

If \(eCtrlMode = eHVACCtrlMode\_Auto\), the variable \(bDirection\) determines whether the output variable \(iValueNumber\) is incremented or decremented when using the input \(bEdgeNewValue\). If \(bDirection = \text{FALSE}\) it is incremented, if \(bDirection = \text{TRUE}\) 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 [490] 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.

\(eHVACErrorCodes\_InvalidParam\_iNumberOfValues = 42\)

\textbf{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 \(FB\_HVACMUX\_REAL\_16\) [395]). The variable is saved persistently.

\textbf{Requirements}

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

\textbf{5.1.7.17} \(FB\_HVACOverwriteAnalog\)

This function block overrides the input \(rValueIn\) if \(eCtrlMode = eHVACCtrlMode\_Manual\) and forwards \(rManualValueIn\) to the output \(rValueOut\).

\textbf{VAR\_INPUT}

\(rValueIn\) :REAL;

\(eCtrlMode\) :E\_HVACCtrlMode;

\(rManualValueIn\) :REAL;

\(rValueIn: \) Analog input value that is forwarded to the output \(rValueOut\) if \(eCtrlMode = eHVACCtrlMode\_Auto.\)
**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: Analog output value.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

#### 5.1.7.18 FB_HVACOverwriteDigital

![FB_HVACOverwriteDigital](image)

This function block overrides the state of the input bIn if eCtrlMode = eHVACCtrlMode_Manual and forwards it to the output bOut.

### VAR_INPUT

- bIn: Digital input variable whose state is forwarded to the output bOut if eCtrlMode = 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 eCtrlMode = eHVACCtrlMode_Manual.

### VAR_OUTPUT

- bOut: Digital output variable.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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.

```plaintext
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
  diQg                      : DINT;   kvar  Qges Auflösung: 0,1 kvar
  dummy                     : DINT;   Reserve, füllt die Struktur auf 120Bytes auf
END_STRUCT
```

```plaintext
VAR_INPUT
  wDataL1, wDataL2, wDataL3 : WORD;
  bySBL1, bySBL2, bySBL3 : BYTE;
  diCurrTransFactor : DINT;
  eModePowerMeasurement : E_HVACPowersMeasurementMode;
  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).

TYPE E_HVACPowerMeasurementMode:
{
eHVACPowerMeasurementMode_AutoAllValues := 0,
eHVACPowerMeasurementMode_Current := 1,
eHVACPowerMeasurementMode_Voltage := 2,
eHVACPowerMeasurementMode_EffectivePower := 3,
eHVACPowerMeasurementMode_PowerFactor := 4,
eHVACPowerMeasurementMode_EnergyConsumption := 5,
eHVACPowerMeasurementMode_PeakCurrentValue := 6, 1)
eHVACPowerMeasurementMode_PeakVoltageValue := 7, 1)
eHVACPowerMeasurementMode_PeakPowerValue := 8)
};
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).
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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 $\cos \phi$ 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**

```plaintext
STRUCT
  fIL1, fIL2, fIL3 : LREAL; A Jeff 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; kVA Sges Scheinleistung
  fWG : 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 [497]).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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 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 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.

**NOTE**
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`.
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 column 1), then iValueOut = iValue1.

Table 3: Multiplexer

If bEnable = TRUE AND eCtrlMode = eHVACCtrlMode_Manual AND iManualValue = 2, then iValueOut = 2.

VAR_INPUT

eDataSecurityType : E_HVACDataSecurityType;
bEnable : BOOL;
bEvent1 - bEvent8 : BOOL;
eCtrlMode : E_HVACCtrlMode;
iManualValue : INT; 0..8
bDirection : BOOL;

eDataSecurityType: If eDataSecurityType := eHVACDataSecurityType_Persistent, the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block FB_HVACPersistentDataHandling must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.
A change of value can be initiated by the building management system, a local operating device or via a write access from TwinCAT. When the computer is restarted the saved data are automatically read back from the flash in the RAM.

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

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

bEnable: The function block is enabled via TRUE. If bEnable = FALSE, then iValueOut is set constantly to 0.

eCtrlMode: This enum specifies the operation mode of the function block (see E_HVACCtrlMode[*]). 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[*] and Table 2[*]. 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 iMenuProfile = 5, then iValueOut = iValue5, see Table 3[*].

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 iMenuProfile = 5, then iValueOut = iValue5, see Table 3[*].

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[*] 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[*] 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 [404] 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 [404] 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 [405].

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

iVarValueNumber: The variable iVarValueNumber indicates the variable from which the value on iVarValueOut is output. If iVarValueOut = iVarValue7, then iVarValueNumber = 7, see Table 1 [404], Table 2 [405] and Table 3 [405].

VAR_IN_OUT
iVarValue1-8 : INT;

iVarValue1-8: The value of the output variable iVarValueOut is determined by the variables iVarValue1 to iVarValue8. iVarValueOut = iVarValueX

If the function block is used to prioritize events, then each of the variables iVarValue1-8 is assigned to an event. iVarValue1 is assigned to the event bEvent1, iVarValue2 to the event bEvent2, iVarValue3 to the event bEvent3, ... and iVarValue8 to the event bEvent8

If the function block is used as a multiplexer, then each of the variables iVarValue1-8 is assigned to the value of iManualValue. If iManualValue = 1, then iVarValueOut = iVarValue1. If iManualValue = 2, then iVarValueOut = iVarValue2,... if iManualValue = 8, then iVarValueOut = iVarValue8.

The variable is saved persistently.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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 [p. 409] 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 [p. 409] column 1), then `iValueOut` has the value of `iValue1`.
The function block can be used as a multiplexer if \( e\text{CtrlMode} = \text{eHVACCtrlMode\_Manual} \). The value of \( i\text{ManualValue} \) refers to one of the VAR\_IN\_OUT variables \( i\text{Value1-16} \), whose value is output via \( i\text{ValueOut} \). If \( b\text{Enable} = \text{TRUE AND eCtrlMode} = \text{eHVACCtrlMode\_Manual AND iManualValue} = 2 \), then \( i\text{ValueOut} = i\text{Value2} \), see Table 3. 

NOTE
A frequently changing variable may not be applied to the VAR\_IN\_OUT variables \( i\text{Value1-16} \) if \( e\text{DataSecurityType} = \text{eHVACDataSecurityType\_Persistent} \). This would lead to premature wear of the storage medium of the controller. If the VAR\_IN\_OUT variables \( i\text{Value1-16} \) change frequently and are not to be stored persistently, then \( e\text{DataSecurityType} \) must be \( \text{eDataSecurityType\_Idle} \).

### Table 1: Prioritization of events in ascending form from 1 to 16

It can be seen in Table 1 that \( b\text{Direction} = \text{FALSE} \). This means that the events determine the output value \( i\text{ValueOut} \) in ascending form from 1 to 16. If \( b\text{Enable} = \text{TRUE AND eCtrlMode} = \text{eHVACCtrlMode\_Auto AND bDirection} = \text{FALSE AND bEvent15} = \text{TRUE} \) (highest occurring event in column 1), then \( i\text{ValueOut} = i\text{Value15} \).

<table>
<thead>
<tr>
<th>( b\text{Enable} = \text{TRUE} )</th>
<th>( e\text{CtrlMode} = \text{eHVACCtrlMode_Auto} )</th>
<th>( b\text{Direction} = \text{FALSE} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b\text{Event1} )</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>( b\text{Event2} )</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>( b\text{Event3} )</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>( b\text{Event4} )</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>( b\text{Event5} )</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>( b\text{Event6} )</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>( b\text{Event7} )</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>( b\text{Event8} )</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>( b\text{Event9} )</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>( b\text{Event10} )</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>( b\text{Event11} )</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>( b\text{Event12} )</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>( b\text{Event13} )</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>( b\text{Event14} )</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>( b\text{Event15} )</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>( b\text{Event16} )</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( i\text{ValueOut} )</th>
<th>Value15</th>
<th>Value8</th>
<th>Value16</th>
<th>Value14</th>
<th>0</th>
<th>Value15</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i\text{ValueNumber} )</td>
<td>15</td>
<td>8</td>
<td>16</td>
<td>14</td>
<td>0</td>
<td>16</td>
</tr>
</tbody>
</table>

### Table 2: Prioritization of events in descending form from 16 to 1

It can be seen in Table 2 that \( b\text{Direction} = \text{TRUE} \). This means that the events determine the output value \( i\text{ValueOut} \) in descending form from 16 to 1. If \( b\text{Enable} = \text{TRUE AND eCtrlMode} = \text{eHVACCtrlMode\_Auto AND bDirection} = \text{TRUE AND bEvent1} = \text{TRUE} \) (lowest occurring event in Table 2), then \( i\text{ValueOut} = i\text{Value1} \).
### Table 3: Multiplexer

If `$bEnable = TRUE AND eCtrlMode = eHVACCtrlMode_Manual AND iManualValue = 2` then `$iValueOut = 2`.

<table>
<thead>
<tr>
<th>bEnable</th>
<th>TRUE</th>
<th>TRUE</th>
<th>FALSE</th>
<th>FALSE</th>
<th>FALSE</th>
<th>TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEvent1</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent2</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent3</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent4</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent5</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent6</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent7</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent8</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent9</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent10</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent11</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent12</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent13</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent14</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent15</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent16</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

### VAR_INPUT

- `eDataSecurityType` : `E_HVACDataSecurityType`
- `bEnable` : `BOOL`
- `bEvent1 - bEvent8` : `BOOL`
- `eCtrlMode` : `E_HVACCtrlMode`
- `iManualValue` : `INT` 0..16
- `bDirection` : `BOOL`

**eDataSecurityType:** If `eDataSecurityType[490]` = `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[479]` must be instantiated once in the main program, which is called cyclically. Otherwise the instantiated FB is not released internally.

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

If `eDataSecurityType[490]` = `eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

### NOTE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType[490]` = `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 $eCtrlMode = eHVACCtlrMode_Auto$, then the function block represents a prioritization of events. If $bEnable = TRUE$ AND $eCtrlMode = eHVACCtlrMode_Auto$, 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 [409] and Table 2 [409]. If $eCtrlMode = eHVACCtlrMode_Manual$, then the function block represents a multiplexer. If $bEnable = TRUE$ AND $eCtrlMode = eHVACCtlrMode_Manual$, then the value of the output $iValueOut$ is controlled via $iManualValue$. If $iManualProfile = 5$, then $iValueOut = iValue5$, see Table 3 [410].

iManualProfile: If the operation mode $eCtrlMode = eHVACCtlrMode_Manual$ (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 [410].

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 $bDirection$. $bEnable = FALSE$ means that the events determine the output value $iValueOut$ in ascending form from 1 to 16. If $bEnable = TRUE$ AND $eCtrlMode = eHVACCtlrMode_Auto$ AND $bDirection = FALSE$ AND $bEvent15 = TRUE$ (highest occurring event in Table 1 [409] 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 $eCtrlMode = eHVACCtlrMode_Auto$ AND $bDirection = TRUE$ AND $bEvent1 = TRUE$ (lowest occurring event in Table 2 [409] column 1), then $iValueOut = iValue1$. The variables $bEvent1-16$ and $bDirection$ are taken into account only if $eCtrlMode = eHVACCtlrMode_Auto$.

bDirection: The function block can be used to prioritize events if $eCtrlMode = eHVACCtlrMode_Auto$. 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 $eCtrlMode = eHVACCtlrMode_Auto$ AND $bDirection = FALSE$ AND $bEvent15 = TRUE$ (highest occurring event in Table 1 [409] 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 $eCtrlMode = eHVACCtlrMode_Auto$ AND $bDirection = TRUE$ AND $bEvent1 = TRUE$ (lowest occurring event in Table 2 [409] column 1), then $iValueOut = iValue1$. The variables $bEvent1-16$ and $bDirection$ are taken into account only if $eCtrlMode = eHVACCtlrMode_Auto$.

VAR_INPUT

+ $iValueOut$: INT;
+ $bActive$: BOOL;
+ $bEdgeNewEvent$: BOOL;
+ $iValueNumber$: INT;

The function block can be used to prioritize events if $eCtrlMode = eHVACCtlrMode_Auto$. 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 $eCtrlMode = eHVACCtlrMode_Auto$ AND $bDirection = FALSE$ AND $bEvent15 = TRUE$ (highest occurring event in Table 1 [409] 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 $eCtrlMode = eHVACCtlrMode_Auto$ AND $bDirection = TRUE$ AND $bEvent1 = TRUE$ (lowest occurring event in Table 2 [409] column 1), then $iValueOut = iValue1$. The function block can be used as a multiplexer if $eCtrlMode = eHVACCtlrMode_Manual$. 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 $eCtrlMode = eHVACCtlrMode_Manual$ AND $iManualValue = 2$, then $iValueOut = iValue2$, see Table 3 [410].

bActive: $bActive$ goes TRUE if

1. $bEnable = TRUE$ AND $eCtrlMode = eHVACCtlrMode_Auto$ and one of the input variables $bEvent1-16$ is TRUE.
2. \( bEnable = \text{TRUE AND } eCtrlMode = eHVACCtrlMode\_Manual \) \& \( iManualValue > 0 \)

**bEdgeNewEvent:** Is TRUE for one PLC cycle if

1. \( bEnable = \text{TRUE AND } eCtrlMode = eHVACCtrlMode\_Auto \) and the decisive event (\( bEvent1-16 \)) for controlling the output \( iValueOut \) by means of \( iValue1-16 \) has changed.

2. \( bEnable = \text{TRUE AND } eCtrlMode = eHVACCtrlMode\_Manual \) and with each change of \( iManualValue \)

**iVarValueNumber:** The variable \( iVarValueNumber \) indicates the variable from which the value on \( iVarValueOut \) is output. If \( iVarValueOut = iVarValue7 \), then \( iVarValueNumber = 7 \), see Table 1 [409], Table 2 [409] and Table 3 [410]

**VAR\_IN\_OUT**

| iVarValue1-16 : INT; | X |

iVarValue1-16: The value of the output variable \( iVarValueOut \) is determined by the variables \( iVarValue1 \) to \( iVarValue16 \).

\( iVarValueOut = iVarValueX \)

If the function block is used to prioritize events, then each of the variables \( iVarValue1-16 \) is assigned to an event. \( iVarValue1 \) is assigned to the event \( bEvent1 \), \( iVarValue2 \) to the event \( bEvent2 \), \( iVarValue3 \) to the event \( bEvent3 \), ..., \( iVarValue16 \) to the event \( bEvent16 \)

If the function block is used as a multiplexer, then each of the variables \( iVarValue1-16 \) is assigned to the value of \( iManualValue \). If \( iManualValue = 1 \), then \( iVarValueOut = iVarValue1 \). If \( iManualValue = 2 \), then \( iVarValueOut = iVarValue2 \).... if \( iManualValue = 16 \), then \( iVarValueOut = iVarValue16 \).

The variable is saved persistently.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

**5.1.7.23 FB\_HVACPriority\_REAL\_8**

![FB_HVACPriority_REAL_8](image)
This function block can be used to prioritize events or as a multiplexer. This selection is made with the help of the Enum \textit{eCtrlMode}.

The function block can be used to prioritize events if \textit{eCtrlMode} = \textit{eHVACCtrlMode\_Auto}. The output \textit{rValueOut} is controlled via the events occurring at the inputs \textit{bEvent1-8}, \textit{rValue1-8} and the control direction of the prioritization \textit{bDirection}.

\textit{bDirection} = \text{FALSE} means that the events have a higher priority in ascending form from 1 to 8. If \textit{bEnable} = \text{TRUE} AND \textit{eCtrlMode} = \textit{eHVACCtrlMode\_Auto} AND \textit{bDirection} = \text{FALSE} AND \textit{bEvent7} = \text{TRUE} (highest occurring event in Table 1\[413\] column 1), then \textit{rValueOut} has the value of \textit{rValue7}.

\textit{bDirection} = \text{TRUE} means that the events have a higher priority in descending form from 8 to 1. If \textit{bEnable} = \text{TRUE} AND \textit{eCtrlMode} = \textit{eHVACCtrlMode\_Auto} AND \textit{bDirection} = \text{TRUE} AND \textit{bEvent1} = \text{TRUE} (lowest occurring event in Table 2\[413\] column 1), then \textit{rValueOut} has the value of \textit{rValue1}.

The function block can be used as a multiplexer if \textit{eCtrlMode} = \textit{eHVACCtrlMode\_Manual}. The value of \textit{iManualValue} refers to one of the VAR\_IN\_OUT variables \textit{rValue1-8}, whose value is output via \textit{rValueOut}. If \textit{bEnable} = \text{TRUE} AND \textit{eCtrlMode} = \textit{eHVACCtrlMode\_Manual} AND \textit{iManualValue} = 2, then \textit{rValueOut} = \textit{rValue2}, see Table 3\[414\].

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
\textit{bEnable} & \textit{eCtrlMode} & \textit{bDirection} & \textit{bEvent1} & \textit{bEvent2} & \textit{bEvent3} & \textit{bEvent4} \\
\hline
\text{TRUE} & \textit{eHVACCtrlMode\_Auto} & \text{FALSE} & \text{FALSE} & \text{FALSE} & \text{FALSE} & \text{FALSE} \\
\text{FALSE} & \text{FALSE} & \text{FALSE} & \text{FALSE} & \text{FALSE} & \text{FALSE} & \text{FALSE} \\
\hline
\end{tabular}
\end{table}

\textbf{Table 1: Prioritization of events in ascending form from 1 to 8}

It can be seen in Table 1 that \textit{bDirection} = \text{FALSE}. This means that the events determine the output value \textit{rValueOut} in ascending form from 1 to 8. If \textit{bEnable} = \text{TRUE} AND \textit{eCtrlMode} = \textit{eHVACCtrlMode\_Auto} AND \textit{bDirection} = \text{FALSE} AND \textit{bEvent7} = \text{TRUE} (highest occurring event in column 1), then \textit{rValueOut} = \textit{rValue7}.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
\textit{rValueNumber} & \textit{rValue1} & \textit{rValue2} & \textit{rValue3} & \textit{rValue4} \\
\hline
7 & 8 & 0 & 3 & 5 \\
\hline
\end{tabular}
\end{table}

\textbf{Table 2: Prioritization of events in descending form from 8 to 1}

It can be seen in Table 2 that \textit{bDirection} = \text{TRUE}. This means that the events determine the output value \textit{rValueOut} in descending form from 8 to 1. If \textit{bEnable} = \text{TRUE} AND \textit{eCtrlMode} = \textit{eHVACCtrlMode\_Auto} AND \textit{bDirection} = \text{TRUE} AND \textit{bEvent1} = \text{TRUE} (lowest occurring event in Table 2\[413\] column 1), then \textit{rValueOut} = \textit{rValue1}.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
\textit{rValueOut} & \textit{rValue7} & 0 & \textit{rValue6} & \textit{rValue5} \\
\hline
\textit{rValueOut} & \textit{rValue7} & \textit{rValue5} & \textit{rValue3} & \textit{rValue1} \\
\hline
\end{tabular}
\end{table}

\textbf{NOTE}

A frequently changing variable may not be applied to the VAR\_IN\_OUT variables \textit{rValue1-8} if \textit{eDataSecurityType} = \textit{eHVACDataSecurityType\_Persistent}. This would lead to premature wear of the storage medium of the controller. If the VAR\_IN\_OUT variables \textit{rValue1-8} change frequently and are not to be stored persistently, then \textit{eDataSecurityType} must be \textit{eDataSecurityType\_Idle}.
If \( bEnable = \text{TRUE} \) AND \( eCtrlMode = eHVACCtrlMode\_Manual \) AND \( iManualValue = 2 \), then \( rValueOut = 2 \).

### VAR_INPUT

- \( eDataSecurityType \) : \( \text{E\_HVACDataSecurityType} \);
- \( bEnable \) : \( \text{BOOL} \);
- \( bEvent1 - bEvent8 \) : \( \text{BOOL} \);
- \( eCtrlMode \) : \( \text{E\_HVACCtrlMode} \);
- \( iManualValue \) : \( \text{INT} \); \( 0..8 \)
- \( bDirection \) : \( \text{BOOL} \);

**eDataSecurityType**: If \( eDataSecurityType[\, 490] = \text{eHVACDataSecurityType\_Persistent} \), 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 \( \text{FB\_HVACPersistentDataHandling} \) must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000\_TC3\_HVAC/Resources.zip/5090272523.zip

If \( eDataSecurityType[\, 490] = \text{eHVACDataSecurityType\_Idle} \) the persistently declared variables are not saved in a fail-safe manner.

#### NOTE

A cyclically changing variable must never be linked with the IN\_OUT variable of a function block, if \( eDataSecurityType[\, 490] = \text{eHVACDataSecurityType\_Persistent} \). It would lead to early wear of the flash memory.

\( bEnable \): The function block is enabled via TRUE. If \( bEnable = \text{FALSE} \), then \( rValueOut \) is set constantly to 0.

\( eCtrlMode \): This enum specifies the operation mode of the function block (see \( \text{E\_HVACCtrlMode[\, 490]} \)). If \( eCtrlMode = \text{eHVACCtrlMode\_Auto} \), then the function block represents a prioritization of events. If \( bEnable = \text{TRUE} \) AND \( eCtrlMode = \text{eHVACCtrlMode\_Auto} \), 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[\, 413] and Table 2[\, 413].
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 [414]

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 [414]

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 [413] 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 [413] 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 [413] 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 [413] 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 [413] 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 [413] 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 [414].

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. \( b\text{Enable} = \text{TRUE AND } e\text{CtrlMode} = e\text{HVACCtrlMode}\_\text{Manual} \) and with each change of \( i\text{ManualValue} \)

**iValueNumber:** The variable \( i\text{ValueNumber} \) indicates the variable from which the value on \( r\text{ValueOut} \) is output. If \( r\text{ValueOut} = r\text{Value7} \), then \( i\text{ValueNumber} = 7 \), see Table 1, Table 2, and Table 3.

**VAR\_IN\_OUT**

\[
\begin{array}{ll}
\text{rValue1-8} & \text{: REAL;}
\end{array}
\]

**rValue1-8:** The value of the output variable \( r\text{ValueOut} \) is determined by the variables \( r\text{Value1} \) to \( r\text{Value8} \).

\( r\text{ValueOut} = r\text{ValueX} \)

If the function block is used to prioritize events, then each of the variables \( r\text{Value1-8} \) is assigned to an event. \( r\text{Value1} \) is assigned to the event \( b\text{Event1} \), \( r\text{Value2} \) to the event \( b\text{Event2} \), \( r\text{Value3} \) to the event \( b\text{Event3} \), ... and \( r\text{Value8} \) to the event \( b\text{Event8} \).

If the function block is used as a multiplexer, then each of the variables \( r\text{Value1-8} \) is assigned to the value of \( i\text{ManualValue} \). If \( i\text{ManualValue} = 1 \), then \( r\text{ValueOut} = r\text{Value1} \). If \( i\text{ManualValue} = 2 \), then \( r\text{ValueOut} = r\text{Value2} \)... If \( i\text{ManualValue} = 8 \), then \( r\text{ValueOut} = r\text{Value8} \).

The variable is saved persistently.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>


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 [p. 418] 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 [p. 418] 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 [419].

**NOTE**

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>
<thead>
<tr>
<th>Table 1: Prioritization of events in ascending form from 1 to 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>It can be seen in Table 1 that <code>bDirection = FALSE</code>. This means that the events determine the output value <code>rValueOut</code> in ascending form from 1 to 16. If <code>bEnable = TRUE</code> AND <code>eCtrlMode = eHVACCtrlMode_Auto</code> AND <code>bDirection = FALSE</code> AND <code>bEvent15 = TRUE</code> (highest occurring event in column 1), then <code>rValueOut = rValue15</code>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>TRUE</th>
<th>TRUE</th>
<th>FALSE</th>
<th>FALSE</th>
<th>FALSE</th>
<th>TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEnable</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>eCtrlMode</td>
<td>eHVACCtrlMode_Auto</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bDirection</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent1</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent2</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent3</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent4</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent5</td>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent6</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent7</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent8</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent9</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent10</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent11</td>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent12</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent13</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent14</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent15</td>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent16</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>rValueOut</td>
<td>rValue15</td>
<td>rValue14</td>
<td>rValue13</td>
<td>rValue12</td>
<td>rValue11</td>
<td>rValue10</td>
</tr>
<tr>
<td>rValueNumber</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2: Prioritization of events in descending form from 16 to 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>It can be seen in Table 2 that <code>bDirection = TRUE</code>. This means that the events determine the output value <code>rValueOut</code> in descending form from 16 to 1. If <code>bEnable = TRUE</code> AND <code>eCtrlMode = eHVACCtrlMode_Auto</code> AND <code>bDirection = TRUE</code> AND <code>bEvent1 = TRUE</code> (lowest occurring event in Table 2 [418] column 1), then <code>rValueOut = rValue1</code>.</td>
</tr>
</tbody>
</table>
If $\text{bEnable} = \text{TRUE}$ AND $\text{eCtrlMode} = \text{eHVACCtrlMode Manual}$ AND $\text{iManualValue} = 2$, then $\text{rValueOut} = 2$.

**Table 3: Multiplexer**

<table>
<thead>
<tr>
<th>bEnable</th>
<th>TRUE</th>
<th>TRUE</th>
<th>FALSE</th>
<th>FALSE</th>
<th>FALSE</th>
<th>TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEvent1</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent2</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent3</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent4</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent5</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent6</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent7</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent8</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent9</td>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent10</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent11</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent12</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent13</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent14</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent15</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bEvent16</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

| rValueOut | 1 | 1 | 9 | 3 | 0 | 1 |
| rValueNumber | 1 | 1 | 9 | 3 | 0 | 1 |

**VAR_INPUT**

- **eDataSecurityType**: If $\text{eDataSecurityType [490]} = \text{eHVACDataSecurityType Persistent}$, 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 $\text{FB_HVACPersistentDataHandling [479]}$ 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/zip/5090272523.zip

If $\text{eDataSecurityType [490]} = \text{eHVACDataSecurityType Idle}$ the persistently declared variables are not saved in a fail-safe manner.

**NOTE**

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if $\text{eDataSecurityType [490]} = \text{eHVACDataSecurityType Persistent}$. 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 [418] and Table 2 [418].
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 [419].

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 [419].

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.
 If bEnable = 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 [418] column 1), then rValueOut = rValue15.
If bEnable = 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 [418] 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.
If bEnable = 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 [418] column 1), then rValueOut = rValue15.
If bEnable = 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 [418] 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.
If bEnable = 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 [418] column 1), then rValueOut = rValue15.
If bEnable = 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 [418] 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 [419].

bActive: bActive goes TRUE if
1. \( b\text{Enable} = \text{TRUE AND } e\text{CtrlMode} = e\text{HVACCtlMode\_Auto} \) and one of the input variables \( b\text{Event1-16} \) is TRUE.

2. \( b\text{Enable} = \text{TRUE AND } e\text{CtrlMode} = e\text{HVACCtlMode\_Manual} \) AND \( i\text{ManualValue} > 0 \)

**bEdgeNewEvent:** Is TRUE for one PLC cycle if

1. \( b\text{Enable} = \text{TRUE AND } e\text{CtrlMode} = e\text{HVACCtlMode\_Auto} \) and the decisive event \( b\text{Event1-16} \) for controlling the output \( r\text{ValueOut} \) by means of \( r\text{Value1-16} \) has changed.

2. \( b\text{Enable} = \text{TRUE AND } e\text{CtrlMode} = e\text{HVACCtlMode\_Manual} \) and with each change of \( i\text{ManualValue} \)

**iVarValueNumber:** The variable \( iVarValueNumber \) indicates the variable from which the value on \( r\text{ValueOut} \) is output. If \( r\text{ValueOut} = r\text{Value7} \), then \( iVarValueNumber = 7 \), see Table 1 [418], Table 2 [418] and Table 3 [419]

### VAR_IN_OUT

\[
r\text{Value1-16} : \text{REAL};
\]

**rValue1-16:** The value of the output variable \( r\text{ValueOut} \) is determined by the variables \( r\text{Value1} \) to \( r\text{Value16} \).

\[
r\text{ValueOut} = r\text{ValueX}
\]

If the function block is used to prioritize events, then each of the variables \( r\text{Value1-16} \) is assigned to an event. \( r\text{Value1} \) is assigned to the event \( b\text{Event1} \), \( r\text{Value2} \) to the event \( b\text{Event2} \), \( r\text{Value3} \) to the event \( b\text{Event3} \), ... and \( r\text{Value16} \) to the event \( b\text{Event16} \)

If the function block is used as a multiplexer, then each of the variables \( r\text{Value1-16} \) is assigned to the value of \( i\text{ManualValue} \). If \( i\text{ManualValue} = 1 \), then \( r\text{ValueOut} = r\text{Value1} \). If \( i\text{ManualValue} = 2 \), then \( r\text{ValueOut} = r\text{Value2} \). ... If \( i\text{ManualValue} = 16 \), then \( r\text{ValueOut} = r\text{Value16} \).

The variable is saved persistently.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

### 5.1.7.25 FB\_HVACCtOptimizedOn

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 [441]`. The pre-start time can be determined approximately with these values. This temperature change function f(ΔT) typically adopts the following course:
Function values within these 10 points are defined by linear equations, while values outside correspond to the function value \( f(\text{AT}_1) \) or \( f(\text{AT}_{10}) \) respectively:

- \( \text{AT} < \text{AT}_1 \): \( f(\text{AT}) = f(\text{AT}_1) \)
- \( \text{AT} > \text{AT}_{10} \): \( f(\text{AT}) = f(\text{AT}_{10}) \)

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 \( \text{uiActualPrestartTime} \), which always contains the current pre-start time, is shown here in violet. The output \( \text{uiPrestartTime} \) 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 building 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 \( b_{Output} \), the prestart module gets informed by its input \( b_{SchedulerActive} \). Then the prestart module starts an internal countdown with the previously output prestart time \( \text{uiPrestartTime} \) [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 \( b_{PrestartActive} \) is set. It can be used e.g. for the quickstart of the boiler.

The output \( \text{uiPrestartTime} \) follows continuously 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 \( b_{WithAdaption} \), 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[427]."

**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[427]."

When adaption is requested, the input \( b_{WithAdaption} \) has to be set at the same time as the input \( b_{SchedulerActive} \). For \( b_{WithAdaption} \) a simple trigger impulse is sufficient. When the input \( b_{SchedulerActive} \) drops down, a started countdown is immediately aborted. The adaption can be disabled via the input \( b_{DisableAdaption} \). 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 above.
Init
flag „mit adaption“ := FALSE
flag „NO“ := FALSE
output uPrestartTime := flag „Aktuelle Vorstartzeit“
output bPrestartActive := FALSE
output udPrestartCountdown := 0

steigende Flanke
bSchedulerActive

TRUE

bWithAdaption
AND NOT bDisableAdaption

TRUE

flag „with adaption“ := TRUE

saving actual values:
- room-temperature-difference
- point of the temperature-change-function (OT), whose outdoor temperature is closest to the actual at the start of the countdown
- flag „within functional area”

output uPrestartTime =

FALSE
TRUE

flag „mit adaption“

Countdownzeit :=
output uPrestartTime

if the prestart-time is 0 and adaption is not preselected, then there is no need to continue.
Step Countdown (prestart-phase)

- If the calculated pre-stop time is 0 minutes, a test countdown of 1 minute will be started nevertheless, in order to evaluate the prestart time. During this countdown, bPreStartActive will NOT be set.

  - Output: uPStartTime = 0
  - Output: bPreStartActive := TRUE

Step adaption

- Calculation:
  - Heating mode selected: \( r\text{RoomSetpointValue} - r\text{RoomTemperature} \leq 0.1 \)
  - Cooling mode selected: \( r\text{RoomTemperature} - r\text{RoomSetpointValue} \leq 0.1 \)

- If flag "with adaption" AND flag "within functional area" AND NOT bDisableAdaption is FALSE, end of prestart-phase without adaption.

- TRUE:
  - 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:

\[
\text{f}(\Delta T)_{\text{NEW}} = \frac{\text{f}(\Delta T)_{\text{OLD}}}{100} \times (100 - r\text{AdaptionFactor}) + \frac{\Delta T}{\Delta t_{\text{calculated}}} \times r\text{AdaptionFactor}
\]

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` [490] = `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` [479] must be instanced once in the main program, which is called cyclically. Otherwise the instance 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/
zip/5090272523.zip

If eDataSecurityType[490] := eHVACDataSecurityType_Idle the persistently declared variables are not 
saved in a fail-safe manner.

NOTE
A cyclically changing variable must never be linked with the IN_OUT variable of a function block,
if eDataSecurityType[490] := 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 
completely 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 [315].
**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/zip/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 adaption factor (in percent) thereby represents the weighting.

\[
\hat{f}(AT)_{NEW} := \frac{f(AT)_{OLD} \cdot (100 - rAdaptionFactor) + (ΔT/Δt)_{calculated} \cdot rAdaptionFactor}{100}
\]

With an adaptation factor of 100% the newly calculated value is adopted fully, while with 0% the old value is retained.

rAdaptionTolerance: If the countdown has expired and the room temperature setpoint has been reached with adaptation activated, no adaptation is carried out because the pre-start time is precisely right. The value rRoomSetpointValue 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 [441].

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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 \( r_{\text{RequestedTempDeviation}} \).

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 temperature there is a characteristic temperature change $\Delta T/\Delta t$, which is illustrated here in a dotted line.

The function block FB_HVACOptimizedOff refers with the variable `stTempChangeFunction` 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`. 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 prestop 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 \textit{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 \cite{437}".

\begin{itemize}
  \item 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 \cite{437}".
\end{itemize}

If adaptation is desired, the inputs \textit{bWithAdaption} and \textit{bSchedulerOff} must be set simultaneously. A pure trigger pulse is sufficient for \textit{bWithAdaption}. A previously started countdown is interrupted immediately if the input \textit{bSchedulerOff} releases. The adaptation can be suppressed by the input \textit{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.
step Init/Start

For the Optimized-Off-function the initial room-temperature-difference is equal to the requested declining of the temperature, i.e., requested change 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 over OTn-OTq.

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 adaption is not preselected, then there is no need to continue.
The input \texttt{bEnable}, whose mode of operation is not shown here, factually deactivates the function block when FALSE. "0" is output as the pre-stop time \texttt{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 $r_{\text{Requested Temp Deviation}}$ 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 $r_{\text{Requested Deviation}}$ is exhausted up to a tolerance $r_{\text{Adaption Tolerance}}$ -> no adaptation takes place.
2. The countdown has not yet elapsed, but the requested deviation $r_{\text{Requested Deviation}}$ has been exceeded -> the pre-stop time $ui_{\text{Prestop Time}}$ was too long.
3. The countdown has elapsed, but the requested deviation $r_{\text{Requested Deviation}}$ is not yet fully exhausted -> the pre-stop time $ui_{\text{Prestop Time}}$ was too short, i.e. switch-off can take place even earlier.

The value $\Delta T/\Delta t$ can now be re-determined on the basis of the elapsed countdown time and the change in the room temperature difference that has taken place:

1. no change.
2. too much time was calculated – new value: "requested deviation $r_{\text{Requested Deviation}}$" / "time elapsed so far"
3. too little time was calculated – new value: ("requested deviation $r_{\text{Requested Deviation}}$" - "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 $\Delta T/\Delta t$ is not necessarily adopted 100% as the new value at the interpolation point. In fact there is an option to mix the new value from a weighting of old and calculated value. This weighting takes place with the aid of the so-called adaptation factor $r_{\text{Adaption Factor}}$:

$$f(AT)_{\text{NEW}} := \frac{f(AT)_{\text{OLD}} \cdot (100 - r_{\text{Adaption Factor}})}{100} + \frac{(\Delta T/\Delta t)_{\text{calculated}} \cdot r_{\text{Adaption Factor}}}{100}$$

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**

- \texttt{eDataSecurityType} : \texttt{E_HVACDataSecurityType};
- \texttt{eCtrlFct} : \texttt{E_BARCtrlFct};
- \texttt{bEnable} : \texttt{BOOL};
- \texttt{bSchedulerOff} : \texttt{BOOL};
- \texttt{bWithAdaption} : \texttt{BOOL};
- \texttt{bDisableAdaption} : \texttt{BOOL};
- \texttt{rOutsideTemperature} : \texttt{REAL};
- \texttt{rRoomTemperature} : \texttt{REAL};
- \texttt{rRequestedTempDeviation} : \texttt{REAL};

**eDataSecurityType:** If \texttt{eDataSecurityType} \texttt{[490]} = \texttt{eHVACDataSecurityType_Persistent}, the persistent \texttt{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 \texttt{FB_HVACPersistentDataHandling} \texttt{[479]} 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/zip/5090272523.zip

If eDataSecurityType[490] = eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

<table>
<thead>
<tr>
<th>NOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType[490] = eHVACDataSecurityType_Persistent. It would lead to early wear of the flash memory.</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uiPrestopTime</td>
<td>UINT</td>
</tr>
<tr>
<td>bPrestopActive</td>
<td>BOOL</td>
</tr>
<tr>
<td>udiCountdownTime</td>
<td>UDINT</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
</tr>
<tr>
<td>udiErrorID</td>
<td>UDINT</td>
</tr>
</tbody>
</table>

**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 [315].
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/zip/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 \( \Delta T/\Delta t \) that has taken place is calculated for the nearest outside temperature interpolation point. However, this is not necessarily adopted 100% as the new value. In fact there is an option to mix the new value from a weighting of old and calculated value. The adaptation factor (in percent) thereby represents the weighting.

\[
\frac{f(A T)_{\text{NEW}}}{f(A T)_{\text{OLD}}} = (100 - rAdaptionFactor) + \left( \frac{\Delta T/\Delta t_{\text{calculated}} \cdot rAdaptionFactor}{100} \right)
\]

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 \( r\text{RequestedDeviation} \) is exhausted, no adaptation is carried out because the pre-stop time is precisely right. The value \( rAdaptionTolerance \) defines an additional tolerance range:

**stTempChangeFunction**: Structure variable of the type ST_HVACTempChangeFunction 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.
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` and `FB_HVACOptimizedOff` 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 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**

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bWrite</td>
<td>BOOL</td>
</tr>
<tr>
<td>rOutsideTemp_1</td>
<td>REAL</td>
</tr>
<tr>
<td>rRoomTempChange_1</td>
<td>REAL</td>
</tr>
<tr>
<td>rOutsideTemp_2</td>
<td>REAL</td>
</tr>
<tr>
<td>rRoomTempChange_2</td>
<td>REAL</td>
</tr>
<tr>
<td>rOutsideTemp_3</td>
<td>REAL</td>
</tr>
<tr>
<td>rRoomTempChange_3</td>
<td>REAL</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
bWrite: A rising edge at this input copies the values entered at the inputs to the pre-start function.

\[(r\text{OutsideTemp}_1 - r\text{RoomTempChange}_1) \ldots (r\text{OutsideTemp}_{10} - r\text{RoomTempChange}_{10}):\] Value pairs of the pre-start function: Room temperature change \((r\text{RoomTempChange})\) in degrees Kelvin per minute at outside temperature \((r\text{OutsideTemp})\) 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 [315].

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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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 \texttt{eDataSecurityType} = \texttt{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 \texttt{FB\_HVACPersistentDataHandling} must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000\_TC3\_HVAC/Resources/zip/5090272523.zip

If \texttt{eDataSecurityType} = \texttt{eHVACDataSecurityType\_Idle} the persistently declared variables are not saved in a fail-safe manner.

\begin{center}
\begin{tabular}{|c|}
\hline
\textbf{NOTE} \\
\hline
A cyclically changing variable must never be linked with the IN\_OUT variable of a function block, if \texttt{eDataSecurityType} = \texttt{eHVACDataSecurityType\_Persistent}. It would lead to early wear of the flash memory. \\
\hline
\end{tabular}
\end{center}

\begin{description}
\item[bSetDefault:] If the variable is TRUE, the default values of the VAR\_IN\_OUT variables are adopted.
\item[bEnable:] The function block is enabled if \texttt{bEnable} = TRUE.
\item[rYin:] Analog input variables of the function block (0\%..100\%).
\item[bReset:] Acknowledge input in the event of a fault.
\end{description}

\textbf{VAR\_OUTPUT}

\begin{description}
\item[bOut]: PWM signal.
\item[rYOut]: Output of the input variable of the function block (0\%..100\%).
\item[bInvalidParameter]: Indicates that there is an incorrect input parameter. \texttt{bInvalidParameter} must be acknowledged with \texttt{bReset}.
\end{description}

\textbf{VAR\_IN\_OUT}

\begin{description}
\item[tPWMPeriod]: Period of the PWM signal. The variable is saved persistently. Preset to 30 min.
\item[tMINPowerOnTime]: Minimum switch-on time of the pulsed output \texttt{bOut}. The variable is saved persistently. Preset to 0 s.
\end{description}
Figure: Scope2 recording for additional explanation of the mode of operation of the function block. The output \((bOut = \text{TRUE})\) is set on enabling the function block \((bEnable = \text{TRUE})\). 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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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`.

<table>
<thead>
<tr>
<th>Function Block: <code>FB_HVACStartAirConditioning</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>eDataSecurityType</code></td>
<td>eModeSeqCtrl</td>
</tr>
<tr>
<td><code>bSetDefault</code></td>
<td>bPreRinse</td>
</tr>
<tr>
<td><code>StartSeqWinter</code></td>
<td>bDampersOpen</td>
</tr>
<tr>
<td><code>StartSeqSummer</code></td>
<td>bSupplyVentilation</td>
</tr>
<tr>
<td><code>bTimeScheduler</code></td>
<td>bExhaustVentilation</td>
</tr>
<tr>
<td><code>bNightCooling</code></td>
<td>bStartPIDControllers</td>
</tr>
<tr>
<td><code>bFreezeProtect</code></td>
<td>bEnableLimitCtrl</td>
</tr>
<tr>
<td><code>bOverHeatingProtect</code></td>
<td>bErrorGeneral</td>
</tr>
<tr>
<td><code>rOutsideTemp</code></td>
<td>bErrorNoHotWater</td>
</tr>
<tr>
<td><code>rReturnWaterTemp</code></td>
<td>bErrorDampers</td>
</tr>
<tr>
<td><code>rSupplyAirTemp</code></td>
<td>bError</td>
</tr>
<tr>
<td><code>rReturnAirTemp</code></td>
<td>bInvalidParameter</td>
</tr>
<tr>
<td><code>bDampersFeedbOpen</code></td>
<td></td>
</tr>
<tr>
<td><code>bEmergencyStop</code></td>
<td></td>
</tr>
<tr>
<td><code>bReset</code></td>
<td></td>
</tr>
<tr>
<td><code>bConvective</code></td>
<td></td>
</tr>
<tr>
<td><code>DelaySupplyVent</code></td>
<td></td>
</tr>
<tr>
<td><code>tDelaySupplyVent</code></td>
<td></td>
</tr>
<tr>
<td><code>tDelayExhaustVent</code></td>
<td></td>
</tr>
<tr>
<td><code>tDifferenceTemp</code></td>
<td></td>
</tr>
<tr>
<td><code>tDelayTempCtrl</code></td>
<td></td>
</tr>
<tr>
<td><code>tReturnLimitCtrl</code></td>
<td></td>
</tr>
<tr>
<td><code>tPreRinseOutsideTemp</code></td>
<td></td>
</tr>
<tr>
<td><code>tPreRinseLimitTemp</code></td>
<td></td>
</tr>
<tr>
<td><code>tPreRinseMaxTime</code></td>
<td></td>
</tr>
<tr>
<td><code>tCurrentSequence</code></td>
<td></td>
</tr>
</tbody>
</table>
In systems with a mixed air chamber it makes sense to enable the controller only after the supply air temperature \( r_{\text{SupplyAirTemp}} \) has reached approximately the value of the exhaust air temperature. The maximum deviation is specified by the parameter \( r_{\text{DifferenceTemp}} \). 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 \( t_{\text{DelayLimitCtrl}} \) has elapsed, the output \( b_{\text{EnableLimitCtrl}} \) is enabled and, hence, the limit value monitoring of the analog inputs is activated.

If the outside temperature lies below the value of \( r_{\text{PreRinseOutsideTemp}} \), the value of \( i_{\text{CurrentSequence}} \) is applied to the output the same as that of \( i_{\text{StartSeqWinter}} \). The number of the pre-heater controller must be entered here (see the chapter on sequence controllers). If the outside temperature is higher than \( r_{\text{PreRinseOutsideTemp}} \), 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 \( i_{\text{StartSeqSummer}} \).

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 \( e_{\text{ModeSeqCtrl}} \) at the output of the function block is set as follows:

```plaintext
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 \( b_{\text{TimeScheduler}}, b_{\text{NightCooling}}, b_{\text{FreezeProtect}} \) and \( b_{\text{OverHeatingProtec}} \). Requests from retention functions are only applied if there is no request from a timer program at the input variable \( b_{\text{TimeScheduler}} \).

The programs for summer night cooling (\( b_{\text{NightCooling}} \)), cooling protection (\( b_{\text{FreezeProtect}} \)) and overheating protection (\( b_{\text{OverHeatingProtec}} \)) 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 \( E_{\text{HVACConvectionMode}} \). This output must be applied to the input variable \( e_{\text{Convection}} \).

A collection of all air conditioning system error messages that lead to the system switching off is applied to the input \( b_{\text{EmergencyStop}} \).

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 \( i_{\text{CurrentSequence}} \) to 0)
- Switch off the fans
- Close the dampers after the timer \( t_{\text{DelaySupplyVent}} \) has elapsed.

VAR_INPUT

- \( e_{\text{DataSecurityType}} \) : E_HVACDataSecurityType;
- \( b_{\text{SetDefault}} \) : BOOL;
- \( i_{\text{StartSeqWinter}} \) : INT;
- \( i_{\text{StartSeqSummer}} \) : INT;
- \( b_{\text{TimeScheduler}} \) : BOOL;
- \( b_{\text{NightCooling}} \) : BOOL;
- \( b_{\text{FreezeProtect}} \) : BOOL;
- \( b_{\text{OverHeatingProtec}} \) : BOOL;
- \( r_{\text{OutsideTemp}} \) : REAL;
- \( r_{\text{ReturnWaterTemp}} \) : REAL;
- \( r_{\text{SupplyAirTemp}} \) : REAL;
VAR_INPUT

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

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

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

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

iStartSeqWinter: The number of the sequence controller for the 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;

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>eConvection</td>
<td>E_HVACConvectionMode;</td>
</tr>
<tr>
<td>tDelaySupplyVent</td>
<td>TIME;</td>
</tr>
<tr>
<td>tDelayExhaustVent</td>
<td>TIME;</td>
</tr>
<tr>
<td>tDelayTempCtrl</td>
<td>TIME;</td>
</tr>
<tr>
<td>rDifferenceTemp</td>
<td>REAL;</td>
</tr>
<tr>
<td>tDelayLimitCtrl</td>
<td>TIME;</td>
</tr>
<tr>
<td>rPreRinseOutsideTemp</td>
<td>REAL;</td>
</tr>
<tr>
<td>rPreRinseLimitTemp</td>
<td>REAL;</td>
</tr>
<tr>
<td>tPreRinseMaxTime</td>
<td>TIME;</td>
</tr>
<tr>
<td>iCurrentSequence</td>
<td>INT;</td>
</tr>
</tbody>
</table>

eConvection: ENUM via which summer night cooling can take place as forced ventilation with fans or as convection ventilation without fans (see E_HVACConvectionMode [490]). 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 pre-rinsing 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 writable again for the switching on or off of the sequence controllers. The variable is saved persistently.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCat 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDataSecurityType</td>
<td>E_HVACDataSecurityType;</td>
</tr>
<tr>
<td>bSetDefault</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bEnable</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>
eDataSecurityType: If eDataSecurityType \[490]\[490]= 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 \[479]\[479] 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/zip/5090272523.zip

If eDataSecurityType \[490]\[490]= eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

NOTE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType \[490]\[490]= 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;
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 ($rMinimumOutsideTemp + 0.2$K), 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

#### 5.1.7.31 FB_HVACSummerNightCoolingEx

**FB_HVACSummerNightCoolingEx**

- eDateSecurityType
- bSetDefault
- bEnable
- rOutsideTemp
- rRoomTemp
- rSetpointRoomTemp
- bReset
- rMinimumOutsideTemp
- rDifferOutsideTempRoomTempOn
- rDifferOutsideTempRoomTempOff
- rDifferRoomTempSetpointRoomTempOn
- udiSecMaxRuntime
- bOn
- bStateDifferOutsideTempMinOutsideTemp
- bStateDifferRoomTempOutsideTemp
- rStateDifferOutsideTempMinOutsideTemp
- rStateDifferRoomTempOutsideTemp
- 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 least one PLC cycle. The function block and the summer night cooling function can be enabled via a timer.*
Program flowchart

VAR_INPUT

\[
\begin{align*}
eDataSecurityType & : \text{E_HVACDataSecurityType;} \\
bSetDefault & : \text{BOOL;} \\
bEnable & : \text{BOOL;} \\
rOutsideTemp & : \text{REAL;} \\
rRoomTemp & : \text{REAL;} \\
rSetpointRoomTemp & : \text{REAL;} \\
bReset & : \text{BOOL;} \\
\end{align*}
\]

**eDataSecurityType**  
If \( eDataSecurityType [490] := \text{E_HVACDataSecurityType\_Persistent}, \) 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 [479] 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/zip/5090272523.zip

If \( eDataSecurityType [490] := \text{E_HVACDataSecurityType\_Idle} \) the persistently declared variables are not saved in a fail-safe manner.
NOTE

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

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

bEnable: The function block is enabled by the PLC program with the input variable bEnable. 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: 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) <= rDifferOutsideTempRoomTempOn.

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

bInvalidParameter: Acknowledge input in the event of a fault.
**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 least one PLC cycle.

**bInvalidParameter:** Indicates that there is an incorrect input parameter. **bInvalidParameter** must be acknowledged with **bReset**.

**VAR_IN_OUT**

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>rMinimumOutsideTemp</td>
<td>REAL</td>
</tr>
<tr>
<td>rDifferOutsideTempRoomTempOn</td>
<td>REAL</td>
</tr>
<tr>
<td>rDifferOutsideTempRoomTempOff</td>
<td>REAL</td>
</tr>
<tr>
<td>rDifferRoomTempSetpointRoomTempOn</td>
<td>REAL</td>
</tr>
<tr>
<td>udiSecMaxRuntime</td>
<td>UDINT</td>
</tr>
</tbody>
</table>

**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 **rDifferOutsideTempRoomTempOn** so that one of the conditions for activating the summer night cooling **bOn** = TRUE is satisfied (0..10).

```plaintext
((rRoomTemp - rOutsideTemp) > rDifferOutsideTempRoomTempOn)
```

**rDifferOutsideTempRoomTempOn** must be 0.4 K higher than **rDifferOutsideTempRoomTempOff**. Furthermore **rDifferOutsideTempRoomTempOn** 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).

```plaintext
((rRoomTemp - rOutsideTemp) < rDifferOutsideTempRoomTempOff)
```

**rDifferOutsideTempRoomTempOff** must be 0.4 K lower than **rDifferOutsideTempRoomTempOn**. In addition, **rDifferOutsideTempRoomTempOff** 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).

```plaintext
((rRoomTemp - rSetpointRoomTemp) > rDifferRoomTempSetpointTemp)
```

**rDifferRoomTempSetpointTemp** 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.
Programming

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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**

- \( udiSec \): 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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**

- \( udiSec \): Variable that displays the seconds (0..4294967)
- \( tTime \): Variable of the type Time.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
5.1.7.34 FB_HVACTimeConSecMs

This function block converts a TIME variable \( t\text{Time} \) into two UDINT variables, \( udi\text{Sec} \) and \( udi\text{Ms} \), 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**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( udi\text{Ms} )</td>
<td>UDINT</td>
</tr>
<tr>
<td>( udi\text{Sec} )</td>
<td>UDINT</td>
</tr>
<tr>
<td>( t\text{Time} )</td>
<td>TIME</td>
</tr>
</tbody>
</table>

- **udiMs**: Variable that displays the milliseconds (0...999).
- **udiSec**: Variable that displays the seconds (0..4294966).
- **tTime**: Variable of the type Time.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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 \( b\text{On} \) 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 \( b\text{SysTakt1Hz} \) 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**

- \( eDataSecurityType : E\_HVACDataSecurityType; \)
- \( bSetDefault : BOOL; \)
- \( bOn : BOOL; \)
- \( bResetHours : BOOL; \)
- \( bResetCount : BOOL; \)
- \( dtSystemtime : DT; \)
- \( bStartWarnLevel : BOOL; \)
- \( bStartAlarmLevel : BOOL; \)
- \( bSysTakt1Hz : BOOL; \)
- \( bReset : BOOL; \)

**NOTE**

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.

**NOTE**

If \( eDataSecurityType \) := \( eHVACDataSecurityType\_Idle \) the persistently declared variables are not saved in a fail-safe manner.

**NOTE**

A cyclically changing variable must never be linked with the IN\_OUT variable of a function block, if \( eDataSecurityType \) := \( eHVACDataSecurityType\_Persistent \). It would lead to early wear of the flash memory.

**bSetDefault:** If the variable is TRUE, the default values of the VAR\_IN\_OUT variables are adopted.

**bOn:** 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

5.1.8 HVAC Time schedule

5.1.8.1 FB_HVACScheduler1ch

<table>
<thead>
<tr>
<th>FB_HVACScheduler1ch</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDataSecurityType</td>
</tr>
<tr>
<td>bOutput</td>
</tr>
<tr>
<td>bSetDefault</td>
</tr>
<tr>
<td>bEnable</td>
</tr>
<tr>
<td>bEnableOff</td>
</tr>
<tr>
<td>tSystemTime</td>
</tr>
<tr>
<td>tNextOn</td>
</tr>
<tr>
<td>tNextOff</td>
</tr>
<tr>
<td>bReset</td>
</tr>
<tr>
<td>bInvalidParameter</td>
</tr>
</tbody>
</table>
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 $t_{on_{-}ss}$ and $t_{off_{-}ss}$ 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 [490] = 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 [479] 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/zip/5090272523.zip

If eDataSecurityType [490] = eHVACDataSecurityType_Idle the persistently declared variables are not saved in a fail-safe manner.

NOTE

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if eDataSecurityType [490] = 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; uiOffsetBefor has the value 150 min. The switch-on time is brought forward by 150 minutes to 11:00.

**VAR_OUTPUT**

- **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**: With the variable stTimeChannel it is possible to generate one switch-on time per day of the week (see ST_HVACTimeChannel[498]).

**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.
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:

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:**

```plaintext
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` and `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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

**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 aTimeChannel remain active.

VAR_INPUT

<table>
<thead>
<tr>
<th>DATA TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDataSecurityType</td>
<td><code>E_HVACDataSecurityType</code></td>
</tr>
<tr>
<td>bSetDefault</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEnable</td>
<td>BOOL</td>
</tr>
<tr>
<td>dtSystemTime</td>
<td>DT</td>
</tr>
<tr>
<td>uiOffsetBefore</td>
<td>UINT</td>
</tr>
<tr>
<td>bReset</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

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

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

Application example: [https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip](https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip)

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

**NOTE**

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

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

**bEnable**: The function block is enabled by the 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 [498]). 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.

Enable | Reset after On | all Days | Mo | Di | Mi | Do | Fr | Sa | So | OnTime hh:mm:ss | OffTime hh:mm:ss

This setting produces:
An array of the type `ST_HVACTimeChannel` (see `ST_HVACTimeChannel`) 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

```plaintext
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

<table>
<thead>
<tr>
<th>Enable</th>
<th>Reset after</th>
<th>Mo</th>
<th>Di</th>
<th>Mi</th>
<th>Do</th>
<th>Fr</th>
<th>Sa</th>
<th>Su</th>
<th>OnTime hh:mm:ss</th>
<th>OffTime hh:mm:ss</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7:20</td>
<td>17:30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16:40</td>
<td>00:00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>00:00</td>
<td>00:00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15:00</td>
<td>03:00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12:00</td>
<td>18:00</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bGoChannelUp</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bGoChannelDown</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>

- **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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>arr7TimeChannel</td>
<td>ARRAY[1..7] OF ST_HVACTimeChannel;</td>
</tr>
<tr>
<td>stCurrChannel</td>
<td>ST_HVACTimeChannel;</td>
</tr>
<tr>
<td>iCurrChannelNumber</td>
<td>INT;</td>
</tr>
<tr>
<td>iNumOfChannelsToUse</td>
<td>INT;</td>
</tr>
</tbody>
</table>

- **arr7TimeChannel**: Array of the data sets of the timer `FB_HVACScheduler7ch` (see `ST_HVACTimeChannel` [498]).
- **stCurrChannel**: Contains the contents of the data set from the array `arr7TimeChannel`, which is selected via the variable `iCurrChannelNumber` (see `ST_HVACTimeChannel` [498]).
- **iCurrChannelNumber**: Selection index for the array `arr7TimeChannel`.
- **iNumOfChannelsToUse**: Maximum (7) permissible index for `iCurrChannelNumber` in case not all channels are needed.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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 \( t\text{On}_{ss} \) and \( t\text{Off}_{ss} \) of the respective time channel.

If the input variable \( b\text{Enable} = \text{FALSE} \), then the output \( b\text{Output} = \text{FALSE} \). The channels arr\( r\text{TimeChannel} \) remain active.

**VAR_INPUT**

- \( e\text{DataSecurityType} \) : E_HVACDataSecurityType;
- \( b\text{SetDefault} \) : BOOL;
- \( b\text{Enable} \) : BOOL;
- \( dt\text{SystemTime} \) : DT;
- \( ui\text{OffsetBefore} \) : UINT;
- \( b\text{Reset} \) : BOOL;

\( e\text{DataSecurityType} \): If \( e\text{DataSecurityType} \) := \( e\text{HVACDataSecurityType\_Persistent} \), the persistent VAR_IN_OUT variables of the function block are stored in the flash of the computer if a value changes. For this to work, the function block \( FB\_HVACPersistentDataHandling \) must be instanced once in the main program, which is called cyclically. Otherwise the instanced FB is not released internally.

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

Application example: [https://infosys.beckhoff.com/content/1033/TF8000\_TC3\_HVAC/Resources/zip/5090272523.zip](https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip)

If \( e\text{DataSecurityType} \) := \( e\text{HVACDataSecurityType\_Idle} \) the persistently declared variables are not saved in a fail-safe manner.

**NOTE**

A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if \( e\text{DataSecurityType} \) := \( e\text{HVACDataSecurityType\_Persistent} \). It would lead to early wear of the flash memory.

\( b\text{SetDefault} \): If the variable is TRUE, the default values of the VAR_IN_OUT variables are adopted.

\( b\text{Enable} \): The function block is enabled by the variable \( b\text{Enable} \). If \( b\text{Enable} = \text{FALSE} \) there is no switching at the output \( b\text{Output} \).

\( dt\text{SystemTime} \): The variable \( dt\text{SystemTime} \) transfers the computer system time to the timer program.

\( ui\text{OffsetBefore} \): 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**

```plaintext
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**

```plaintext
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` [498]). 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.

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**

```plaintext
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

<table>
<thead>
<tr>
<th>Enable</th>
<th>Reset after On</th>
</tr>
</thead>
<tbody>
<tr>
<td>all Days</td>
<td>Mo</td>
</tr>
<tr>
<td></td>
<td>07:20</td>
</tr>
<tr>
<td></td>
<td>16:40</td>
</tr>
<tr>
<td></td>
<td>00:00</td>
</tr>
<tr>
<td></td>
<td>15:00</td>
</tr>
<tr>
<td></td>
<td>12:00</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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 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[498]).

iCurrChannelNumber: Selection index for the array arr28TimeChannel.

iNumOfChannelsToUse: Maximum permissible index for iCurrChannelNumber, if not all channels are used.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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[473], 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;

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[479] 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/zip/5090272523.zip

If `eDataSecurityType [490] := eHVACDataSecurityType_Idle` the persistently declared variables are not saved in a fail-safe manner.

**NOTE**
A cyclically changing variable must never be linked with the IN_OUT variable of a function block, if `eDataSecurityType [490] := 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**

```plaintext
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**

```plaintext
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

<table>
<thead>
<tr>
<th>Reset after</th>
<th>ON Day Month</th>
<th>OFF Day Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>24 12</td>
<td>26 12</td>
</tr>
<tr>
<td></td>
<td>3 10</td>
<td>3 10</td>
</tr>
<tr>
<td></td>
<td>18 7</td>
<td>19 7</td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td>1 1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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

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

Application example: https://infosys.beckhoff.com/content/1033/TF8000_TC3_HVAC/Resources/zip/5090272523.zip

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

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

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

**bEnable:** The function block is enabled by the 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`).

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:

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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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 \textit{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 \textit{NT\_GetTime} and \textit{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 \textit{sNETID} and \textit{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**

- \texttt{bEnable} : BOOL;
- \texttt{sNETID} : T_AmsNetId;
- \texttt{tGetSystemTime} : TIME;
- \texttt{bGetSystemTime} : BOOL;

\textbf{bEnable}: Enables the function block. If \textit{bEnable} = TRUE, then the RTC clock is initialized with the NT system time.

\textbf{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.

\textbf{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 \textit{tGetSystemTime}, the RTC clock can be synchronized via the input variable \textit{bGetSystemTime}.

\textbf{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 \textit{tGetSystemTime}.

**VAR\_OUTPUT**

- \texttt{bReadyGetSystemTime} : BOOL;
- \texttt{bReadyRTC} : BOOL;
- \texttt{stSystemTime} : TIMESTRUCT;
- \texttt{dtSystemTime} : DT;
- \texttt{uiYear} : UINT;

476 Version: 1.1 TF8000
Programming

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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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:

```plaintext
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.

`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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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!!
Programming

### VAR_INPUT

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNETID</td>
<td>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.</td>
</tr>
<tr>
<td>TMOUT</td>
<td>States the length of the timeout that may not be exceeded by execution of the ADS command.</td>
</tr>
<tr>
<td>ePersistentMode</td>
<td>Mode in which the persistent data should be written. Also see E_PersistentMode (see Tc2_Utilities library).</td>
</tr>
</tbody>
</table>

### VAR_OUTPUT

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>srcNETID</th>
<th>T_AmsNetId;</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcFileName</td>
<td>T_MaxString;</td>
</tr>
<tr>
<td>destNETID</td>
<td>T_AmsNetId;</td>
</tr>
<tr>
<td>destFileName</td>
<td>T_MaxString;</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL;</td>
</tr>
<tr>
<td>tAdsTimeOut</td>
<td>TIME;</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>bDone</th>
<th>BOOL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL;</td>
</tr>
<tr>
<td>udiErrorID</td>
<td>UDINT;</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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

![Beckhoff CX Configuration Tool](image)
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.

$stSystemtime$: 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.

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: $bInvalidParameter = TRUE$ 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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 \( t_{TaskCycleTime} \) is output on the output variable \( t_{CycleTime} \).

The \( t_{TaskCycleTime} \) 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

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.
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**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bSetDefault</td>
<td>If the variable is TRUE, the value of the input variable bVar_Default is adopted.</td>
</tr>
<tr>
<td>bVar_Default</td>
<td>Default value defined by the user. The variable is of the data type BOOL.</td>
</tr>
</tbody>
</table>

**VAR_IN_OUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bVar</td>
<td>Variable that is programmed fail-safe by the user.</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

5.1.10.2  FB_HVACPersistent_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_HVACPersistent_XX.

By way of example the function block FB_HVACPersistent_Bool is described here.
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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

### 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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

#### 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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>lrIN</td>
<td>LREAL;</td>
</tr>
<tr>
<td>iPrecision</td>
<td>INT;</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required library</th>
<th>required function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1 build 4022.16 or higher</td>
<td>Tc2_HVAC V3.3.1.0</td>
<td>TF8000</td>
</tr>
</tbody>
</table>

### 5.2 DUTs

#### 5.2.1 E_HVAC2PointActuatorMode

**TYPE** E_HVAC2PointActuatorMode:

```plaintext
{  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**: Building Management System; remote control of the actuator from the management level.

**OP**: Operator Panel; local control of the actuator from a local control panel.

#### 5.2.2 E_HVAC2PointCtrlMode

**TYPE** E_HVAC2PointCtrlMode:

```plaintext
{  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**: Building Management System; remote control of the actuator from a BMS.

**OP**: Operator Panel; local operation of the actuator from an OP.

#### 5.2.3 E_HVAC3PointActuatorMode

**TYPE** E_HVAC3PointActuatorMode:

```plaintext
{  eHVAC3PointActuatorMode_Auto_BMS := 0,
  eHVAC3PointActuatorMode_Open_BMS := 1,
  eHVAC3PointActuatorMode_Close_BMS := 2,
};
END_TYPE
```
5.2.4 E_HVACActuatorMode

```typescript
TYPE E_HVACActuatorMode :
   { 
      eHVACActuatorMode_Auto_BMS := 0,
      eHVACActuatorMode_Off_BMS := 1,
      eHVACActuatorMode_Speed1_BMS := 2,
      eHVACActuatorMode_Speed2_BMS := 3,
      eHVACActuatorMode_Speed3_BMS := 4,
      eHVACActuatorMode_Speed4_BMS := 5,
      eHVACActuatorMode_Auto_OP := 6,
      eHVACActuatorMode_Off_OP := 7,
      eHVACActuatorMode_Speed1_OP := 8,
      eHVACActuatorMode_Speed2_OP := 9,
      eHVACActuatorMode_Speed3_OP := 10,
      eHVACActuatorMode_Speed4_OP := 11
   }
END_TYPE
```

.. _BMS: Building Management System; remote control of the actuator from an BMS.

.. _OP: Operator Panel; local operation of the actuator from an OP.

5.2.5 E_HVACAirConditioning2SpeedMode

```typescript
TYPE E_HVACAirConditioning2SpeedMode:
   { 
      eHVACAirConditioning2SpeedMode_Off := 0,
      eHVACAirConditioning2SpeedMode_Heating := 1,
      eHVACAirConditioning2SpeedMode_Cooling := 2,
      eHVACAirConditioning2SpeedMode_HeatingAndCooling := 3
   }
END_TYPE
```

5.2.6 E_HVACAnalogOutputMode

```typescript
TYPE E_HVACAnalogOutputMode :
   { 
      eHVACAnalogOutputMode_Auto_BMS := 0,
      eHVACAnalogOutputMode_Manual_BMS := 1,
      eHVACAnalogOutputMode_Auto_OP := 2,
      eHVACAnalogOutputMode_Manual_OP := 3
   }
END_TYPE
```

.. _BMS: Building Management System; remote control of the actuator from an BMS.

.. _OP: Operator Panel; local operation of the actuator from an OP.

5.2.7 E_HVACAntiBlockingMode

```typescript
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, (*Ohm1_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, Ohm1_10000*)
eHVACBusTerminal_KL3201_0028 := 18, (*not supported, high resolution 0,01°C*)
eHVACBusTerminal_KL3201_0029 := 19, (*Ni1000_LS, Landiss&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, (*Ohm1_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, Ohm1_10000*)
eHVACBusTerminal_KL3202_0028 := 39, (*not supported, high resolution 0,01°C*)
eHVACBusTerminal_KL3202_0029 := 40, (*Ni1000_LS, Landiss&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, (*Ohm1_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, Ohm1_10000*)
eHVACBusTerminal_KL3204_0028 := 60, (*not supported, high resolution 0,01°C*)
eHVACBusTerminal_KL3204_0029 := 61, (*Ni1000_LS, Landiss&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*)
*/
}
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_rTi := 6,
  eHVACErrorCodes_InvalidParam_rTv := 7,
  eHVACErrorCodes_InvalidParam_rTd := 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_tNoFeedbackActuator := 26,
  eHVACErrorCodes_InvalidParam_iNumberOfSequences := 27,
  eHVACErrorCodes_InvalidParam_iMyNumberOfSequences := 28,
  eHVACErrorCodes_InvalidParam_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_HVACPwerMeasurementMode

TYPE E_HVACPwerMeasurementMode:
{
    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:
{
    eHVACRegOutsideTemp_NoRequest := 0,
    eHVACRegOutsideTemp_OTLowerLimit  := 1,
    eHVACRegOutsideTemp_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:
{
    eHVACRegValve_NoRequest := 0,
    eHVACRegValve_OrValvePosHigherLimit := 1,
    eHVACRegValve_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
}
```

### 5.2.22 E_HVACSequenceCtrlMode

The controller is enabled and controlled within the sequence via this ENUM, depending on the system operating modes.

```plaintext
eHVACSequenceCtrlMode_Stop := 0,
eHVACSequenceCtrlMode_On := 1,
eHVACSequenceCtrlMode_NightCooling := 2,
eHVACSequenceCtrlMode_FreezeProtection := 3,
eHVACSequenceCtrlMode_OverheatingProtection := 4,
eHVACSequenceCtrlMode_NightCoolingAndOverheatingProtection := 5
)
```

#### 5.2.23 E_HVACSetpointHeatingMode

```plaintext
eHVACSetpointHeatingMode_Off := 0,
eHVACSetpointHeatingMode_OnOutsideTemp := 1,
eHVACSetpointHeatingMode_OnDate := 2,
eHVACSetpointHeatingMode_OnPermanent := 3,
eHVACSetpointHeatingMode_OnFreezeProtec := 4,
eHVACSetpointHeatingMode_OnNight := 5,
eHVACSetpointHeatingMode_OnDay := 6
)
```

**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 \( r_{OutsideTempDamped} \) is smaller than the value of \( r_{HeatingLimit} \). The feed temperature setpoint calculated by the heating characteristic curve or the room setpoint module is routed through to the output \( i_{Setpoint} \). The output \( b_{Heating} \) 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 \( t_{DelayHeatingOn} \) and \( t_{DelayHeatingOff} \).

**eHVACSetpointHeatingMode_OnDate:** Heating period according to date. The heating circuit is enabled from the date \( i_{On\_Day} / i_{On\_Month} \) until \( i_{Off\_Day} / i_{Off\_Month} \). 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_NTC1k3_A1 := 3, (*NTC1k, 3K3 A1, negative, S+S*)
    eHVACTemperatureCurve_NTC2k2_A1 := 5, (*NTC2.2k, 2.2K2 A1, negative, S+S*)
    eHVACTemperatureCurve_NTC3k3_A1 := 6, (*NTC3.3k, 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;
END_STRUCT
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.
5.2.32 ST_HVACHoliday

```c
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

```c
TYPE ST_HVACI_Ctrl :
STRUCT
  rIntegralHigh : REAL;
  rIntegralLow : REAL;
  udiSecDelayHigh : UDINT;
  udiSecDelayLow : UDINT;
END_STRUCT
END_TYPE
```

- **rIntegralHigh**: Positive value for the upper limit at which the integration of the I-transfer element is stopped.
- **rIntegralLow**: Positive value for the lower limit at which the integration of the I-transfer element is stopped.
- **udiSecDelayHigh**: Delay time after whose expiry the I-transfer element is activated.
- **udiSecDelayLow**: Delay time after whose expiry the I-transfer element is activated.

5.2.34 ST_HVACPeriod

```c
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

```c
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

```plaintext
TYPE ST_HVACPowerMeasurement :
STRUCT
  diIL1, diIL2, diIL3           : DINT;
  diUL1, diUL2, diUL3           : DINT;
  diPL1, diPL2, diPL3           : DINT;
  diPg                           : DINT;
  dCosPhil1, dCosPhil2, dCosPhil3 : DINT;
  dCosPhi                        : DINT;
  dX1L, dX1L2, dX1L3            : DINT;
  dX2L, dX2L2, dX2L3            : DINT;
  dX3L, dX3L2, dX3L3            : DINT;
  dXg                            : DINT;
  dummy                          : DINT;
END_STRUCT
END_TYPE
```

### 5.2.37 ST_HVACPowerMeasurementEx

```plaintext
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;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
  fMaxL1, fMaxL2, fMaxL3         : LREAL;
END_STRUCT
END_TYPE
```

### 5.2.38 ST_HVACPowerRangeTable

```plaintext
TYPE ST_HVACPowerRangeTable :
STRUCT
  iAggregate        : INT;
  iAggregateStep    : INT;
  rY_Max            : REAL;
  rY_Min            : REAL;
  rIntegralHigh     : REAL;
  rIntegralLow      : REAL;
END_STRUCT
END_TYPE
```
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

```plaintext
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

Structure of the node points for the temperature change function in the modules FB_HVACOptimizedOn [421] and FB_HVACOptimizedOff [431].

\[
\text{TYPE} \quad \text{ST_HVACPrestartFunction} : \\
\quad \text{STRUCT} \\
\quad \quad \text{rOutsideTemp} : \text{ARRAY}[1..10] \text{ OF REAL; \{*[degC]*\}} \\
\quad \quad \text{rRoomTempChange} : \text{ARRAY}[1..10] \text{ OF REAL; \{*[degK/min]*\}} \\
\quad \text{END_STRUCT} \\
\text{END_TYPE}
\]

\text{rOutsideTemp} / \text{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

\[
\text{TYPE} \quad \text{E_BARCtrlFct} : \\
\quad \{ \\
\quad \quad \text{eBARCtrlFct_Off} := 0, \\
\quad \quad \text{eBARCtrlFct_Heating} := 1, \\
\quad \quad \text{eBARCtrlFct_Cooling} := 2 \\
\quad \}; \\
\text{END_TYPE}
\]

\text{eBARCtrlFct_Off}: Function selection OFF active.

\text{eBARCtrlFct_Heating}: Function selection for heating operation active.

\text{eBARCtrlFct_Cooling}: Function selection for cooling operation active.

5.2.41.2  E_BAREnergyLevel

\[
\text{TYPE} \quad \text{E_BAREnergyLevel} : \\
\quad \{ \\
\quad \quad \text{eBAREnergyLevel_AUTO_I} := 0, \\
\quad \quad \text{eBAREnergyLevel_Protection} := 1, \\
\quad \quad \text{eBAREnergyLevel_Economy} := 2, \\
\quad \quad \text{eBAREnergyLevel_PreComfort} := 3, \\
\quad \quad \text{eBAREnergyLevel_Comfort} := 4, \\
\quad \quad \text{eBAREnergyLevel_AUTO_II} := 5 \\
\quad \}; \\
\text{END_TYPE}
\]

\text{eBAREnergyLevel_AUTO_I}: Automatic-mode-I means, in this mode a switch to a higher energy level is possible if the upper energy level are already TRUE. In the mode the state of the inputs \text{bProtection}, \text{bEconomy}, \text{bPreComfort} and \text{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_BARPposMode

Enumerator for the definition of the positioning mode.

```plaintext
TYPE E_BARPposMode :
{
    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 [303];

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 [245] 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.

```plaintext
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.

```plaintext
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.

```plaintext
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[303]. It is set to TRUE by the function block FB_BARBlindPositionEntry[245] if the data entered correspond to the validity criteria described.

5.2.41.7 ST_BARCorner

Information about window corners

```plaintext
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).

```plaintext
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;
  stBARSetpointRoom_PreComfortCool : REAL:= 28.0;
  stBARSetpointRoom_EconomyCool : REAL:= 35.0;
  stBARSetpointRoom_ProtectionCool : REAL:= 40.0;
END_STRUCT
END_TYPE

The values in the structure are defined with the preset values.

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 [283]. The blind actuators FB_BARSunblindActuator [270] and FB_BARRollerBlind [284] 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_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_bHVACPersUsed: NOVRAM has been written to.

g_bHVACPersWriteBusy: Persistent data have been created.

g_bHVACParamsChanged: Triggers a backup.

5.3.2 Parameter

Global parameters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>iRowsPerFacade</td>
<td>INT</td>
</tr>
<tr>
<td>iColumnsPerFacade</td>
<td>INT</td>
</tr>
<tr>
<td>iShadingObjects</td>
<td>INT</td>
</tr>
<tr>
<td>usiMaxSunblindScenes</td>
<td>USINT</td>
</tr>
<tr>
<td>g_tHVACWriteBackupDateTime</td>
<td>TIME</td>
</tr>
<tr>
<td>g_udiMaxNoOfBytesInStruct</td>
<td>UDINT</td>
</tr>
<tr>
<td>g_udiMaxSizeOfString</td>
<td>UDINT</td>
</tr>
<tr>
<td>g_iMaxOfScale_nPoint</td>
<td>INT</td>
</tr>
<tr>
<td>g_iMaxNumberOfSteps</td>
<td>INT</td>
</tr>
<tr>
<td>g_iMinNumberOfSteps</td>
<td>INT</td>
</tr>
<tr>
<td>g_iMaxNumberOfProfiles</td>
<td>INT</td>
</tr>
<tr>
<td>g_iMinNumberOfProfiles</td>
<td>INT</td>
</tr>
<tr>
<td>g_iMaxNumberOfAggregates</td>
<td>INT</td>
</tr>
<tr>
<td>g_iMinNumberOfAggregates</td>
<td>INT</td>
</tr>
<tr>
<td>g_iAggregateMinNumberOfSteps</td>
<td>INT</td>
</tr>
<tr>
<td>g_iAggregateMaxNumberOfSteps</td>
<td>INT</td>
</tr>
<tr>
<td>g_iMinNumberOfSequences</td>
<td>INT</td>
</tr>
<tr>
<td>g_iMaxNumberOfSequences</td>
<td>INT</td>
</tr>
</tbody>
</table>

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 [496] (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.
Switching to next sequence after 10 seconds

\( r_{XW} = 1 \)
\( r_{DeadBand} = 10 \)

\[ \Delta t = 10 \text{ s} \]

Switching to next sequence after 5 seconds

\( r_{XW} = 2 \)
\( r_{DeadBand} = 10 \)

\[ \Delta t = 5 \text{ s} \]
### 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.

<table>
<thead>
<tr>
<th>Value of: <code>E_HVACSequenceCtrlMode</code></th>
<th>0 (Stop)</th>
<th>1 (On)</th>
<th>2 (night cooling)</th>
<th>3 (support operation)</th>
<th>4 (overheating protection)</th>
<th>5 (night cooling and overheating protection)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_HVACMasterSequenceCtrl Master controller</td>
<td>disabled 0 %</td>
<td>Enable</td>
<td>disabled 0 %</td>
<td>Enable <code>rY=ZuluftMax. Temp</code></td>
<td>Enable <code>rY=ZuluftMin. Temp</code></td>
<td>Enable <code>rY=ZuluftMin. Temp</code></td>
</tr>
<tr>
<td>FB_HVACPID PreHeating Pre-heater</td>
<td>disabled 0 %</td>
<td>Enable</td>
<td>disabled 0 %</td>
<td>Enable 0 % - 100 %</td>
<td>disabled 0 %</td>
<td>disabled 0 %</td>
</tr>
<tr>
<td>FB_HVACPID ReHeating Re-heater</td>
<td>disabled 0 %</td>
<td>Enable</td>
<td>disabled 0 %</td>
<td>disabled 0 %</td>
<td>disabled 0 %</td>
<td>disabled 0 %</td>
</tr>
<tr>
<td>FB_HVACPID Cooling Cooler</td>
<td>disabled 0 %</td>
<td>Enable</td>
<td>disabled 0 %</td>
<td>disabled 0 %</td>
<td>Enable 0 % - 100 %</td>
<td>Enable 0 % - 100 %</td>
</tr>
<tr>
<td>FB_HVACPID EnergyRecovery Heat recovery</td>
<td>disabled 0 %</td>
<td>Enable</td>
<td>disabled 0 %</td>
<td>disabled 0 %</td>
<td>disabled 0 %</td>
<td>disabled 0 %</td>
</tr>
<tr>
<td>FB_HVACPID MixedAir Mixed air flaps</td>
<td>disabled 0 %</td>
<td>Enable</td>
<td>max. outside air rate 100 %</td>
<td>0 % air circulation only</td>
<td>0 % air circulation only</td>
<td>max. outside air rate 100 %</td>
</tr>
</tbody>
</table>

### 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.

**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 her internet pages: [https://www.beckhoff.com](https://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
Fax: +49 5246 963 9157
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
Fax: +49 5246 963 479
e-mail: service@beckhoff.com

Beckhoff Headquarters

Beckhoff Automation GmbH & Co. KG
Huelshorstweg 20
33415 Verl
Germany

Phone: +49 5246 963 0
Fax: +49 5246 963 198
e-mail: info@beckhoff.com
web: https://www.beckhoff.com
6.4 ADS Return Codes

Grouping of error codes: 0x000..., 0x511..., 0x700..., 0x1000...

Global error codes

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>HRESULT</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>0</td>
<td>0x9811 0000</td>
<td>ERR_NOERROR</td>
<td>No error.</td>
</tr>
<tr>
<td>0x1</td>
<td>1</td>
<td>0x9811 0001</td>
<td>ERR_INTERNAL</td>
<td>Internal error.</td>
</tr>
<tr>
<td>0x2</td>
<td>2</td>
<td>0x9811 0002</td>
<td>ERR_NORTIME</td>
<td>No real-time.</td>
</tr>
<tr>
<td>0x3</td>
<td>3</td>
<td>0x9811 0003</td>
<td>ERR_ALLOCCLOCKEDMEM</td>
<td>Allocation locked – memory error.</td>
</tr>
<tr>
<td>0x4</td>
<td>4</td>
<td>0x9811 0004</td>
<td>ERR_INSERTMAILBOX</td>
<td>Mailbox full – the ADS message could not be sent. Reducing the number of ADS messages per cycle will help.</td>
</tr>
<tr>
<td>0x5</td>
<td>5</td>
<td>0x9811 0005</td>
<td>ERR_WRONGRECEIVEHMSG</td>
<td>Wrong HMSG.</td>
</tr>
<tr>
<td>0x6</td>
<td>6</td>
<td>0x9811 0006</td>
<td>ERR_TARGETPORTNOTFOUND</td>
<td>Target port not found – ADS server is not started or is not reachable.</td>
</tr>
<tr>
<td>0x7</td>
<td>7</td>
<td>0x9811 0007</td>
<td>ERR_TARGETMACHINENOTFOUND</td>
<td>Target computer not found – AMS route was not found.</td>
</tr>
<tr>
<td>0x8</td>
<td>8</td>
<td>0x9811 0008</td>
<td>ERR_UNKNOWNCMDID</td>
<td>Unknown command ID.</td>
</tr>
<tr>
<td>0x9</td>
<td>9</td>
<td>0x9811 0009</td>
<td>ERR_BADTASKID</td>
<td>Invalid task ID.</td>
</tr>
<tr>
<td>0xA</td>
<td>10</td>
<td>0x9811 000A</td>
<td>ERR_NOIO</td>
<td>No IO.</td>
</tr>
<tr>
<td>0xB</td>
<td>11</td>
<td>0x9811 000B</td>
<td>ERR_UNKNOWNAMSCMD</td>
<td>Unknown AMS command.</td>
</tr>
<tr>
<td>0xC</td>
<td>12</td>
<td>0x9811 000C</td>
<td>ERR_WIN32ERROR</td>
<td>Win32 error.</td>
</tr>
<tr>
<td>0xD</td>
<td>13</td>
<td>0x9811 000D</td>
<td>ERR_PORTNOTCONNECTED</td>
<td>Port not connected.</td>
</tr>
<tr>
<td>0xE</td>
<td>14</td>
<td>0x9811 000E</td>
<td>ERR_INVALIDAMSLENGTH</td>
<td>Invalid AMS length.</td>
</tr>
<tr>
<td>0xF</td>
<td>15</td>
<td>0x9811 000F</td>
<td>ERR_INVALIDAMSNETID</td>
<td>Invalid AMS Net ID.</td>
</tr>
<tr>
<td>0x10</td>
<td>16</td>
<td>0x9811 0010</td>
<td>ERR_LOWINSTLEVEL</td>
<td>Installation level is too low – TwinCAT 2 license error.</td>
</tr>
<tr>
<td>0x11</td>
<td>17</td>
<td>0x9811 0011</td>
<td>ERR_NODEBUGINTAVAILABLE</td>
<td>No debugging available.</td>
</tr>
<tr>
<td>0x12</td>
<td>18</td>
<td>0x9811 0012</td>
<td>ERR_PORTDISABLE</td>
<td>Port disabled – TwinCAT system service not started.</td>
</tr>
<tr>
<td>0x13</td>
<td>19</td>
<td>0x9811 0013</td>
<td>ERR_PORTALREADYCONNECTED</td>
<td>Port already connected.</td>
</tr>
<tr>
<td>0x14</td>
<td>20</td>
<td>0x9811 0014</td>
<td>ERR_AMSSYNC_W32ERROR</td>
<td>AMS Sync Win32 error.</td>
</tr>
<tr>
<td>0x15</td>
<td>21</td>
<td>0x9811 0015</td>
<td>ERR_AMSSYNC_TIMEOUT</td>
<td>AMS Sync Timeout.</td>
</tr>
<tr>
<td>0x16</td>
<td>22</td>
<td>0x9811 0016</td>
<td>ERR_AMSSYNC_AMSERROR</td>
<td>AMS Sync error.</td>
</tr>
<tr>
<td>0x17</td>
<td>23</td>
<td>0x9811 0017</td>
<td>ERR_AMSSYNC_NOINDEXINMAP</td>
<td>No index map for AMS Sync available.</td>
</tr>
<tr>
<td>0x18</td>
<td>24</td>
<td>0x9811 0018</td>
<td>ERR_INVALIDAMSPORT</td>
<td>Invalid AMS port.</td>
</tr>
<tr>
<td>0x19</td>
<td>25</td>
<td>0x9811 0019</td>
<td>ERR_NOMEMORY</td>
<td>No memory.</td>
</tr>
<tr>
<td>0x1A</td>
<td>26</td>
<td>0x9811 001A</td>
<td>ERR_TCPSEND</td>
<td>TCP send error.</td>
</tr>
<tr>
<td>0x1B</td>
<td>27</td>
<td>0x9811 001B</td>
<td>ERR_HOSTUNREACHABLE</td>
<td>Host unreachable.</td>
</tr>
<tr>
<td>0x1C</td>
<td>28</td>
<td>0x9811 001C</td>
<td>ERR_INVALIDAMSFRAAGMENT</td>
<td>Invalid AMS fragment.</td>
</tr>
<tr>
<td>0x1D</td>
<td>29</td>
<td>0x9811 001D</td>
<td>ERR_TLSSEND</td>
<td>TLS send error – secure ADS connection failed.</td>
</tr>
<tr>
<td>0x1E</td>
<td>30</td>
<td>0x9811 001E</td>
<td>ERR_ACCESSDENIED</td>
<td>Access denied – secure ADS access denied.</td>
</tr>
</tbody>
</table>

Router error codes
<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>HRESULT</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x500</td>
<td>1280</td>
<td>0x9811 0500</td>
<td>ROUTERERR_NOLOCKEDMEMORY</td>
<td>Locked memory cannot be allocated.</td>
</tr>
<tr>
<td>0x501</td>
<td>1281</td>
<td>0x9811 0501</td>
<td>ROUTERERR_RESIZEMEMORY</td>
<td>The router memory size could not be changed.</td>
</tr>
<tr>
<td>0x502</td>
<td>1282</td>
<td>0x9811 0502</td>
<td>ROUTERERR_MAILBOXFULL</td>
<td>The mailbox has reached the maximum number of possible messages.</td>
</tr>
<tr>
<td>0x503</td>
<td>1283</td>
<td>0x9811 0503</td>
<td>ROUTERERR_DEBUGBOXFULL</td>
<td>The Debug mailbox has reached the maximum number of possible messages.</td>
</tr>
<tr>
<td>0x504</td>
<td>1284</td>
<td>0x9811 0504</td>
<td>ROUTERERR_UNKNOWNPORTTYPE</td>
<td>The port type is unknown.</td>
</tr>
<tr>
<td>0x505</td>
<td>1285</td>
<td>0x9811 0505</td>
<td>ROUTERERR_NOTINITIALIZED</td>
<td>The router is not initialized.</td>
</tr>
<tr>
<td>0x506</td>
<td>1286</td>
<td>0x9811 0506</td>
<td>ROUTERERR_PORTALREADYINUSE</td>
<td>The port number is already assigned.</td>
</tr>
<tr>
<td>0x507</td>
<td>1287</td>
<td>0x9811 0507</td>
<td>ROUTERERR_NOTREGISTERED</td>
<td>The port is not registered.</td>
</tr>
<tr>
<td>0x508</td>
<td>1288</td>
<td>0x9811 0508</td>
<td>ROUTERERR_NOMOREQUEUES</td>
<td>The maximum number of ports has been reached.</td>
</tr>
<tr>
<td>0x509</td>
<td>1289</td>
<td>0x9811 0509</td>
<td>ROUTERERR_INVALIDPORT</td>
<td>The port is invalid.</td>
</tr>
<tr>
<td>0x50A</td>
<td>1290</td>
<td>0x9811 050A</td>
<td>ROUTERERR_NOTACTIVATED</td>
<td>The router is not active.</td>
</tr>
<tr>
<td>0x50B</td>
<td>1291</td>
<td>0x9811 050B</td>
<td>ROUTERERR_FRAGMENTBOXFULL</td>
<td>The mailbox has reached the maximum number for fragmented messages.</td>
</tr>
<tr>
<td>0x50C</td>
<td>1292</td>
<td>0x9811 050C</td>
<td>ROUTERERR_FRAGMENTTIMEOUT</td>
<td>A fragment timeout has occurred.</td>
</tr>
<tr>
<td>0x50D</td>
<td>1293</td>
<td>0x9811 050D</td>
<td>ROUTERERR_TOBEREMOVED</td>
<td>The port is removed.</td>
</tr>
</tbody>
</table>

**General ADS error codes**
<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>HRESULT</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x700</td>
<td>1792</td>
<td>0x9811 0700</td>
<td>ADSERR_DEVICE_ERROR</td>
<td>General device error.</td>
</tr>
<tr>
<td>0x701</td>
<td>1793</td>
<td>0x9811 0701</td>
<td>ADSERR_DEVICE_SRVNOTSUPP</td>
<td>Service is not supported by the server.</td>
</tr>
<tr>
<td>0x702</td>
<td>1794</td>
<td>0x9811 0702</td>
<td>ADSERR_DEVICE_INVALIDGRP</td>
<td>Invalid index group.</td>
</tr>
<tr>
<td>0x703</td>
<td>1795</td>
<td>0x9811 0703</td>
<td>ADSERR_DEVICE_INVALIDOFFSET</td>
<td>Invalid index offset.</td>
</tr>
<tr>
<td>0x704</td>
<td>1796</td>
<td>0x9811 0704</td>
<td>ADSERR_DEVICE_INVALIDACCESS</td>
<td>Reading or writing not permitted.</td>
</tr>
<tr>
<td>0x705</td>
<td>1797</td>
<td>0x9811 0705</td>
<td>ADSERR_DEVICE_INVALIDSIZE</td>
<td>Parameter size not correct.</td>
</tr>
<tr>
<td>0x706</td>
<td>1798</td>
<td>0x9811 0706</td>
<td>ADSERR_DEVICE_INVALIDDATA</td>
<td>Invalid data values.</td>
</tr>
<tr>
<td>0x707</td>
<td>1799</td>
<td>0x9811 0707</td>
<td>ADSERR_DEVICE_NOTREADY</td>
<td>Device is not ready to operate.</td>
</tr>
<tr>
<td>0x708</td>
<td>1800</td>
<td>0x9811 0708</td>
<td>ADSERR_DEVICE_BUSY</td>
<td>Device is busy.</td>
</tr>
<tr>
<td>0x709</td>
<td>1801</td>
<td>0x9811 0709</td>
<td>ADSERR_DEVICE_INVALIDCONTEXT</td>
<td>Invalid operating system context. This can result from use of ADS function blocks in different tasks. It may be possible to resolve this through Multi-task data access synchronization in the PLC.</td>
</tr>
<tr>
<td>0x70A</td>
<td>1802</td>
<td>0x9811 070A</td>
<td>ADSERR_DEVICE_NOMEMORY</td>
<td>Insufficient memory.</td>
</tr>
<tr>
<td>0x70B</td>
<td>1803</td>
<td>0x9811 070B</td>
<td>ADSERR_DEVICE_INVALIDPARAM</td>
<td>Invalid parameter values.</td>
</tr>
<tr>
<td>0x70C</td>
<td>1804</td>
<td>0x9811 070C</td>
<td>ADSERR_DEVICE_NOTFOUND</td>
<td>Not found (files,...).</td>
</tr>
<tr>
<td>0x70D</td>
<td>1805</td>
<td>0x9811 070D</td>
<td>ADSERR_DEVICE_SYNTAX</td>
<td>Syntax error in file or command.</td>
</tr>
<tr>
<td>0x70E</td>
<td>1806</td>
<td>0x9811 070E</td>
<td>ADSERR_DEVICE_INCOMPATIBLE</td>
<td>Objects do not match.</td>
</tr>
<tr>
<td>0x70F</td>
<td>1807</td>
<td>0x9811 070F</td>
<td>ADSERR_DEVICE_EXISTS</td>
<td>Object already exists.</td>
</tr>
<tr>
<td>0x710</td>
<td>1808</td>
<td>0x9811 0710</td>
<td>ADSERR_DEVICE_SYMBOLNOTFOUND</td>
<td>Symbol not found.</td>
</tr>
<tr>
<td>0x711</td>
<td>1809</td>
<td>0x9811 0711</td>
<td>ADSERR_DEVICE_SYMBOLVERSIONINVALID</td>
<td>Invalid symbol version. This can occur due to an online change. Create a new handle.</td>
</tr>
<tr>
<td>0x712</td>
<td>1810</td>
<td>0x9811 0712</td>
<td>ADSERR_DEVICE_INVALIDSTATE</td>
<td>Device (server) is in invalid state.</td>
</tr>
<tr>
<td>0x713</td>
<td>1811</td>
<td>0x9811 0713</td>
<td>ADSERR_DEVICE_TRANSMODENOTSUPP</td>
<td>AdsTransMode not supported.</td>
</tr>
<tr>
<td>0x714</td>
<td>1812</td>
<td>0x9811 0714</td>
<td>ADSERR_DEVICE_NOTIFYHANDINVALID</td>
<td>Notification handle is invalid.</td>
</tr>
<tr>
<td>0x715</td>
<td>1813</td>
<td>0x9811 0715</td>
<td>ADSERR_DEVICE_CLIENTUNKNOWN</td>
<td>Notification client not registered.</td>
</tr>
<tr>
<td>0x716</td>
<td>1814</td>
<td>0x9811 0716</td>
<td>ADSERR_DEVICE_NOMOREHDLS</td>
<td>No further handle available.</td>
</tr>
<tr>
<td>0x717</td>
<td>1815</td>
<td>0x9811 0717</td>
<td>ADSERR_DEVICE_INVALIDWATCHSIZE</td>
<td>Notification size too large.</td>
</tr>
<tr>
<td>0x718</td>
<td>1816</td>
<td>0x9811 0718</td>
<td>ADSERR_DEVICE_NOTINIT</td>
<td>Device not initialized.</td>
</tr>
<tr>
<td>0x719</td>
<td>1817</td>
<td>0x9811 0719</td>
<td>ADSERR_DEVICE_TIMEROUT</td>
<td>Device has a timeout.</td>
</tr>
<tr>
<td>0x71A</td>
<td>1818</td>
<td>0x9811 071A</td>
<td>ADSERR_DEVICE_NOINTERFACE</td>
<td>Interface query failed.</td>
</tr>
<tr>
<td>0x71B</td>
<td>1819</td>
<td>0x9811 071B</td>
<td>ADSERR_DEVICE_INVALIDINTERFACE</td>
<td>Wrong interface requested.</td>
</tr>
<tr>
<td>0x71C</td>
<td>1820</td>
<td>0x9811 071C</td>
<td>ADSERR_DEVICE_INVALIDCLSID</td>
<td>Class ID is invalid.</td>
</tr>
<tr>
<td>0x71D</td>
<td>1821</td>
<td>0x9811 071D</td>
<td>ADSERR_DEVICE_INVALIDOBJID</td>
<td>Object ID is invalid.</td>
</tr>
<tr>
<td>0x71E</td>
<td>1822</td>
<td>0x9811 071E</td>
<td>ADSERR_DEVICE_PENDING</td>
<td>Request pending.</td>
</tr>
<tr>
<td>0x71F</td>
<td>1823</td>
<td>0x9811 071F</td>
<td>ADSERRDEVICE_ABORTED</td>
<td>Request is aborted.</td>
</tr>
<tr>
<td>0x720</td>
<td>1824</td>
<td>0x9811 0720</td>
<td>ADSERR_DEVICE_WARNING</td>
<td>Signal warning.</td>
</tr>
<tr>
<td>0x721</td>
<td>1825</td>
<td>0x9811 0721</td>
<td>ADSERR_DEVICE_INVALIDARRAYIDX</td>
<td>Invalid array index.</td>
</tr>
<tr>
<td>0x722</td>
<td>1826</td>
<td>0x9811 0722</td>
<td>ADSERR_DEVICE_SYMBOLNOTACTIVE</td>
<td>Symbol not active.</td>
</tr>
<tr>
<td>0x723</td>
<td>1827</td>
<td>0x9811 0723</td>
<td>ADSERR_DEVICE_ACCESSDENIED</td>
<td>Access denied.</td>
</tr>
<tr>
<td>0x724</td>
<td>1828</td>
<td>0x9811 0724</td>
<td>ADSERR_DEVICE_LICENSENOTFOUND</td>
<td>Missing license.</td>
</tr>
<tr>
<td>0x725</td>
<td>1829</td>
<td>0x9811 0725</td>
<td>ADSERR_DEVICE_LICENSEEXPIRED</td>
<td>License expired.</td>
</tr>
<tr>
<td>0x726</td>
<td>1830</td>
<td>0x9811 0726</td>
<td>ADSERR_DEVICE_LICENSEEXCEEDED</td>
<td>License exceeded.</td>
</tr>
<tr>
<td>0x727</td>
<td>1831</td>
<td>0x9811 0727</td>
<td>ADSERR_DEVICE_LICENSEINVALID</td>
<td>Invalid license.</td>
</tr>
<tr>
<td>0x728</td>
<td>1832</td>
<td>0x9811 0728</td>
<td>ADSERR_DEVICE_LICENSESYSTEMID</td>
<td>License problem: System ID is invalid.</td>
</tr>
<tr>
<td>0x729</td>
<td>1833</td>
<td>0x9811 0729</td>
<td>ADSERR_DEVICE_LICENSEENOTLIMIT</td>
<td>License not limited in time.</td>
</tr>
<tr>
<td>0x72A</td>
<td>1834</td>
<td>0x9811 072A</td>
<td>ADSERR_DEVICE_LICENSEFUTUREISSUE</td>
<td>License problem: Time in the future.</td>
</tr>
<tr>
<td>0x72B</td>
<td>1835</td>
<td>0x9811 072B</td>
<td>ADSERR_DEVICE_LICENSESETIMETOLONG</td>
<td>License period too long.</td>
</tr>
<tr>
<td>0x72C</td>
<td>1836</td>
<td>0x9811 072C</td>
<td>ADSERR_DEVICE_EXCEPTION</td>
<td>Exception at system startup.</td>
</tr>
<tr>
<td>0x72D</td>
<td>1837</td>
<td>0x9811 072D</td>
<td>ADSERR_DEVICE_LICENSEDEPLEXICATED</td>
<td>License file read twice.</td>
</tr>
<tr>
<td>0x72E</td>
<td>1838</td>
<td>0x9811 072E</td>
<td>ADSERR_DEVICE_LICENSESIGNATUREINVALID</td>
<td>Invalid signature.</td>
</tr>
<tr>
<td>0x72F</td>
<td>1839</td>
<td>0x9811 072F</td>
<td>ADSERR_DEVICE_CERTIFICATEINVALID</td>
<td>Invalid certificate.</td>
</tr>
<tr>
<td>0x730</td>
<td>1840</td>
<td>0x9811 0730</td>
<td>ADSERR_DEVICE_LICENSEOEMNOTFOUND</td>
<td>Public key not known from OEM.</td>
</tr>
<tr>
<td>0x731</td>
<td>1841</td>
<td>0x9811 0731</td>
<td>ADSERR_DEVICE_LICENSERESTRICTED</td>
<td>License not valid for this system ID.</td>
</tr>
<tr>
<td>0x732</td>
<td>1842</td>
<td>0x9811 0732</td>
<td>ADSERR_DEVICE_LICENSEDEMODENIED</td>
<td>Demo license prohibited.</td>
</tr>
<tr>
<td>0x733</td>
<td>1843</td>
<td>0x9811 0733</td>
<td>ADSERR_DEVICE_INVALIDFNCID</td>
<td>Invalid function ID.</td>
</tr>
<tr>
<td>0x734</td>
<td>1844</td>
<td>0x9811 0734</td>
<td>ADSERR_DEVICE_OUTOFRANGE</td>
<td>Outside the valid range.</td>
</tr>
<tr>
<td>0x735</td>
<td>1845</td>
<td>0x9811 0735</td>
<td>ADSERR_DEVICE_INVALIDALIGNMENT</td>
<td>Invalid alignment.</td>
</tr>
</tbody>
</table>
### TCP Winsock error codes

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x274C</td>
<td>10060</td>
<td>WSAETIMEOUT</td>
<td>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.</td>
</tr>
<tr>
<td>0x274D</td>
<td>10061</td>
<td>WSAECONNREFUSED</td>
<td>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.</td>
</tr>
<tr>
<td>0x2751</td>
<td>10065</td>
<td>WSAEHOSTUNREACH</td>
<td>No route to host - a socket operation referred to an unavailable host.</td>
</tr>
</tbody>
</table>

### RTime error codes

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>HRESULT</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1000</td>
<td>4096</td>
<td>0x9811 1000</td>
<td>RTERR INTERNAL</td>
<td>Internal error in the real-time system.</td>
</tr>
<tr>
<td>0x1001</td>
<td>4097</td>
<td>0x9811 1001</td>
<td>RTERR_BATIMERPERIODS</td>
<td>Timer value is not valid.</td>
</tr>
<tr>
<td>0x1002</td>
<td>4098</td>
<td>0x9811 1002</td>
<td>RTERR_INVALIDTASKPTR</td>
<td>Task pointer has the invalid value 0 (zero).</td>
</tr>
<tr>
<td>0x1003</td>
<td>4099</td>
<td>0x9811 1003</td>
<td>RTERR_INVALIDSTACKPTR</td>
<td>Stack pointer has the invalid value 0 (zero).</td>
</tr>
<tr>
<td>0x1004</td>
<td>4100</td>
<td>0x9811 1004</td>
<td>RTERR_PRIOEXISTS</td>
<td>The request task priority is already assigned.</td>
</tr>
<tr>
<td>0x1005</td>
<td>4101</td>
<td>0x9811 1005</td>
<td>RTERR_NOMORETCB</td>
<td>No free TCB (Task Control Block) available. The maximum number of TCBs is 64.</td>
</tr>
<tr>
<td>0x1006</td>
<td>4102</td>
<td>0x9811 1006</td>
<td>RTERR_NOMORESEMAS</td>
<td>No free semaphores available. The maximum number of semaphores is 64.</td>
</tr>
<tr>
<td>0x1007</td>
<td>4103</td>
<td>0x9811 1007</td>
<td>RTERR_NOMOREQUEUES</td>
<td>No free space available in the queue. The maximum number of positions in the queue is 64.</td>
</tr>
<tr>
<td>0x100D</td>
<td>4109</td>
<td>0x9811 100D</td>
<td>RTERR_EXTIRQALREADYDEF</td>
<td>An external synchronization interrupt is already applied.</td>
</tr>
<tr>
<td>0x100E</td>
<td>4110</td>
<td>0x9811 100E</td>
<td>RTERR_EXTIRQNOTDEF</td>
<td>No external sync interrupt applied.</td>
</tr>
<tr>
<td>0x100F</td>
<td>4111</td>
<td>0x9811 100F</td>
<td>RTERR_EXTIRQINSTALLFAILED</td>
<td>Application of the external synchronization interrupt has failed.</td>
</tr>
<tr>
<td>0x1010</td>
<td>4112</td>
<td>0x9811 1010</td>
<td>RTERR_IRQLNOTLESSOREQUAL</td>
<td>Call of a service function in the wrong context</td>
</tr>
<tr>
<td>0x1017</td>
<td>4119</td>
<td>0x9811 1017</td>
<td>RTERR_VMXNOTSUPPORTED</td>
<td>Intel VT-x extension is not supported.</td>
</tr>
<tr>
<td>0x1018</td>
<td>4120</td>
<td>0x9811 1018</td>
<td>RTERR_VMXDISABLED</td>
<td>Intel VT-x extension is not enabled in the BIOS.</td>
</tr>
<tr>
<td>0x1019</td>
<td>4121</td>
<td>0x9811 1019</td>
<td>RTERR_VMXCONTROLMISSING</td>
<td>Missing function in Intel VT-x extension.</td>
</tr>
<tr>
<td>0x101A</td>
<td>4122</td>
<td>0x9811 101A</td>
<td>RTERR_VMXENABLEFAILS</td>
<td>Activation of Intel VT-x fails.</td>
</tr>
</tbody>
</table>

### More Winsock error codes: Win32 error codes