BECKHOFF New Automation Technology

Handbuch | DE

TS8000

TwinCAT 2 | PLC HVAC

Supplement | Building Automation

31.08.2023 | Version: 1.2

Inhaltsverzeichnis

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.
Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der Dokumentation und der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.
Das Fachpersonal ist verpflichtet, für jede Installation und Inbetriebnahme die zu dem betreffenden Zeitpunkt veröffentliche Dokumentation zu verwenden.

Das Fachpersonal hat sicherzustellen, dass die Anwendung bzw. der Einsatz der beschriebenen Produkte alle Sicherheitsanforderungen, einschließlich sämtlicher anwendbaren Gesetze, Vorschriften, Bestimmungen und Normen erfüllt.

Disclaimer

Diese Dokumentation wurde sorgfältig erstellt. Die beschriebenen Produkte werden jedoch ständig weiter entwickelt.
Wir behalten uns das Recht vor, die Dokumentation jederzeit und ohne Ankündigung zu überarbeiten und zu ändern.
Aus den Angaben, Abbildungen und Beschreibungen in dieser Dokumentation können keine Ansprüche auf Änderung bereits gelieferter Produkte geltend gemacht werden.

Marken

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® und XPlanar® sind eingetragene und lizenzierte Marken der Beckhoff Automation GmbH.
Die Verwendung anderer in dieser Dokumentation enthaltenen Marken oder Kennzeichen durch Dritte kann zu einer Verletzung von Rechten der Inhaber der entsprechenden Bezeichnungen führen.

Patente

Die EtherCAT-Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.

EtherCAT® ist eine eingetragene Marke und patentierte Technologie lizenziert durch die Beckhoff Automation GmbH, Deutschland

Copyright

1.2 Zu Ihrer Sicherheit

Sicherheitsbestimmungen

Lesen Sie die folgenden Erklärungen zu Ihrer Sicherheit.
Beachten und befolgen Sie stets produktspezifische Sicherheitshinweise, die Sie gegebenenfalls an den entsprechenden Stellen in diesem Dokument vorfinden.

Haftungsausschluss

Die gesamten Komponenten werden je nach Anwendungsbestimmungen in bestimmten Hard- und Software-Konfigurationen ausgeliefert. Änderungen der Hard- oder Software-Konfiguration, die über die dokumentierten Möglichkeiten hinausgehen, sind unzulässig und bewirken den Haftungsausschluss der Beckhoff Automation GmbH & Co. KG.

Qualifikation des Personals

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs-, Automatisierungs- und Antriebstechnik, das mit den geltenden Normen vertraut ist.

Signalwörter

Im Folgenden werden die Signalwörter eingeordnet, die in der Dokumentation verwendet werden. Um Personen- und Sachschäden zu vermeiden, lesen und befolgen Sie die Sicherheits- und Warnhinweise.

Warnungen vor Personenschäden

⚠ GEFAHR

Es besteht eine Gefährdung mit hohem Risikograd, die den Tod oder eine schwere Verletzung zur Folge hat.

⚠ WARNUNG

Es besteht eine Gefährdung mit mittlerem Risikograd, die den Tod oder eine schwere Verletzung zur Folge haben kann.

⚠ VORSICHT

Es besteht eine Gefährdung mit geringem Risikograd, die eine mittelschwere oder leichte Verletzung zur Folge haben kann.

Warnung vor Umwelt- oder Sachschäden

HINWEIS

Es besteht eine mögliche Schädigung für Umwelt, Geräte oder Daten.

Information zum Umgang mit dem Produkt

Diese Information beinhaltet z. B.:
Handlungsempfehlungen, Hilfestellungen oder weiterführende Informationen zum Produkt.

1.3 Hinweise zur Informationssicherheit

Die Produkte der Beckhoff Automation GmbH & Co. KG (Beckhoff) sind, sofern sie online zu erreichen sind, mit Security-Funktionen ausgestattet, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen. Trotz der Security-Funktionen sind die Erstellung, Implementierung und ständige Aktualisierung eines ganzheitlichen Security-Konzepts für den Betrieb notwendig, um die jeweilige Anlage, das System, die Maschine und die Netzwerke gegen Cyber-Bedrohungen zu schützen. Die von Beckhoff verkauften Produkte bilden dabei nur einen Teil des gesamtheitlichen Security-Konzepts. Der Kunde ist dafür verantwortlich, dass unbefugte Zugriffe durch Dritte auf seine Anlagen, Systeme, Maschinen und Netzwerke verhindert werden. Letztere sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn entsprechende Schutzmaßnahmen eingerichtet wurden.

Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem https://www.beckhoff.de/secguide.

Die Produkte und Lösungen von Beckhoff werden ständig weiterentwickelt. Dies betrifft auch die Security-Funktionen. Aufgrund der stetigen Weiterentwicklung empfiehlt Beckhoff ausdrücklich, die Produkte ständig auf dem aktuellen Stand zu halten und nach Bereitstellung von Updates diese auf die Produkte aufzuspielen. Die Verwendung veralteter oder nicht mehr unterstützter Produktversionen kann das Risiko von Cyber-Bedrohungen erhöhen.

Um stets über Hinweise zur Informationssicherheit zu Produkten von Beckhoff informiert zu sein, abonnieren Sie den RSS Feed unter https://www.beckhoff.de/secinfo.

2 Einleitung

Vorwort

Komfort, Energieeinsparung, geringe Investitions- und Betriebskosten und somit ein schneller Return Of Investment sind die hochgesteckten Ziele eines Gebäudeautomationssystems. Eine Grundvoraussetzung dafür ist ein durchgängiges, aufeinander abgestimmtes Steuerungssytem für die Automation aller technischen Ausbaugewerke.

Durch den Einsatz der HVAC-Bibliothek sind alle SPS-Programme von der Heizungszentrale über die Vollklimaanlage bis hin zu den Raumautomationsfunktionen mit TwinCAT PLC Control programmierbar und stehen innerhalb der Gebäudeautomationsbibliotheken als Bausteine zur Verfügung.
Dem Beckhoff Systemintegrator wird damit ein großes Spektrum an etablierten und geprüften Funktionen zur Verfügung gestellt. Dieses erleichtert die Implementierung der komplexen und interoperablen Gebäudeautomationsfunktionen und senkt den Engineering Aufwand. Zusätzlich werden die Lebenszykluskosten des Systems reduziert, weil sich der Aufwand für die Wartung und Instandhaltung auf ein und die selbe Programmiersoftware und Hardwareplattform konzentriert.

Mit den Raumautomationsfunktionen für die Klimatisierung, die Beleuchtung und den Sonnenschutz ist es möglich Systeme zu errichten, welche den höchsten Energieeffizienzstandard der EN 15232 Klasse A erreichen.

Zudem bietet die objektorientierte Kapselung der Gebäudeautomationsfunktionen die folgenden Vorteile:

- Schnelle Erstellung der Anlagenprogramme.
- Schnelle Parametrierung und Inbetriebnahme der Anlagen.
- Gewährleistung eines stets sehr hohen Anlagenfunktionsumfang.
- Gute Lesbarkeit der Programme (Voraussetzung für eine langjährige Wartbarkeit und Erweiterbarkeit der Anlagen).
- Gute Wiederverwendbarkeit einmal erstellter Vorlagen für Anlagen oder Anlagenbaugruppen.
- Leichte Einarbeitung des Personals.
- Leichte Erweiterung und Änderung bereits existierender Anlagen.
- Vorgaben für eine klare objektorientierte Struktur zur Erstellung von Visualisierungsobjekten in MMI- und SCADA-Systemen.
- Bessere Dokumentierbarkeit der Programme.

2.1 Zielgruppen

Diese Softwarebibliothek ist für Gebäudeautomations-Systempartner der Beckhoff Automation GmbH. Die Systempartner sind tätig in dem Bereich Gebäudeautomation und beschäftigen sich mit Errichtung, Inbetriebsetzung, Erweiterung, Wartung und Service von mess-, steuer- und regelungstechnischen Anlagen der technischen Gebäudeausrüstung.

2.2 Anforderungsprofil des Anwenders

Für den Nutzer dieser Bibliothek werden die folgenden Grundkenntnisse vorausgesetzt.

- TwinCAT PLC Control
- TwinCAT System Manager
- PC und Netzwerkkenntnisse
- Aufbau und Eigenschaften der Beckhoff Embedded-PC und deren Busklemmensystem.
- Technologie von Heizungs-, Lüftungs-, Klima- und Sanitäranlagen
- Einschlägige Sicherheitsvorschriften der technischen Gebäudeausrüstung

2.3 Allgemeine technische Eigenschaften

2.3.1 Einbindung in TwinCAT

Entwicklungsumgebung	Zielplattform	Einzubindende TwinCAT SPS Bibliotheken	Einzubindende TwinCAT Standard SPS Bibliotheken
TwinCAT v2.10 ab Build 1302	PC oder CX (x86) CX (ARM)	TcHVAC.lib; TcFloatPC.lib; TcBaseMath.lib; TcMath.lib	TcBase.lib; TcIoFunctions.lib; TcPlcCoupler.lib; TcSystem.lib; TcUtilities.lib

2.3.2 Hardwareanforderungen

Die HLK-Bibliothek ist auf allen PC basierenden Hardware-Plattformen einsetzbar. Ideale Zielplattform für die Anwendungen der Heizungs-, Lüftungs-, Klima- und Sanitärtechnik sind die Embedded PC´s der Baureihe CX.

2.3.3 Remanente Daten

Die Steuerungen haben entweder ein NOVRAM und/oder einen Flash zur Speicherung von remanenten Daten.
Zur automatischen Speichern nach einer Parameteränderung werden die IN_OUT-Variablen auf Änderung Ihrer Werte überwacht. Bei Änderung wird eine bibliotheksinterne Variable angetriggert mit der die Funktionsbausteine FB_HVACNOVRAMDataHandling oder FB_HVACPersistentDataHandling aktiv werden.

Bei *eDataSecurityType:= eHVACDataSecurityType_Persistent* werden die IN_OUT-Variablen in einer Binärdatei im Flash gespeichert. Voraussetzung hierfür ist die einmalige Instanzierung des Funktionsbausteines FB_HVACPersistantDataHandling. Bei *eDataSecurityType:= eHVACDataSecurityType_Idle* ist das Schreiben der IN_OUT-Variablen deaktiviert.

Um sie dennoch remanent zu halten, müssen die Variablen lokiert bzw. adressiert werden. Damit werden Sie bei einer Wertänderung ins NOVRAM gespeichert. Voraussetzung hierfür ist die Instanzierung des Funktionsbausteins FB_HVACNOVRAMDataHandling, sowie eine Instanz des jeweiligen Datentyps, der gesichert werden soll.
Beispiel: Bei einer Variablen vom Typ BYTE muss eine Instanz vom FB_HVACNOVRAM_Byte verwendet werden.

HINWEIS

Zerstörung Flash

Bei eDataSecurityType:= eHVACDataSecurityType_Persistent darf eine als persistent deklarierte IN_OUT-Variable nicht zyklisch beschrieben werden, da sonst der Flash vorzeitig zerstört wird. Es ist dringend zu empfehlen zu diesem Thema auch die Dokumentation der Funktionsbausteine FB_HVACNOVRAMDataHandling oder FB_HVACPersistentDataHandling zu lesen und sich mit den in der Dokumentation enthaltenen Beispielen vertraut zu machen!

2.3.4 Voreinstellwerte

Für alle Anlagenparameter sind innerhalb der Funktionsbausteine Voreinstellwerte deklariert. Beim Neustart der Steuerung wird geprüft, ob im Flash oder im NOVRAM der Steuerung bereits Werte vorhanden sind. Sind Werte in der Binärdatei des Flash oder im NOVRAM vorhanden, dann werden diese im ersten Zyklus automatisch auf die IN_OUT-Variablen zurück geschrieben. Die Steuerung startet also automatisch mit den zuletzt remanent gespeicherten Daten.

Mit einer steigenden Flanke an der Eingangsvariablen *bSetDefault* können jederzeit die Voreinstellwerte aktiviert werden.

2.3.5 Wertebereichsüberwachung

Für jeden Eingangsparameter der Funktionsbausteine ist ein Wertebereich definiert. Bei der Eingabe eines Wertes außerhalb seines zulässigen Bereiches, wird vom Funktionsbaustein automatisch der zuletzt gültige Wert zurückgeschrieben. Der Versuch der Eingabe eines unzulässigen Parameterwertes wird mit einem TRUE an der Ausgangsvariablen *bInvalidParameter* signalisiert. Durch eine steigende Flanke an der Eingangsvariablen *bReset* kann die Variable *bInvalidParameter* wieder zurückgesetzt werden.

3 Funktionsbausteine

HLK Aktoren

Name	Beschreibung
FB_HVAC2PointActuator [▶ 20]	Ansteuerung von Zweipunktklappen oder -ventilen
FB_HVAC3PointActuator [▶ 22]	Ansteuerung von Dreipunktklappen oder -ventilen
FB_HVACCirculationPump [▶ 25]	Steuerung von Pumpen
FB_HVACCirculationPumpEx [▶ 28]	Steuerung von Pumpen, das Steuern des Ausgangs bPump wurde gegenüber FB_HVACCirculationPump geändert
FB_HVACMotor1Speed [▶ 40]	Steuerung von einstufigen Antrieben
FB_HVACMotor2Speed [▶ 44]	Steuerung von zweistufigen Antrieben
FB_HVACMotor3Speed [▶ 50]	Steuerung von dreistufigen Antrieben
FB_HVACMux8 [▶ 398]	wertet den FIFO-Speicher des FB_HVACRedundancyCtrlEx aus
FB_HVACRedundancyCtrl [▶ 57]	Steuerung von bis zu acht Aktoren in Abhängigkeit der Laufzeit; nutzt intern einen Timer zur Ermittlung der Laufzeit
FB_HVACRedundancyCtrlEx [▶ 60]	Steuerung von bis zu acht Aktoren in Abhängigkeit der Laufzeit; die Laufzeiten der Aktoren müssen als Stundenwerte von außen angelegt werden.

HLK Analogmodule

Name	Beschreibung
FB_HVACAnalogInput [▶ 62]	Erfassung analoger Eingangssignale
FB_HVACAnalogOutput [▶ 65]	Ansteuerung von stetigen Stellorganen
FB_HVACAnalogOutputEx [▶ 67]	Ansteuerung von stetigen Stellorgane mit integrierter Skalierungsfunktion
FB_HVACAnalogTo3Point [▶ 71]	Umwandlung eines analogen Stellsignals in ein Dreipunktschrittsignal
FB_HVACConfigureKL32xx [▶ 77]	Parametrierung des angeschlossenen Sensortyps an einem Eingangskanal aus der SPS heraus
FB_HVACScale [▶ 81]	Skalierungsbaustein
FB_HVACScale_nPoint [▶ 82]	Nachbildung von Kennlinien in der SPS
FB_HVACTemperatureCurve [▶ 86]	Innerhalb des FB sind unterschiedliche Temperaturkennlinien hinterlegt, die abgebildet werden können.
FB_HVACTemperatureSensor [▶ 91]	Erfassung von Temperaturwerten in 1/10 °C, abgestimmt auf die KL320x Busklemmen
FB_HVACTemperatureSensorEx [▶ 94]	Erfassung von Temperaturwerten in 1/10 °C, abgestimmt auf die KL320x Busklemmen; ohne Filter 2.Ordnung, dafür mit einer Glättungsfunktion
FB_HVACTemperatureSensorEx2 [▶ 98]	Erfassung von Temperaturwerten in 1/10 °C oder 1/100°C, abgestimmt auf die KL320x Busklemmen; ohne Filter 2.Ordnung, dafür mit einer Glättungsfunktion

HLK Regler

Name	Beschreibung
FB_HVAC2PointCtrl [▶ 252]	2-Punkt Regler
FB_HVAC2PointCtrlSequence [▶ 296]	2-Punkt Sequenz-Regler
FB_HVACBasicSequenceCtrl [▶ 304]	allgemeiner Sequenzregler
FB_HVACI_CtrlStep [▶ 255]	I-Übertagungsglied zur Stufensteuerung in Verbindung mit dem FB_HVACPowerRangeTable

Name	Beschreibung
FB_HVACI_CtrlStepEx [▶ 264]	I-Übertagungsglied zur Stufensteuerung in Verbindung mit dem FB_HVACPowerRangeTable
FB_HVACMasterSequenceCtrl [▶ 307]	Führungsregler in einer RLT-Anlage
FB_HVACPIDCooling [▶ 310]	PID-Regler Kühler
FB_HVACPIDCtrl [▶ 273]	PID-Regler
FB_HVACPIDCtrl_Ex [▶ 275]	PID-Regler (erweitert)
FB_HVACPIDDehumidify [▶ 313]	PID-Regler Entfeuchtung
FB_HVACPIDEnergyRecovery [▶ 316]	PID-Regler Wärmerückgewinnung
FB_HVACPIDHumidify [▶ 319]	PID-Regler Befeuchtung
FB_HVACPIDMixedAir [▶ 323]	PID-Regler Mischluftkammer
FB_HVACPIDPreHeating [▶ 326]	PID-Regler Vorerhitzer
FB_HVACPIDReHeating [▶ 329]	PID-Regler Nacherhitzer
FB_HVACPowerRangeTable [▶ 281]	Leistungsstufentabelle zur seqenziellen Regelung von Leistungsrerzeugern wie beispielsweise Heizkesseln oder Kältemaschinen

HLK Sollwertmodule

Name	Beschreibung
FB_HVACHeatingCurve [▶ 332]	Ermittlung der Vorlauftemperatur in Abhängigkeit der Außentemperatur über vier Stützpunkte
FB_HVACHeatingCurveEx [▶ 335]	Ermittlung der Vorlauftemperatur in Abhängigkeit der Außentemperatur
FB_HVACOutsideTempDamped [▶ 338]	Ermittlung der gedämpften Außentemperatur
FB_HVACSetpointHeating [▶ 339]	Steuerung eines Heizkreises mit unterschiedlichen Betriebsarten
FB_HVACSetpointRamp [▶ 341]	Sollwertrampe
FB_HVACSummerCompensation [▶ 344]	Anhebung des Raumsollwertes in Abhängigkeit der Außentemperatur

HLK Sonderfunktionen

Name	Beschreibung
FB_HVACAirConditioning2Speed [▶ 347]	Ansteuerung von RLT-Anlagen mit zweistufigen Ventilatoren
FB_HVACAlarm [▶ 350]	Alarmbaustein
FB_HVACAntiBlockingDamper [▶ 351]	Blockierschutz für Klappenantriebe
FB_HVACAntiBlockingPump [▶ 353]	Blockierschutz für Pumpen
FB_HVACBlink [▶ 357]	Blinksequenz
FB_HVACCmdCtrl_8 [▶ 358]	Mit dem Funktionsbaustein können einzelne Aggregate einer Anlage in einer bestimmten Reihenfolge sequentiell ein- bzw. ausgeschaltet werden. *FB_HVACCmdCtrl_8* kann als Startbaustein einer Lüftungsanlage eingesetzt werden.
FB_HVACCmdCtrlSystem1Stage [▶ 368]	Anlagenschalter einstufig
FB_HVACCmdCtrlSystem2Stage [▶ 376]	Anlagenschalter zweistufig
FB_HVACConvertEnum [▶ 390]	Konvertiert ein Enum in einen Integer-Wert und umgekehrt. Diese Konvertierung ist speziell für Enums geeignet, die als VAR_IN_OUT-Variablen an Funktionsbausteinen verwendet werden.
FB_HVACEnthalpy [▶ 391]	Ermittlung des Taupunktes, der spezifischen Enthalpie und der absoluten Feuchte.

Name	Beschreibung
FB_HVACFixedLimit [▶ 393]	Grenzwertschalter
FB_HVACFreezeProtectionHeater [▶ 395]	Frostschutzüberwachung
FB_HVACMUX_INT_16 [▶ 399]	Funktionsbaustein beinhaltet zwei verschiedene Typen von Multiplexern
FB_HVACMUX_INT_8 [▶ 403]	Funktionsbaustein beinhaltet zwei verschiedene Typen von Multiplexern
FB_HVACMUX_REAL_16 [▶ 407]	Funktionsbaustein beinhaltet zwei verschiedene Typen von Multiplexern
FB_HVACMUX_REAL_8 [▶ 407]	Funktionsbaustein beinhaltet zwei verschiedene Typen von Multiplexern
FB_HVACOverwriteAnalog [▶ 415]	analoge Übersteuerung in Handbetrieb
FB_HVACOverwriteDigital [▶ 415]	digitale Übersteuerung in Handbetrieb
FB_HVACPowerMeasurementKL3403 [▶ 416]	Erfassung der gemessenen/berechneten Werte der 3-Phasen-Leistungsmessklemme (KL3403)
FB_HVACPowerMeasurementKL3403Ex [▶ 419]	Im Vergleich zu dem FB_HVACPowerMeasurementKL3403 werden die Ergebnisse im LREAL Format ausgegeben. Die Ausgabe wurde um die Frequenzen der drei Phasen erweitert.
FB_HVACPriority_INT_16 [▶ 421]	Funktionsbaustein kann als Priorisierung von Ereignissen oder als Muliplexer eingesetzt werden
FB_HVACPriority_INT_8 [▶ 426]	Funktionsbaustein kann als Priorisierung von Ereignissen oder als Muliplexer eingesetzt werden
FB_HVACPriority_REAL_16 [▶ 430]	Funktionsbaustein kann als Priorisierung von Ereignissen oder als Muliplexer eingesetzt werden
FB_HVACPriority_REAL_8 [▶ 435]	Funktionsbaustein kann als Priorisierung von Ereignissen oder als Muliplexer eingesetzt werden
FB_HVACOptimizedOff [▶ 448]	Funktionsbaustein zum optimierten Ausschalten (Vorstopp) von Heizkesseln und Klimaanlagen in Verbindung mit Schaltzeitbausteinen.
FB_HVACOptimizedOn [▶ 439]	Funktionsbaustein zum optimierten Einschalten von Heizkesseln und Klimaanlagen in Verbindung mit Schaltzeitbausteinen.
FB_HVACTempChangeFunctionEntry [▶ 459]	Funktionsbaustein zur Eingabe der Stützstellen der Vorstartfunktion
FB_HVACPWM [▶ 461]	PWM-Baustein
FB_HVACStartAirConditioning [▶ 464]	Startprogramm einer RLT-Anlage
FB_HVACSummerNightCooling [▶ 468]	Sommernachtkühlung
FB_HVACSummerNightCoolingEx [▶ 471]	Sommernachtkühlung
FB_HVACTimeCon [▶ 478]	Konvertiert eine TIME-Variable in drei UDINT-Variablen (udiSec, udiMin, udiHour)
FB_HVACTimeConSec [▶ 479]	Konvertiert eine TIME-Variable in eine UDINT-Variablen (udiSec)
FB_HVACTimeConSecMs [▶ 479]	Konvertiert eine TIME-Variable in zwei UDINT-Variablen (udiMs, udiSec)
FB_HVACWork [▶ 480]	Betriebsstundenerfassung

HLK Uhren

Name	Beschreibung
FB_HVACScheduler1ch [▶ 482]	Wochenzeitschaltuhr mit 1 Zeitkanal
FB_HVACScheduler7ch [▶ 485]	Wochenzeitschaltuhr mit 7 Zeitkanälen

Name	Beschreibung
FB_HVACScheduler7TCHandling [▶ 489]	FB mit dem eine einzelne Zeile in dem Datenarray der Zeitschaltuhr parametriert werden kann.
FB_HVACScheduler28ch [▶ 490]	Wochenzeitschaltuhr mit 28 Zeitkanälen
FB_HVACScheduler28TCHandling [▶ 494]	FB mit dem eine einzelne Zeile in dem Datenarray der Zeitschaltuhr parametriert werden kann.
FB_HVACSchedulerSpecialPeriods [▶ 495]	Jahreszeitschaltplaner mit Tag, Monat und genauer Uhrzeitvorgabe
FB_HVACSchedulerPublicHolidays [▶ 498]	Jahreszeitschaltplaner mit Tag und Monat

HLK System

Name	Beschreibung
FB_HVACGetSystemTime [▶ 500]	Realisierung einer internen Uhr in der TwinCAT SPS
FB_HVACNOVRAMDataHandling [▶ 502]	Daten werden spannungsausfallsicher ins NOVRAM geschrieben
FB_HVACPersistentDataHandling [▶ 505]	Daten werden spannungsausfallsicher in eine Datei geschrieben
FB_HVACPersistentDataFileCopy [▶ 507]	kopieren von Binärdateien zw. lokalen- oder remote-TwinCAT PC auf den lokalen TwinCAT PC
FB_HVACSetLocalTime [▶ 508]	Setzt die lokale Windows-Systemzeit und das Datum eines TwinCAT-Systems
FB_HVACSystemTaskInfo [▶ 510]	Ermittlung von Systemvariablen der Task

HLK Backup Bausteine

Name	Beschreibung
FB_HVACNOVRAM_xyz [▶ 512]	FBs für Standard Datentypen
FB_HVACPersistent_xyz [▶ 517]	FBs für Standard Datentypen

Raumfunktionen Beleuchtung

Name	Beschreibung
FB_BARLightActuator [▶ 134]	Ausgabe eines vorgegebenen prozentualen Dimmwertes über eine Rampenfunktion. Die Ausgabe erfolgt wahlweise in Prozent, INTEGER oder BOOL. Dieser Baustein umfasst ebenfalls einen Lichtszenenspeicher von 21 einstellbaren Dimmwerten.
FB_BARLightCircuit [▶ 137]	Einfacher Lichtschaltkreis ohne Dimmfunktionalität
FB_BARLightCircuitDim [▶ 138]	Einfacher Lichtschaltkreis mit Dimmfunktionalität
FB_BARAutomaticLight [▶ 120]	Präsenzgesteuertes Automatiklicht mit Ausschaltverzögerung
FB_BARStairwellAutomatic [▶ 142]	Treppenhausbeleuchtung mit Vorwarnsequenz
FB_BARTwilightAutomatic [▶ 145]	Dämmerungsschaltung
FB_BARDaylightControl [▶ 130]	Tageslichtsteuerung ohne Dimmvorgänge
FB_BARConstantLightControl [▶ 123]	Konstantlichtregelung mit kontinuierlicher Ausgabe von Analogwerten

Raumfunktionen Verschattung (siehe auch Übersicht)

Übersicht [▶ 148]

Name	Beschreibung
FB_BARBlindPositionEntry [▶ 164]	Sonnenschutzfunktion: Eingabe von Jalousiepositionen.
FB_BARSunblindEvent [▶ 201]	Ausgabe eines vorgegebenen prozentualen Jalousieposition und -stellung

Name	Beschreibung
FB_BARSunblindWeatherProtection [▶ 223]	Witterungsschutz-Funktion
FB_BARSunblindSwitch [▶ 211]	Handbedienung
FB_BARSunblindScene [▶ 207]	Handbedienung mit Szenenanwahl und -programmierung
FB_BARSunblindTwilightAutomatic [▶ 221]	Dämmerungsautomatik
FB_BARSunblindThermoAutomatic [▶ 214]	Thermoautomatik
FB_BARSunProtectionEx [▶ 226]	Sonnenschutzfunktion, siehe Übersicht Sonnenschutzautomatik (Verschattungskorrektur) [▶ 150]
FB_BARShadingObjectsEntry [▶ 186]	Verschattungskorrektur: Eingabe von Verschattungsobjekten per Baustein.
FB_BARReadShadingObjectsList [▶ 177]	Verschattungskorrektur: Eingabe von Verschattungsobjekten per Datenliste (csv).
FB_BARFacadeElementEntry [▶ 168]	Verschattungskorrektur: Eingabe von Fassadenelementen per Baustein.
FB_BARReadFacadeElementList [▶ 172]	Verschattungskorrektur: Eingabe von Fassadenelementen per Datenliste (csv).
FB_BARShadingCorrection [▶ 180] / FB_BARShadingCorrectionSouth [▶ 183]	Verschattungskorrektur-Baustein (FB_BarShadingCorrectionSouth ist für die Südhalbkugel gültig)
FB_BARDelayedHysteresis [▶ 167]	Überprüfung Außenhelligkeit mit Verzögerung und Hysterese
FB_BARWithinRangeAzimuth [▶ 229]	Überprüfung gültiger Sonnenstand Sonnenrichtungsbereich (Azimuthwinkel)
FB_BARWithinRangeElevation [▶ 232]	Überprüfung gültiger Sonnenstand Sonnenhöhenbereich (Elevationswinkel)
FB_BARSunblindPrioritySwitch [▶ 203]	Prioritätssteuerung
FB_BARSunblindActuator [▶ 189] / FB_BARSunblindActuatorEx [▶ 194]	Jalousie-Aktor
FB_BARSMISunblindActuator [▶ 200]	SMI Jalousie-Aktor
FB_BARRollerBlind [▶ 204]	Rollladen-Aktor
FB_BARSMIRollerBlind [▶ 206]	SMI Rollladen-Aktor

Raumfunktionen Regler

Name	Beschreibung
FB_BARPICtrl [▶ 115]	Einfacher PI-Regler mit Eingabe über das Proportionalband

Raumfunktionen Klimatisierung

Name	Beschreibung
FB_BAREnergyLevel [▶ 102]	Funktionsbaustein dient der Anpassung der Energieabgabe an die Nutzung des Gebäudes.
FB_BARFanCoil [▶ 104]	Funktionsbaustein bildet einen 3-stufigen Ventilator mit der entsprechenden Schalthysterese ab.
FB_BARFctSelection [▶ 107]	Funktionsbaustein dient zur Freigabe einer Raumheizung oder Raumkühlung.
FB_BARSetpointRoom [▶ 110]	Funktionsbaustein weist den vier Energieniveaus jeweils einen Sollwert für Kühlbetrieb und Heizbetreib zu.

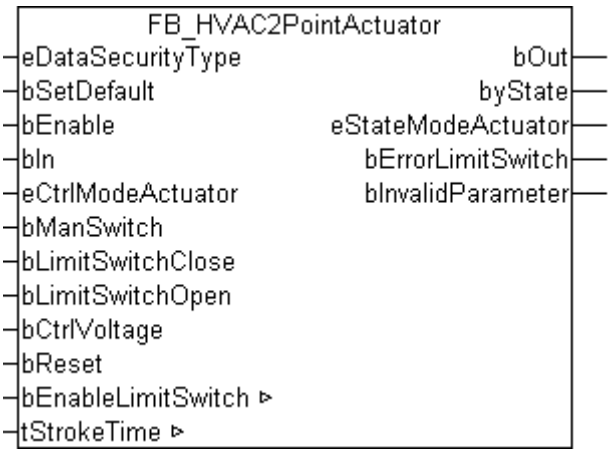
Übersicht Bibliothek

Datum	Version	Erzeugt mit TwinCAT Version	Bemerkungen
29.10.2008	1.0.0	V2.10.0 (Build 1328)	first Release

Datum	Version	Erzeugt mit TwinCAT Version	Bemerkungen
29.10.2009	1.1.0	V2.11.0 (Build 1536)	new FBs (FB_HVACRedundancyCtrlEx; FB_HVACTemperatureSensorEx; FB_HVACEnthalpy; FB_HVACTimeCon; FB_HVACTimeConSec; FB_HVACMux8)
12.04.2010	1.2.7	V2.11.0 (Build 1539)	new FBs (FB_HVACSetLocalTime; FB_HVACAnalogOutputEx; FB_HVACConfigureKL32xx; FB_HVACScale_nPoint; FB_HVACTemperatureCurve) new Function F_HVACRoundLREAL_EX
04.08.2010	1.3.0	V2.11.0 (Build 1547)	new FBs (FB_HVAC2PointCtrlSequence; FB_HVACPowerMeasurementKL3403Ex; FB_HVACScheduler7TCHandling; FB_HVACScheduler28TCHandling)
13.01.2011	1.9.0	V2.11.0 (Build 1552)	new FBs (FB_HVACTimeConSecMs; FB_HVACI_CtrlStep; FB_HVACPowerRangeTable; FB_HVACPriority_REAL_8; FB_HVACPriority_REAL_16; FB_HVACPriority_INT_16; FB_HVACPriority_INT_8; FB_HVACMUX_INT_8; FB_HVACMUX_INT_16; FB_HVACMUX__16; FB_HVACMUX_INT_8; FB_HVACCirculationPumpEx; FB_HVACHeatingCurveEx)
08.02.2011	1.10.1	V2.11.0 (Build 1552)	new FB (FB_HVACConvertEnum)
16.08.2011	1.11.1	V2.11.0 (Build 1552)	new FB (FB_HVACSummernightCoolingEx)
28.12.2011	1.12.0	V2.11.0 (Build 1552)	neu FB_HVACCmdCtrl_8
30.03.2012	1.13.0	V2.11.0 (Build 2218)	mit den FBs für die Raumfunktionen
30.06.2012	1.14.0	V2.11.0 (Build 2224)	new FB_HVACI_CtrlStepEx

3.1 HLK Aktoren

3.1.1 FB_HVAC2PointActuator



Anwendung

Dieser Funktionsbaustein dient der Ansteuerung von Zweipunktventilen oder Zweipunktklappen.

VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Mit der Eingangsvariablen *bEnable*wird der Baustein vom SPS-Programm frei gegeben. Ohne Freigabe des Bausteins bleibt das Stellorgan immer geschlossen. Der Ausgang *bOut* ist dauerhaft FALSE.

bIn: Im Automatikbetrieb wird das Stellorgan über ein FALSE zugefahren und über ein TRUE aufgefahren.

eCtrlModeActuator: Enum, das die Betriebsart festlegt.

bManSwitch: Besitzt der Zweipunktantrieb einen Hand-/Not-Schalter im Schaltschrank, dann kann dieser an den Eingang *bManSwitch* angeschlossen werden, somit wird der Status vom Hand-/Not-Schalter überwacht. Bei *bManSwitch*= FALSE wird der Ausgang *bOut* des Antriebes auf FALSE gesetzt.

bLimitSwitchClose: Stellorganrückmeldung TRUE, wenn das Stellorgan komplett geschlossen ist.

bLimitSwitchOpen: Stellorganrückmeldung TRUE, wenn das Stellorgan komplett geöffnet ist.

bCtrlVoltage: Der Parameter *bCtrlVoltage* dient zur Kontrolle der Steuerspannung. Die Steuerspannung steht an, wenn die Variable *bCtrlVoltage*TRUE ist. Bei einem Ausfall der Steuerspannung wird die Feedbackkontrolle unterdrückt.

bReset: Quittierungseingang bei einer Störung.

VAR_OUTPUT

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

bOut: An diesem Ausgang wird das Stellorgan angeschlossen (FALSE= Zufahren; TRUE= Auffahren).

byState: Zeigt den Status der Ansteuerung vom Stellorgan an:
byState.0:= Enable
byState.1:= Manual Switch
byState.2:= Enable Feedback Control
byState.3:= Control Voltage
byState.4:= Reset
byState.5:= bOut

eStateModeActuator: Zeigt, in welchem Betriebsmodus das Stellorgan ist.

bErrorLimitSwitch: Wird TRUE, wenn kein Endschalter nach der eingestellten Verfahrzeit ausgelöst hat. *bErrorLimitSwitch* wird mit einer positiven Flanke am Eingang *bReset*quittiert.

bInvalidParameter: TRUE, wenn bei der Plausibilitätsüberprüfung ein Fehler aufgetreten ist. Die Meldung muss mit *bReset* quittiert werden.

VAR_IN_OUT

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

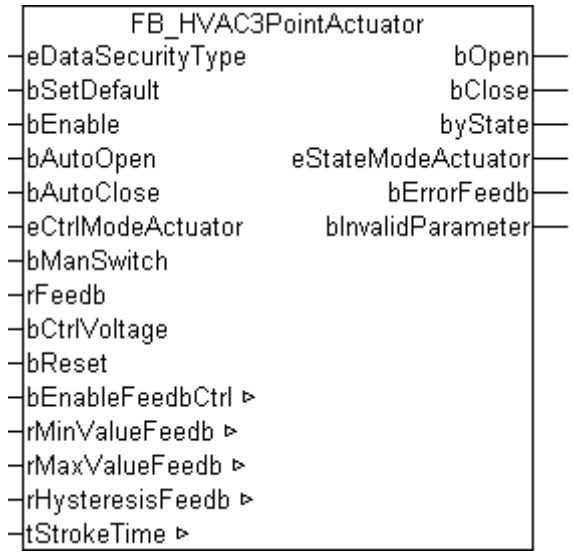
bEnableLimitSwitch: Ist der Eingang TRUE, dann ist die Funktionskontrolle des Antriebs mittels der Endlagenschalter aktiviert.

tStrokeTime: Hier muss zur korrekten Einstellung der Funktionskontrolle die Verfahrzeit des Antriebes vom ganz geschlossenen bis zum vollständig geöffneten Antrieb angegeben werden (0s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 200s.

Dokumente hierzu

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

3.1.2 FB_HVAC3PointActuator



Anwendung

Dieser Funktionsbaustein dient zur Ansteuerung von Dreipunktventilen oder Dreipunktklappen mit oder ohne stetiger Stellungsrückmeldung.
Häufige Anwendung findet der Baustein in Verbindung mit dem Funktionsbaustein FB_HVACAnalogTo3Point. [▶ 71]

VAR_INPUT

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

```
bAutoOpen           : BOOL;
bAutoClose          : BOOL;
eCtrlModeActuator   : E_HVAC3PointActuatorMode;
bManSwitch          : BOOL;
rFeedb              : REAL;
bCtrlVoltage        : BOOL;
bReset              : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Mit der Eingangsvariablen *bEnable*wird der Baustein vom SPS-Programm frei gegeben. Ohne Freigabe des Bausteins bleibt das Dreipunktstellorgan immer ganz geschlossen. Der Ausgang *bClose* ist dauerhaft TRUE.

bAutoOpen / bAutoClose: Im Automatikbetrieb wird der Dreipunktantrieb von den Eingangsvariablen *bAutoClose* und *bAutoOpen*gesteuert.

eCtrlModeActuator: Enum, das die Betriebsart festlegt.

bManSwitch: Besitzt der Dreipunktantrieb einen Hand-/Not-Schalter im Schaltschrank, dann kann dieser an den Eingang *bManSwitch* angeschlossen werden, somit wird der Status vom Hand-/Not-Schalter überwacht. Bei *bManSwitch*= FALSE wird der Ausgang *bOut* des Antriebes auf FALSE gesetzt.

rFeedb: Analoge Stellungsrückmeldung vom Stellorgan (0%..100%).

bCtrlVoltage: Parameter zur Kontrolle der Steuerspannung. Die Steuerspannung steht an, wenn die Variable *bCtrlVoltage*TRUE ist. Bei einem Ausfall der Steuerspannung wird die Feedbackkontrolle unterdrückt.

bReset: Quittierungseingang bei einer Störung.

VAR_OUTPUT

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

bOpen: An diesem Ausgang wird das Signal zum Auffahren des Dreipunktantriebes angeschlossen.

bClose: An diesem Ausgang wird das Signal zum Zufahren des Dreipunktantriebes angeschlossen.

byState: Zeigt den Status der Ansteuerung vom Stellorgan an:
byState.0:= Enable
byState.1:= Manual Switch

byState.2:= Enable Feedback Control
byState.3:= Control Voltage
byState.4:= Reset

eStateModeActuator: Zeigt in welchem Betriebsmodus das Stellorgan ist.

bErrorFeedb: Beim Setzen des Ausgangs *bClose*muss sich die Position des Antriebs innerhalb der Zeit *tStrokeTime* mindestens um den Betrag von *rHysteresisFeedb*verringert haben.
Beim Setzen des Ausgangs *bOpen*muss sich die Position des Antriebs innerhalb der Zeit *tStrokeTime* mindestens um den Betrag von *rHysteresisFeedb* erhöht haben.
Ist die tatsächliche Position nach einem Verfahrbefehl innerhalb der vorgegebenen Zeit nicht innerhalb des Toleranzbereichs, dann wird dieses am Ausgang *bErrorFeedb* mit TRUE signalisiert. Beide Ausgänge *bOpen*und *bClose*werden FALSE**.** Die Störung wird mit einer positiven Flanke am Eingang *bReset*quittiert.

bInvalidParameter: TRUE, wenn bei der Plausibilitätsüberprüfung ein Fehler aufgetreten ist. Die Meldung muss mit *bReset* quittiert werden.

VAR_IN_OUT

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

bEnableFeedbCtrl: Ist der Eingang TRUE, dann ist die Kontrolle des Rückführungssignals aktiviert. Die Variable wird persistent gespeichert.

rMinValueFeedb: Dient zur Skalierung der analogen Stellungsrückmeldung. rMinValueFeedb beinhaltet den Wert des Analogsignals bei komplett geschlossenem Stellorgan (0%..100%). Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rMaxValueFeedb: Dient zur Skalierung der analogen Stellungsrückmeldung. rMaxValueFeedb beinhaltet den Wert des Analogsignals bei komplett geöffnetem Stellorgan (0%..100%). Die Variable wird persistent gespeichert. Voreingestellt auf 0.

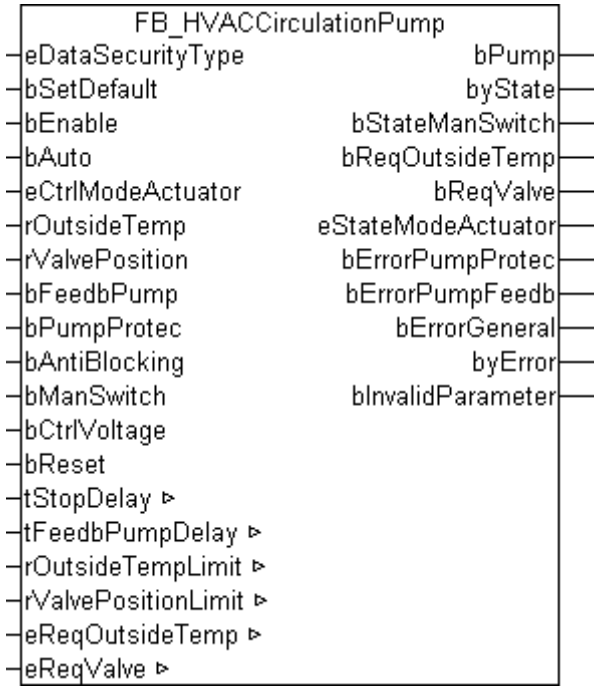
rHysteresisFeedb: Bedingt durch die Verfahrzeit des Antriebes, ist bei einem Sprung des Positionssollwertes die Stellungsrückführung immer nacheilend. Mit der Variablen *rHysteresisFeedbCtrl* wird ein Bereich festgelegt, innerhalb dessen der Positionssollwert des Stellorgans vom Rückführsignal abweichen kann, ohne dass die Feedbackkontrolle (bErrorFeedb) anspricht (0%..100%). Die Variable wird persistent gespeichert. Voreingestellt auf 10.

tStrokeTime: Durch das Nacheilen der tatsächlichen Position in Bezug auf die Sollposition wird das Ansprechen der Feedbackkontrolle bei einer Überschreitung der maximal zulässigen Differenz durch die Zeitvariable *tStrokeTime* [s] verzögert. Wenn das Stellorgan ganz geschlossen und einen Sollwertsprung auf 100% erhält, sollte als Zeit mindestens die Verfahrzeit des Antriebes über seinen gesamten Verfahrweg eingegeben werden (0s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 200s.

Dokumente hierzu

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

3.1.3 FB_HVACCirculationPump



Anwendung

Dieser Funktionsbaustein dient zur Steuerung von Pumpen in der HLK-Technik.

VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Freigabe des Bausteines, wenn *bEnable* = TRUE ist.

bAuto: Eingang für die Anforderung aus dem Automatikprogramm. Die Anforderung wird vom *eCtrlModeActuator* übersteuert.

eCtrlModeActuator: Enum, das die Betriebsart festlegt.

rOutsideTemp: Eingang zur Übergabe des Außentemperaturwertes.

rValvePosition: Befindet sich in dem hydraulischen Kreis der Pumpe ein Regelventil, dann muss hier die Stellung des Regelventils angelegt werden.

bFeedbPump: Dieser Eingang dient zur Anzeige der Betriebsmeldung in einem Visualisierungssystem und zur Überwachung der Pumpenfunktion.

bPumpProtec: Am Eingang *bPumpProtec* wird eine Störmeldung der Pumpe angeschlossen. Eine Störung der Pumpe steht an, wenn der Eingang *bPumpProtec* FALSE ist. Bei einer anstehenden Störung wird der Ausgang *bPump* FALSE. Ein erneuter Anlauf der Pumpe ist nur nach einer Quittierung am Eingang *bReset* möglich.

bAntiBlocking: Eingang zur Übergabe der Antiblockieranforderung, d.h. bei TRUE ist die Anforderung aktiv.

bManSwitch: Besitzt die Pumpe im Schaltschrank einen Hand- / Notschalter, dann kann dieser an den Eingang *bManSwitch* angeschlossen werden, und somit wird der Status des Hand- / Notschalters überwacht. Bei *bManSwitch*= FALSE wird der Ausgang *bPump* gesperrt. Der Ausgang *bPump* kann nur eingeschaltet werden, wenn *bManSwitch* = TRUE ist (Ruhestromprinzip).

bCtrlVoltage: Zur Meldeschauerunterdrückung wird die Störmeldung von *bPumpProtec* nur erfasst, wenn der Eingang *bCtrlVoltage* TRUE ist.

bReset: Quittierungseingang bei einer Störung.

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: Ausgangsvariable zur Ansteuerung einer Pumpe.

byState: Ausgabe des Pumpenstatus als 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: Statusmeldung des Hand/Not-Schalter. Ein TRUE signalisiert, dass die Hand/Not-Bedienebene aktiviert ist.

bReqOutsideTemp: Falls die Bedingung zur Anforderung der Pumpe in Abhängigkeit der Außentemperatur TRUE ist, wird die Ausgangsvariable *bReqOutsideTemp* TRUE.

bReqValve: Falls die Bedingung zur Anforderung der Pumpe in Abhängigkeit der Ventilstellung erreicht ist, wird die Variable *bReqValve* TRUE.

eStateModeActuator: Zeigt, in welchem Betriebsmodus das Stellorgan ist.

bErrorPumpProtec: Fehler von der Pumpe.

bErrorPumpFeedb: Wenn der Eingang *bFeedbPump* nach dem Setzen des Ausganges *bPump* nicht innerhalb der Zeit von *tFeedbPumpDelay* (tFeedbPumpDelay muss > t#0s sein) auf TRUE geht, dann wird dieses als Störung erkannt und dieser Ausgang wird auf TRUE gesetzt und der Ausgang *bPump* wird auf FALSE gesetzt. Diese Fehlermeldung muss mit *bReset* quittiert werden.

bErrorGeneral: Es liegt allgemein ein Fehler an.

byError: Ausgabe der Fehler als Byte.
byError.1 := bInvalidParameter
byError.2 := bErrorGeneral
byError.3 := bErrorPumpProtec
byError.4 := bErrorPumpFeedb

bInvalidParameter: TRUE, wenn bei der Plausibilitätsüberprüfung ein Fehler aufgetreten ist. Die Meldung muss mit *bReset* quittiert werden.

VAR_IN_OUT

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

tStopDelay: Mit der Zeit *tStopDelay* [s] wird das Abschalten der Pumpe nach dem Entfall der Einschaltbedingungen verzögert. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

tFeedbPumpDelay: Die Überwachungsfunktion der Pumpenbetriebsrückmeldung [s] ist nur aktiv, wenn *tFeedbPumpDelay* > t#0s ist. Bei *tFeedbPumpDelay* = t#0s ist die Überwachungsfunktion deaktiviert (0s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

rOutsideTempLimit: Wert [°C] oberhalb bzw. unterhalb dessen die Pumpe in Abhängigkeit der Außentemperatur ein- bzw. ausgeschaltet wird (-60°c..60°C). Die Variable wird persistent gespeichert. Voreingestellt auf 10°C.

rValvePositionLimit: Schwellwert für die Stellung eines der Pumpe zugehörigen Regelventils, ab dem die Pumpe automatisch einschalten soll. z.B. Erhitzerpumpe (0%..100%). Die Variable wird persistent gespeichert. Voreingestellt auf 3%.

eReqOutsideTemp: In Abhängigkeit der Außentemperatur kann die Pumpe z.B. zu Frostschutzzwecken bei der Unterschreitung des Temperaturgrenzwertes *rOutsideTempLimit* zwangsweise eingeschaltet werden. Voraussetzung ist, dass *bEnable* = TRUE ist und die Pumpe im Automatikbetrieb ist. Die Variable wird persistent gespeichert.

HINWEIS

Hand-Aus übersteuert die Frostschutzfunktion!

eReqValve: In Abhängigkeit der Stellung des zur Pumpe gehörigen Ventils kann die Pumpe bei Überschreitung des Schwellwertes von *rValvePositionLimit* eingeschaltet werden. Die Aktivierung der Einschaltung über die Ventilstellung erfolgt über das ENUM. Außerdem wird mit dem ENUM bestimmt, ob für die Kombination der temperatur- und ventilstellungsabhängigen Einschaltbedingungen eine ODER - bzw. UND - VERKNÜPFUNG gilt.

Eine Übersicht aller möglichen Kombinationen zeigt die folgende Tabelle:

eReqOutsideTemp	eReqValve	Funktion	Anwendung
OTLowerLimit	NoRequest	outsidetemp lower limit	
OTLowerLimit	OrValvePosHigherLimit	outsidetemp OR valve higher limit	Heizkreis, Lufterhitzer
OTLowerLimit	AndValvePosHigherLimit	outsidetemp AND valve higher limit	

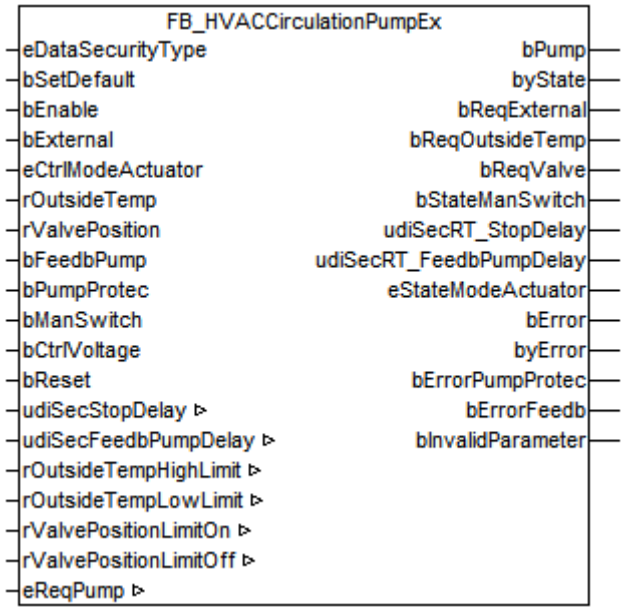
eReqOutsideTemp	eReqValve	Funktion	Anwendung
OTHigherLimit	NoRequest	outsidetemp higher limit	
OTHigherLimit	OrValvePosHigherLimit	outsidetemp OR valve higher limit	
OTHigherLimit	AndValvePosHigherLimit	outsidetemp AND valve higher limit	Kühlerpumpe
NoRequest	NoRequest	no request	Primärpumpe
NoRequest	OrValvePosHigherLimit	valve higher limit	
NoRequest	AndValvePosHigherLimit	not valid	

Durch die Kombinationen der beiden Variablen *eReqOutsideTemp* und *eReqValve*kann dieser Funktionsbaustein den Erfordernissen eines Heizkreises, eines Lufterhitzers, eines Luftkühler oder einer Zubringerpumpe angepasst werden.

Dokumente hierzu

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

3.1.4 FB_HVACCirculationPumpEx



Anwendung

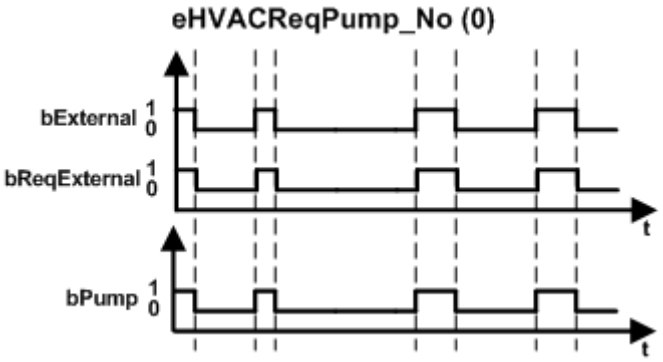
Dieser Funktionsbaustein dient zur Steuerung von Pumpen in der HLK-Technik.

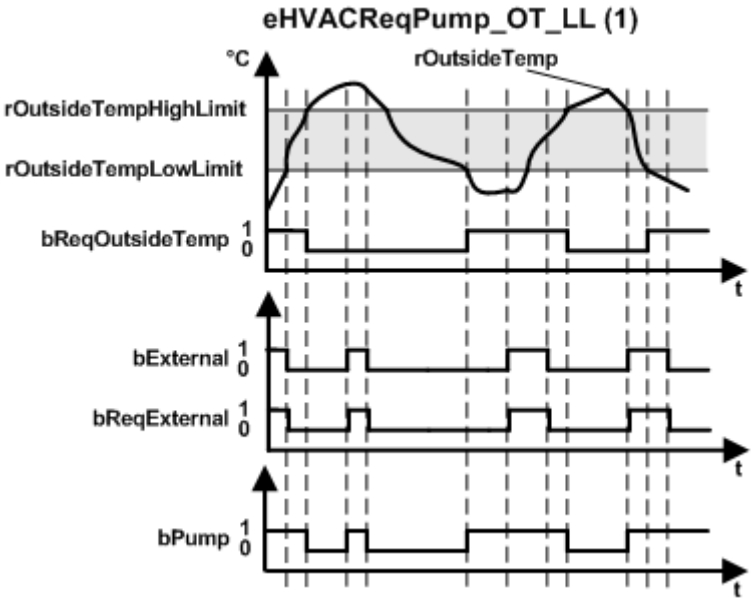
Es müssen zur Ansteuerung der Pumpe über *bPump* vorab die folgenden Einschaltbedingungen erfüllt sein:

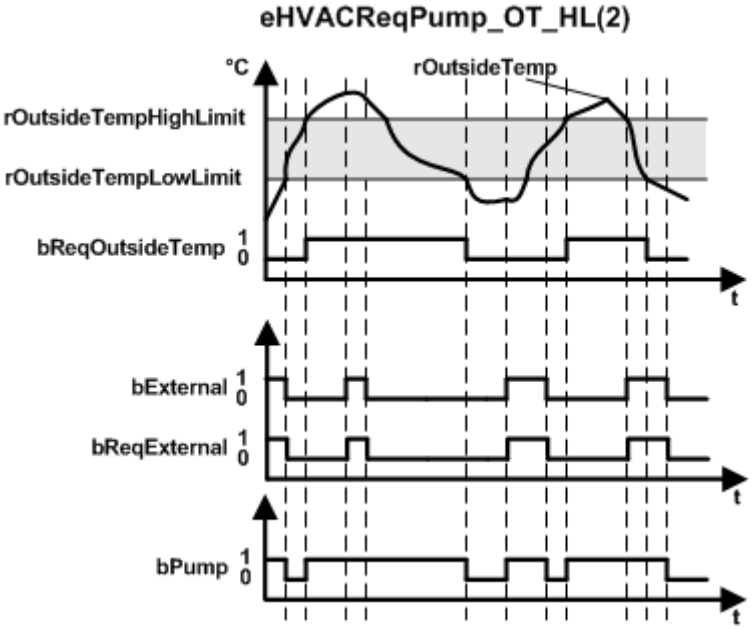
bEnable = TRUEAND*bErrorPumpProtec* = FALSEAND*bManSwitch* = TRUEAND*bCtrlVoltage* = TRUE

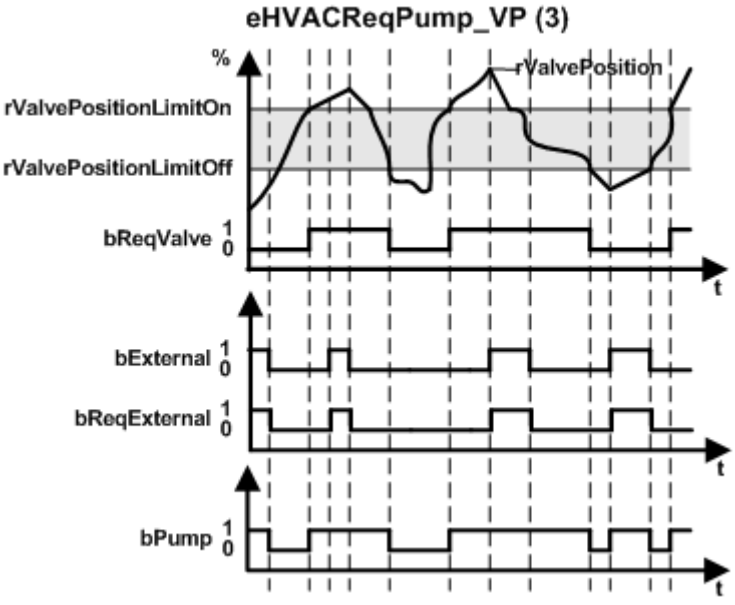
Ist eine der Einschaltbedingungen nicht erfüllt, so ist der Ausgang *bPump* konstant FALSE.

Anhand der folgenden Diagramme ist zu sehen, wie der Ausgang *bPump* in Abhängigkeit des Enums *eReqPump*, des Eingangs *bExternal*, der Außentemperatur *rOutSideTemp* und der Ventilstellung *rValvePosition* gesteuert wird. Die oben erwähnten Einschaltbedingungen müssen dazu erfüllt sein und eine der beiden Betriebsarten *eHVACActuatorMode_Auto_BMS* oder *eHVACActuatorMode_Auto_OP* vorgewählt sein.

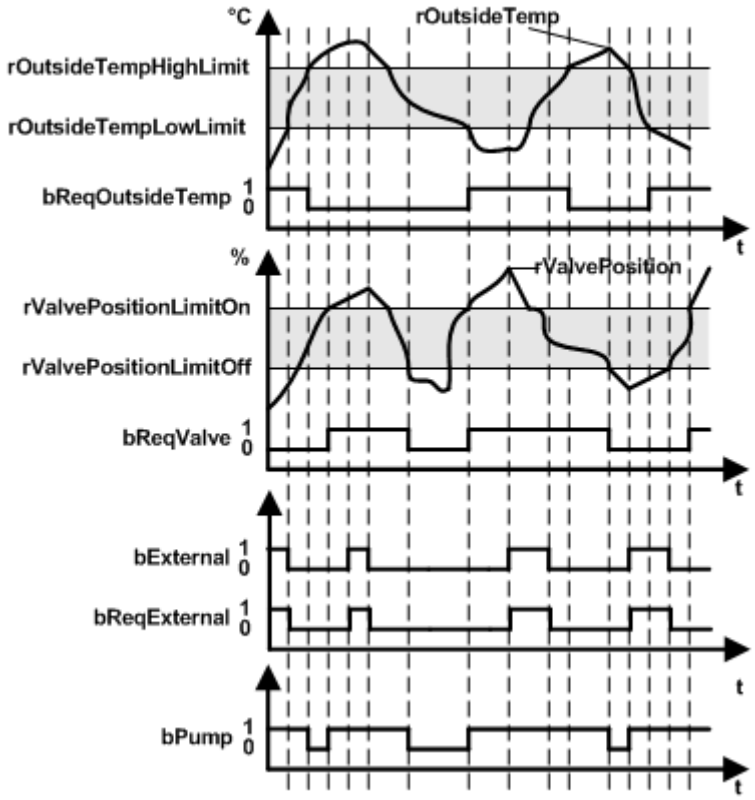

eHVACReqPump_No (0)
bExternal
1
0
bReqExternal
1
0
t
bPump
1
0
t


eHVACReqPump_OT_LL (1)
°C
rOutsideTemp
rOutsideTempHighLimit
rOutsideTempLowLimit
bReqOutsideTemp
bExternal
bReqExternal
bPump
1
0
t

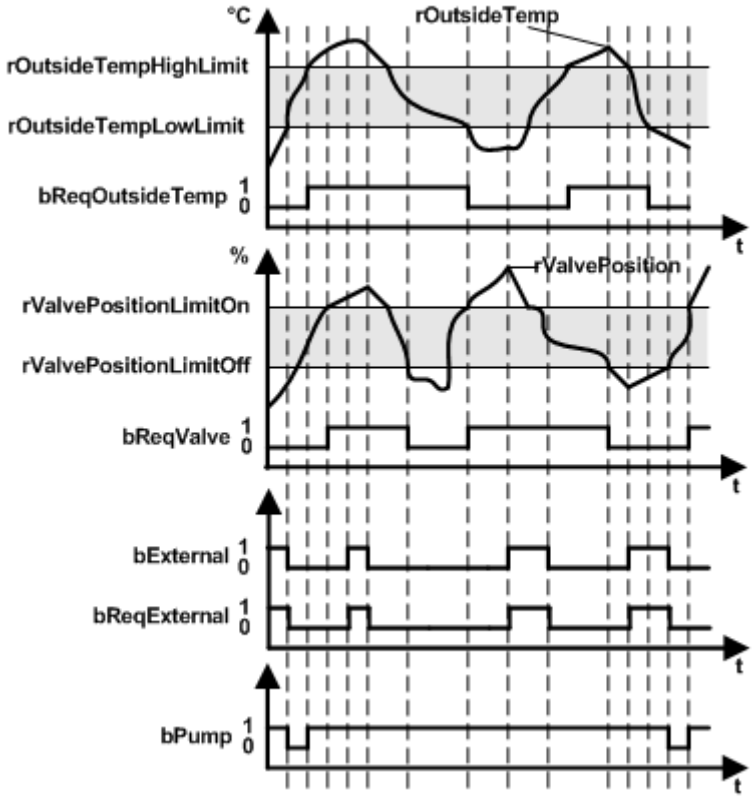

eHVACReqPump_OT_HL(2)
°C
rOutsideTemp
rOutsideTempHighLimit
rOutsideTempLowLimit
bReqOutsideTemp
bExternal
bReqExternal
bPump
1
0
t


eHVACReqPump_VP (3)
%
rValvePosition
rValvePositionLimitOn
rValvePositionLimitOff
bReqValve
bExternal
bReqExternal
bPump
1
0
t

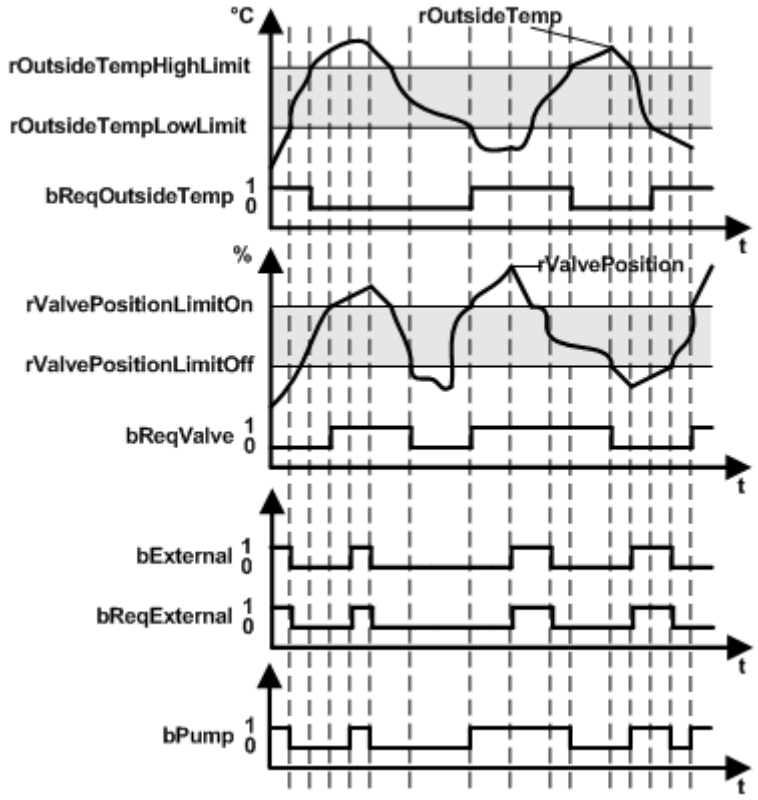
eHVACReqPump_OT_LL_OR_VP (4)



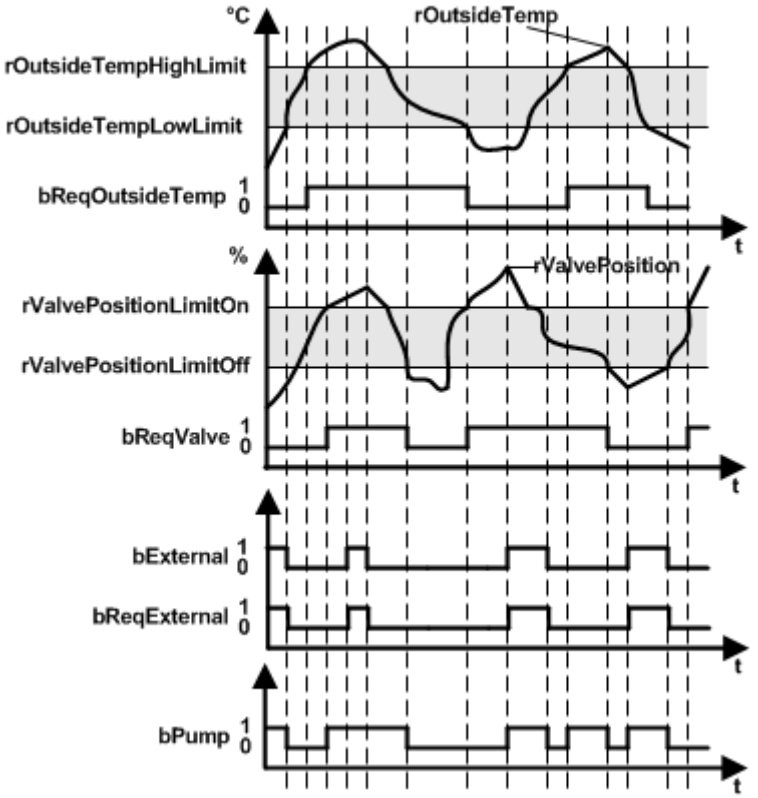
eHVACReqPump_OT_HL_OR_VP (5)



eHVACReqPump_OT_LL_AND_VP (6)



eHVACReqPump_OT_HL_AND_VP (7)



eHVACReqPump_No: Es besteht keine Anforderung zur Steuerung der Pumpe seitens des Enums

eHVACReqPump_OT_LL: Die Außentemperatur (OT = *rOutsideTemp*) muss kleiner sein als *rOutsideTempLowLimit* (LL = Lower Limit)

eHVACReqPump_OT_HL: Die Außentemperatur (OT = *rOutsideTemp*) muss größer sein als *rOutsideTempHighLimit* (HL = Higher Limit)

eHVACReqPump_VP: Die Ventilstellung (VP = *rValvePosition*) muss größer sein als *rValvePositionLimitOn*

eHVACReqPump_OT_LL_OR_VP: Die Außentemperatur (OT = *rOutsideTemp*) muss kleiner sein als *rOutsideTempLowLimit* (LL = Lower Limit) ODER die Ventilstellung (VP = *rValvePosition*) muss größer sein als *rValvePositionLimitOn*

eHVACReqPump_OT_HL_OR_VP: Die Außentemperatur (OT = *rOutsideTemp*) muss größer sein als *rOutsideTempHighLimit* (HL = Higher Limit) ODER die Ventilstellung (VP = *rValvePosition*) muss größer sein als *rValvePositionLimitOn*

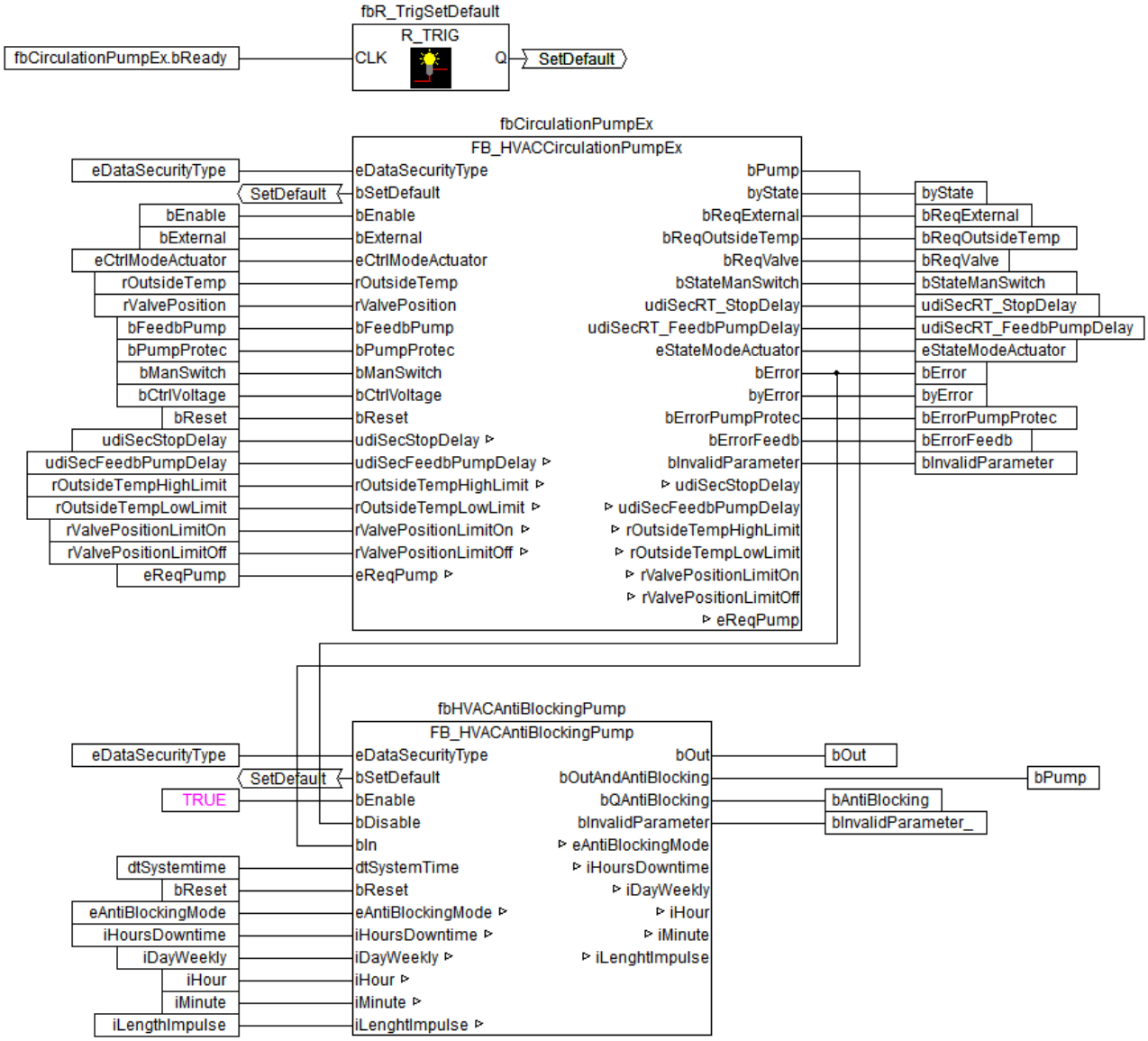
eHVACReqPump_OT_LL_AND_VP: Die Außentemperatur (OT = *rOutsideTemp*) muss kleiner sein als *rOutsideTempLowLimit* (LL = Lower Limit) UND die Ventilstellung (VP = *rValvePosition*) muss größer sein als *rValvePositionLimitOn*

eHVACReqPump_OT_HL_AND_VP: Die Außentemperatur (OT = *rOutsideTemp*) muss größer sein als *rOutsideTempHighLimit* (HL = Higher Limit) UND die Ventilstellung (VP = *rValvePosition*) muss größer sein als *rValvePositionLimitOn*

Der Ausgang *bPump* schaltet sich nach dem Entfall der Einschaltbedingungen in Abhängigkeit des Enums *eReqPump*, des Eingangs *bExternal*, der Außentemperatur *rOutSideTemp* oder der Ventilstellung *rValvePosition* verzögert nach Ablauf der Zeitangabe *udiSecStopDelay* aus.

bError wird TRUE, wenn bErrorPumpProtecTRUE ist. Zur Abschaltung des Ausgangs bPump führt aber nur die Störung bErrorPumpProtec. Soll die Störmeldung bErrorFeedb ebenfalls zur Abschaltung der Pumpe führen, so muss die Variable nach dem Aufruf des Funktionsbausteins mit dem Ausgang bPumpAND-verknüpft werden. Die Störmeldung bErrorFeedb ist nur in der Betriebsart eHVACActuatorMode_Auto_BMSOReHVACActuatorMode_Auto_OP aktiv und wenn die Zeitangabe udiSecFeedbPumpDelay größer 0 ist.

Beispiel Antiblockierschutz



Anwendungsbeispiel

Das Anwendungsbeispiel zeigt den Funktionsbaustein FB_HVACirculationPumpEx. Die Darstellung des Beispiels gibt es in den Programmiersprachen ST und CFC. Das Programmbeispiel **P_CFC_CirculationPumpEx.PRG** für die Programmiersprachen CFC ist im Ordner **Language CFC** > **Actuator** zu finden, das Programmbeispiel **P_ST_CirculationPumpEx.PRG** für die Programmiersprachen ST im Ordner **Language Structur Text** > **Actuator**.

Download	Benötige Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

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

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

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Freigabe des Bausteines, wenn *bEnable* =TRUE ist. Ist *bEnable* = FALSE, so ist der Ausgang *bPump* konstant FALSE.

bExternal: Über den Eingang *bExternal* kann der Ausgang *bPump* direkt ein- oder ausgeschaltet werden. Dazu müssen folgende Bedingungen erfüllt sein: *bEnable* = TRUEAND*bErrorPumpProtec* = FALSEAND*bManSwitch* = TRUEAND*bCtrlVoltage* = TRUEAND*eCtrlModeActuator* = *eHVACActuatorMode_Auto_BMS*OR*eHVACActuatorMode_Auto_OP*AND*eReqPump*=*eHVACRequestPump_NoRequest*. Ansonsten steuert *bExternal* in Abhängigkeit von *bReqOutsideTemp* und *bReqValve* den Ausgang *bPump*, siehe Anwendung [▶ 28].
bExternal ist nur in der Betriebsart *eHVACActuatorMode_Auto_BMS*OR*eHVACActuatorMode_Auto_OP* aktiv.

eCtrlModeActuator: Enum, welches die Betriebsart festlegt. Folgende Betriebsarten werden von dem Funktionsbaustein *FB_HVACCirculationPumpEx* unterstützt: *eHVACActuatorMode_Auto_BMS, eHVACActuatorMode_Auto_OP, eHVACActuatorMode_Speed1_BMS, eHVACActuatorMode_Speed1_OP, eHVACActuatorMode_Off_BMS, eHVACActuatorMode_Off_OP*.
Die beiden Betriebsarten *eHVACActuatorMode_Auto_BMS, eHVACActuatorMode_Auto_OP* bedeuten, dass der Funktionsbaustein sich im Automatikbetrieb befindet.
Über die beiden Betriebsarten *eHVACActuatorMode_Speed1_BMS, eHVACActuatorMode_Speed1_OP* kann der Ausgang *bPump* direkt eingeschaltet werden, wenndie folgenden Bedingungen erfüllt sind: *bEnable* = TRUEAND*bErrorPumpProtec* = FALSEAND*bManSwitch* = TRUEAND*bCtrlVoltage* = TRUE
Die Betriebsarten *eHVACActuatorMode_Off_BMS, eHVACActuatorMode_Off_OP* versetzen den Ausgang *bPump* in den Zustand FALSE.
Liegt ein nicht korrekter Variablenwert an *eCtrlModeActuator* an, dann wird der letzte gültige Variablenwert genommen. Der Status des Enums *eCtrlModeActuator* wird über *eStateModeActuator* ausgegeben.

rOutsideTemp: Eingang zur Übergabe des Außentemperaturwertes. In Abhängigkeit der Außentemperatur kann die Pumpe bei der Unter- oder Überschreitung der Temperaturgrenzwerte *rOutsideTempLowLimit / rOutsideTempHighLimit* eingeschaltet werden. Dieses ist abhängig von dem Enum *eReqPump* zur Anforderung der Pumpe, siehe Anwendung [▶ 28].
rOutsideTemp ist nur in der Betriebsart *eHVACActuatorMode_Auto_BMS*OR*eHVACActuatorMode_Auto_OP* aktiv.

rValvePosition: Eingang zur Übergabe der Ventilstellung des Regelkreises. In Abhängigkeit der Stellung des zur Pumpe gehörigen Ventils *rValvePosition* kann die Pumpe bei Überschreitung des Schwellwertes *rValvePositionLimitOn* eingeschaltet werden. Dieses ist abhängig von dem Enum *eReqPump* zur

Anforderung der Pumpe, siehe Anwendung [▶ 28].
rValvePosition ist nur in der Betriebsart *eHVACActuatorMode_Auto_BMS*OR*eHVACActuatorMode_Auto_OP* aktiv.

bFeedbPump: Rückmeldung von der Pumpe oder eines Relaiskontaktes, dass die Pumpe eingeschaltet ist. Ist der Ausgang *bPump* = TRUE, so muss innerhalb der Zeitangabe *udiSecFeedbPumpDelay* der Eingang *bFeedbPump* = TRUE sein und dieses so lange bleiben, bis *bPump* = FALSE ist. Ansonsten wird dieser Fehler über die Ausgangsvariable *bErrorFeedb* angezeigt. Nach der Behebung der Störung muss diese am Eingang *bReset* quittiert werden. Die Fehlermeldung *bErrorFeedb* hat keinen Einfluss auf die Ansteuerung des Ausgangs *bPump*.
bFeedbPump ist nur in der Betriebsart *eHVACActuatorMode_Auto_BMS*OR*eHVACActuatorMode_Auto_OP* aktiv und wenn die Zeitangabe *udiSecFeedbPumpDelay* größer 0 ist.
Sollte keine Rückmeldung über den Pumpenstatus vorhanden sein, aber trotzdem die Funktion realisiert werden, so darf nicht dauerhaft ein TRUE an den Eingang *bFeedbPump* angelegt werden. Dieses würde im ausgeschalteten Zustand zu einer Störung führen *bErrorFeedb* = TRUE. In diesem Fall sollte der Ausgang *bPump* an den Eingang *bFeedbPump* angelegt werden.

bPumpProtec: An dem Eingang *bPumpProtec* wird der Motorschutz der Pumpe angeschlossen. Eine Störung der Pumpe steht an, wenn der Eingang *bPumpProtec*FALSE ist. Bei einer anstehenden Störung wird der Ausgang *bPump*FALSE und die Störung wird mittels der Variablen *bErrorPumpProtec* angezeigt. Der Ausgang *bPump* kann nur eingschaltet werden, wenn *bPumpProtec* = TRUE ist (Ruhestromprinzip). Nach der Behebung der Störung muss diese am Eingang *bReset* quittiert werden.

bManSwitch: Besitzt die Pumpe einen Hand- / Notschalter, dann kann dieser an den Eingang *bManSwitch* angeschlossen werden. Es wird der Status des Hand- / Notschalters überwacht. Bei *bManSwitch*= FALSE wird der Ausgang *bPump* gesperrt. Der Ausgang *bPump* kann nur eingschaltet werden, wenn *bManSwitch* = TRUE ist (Ruhestromprinzip). Über die Ausgangsvariable *bStateManSwitch* wird der Status von *bManSwitch* angezeigt.

bCtrlVoltage: An dem Eingang *bCtrlVoltage* wird die Überwachung der Steuerspannung angelegt. Ist *bCtrlVoltage* = FALSE, so ist der Ausgang *bPump* konstant FALSE. Zur Meldeschauerunterdrückung werden die Störmeldungen *bPumpProtec, bErrorFeedb* und *bError* nur erfasst, wenn der Eingang *bCtrlVoltage*TRUE ist.

bReset: Quittierungseingang bei einer Störung nach deren Behebung.

VAR_OUTPUT

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

bPump: Ausgangsvariable zur Ansteuerung einer Pumpe. Es müssen zur Ansteuerung der Pumpe über *bPump* vorab die folgenden Bedingungen erfüllt sein: *bEnable* = TRUEAND*bErrorPumpProtec* = FALSEAND*bManSwitch* = TRUEAND*bCtrlVoltage* = TRUE. Sind diese Bedingungen erfüllt, so kann die Pumpe direkt über die Betriebsart *eCtrlModeActuator* eingeschaltet werden oder über verschiedene Möglichkeiten im Automatikbetrieb, siehe Anwendung [▶ 28].
Der Ausgang *bPump* schaltet sich nach dem Entfall der Einschaltbedingungen in Abhängigkeit des Enums *eReqPump*, des Eingangs *bExternal*, der Außentemperatur *rOutSideTemp* oder der Ventilstellung *rValvePosition* verzögert nach Ablauf der Zeitangabe *udiSecStopDelay* aus.

byState: Ausgabe des Pumpenstatus
byState.0 := *bEnable*;
byState.1 := *bPump*;
byState.2 := *bReqExternal*;
byState.3 := *bReqOutsideTemp*;
byState.4 := *bReqValve*;

byState.5 := *bFeedbPump*;
byState.6 := NOT*bManSwitch*;
byState.7 := *bCtrlVoltage*;

bReqExternal: Falls die Bedingung *bExternal* =TRUE zur Einschaltung der Pumpe über *bPump* erfüllt ist, wird die Ausgangsvariable *bReqExternal*TRUE. Alle Einschaltbedingungen und die Verwendung des Ausgangs *bReqExternal* zur Ansteuerung von *bPump* sind in der Anwendung [▶ 28] beschrieben.

bReqOutsideTemp: Falls die Bedingung zur Anforderung der Pumpe über *bPump* in Abhängigkeit der Außentemperatur *rOutsideTemp* erreicht ist, wird die Ausgangsvariable *bReqOutsideTemp*TRUE. Alle Einschaltbedingungen und die Verwendung des Ausgangs *bReqOutsideTemp* zur Ansteuerung von *bPump* sind in der Anwendung [▶ 28] beschrieben.

bReqValve: Falls die Bedingung zur Anforderung der Pumpe über *bPump* in Abhängigkeit der Ventilstellung *rValvePosition* erreicht ist, wird die Ausgangsvariable *bReqValve*TRUE. Alle Einschaltbedingungen und die Verwendung des Ausgangs *bReqValve* zur Ansteuerung von *bPump* sind in der Anwendung [▶ 28] beschrieben.

bStateManSwitch: Statusmeldung des Hand/Not-Schalter. Ein TRUE signalisiert, dass die Hand/Not-Bedienebene aktiviert ist. *bStateManSwitch* = NOT*bManSwitch*

udiSecRT_StopDelay: Ist die Betriebsart *eHVACActuatorMode_Auto_BMS*OR*eHVACActuatorMode_Auto_OP* ausgewählt, so wird der Ausgang *bPump* nach dem Entfall der Einschaltbedingungen in Abhängigkeit des Enums *eReqPump*, des Eingangs *bExternal*, der Außentemperatur *rOutSideTemp* oder der Ventilstellung *rValvePosition* verzögert nach Ablauf von *udiSecRT_StopDelay* ausgeschaltet. Die Ausgabe erfolgt in Sekunden.

udiSecRT_FeedbPumpDelay: Ist der Ausgang *bPump* = TRUE, so muss innerhalb des Zeitablaufs von *udiSecRT_FeedbPumpDelay* der Eingang *bFeedbPump* = TRUE sein und dieses so lange bleiben, bis *bPump* = FALSE ist. Ansonsten wird dieser Fehler über die Ausgangsvariable *bErrorFeedb* angezeigt. *udiSecRT_FeedbPumpDelay* ist nur in der Betriebsart *eHVACActuatorMode_Auto_BMS*OR*eHVACActuatorMode_Auto_OP* aktiv und wenn die Zeitangabe *udiSecFeedbPumpDelay* größer 0 ist. Die Ausgabe erfolgt in Sekunden.

eStateModeActuator: Zeigt an, in welchem Betriebsmodus der Funktionsbaustein sich befindet. *eStateModeActuator* ist gleich *eCtrlModeActuator*.

bError: *bError* wird TRUE, wenn *bErrorPumpProtec*TRUE ist. Zur Abschaltung des Ausgangs *bPump* führt aber nur die Störung *bErrorPumpProtec*. Soll die Störmeldung *bErrorFeedb* ebenfalls zur Abschaltung der Pumpe führen, so müßte die Variable nach dem Aufruf des Funktionsbausteins mit dem Ausgang *bPump*AND-verknüpft werden.

byError: Ausgabe der Fehler als Byte.
byError.1 := bInvalidParameter
byError.2 := bError
byError.3 := bErrorPumpProtec
byError.4 := bErrorFeedb

bErrorPumpProtec: An dem Eingang *bPumpProtec* wird der Motorschutz der Pumpe angeschlossen. Eine Störung der Pumpe steht an, wenn der Eingang *bPumpProtec*FALSE ist. Bei einer anstehenden Störung wird der Ausgang *bPump*FALSE und die Störung wird mittels der Variablen *bErrorPumpProtec* angezeigt. Der Ausgang *bPump* kann nur eingschaltet werden, wenn *bPumpProtec* = TRUE ist (Ruhestromprinzip). Nach der Behebung der Störung muss diese am Eingang *bReset* quittiert werden.

bErrorFeedb: Ist der Ausgang *bPump* = TRUE, so muss innerhalb der Zeitangabe *udiSecFeedbPumpDelay* der Eingang *bFeedbPump* = TRUE sein und dieses so lange bleiben, bis *bPump* = FALSE ist. Ansonsten wird dieser Fehler über die Ausgangsvariable *bErrorFeedb* angezeigt. Nach der Behebung der Störung muss diese am Eingang *bReset* quittiert werden. Die Fehlermeldung *bErrorFeedb* hat keinen Einfluss auf die Ansteuerung des Ausgangs *bPump*.
bErrorFeedb ist nur in der Betriebsart *eHVACActuatorMode_Auto_BMS*OR*eHVACActuatorMode_Auto_OP* aktiv und wenn die Zeitangabe *udiSecFeedbPumpDelay* größer 0 ist.

bInvalidParameter: Wird TRUE, wenn bei der Plausibilitätsüberprüfung der folgenden Variablen ein Fehler aufgetreten ist: *rOutsideTempHighLimit, rOutsideTempLowLimit*, *rValvePositionLimitOn, rValvePositionLimitOff*
Die Meldung muss mit *bReset* quittiert werden.

VAR_IN_OUT

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

udiSecStopDelay: Mit der Zeitverzögerung *udiSecStopDelay* [s] wird das Abschalten der Pumpe nach dem Entfall der Einschaltbedingungen verzögert. Die Eingabe erfolgt in Sekunden (0s..4294967s). Ist die Betriebsart *eHVACActuatorMode_Auto_BMS* OR *eHVACActuatorMode_Auto_OP* ausgewählt, so wird der Ausgang *bPump* nach dem Entfall der Einschaltbedingungen in Abhängigkeit des Enums *eReqPump*, des Eingangs *bExternal*, der Außentemperatur *rOutSideTemp* oder der Ventilstellung *rValvePosition* verzögert nach Ablauf der Zeitverzögerung *udiSecStopDelay* ausgeschaltet. Die Angabe erfolgt in Sekunden. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

udiSecFeedbPumpDelay: Ist der Ausgang *bPump* = TRUE, so muss innerhalb der Zeitverzögerung von *udiSecFeedbPumpDelay* [s] der Eingang *bFeedbPump* = TRUE sein und dieses so lange bleiben, bis *bPump* = FALSE ist. Ansonsten wird dieser Fehler über die Ausgangsvariable *bErrorFeedb* angezeigt (0s..4294967s).
udiSecFeedbPumpDelay ist nur in der Betriebsart *eHVACActuatorMode_Auto_BMS* OR *eHVACActuatorMode_Auto_OP* aktiv und wenn die Zeitangabe *udiSecFeedbPumpDelay* größer 0 ist. Die Eingabe erfolgt in Sekunden. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rOutsideTempHighLimit: Wert oberhalb dessen die Pumpe in Abhängigkeit der Außentemperatur *rOutsideTemp* und dem Enum *eReqPump* ein- bzw. ausgeschaltet wird (-60°C..60°C), siehe Anwendung [▶ 28].
rOutsideTempHighLimit ist nur in der Betriebsart *eHVACActuatorMode_Auto_BMS* OR *eHVACActuatorMode_Auto_OP* aktiv.
Liegt ein nicht korrekter Variablenwert an *rOutsideTempHighLimit* an, dann wird der letzte gültige Variablenwert genommen. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt. Die Variable wird persistent gespeichert. Voreingestellt auf 4.

rOutsideTempLowLimit: Wert unterhalb dessen die Pumpe in Abhängigkeit der Außentemperatur *rOutsideTemp* und dem Enum *eReqPump* ein- bzw. ausgeschaltet wird (-60°C..60°C), siehe Anwendung [▶ 28].
rOutsideTempLowLimit ist nur in der Betriebsart *eHVACActuatorMode_Auto_BMS* OR *eHVACActuatorMode_Auto_OP* aktiv.
Liegt ein nicht korrekter Variablenwert an *rOutsideTempLowLimit* an, dann wird der letzte gültige Variablenwert genommen. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt. Die Variable wird persistent gespeichert. Voreingestellt auf 1.

rValvePositionLimitOn: Schwellwert für die Stellung eines der Pumpe zugehörigen Regelventils *rValvePosition* ab dem die Pumpe bei Überschreitung automatisch einschalten soll, z.B. Erhitzerpumpe (0%..100%), siehe Anwendung [▶ 28].
rValvePositionLimitOn ist nur in der Betriebsart *eHVACActuatorMode_Auto_BMS* OR *eHVACActuatorMode_Auto_OP* aktiv.
rValvePositionLimitOn darf nicht kleiner sein als *rValvePositionLimitOff*. Ansonsten wird der letzte gültige Variablenwert genommen und *bInvalidParameter* wird gesetzt.
Liegt ein nicht korrekter Variablenwert an *rValvePositionLimit* an, dann wird der letzte gültige Variablenwert genommen. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt. Die Variable wird persistent gespeichert. Voreingestellt auf 5.

rValvePositionLimitOff: Schwellwert für die Stellung eines der Pumpe zugehörigen Regelventils *rValvePosition* ab dem die Pumpe bei Untererschreitung automatisch abschalten soll, z.B. Erhitzerpumpe (0%..100%), siehe Anwendung [▶ 28].
rValvePositionLimitOff ist nur in der Betriebsart *eHVACActuatorMode_Auto_BMS* OR *eHVACActuatorMode_Auto_OP* aktiv.
rValvePositionLimitOff darf nicht größer sein als *rValvePositionLimitOn*. Ansonsten wird der letzte gültige Variablenwert genommen und *bInvalidParameter* wird gesetzt.
Liegt ein nicht korrekter Variablenwert an *rValvePositionLimit* an, dann wird der letzte gültige Variablenwert genommen. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt. Die Variable wird persistent gespeichert. Voreingestellt auf 1.

eReqPump: Mittels des Enums *eReqPump* können Einschaltbedingungen oder Kombinationen von Einschaltbedingnungen zur Einschaltung der Pumpe eingestellt werden.
Die Einschaltbedingungen sind folgende:

- in Abhängigkeit der Außentemperatur kann die Pumpe bei der Unter- oder Überschreitung der Temperaturgrenzwerte *rOutsideTempLowLimit* / *rOutsideTempHighLimit* eingeschaltet werden.

- in Abhängigkeit der Stellung des zur Pumpe gehörigen Ventils *rValvePosition* kann die Pumpe bei Überschreitung des Schwellwertes *rValvePositionLimitOn* eingeschaltet werden.

Ausserdem können mit dem Enum Kombinationen bestimmt werden, ob für die temperatur- und ventilstellungsabhängigen Einschaltbedingungen eine ODER - bzw. UND - VERKNÜPFUNG gilt.

Die folgenden Einschaltbedingungen oder Kombinationen von Einschaltbedingnungen können über das Enum eingestellt werden um den Ausgang *bPump* zu steuern:

eHVACReqPump_No: Es besteht keine Anforderung zur Steuerung der Pumpe seitens des Enums

eHVACReqPump_OT_LL: Die Außentemperatur (OT = *rOutsideTemp*) muss kleiner sein als *rOutsideTempLowLimit* (LL = Lower Limit)

eHVACReqPump_OT_HL: Die Außentemperatur (OT = *rOutsideTemp*) muss größer sein als *rOutsideTempHighLimit* (HL = Higher Limit)

eHVACReqPump_VP: Die Ventilstellung (VP = *rValvePosition*) muss größer sein als *rValvePositionLimitOn*

eHVACReqPump_OT_LL_OR_VP: Die Außentemperatur (OT = *rOutsideTemp*) muss kleiner sein als *rOutsideTempLowLimit* (LL = Lower Limit) ODER die Ventilstellung (VP = *rValvePosition*) muss größer sein als *rValvePositionLimitOn*

eHVACReqPump_OT_HL_OR_VP: Die Außentemperatur (OT = *rOutsideTemp*) muss größer sein als *rOutsideTempHighLimit* (HL = Higher Limit) ODER die Ventilstellung (VP = *rValvePosition*) muss größer sein als *rValvePositionLimitOn*

eHVACReqPump_OT_LL_AND_VP: Die Außentemperatur (OT = *rOutsideTemp*) muss kleiner sein als *rOutsideTempLowLimit* (LL = Lower Limit) UND die Ventilstellung (VP = *rValvePosition*) muss größer sein als *rValvePositionLimitOn*

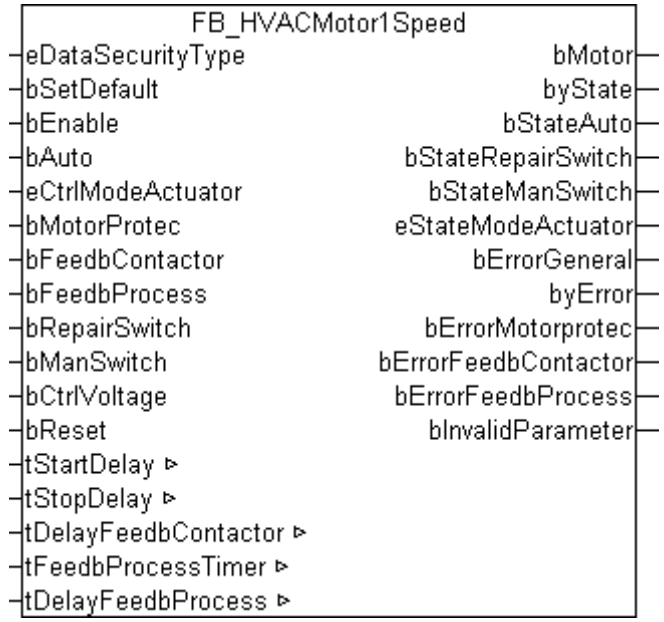
eHVACReqPump_OT_HL_AND_VP: Die Außentemperatur (OT = *rOutsideTemp*) muss größer sein als *rOutsideTempHighLimit* (HL = Higher Limit) UND die Ventilstellung (VP = *rValvePosition*) muss größer sein als *rValvePositionLimitOn*

Vorraussetzungen bei der Verwendung des Enums *eReqPump* sind, dass *bEnable* = TRUEAND*bErrorPumpProtec* = FALSEAND*bManSwitch* = TRUEAND*bCtrlVoltage* = TRUEAND*eCtrlModeActuator* = *eHVACActuatorMode_Auto_BMS*OR*eHVACActuatorMode_Auto_OP* ist. Liegt ein nicht korrekter Variablenwert an *eReqPump* an, dann wird der letzte gültige Variablenwert genommen. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

Dokumente hierzu

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

3.1.5 FB_HVACMotor1Speed



Anwendung

Dieser Funktionsbaustein dient zur Steuerung eines einstufigen Antriebs in der HLK-Technik. Er eignet sich für Ventilatoren.

Anwendungsbeispiel

Download	Benötige Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Freigabe des Bausteins, wenn *bEnable* = TRUE ist. Ist *bEnable* = FALSE, so wird der Antrieb über die angegebene Zeitvariable *tStopDelay* ausgeschaltet.

bAuto: *bAuto* ist nur dann aktiv, wenn die Betriebsart *eCtrlModeActuator* entweder auf *eHVACActuatorMode_Auto_BMS* oder *eHVACActuatorMode_Auto_OP*steht.
Wenn die Eingangsvariable *bAuto* = TRUE ist, so wird dem Funktionsbaustein vorgegeben,dass der Antrieb laufen soll. Wenn *bAuto* = FALSE ist, so wird der Antrieb über die angegebene Zeitvariable *tStopDelay* verzögert ausgeschaltet.

eCtrlModeActuator: Enum, über welches die Betriebsart des Motors vorgegeben wird. Bei falscher Angabe wird intern mit der letzten, gültigen Betriebsart weiter gearbeitet. Bei der Erstinbetriebnahme ist diese *eHVACActuatorMode_Auto_BMS*. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

bMotorProtec: Eingang für den Motorschutz. Eine Störung vom Motorschutz steht an, wenn der Eingang *bMotorProtec* = FALSE ist (Ruhestromprinzip). Bei einer anstehenden Störung wird der Ausgang *bMotor* = FALSE, die Störung wird am Ausgang des Funktionsbausteins durch *bErrorMotorprotec* signalisiert. Ein erneuter Anlauf des Motors kann nur erfolgen, wenn die Störung behoben und am Eingang *bReset* quittiert wurde.

bFeedbContactor: Rückmeldung vom Leistungsteil des Motors. Die Betriebsrückmeldung steht an, wenn der Eingang *bFeedbContactor* = TRUE ist. Falls diese Rückmeldung nach dem Einschalten des Motors nicht nach der mit *tDelayFeedbContactor* einstellbaren Zeit ansteht, so wird der Ausgang *bErrorFeedbContactor* gesetzt um die Störung zu signalisieren. Bei einer anstehenden Störung wird der Ausgang *bMotor* = FALSE. Ein erneuter Anlauf des Motors kann nur erfolgen, wenn die Störung behoben und am Eingang *bReset* quittiert wurde. Sollte keine Rückmeldung vom Leistungsteil des Motors vorhanden sein, so muss an dem Eingang *bFeedbContactor* die Ausgangsvariable *bMotor* angelegt werden.

Sollte keine Rückmeldung vom Leistungsteil des Motors vorhanden sein, so muss an dem Eingang bFeedbContactor die Ausgangsvariable bMotor angelegt werden. Siehe Anwendungsbeispiel

bFeedbProcess: Am Eingang *bFeedbProcess* kann eine Prozessrückmeldung z.B. von einem Keilriemenwächter oder Strömungswächter angeschlossen werden. Die Prozessrückmeldung steht an, wenn der Eingang *bFeedbProcess* = TRUE ist (Ruhestromprinzip). Falls die Prozessrückmeldung in der Hochlaufphase nach dem Einschalten des Motors nicht nach der einstellbaren Zeit *tFeedbProcessTimer* ansteht, schaltet der Antrieb ab und signalisiert an dem Ausgang *bErrorFeedbProcess* eine Störung. Ein erneuter Anlauf des Motors kann nur erfolgen, wenn die Störung behoben und am Eingang *bReset* quittiert wurde. Um ein unerwünschtes Abschalten des Antriebs während des Betriebes durch die Prozessüberwachung bei z.B. kurzzeitigen Druckschwankungen zu vermeiden, kann das Ansprechen des Eingangs *bFeedbProcess* durch die Zeit *tDelayFeedbProcess* verzögert werden.

bRepairSwitch: Mit dem Eingang *bRepairSwitch* wird der Status des Reparaturschalters überwacht. Der Motor kann nur dann eingeschaltet werden, wenn *bRepairSwitch***=** TRUE ist (Ruhestromprinzip). Bei einem ausgeschalteten Reparaturschalter *bRepairSwitch* = FALSE wird der Ausgang *bMotor* = FALSE.

bManSwitch: Mit dem Eingang *bManSwitch* wird der Status des Hand/Not-Schalters überwacht. Der Motor kann nur dann eingeschaltet werden, wenn *bManSwitch*= TRUE ist (Ruhestromprinzip). Bei einem ausgeschalteten Hand-/Not-Schalter *bManSwitch*= FALSE wird der Ausgang *bMotor* = FALSE.

bCtrlVoltage: Mit dem Eingang *bCtrlVoltage* wird die Steuerspannung überwacht. Der Motor kann nur dann eingeschaltet werden, wenn *bCtrlVoltage* = TRUE ist (Ruhestromprinzip). Bei ausgeschalteter Steuerspannung *bCtrlVoltage***=** FALSE wird der Ausgang *bMotor* = FALSE. Um einen Meldeschauer an Störmeldungen bei Ausfall der Steuerspannung zu vermeiden, werden die Störmeldungen des Funktionsbausteines unterdrückt. Liegt die Steuerspannung wieder an, so werden die Störmeldungen wieder freigegeben.

bReset: Quittierungseingang bei einer Störung.

VAR_OUTPUT

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

bMotor: Ausgangsvariable zur Ansteuerung eines 1-stufigen Motors.

byState: Statusbyte über den Betriebszustand des Funktionsbausteines.
byState.0 := *bEnable;*
byState.1 := *bMotor;*
byState.2 := *bStateAuto;*
byState.5 := *bStateRepairSwitch;*
byState.6 := *bStateManSwitch;*
byState.7 := *bCtrlVoltage;*

bStateAuto: Statusmeldung für die Automatikvorwahl, wenn die Betriebsart *eCtrlModeActuator* entweder auf *eHVACActuatorMode_Auto_BMS* oder *eHVACActuatorMode_Auto_OP*stehtund über die Eingangsvariable *bAuto* die Stufe 1 aktiviert wurde.

bStateRepairSwitch: Statusmeldung des Reparaturschalters. Ein TRUE signalisiert, dass der Reparaturschalter ausgeschaltet ist.

bStateManSwitch: Statusmeldung des Hand/Not-Schalter. Ein TRUE signalisiert, dass die Hand/Not-Bedienebene aktiviert ist.

eStateModeActuator: Enum, über das der Status der Betriebsart des Motors an die Steuerung zurück gegeben wird.

bErrorGeneral: Die Störungsmeldung *bErrorGeneral* wird TRUE, sobald eine der Störmeldungen *bErrorMotorprotec, bErrorFeedbContactor* oder *bErrorFeedbProcess* = TRUE ist. Der Ausgang *bMotor* wird dann auf FALSE gesetzt und erst wieder fei gegeben, wenn die Störung behoben ist und mit *bReset* quittiert wurde.

byError: Liefert alle Fehlermeldungen und Warnungen des Funktionsbausteins.
byError.1 := *bInvalidParameter;*
byError.2 := *bErrorGeneral*;
byError.3 := *bErrorMotorprotec*;
byError.4 := *bErrorFeedbContactor;*
byError.5 := *bErrorFeedbProcess;*

bErrorMotorprotec: Fehler Motorschutz, siehe Eingangsvariable *bMotorProtec.*

bErrorFeedbContactor: Fehler Rückmeldung vom Leistungsteil, siehe Eingangsvariable *bFeedbContactor.*

Sollte keine Rückmeldung vom Leistungsteil des Motors vorhanden sein, so muss an dem Eingang bFeedbContactor die Ausgangsvariable bMotor angelegt werden. Siehe Anwendungsbeispiel

bErrorFeedbProcess: Fehler der Prozessrückmeldung, siehe Eingangsvariable *bFeedbackProcess.*

bInvalidParameter: Zeigt an, dass ein falscher Parameter an einer der Variablen *eCtrlModeActuator*, *tStartDelay*, *tStopDelay*, *tDelayFeedbContactor*, *tFeedbProcessTimer* oder *tDelayFeedbProcess* anliegt. Eine falsche Parameterangabe führt nicht zum Stillstand des Bausteins, siehe Beschreibung der Variablen. Nach Behebung der falschen Parameterangabe muss die Meldung *bInvalidParameter* mit *bReset* quittiert werden.

VAR_IN_OUT

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

tStartDelay: Mit der Zeit *tStartDelay* [s] wird der Anlauf des Motors nach der Freigabe und Einschaltung über die Betriebsart des Motors verzögert (0s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 0s.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

tStopDelay: Mit der Zeit *tStopDelay* [s] wird das Ausschalten des Motors durch Ausschalten über die Betriebsart des Motors verzögert (0s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 0s.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

tDelayFeedbContactor: Zeitverzögerung [ms] der Rückmeldung des Leistungsteils nach dem Einschalten des Motors. Wenn diese abgelaufen ist und *bFeedbContactor* = FALSE, dann wird dies über die Störmeldung *bErrorFeedbContactor* der Steuerung zurück geliefert (100ms..3600ms). Die Variable wird persistent gespeichert. Voreingestellt auf 100ms.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

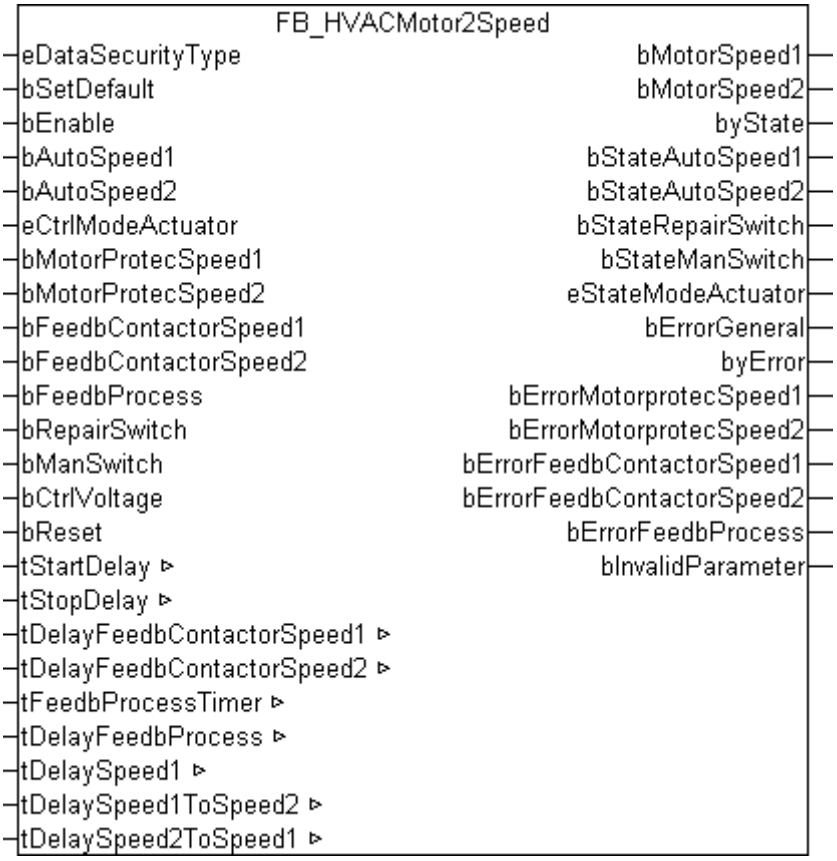
tFeedbProcessTimer: Zeitverzögerung der Prozessrückmeldung *bFeedbProcess* [s] nach dem Einschalten des Motors. Wenn diese abgelaufen ist und *bFeedbProcess* = FALSE, dann wird dies über die Störmeldung *bErrorFeedbProcess* der Steuerung zurück geliefert (0s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 0s.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

tDelayFeedbProcess: Um ein unerwünschtes Abschalten des Antriebs während des Betriebs durch die Prozessüberwachung *bFeedbProcess* bei z.B. kurzzeitigen Druckschwankungen zu vermeiden, kann das Ansprechen des Eingangs *bFeedbProcess* durch die Zeit *tDelayFeedbProcess* [s] verzögert werden (0s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 0s.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

Dokumente hierzu

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

3.1.6 FB_HVACMotor2Speed



Anwendung

Dieser Funktionsbaustein dient zur Steuerung eines zweistufigen Antriebs in der HLK-Technik. Der Funktionsbaustein läuft immer in Stufe 1, der kleineren Leistungsstufe, an. Er kann nicht direkt in Stufe 2 eingeschaltet werden. Bei einem Neustart, Wegnahme der Freigabe, im Fehlerfall oder Ausschaltung des Motors über die Betriebsart ist ein Neuanlauf des Motors für die Zeitdauer von *tDelaySpeed2ToSpeed1* gesperrt.

Anwendungsbeispiel

Download	Benötige Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Freigabe des Bausteins, wenn *bEnable* = TRUE ist. Ist *bEnable* = FALSE, so wird der Antrieb über die angegebene Zeitvariable *tStopDelay* in der jeweiligen Einschaltstufe verzögert ausgeschaltet. Bei vorhandener Freigabe ist ein Neuanlauf des Motors für die Zeitdauer von *tDelaySpeed2ToSpeed1* gesperrt.

bAutoSpeed1: *bAutoSpeed1* ist nur dann aktiv, wenn die Betriebsart *eCtrlModeActuator* entweder auf *eHVACActuatorMode_Auto_BMS* oder *eHVACActuatorMode_Auto_OP*steht.
Wenn die Eingangsvariablen *bAutoSpeed1* = TRUE und *bAutoSpeed2* = FALSE sind, so wird dem Funktionsbaustein vorgegeben,dass der Antrieb in Stufe 1 laufen soll. Wenn *bAutoSpeed1* = FALSE ist, so wird der Antrieb über die angegebene Zeitvariable *tStopDelay* verzögert ausgeschaltet.

bAutoSpeed2: *bAutoSpeed2* ist nur dann aktiv, wenn die Betriebsart *eCtrlModeActuator* entweder auf *eHVACActuatorMode_Auto_BMS* oder *eHVACActuatorMode_Auto_OP*steht.
Wenn die Eingangsvariable *bAutoSpeed1* und *bAutoSpeed2* = TRUE sind, so wird dem Funktionsbaustein vorgegeben, dass der Antrieb in Stufe 2 laufen soll.Wenn *bAutoSpeed1* und*bAutoSpeed2* = FALSE sind, so wird der Antrieb über die angegebene Zeitvariable *tStopDelay* verzögert ausgeschaltet.

eCtrlModeActuator: Enum, über welches die Betriebsart des Motors vorgegeben wird. Bei falscher Angabe wird intern mit der letzten, gültigen Betriebsart weiter gearbeitet. Bei der Erstinbetriebnahme ist diese *eHVACActuatorMode_Auto_BMS*. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

bMotorProtecSpeed1: Eingang für den Motorschutz Stufe 1. Eine Störung vom Motorschutz steht an, wenn der Eingang *bMotorProtecSpeed1* = FALSE ist (Ruhestromprinzip). Bei einer anstehenden Störung werden die Ausgänge *bMotorSpeed1* und *bMotorSpeed2* = FALSE, die Störung wird am Ausgang des Funktionsbausteins durch *bErrorMotorprotecSpeed1* signalisiert. Ein erneuter Anlauf des Motors kann nur erfolgen, wenn die Störung behoben und am Eingang *bReset* quittiert wurde.

bMotorProtecSpeed2: Eingang für den Motorschutz Stufe 2. Eine Störung vom Motorschutz steht an, wenn der Eingang *bMotorProtecSpeed2* = FALSE ist (Ruhestromprinzip). Bei einer anstehenden Störung werden die Ausgänge *bMotorSpeed1* und *bMotorSpeed2* = FALSE, die Störung wird am Ausgang des Funktionsbausteins durch *bErrorMotorprotecSpeed2* signalisiert. Ein erneuter Anlauf des Motors kann nur erfolgen, wenn die Störung behoben und am Eingang *bReset* quittiert wurde.

bFeedbContactorSpeed1: Rückmeldung vom Leistungsteil des Motors für die Stufe 1. Die Betriebsrückmeldung steht an, wenn der Eingang *bFeedbContactorSpeed1* = TRUE ist. Falls diese Rückmeldung nach dem Einschalten des Motors nicht nach der mit *tDelayFeedbContactorSpeed1* einstellbaren Zeit ansteht, wird der Ausgang *bErrorFeedbContactorSpeed1* gesetzt um eine Störung zu signalisieren. Bei einer anstehenden Störung werden die Ausgänge *bMotorSpeed1* und *bMotorSpeed2* = FALSE. Ein erneuter Anlauf des Motors kann nur erfolgen, wenn die Störung behoben und am Eingang

bReset quittiert wurde. Sollte keine Rückmeldung vom Leistungsteil des Motors für die Stufe 1 vorhanden sein, so muss an dem Eingang *bFeedbContactorSpeed1* die Ausgangsvariable *bMotorSpeed1* angelegt werden.

Sollte keine Rückmeldung vom Leistungsteil des Motors vorhanden sein, so muss an dem Eingang bFeedbContactorSpeed1 die Ausgangsvariable bMotorSpeed1 angelegt werden. Siehe Anwendungsbeispiel

bFeedbContactorSpeed2: Rückmeldung vom Leistungsteil des Motors für die Stufe 2. Die Betriebsrückmeldung steht an, wenn der Eingang *bFeedbContactorSpeed2* = TRUE ist. Falls diese Rückmeldung nach dem Einschalten des Motors nicht nach der mit *tDelayFeedbContactorSpeed2* einstellbaren Zeit ansteht, wird der Ausgang *bErrorFeedbContactorSpeed2* gesetzt um eine Störung zu signalisieren. Bei einer anstehenden Störung werden die Ausgänge *bMotorSpeed1* und *bMotorSpeed2* = FALSE. Ein erneuter Anlauf des Motors kann nur erfolgen, wenn die Störung behoben und am Eingang *bReset* quittiert wurde. Sollte keine Rückmeldung vom Leistungsteil des Motors für die Stufe 2 vorhanden sein, so muss an dem Eingang *bFeedbContactorSpeed2* die Ausgangsvariable *bMotorSpeed2* angelegt werden.

Sollte keine Rückmeldung vom Leistungsteil des Motors vorhanden sein, so muss an dem Eingang bFeedbContactorSpeed2 die Ausgangsvariable bMotorSpeed2 angelegt werden. Siehe Anwendungsbeispiel

bFeedbProcess: Am Eingang *bFeedbProcess* kann eine Prozessrückmeldung z.B. von einem Keilriemenwächter oder Strömungswächter angeschlossen werden. Die Prozessrückmeldung steht an, wenn der Eingang *bFeedbProcess* = TRUE ist (Ruhestromprinzip). Falls die Prozessrückmeldung in der Hochlaufphase nach dem Einschalten des Motors nicht nach der einstellbaren Zeit *tFeedbProcessTimer* ansteht, schaltet der Antrieb ab und signalisiert an dem Ausgang *bErrorFeedbProcess* eine Störung. Ein erneuter Anlauf des Motors kann nur erfolgen, wenn die Störung behoben und am Eingang *bReset* quittiert wurde. Um ein unerwünschtes Abschalten des Antriebs während des Betriebes durch die Prozessüberwachung *bFeedbProcess* bei z.B. kurzzeitigen Druckschwankungen zu vermeiden, kann das Ansprechen des Eingangs *bFeedbProcess* durch die Zeit *tDelayFeedbProcess* verzögert werden.
Die Prozessrückmeldung ist aktiv, wenn entweder *bMotorSpeed1* oder *bMotorSpeed2* = TRUE ist.

bRepairSwitch: Mit dem Eingang *bRepairSwitch* wird der Status des Reparaturschalters überwacht. Der Motor kann nur dann eingeschaltet werden, wenn *bRepairSwitch* = TRUE ist (Ruhestromprinzip). Bei einem ausgeschalteten Reparaturschalter *bRepairSwitch* = FALSE werden die Ausgänge *bMotorSpeed1* und *bMotorSpeed2* = FALSE. Ist der Status des Reparaturschalters TRUE, so ist ein Neuanlauf des Motors für die Zeitdauer von *tDelaySpeed2ToSpeed1* gesperrt.

bManSwitch: Mit dem Eingang *bManSwitch* wird der Status des Hand/Not-Schalters überwacht. Der Motor kann nur dann eingeschaltet werden, wenn *bManSwitch*= TRUE ist (Ruhestromprinzip). Bei einem ausgeschalteten Hand-/Not-Schalter *bManSwitch*= FALSE werden die Ausgänge *bMotorSpeed1* und *bMotorSpeed2* = FALSE. Ist der Status des Hand-/Not-Schalters TRUE, so ist ein Neuanlauf des Motors für die Zeitdauer von *tDelaySpeed2ToSpeed1* gesperrt.

bCtrlVoltage: Mit dem Eingang *bCtrlVoltage* wird Steuerspannung überwacht. Der Motor kann nur dann eingeschaltet werden, wenn *bCtrlVoltage* = TRUE ist (Ruhestromprinzip). Bei ausgeschalteter Steuerspannung *bCtrlVoltage*= FALSE werden die Ausgänge *bMotorSpeed1* und *bMotorSpeed2* = FALSE. Um einen Meldeschauer an Störmeldungen bei Ausfall der Steuerspannung zu vermeiden, werden die Störmeldungen des Funktionsbausteins unterdrückt. Liegt die Steuerspannung wieder an, so werden die Störmeldungen wieder freigegeben. Bei vorhandener Steuerspannung ist ein Neuanlauf des Motors für die Zeitdauer von *tDelaySpeed2ToSpeed1* gesperrt.

bReset: Quittierungseingang bei einer Störung.

VAR_OUTPUT

```
bMotorSpeed1               : BOOL;
bMotorSpeed2               : BOOL;
byState                    : BYTE;
bStateAutoSpeed1           : BOOL;
bStateAutoSpeed2           : BOOL;
bStateRepairSwitch         : BOOL;
bStateManSwitch            : BOOL;
eStateModeActuator         : E_HVACActuatorMode;
```

```
bErrorGeneral              : BOOL;
byError                    : BYTE;
bErrorMotorprotecSpeed1    : BOOL;
bErrorMotorprotecSpeed2    : BOOL;
bErrorFeedbContactorSpeed1 : BOOL;
bErrorFeedbContactorSpeed2 : BOOL;
bErrorFeedbProcess         : BOOL;
bInvalidParameter          : BOOL;
```

bMotorSpeed1: Ausgangsvariable zur Ansteuerung der Stufe 1 des zweistufigen Antriebs.

bMotorSpeed2: Ausgangsvariable zur Ansteuerung der Stufe 2 des zweistufigen Antriebs.

byState: Statusbyte über den Betriebszustand des Funktionsbausteines
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: Statusmeldung für die Automatikvorwahl Stufe 1, wenn die Betriebsart *eCtrlModeActuator* entweder auf *eHVACActuatorMode_Auto_BMS* oder *eHVACActuatorMode_Auto_OP*stehtund über die Eingangsvariable *bAutoSpeed1* die Stufe 1 aktiviert wurde.

bStateAutoSpeed2: Statusmeldung für die Automatikvorwahl Stufe 2, wenn die Betriebsart *eCtrlModeActuator* entweder auf *eHVACActuatorMode_Auto_BMS* oder *eHVACActuatorMode_Auto_OP*stehtund über die Eingangsvariable *bAutoSpeed2* die Stufe 2 aktiviert wurde.

bStateRepairSwitch: Statusmeldung des Reparaturschalters. Ein TRUE signalisiert, dass der Reparaturschalter ausgeschaltet ist.

bStateManSwitch: Statusmeldung des Hand/Not-Schalter. Ein TRUE signalisiert, dass die Hand/Not-Bedienebene aktiviert ist.

eStateModeActuator: Enum, über das der Status der Betriebsart des Motors an die Steuerung zurückgegeben wird.

bErrorGeneral: Die Störungsmeldung *bErrorGeneral* wird TRUE, sobald eine der Störmeldungen *bErrorMotorprotecSpeed1, bErrorMotorprotecSpeed2, bErrorFeedbContactorSpeed1, bErrorFeedbContactorSpeed2* oder *bErrorFeedbProcess* = TRUE ist. Die Ausgänge *bMotorSpeed1* und *bMotorSpeed2* werden auf FALSE gesetzt und erst wieder fei gegeben, wenn die Störung behoben ist und mit *bReset* quittiert wurde. Nach Behebung der Störung ist ein Neuanlauf des Motors für die Zeitdauer von *tDelaySpeed2ToSpeed1* gesperrt.

byError: Liefert alle Fehlermeldungen und Warnungen des Funktionsbausteins**.**
byError.1 := *bInvalidParameter;*
byError.2 := *bErrorGeneral*;
byError.3 := *bErrorMotorprotecSpeed1*;
byError.4 := *bErrorMotorprotecSpeed2;*
byError.5 := *bErrorFeedbContactorSpeed1;*
byError.6 := *bErrorFeedbContactorSpeed2*;
byError.7 := *bErrorFeedbProcess;*

bErrorMotorprotecSpeed1: Fehler Motorschutz, siehe Eingangsvariable *bMotorProtecSpeed1*

bErrorMotorprotecSpeed2: Fehler Motorschutz, siehe Eingangsvariable *bMotorProtecSpeed2*

bErrorFeedbContactorSpeed1: Fehler Rückmeldung vom Leistungsteil, siehe Eingangsvariable *bFeedbContactorSpeed1*

Sollte keine Rückmeldung vom Leistungsteil des Motors vorhanden sein, so muss an dem Eingang bFeedbContactorSpeed1 die Ausgangsvariable bMotorSpeed1 angelegt werden. Siehe Anwendungsbeispiel

bErrorFeedbContactorSpeed2: Fehler Rückmeldung vom Leistungsteil, siehe Eingangsvariable *bFeedbContactorSpeed2*

Sollte keine Rückmeldung vom Leistungsteil des Motors vorhanden sein, so muss an dem Eingang bFeedbContactorSpeed2 die Ausgangsvariable bMotorSpeed2 angelegt werden. Siehe Anwendungsbeispiel

bErrorFeedbProcess: Fehler der Prozessrückmeldung, siehe Eingangsvariable *bFeedbackProcess*

bInvalidParameter: Zeigt an, dass ein falscher Parameter an einer der Variablen *eCtrlModeActuator*, *tStartDelay*, *tStopDelay*, *tDelayFeedbContactorSpeed1, tDelayFeedbContactorSpeed2*, *tFeedbProcessTimer,tDelayFeedbProcess, tDelaySpeed1* oder *tDelaySpeed1ToSpeed2* anliegt. Eine falsche Parameterangabe führt nicht zum Stillstand des Bausteins, siehe Beschreibung der Variablen. Nach Behebung der falschen Parameterangabe muss die Meldung *bInvalidParameter* mit *bReset* quittiert werden.

VAR_IN_OUT

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

tStartDelay: Mit der Zeit *tStartDelay* [s] wird der Anlauf des Motors nach der Freigabe und Einschaltung über die Betriebsart des Motors verzögert (0s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 0s.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

tStopDelay: Mit der Zeit *tStopDelay* [s] wird das Ausschalten des Motors in der jeweiligen Einschaltstufe entweder durch die Wegnahme der Freigabe *bEnable* = FALSE, durch Ausschalten über die Betriebsart *eCtrlModeActuator* oder im Automatikbetrieb durch Ausschalten der Eingangsvariablen *bAutoSpeed1* und *bAutoSpeed2* = FALSE verzögert. Ist das verzögerte Ausschalten des Motors aktiviert, kann dieses nicht mehr abgebrochen werden (0s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 0s.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

tDelayFeedbContactorSpeed1: Zeitverzögerung der Rückmeldung des Leistungsteils nach dem Einschalten des Motors. Wenn diese abgelaufen ist und *bFeedbContactorSpeed1* = FALSE, dann wird dies über die Störmeldung *bErrorFeedbContactorSpeed1* der Steuerung zurück geliefert (100ms..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 100ms.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

tDelayFeedbContactorSpeed2: Zeitverzögerung der Rückmeldung des Leistungsteils nach dem Einschalten des Motors. Wenn diese abgelaufen ist und *bFeedbContactorSpeed2* = FALSE, dann wird dies über die Störmeldung *bErrorFeedbContactorSpeed2* der Steuerung zurück geliefert (100ms..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 100ms.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

tFeedbackProcessTimer: Zeitverzögerung [s] der Prozessrückmeldung *bFeedbProcess* in der Hochlaufphase des Motors. Wenn diese abgelaufen ist und *bFeedbProcess* = FALSE, dann wird dies über die Störmeldung *bErrorFeedbProcess* der Steuerung zurück geliefert (0s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 0s.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

tDelayFeedbProcess: Um ein unerwünschtes Abschalten des Antriebs während des Betriebs durch die Prozessüberwachung *bFeedbProcess* bei z.B. kurzzeitigen Druckschwankungen zu vermeiden, kann das Ansprechen des Eingangs *bFeedbProcess* durch die Zeit *tDelayFeedbProcess* [s] verzögert werden (0s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 0s.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

tDelaySpeed1: Zeitverzögerung [s] für die Hochlaufphase des Motors in Stufe 1 (1s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 3s.
Nach Ablauf dieser Zeit kann der Motor von der ersten in die zweite Stufe geschaltet werden, wenn die Betriebsart für Stufe 2 angewählt ist.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

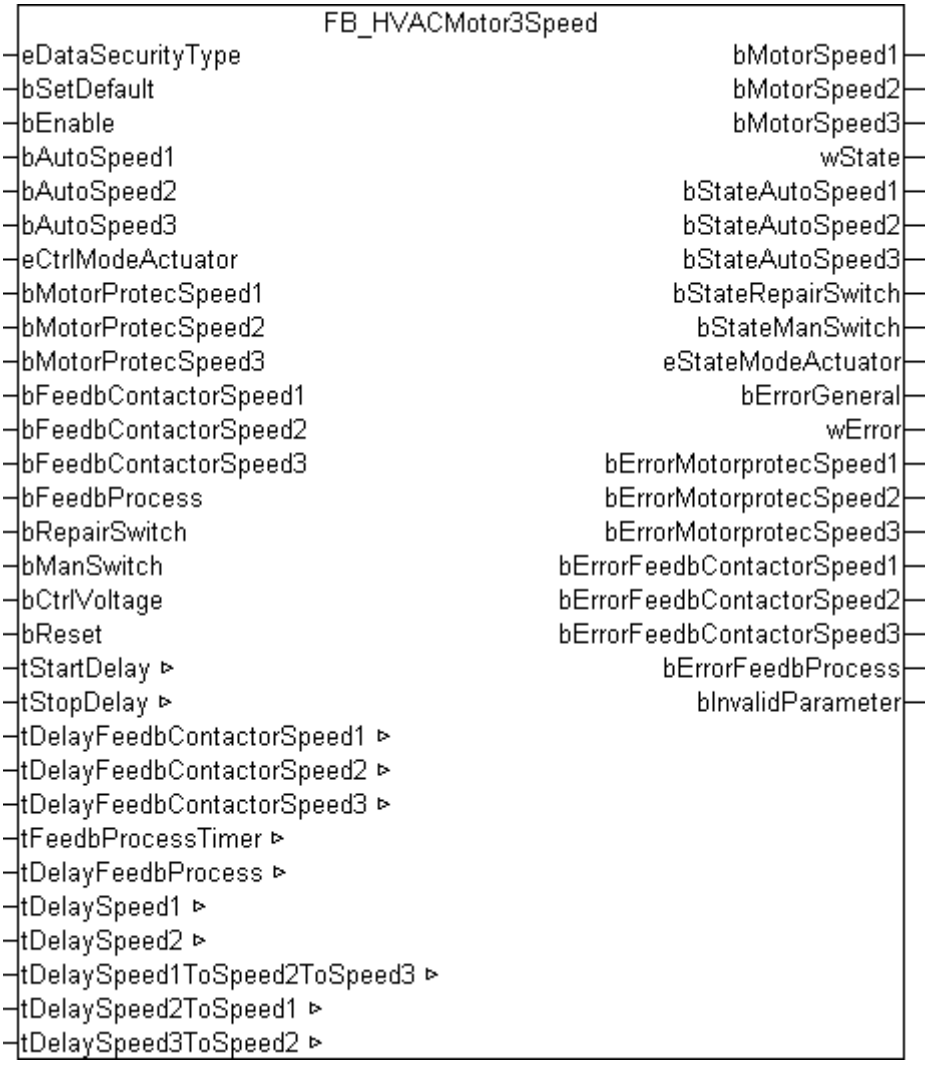
tDelaySpeed1ToSpeed2: Zeitverzögerung für die Umschaltphase des Motors von Stufe 1 in Stufe 2, so dass kurzzeitig die beiden Ausgänge *bMotorSpeed1* und *bMotorSpeed2* FALSE sind (100ms..10s). Die Variable wird persistent gespeichert. Voreingestellt auf 250ms. Die Zeitverzögerung dient zur Schonung der Motorwicklungen.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

tDelaySpeed2ToSpeed1: Zeitverzögerung [s] für die Umschaltphase des Motors von Stufe 2 in Stufe 1 (1s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 10s. In dieser Phase sind die beiden Ausgänge *bMotorSpeed1* und *bMotorSpeed2* für die Zeitdauer von *tDelaySpeed2ToSpeed1*auf FALSE um die Geschwindigkeit des Motors bei der Umschaltung in Stufe 1 zu drosseln.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

Dokumente hierzu

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

3.1.7 FB_HVACMotor3Speed



Anwendung

Dieser Funktionsbaustein dient zur Steuerung eines dreistufigen Antriebs in der HLK-Technik. Der Funktionsbaustein läuft immer in Stufe 1, der kleinsten Leistungsstufe, an und schaltet dann je nach Anforderung in Stufe 2 oder Stufe 3. Er kann nicht direkt in Stufe 2 oder 3 eingeschaltet werden. Bei einem Neustart, Wegnahme der Freigabe, im Fehlerfall oder Ausschaltung des Motors über die Betriebsart ist ein Neuanlauf des Motors für die Zeitdauer von *tDelaySpeed2ToSpeed1* + *tDelaySpeed3ToSpeed2* gesperrt.

Anwendungsbeispiel

Download	Benötige Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

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

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

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Freigabe des Bausteines, wenn *bEnable* = TRUE ist. Ist *bEnable* = FALSE, so wird der Antrieb über die angegebene Zeitvariable *tStopDelay* in der jeweiligen Einschaltstufe verzögert ausgeschaltet.
Bei vorhandener Freigabe ist ein Neuanlauf des Motors für die Zeitdauer von *tDelaySpeed2ToSpeed1* + *tDelaySpeed3ToSpeed2* gesperrt.

bAutoSpeed1: *bAutoSpeed1* ist nur dann aktiv, wenn die Betriebsart *eCtrlModeActuator* entweder auf *eHVACActuatorMode_Auto_BMS* oder *eHVACActuatorMode_Auto_OP*steht.
Wenn die Eingangsvariablen *bAutoSpeed1* = TRUE, *bAutoSpeed2* = FALSE und *bAutoSpeed3* = FALSE sind, so wird dem Funktionsbaustein vorgegeben,dass der Antrieb in Stufe 1 laufen soll.
Wenn *bAutoSpeed1* = FALSE ist, so wird der Antrieb über die angegebene Zeitvariable *tStopDelay* verzögert ausgeschaltet.

bAutoSpeed2: *bAutoSpeed2* ist nur dann aktiv, wenn die Betriebsart *eCtrlModeActuator* entweder auf *eHVACActuatorMode_Auto_BMS* oder *eHVACActuatorMode_Auto_OP*steht.
Wenn die Eingangsvariablen *bAutoSpeed1* = TRUE, *bAutoSpeed2* = TRUE und *bAutoSpeed3* = FALSE sind, so wird dem Funktionsbaustein vorgegeben,dass der Antrieb in Stufe 2 laufen soll.
Wenn *bAutoSpeed1* und *bAutoSpeed2* = FALSE sind, so wird der Antrieb über die angegebene Zeitvariable *tStopDelay* verzögert ausgeschaltet.

bAutoSpeed3: *bAutoSpeed3* ist nur dann aktiv, wenn die Betriebsart *eCtrlModeActuator* entweder auf *eHVACActuatorMode_Auto_BMS* oder *eHVACActuatorMode_Auto_OP*steht.
Wenn die Eingangsvariablen *bAutoSpeed1* = TRUE, *bAutoSpeed2* = TRUE und *bAutoSpeed3* = TRUE sind, so wird dem Funktionsbaustein vorgegeben,dass der Antrieb in Stufe 3 laufen soll.
Wenn *bAutoSpeed1, bAutoSpeed2* und *bAutoSpeed3* = FALSE sind, so wird der Antrieb über die angegebene Zeitvariable *tStopDelay* verzögert ausgeschaltet.

eCtrlModeActuator: Enum, über welches die Betriebsart des Motors vorgegeben wird. Bei falscher Angabe wird intern mit der letzten, gültigen Betriebsart weiter gearbeitet. Bei der Erstinbetriebnahme ist diese *eHVACActuatorMode_Auto_BMS*. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

bMotorProtecSpeed1: Eingang für den Motorschutz Stufe 1. Eine Störung vom Motorschutz steht an, wenn der Eingang *bMotorProtecSpeed1* = FALSE ist (Ruhestromprinzip). Bei einer anstehenden Störung werden die Ausgänge *bMotorSpeed1*, *bMotorSpeed2* und *bMotorSpeed3* = FALSE, die Störung wird am Ausgang des Funktionsbausteins durch *bErrorMotorprotecSpeed1* signalisiert. Ein erneuter Anlauf des Motors kann nur erfolgen, wenn die Störung behoben und am Eingang *bReset* quittiert wurde.

bMotorProtecSpeed2: Eingang für den Motorschutz Stufe 2. Eine Störung vom Motorschutz steht an, wenn der Eingang *bMotorProtecSpeed2* = FALSE ist (Ruhestromprinzip). Bei einer anstehenden Störung werden die Ausgänge *bMotorSpeed1*, *bMotorSpeed2* und *bMotorSpeed3* = FALSE, die Störung wird am Ausgang des Funktionsbausteins durch *bErrorMotorprotecSpeed2* signalisiert. Ein erneuter Anlauf des Motors kann nur erfolgen, wenn die Störung behoben und am Eingang *bReset* quittiert wurde.

bMotorProtecSpeed3: Eingang für den Motorschutz Stufe 3. Eine Störung vom Motorschutz steht an, wenn der Eingang *bMotorProtecSpeed3* = FALSE ist (Ruhestromprinzip). Bei einer anstehenden Störung werden die Ausgänge *bMotorSpeed1*, *bMotorSpeed2* und *bMotorSpeed3* = FALSE, die Störung wird am Ausgang des Funktionsbausteins durch *bErrorMotorprotecSpeed3* signalisiert. Ein erneuter Anlauf des Motors kann nur erfolgen, wenn die Störung behoben und am Eingang *bReset* quittiert wurde.

bFeedbContactorSpeed1: Rückmeldung vom Leistungsteil des Motors für die Stufe 1. Die Betriebsrückmeldung steht an, wenn der Eingang *bFeedbContactorSpeed1* = TRUE ist. Falls diese Rückmeldung nach dem Einschalten des Motors nicht nach der mit *tDelayFeedbContactorSpeed1* einstellbaren Zeit ansteht, so wird der Ausgang *bErrorFeedbContactorSpeed1* gesetzt um eine Störung zu signalisieren. Bei einer anstehenden Störung werden die Ausgänge *bMotorSpeed1*, *bMotorSpeed2* und *bMotorSpeed3* = FALSE. Ein erneuter Anlauf des Motors kann nur erfolgen, wenn die Störung behoben und am Eingang *bReset* quittiert wurde. Sollte keine Rückmeldung vom Leistungsteil des Motors für die Stufe 1 vorhanden sein, so muss an dem Eingang *bFeedbContactorSpeed1* die Ausgangsvariable *bMotorSpeed1* angelegt werden.

Sollte keine Rückmeldung vom Leistungsteil des Motors vorhanden sein, so muss an dem Eingang bFeedbContactorSpeed1 die Ausgangsvariable bMotorSpeed1 angelegt werden. Siehe Anwendungsbeispiel

bFeedbContactorSpeed2: Rückmeldung vom Leistungsteil des Motors für die Stufe 2. Die Betriebsrückmeldung steht an, wenn der Eingang *bFeedbContactorSpeed2* = TRUE ist. Falls diese Rückmeldung nach dem Einschalten des Motors nicht nach der mit *tDelayFeedbContactorSpeed2* einstellbaren Zeit ansteht, so wird der Ausgang *bErrorFeedbContactorSpeed2* gesetzt um eine Störung zu signalisieren. Bei einer anstehenden Störung werden die Ausgänge *bMotorSpeed1*, *bMotorSpeed2* und *bMotorSpeed3* = FALSE. Ein erneuter Anlauf des Motors kann nur erfolgen, wenn die Störung behoben und am Eingang *bReset* quittiert wurde. Sollte keine Rückmeldung vom Leistungsteil des Motors für die Stufe 2 vorhanden sein, so muss an dem Eingang *bFeedbContactorSpeed2* die Ausgangsvariable *bMotorSpeed2* angelegt werden.

Sollte keine Rückmeldung vom Leistungsteil des Motors vorhanden sein, so muss an dem Eingang bFeedbContactorSpeed2 die Ausgangsvariable bMotorSpeed2 angelegt werden. Siehe Anwendungsbeispiel

bFeedbContactorSpeed3: Rückmeldung vom Leistungsteil des Motors für die Stufe 3. Die Betriebsrückmeldung steht an, wenn der Eingang *bFeedbContactorSpeed3* = TRUE ist. Falls diese Rückmeldung nach dem Einschalten des Motors nicht nach der mit *tDelayFeedbContactorSpeed3* einstellbaren Zeit ansteht, so wird der Ausgang *bErrorFeedbContactorSpeed3* gesetzt um eine Störung zu signalisieren. Bei einer anstehenden Störung werden die Ausgänge *bMotorSpeed1*, *bMotorSpeed2* und *bMotorSpeed3* = FALSE. Ein erneuter Anlauf des Motors kann nur erfolgen, wenn die Störung behoben und am Eingang *bReset* quittiert wurde. Sollte keine Rückmeldung vom Leistungsteil des Motors für die Stufe 3 vorhanden sein, so muss an dem Eingang *bFeedbContactorSpeed3* die Ausgangsvariable *bMotorSpeed3* angelegt werden.

Sollte keine Rückmeldung vom Leistungsteil des Motors vorhanden sein, so muss an dem Eingang bFeedbContactorSpeed3 die Ausgangsvariable bMotorSpeed3 angelegt werden. Siehe Anwendungsbeispiel

bFeedbProcess: Am Eingang *bFeedbProcess* kann eine Prozessrückmeldung z.B. von einem Keilriemenwächter oder Strömungswächter angeschlossen werden. Die Prozessrückmeldung steht an, wenn der Eingang *bFeedbProcess* = TRUE ist (Ruhestromprinzip). Falls die Prozessrückmeldung in der Hochlaufphase nach dem Einschalten des Motors nicht nach der einstellbaren Zeit *tFeedbProcessTimer* ansteht, so schaltet der Antrieb ab und signalisiert an dem Ausgang *bErrorFeedbProcess* eine Störung. Ein erneuter Anlauf des Motors kann nur erfolgen, wenn die Störung behoben und am Eingang *bReset* quittiert wurde. Um ein unerwünschtes Abschalten des Antriebs während des Betriebs durch die Prozessüberwachung *bFeedbProcess* bei z.B. kurzzeitigen Druckschwankungen zu vermeiden, so kann das

Ansprechen des Eingangs *bFeedbProcess* durch die Zeit *tDelayFeedbProcess* verzögert werden.
Die Prozessrückmeldung ist aktiv, wenn entweder *bMotorSpeed1*, *bMotorSpeed2* oder *bMotorSpeed3* = TRUE ist.

bRepairSwitch: Mit dem Eingang *bRepairSwitch* wird der Status des Reparaturschalters überwacht. Der Motor kann nur dann eingeschaltet werden, wenn *bRepairSwitch* = TRUE ist (Ruhestromprinzip). Bei einem ausgeschalteten Reparaturschalter *bRepairSwitch* = FALSE werden die Ausgänge *bMotorSpeed1*, *bMotorSpeed2* und *bMotorSpeed3* = FALSE. Ist der Status des Reparaturschalters TRUE, so ist ein Neuanlauf des Motors für die Zeitdauer von *tDelaySpeed2ToSpeed1* + *tDelaySpeed3ToSpeed2* gesperrt.

bManSwitch: Mit dem Eingang *bManSwitch* wird der Status des Hand/Not-Schalters überwacht. Der Motor kann nur dann eingeschaltet werden, wenn *bManSwitch*= TRUE ist (Ruhestromprinzip). Bei einem ausgeschalteten Hand/Not-Schalter *bManSwitch*= FALSE werden die Ausgänge *bMotorSpeed1*, *bMotorSpeed2* und *bMotorSpeed3* = FALSE. Ist der Status des Hand/Not-Schalters TRUE, so ist ein Neuanlauf des Motors für die Zeitdauer von *tDelaySpeed2ToSpeed1* + *tDelaySpeed3ToSpeed2* gesperrt.

bCtrlVoltage: Mit dem Eingang *bCtrlVoltage* wird Steuerspannung überwacht. Der Motor kann nur dann eingeschaltet werden, wenn *bCtrlVoltage* = TRUE ist (Ruhestromprinzip). Bei ausgeschalteter Steuerspannung *bCtrlVoltage*= FALSE werden die Ausgänge *bMotorSpeed1*, *bMotorSpeed2* und *bMotorSpeed3* = FALSE. Um einen Meldeschauer an Störmeldungen bei Ausfall der Steuerspannung zu vermeiden, werden die Störmeldungen des Funktionsbausteins unterdrückt. Liegt die Steuerspannung wieder an, so werden die Störmeldungen wieder freigegeben. Bei vorhandener Steuerspannung ist ein Neuanlauf des Motors für die Zeitdauer von *tDelaySpeed2ToSpeed1* + *tDelaySpeed3ToSpeed2* gesperrt.

bReset: Eingang zur Quittierung der Störungen über eine steigende Flanke.

VAR_OUTPUT

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

bMotorSpeed1: Ausgangsvariable zur Ansteuerung der Stufe 1 des dreistufigen Antriebs.

bMotorSpeed2: Ausgangsvariable zur Ansteuerung der Stufe 2 des dreistufigen Antriebs.

bMotorSpeed3: Ausgangsvariable zur Ansteuerung der Stufe 3 des dreistufigen Antriebs.

wState: Statusword über den Betriebszustand des Funktionsbausteins.
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: Statusmeldung für die Automatikvorwahl Stufe 1, wenn die Betriebsart *eCtrlModeActuator* entweder auf *eHVACActuatorMode_Auto_BMS* oder *eHVACActuatorMode_Auto_OP*stehtund über die Eingangsvariable *bAutoSpeed1* die Stufe 1 aktiviert wurde.

bStateAutoSpeed2: Statusmeldung für die Automatikvorwahl Stufe 2, wenn die Betriebsart *eCtrlModeActuator* entweder auf *eHVACActuatorMode_Auto_BMS* oder *eHVACActuatorMode_Auto_OP*stehtund über die Eingangsvariable *bAutoSpeed2* die Stufe 2 aktiviert wurde.

bStateAutoSpeed3: Statusmeldung für die Automatikvorwahl Stufe 3, wenn die Betriebsart *eCtrlModeActuator* entweder auf *eHVACActuatorMode_Auto_BMS* oder *eHVACActuatorMode_Auto_OP*stehtund über die Eingangsvariable *bAutoSpeed3* die Stufe 3 aktiviert wurde.

bStateRepairSwitch: Statusmeldung des Reparaturschalters. Ein TRUE signalisiert, dass der Reparaturschalter ausgeschaltet ist.

bStateManSwitch: Statusmeldung des Hand/Not-Schalter. Ein TRUE signalisiert, dass die Hand/Not-Bedienebene aktivert ist.

eStateModeActuator: Enum, über das der Status der Betriebsart des Motors an die Steuerung zurückgegeben wird.

bErrorGeneral: Die Störungsmeldung *bErrorGeneral* wird TRUE, sobald eine der Störmeldungen *bErrorMotorprotecSpeed1, bErrorMotorprotecSpeed2, bErrorMotorprotecSpeed3, bErrorFeedbContactorSpeed1, bErrorFeedbContactorSpeed2, bErrorFeedbContactorSpeed23* oder *bErrorFeedbProcess* = TRUE ist. Die Ausgänge *bMotorSpeed1, bMotorSpeed2* und *bMotorSpeed3* werden auf FALSE gesetzt und erst wieder fei gegeben, wenn die Störung behoben ist und mit *bReset* quittiert wurde. Nach Behebung der Störung ist ein Neuanlauf des Motors für die Zeitdauer von *tDelaySpeed2ToSpeed1* + *tDelaySpeed3ToSpeed2* gesperrt.

wError: Liefert alle Fehlermeldungen und Warnungen des Funktionsbausteines**.**
wError.1 := *bInvalidParameter;*
wError.2 := *bErrorGeneral*;
wError.3 := *bErrorMotorprotecSpeed1*;
wError.4 := *bErrorMotorprotecSpeed2;*
wError.5 := *bErrorMotorprotecSpeed3*;
wError.6 := *bErrorFeedbContactorSpeed1;*
wError.7 := *bErrorFeedbContactorSpeed2*;
wError.8 := *bErrorFeedbContactorSpeed3*;
wError.9 := *bErrorFeedbProcess;*

bErrorMotorprotecSpeed1: Fehler Motorschutz, siehe Eingangsvariable *bMotorProtecSpeed1*

bErrorMotorprotecSpeed2: Fehler Motorschutz, siehe Eingangsvariable *bMotorProtecSpeed2*

bErrorMotorprotecSpeed3: Fehler Motorschutz, siehe Eingangsvariable *bMotorProtecSpeed3*

bErrorFeedbContactorSpeed1: Fehler Rückmeldung vom Leistungsteil, siehe Eingangsvariable *bFeedbContactorSpeed1*

Sollte keine Rückmeldung vom Leistungsteil des Motors vorhanden sein, so muss an dem Eingang bFeedbContactorSpeed1 die Ausgangsvariable bMotorSpeed1 angelegt werden. Siehe Anwendungsbeispiel

bErrorFeedbContactorSpeed2: Fehler Rückmeldung vom Leistungsteil, siehe Eingangsvariable *bFeedbContactorSpeed2*

Sollte keine Rückmeldung vom Leistungsteil des Motors vorhanden sein, so muss an dem Eingang bFeedbContactorSpeed2 die Ausgangsvariable bMotorSpeed2 angelegt werden. Siehe Anwendungsbeispiel

bErrorFeedbContactorSpeed3: Fehler Rückmeldung vom Leistungsteil, siehe Eingangsvariable *bFeedbContactorSpeed3*

Sollte keine Rückmeldung vom Leistungsteil des Motors vorhanden sein, so muss an dem Eingang bFeedbContactorSpeed3 die Ausgangsvariable bMotorSpeed3 angelegt werden. Siehe Anwendungsbeispiel

bErrorFeedbProcess: Fehler der Prozessrückmeldung, siehe Eingangsvariable *bFeedbackProcess*

bInvalidParameter: Zeigt an, dass ein falscher Parameter an einer der Variablen *eCtrlModeActuator*, *tStartDelay*, *tStopDelay*, *tDelayFeedbContactorSpeed1, tDelayFeedbContactorSpeed2*, *tDelayFeedbContactorSpeed3,tFeedbProcessTimer,tDelayFeedbProcess, tDelaySpeed1, tDelaySpeed2, tDelaySpeed1ToSpeed2ToSpeed3, tDelaySpeed2ToSpeed1* oder *tDelaySpeed3ToSpeed2* anliegt. Eine falsche Parameterangabe führt nicht zum Stillstand des Bausteins, siehe Beschreibung der Variablen. Nach Behebung der falschen Parameterangabe muss die Meldung *bInvalidParameter* mit *bReset* quittiert werden.

VAR_IN_OUT

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

tStartDelay: Mit der Zeit *tStartDelay* [s] wird der Anlauf des Motors nach der Freigabe und Einschaltung über die Betriebsart des Motors verzögert (0s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 0s.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

tStopDelay: Mit der Zeit *tStopDelay* [s] wird das Ausschalten des Motors in der jeweiligen Einschaltstufe entweder durch die Wegnahme der Freigabe *bEnable* = FALSE, durch Ausschalten über die Betriebsart *eCtrlModeActuator* oder im Automatikbetrieb durch Ausschalten der Eingangsvariablen *bAutoSpeed1, bAutoSpeed2* und *bAutoSpeed3* = FALSE verzögert. Ist das verzögerte Ausschalten des Motors aktiviert, kann dieses nicht mehr abgebrochen werden (0s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 0s.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

tDelayFeedbContactorSpeed1: Zeitverzögerung der Rückmeldung des Leistungsteils nach dem Einschalten des Motors. Wenn diese abgelaufen ist und *bFeedbContactorSpeed1* = FALSE, dann wird dies über die Störmeldung *bErrorFeedbContactorSpeed1* der Steuerung zurück geliefert (100ms..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 100ms.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weiter gearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

tDelayFeedbContactorSpeed2: Zeitverzögerung der Rückmeldung des Leistungsteils nach dem Einschalten des Motors. Wenn diese abgelaufen ist und *bFeedbContactorSpeed2* = FALSE, dann wird dies über die Störmeldung *bErrorFeedbContactorSpeed2* der Steuerung zurück geliefert (100ms..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 100ms.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

tDelayFeedbContactorSpeed3: Zeitverzögerung der Rückmeldung des Leistungsteils nach dem Einschalten des Motors. Wenn diese abgelaufen ist und *bFeedbContactorSpeed3* = FALSE, dann wird dies über die Störmeldung *bErrorFeedbContactorSpeed3* der Steuerung zurück geliefert (100ms..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 100ms.

Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

tFeedbackProcessTimer: Zeitverzögerung [s] der Prozessrückmeldung *bFeedbProcess* in der Hochlaufphase des Motors. Wenn diese abgelaufen ist und *bFeedbProcess* = FALSE, dann wird dies über die Störmeldung *bErrorFeedbProcess* der Steuerung zurück geliefert (0s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 0s.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

tDelayFeedbProcess: Um ein unerwünschtes Abschalten des Antriebs während des Betriebs durch die Prozessüberwachung *bFeedbProcess* bei z.B. kurzzeitigen Druckschwankungen zu vermeiden, kann das Ansprechen des Eingangs *bFeedbProcess* durch die Zeit *tDelayFeedbProcess* [s] verzögert werden (0s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 0s.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

tDelaySpeed1: Zeitverzögerung [s] für die Hochlaufphase des Motors in Stufe 1 (1s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 3s.
Nach Ablauf dieser Zeit kann der Motor von der ersten in die zweite Stufe geschaltet werden, wenn die Betriebsart für Stufe 2 angewählt ist.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

tDelaySpeed2: Zeitverzögerung [s] für die Hochlaufphase des Motors in Stufe 2 (1s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 3s.
Nach Ablauf dieser Zeit kann der Motor von der zweiten in die dritte Stufe geschaltet werden, wenn die Betriebsart für Stufe 3 angewählt ist.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

tDelaySpeed1ToSpeed2ToSpeed3: Zeitverzögerung für die Umschaltphase des Motors von Stufe 1 in Stufe 2 oder von Stufe 2 in Stufe 3, so dass kurzzeitig die Ausgänge *bMotorSpeed1*, *bMotorSpeed2* und *bMotorSpeed3* FALSE sind (1s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 250ms. Die Zeitverzögerung dient zur Schonung der Motorwicklungen.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

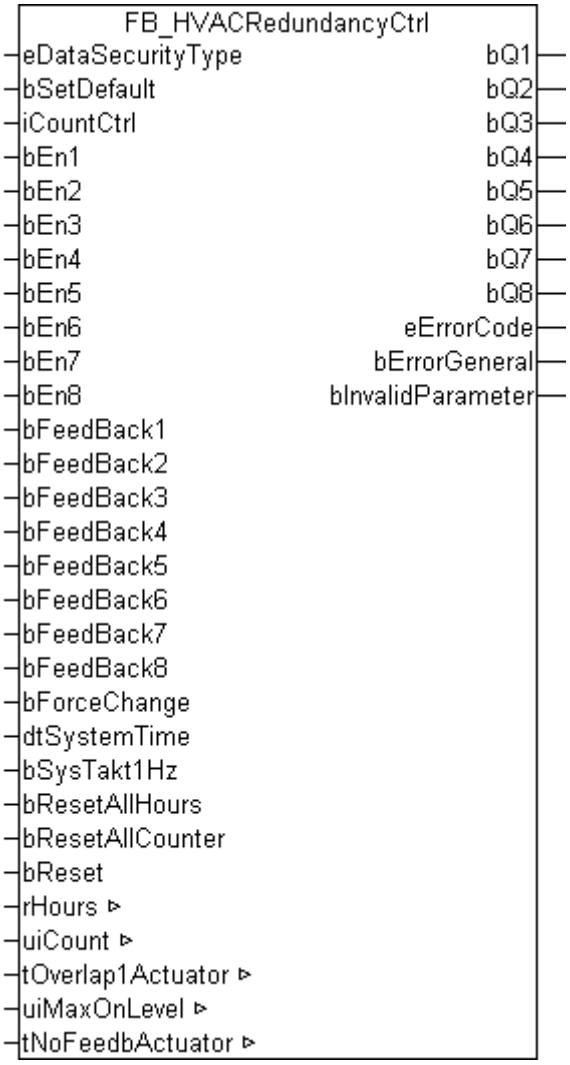
tDelaySpeed2ToSpeed1: Zeitverzögerung [s] für die Umschaltphase des Motors von Stufe 2 in Stufe 1 (1s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 10s. In dieser Phase sind die Ausgänge *bMotorSpeed1, bMotorSpeed2* und *bMotorSpeed3* für die Zeitdauer von *tDelaySpeed2ToSpeed1*auf FALSE um die Geschwindigkeit des Motors bei der Umschaltung in Stufe 1 zu drosseln.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

tDelaySpeed3ToSpeed2: Zeitverzögerung [s] für die Umschaltphase des Motors von Stufe 3 in Stufe 2 (1s..3600s). Die Variable wird persistent gespeichert. Voreingestellt auf 10s. In dieser Phase sind die Ausgänge *bMotorSpeed1, bMotorSpeed2* und *bMotorSpeed3* für die Zeitdauer von *tDelaySpeed3ToSpeed2* auf FALSE um die Geschwindigkeit des Motors bei der Umschaltung in Stufe 2 zu drosseln.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

Dokumente hierzu

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

3.1.8 FB_HVACRedundancyCtrl



Anwendung

Dieser Funktionsbaustein dient zum Ansteuern einer bestimmten Anzahl von Aktoren, z.B. von Pumpen, aus einem Pool von 8 Aktoren. Die Logik sucht sich aus allen freigegebenen Aktoren diejenigen aus, die die geringste Laufzeit aufweisen und schaltet diese entsprechend der Laufzeitreihenfolge ein. Dies erfolgt solange, bis die über *iCountCtrl* vorgegebene Anzahl erreicht ist. Intern werden die Aktoren über einen FIFO-Speicher geführt, so dass die Aktoren in derselben Reihenfolge wieder abgeschaltet werden. Mit der Zeitvorgabe *uiMaxOnLevel* oder durch den Trigger *bForceChnage*, kann ein Umschalten im Betrieb erfolgen. Es wird der am längsten angesteuerte Aktor abgeschaltet und derjenige mit der geringsten Laufzeit dazugeschaltet. Um bei Pumpen einen hydraulischen Schlag zu vermeiden, kann über *tOverlap1Actuator* eine Überlappungszeit vorgeben werden. Diese Zeit gilt nur für den Fall, wenn ein Wechsel zwischen zwei Aktoren stattfindet.

Für die Ermittlung der Laufzeiten wird intern der Funktionsbaustein FB_HVACWork [▶ 480] instanziert und die IN-OUT-Variablen nach außen weitergeben (*rHours* und *uiCount*). Die Betriebszeiterfassung wird über die **Feedback** Eingänge angesteuert. Sollte kein Rückmeldungssignal vom Aktor zur Verfügung stehen, so muss der Aktor-Ausgang auf den Feedback-Eingang zurückgeführt werden.

VAR_INPUT

VAR_INPUT

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

```
bForceChange              : BOOL;
dtSystemTime              : DT;
bSysTakt1Hz               : BOOL;
bResetAllHours            : BOOL;
bResetAllCounter          : BOOL;
bReset                    : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

iCountCtrl: Anzahl der einzuschaltenden Aktoren (0..8).

bEn1 - 8: Freigabe für den entsprechenden Aktor.

bFeedBack1 - 8: Betriebsrückmeldung vom Aktor. Eine Auswertung findet nur statt, wenn *tNoFeedbActuator* > *t#0s* ist.

bForceChange: Eine positive Flanke an dem Eingang schaltet den ersten im FIFO befindlichen Aktor aus, und schaltet den Aktor aus dem Pool mit der geringsten Laufzeit zu.

dtSystemTime: Systemzeit.

bSysTakt1Hz: Taktsignal mit 1Hz als Ersatz für dtSystemTime, sollte dtSystemTime nicht vorhanden sein, oder länger als 2s keine Wertänderung aufweisen, wird als Ersatz das Taktsignal genutzt.

bResetAllHours: Setzt alle Betriebsstundenzähler zurück.

bResetAllCounter: Setzt alle Einschaltzähler zurück.

bReset: Quittierungseingang bei einer Störung.

VAR_IN_OUT

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

rHours[1..8]: Betriebsstunden [h] mit einer Auflösung von 1/100 Stunden (intern mit 1s). Die Variable wird persistent gespeichert.

uiCount[1..8]: Einschaltzyklenzähler. Die Variable wird persistent gespeichert.

tOverlap1Actuator: Überlappungszeit für den Fall, wenn ein Wechsel zwischen zwei Aktoren stattfindet (0ms..1min). Die Variable wird persistent gespeichert. Voreingestellt auf 20s.

uiMaxOnLevel: Zeit in Stunden [h] die ein Aktor max. Eingeschaltet sein darf (0h..1000h). Erzwingt einen Aktorenwechsel nur, wenn ein Aktor zum Einschalten zur Verfügung steht. Die Variable wird persistent gespeichert. Voreingestellt auf 200h.

tNoFeedbActuator: Zeit die vergehen darf bis die fehlende Betriebsrückmeldung vom Aktor zum *bErrorGeneral* = TRUE führt (0ms..60s). Ist die Zeit gleich 0, erfolgt keine Auswertung. Die Variable wird persistent gespeichert. Voreingestellt auf 3s.

VAR_OUTPUT

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

bQ1 - 8: Aktoreinsignal.

eErrorCode: Zeigt an, welcher Aktor keine Betriebsrückmeldung innerhalb der vorgegebenen Zeitspanne gezeigt hat. Die Erkennung dieser Fehlergruppe wird durch eine Zeit größer 0 in der Variable *tNoFeedbActuator* aktiviert. *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,

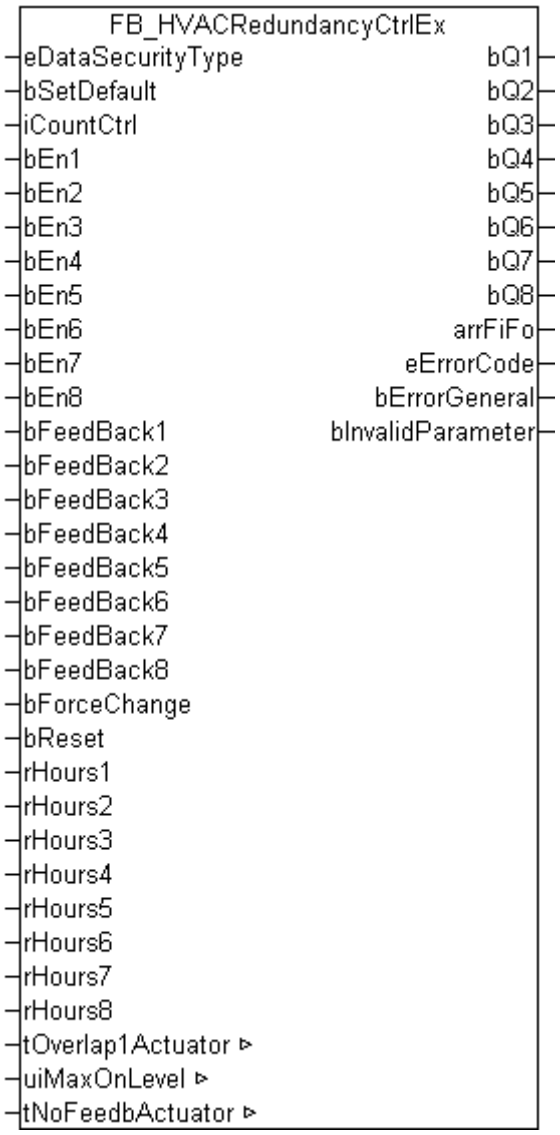
bErrorGeneral: Fehler bei der Auswertung der Betriebsrückmeldung.

bInvalidParameter: TRUE, wenn bei der Plausibilitätsüberprüfung ein Fehler aufgetreten ist. Die Meldung muss mit *bReset* quittiert werden.

Dokumente hierzu

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

3.1.9 FB_HVACRedundancyCtrlEx



Anwendung

Dieser Funktionsbaustein dient zum Ansteuern einer bestimmten Anzahl von Aktoren, z.B. von Pumpen, aus einem Pool von 8 Aktoren. Die Logik sucht sich aus allen freigegebenen Aktoren diejenigen aus, welche die geringste Laufzeit aufweisen und schaltet diese entsprechend der Laufzeitreihenfolge ein. Dies erfolgt solange, bis die über *iCountCtrl* vorgegebene Anzahl erreicht ist. Intern werden die Aktoren über einen FIFO-Speicher geführt, so dass die Aktoren in derselben Reihenfolge wieder abgeschaltet werden. Mit der Zeitvorgabe *uiMaxOnLevel* oder durch den Trigger *bForceChnage*, kann ein Umschalten im Betrieb erfolgen. Es wird der am längsten angesteuerte Aktor abgeschaltet und derjenige mit der geringsten Laufzeit dazugeschaltet. Um bei Pumpen einen hydraulischen Schlag zu vermeiden, kann über *tOverlap1Actuator* eine Überlappungszeit vorgeben werden. Diese Zeit gilt nur für den Fall, wenn ein Wechsel zwischen zwei Aktoren stattfindet.
Im Gegensatz zum FB_HVACRedundancyCtrl [▶ 57] wird für die Ermittlung der Laufzeiten kein interner Timer genutzt, sondern die Zeiten müssen als Stundenwerte von außen als Var_Input angelegt werden.

VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

iCountCtrl : Anzahl der einzuschaltenden Aktoren

bEn1 - 8: Freigabe für den entsprechenden Aktor.

bFeedBack1 - 8: Betriebsrückmeldung vom Aktor. Eine Auswertung findet nur statt, wenn *tNoFeedbActuator* > *t#0s* ist.

bForceChange: Eine positive Flanke an dem Eingang schaltet den ersten im FIFO befindlichen Aktor aus, und schaltet den Aktor aus dem Pool mit der geringsten Laufzeit zu.

rHours: Betriebsstunden

VAR_IN_OUT

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

tOverlap1Actuator: Überlappungszeit für den Fall, wenn ein Wechsel zwischen zwei Aktoren stattfindet (0ms..1min). Die Variable wird persistent gespeichert. Voreingestellt auf 20s.

uiMaxOnLevel: Zeit in Stunden die ein Aktor max. Eingeschaltet sein darf (0h..1000h). Erzwingt einen Aktorenwechsel nur, wenn ein Aktor zum Einschalten zur Verfügung steht. Die Variable wird persistent gespeichert. Voreingestellt auf 200h.

tNoFeedbActuator: Zeit die vergehen darf bis die fehlende Betriebsrückmeldung vom Aktor zum *bErrorGeneral* = TRUE führt (0ms..60s). Ist die Zeit gleich 0, erfolgt keine Auswertung. Die Variable wird persistent gespeichert. Voreingestellt auf 3s.

VAR_OUTPUT

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

bQ1 - 8: Aktoreinsignal.

arrFiFo: Tabelle des mit den Informationen welcher Aktor eingeschaltet ist (Aktornummer). Die Reihenfolge gibt die Ausschaltfolge an. arFifo[1] = Nr. des Aktors der als nächster ausgeschaltet wird.

eErrorCode: Zeigt an, welcher Aktor keine Betriebsrückmeldung innerhalb der vorgegebenen Zeitspanne gezeigt hat. Die Erkennung dieser Fehlergruppe wird durch eine Zeit größer 0 in der Variable *tNoFeedbActuator* aktiviert. *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: Fehler bei der Auswertung der Betriebsrückmeldung.

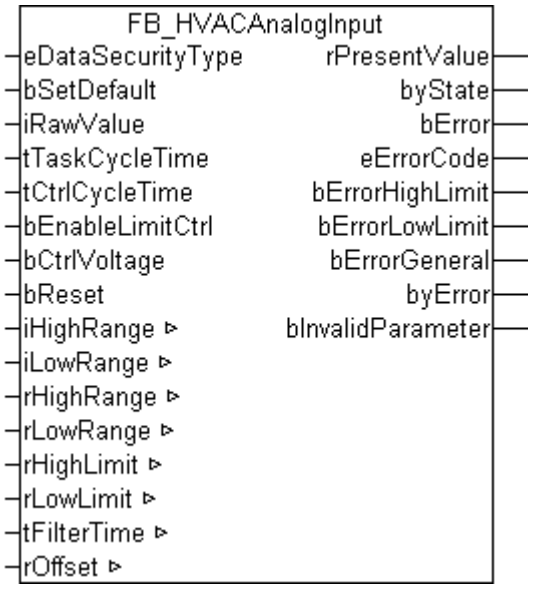
bInvalidParameter: TRUE, wenn bei der Plausibilitätsüberprüfung ein Fehler aufgetreten ist. Die Meldung muss mit *bReset* quittiert werden.

Dokumente hierzu

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

3.2 HLK Analogmodule

3.2.1 FB_HVACAnalogInput



Anwendung

Dieser Funktionsbaustein dient zur Erfassung und Skalierung von analogen Eingangssignalen. Mittels der Klemmen KL30xx, KL31xx und KL32xx können die Normsignale 0-20mA , 4-20mA, 0-10Volt und 10- 5000 Ohm erfasst und in physikalische Werte umgerechnet werden.

VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

iRawValue: Mit dem Parameter iRawValue wird dem Funktionsbaustein der Rohwert von der Klemme übergeben.

tTaskCycleTime: Zykluszeit, mit der der Funktionsbaustein aufgerufen wird. Diese entspricht der Task-Zykluszeit der aufrufenden Task, wenn der Baustein in jedem Zyklus aufgerufen wird.

tCtrlCycleTime: Zykluszeit, mit der der Funktionsbaustein bearbeitet wird. Diese muss größer oder gleich der TaskCycleTime sein. Der Funktionsbaustein berechnet mit dieser Eingangsgröße intern, ob die Zustands- und Ausgangsgrößen im aktuellen Zyklus aktualisiert werden müssen.

bEnableLimitCtrl: Die Grenzwertüberwachung wird erst aktiviert, wenn die Variable *bEnableLimitCtrl* TRUE ist. So kann die Grenzwertüberwachung mit einem Timer so lange verzögert werden bis sich die Heizungs- oder Klimaanlage in einem eingeregelten Zustand befindet. Bei Raumlufttechnischen Anlagen erfolgt diese Freigabe in der Regel vom Anlagenstartprogramm. Siehe hierzu auch FB_HVACStartAirConditioning.

bCtrlVoltage: Durch die Überprüfung der Fühlerversorgungsspannung mit dem Eingang *bCtrlVoltage* werden Störmeldungen unterdrückt wenn die Versorgungsspannung der Sensoren nicht vorhanden ist. Wenn die Fühlerversorgungsspannung vorhanden ist, steht am Eingang *bCtrlVoltage* ein TRUE an.

bReset: Quittierungseingang bei einem Fehler. Ausserdem kann über diesen Eingang der Filter 2. Ordnung auf das anstehende Messsignal synchronisiert werden, so dass dieses am Ausgang *rPresentValue* ausgegeben wird.

VAR_OUTPUT

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

rPresentValue: Ermittelter Ausgangswert.

byState: Status des Bausteines.
byState.1:= TRUE, Grenzwertüberwachung ist aktiviert.
byState.7:= TRUE, Fühlerspannungsversorgung liegt an.

bError: Die Variable *bError*wird TRUE bei einem internen Fehler des Funktionsbausteines.

eErrorCode *:* Enthält den spezifischen Fehlercode im Zusammenhang mit dem *bError*.

bErrorHighLimit: TRUE, wenn der obere Grenzwert erreicht ist.

bErrorLowLimit: TRUE, wenn der untere Grenzwert erreicht ist.

bErrorGeneral: TRUE, wenn eine Einzelstörmeldung aus dem Prozess anliegt.

byError: Ausgabe der Fehler als Byte.
byError.0:= bError
byError.1:= bInvalidParameter
byError.2:= bErrorGeneral
byError.3:= bErrorLowLimit ist TRUE, wenn der untere Grenzwert unterschritten wird.
byError.4:= bErrorHighLimit ist TRUE, wenn der obere Grenzwert überschritten wird.

bInvalidParameter: TRUE, wenn bei der Plausibilitätsüberprüfung ein Fehler aufgetreten ist. Die Meldung muss mit *bReset* quittiert werden.

VAR_IN_OUT

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

iHighRange: Oberer Rohwert von der Eingangsvariable *iRawValue*. Die Variable wird persistent gespeichert. Voreingestellt auf 32767.

iLowRange: Unterer Rohwert von der Eingangsvariable *iRawValue*. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rHighRange: Der obere skalierte Messwert. Die Variable wird persistent gespeichert. Voreingestellt auf 100.

rLowRange: Der untere skalierte Messwert. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rHighLimit: Falls der skalierte Messwert größer ist als der obere Grenzwert *rHighLimit* kann ein unzulässig hoher Messwert erreicht sein. Der Funktionsbaustein meldet diesen Fehler, indem die Variable *bErrHighLimit* auf TRUE gesetzt wird. Die Variable wird persistent gespeichert. Voreingestellt auf 100.

rLowLimit: Falls der skalierte Messwert kleiner ist als der untere Grenzwert *rLowLimit* kann ein unzulässig niedriger Messwert erreicht sein. Der Funktionsbaustein meldet diesen Fehler, indem die Variable *bErrLowLimit* auf TRUE gesetzt wird. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

tFilterTime: Um starke Schwankungen und Sprünge des Messsignals zu vermeiden, ist der Funktionsbaustein mit zwei Filter 1. Ordnung versehen. Beide Filter arbeiten mit der gleichen Zeitkonstante. Die Filterkonstanten werden mit der Variable *tFilterTime* [s] bestimmt (0..3600). Die Variable wird persistent gespeichert. Voreingestellt auf 2s.

rOffset: Mit dem Offset wird die mittels der zwei Punkte zur Umrechnung ermittelte Geradengleichung parallel nach unten oder oben verschoben. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

Mit den zwei Wertepaaren *iHighRange/rHighRange* und *iLowRange/rLowRange* erfolgt eine lineare Umwandlung des Rohwertes in die physikalische Einheit. *iHighRange* und *iLowRange* entsprechen den Rohwerten. *rHighLimit* und *rLowLimit* entsprechen den skalierten Werten in der physikalischen Einheit des zu messenden Signals.

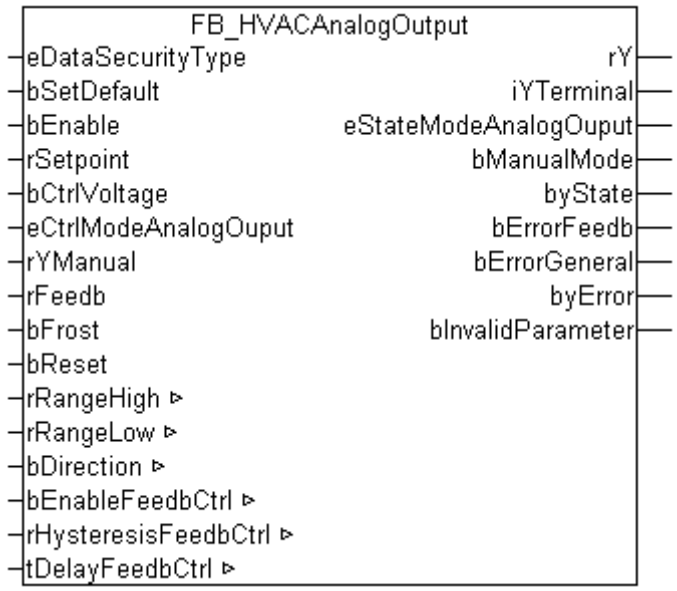
Der Ausgangswert ergibt sich aus:

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

Dokumente hierzu

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

3.2.2 FB_HVACAnalogOutput



Anwendung

Dieser Funktionsbaustein dient zur Ansteuerung stetiger Stellorgane wie beispielsweise Ventile oder Klappen mit einer Stellungsrückmeldung.

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: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Freigabe des Bausteins, wenn *bEnable* = TRUE ist. Ohne Freigabe des Bausteins wird am Ausgang *rY* der Wert Null ausgegeben.

rSetpoint: Mit der Variablen *rSetpoint* wird dem Funktionsbaustein der Sollwert für den Analogausgang übergeben.

bCtrlVoltage: Die Steuerspannung steht an, wenn die Variable *bCtrlVoltage* TRUE ist. Bei einem Ausfall der Steuerspannung wird die Rückmeldungskontrolle deaktiviert damit keine Fehlalarme entstehen.

eCtrlModeAnalogOutput: Enum, das die Betriebsart festlegt.
TYPE E_HVACAnalogOutputMode :
(
eHVACAnalogOutputMode_Auto_BMS := 0,
eHVACAnalogOutputMode_Manual_BMS := 1,
eHVACAnalogOutputMode_Auto_OP := 2,
eHVACAnalogOutputMode_Manual_OP := 3
);
END_TYPE

rYManual: Analoger Eingangswert, der in Handbetrieb an den Ausgang rY weitergeleitet wird.

rFeedb: Analoge Stellungsrückmeldung vom Stellorgan.

bFrost: Dieser Eingang dient der Frostschutzfunktion eines Lufterhitzers. Sobald der Eingang *bFrost* TRUE ist, wird rY auf die maximale Größe (*rRangeHigh*) und *iYTerminal* auf 32767 gesetzt. Der Ausgang *rY*und iYTerminal bleiben solange gesetzt bis der Eingang *bFrost* wieder FALSE ist.

bReset: Quittierungseingang bei einer Störung.

VAR_OUTPUT

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

rY: Zeigt die aktuelle Größe des Stellsignals in % an.

iYTerminal: *S*tellt die Größe des Ausgangssignals skaliert auf den Wertebereich von 0 bis 32767 dar.

eStateModeAnalogOutput: Enum, das die Betriebsart anzeigt.

bManualMode: Der Analogausgang befindet sich im manuellen Betriebsmodus.

byState: Statusbyte über den Betriebszustand des Funktionsbausteines.
byState.0 := *bEnable;*
byState.1 := *bManualMode;*
byState.2 := *bFrost;*
byState.7 := *bCtrlVoltage;*

bErrorFeedb: Fehler Rückführsignal.

bErrorGeneral: Ist eine Sammelmeldung aller Störungen des Funktionsbausteins.

byError: Liefert alle Fehlermeldungen und Warnungen.
byError.1 := bInvalidParameter
byError.2 := bErrorGeneral
byError.3 := bErrorFeedb

bInvalidParameter: TRUE, wenn bei der Plausibilitätsüberprüfung ein Fehler aufgetreten ist. Die Meldung muss mit *bReset* quittiert werden.

VAR_IN_OUT

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

rRangeHigh/rRangeLow: Mit diesen beiden Variablen wird der Wertebereich von *rY* definiert (0%..100%). Die Variable wird persistent gespeichert.

Beispiel: rSetpoint = 100, rRangeLow = 0, rRangeHigh = 200,
==> rY = 50
==> rYTerminal = 16384

rSetpoint = 200, rRangeLow = 0, rRangeHigh = 200,
==> rY = 100
==> rYTerminal = 32767

bDirection: Mit der Variablen *bDirection* wird das Ausgangssignal von *rY* invertiert. FALSE entspricht dem direkten Wirksinn. Die Variable wird persistent gespeichert.

bEnableFeedbCtrl: Stetige Stellantriebe besitzen oft eine Stellungsrückmeldung. Mittels der Stellungsrückmeldung wird die Funktion des Stellantriebs überwacht. Die Überwachung der Stellungsrückmeldung ist aktiviert, wenn Die Variable *bEnableFeedbCtrl* TRUE ist. Die Variable wird persistent gespeichert.

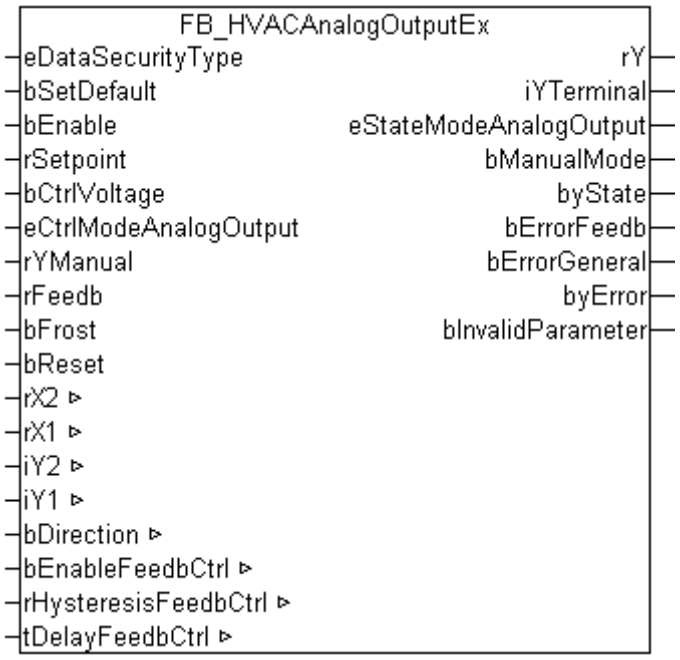
rHysteresisFeedbCtrl: Bedingt durch die Verfahrzeit typischer Antriebe in der Heizungs- und Klimatechnik ist bei einem Sollwertsprung für die Position des Stellantriebs, das Rückführsignal nacheilend. Mit der Variablen *rHysteresisFeedbCtrl* wird ein Bereich festgelegt innerhalb dessen der Positionssollwert *rY* des Stellorgans vom Rückführsignal abweichen darf (0..32767). Die Variable wird persistent gespeichert.

tDelayFeedbCtrl: Ist die Abweichung zwischen der Soll- und Istposition des Stellantriebs größer als +/- *rHysteresisFeedbCtrl*, dann wird das Ansprechen des Ausgangs *bErrorFeedb* um die Zeit des Timers *tDelayFeedbCtrl* [s]verzögert (0s..50s). Die Variable wird persistent gespeichert.

Dokumente hierzu

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

3.2.3 FB_HVACAnalogOutputEx



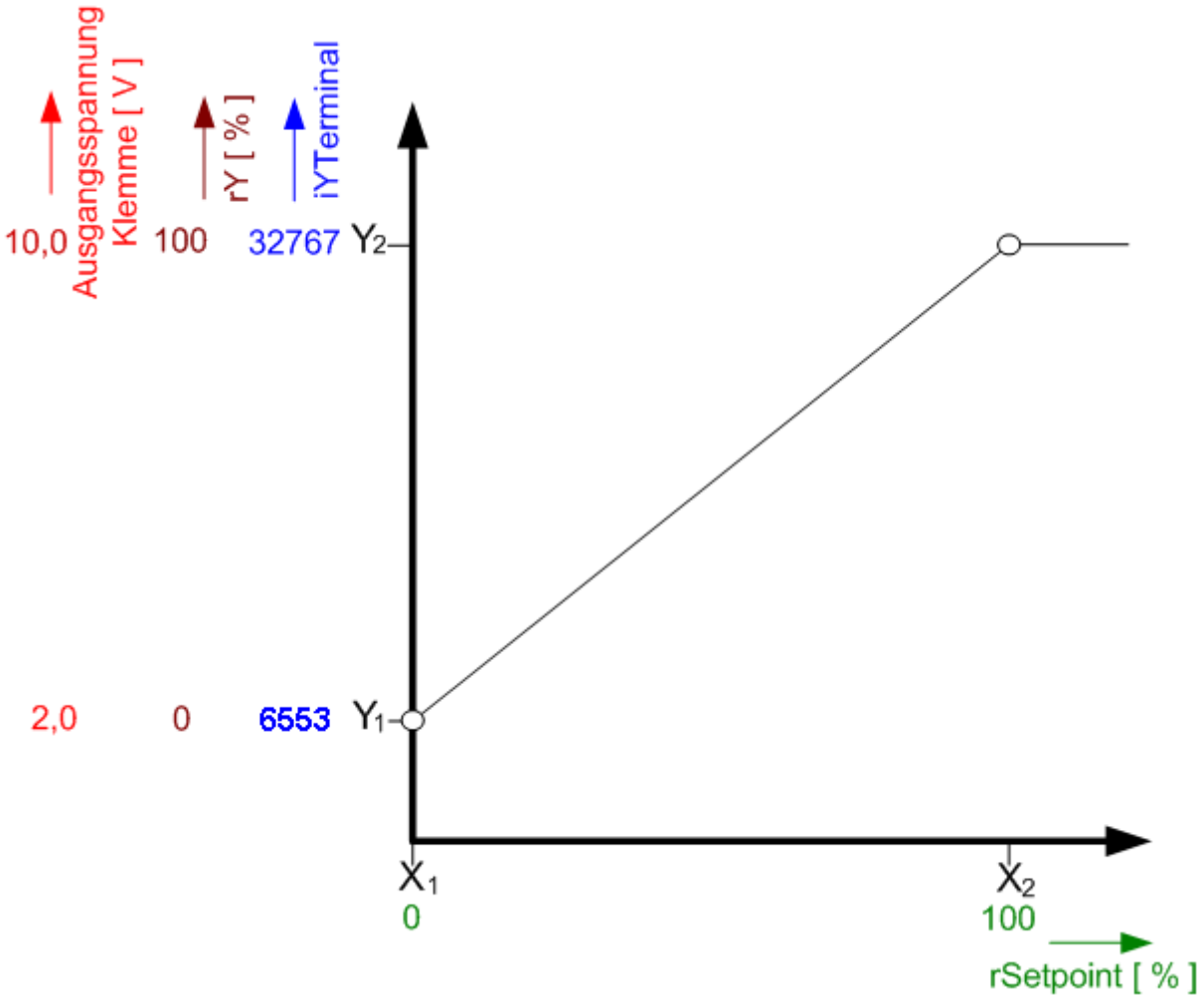
Anwendung

Dieser Funktionsbaustein dient zur Ansteuerung stetiger Stellorgane wie beispielsweise Ventile oder Klappen mit einer Stellungsrückmeldung.
Im Vergleich zu dem FB_HVACAnalogOutput ist in diesem Funktionsbaustein eine Skalierungsfunktion integriert. Die Funktion lautet: y=m*x +b.

Anwendungsbeispiel:

Klappenstellantrieb Arbeitsbereich 2-10V mit der KL4404 (Signalspannung 0-10Volt)

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



VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Freigabe des Bausteins, wenn *bEnable* = TRUE ist. Ohne Freigabe des Bausteins wird am Ausgang *rY* der Wert Null ausgegeben.

rSetpoint: Mit der Variablen *rSetpoint* wird dem Funktionsbaustein der Sollwert für den Analogausgang übergeben.

bCtrlVoltage: Die Steuerspannung steht an, wenn die Variable *bCtrlVoltage* TRUE ist. Bei einem Ausfall der Steuerspannung wird die Rückmeldungskontrolle deaktiviert damit keine Fehlalarme entstehen.

eCtrlModeAnalogOutput: Enum, das die Betriebsart festlegt.
TYPE E_HVACAnalogOutputMode :
(
eHVACAnalogOutputMode_Auto_BMS := 0,
eHVACAnalogOutputMode_Manual_BMS := 1,
eHVACAnalogOutputMode_Auto_OP := 2,
eHVACAnalogOutputMode_Manual_OP := 3
);
END_TYPE

rYManual: Analoger Eingangswert, der in Handbetrieb an den Ausgang rY weitergeleitet wird.

rFeedb: Analoge Stellungsrückmeldung vom Stellorgan.

bFrost: Dieser Eingang dient der Frostschutzfunktion eines Lufterhitzers. Sobald der Eingang *bFrost* TRUE ist, wird rY auf die maximale Größe (*rRangeHigh*) und *iYTerminal* auf 32767 gesetzt. Der Ausgang *rY*und iYTerminal bleiben solange gesetzt bis der Eingang *bFrost* wieder FALSE ist.

bReset: Quittierungseingang bei einer Störung.

VAR_OUTPUT

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

rY: Zeigt die aktuelle Größe des Stellsignals in % an (0%..100%).

iYTerminal: *S*tellt die Größe des Ausgangssignals skaliert auf den Wertebereich von 0 bis 32767 dar.

eStateModeAnalogOutput: Enum, das die Betriebsart anzeigt.

bManualMode: Der Analogausgang befindet sich im manuellen Betriebsmodus.

byState: Statusbyte über den Betriebszustand des Funktionsbausteines.
byState.0 := *bEnable;*
byState.1 := *bManualMode;*
byState.2 := *bFrost;*
byState.7 := *bCtrlVoltage;*

bErrorFeedb: Fehler Rückführsignal.

bErrorGeneral: Ist eine Sammelmeldung aller Störungen des Funktionsbausteins.

byError: Liefert alle Fehlermeldungen und Warnungen.
byError.1 := bInvalidParameter
byError.2 := bErrorGeneral
byError.3 := bErrorFeedb

bInvalidParameter: TRUE, wenn bei der Plausibilitätsüberprüfung ein Fehler aufgetreten ist. Die Meldung muss mit *bReset* quittiert werden.

VAR_IN_OUT

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

rX2: Oberer Grenzwert an der X-Achse (-32767..32767). Die Variable wird persistent gespeichert.

rX1: Unterer Grenzwert an der X-Achse (-32767..32767). Die Variable wird persistent gespeichert.

rY2: Oberer Grenzwert an der Y-Achse (0..32767). Die Variable wird persistent gespeichert.

rY1: Unterer Grenzwert an der Y-Achse (0..32767). Die Variable wird persistent gespeichert.

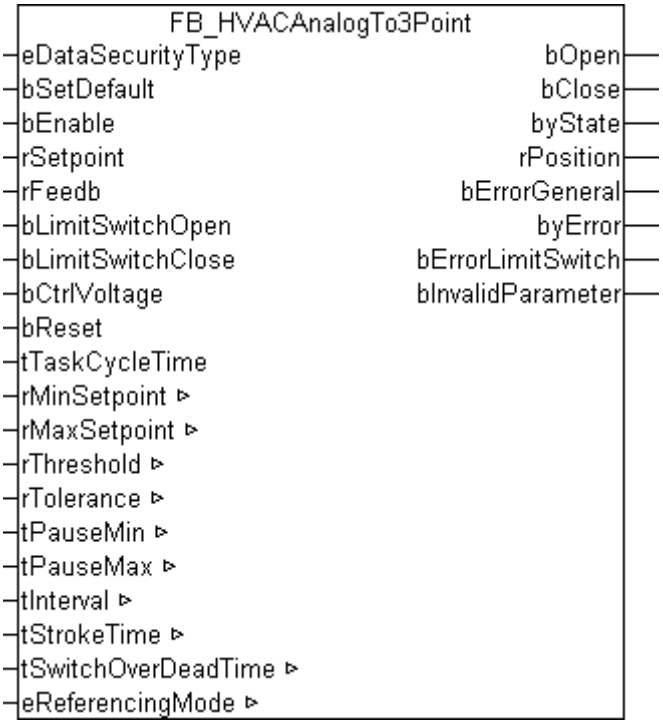
bDirection: Mit der Variablen *bDirection* wird das Ausgangssignal von *rY* invertiert. FALSE entspricht dem direkten Wirksinn. Die Variable wird persistent gespeichert.

bEnableFeedbCtrl: Stetige Stellantriebe besitzen oft eine Stellungsrückmeldung. Mittels der Stellungsrückmeldung wird die Funktion des Stellantriebs überwacht. Die Überwachung der Stellungsrückmeldung ist aktiviert, wenn Die Variable *bEnableFeedbCtrl* TRUE ist. Die Variable wird persistent gespeichert.

rHysteresisFeedbCtrl: Bedingt durch die Verfahrzeit typischer Antriebe in der Heizungs- und Klimatechnik ist bei einem Sollwertsprung für die Position des Stellantriebs, das Rückführsignal nacheilend. Mit der Variablen *rHysteresisFeedbCtrl* wird ein Bereich festgelegt innerhalb dessen der Positionssollwert *rY* des Stellorgans vom Rückführsignal abweichen darf (0..32767). Die Variable wird persistent gespeichert.

tDelayFeedbCtrl: Ist die Abweichung zwischen der Soll- und Istposition des Stellantriebs größer als +/- *rHysteresisFeedbCtrl*, dann wird das Ansprechen des Ausgangs *bErrorFeedb* um die Zeit des Timers *tDelayFeedbCtrl* verzögert (0s..50s). Die Variable wird persistent gespeichert.

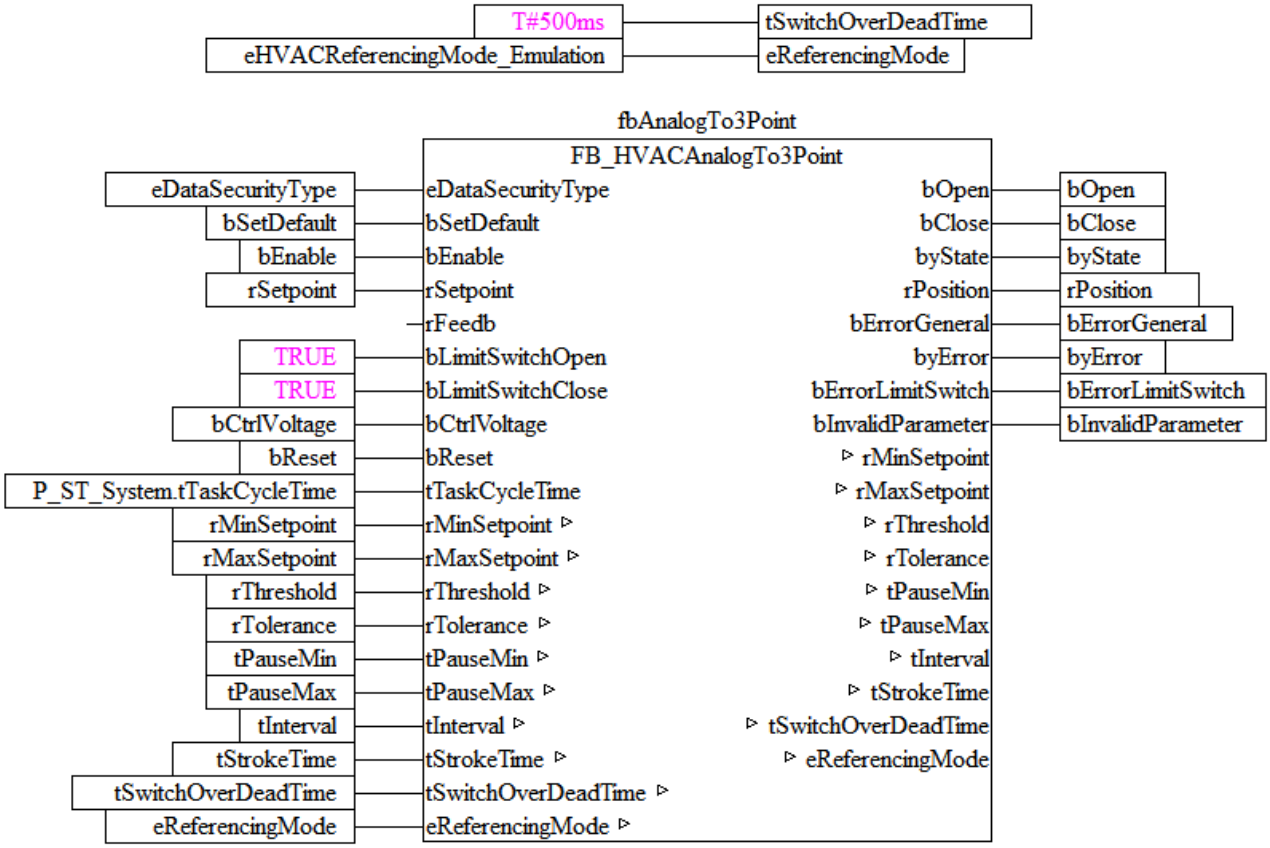
3.2.4 FB_HVACAnalogTo3Point



Anwendung

Dieser Funktionsbaustein dient zur Umwandlung eines analogen Stellsignals in ein Dreipunktschrittsignal. Damit können Dreipunktklappen oder -ventile von einem Regler mit einem stetigen Stellsignal angesteuert werden. Der Funktionsbaustein *FB_HVACAnalogTo3Point* funktioniert mit oder ohne einer stetigen Stellungsrückmeldung des Antriebs.

Beispiel: 3-Punkt-Antrieb ohne Feedback und Endlagenschalter



VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Freigabe des Bausteines, wenn *bEnable* = TRUE ist. Bei anliegender Steuerspannung, keinem Fehler und nicht vorhandener Freigabe *bEnable* = FALSE werden die Ausgänge *bClose* **=** TRUE und *bOpen* = FALSE. Wenn der Referenzmodus *eReferencingMode* = *eHVACReferencingMode_Emulation* ist, dann bleibt der Ausgang *bClose* solange gesetzt, bis entweder der Endlagenschalter angefahren ist, die Freigabe wieder da ist oder eine Störung anliegt.
Wenn der Referenzmode *eReferencingMode* = *eHVACReferencingMode_AnalogFeedback* ist, dann bleibt der Ausgang *bClose* solange gesetzt, bis die Freigabe wieder da ist, eine Störung anliegt oder *rFeedback* = *rMinSetpoint* ist.

rSetpoint: Mit der Variable *rSetpoint*wird dem Funktionsbaustein der Sollwert für die Lage der Klappe oder des Ventils übergeben. Mit den Parametern *rMaxSetpoint* und *rMinSetpoint* wird der Wertebereich von *rSetpoint*festgelegt.
Liegt ein nicht korrekter Variablenwert an *rSetpoint* an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit *rMinSetpoint* intern weiter gearbeitet. *bInvalidParameter* wird bei falschem Variablenwert gesetzt, der Funktionsbaustein arbeitet normal weiter.

rFeedb: Die Eingangsvariable *rFeedb* ist nur aktiv, wenn der Referenzierungsmodus des Funktionsbausteins *eReferencingMode* = *eHVACReferencingMode_AnalogFeedback* ist. Mit dem skalierten Eingangssignal *rFeedb*wird die tatsächliche Position des Antriebs zurückgemeldet. Die Skalierung von *rFeedb* muss übereinstimmend mit der Skalierung von *rSetpoint* sein. Mit den Parametern *rMaxSetpoint* und *rMinSetpoint* wird der Wertebereich von *rFeedb* festgelegt. *rFeedb* wird über die Ausgangsvariable *rPosition* an die Steuerung zurückgegeben.
Liegt ein nicht korrekter Variablenwert an *rFeedb* an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit *rMinSetpoint* intern weiter gearbeitet. *bInvalidParameter* wird bei falschem Variablenwert gesetzt, der Funktionsbaustein arbeitet normal weiter.

bLimitSwitchOpen/bLimitSwitchClose: Besitzt der Dreipunktstellantrieb Endlagenschalter, dann können diese an den Eingängen *bLimitSwitchOpen* und *bLimitSwitchClose* angeschlossen werden.
Die Endlagenschalter dienen in dem Referenzierungsmodus *eReferencingMode* = *eHVACReferencingMode_Emulation* dem Funktionsbaustein zur Referenzierung. Je nach Fahrtrichtung wird mit einer fallenden Flanke an einem der beiden Eingänge *bLimitSwitchOpen/bLimitSwitchClose*die aktuell errechnete Position *rPositon* automatisch auf den dementsprechenden Wert von *rMaxSetpoint/rMinSetpoint* gesetzt. Dieses geschieht auch im Stillstand des Antriebs.
Egal in welchem Referenzierungsmodus sich der Funktionsbaustein befindet, bleibt nach dem Erreichen des jeweiligen Endlagenschalters der dazu gehörige Ausgang *bOpen*oder *bClose* auf TRUE.
Die Endlagenschalter müssen als Öffnerkontakt an den Funktionsbaustein angelegt werden. Wenn keine vorhanden sind, muss an den beiden Eingängen *bLimitSwitchOpen/bLimitSwitchClose* ein TRUE angelegt werden.

bCtrlVoltage: Durch die Eingangsvariable *bCtrlVoltage* wird überprüft, ob die Steuerspannung anliegt. Bei einem Ausfall der Steuerspannung werden die beiden Ausgänge *bOpen* und *bClose* auf FALSE gesetzt. Da viele Störmeldungen nach dem Ruhestromprinzip aufgebaut sind, würde nach dem Ausfall der Steuerspannung ein Meldeschauer von Störmeldungen stattfinden. Deshalb werden bei einem Ausfall der Steuerspannung in dem *FB_HVACAnalogTo3Point* alle Störmeldungen unterdrückt.
Die Steuerspannung ist vorhanden, wenn am Eingang *bCtrlVoltage* ein TRUE ansteht.

bReset: Quittierungseingang bei einer Störung.

tTaskCycleTime: Zykluszeit, mit der der Funktionsbaustein aufgerufen wird. Diese entspricht der Task-Zykluszeit der aufrufenden Task, wenn der Baustein in jedem Zyklus aufgerufen wird.
Liegt ein nicht korrekter Zeitwert von T#0s an *tTaskCycleTime* an, dann wird intern die *tTaskCycleTime* auf T#1ms gesetzt. *bInvalidParameter* wird bei falschem Variablenwert gesetzt.

VAR_OUTPUT

```
bOpen             : BOOL;
bClose            : BOOL;
byState           : BYTE;
rPosition         : REAL;
bErrorGeneral     : BOOL;
```

```
byError           : BYTE;
bErrorLimitSwitch : BOOL;
bInvalidParameter : BOOL;
```

bOpen: Ausgang zum Auffahren des Dreipunktantriebes.

bClose: Ausgang zum Zufahren des Dreipunktantriebes.

byState: Statusbyte über den Betriebszustand des Funktionsbausteins.
byState.0 := *bEnable*
byState.1 := *bOpen*
byState.2 := *bClose*
byState.3 := *bLimitSwitchOpen*
byState.4 := *bLimitSwitchClose*
byState.7 := *bCtrlVoltage*

rPosition: Mit *rPosition* wird entweder die gemessene oder die errechnete Position des Antriebs an die Steuerung zurück gegeben.
Wenn *eReferencingMode* = *eHVACReferencingMode_AnalogFeedback* ist, dann wird *rFeedb* über die Ausgangsvariable *rPosition* an die Steuerung zurückgegeben. Ist *rFeedb* > *rMaxSetpoint*, dann ist *rPosition* = *rMaxSetpoint* und *bInvalidParameter* wird gesetzt. Ist *rFeedb* < *rMinSetpoint*, dann ist *rPosition* = *rMinSetpoint* und *bInvalidParameter* wird gesetzt.
Wenn *eReferencingMode* = *eHVACReferencingMode_Emulation* ist, dann wird die tatsächliche Position des Antriebs anhand einer Berechnung emuliert und über *rPosition* an die Steuerung zurückgegeben.

bErrorGeneral: Die Fehlermeldung *bErrorGeneral* wird TRUE, sobald *bErrorGeneralLimitSwitch* = TRUE ist. Die Ausgänge *bOpen* und *bClose* werden dann auf FALSE gesetzt und erst wieder fei gegeben, wenn der Fehler behoben ist und mit *bReset* quittiert wurde.

byError: Liefert alle Fehlermeldungen und Warnungen des Funktionsbausteines
byError.1 := *bInvalidParameter*
byError.2 := *bErrorGeneral*
byError.3 := *bErrorLimitSwitch*

bErrorLimitSwitch: Wird TRUE, wenn entweder beide Endlagenschalter gleichzeitig anstehen oder ((rMaxSetpoint - rThreshold) < rSetpoint) und bOpen = TRUE) oder ((rMinSetpoint + rThreshold) > rSetpoint) und bClose = TRUE) ist. Der Fehler *bErrorLimitSwitch* kann nur auftreten, wenn *eReferencingMode* = *eEmulation* ist.

bInvalidParameter: Zeigt an, dass ein falscher Parameter an einer der Variablen *rMinSetpoint*, *rMaxSetpoint, rThreshold*, *rTolerance*, *tPauseMin*, *tPauseMax*, *tInterval*, *tStrokeTime*, *tSwitchOverDeadTime*, *eReferencingMode*, *rSetpoint* oder *rFeedback* anliegt. Eine falsche Parameterangabe führt nicht zum Stillstand des Bausteines, siehe Beschreibung der Variablen. Nach Behebung der falschen Parameterangabe muss die Meldung *bInvalidParameter* mit *bReset* quittiert werden.

VAR_IN_OUT

VAR_IN_OUT

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

rMinSetpoint/**rMaxSetpoint:** Mit den Parametern *rMaxSetpoint* (0..32767) und *rMinSetpoint* (0..32767) wird der Wertebereich von *rSetpoint* festgelegt. *rMaxSetpoint* muss größer als *rMinSetpoint* sein. Außerdem werden die beiden Variablen *rMaxSetpoint* und *rMinSetpoint* bei der Berechnung der variablen Puls-Pausenmodulation eingebunden, siehe Bild 1.1.
Liegt ein falscher Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, wird mit dem Voreinstellwert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt, der Funktionsbaustein arbeitet normal weiter. Die Variablen werden persistent gespeichert. *rMinSetpoint* voreingestellt auf 0. *rMaxSetpoint* voreingestellt auf 100.

rThreshold/rTolerance: Ist die Abweichung zwischen dem Positionssollwert *rSetpoint* und dem errechneten oder gemessenen Positionsistwert *rPosition* des Stellorgans größer als der mit der Variable *rThreshold* (0,001..32767) eingestellte Schwellwert, dann beginnt der Funktionsbaustein abhängig von dem Betrag der Regelabweichung durch Takten der Ausgänge *bOpen* oder *bClose* die Position zu korrigieren. Die Korrektur erfolgt so lange, bis die Abweichung kleiner als der Wert *rTolerance* (0,001..32767) ist. Mit dem Betrag von *rThreshold – rTolerance* wird eine Hysterese für die Auf- und Zufahrbewegung des Antriebs definiert. Diese ist erforderlich, damit der Antrieb nicht bei kleinsten Stellwertänderungen reagiert. Es wird damit der Verschleiß des Antriebs und der Relais reduziert. *rThreshold* muss größer als *rTolerance* sein.
rSetpoint - rPosition > = *rThreshold* Ansteuerung von *bOpen*
rPosition- rSetpoint > = *rThreshold* Ansteuerung von *bClose*
Liegt ein falscher Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Voreinstellwert weitergearbeitet.
bInvalidParameter wird bei falscher Parameterangabe gesetzt, der Funktionsbaustein arbeitet normal weiter. Die Variablen werden persistent gespeichert. *rThreshold* voreingestellt auf 5,0. *rTolerance* voreingestellt auf 2,0.

tPauseMin/tPauseMax/tInterval: Mit den Parametern *tPauseMin* [s] (0..3600), *tPauseMax* [s] (0..3600)und *tInterval* [s] (0..3600) kann eine variable Puls-Pausenmodulation zum Takten des Dreipunktantriebs eingestellt werden. Durch die Parameter *tPauseMin* und *tPauseMax* wird die Pausenlänge als Funktion der Regelabweichung definiert. Bei einer kleinen Abweichung ist die Pausenzeit im Verhältnis zur Intervallzeit lang. Analog ist die Pausenzeit bei einer großen Abweichung kurz. Damit ist die effektive Verfahrgeschwindigkeit des Dreipunktantriebes bei großen Regelabweichungen größer als bei kleinen, siehe nachfolgendes Bild. Folgende Bedingung muss auf jeden Fall erfüllt sein: *tPauseMin* < *tPauseMax* < *tInterval*
Liegt ein falscher Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Voreinstellwert weitergearbeitet.
bInvalidParameter wird bei falscher Parameterangabe gesetzt, der Funktionsbaustein arbeitet normal weiter. Die Variablen werden persistent gespeichert. *tPauseMin* voreingestellt auf 2s. *tPauseMax* voreingestellt auf 8s. *tInterval* voreingestellt auf 10s.

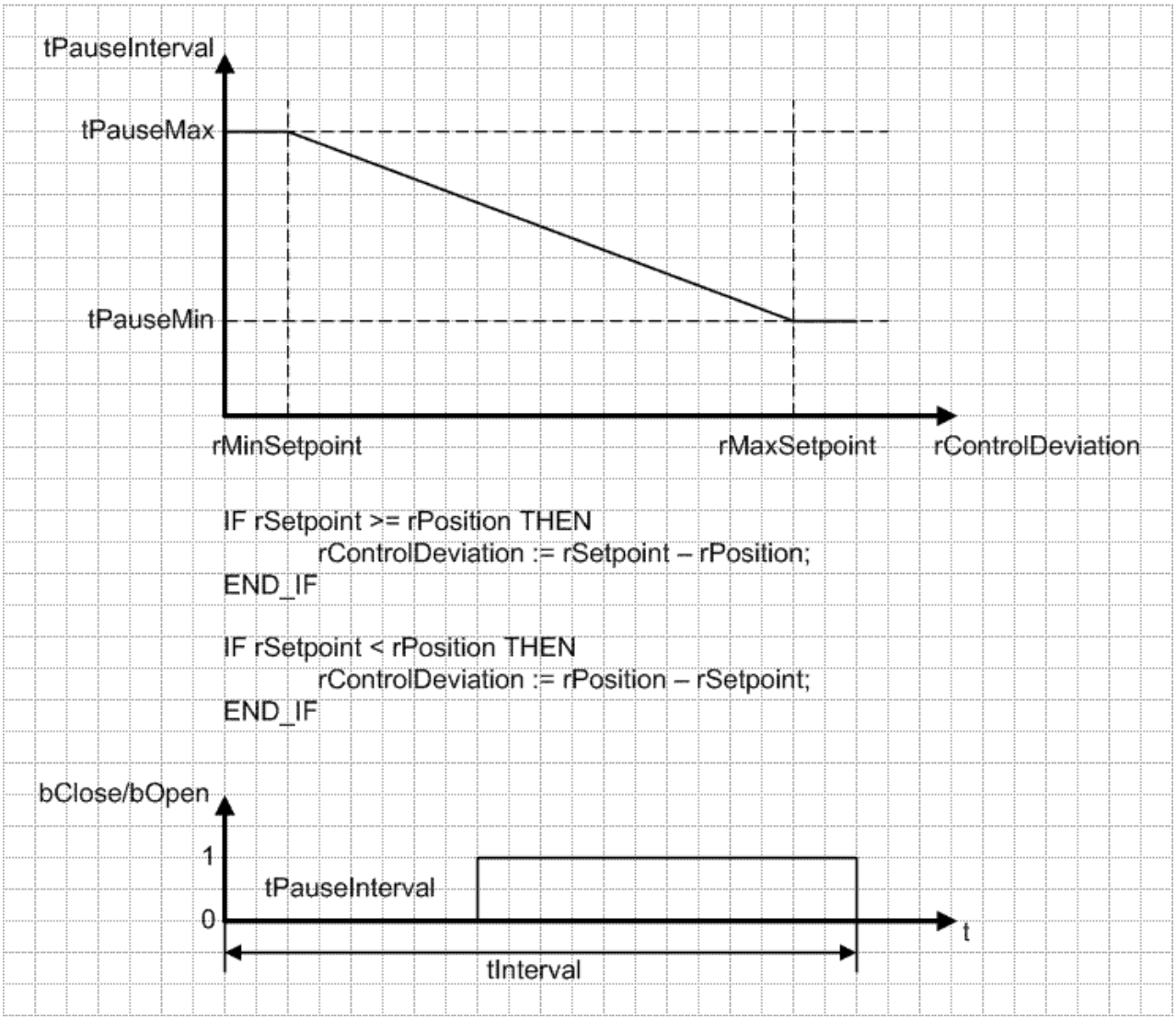
tStrokeTime: Die Variable *tStrokeTime* [s] gibt die komplette Verfahrzeit des Antriebs an. Wenn der Antrieb keine stetige Stellungsrückmeldung besitzt, wird die tatsächliche Position des Antriebs anhand einer Berechnung emuliert. Eine genaue Eingabe der Gesamtverfahrzeit des Stellorgans ist deshalb wichtig. Folgende Bedingung muss auf jeden Fall erfüllt sein: *tStrokeTime* > *tInterval*
Liegt ein falscher Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Voreinstellwert weitergearbeitet.
bInvalidParameter wird bei falscher Parameterangabe gesetzt, der Funktionsbaustein arbeitet normal weiter. Die Variable wird persistent gespeichert. Voreingestellt auf 200s.

tSwitchOverDeadTime: Verweildauer beim Richtungswechsel (0..3600). Während dieser Zeit sind beide Ausgänge zurückgesetzt.
Liegt ein falscher Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Voreinstellwert weitergearbeitet.
bInvalidParameter wird bei falscher Parameterangabe gesetzt, der Funktionsbaustein arbeitet normal weiter. Die Variable wird persistent gespeichert. Voreingestellt auf 500ms.

eReferencingMode: Enum, über das der Referenzierungsmodus des Bausteins vorgegeben wird.
Je nach Ausstattung des verwendeten Stellantriebs erfolgt das Referenzieren der Position in Abhängigkeit von *eReferencingMode*.
Ist *eReferencingMode* = *eHVACReferencingMode_Emulation,* so wird die Position des Antriebs anhand der Ansteuerung von *bOpen* und *bClose* mittels *tStrokeTime* errechnet. Mit zunehmenden Betriebsstunden des Dreipunktventils oder der Dreipunktklappe können jedoch Abweichungen durch mechanische Ungenauigkeiten im Antrieb entstehen. Um einen automatischen Abgleich zwischen der Ist- und der errechneten Position zu erreichen, wird einer der Ausgänge *bOpen* oder *bClose* bei Erreichen der errechneten Position *rPosition* von *rMaxSetpoint* oder *rMinSetpoint* auf TRUE gesetzt und sie erhält dementsprechend den Wert des Parameters *rMaxSetpoint* oder *rMinSetpoint*. Eine fallende Flanke an den Signaleingängen *bLimitSwitchOpen*und *bLimitSwitchClose* wird für die Referenzierung der Position des Antriebs auf *rMaxSetpoint* oder *rMinSetpoint* genutzt. Nach dem Erreichen des jeweiligen Endlagenschalters wird der dazu gehörige Ausgang *bOpen*oder *bClose* auf TRUE gesetzt bleiben.
Ist *eReferencingMode* = *eHVACReferencingMode_AnalogFeedback* so wird die Position des Antriebs anhand des Signals *rFeedback* übergeben. Nach dem Erreichen des jeweiligen Endlagenschalters wird der dazu gehörige Ausgang *bOpen*oder *bClose* auf TRUE gesetzt bleiben.
Liegt ein falscher Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen.

Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Voreinstellwert weitergearbeitet.
bInvalidParameter wird bei falscher Parameterangabe gesetzt, der Funktionsbaustein arbeitet normal weiter.
Die Variable wird persistent gespeichert. Voreingestellt auf 0.

Bild 1.1



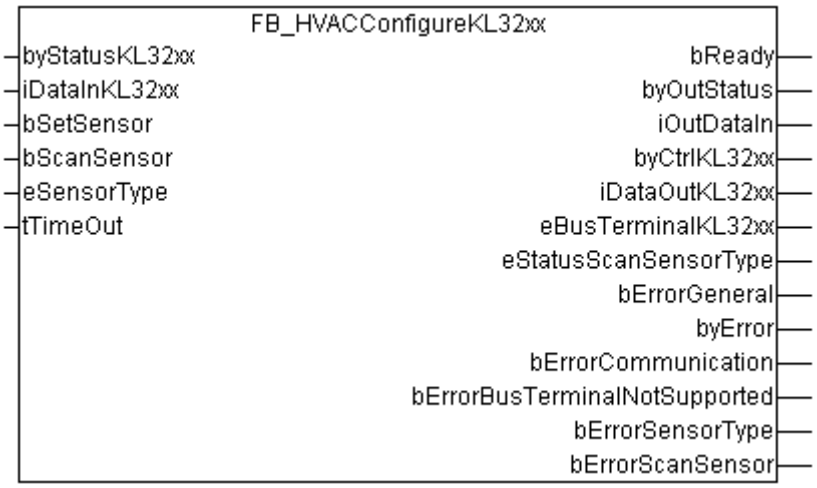
Beispiel

tInterval	**120 Sekunden**
tPauseMin	10 Sekunden
tPauseMax	60 Sekunden
rSetpoint	100%
rPosition	50%
tPauseInterval / bClose OR bOpen	35 Sekunden/85 Sekunden
rSetpoint	50%
rPosition	25%
tPauseInterval / bClose OR bOpen	47,5 Sekunden/72,5 Sekunden

Dokumente hierzu

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

3.2.5 FB_HVACConfigureKL32xx



Anwendung

Mit dem Funktionsbaustein *FB_HVACConfigureKL32xx* kann aus der PLC heraus der Sensortyp eines Kanals der KL3201/02/04, KL3208-0010 oder der KL3228 eingestellt werden. Dazu müssen die Variablen *byStatusKL32xx, iDataInKL32xx, byCtrlKL32xx* und *iDataOutKL32xx* im TwinCAT System Manager mit den Variablen Status, Daten Ein, Kontrolle und Daten Aus eines Kanals der KL3201/02/04 oder der KL3228 verknüpft sein.
Mit dem Enum *eSensorType* können die in Tabelle 1 vorgegebenen Widerstandselemente ausgewählt werden.
Die Konfigurationssoftware KS2000 wird für das Einstellen des Temperatursensors nicht mehr benötigt.

Bei der Widerstandsmessung 10 bis 5000Ω der KL32xx entspricht 1 Digit = 0,5Ω, d.h. der angezeigte Rohwert muss in der PLC durch 2 dividiert werden. Beispiel: 2500Ω würden in der Steuerung mit einem Rohwert von 5000 dargestellt werden. Der Rohwert muss in der PLC durch 2 dividiert werden um auf den ohmschen Wert von 2500Ω zu kommen.

Die Widerstandsmessung 10-10000Ω ist nur mit der Sonderklemme KL320x-0027 möglich. An dieser Sonderklemme kann ausschließlich nur die Widerstandsmessung 10...10000Ω betrieben werden.

Die EtherCAT-Klemme EL3692 ist eine Widerstandsmessklemme, die den Messbereich bis 10MΩ abdeckt.

Anwendungsbeispiel

Download	Benötige Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

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

byStatusKL32xx: Hier muss das Statusbyte (Status) der Busklemme im TwinCAT System Manager für den jeweiligen Kanal zugewiesen werden. Ist dies nicht der Fall, so wird mit *bErrorCommunication* dieses als Fehler angezeigt, wenn mit einer steigenden Flanke an *bSetSensor*/*bScanSensor* der Funktionsbaustein aktiviert wurde. Ist das Scannen oder Setzen mit einer steigenden Flanke an *bSetSensor*/*bScanSensor*

abgeschlossen, so wird dieses mit einem TRUE an *bReady* angezeigt und *byOutStatus := byStatusKL32xx.* Ist das Scannen oder Setzen aktiviert, *bReady* = FALSE, so wird an *byOutStatus* der Wert von *byStatusKL32xx* ausgegeben, der vor der Aktivierung anstand.

iDataInKL32xx: Hier muss der Rohwert (Daten Ein) der Busklemme im TwinCAT System Manager für den jeweiligen Kanal zugewiesen werden. Ist dies nicht der Fall, so wird mit *bErrorCommunication* dieses als Fehler angezeigt, wenn mit einer steigenden Flanke an *bSetSensor*/*bScanSensor* der Funktionsbaustein aktiviert wurde. Ist das Scannen oder Setzen mit einer steigenden Flanke an *bSetSensor*/*bScanSensor* abgeschlossen, so wird dieses mit einem TRUE an *bReady* angezeigt und *iOutDataIn := iDataInKL32xx .* Ist das Scannen oder Setzen aktiviert, *bReady* = FALSE, so wird an *iOutDataIn* der Wert von *iDataInKL32xx* ausgegeben, der vor der Aktivierung anstand.

bSetSensor: Mit einer steigenden Flanke an *bSetSensor* wird der am Enum *eSensorType* vorgegebene Sensortyp in der Busklemme eingestellt. Ist der Vorgang abgeschlossen, wird intern im Baustein das Scannen aktiviert und über das Enum *eStatusScanSensorType* wird angegeben, welcher Sensortyp eingestellt wurde.

bScanSensor: Mit einer steigenden Flanke an *bScanSensor* wird das Scannen des jeweiligen Kanals der Busklemme aktiviert. Über das Enum *eStatusScanSensorType* wird angegeben, welcher Sensortyp eingestellt wurde.

eSensorType: Enum mit welchem der Sensortyp für den jeweiligen Kanal für das Setzen über *bSetSensor* angegeben wird.

tTimeOut: Gibt die Zeit bis zum Abbruch der Funktion an und *bErrorCommunication* wird TRUE. Wird keine Zeit angegeben, so wird intern im Baustein mit T#5s gearbeitet.

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: Wird der Funktionsbaustein über eine steigende Flanke an *bSetSensor/bScanSensor* aktiviert, so wird *bReady* = FALSE. Ist der Vorgang des Scannen oder Setzen des Sensortypen abgeschlossen, wird *bReady* = TRUE.

byOutStatus: Ist das Scannen oder Setzen mit einer steigenden Flanke an *bSetSensor*/*bScanSensor* abgeschlossen, so wird dies mit einem TRUE an *bReady* angezeigt und *byOutStatus := byStatusKL32xx.* Ist das Scannen oder Setzen aktiviert, *bReady* = FALSE, so wird an *byOutStatus* der Wert von *byStatusKL32xx* ausgegeben, der vor der Aktivierung anstand.

iOutDataIn: Ist das Scannen oder Setzen mit einer steigenden Flanke an *bSetSensor*/*bScanSensor* abgeschlossen, so wird dies mit einem TRUE an *bReady* angezeigt und *iOutDataIn := iDataInKL32xx .* Ist das Scannen oder Setzen aktiviert, *bReady* = FALSE, so wird an *iOutDataIn* der Wert von *iDataInKL32xx* ausgegeben, der vor der Aktivierung anstand.

byCtrlKL32xx: Hier muss das Kontrollbyte (Kontrolle) der Busklemme im TwinCAT System Manager für den jeweiligen Kanal zugewiesen werden. Ist dies nicht der Fall, so wird mit *bErrorCommunication* dieses als Fehler angezeigt, wenn mit einer steigenden Flanke an *bSetSensor*/*bScanSensor* der Funktionsbaustein aktiviert wurde.

iDataOutKL32xx: Hier muss die Datenausgabe (Daten Aus) der Busklemme im TwinCAT System Manager für den jeweiligen Kanal zugewiesen werden. Ist dies nicht der Fall, so wird mit *bErrorCommunication* dieses als Fehler angezeigt, wenn mit einer steigenden Flanke an *bSetSensor*/*bScanSensor* der Funktionsbaustein aktiviert wurde.

eBusTerminalKL32xx: Enum, welches den Busklemmentyp anzeigt, wenn an *bSetSensor/bScanSensor* mit einer steigenden Flanke der Funktionsbaustein aktiviert wurde.

eStatusScanSensorType: Enum, welches den Sensortyp des jeweiligen Kanals anzeigt, wenn an *bSetSensor/bScanSensor* mit einer steigenden Flanke der Funktionsbaustein aktiviert wurde.

bErrorGeneral: Wird TRUE, sobald entweder *bErrorCommunication, bErrorBusTerminalNotSupported, bErrorSensorType* oder *bErrorScanSensor* TRUE ist.

byError: Liefert alle Fehlermeldungen und Warnungen des Funktionsbausteines
byError.2 := *bErrorGeneral*;
byError.3 := *bErrorCommunication*;
byError.4 := *bErrorBusTerminalNotSupported*;
byError.5 := *bErrorSensorType*;
byError.6 := *bErrorScanSensor*;

bErrorCommunication: Wird TRUE, wenn z.B. die Variablen *byCtrlKL32xx, iDataOutKL32xx, byStatusKL32xx* und *iDataInKL32xx* nicht einem Kanal der Busklemme zugewiesen wurden. Beim Scannen oder Setzen des Sensortyps ist dann die Kommunikation zur Busklemme unterbrochen. bErrorCommunication wird ebenfalls TRUE, wenn die *tTimeOut*-Zeit beim Scannen oder Setzen des Sensortypen nicht ausreichend ist.

bErrorBusTerminalNotSupported: Wird TRUE, wenn die Busklemme von dem Funktionsbaustein nicht unterstützt wird.

bErrorSensorType: Wird TRUE, wenn der vorgegebene Sensortyp an *eSensorType* von der Busklemme nicht unterstützt wird.

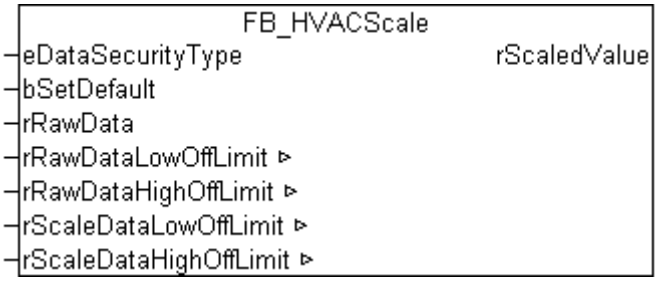
bErrorScanSensor: Wird TRUE, wenn beim Scannen der Sensortyp nicht von den im Enum *eBusTerminalKL32xx* angegebenen Busklemmen unterstützt wird.

Tab. 1: Tabelle 1: Sensorarten auswählbar über das Enum eTemperatureCharacteristic

Sensorart	KL3201/2/4-0000		KL3201/2/4-0010		KL3201/2/4-0012		KL3201/2/4-0014		KL3201/2/4-0016		KL3201/2/4-0020	
		Rohwert		Rohwert		Rohwert		Rohwert		Rohwert		Rohwert
PT100	**x**	**1/10°C**	**x**	**1/10°C**	**x**	**1/10°C**	**x**	**1/10°C**	**x**	**1/10°C**	**x**	**1/10°C**
PT200	**x**	**1/10°C**	**x**	**1/10°C**	**x**	**1/10°C**	**x**	**1/10°C**	**x**	**1/10°C**	**x**	**1/10°C**
PT500	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C
PT1000	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C
Ni100	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C
Ni120	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C
Ni1000(DIN)	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C
Ni1000(Tk5000,LS)	**not supported**		**not supported**		**not supported**		**not supported**		**not supported**		**not supported**	
Widerstandsmessung 10 ... 1200Ω	**x**		**x**		**x**		**x**		**x**		**x**	
Widerstandsmessung 10 ... 5000Ω	**x**		**x**		**x**		**x**		**x**		**x**	

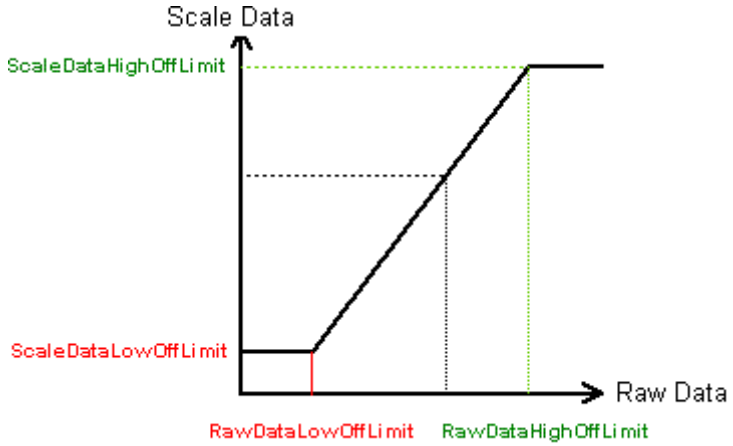
Sensorart	KL3201/2/4-0000		KL3201/2/4-0010		KL3201/2/4-0012		KL3201/2/4-0014		KL3201/2/4-0016		KL3201/2/4-0020	
		Rohwert		Rohwert		Rohwert		Rohwert		Rohwert		Rohwert
PT100	**x**	**1/10°C**	**x**	**1/10°C**	**x**	**1/10°C**	**x**	**1/10°C**	**x**	**1/10°C**	**x**	**1/10°C**
PT200	**x**	**1/10°C**	**x**	**1/10°C**	**x**	**1/10°C**	**x**	**1/10°C**	**x**	**1/10°C**	**x**	**1/10°C**
Widerstandsmessung 10 ... 10000 Ω	**not supported**		**not supported**		**not supported**		**not supported**		**not supported**		**not supported**	
Sensorart	KL3201/2/4-0023		KL3201/2/4-0025		KL3201/2/4-0029		KL3201/2/4-0031		KL3228		KL3208-0010	
		Rohwert		Rohwert		Rohwert		Rohwert		Rohwert		Rohwert
PT100	x	1/10°C	x	1/10°C	x	1/10°C	x	1/10°C	not supported		not supported	
PT200	x	1/10°C	x	1/10°C	x	1/10°C	x	1/10°C	not supported		not supported	
PT500	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**not supported**		**not supported**	
PT1000	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/100°C
Ni100	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**not supported**		**not supported**	
Ni120	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**not supported**		**not supported**	
Ni1000(DIN)	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/10°C	**x**	1/100°C
Ni1000(Tk5000,LS)	**not supported**		**not supported**		**x**	1/10°C	**not supported**		**x**	1/10°C	**x**	1/100°C
Widerstandsmessung 10 ... 1200Ω	**x**		**x**		**x**		**x**		**not supported**		**not supported**	
Widerstandsmessung 10 ... 5000Ω	**x**		**x**		**x**		**x**		**not supported**		**not supported**	
Widerstandsmessung 10 ... 10000 Ω	**not supported**		**not supported**		**not supported**		**not supported**		**not supported**		**not supported**	

3.2.6 FB_HVACScale



Anwendung

Ein analoger Rohwert wird auf den angegebenen Messbereich skaliert und als Funktionswert zurückgegeben. Überschreitet der Rohwert den oberen oder unteren Messbereich, so wird der entsprechende Grenzwert ausgegeben. Zwischen dem oberen und unteren Grenzwert der Rohdaten, muss mindestens ein Unterschied von 0.01 vorhanden sein. Ist dieses nicht der Fall, so wird der untere Grenzwert ausgegeben. Der Unterschied zwischen den Grenzen ist notwendig, um bei der internen Berechnung der Geradengleichung eine Division durch Null zu vermeiden.



VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType [▶ 529]:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

rRawData: Rohwert von der Klemme.

VAR_OUTPUT

```
rScaledValue          :  REAL;
```

rScaledValue: Skalierter Wert.

VAR_IN_OUT

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

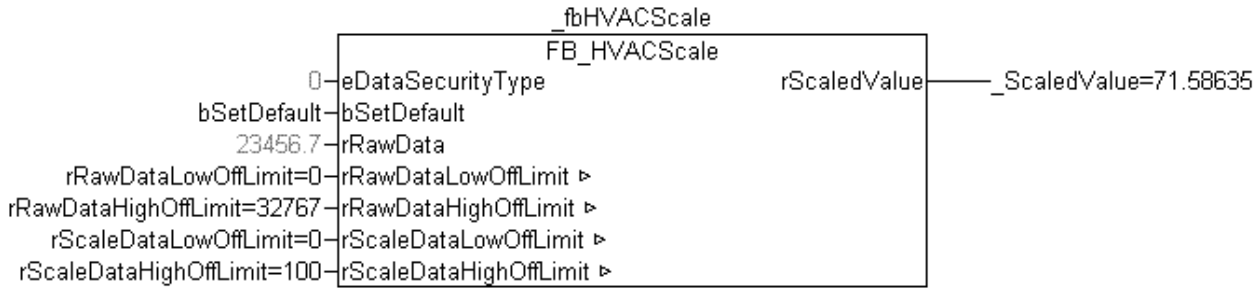
rRawDataLowOffLimit: Unterer Grenzwert vom Rohwert. Voreingestellt auf 0.

rRawDataHighOffLimit: Oberer Grenzwert vom Rohwert. Voreingestellt auf 32767.

rScaleDataLowOffLimit: Unterer Grenzwert vom skalierten Messwert. Voreingestellt auf 0.

rScaleDataHighOffLimit: Oberer Grenzwert vom skalierten Messwert. Voreingestellt auf 100.

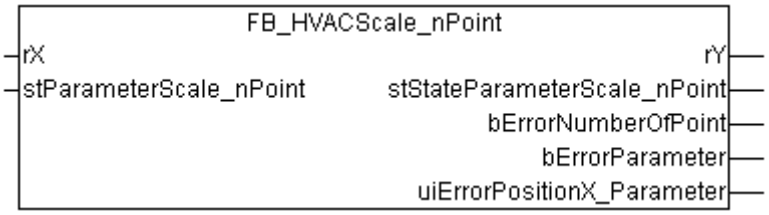
Beispiel in FUP:



Dokumente hierzu

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

3.2.7 FB_HVACScale_nPoint



Anwendung

Mit dem Funktionsbaustein *FB_HVACScale_nPoint* können Kennlinien in der SPS nachgebildet werden. Diese können z.B. positive oder negative Temperaturkoeffizienten (PTC/NTC) sein. Der analoge Widerstandsrohwert eines PTC/NTC wird an den Eingang *rX* angelegt und über den Ausgang *rY* als Temperaturwert ausgegeben. Die einzelnen Parameter und die Anzahl der einzelnen Punkte der X-Y-Achse

der Kennlinie werden mit der Struktur *stParameterScale_nPoint* vorgegeben.
Mit den Busklemmen KL32xx können in der SPS/PLC analoge Widerstandsrohwerte dargestellt werden. In der Dokumentation zum FB_HVACConfigureKL32xx [▶ 77] sind detaillierte Informationen zu finden.

Bei der Widerstandsmessung 10 bis 5000Ω der KL32xx entspricht 1 Digit = 0,5Ω, d.h. der angezeigte Rohwert muss in der PLC durch 2 dividiert werden. Beispiel: 2500Ω würden in der Steuerung mit einem Rohwert von 5000 dargestellt werden. Der Rohwert muss in der PLC durch 2 dividiert werden um auf den ohmschen Wert von 2500Ω zu kommen.

Die Widerstandsmessung 10-10000Ω ist nur mit der Sonderklemme KL320x-0027 möglich. An dieser Sonderklemme kann ausschließlich nur die Widerstandsmessung 10...10000Ω betrieben werden.

Die EtherCAT-Klemme EL3692 ist eine Widerstandsmessklemme, die den Messbereich bis 10MΩ abdeckt.

Anwendungsbeispiel

Download	Benötigte Bibliothek
TcHVAC.pro [▶ 540]	TcHVAC.lib

VAR_INPUT

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

rX: Rohwert der auf den angegebenen Messwert der Struktur *stParameterScale_nPoint* skaliert und über *rY* ausgegeben wird. Der Rohwert kann z.B. der Widerstandsrohwert der KL32xx sein. Die Klemme muss dafür auf Widerstandsmessung eingestellt sein. In der Dokumentation FB_HVACConfigureKL32xx [▶ 77] sind detaillierte Informationen zur Vorgehensweise zu finden.

stParameterScale_nPoint: Struktur über der die einzelnen Punkte der X-Y-Koordinaten ihre Wertigkeit bekommen. Egal welche Kennlinie nachgebildet werden soll, es muss entweder *stParameterScale_nPoint.rX[1]* < *stParameterScale_nPoint.rX[2]* < *stParameterScale_nPoint.rX[n]* ODER *stParameterScale_nPoint.rX[1]* > *stParameterScale_nPoint.rX[2]* > *stParameterScale_nPoint.rX[n]* sein.
stParameterScale_nPoint.iNumberOfPoint darf nicht kleiner als 2 ODER größer *g_iMaxNoOfScale_nPoint(60)* sein.
stParameterScale_nPoint.rX[1..iNumberOfPoint] - Array welches die Wertigkeit der einzelnen Punkte der X-Achse beinhaltet. Wie viele Punkte angegeben werden, ist abhängig von *iNumberOfPoint*.
stParameterScale_nPoint.rY[1..iNumberOfPoint] - Array welches die Wertigkeit der einzelnen Punkte der Y-Achse beinhaltet. Wie viele Punkte angegeben werden, ist abhängig von *iNumberOfPoint*.

VAR_OUTPUT

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

rY: *rY* ist der skalierte Temperaturwert des vorgegebenen Rohwertes *rX*.

stStateParameterScale_nPoint: Status der Struktur *stParameterScale_nPoint.* Ist *bErrorNumberOfPoint* oder *bErrorParamter* = TRUE, so ist der Status der einzelnen Parameter in der Struktur *stStateParameterScale_nPoint* = 0. Ist *stParameterScale_nPoint.iNumberofPoint* = 20, so werden die einzelnen Parameter von *stStateParameterScale_nPoint.rX[1..20] / stStateParameterScale_nPoint.rY[1..20]* angezeigt, ab 21 bis 60 ist der Status = 0.

bErrorNumberOfPoint: Ist *iNumberOfPoint* < 2 oder *iNumberOfPoint* > *g_iMaxNoOfScale_nPoint*, dann ist *bErrorNumberOfPoint* = TRUE und an der Ausgangsvariable *rY* wird eine 0 ausgegeben. Ist der Fehler behoben, so wird *bErrorNumberOfPoint* = FALSE.

bErrorParameter: Bei der Parametrierung der Struktur *stScaleTemperatureCharacteristic* muss darauf geachtet werden, dassentweder *stParameterScale_nPoint.rX[1] > stParameterScale_nPoint.rX[2] > stParameterScale_nPoint.rX[n]* ODER *stParameterScale_nPoint.rX[1] < stParameterScale_nPoint.rX[2] < stParameterScale_nPoint.rX[n]* ist. Wird dieses nicht eingehalten, so wird *bErrorParameter* = TRUE. Ist der Fehler behoben, so wird *bErrorParameter* = FALSE. Über die Ausgangsvariablen *uiErrorPositionX_Parameter* wird die genaue Feldposition in dem Array *stParameterScale_nPoint.rX[uiErrorPositionX_Parameter]* angegeben.

uiErrorPositionX_Parameter: Ist *bErrorParameter* = TRUE, so wird mit *uiErrorPositionX_Parameter* die genaue Feldposition in dem Array *stParameterScale_nPoint.rX[uiErrorPositionX_Parameter]* angegeben an welcher der Fehler aufgetreten ist.

VAR_GLOBAL CONSTANT

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

g_iMaxNoOfScale_nPoint: Globale Konstante, welche die maximale Anzahl der Punkte der X-Y-Koordinaten der Struktur ST_HVACParameterScale_nPoint [▶ 535] festlegt.

Formeln für die Geradengleichung, Zweipunkteform:

m = Steigung

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

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

Y = Y1 + m * (X - X1)

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

Beispielkennlinie 1mit Berechnung

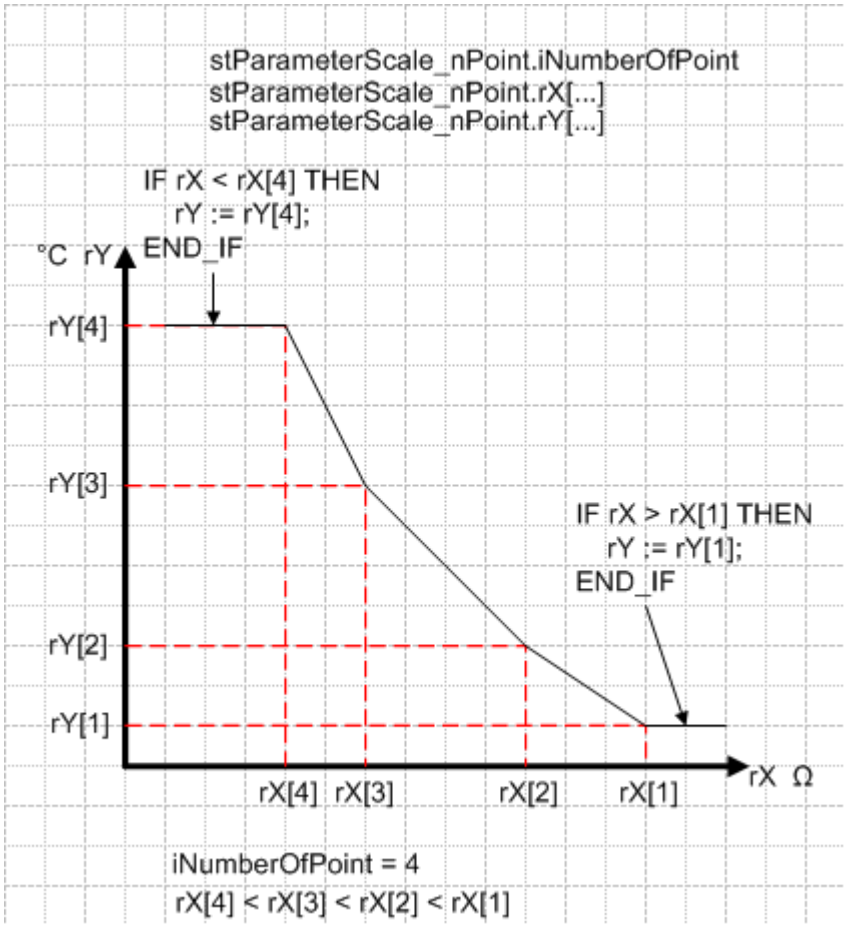


Abb. 1: FB_HVACScale_nPoint_2

m = rM = Steigung
X = rX = 1199.8
X2 = stParameterScale_nPoint.rX[2] = 1209
X3 = stParameterScale_nPoint.rX[3] = 1163
Y2 = stParameterScale_nPoint.rY[2] = 20
Y3 = stParameterScale_nPoint.rY[3] = 21

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

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

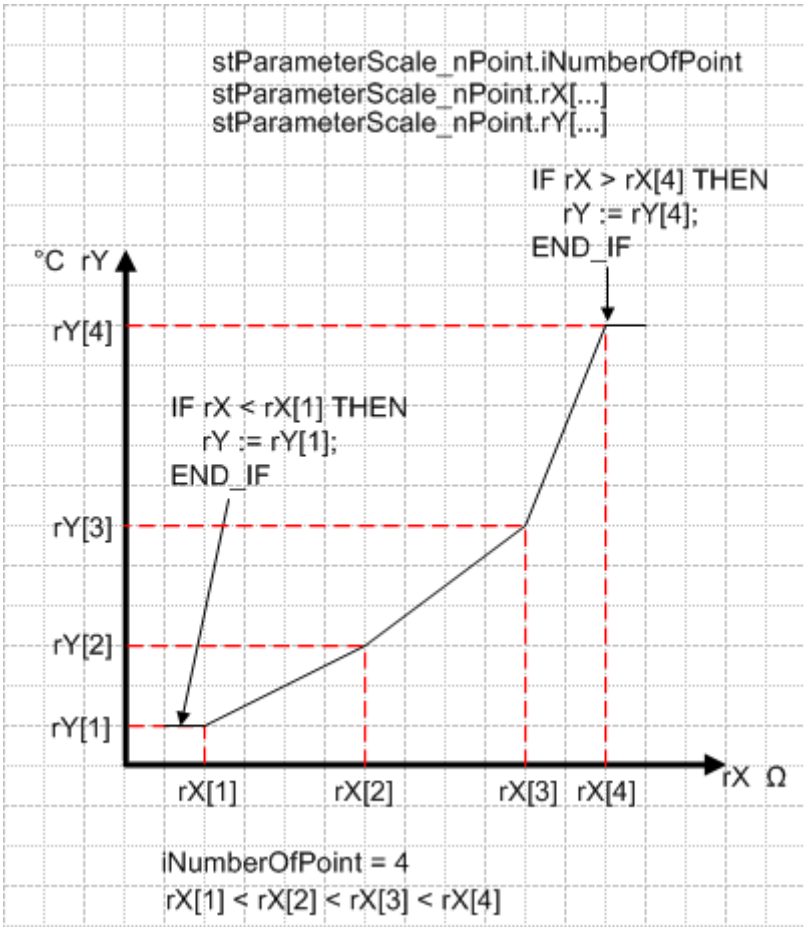
rM = (21 - 20) / (1163 - 1209) = -0.02174

Y = Y2 + m * (X - X2)

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

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

Beispielkennlinie 2mit Berechnung



m = rM = Steigung
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])

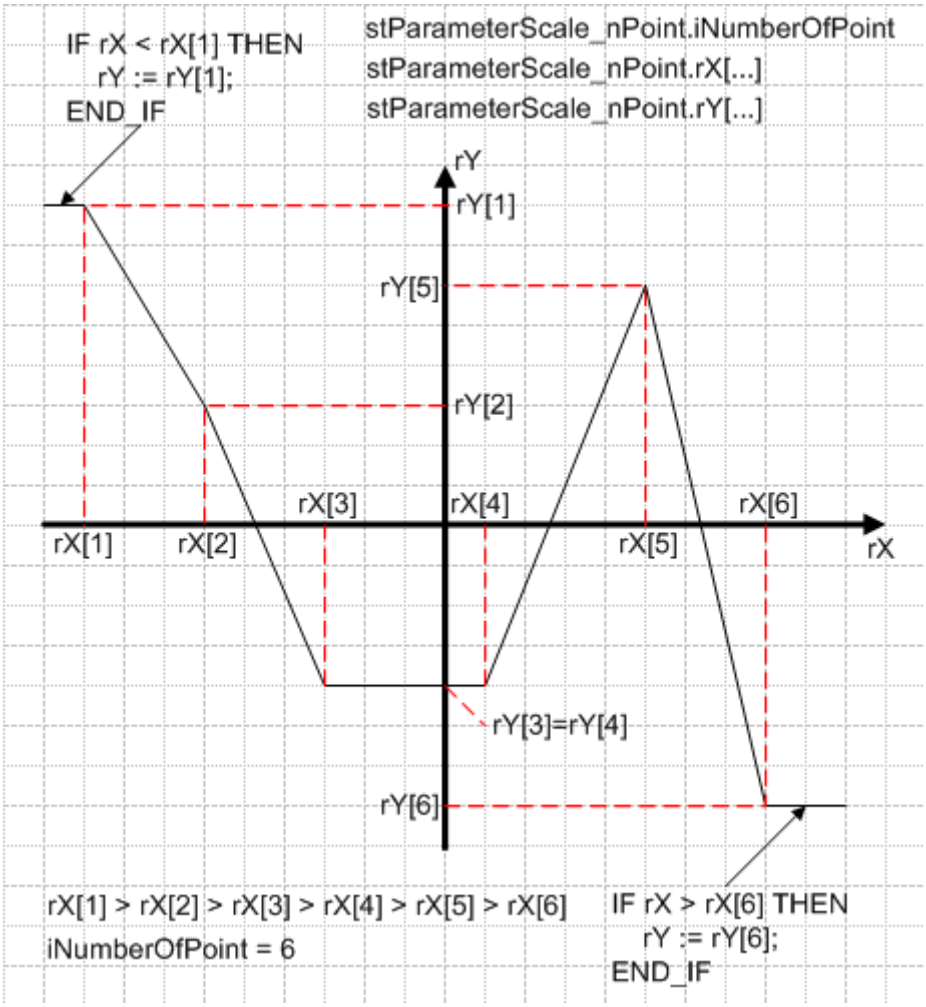
rM = (25 - 20) / 1142 - 1112) = 0.166

Y = Y3 + m * (X - X3)

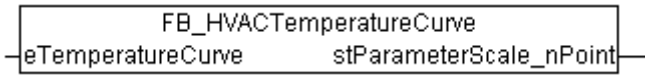
rY = stParameterScale_nPoint.rY[3] + rM * (rX - stParameterScale_nPoint.rX[3])

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

Beispielkennlinie 3



3.2.8 FB_HVACTemperatureCurve



Anwendung

Über das Enum *eTemperatureCurve* wird eine intern im Funktionsbaustein hinterlegte Temperaturkennlinie aus den Tabellen 1&2 ausgewählt und über die Struktur *stParameterScale_nPoint* ausgegeben. Mit dieser Struktur kann dann im Zusammenhang mit dem Funktionsbaustein FB_HVACScale_nPoint [▶ 82] Kennlinien nachgebildet werden. Um an den Widerstandsrohwert eines Sensors zu gelangen, müssen die Busklemmen KI32xx auf Widerstandsmessung eingestellt werden, siehe FB_HVACConfigureKL32xx [▶ 77].

Bei der Widerstandsmessung 10 bis 5000Ω der KL32xx entspricht 1 Digit = 0,5Ω, d.h. der angezeigte Rohwert muss in der PLC durch 2 dividiert werden. Beispiel: 2500Ω würden in der Steuerung mit einem Rohwert von 5000 dargestellt werden. Der Rohwert muss in der PLC durch 2 dividiert werden um auf den ohmschen Wert von 2500Ω zu kommen.

Die Widerstandsmessung 10-10000Ω ist nur mit der Sonderklemme KL320x-0027 möglich. An dieser Sonderklemme kann ausschließlich nur die Widerstandsmessung 10...10000Ω betrieben werden.

Die EtherCAT-Klemme EL3692 ist eine Widerstandsmessklemme, die den Messbereich bis 10MΩ abdeckt.

Anwendungsbeispiel

Download	Benötigte Bibliothek
TcHVAC.pro [▶ 540]	TcHVAC.lib

VAR_INPUT

```
eTemperatureCurve           : E_HVACTemperatureCurve;
```

eTemperatureCurve: Enum über welches eine intern im Funktionsbaustein hinterlegte Temperaturkennlinie aus den Tabellen 1&2 ausgewählt werden kann.

VAR_OUTPUT

```
stParameterScale_nPoint     : ST_HVACParameterScale_nPoint;
```

stParameterScale_nPoint: Struktur, die die Anzahl der Punkte und deren Wertigkeit der X-Y-Koordinaten beinhaltet. stParameterScale_nPoint beinhaltet je nach Vorgabe über das Enum eTemperatureCurve die in den Tabellen 1&2 hinterlegten Kennlinien.

Beispiel:

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;

Tab. 2: Tabelle 1: Vorgegebene Temperaturkennlinien auswählbar über das Enum eTemperatureCurve

	S+S Sensortyp 1K3 A1 NTC 1 kOhm	S+S Sensortyp 1.8K3 A1 NTC 1.8 kOhm	S+S Sensortyp 2.2K3 A1 NTC 2.2 kOhm	S+S Sensortyp 3.3K3 A1 NTC 3 kOhm	S+S Sensortyp Nl1000 DIN	S+S Sensortyp PT1000 DIN	S+S Sensortyp Ni1000/ TCR (LAN1) Ni1000Tk5 000
eHVAC-Tempera-tureCha-racteristic_	NT-C1k_3_A1	NT-C1k8_3_A1	NT-C2k2_3_A1	NT-C3k3_3_A1	Ni1000_DIN	Pt1000_DIN	Ni1000Tk5000_TCR
°C	Ω	Ω	Ω	Ω	Ω	Ω	Ω
- 50	32886				743	803	790.8
- 40	18641				791	843	826.8
- 30	11130	21695	27886	53093	842	882	871.7
- 20	6777	12987	16502	29125	893	922	913.4
- 15	5341	10153	12844	21887	920	941	934.7
- 10	4247	8011	10070	16599	946	961	956.2
- 5	3390	6347	8134	12698	973	980	978
0	2728	5071	6452	9795	1000	1000	1000
1	2613	4851	6164	9309			1004.4
2	2503	4640	5891	8849			1008.9
3	2399	4441	5631	8415			1013.3
4	2300	4252	5384	8005			1017.8
5	2205	4071	5150	7617	1028	1020	1022.3
6	2115	3899	4927	7251			1026.7
7	2030	3738	4715	6905			1031.2
8	1948	3582	4513	6575			1035.8
9	1870	3434	4321	6265			1040.3
10	1796	3294	4138	5971	1056	1039	1044.8
11	1724	3158	3964	5691			1049.3
12	1656	3029	3797	5427			1053.9
13	1590	2905	3639	5177			1058.4
14	1528	2788	3488	4938			1063
15	1469	2677	3345	4713	1084	1058	1067.6
16	1412	2570	3207	4500			1072.2
17	1358	2468	3076	4298			1076.8
18	1306	2371	2952	4104			1081.4
19	1256	2277	2832	3922			1086
20	1209	2189	2719	3747	1112	1078	1090.7
21	1163	2103	2610	3582			1095.3
22	1120	2023	2506	3426			1100
23	1078	1945	2407	3277			1104.6
24	1038	1871	2289	3135			1109.3
25	1000	1800	2200	3000	1142	1098	1114
26	963	1732	2115	2872			1120
27	928	1667	2034	2750			1123.4
28	894	1604	1957	2634			1128.1
29	862	1545	1883	2522			1132.9
30	831	1488	1812	2417	1171	1117	1137.6

	S+S Sensortyp 1K3 A1 NTC 1 kOhm	S+S Sensortyp 1.8K3 A1 NTC 1.8 kOhm	S+S Sensortyp 2.2K3 A1 NTC 2.2 kOhm	S+S Sensortyp 3.3K3 A1 NTC 3 kOhm	S+S Sensortyp NI1000 DIN	S+S Sensortyp PT1000 DIN	S+S Sensortyp Ni1000/ TCR (LAN1) Ni1000Tk5 000
eHVAC-Tempera-tureCha-racteristic_	NT-C1k_3_A1	NT-C1k8_3_A1	NT-C2k2_3_A1	NT-C3k3_3_A1	Ni1000_DIN	Pt1000_DIN	Ni1000Tk5000_TCR
°C	Ω	Ω	Ω	Ω	Ω	Ω	Ω
35	694	1235	1500	1960	1200	1136	1161.5
40	583	1031	1248	1597	1230	1155	1185.7
45	491	865	1043	1310	1261	1175	1210.2
50	416	729	876	1081	1291	1194	1235
55	354	616	738	896	1322	1213	1260.1
60	302	524	626	746	1353	1232	1285.4
65	259	447	532	625	1385	1252	1311.1
70	223	383	454	526	1417	1271	1337.1
75	192	329	390	444	1450	1290	1363.5
80	167	284	335	346	1483	1309	1390.1
85	145	246	289	321	1516	1328	1417.1
90	127	214	251	275	1549	1347	1444.4
95	111	187	218	236	1584	1366	1472
100	97	163	190	204	1618	1385	1500
105	88	143	167	176			1528.3
110	76	126	146	138	1688	1423	1557
115		111		120			1586
120		99		105	1760	1461	1625.4
125		88		92			
130		80		81	1833	1498	
140		62		64	1909	1536	
150		50		50	1987	1573	
160					2066	1611	
170					2148	1648	
180					2232	1685	
190						1722	
200						1758	
210						1795	
220						1832	
230						1868	
240						1905	
250						1941	
260						1977	
270						2013	
280						2049	
290						2085	
300						2121	
310						2156	
320						2191	

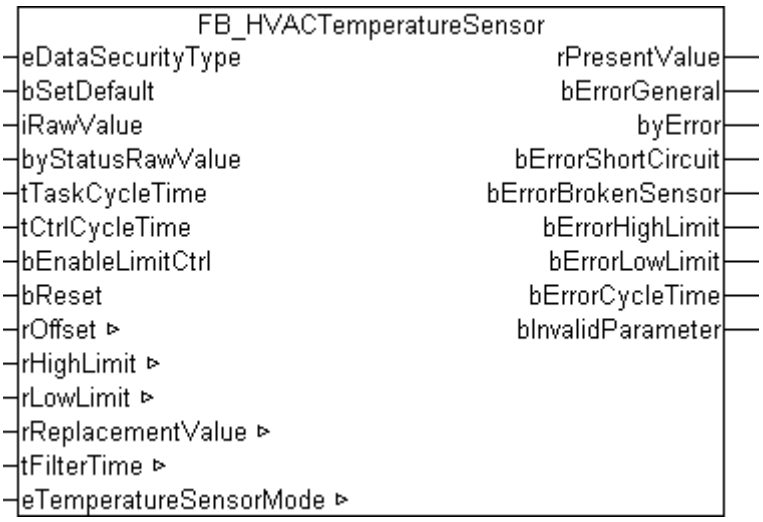
	S+S Sensortyp 1K3 A1 NTC 1 kOhm	S+S Sensortyp 1.8K3 A1 NTC 1.8 kOhm	S+S Sensortyp 2.2K3 A1 NTC 2.2 kOhm	S+S Sensortyp 3.3K3 A1 NTC 3 kOhm	S+S Sensortyp NI1000 DIN	S+S Sensortyp PT1000 DIN	S+S Sensortyp Ni1000/ TCR (LAN1) Ni1000Tk5 000
eHVAC-Tempera-tureCha-racteristic_	NT-C1k_3_A1	NT-C1k8_3_A1	NT-C2k2_3_A1	NT-C3k3_3_A1	Ni1000_DIN	Pt1000_DIN	Ni1000Tk5 000_TCR
°C	Ω	Ω	Ω	Ω	Ω	Ω	Ω
330						2227	
340						2262	
350						2297	
360						2332	
370						2367	
380						2401	
390						2436	
400						2470	

Tab. 3: Tabelle 2: Vorgegebene Temperaturkennlinien auswählbar über das Enum eTemperatureCurve

	Thermokon Sensortyp NTC 1.8 kOhm	Thermokon Sensortyp Ni1000 Tk5000
eHVACTemperatureCharacteri-stic_	NTC1k8	Ni1000Tk5000
°C	Ω	Ω
- 50		790.88
- 40		830.83
- 30	24500	871.69
- 20	14000	913.48
- 10	8400	956.24
0	5200	1000
10	3330	1044.79
20	2200	1090.65
25	1800	1113.99
30	1480	1137.61
40	1040	1185.71
50	740	1234.97
60	540	1285.44
70	402	1337.14
80	306	1390.12
90	240	1444.39
100	187	1500
110	149	1556.98
120	118	1615.36
130	95	1675.18
140	77	1736.47

	Thermokon Sensortyp NTC 1.8 kOhm	Thermokon Sensortyp Ni1000 Tk5000
eHVACTemperatureCharacteristic_	**NTC1k8**	**Ni1000Tk5000**
°C	**Ω**	**Ω**
150	64	1799.26

3.2.9 FB_HVACTemperatureSensor



Anwendung

Dieser Funktionsbaustein dient zur Erfassung und Weiterverarbeitung von Temperaturwerten für die Fühlertypen PT100, PT200, PT1000, NI100, NI120, NI1000. Der Funktionsbaustein *FB_HVACTemperatureSensor* ist abgestimmt auf die Busklemmen KL320x. Diese Busklemmen können entweder vorkonfiguriert bestellt werden oder sie können softwaremäßig auf den entsprechenden Fühlertypen eingestellt werden.

VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

iRawValue: Rohwert des Temperaturfühlers in 1/10°C von der Busklemme

byStatusRawValue: Statusbyte des Temperaturfühlers von der Busklemme. Dient zur Fehlerdiagnose, z.B. Drahtbruch oder Kurzschluss.

tTaskCycleTime: Zykluszeit, mit der der Funktionsbaustein aufgerufen wird. Diese entspricht der Task-Zykluszeit der aufrufenden Task, wenn der Baustein in jedem Zyklus aufgerufen wird. *tTaskCycleTime* muß größer als T#0ms sein.

tCtrlCycleTime: Mit der Variablen *tCtrlCycleTime* wird die Zykluszeit vorgegeben mit welcher intern der Filter 2. Ordnung abgearbeitet wird. Die Zykluszeit *tCtrlCycleTime* muß größer oder gleich der *tTaskCycleTime* sein. Ist dieses nicht der Fall, so steht eine Störung *bErrorCycleTime* an und an *rPresentValue* wird entsprechend dem eingestellten Modus *eTemperatureSensorMode* entweder der Ersatzwert *rReplacementValue* oder der letzte gültige Messwert ausgegeben.

bEnableLimitCtrl: Freigabe für die Limit-Überwachung *rHighLimit* und *rLowLimit*

bReset: Quittierungseingang bei einem Fehler. Außerdem kann über diesen Eingang der Filter 2. Ordnung auf das anstehende Messsignal synchronisiert werden, so dass dieses am Ausgang *rPresentValue* ausgegeben wird.

VAR_OUTPUT

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

rPresentValue: Temperaturausgabevariable mit einer Nachkommastellen, *rPresentValue* = (*iRawValue* / 10.0) + rOffset, wenn *bErrorShortCircuit* oder *bErrorBrokenSensor* = TRUE, dann ist *rPresentValue* abhängig von der Betriebsart *eTemperatureSensorMode.*

bErrorGeneral: Die Störungsmeldung *bErrorGeneral* wird TRUE, sobald eine der Störmeldungen *bErrorHighLimit, bErrorLowLimit, bErrorCycleTime, bErrorShortCircuit* oder *bErrorBrokenSensor* = TRUE ist. Der Wert der Ausgangsvariable *rPresentValue* ist dann abhängig von der Betriebsart *eTemperatureSensorMode* und wird frei gegeben, wenn die Störung behoben ist und je nach Betriebsart mit *bReset* quittiert wurde.

byError: Liefert alle Fehlermeldungen und Warnungen,
byError.1 := *bInvalidParameter*
byError.2 := *bErrorGeneral*
byError.3 := *bErrorHighLimit*
byError.4 := *bErrorLowLimit*
byError.5 := *bErrorShortCircuit*
byError.6 := *bErrorBrokenSensor*
byError.7 := *bErrorCycleTime*

bErrorShortCircuit: Fehler, Kurzschluss an dem angeschlossenem Temperatursensor. Nach Behebung des Fehlers ist die Quittierung abhängig von der Betriebsart.

bErrorBrokenSensor: Fehler, Drahtbruch an dem angeschlossenem Temperatursensor. Nach Behebung des Fehlers ist die Quittierung abhängig von der Betriebsart.

bErrorHighLimit: Warnung oberer Grenzwert überschritten, wird TRUE wenn *rPresentValue* >= *rHighLimit* ist. Die Warnung, dass der oberer Grenzwert überschritten ist, kann erst quittiert werden, wenn *rPresentValue* <= *rHighLimit* -1.0 für die Zeitdauer von 5 Sekunden ist.

bErrorLowLimit: Warnung unterer Grenzwert unterschritten, wird TRUE wenn *rPresentValue* <= *rLowLimit* ist. Die Warnung, dass der untere Grenzwert unterschritten ist, kann erst quittiert werden, wenn *rPresentValue* >= *rLowLimit* +1.0 für die Zeitdauer von 5 Sekunden ist

bErrorCycleTime: Fehler, der durch falsche Zeitangabe an den Eingangsvariablen *tTaskCycleTime* und *tCtrlCycleTime* entsteht und nach Behebung des Fehlers quittiert werden muß.

bInvalidParameter: Zeigt an, dass ein falscher Parameter an einer der Variablen *rOffset, rHighLimit, rLowLimit, rReplacementValue, tFilterTime und eTemperatureSensorMode* anliegt. Eine falsche Parameterangabe führt nicht zum Stillstand des Bausteins, siehe Beschreibung der Variablen. Nach Behebung der falschen Parameterangabe muss die Meldung *bInvalidParameter* mit *bReset* quittiert werden.

VAR_IN_OUT

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

rOffset: Temperaturabgleich in Kelvin (-50..+50), *rPresentValue* = (*iRawValue* / 10.0) + *rOffset.* Die Variable wird persistent gespeichert. Voreingestellt auf 0.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

rHighLimit: Oberer Grenzwert (-250..+850), wenn *rPresentValue* >= *rHighLimit* , dann wird der Ausgang *bErrorHighLimit* gesetzt. *rHighLimit* muss größer sein als *rLowLimit.* Die Variable wird persistent gespeichert. Voreingestellt auf 120.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

rLowLimit: Unterer Grenzwert (-250..+850), wenn *rPresentValue* <= *rLowLimit*, dann wird der Ausgang *bErrorLowLimit* gesetzt. Die Variable wird persistent gespeichert. Voreingestellt auf -60.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

rReplacementValue: Ersatzwert (-250..+850), der bei den Fehlern *bErrorShortCircuit* und *bErrorBrokenSensor* an den Ausgang *rPresentValue* ausgegeben wird, wenn die Betriebsart *eTemperatureSensorMode* = *eHVACTemperatureSensorMode_ReplacementValue* oder *eTemperatureSensorMode* = *eHVACTemperatureSensorMode_AutoResetReplacementValue* ausgewählt ist. Die Variable wird persistent gespeichert. Voreingestellt auf 0.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

tFilterTime: Filterkonstante (1ms..100s). Der Funktionsbaustein ist mit einem Filter 2. Ordnung versehen um starke Schwankungen und Sprünge des Messsignals zu vermeiden. Bei einem Neustart der Steuerung oder nach Behebung der Fehler *bErrorShortCircuit* oder *bErrorBrokenSensor* wird der Filter 2. Ordnung sofort und zusätzlich nach 2 Sekunden auf das anstehende Messsignal synchronisiert, so dass dieses am Ausgang *rPresentValue* ausgegeben wird. Während des laufenden Betriebes kann das Messsignal über den Eingang *bReset* synchronisiert werden. Die Variable wird persistent gespeichert. Voreingestellt auf 10s.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

eTemperatureSensorMode: Enum, über welches die Betriebsart des Bausteines vorgegeben wird. Die Variable wird persistent gespeichert. Voreingestellt auf 3.

eTemperatureSensorMode = eHVACTemperatureSensorMode_ReplacementValue: Wenn bErrorShortCircuit oder bErrorBrokenSensor = TRUE, dann ist rPresentValue = rReplacementValue. Nach Behebung des Fehlers muss der Funktionsbaustein durch eine steigende Flanke an der Eingangsvariablen bReset quittiert werden.

eTemperatureSensorMode = eHVACTemperatureSensorMode_LastValue: Wenn bErrorShortCircuit oder bErrorBrokenSensor = TRUE, dann wird an der Ausgangsvariablen rPresentValue der zuletzt gültige Temperaturwert, der 10 Sekunden vorher anstand, ausgegeben. Nach Behebung des Fehlers muss dieser durch eine steigende Flanke an der Eingangsvariablen bReset quittiert werden.

eTemperatureSensorMode = eHVACTemperatureSensorMode_AutoResetReplacementValue : Wenn bErrorShortCircuit oder bErrorBrokenSensor = TRUE, dann ist rPresentValue = rReplacementValue. Nach Behebung des Fehlers quittiert sich der Funktionsbaustein automatisch.

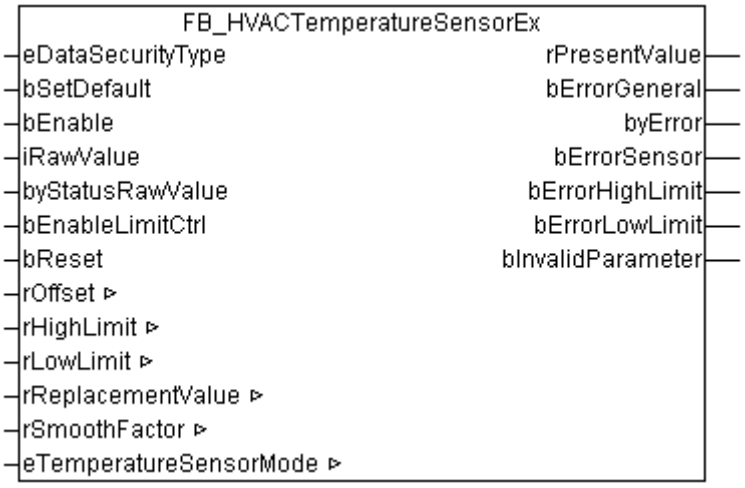
eTemperatureSensorMode = eHVACTemperatureSensorMode_AutoResetLastValue: Wenn bErrorShortCircuit oder bErrorBrokenSensor = TRUE, dann wird an der Ausgangsvariable rPresentValue der zuletzt gültige Temperaturwert, der 10 Sekunden vorher anstand, ausgegeben. Nach Behebung des Fehlers quittiert sich der Funktionsbaustein automatisch.

Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. bInvalidParameter wird bei falscher Parameterangabe gesetzt.

Dokumente hierzu

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

3.2.10 FB_HVACTemperatureSensorEx



Anwendung

Dieser Funktionsbaustein dient zur Erfassung und Weiterverarbeitung von Temperaturwerten wie z.B. für die Fühlertypen PT100, PT200, PT1000, NI100, NI120, NI1000, NI1000Tk5000. Der Funktionsbaustein *FB_HVACTemperatureSensorEx*ist abgestimmt auf die Busklemmen KL3201/02/04 und KL3228. Diese Busklemmen können entweder vorkonfiguriert bestellt werden oder softwaremäßig auf den entsprechenden Fühlertypen eingestellt werden.
Über die Eingangsvariable *iRawValue* wird der Temperaturrohwert in 1/10°C dem Funktionsbaustein übergeben und über *rPresentValue* als Fließkommazahl ausgeben. *iRawValue* kann z.B. direkt mit dem Temperaturrohwert der folgenden Busklemmen verknüpft werden: KL3201/02/04 und KL3228.

Der Ausgabewert *rPresentValue* ist abhängig von der folgenden Glättungsfunktion:

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

rPresentValueOld ist der Wert von *rPresentValue*, der einen SPS-Zyklus vorher ausgegeben wurde. Wenn *bEnable* = TRUE wird, so ist für einen SPS-Zyklus *rPresentValue* = rPresentValueOld. Ist *bErrorSensor* = TRUE, der Fehler behoben und *bErrorSensor* = FALSE, so ist für einen SPS-Zyklus *rPresentValue* = rPresentValueOld.

Über die Eingangsvariable *byStatusRawValue* wird der Status des angeschlossenen Temperatursensors überwacht und im Fehlerfall über die Variable *bErrorSensor* der Steuerung zurück geliefert. *byStatusRawValue* kann z.B. direkt mit dem Statusbyte der folgenden Busklemmen verknüpft werden: KL3201/02/04 und KL3228.
Mit *rHighLimit/rLowLimit* können Temperaturgrenzwerte festgelegt werden.

Im Gegensatz zum FB_HVACTemperatureSensor [▶ 91] hat dieser Funktionsbaustein die Eingangsvariable *bEnable*, die von Nutzen ist, wenn die Sensorenkennlinien in den Busklemmen KL3201/02/04 und KL3228 aus der SPS heraus über den Funktionsbaustein FB_HVACConfigureKL32xx [▶ 77] eingestellt werden sollen. Der Filter 2. Ordnung im FB_HVACTemperatureSensor [▶ 91] wird in diesem Funktionsbaustein durch die oben beschriebene Glättungsfunktion ersetzt. Der Ausgang *bErrorSensor* ist neu und ersetzt die beiden Ausgänge bErrorShortCircuit/bErrorBrokenSensor. Diese Ausgänge sind aber weiterhin im Errorbyte *byError* vorhanden.

Anwendungsbeispiel

Download	Benötige Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Freigabe des Bausteins, wenn *bEnable* = TRUE ist. Ist *bEnable* = FALSE, so wird am Ausgang *rPresentValue* der Wert von *rReplacementValue* ausgegeben. Sämtliche Störmeldungen und *bInvalidParameter* werden auf FALSE gesetzt. Wenn bEnable = TRUE wird, so ist für einen SPS-Zyklus *rPresentValue* = rPresentValueOld.

iRawValue: Rohwert des Temperaturfühlers in 1/10°C von der Busklemme.

byStatusRawValue: Statusbyte des Temperaturfühlers von der Busklemme. Dient zur Fehlerdiagnose, z.B. Drahtbruch oder Kurzschluss. Ist die KL32xx auf Widerstandsmessung (Ω) eingestellt, so gibt es keine Fehlerdiagnose.

bEnableLimitCtrl: Freigabe für die Limit-Überwachung *rHighLimit* und *rLowLimit*

bReset: Quittierungseingang bei einem Fehler mit einer steigenden Flanke ab *bReset*. In Abhängigkeit der Betriebsart *eTemperatureSensorMode* werden Fehler entweder mit *bReset* oder automatisch quittiert.

VAR_OUTPUT

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

rPresentValue: Temperaturausgabevariable mit einer Nachkommastelle.
Nach folgender Formel wird der Wert für *rPresentValue* berechnet und ausgegeben:

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

rPresentValueOld ist der Wert von *rPresentValue*, der einen SPS-Zyklus vorher ausgegeben wurde. Wenn bEnable = TRUE wird, so ist für einen SPS-Zyklus *rPresentValue* = rPresentValueOld. Ist *bErrorSensor* = TRUE, der Fehler behoben und *bErrorSensor* = FALSE, so ist für einen SPS-Zyklus *rPresentValue* = rPresentValueOld.
Ist *bErrorSensor* = TRUE, dann ist der Wert von *rPresentValue* abhängig von der Betriebsart *eTemperatureSensorMode.*

bErrorGeneral: Die Störungsmeldung *bErrorGeneral* wird TRUE, sobald eine der Störmeldungen *bErrorHighLimit, bErrorLowLimit* oder *bErrorSensor* = TRUE ist. Der Wert der Ausgangsvariable *rPresentValue* ist dann abhängig von der Betriebsart *eTemperatureSensorMode* und wird frei gegeben, wenn die Störung behoben ist und je nach Betriebsart *eTemperatureSensorMode* mit *bReset* quittiert wurde.

byError: Liefert alle Fehlermeldungen und Warnungen,
byError.1 := *bInvalidParameter*
byError.2 := *bErrorGeneral*
byError.3 := *bErrorHighLimit*
byError.4 := *bErrorLowLimit*
byError.5 := *bErrorShortCircuit*
byError.6 := *bErrorBrokenSensor*
byError.7 := *bErrorSensor*

byError.5 := *bErrorShortCircuit:* Fehler, Kurzschluss an dem angeschlossenem Temperatursensor. Nach Behebung des Fehlers wird die Meldung in Abhängigkeit der Betriebsart *eTemperatureSensorMode* entweder mit *bReset* oder automatisch quittiert.
byError.6 := *bErrorBrokenSensor:*Fehler, Drahtbruch an dem angeschlossenem Temperatursensor. Nach Behebung des Fehlers wird die Meldung in Abhängigkeit der Betriebsart *eTemperatureSensorMode* entweder mit *bReset* oder automatisch quittiert.

bErrorSensor: Wird TRUE, wenn *byError*.5/*bErrorShortCircuit* oder *byError*.6/*bErrorBrokenSensor* = TRUE ist. Nach Behebung des Fehlers wird die Meldung in Abhängigkeit der Betriebsart *eTemperatureSensorMode* entweder mit *bReset* oder automatisch quittiert. Ist *bErrorSensor* = TRUE, der Fehler behoben und *bErrorSensor* = FALSE, so ist für einen SPS-Zyklus *rPresentValue* = rPresentValueOld.

bErrorHighLimit: Warnung oberer Grenzwert überschritten, wird TRUE wenn *rPresentValue* >= *rHighLimit* ist. Die Warnung, dass der obere Grenzwert überschritten ist, kann erst quittiert werden, wenn *rPresentValue* <= *rHighLimit* -1.0 für die Zeitdauer von 5 Sekunden ist. Die Warnung wird in Abhängigkeit der Betriebsart *eTemperatureSensorMode* entweder mit *bReset* oder automatisch quittiert.

bErrorLowLimit: Warnung unterer Grenzwert unterschritten, wird TRUE wenn *rPresentValue* <= *rLowLimit* ist. Die Warnung, dass der untere Grenzwert unterschritten ist, kann erst quittiert werden, wenn *rPresentValue* >= *rLowLimit* +1.0 für die Zeitdauer von 5 Sekunden ist. Die Warnung wird in Abhängigkeit der Betriebsart *eTemperatureSensorMode* entweder mit *bReset* oder automatisch quittiert.

bInvalidParameter: Zeigt an, dass ein falscher Parameter an einer der Variablen *rHighLimit, rLowLimit, rSmoothFactor* oder *eTemperatureSensorMode* anliegt. Eine falsche Parameterangabe führt nicht zum Stillstand des Bausteins, siehe Beschreibung der Variablen. Nach Behebung der falschen Parameterangabe wird die Meldung *bInvalidParameter* in Abhängigkeit der Betriebsart *eTemperatureSensorMode* entweder mit *bReset* oder automatisch quittiert.

VAR_IN_OUT

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

rOffset: Temperaturabgleich in Kelvin, *rPresentValue* = (*iRawValue* / 10.0) + *rOffset.* Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rHighLimit: Oberer Grenzwert. Wenn *rPresentValue* >= *rHighLimit*, dann wird der Ausgang *bErrorHighLimit* gesetzt. *rHighLimit* muss größer sein als *rLowLimit.* Die Variable wird persistent gespeichert. Voreingestellt auf 120.
Liegt ein nicht korrekter Variablenwert an dann wird, wenn vorhanden, der letzte gültige Variablenwert genommen. Wenn kein gültiger letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

rLowLimit: Unterer Grenzwert. Wenn *rPresentValue* <= *rLowLimit*, dann wird der Ausgang *bErrorLowLimit* gesetzt. Die Variable wird persistent gespeichert. Voreingestellt auf -60.
Liegt ein nicht korrekter Variablenwert an dann wird, wenn vorhanden, der letzte gültige Variablenwert genommen. Wenn kein gültiger letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

rReplacementValue: Ersatzwert, der bei den Fehlern *bErrorShortCircuit* und *bErrorBrokenSensor* an den Ausgang *rPresentValue* ausgegeben wird, wenn die Betriebsart *eTemperatureSensorMode* = *eHVACTemperatureSensorMode_ReplacementValue* oder *eTemperatureSensorMode* = *eHVACTemperatureSensorMode_AutoResetReplacementValue* ausgewählt ist. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rSmoothFactor: Glättungsfaktor (>=1) für den Ausgabewert *rPresentValue*. Die Variable wird persistent gespeichert. Voreingestellt auf 100.
Nach folgender Formel wird der Wert für *rPresentValue* berechnet und ausgegeben:
rPresentValue := ((iRawValue / 10 + rOffset) - rPresentValueOld) / rSmoothFactor + rPresentValueOld;
rPresentValueOld ist der Wert von *rPresentValue*, der einen SPS-Zyklus zuvor ausgegeben wurde.
Ist *rSmoothFactor* = 1, dann ist *rPresentValue* := ((*iRawValue* / 10 + *rOffset*)
Liegt ein nicht korrekter Variablenwert an *rSmoothFactor* an, dann wird, wenn vorhanden, der letzte gültige Variablenwert genommen. Wenn kein gültiger letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

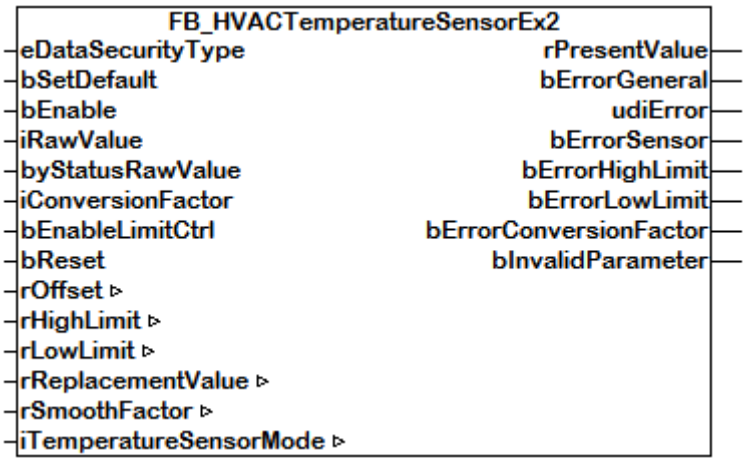
eTemperatureSensorMode: Enum, über welches die Betriebsart des Bausteines vorgegeben wird.
eTemperatureSensorMode = *eHVACTemperatureSensorMode_ReplacementValue*: Wenn *bErrorSensor* = TRUE, dann ist *rPresentValue* = *rReplacementValue.* Nach Behebung des Fehlers muss der Funktionsbaustein durch eine steigende Flanke an der Eingangsvariablen *bReset* quittiert werden.
eTemperatureSensorMode = *eHVACTemperatureSensorMode_LastValue*: Wenn *bErrorSensor* = TRUE, dann wird an der Ausgangsvariablen *rPresentValue* der zuletzt gültige Temperaturwert, der 10 Sekunden vorher anstand, ausgegeben. Nach Behebung des Fehlers muss dieser durch eine steigende Flanke an der Eingangsvariablen *bReset* quittiert werden.
eTemperatureSensorMode = *eHVACTemperatureSensorMode_AutoResetReplacementValue* : Wenn *bErrorShortCircuit* oder *bErrorBrokenSensor* = TRUE, dann ist *rPresentValue* = *rReplacementValue.* Nach Behebung des Fehlers quittiert sich der Funktionsbaustein automatisch.
eTemperatureSensorMode = *eHVACTemperatureSensorMode_AutoResetLastValue*: Wenn *bErrorSensor* = TRUE, dann wird an der Ausgangsvariable *rPresentValue* der zuletzt gültige Temperaturwert, der 10 Sekunden vorher anstand, ausgegeben. Nach Behebung des Fehlers quittiert sich der Funktionsbaustein automatisch.
Liegt ein nicht korrekter Variablenwert an *eTemperatureSensorMode* an, dann wird mit dem Default-Wert weiter gearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

Die Variable wird persistent gespeichert. Voreingestellt auf 3.

Dokumente hierzu

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

3.2.11 FB_HVACTemperatureSensorEx2



Anwendung

Dieser Funktionsbaustein dient zur Erfassung und Weiterverarbeitung von Temperaturwerten wie z.B. für die Fühlertypen PT100, PT200, PT1000, NI100, NI120, NI1000, NI1000Tk5000. Der Funktionsbaustein *FB_HVACTemperatureSensorEx***2** ist abgestimmt auf die Busklemmen KL3201/02/04, KL3222, KL3228 und KL3208-0010. Diese Busklemmen können entweder vorkonfiguriert bestellt oder softwaremäßig auf den entsprechenden Fühlertypen eingestellt werden.
Über die Eingangsvariable *iRawValue* wird der Temperaturrohwert entweder in 1/10 oder 1/100°C dem Funktionsbaustein übergeben und über *rPresentValue* als Fließkommazahl ausgeben. *iRawValue* kann z.B. direkt mit dem Temperaturrohwert der folgenden Busklemmen verknüpft werden: KL3201/02/04, KL3222, KL3228 und KL3208-0010.

Der Ausgabewert *rPresentValue* ist abhängig von einer der beiden folgenden Glättungsfunktionen:

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

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

rPresentValueOld ist der Wert von *rPresentValue*, der einen SPS-Zyklus vorher ausgegeben wurde. Wenn *bEnable* = TRUE wird, so ist für einen SPS-Zyklus *rPresentValue* = rPresentValueOld. Ist *bErrorSensor* = TRUE, der Fehler behoben und *bErrorSensor* = FALSE, so ist für einen SPS-Zyklus *rPresentValue* = rPresentValueOld.

Über die Eingangsvariable *byStatusRawValue* wird der Status des angeschlossenen Temperatursensors überwacht und im Fehlerfall über die Variable *bErrorSensor* der Steuerung zurück geliefert.
byStatusRawValue kann z.B. direkt mit dem Statusbyte der folgenden Busklemmen verknüpft werden: KL3201/02/04, KL3222, KL3228 und KL3208-0010.
Mit *rHighLimit/rLowLimit* können Temperaturgrenzwerte festgelegt werden.

Im Gegensatz zum FB_HVACTemperatureSensor [▶ 91] hat dieser Funktionsbaustein die Eingangsvariable *bEnable*, die von Nutzen ist, wenn die Sensorenkennlinien in den Busklemmen KL3201/02/04, KL3222, KL3228 und KL3208-0010 aus der SPS heraus über den Funktionsbaustein FB_HVACConfigureKL32xx [▶ 77] eingestellt werden sollen. Der Filter 2. Ordnung im FB_HVACTemperatureSensor [▶ 91] wird in diesem Funktionsbaustein durch die oben beschriebene Glättungsfunktion ersetzt. Der Ausgang *bErrorSensor* ist neu und ersetzt die beiden Ausgänge bErrorShortCircuit/bErrorBrokenSensor. Diese Ausgänge sind aber weiterhin im Errorbyte *udiError* vorhanden.

Im Gegensatz zum FB_HVACTemperatureSensorEx [▶ 94] können an diesem Funktionsbaustein Sensoren mit dem Temperaturrohwert 1/10 oder 1/100°C übergeben werden.

Anwendungsbeispiel

Download	Benötigte Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

```
eDataSecurityType  : E_HVACDataSecurityType;
bSetDefault        : BOOL;
bEnable            : BOOL;
iRawValue          : INT;
byStatusRawValue   : BYTE;
iConversionFactor  : INT;              0..1
bEnableLimitCtrl   : BOOL;
bReset             : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Freigabe des Bausteins, wenn *bEnable* = TRUE ist. Ist *bEnable* = FALSE, so wird am Ausgang *rPresentValue* der Wert von *rReplacementValue* ausgegeben. Sämtliche Störmeldungen und *bInvalidParameter* werden auf FALSE gesetzt. Wenn bEnable = TRUE wird, so ist für einen SPS-Zyklus *rPresentValue* = rPresentValueOld.

iRawValue: Rohwert des Temperaturfühlers entweder in 1/10 oder 1/100°C von der Busklemme.

byStatusRawValue: Statusbyte des Temperaturfühlers von der Busklemme. Dient zur Fehlerdiagnose, z.B. Drahtbruch oder Kurzschluss. Ist die KL32xx auf Widerstandsmessung (Ω) eingestellt, so gibt es keine Fehlerdiagnose.

iConversionFactor: Umrechnungsfaktor für den Ausgabewert *rPresentValue*.

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

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

Wird an *iConversionFactor* ein Wert ausserhalb seines Bereiches von 0 bis 1 angegeben, so wird dieses mit *bErrorConversionFactor* = TRUE signalisiert.

bEnableLimitCtrl: Freigabe für die Limit-Überwachung *rHighLimit* und *rLowLimit*

bReset: Quittierungseingang bei einem Fehler mit einer steigenden Flanke an *bReset*. In Abhängigkeit der Betriebsart *iTemperatureSensorMode* werden Fehler entweder mit *bReset* oder automatisch quittiert.

VAR_OUTPUT

```
rPresentValue          : REAL;
bErrorGeneral          : BOOL;
udiError               : UDINT;
bErrorSensor           : BOOL;
bErrorHighLimit        : BOOL;
bErrorLowLimit         : BOOL;
bErrorConversionFactor: BOOL;
bInvalidParameter      : BOOL;
```

rPresentValue: Temperaturausgabevariable mit zwei Nachkommastellen.
Nach folgender Formel wird der Wert für *rPresentValue* berechnet und ausgegeben:

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

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

rPresentValueOld ist der Wert von *rPresentValue*, der einen SPS-Zyklus vorher ausgegeben wurde. Wenn bEnable = TRUE wird, so ist für einen SPS-Zyklus *rPresentValue* = rPresentValueOld. Ist *bErrorSensor* = TRUE, der Fehler behoben und *bErrorSensor* = FALSE, so ist für einen SPS-Zyklus *rPresentValue* = rPresentValueOld.
Ist *bErrorSensor* = TRUE, dann ist der Wert von *rPresentValue* abhängig von der Betriebsart *iTemperatureSensorMode.*

bErrorGeneral: Die Störungsmeldung *bErrorGeneral* wird TRUE, sobald eine der Störmeldungen *bErrorHighLimit, bErrorLowLimit* oder *bErrorSensor* = TRUE ist. Der Wert der Ausgangsvariable *rPresentValue* ist dann abhängig von der Betriebsart *iTemperatureSensorMode* und wird frei gegeben, wenn die Störung behoben ist und je nach Betriebsart *iTemperatureSensorMode* mit *bReset* quittiert wurde.

udiError: Liefert alle Fehlermeldungen und Warnungen,
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:* Fehler, Kurzschluss an dem angeschlossenem Temperatursensor. Nach Behebung des Fehlers wird die Meldung in Abhängigkeit der Betriebsart *iTemperatureSensorMode* entweder mit *bReset* oder automatisch quittiert.
byError.6 := *bErrorBrokenSensor:*Fehler, Drahtbruch an dem angeschlossenem Temperatursensor. Nach Behebung des Fehlers wird die Meldung in Abhängigkeit der Betriebsart *iTemperatureSensorMode* entweder mit *bReset* oder automatisch quittiert.

bErrorSensor: Wird TRUE, wenn *byError*.5 / *bErrorShortCircuit* oder *byError*.6 / *bErrorBrokenSensor* = TRUE ist. Nach Behebung des Fehlers wird die Meldung in Abhängigkeit der Betriebsart *iTemperatureSensorMode* entweder mit *bReset* oder automatisch quittiert. Ist *bErrorSensor* = TRUE, der Fehler behoben und *bErrorSensor* = FALSE, so ist für einen SPS-Zyklus *rPresentValue* = rPresentValueOld.

bErrorHighLimit: Warnung oberer Grenzwert überschritten, wird TRUE wenn *rPresentValue* >= *rHighLimit* ist. Die Warnung, dass der obere Grenzwert überschritten ist, kann erst quittiert werden, wenn *rPresentValue* <= *rHighLimit* -1.0 für die Zeitdauer von 5 Sekunden ist. Die Warnung wird in Abhängigkeit der Betriebsart *iTemperatureSensorMode* entweder mit *bReset* oder automatisch quittiert.

bErrorLowLimit: Warnung unterer Grenzwert unterschritten, wird TRUE wenn *rPresentValue* <= *rLowLimit* ist. Die Warnung, dass der untere Grenzwert unterschritten ist, kann erst quittiert werden, wenn *rPresentValue* >= *rLowLimit* +1.0 für die Zeitdauer von 5 Sekunden ist. Die Warnung wird in Abhängigkeit der Betriebsart *iTemperatureSensorMode* entweder mit *bReset* oder automatisch quittiert.

bErrorConversionFactor: Wird an *iConversionFactor* ein Wert ausserhalb seines Bereiches von 0 bis 1 angegeben, so wird dieses mit *bErrorConversionFactor* = TRUE signalisiert. Die Meldung ist nach Behebung der Ursache nicht quittierungspflichtig.

bInvalidParameter: Zeigt an, dass ein falscher Parameter an einer der Variablen *rHighLimit, rLowLimit, rSmoothFactor, iConversionFactor* oder *iTemperatureSensorMode* anliegt. Eine falsche Parameterangabe führt nicht zum Stillstand des Bausteins, siehe Beschreibung der Variablen. Nach Behebung der falschen Parameterangabe wird die Meldung *bInvalidParameter* in Abhängigkeit der Betriebsart *iTemperatureSensorMode* entweder mit *bReset* oder automatisch quittiert.

VAR_IN_OUT

```
rOffset               : REAL;
rHighLimit            : REAL;
rLowLimit             : REAL;
rReplacementValue     : REAL;
rSmoothFactor         : REAL;
iTemperatureSensorMode:INT;
```

rOffset: Temperaturabgleich in Kelvin. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rHighLimit: Oberer Grenzwert. Die Variable wird persistent gespeichert. Voreingestellt auf 120.
Wenn *rPresentValue* >= *rHighLimit*, dann wird der Ausgang *bErrorHighLimit* gesetzt. *rHighLimit* muss größer sein als *rLowLimit.*
Liegt ein nicht korrekter Variablenwert an dann wird, wenn vorhanden, der letzte gültige Variablenwert genommen. Wenn kein gültiger letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

rLowLimit: Unterer Grenzwert. Die Variable wird persistent gespeichert. Voreingestellt auf -60.
Wenn *rPresentValue* <= *rLowLimit*, dann wird der Ausgang *bErrorLowLimit* gesetzt.
Liegt ein nicht korrekter Variablenwert an dann wird, wenn vorhanden, der letzte gültige Variablenwert genommen. Wenn kein gültiger letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

rReplacementValue: Ersatzwert, der bei den Fehlern *bErrorShortCircuit* und *bErrorBrokenSensor* an den Ausgang *rPresentValue* ausgegeben wird, wenn die Betriebsart *iTemperatureSensorMode = 0* oder *iTemperatureSensorMode = 2* ausgewählt ist.
Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rSmoothFactor: Glättungsfaktor (>=1) für den Ausgabewert *rPresentValue*.
Die Variable wird persistent gespeichert. Voreingestellt auf 100.

Nach folgender Formel wird der Wert für *rPresentValue* berechnet und ausgegeben:

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

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

rPresentValueOld ist der Wert von *rPresentValue*, der einen SPS-Zyklus zuvor ausgegeben wurde.

Liegt ein nicht korrekter Variablenwert an *rSmoothFactor* an, dann wird, wenn vorhanden, der letzte gültige Variablenwert genommen. Wenn kein gültiger letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

iTemperatureSensorMode: Gibt die Betriebsart des Bausteines vor.
iTemperatureSensorMode = 0: Wenn *bErrorSensor* = TRUE, dann ist *rPresentValue = rReplacementValue.* Nach Behebung des Fehlers muss der Funktionsbaustein durch eine steigende Flanke an der Eingangsvariablen *bReset* quittiert werden.
iTemperatureSensorMode = 1: Wenn *bErrorSensor* = TRUE, dann wird an der Ausgangsvariablen *rPresentValue* der zuletzt gültige Temperaturwert, der 10 Sekunden vorher anstand, ausgegeben. Nach Behebung des Fehlers muss dieser durch eine steigende Flanke an der Eingangsvariablen *bReset* quittiert werden.
iTemperatureSensorMode = 2 : Wenn *bErrorShortCircuit* oder *bErrorBrokenSensor* = TRUE, dann ist *rPresentValue = rReplacementValue.* Nach Behebung des Fehlers quittiert sich der Funktionsbaustein automatisch.
iTemperatureSensorMode = 3: Wenn *bErrorSensor* = TRUE, dann wird an der Ausgangsvariable *rPresentValue* der zuletzt gültige Temperaturwert, der 10 Sekunden vorher anstand, ausgegeben. Nach Behebung des Fehlers quittiert sich der Funktionsbaustein automatisch.

Liegt ein nicht korrekter Variablenwert an *iTemperatureSensorMode* an, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.
Die Variable wird persistent gespeichert. Voreingestellt auf 3.

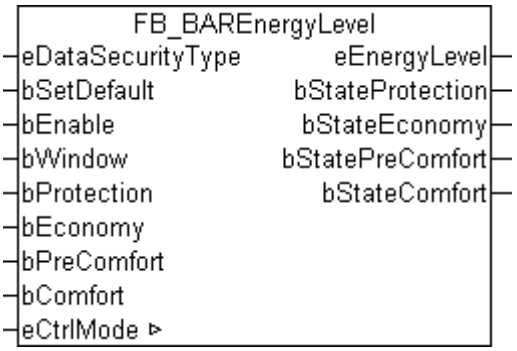
Dokumente hierzu

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

3.3 HLK Raumfunktionen

3.3.1 Klimatisierung

3.3.1.1 FB_BAREnergyLevel



Applikation

Der Funktionsbaustein dient der Anpassung der Energieabgabe an die Nutzung des Gebäudes. Die Raumnutzungsart wird von der GLT vorgegeben. Je länger ein Gebäude bzw. ein Raum nicht genutzt wird, desto weiter kann sein Energieniveau abgesenkt werden. Das von dem Funktionsbaustein aktuell ausgewählte Energieniveau wird an die Raumtemperaturregelung übertragen.

Protection:

Diese Betriebsart wird bei langen Abwesenheitszeiten z.B. in den Betriebsferien oder auch bei einem geöffneten Fenster aktiviert. Das Energieniveau ist sehr niedrig und dient lediglich dem Schutz des Gebäudes vor Frost- oder Überhitzungsschäden.

Economy:

Das Energieniveau Economy wird für den Absenkbetrieb genutzt. Der Absenkbetrieb wird zum Beispiel Nachts durch einen Zeitschaltplan aktiviert.

PreComfort:

Das Energieniveau PreComfort ist für einen ungenutzten Raum, der jedoch in Kürze wieder belegt sein kann. Die Aktivierung des Bereitschaftsbetriebes geschieht häufig durch einen Zeitschaltplan.

Comfort:
Wenn der Raum belegt ist befindet er sich im Komfortbetrieb. Die Aktivierung des Komfortbetriebes kann durch eine Zeitschaltplanung oder durch eine Anwesenheitserkennung erfolgen.

VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Über ein TRUE an diesem Eingang wird der Funktionsbaustein aktiviert.

bWindow: An diesen Eingang wird der Fensterkontakt angeschlossen. TRUE, entspricht das Fenster ist AUF. FALSE, entspricht das Fenster ist ZU.

bProtection: Mit dem Eingang bProtection wird der Schutzbetrieb aktiviert. Der Schutzbetrieb ist aktiv, wenn der Eingang TRUE ist.

bEconomy: Mit dem Eingang bEconomy wird der Absenkbetrieb aktiviert. Der Absenkbetrieb ist aktiv, wenn der Eingang TRUE ist.

bPreComfort: Mit diesem Eingang wird das Bereitschaftsniveau aktiviert. Das Bereitschaftsniveau ist aktiv, wenn der Eingang TRUE ist.

bComfort: Bei Raumbelegung wird mit diesem Eingang das Komfortniveau aktiviert.

VAR_OUTPUT

```
eEnergyLevel      : E_BAREnergyLevel;
bStateProtection  : BOOL;
bStateEconomy     : BOOL;
bStatePreComfort  : BOOL;
bStateComfort     : BOOL;
```

eEnergyLevel: Dieser Ausgang enthält das aktuelle Energieniveau.

bStateProtection: Der Status von dem *bProtection* Eingang wird in der Betriebsart *eBAREnergyLevel_AUTO_I* und *eBAREnergyLevel_AUTO_II* nach außen weitergegeben.

bStateEconomy: Der Status von dem *bEconomy* Eingang wird in der Betriebsart *eBAREnergyLevel_AUTO_I* und *eBAREnergyLevel_AUTO_II* nach außen weitergegeben.

bStatePreComfort: Der Status von dem *bPreComfort* Eingang wird in der Betriebsart *eBAREnergyLevel_AUTO_I* und *eBAREnergyLevel_AUTO_II* nach außen weitergegeben.

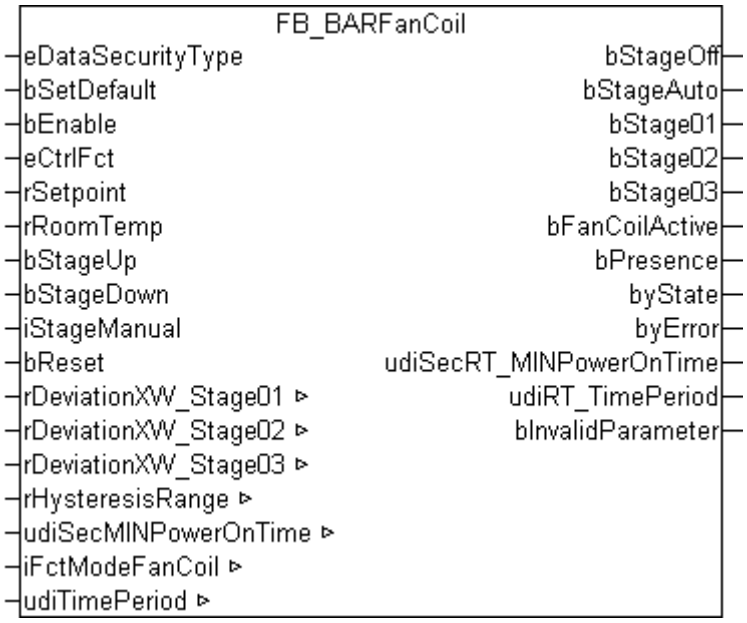
bStateComfort: Der Status von dem *bComfort* Eingang wird in der Betriebsart *eBAREnergyLevel_AUTO_I* und *eBAREnergyLevel_AUTO_II* nach außen weitergegeben.

VAR_IN_OUT

```
eCtrlMode     : E_BAREnergyLevel;
```

eCtrlMode: Über dieses ENUM kann die Betriebsart von der Gebäudeleitebene aus vorgewählt werden. Die Variable wird persistent gespeichert. Voreingestellt auf Automatik.

3.3.1.2 FB_BARFanCoil



Anwendung

Der Funktionsbaustein bildet einen 3-stufigen Ventilator mit der entsprechenden Schalthysterese ab, diese ist für alle drei Stufen gleich. Über die Regelabweichung des Raumtemperatur-Istwert zum Raumtemperatur-Sollwert wird die Drehzahl stufenweise eingestellt. Des Weiteren hat man die Möglichkeit über den Eingang *iStageManual* bzw. *bStageUp* oder *bStageDown* die Ventilatorsteuerung manuell zu übersteuern. Über den Eingang *udiSecMINPowerOnTime* kann eine Mindesteinschaltzeit eingestellt werden, die dann für jede Stufe gültig ist.

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: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Ist die Variable *bEnable* TRUE, dann ist der Funktionsbaustein aktiviert. Bei einem FALSE ist keine Ventilatorstufe angesteuert.

eCtrlFct: Dieser Eingang wird mit dem Ausgang *eCtrlFct* von dem **FB_BARFctSelection** verbunden. Diese Information ist wichtig um zu wissen, ob die Anlage sich im Heiz- oder Kühlbetrieb befindet. Im Automatikbetrieb werden die Ventilatorstufen nur dann angesteuert, wenn z.B. laut Regelabweichung ist die Anforderung für Heizen aktiv ist und die Anlage befindet sich im Heizbetrieb. Oder laut Regelabweichung ist die Anforderung für Kühlen aktiv und die Anlage befindet sich im Kühlbetrieb.

rSetpoint: Eingang für die Solltemperatur.

rRoomTemp: Eingang für die Raumtemperatur.

bStageUp: Lokale Verstellung der Ventilatorstufe, schrittweise Hochtasten.

bStageDown: Lokale Verstellung der Ventilatorstufe, schrittweise Runtertasten.

iStageManual: Über diesen Eingang kann/wird die manuelle Ventilatorstufe zentral eingestellt.
iStageManual: = 0 entspricht der Ventilatorstufe AUS
iStageManual: = 1 entspricht der Ventilatorstufe AUTO
iStageManual: = 2 entspricht der Ventilatorstufe01 aktiv
iStageManual: = *3*, entspricht der Ventilatorstufe02 aktiv
iStageManual: = *4*, entspricht der Ventilatorstufe03 aktiv

bReset: Quittierungseingang bei einer Störung oder bei einem falschen 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, Ventilatorstufen sind ausgeschaltet.

bStageAuto: TRUE, Ventilatorsteuerung befindet sich im Automatikbetrieb.

bStage01: TRUE, Ventilatorstufe01 aktiv.

bStage02: TRUE, Ventilatorstufe02 aktiv.

bStage03: TRUE, Ventilatorstufe03 aktiv.

bFanCoilActive: TRUE, wenn eine von den drei Ventilatorstufen aktiv ist. Dieser Ausgang kann/wird zur Freigabe von Regler verwendet, damit ein Hitze- bzw. Kältestau vermieden wird.

bPresence: TRUE entspricht, dass über die Eingänge *bStageUp*, *bStageDown* oder *iStageManual* Präsenz detektiert wurde.

byState: Zeigt den Status der Ventilatorsteuerung an.
byState.0:= Baustein ist aktiviert
byState.3:= manuelle Ventilatorstufen Vorgabe ist aktiv
byState.4:= bReset
byState.5:= Ventilatorstufe01 aktiv
byState.6:= Ventilatorstufe02 aktiv
byState.7:= Ventilatorstufe03 aktiv

byError: Ausgabe der Fehler als Byte.
byError.1:= bInvalidParameter

udiSecRT_MINPowerOnTime: Zeigt, die verbleibende Zeit der Mindesteinschaltdauer an.

udiRT_TimePeriod: Zeigt, die verbleibende Zeit der manuellen Übersteuerung an.

bInvalidParameter: Zeigt an, dass ein falscher Eingangsparameter anliegt. *bInvalidParameter* muss mit *bReset* quittiert werden.

VAR_IN_OUT

```
rDeviationXW_Stage01    : REAL;
rDeviationXW_Stage02    : REAL;
rDeviationXW_Stage03    : REAL;
rHysteresisRange        : REAL;
udiSecMINPowerOnTime    : UDINT;
iFctModeFanCoil         : INT;
udiTimePeriod           : UDINT;
```

rDeviationXW_Stage01: Grenzwert der Regelabweichung für die Ventilatorstufe01. Die Variable wird persistent gespeichert. Voreingestellt auf 0,7.

rDeviationXW_Stage02: Grenzwert der Regelabweichung für die Ventilatorstufe02. Die Variable wird persistent gespeichert. Voreingestellt auf 1,7.

rDeviationXW_Stage03: Grenzwert der Regelabweichung für die Ventilatorstufe03. Die Variable wird persistent gespeichert. Voreingestellt auf 2,1.

rHysteresisRange: Hysterese Bereich, der um den Grenzwert gelegt wird.
Beispiel: Ein Grenzwert für von 0.7 und einem Hysteresebereich von 0.2 hat zur Folge, dass die Ventilatorstufe01 bei einer Regelabweichung > 0.8 eingeschaltet wird.
Und bei einer Regelabweichung < 0.6 wird die Ventilatorstufe01 ausgeschaltet.
Die Variable wird persistent gespeichert. Voreingestellt auf 0,2.

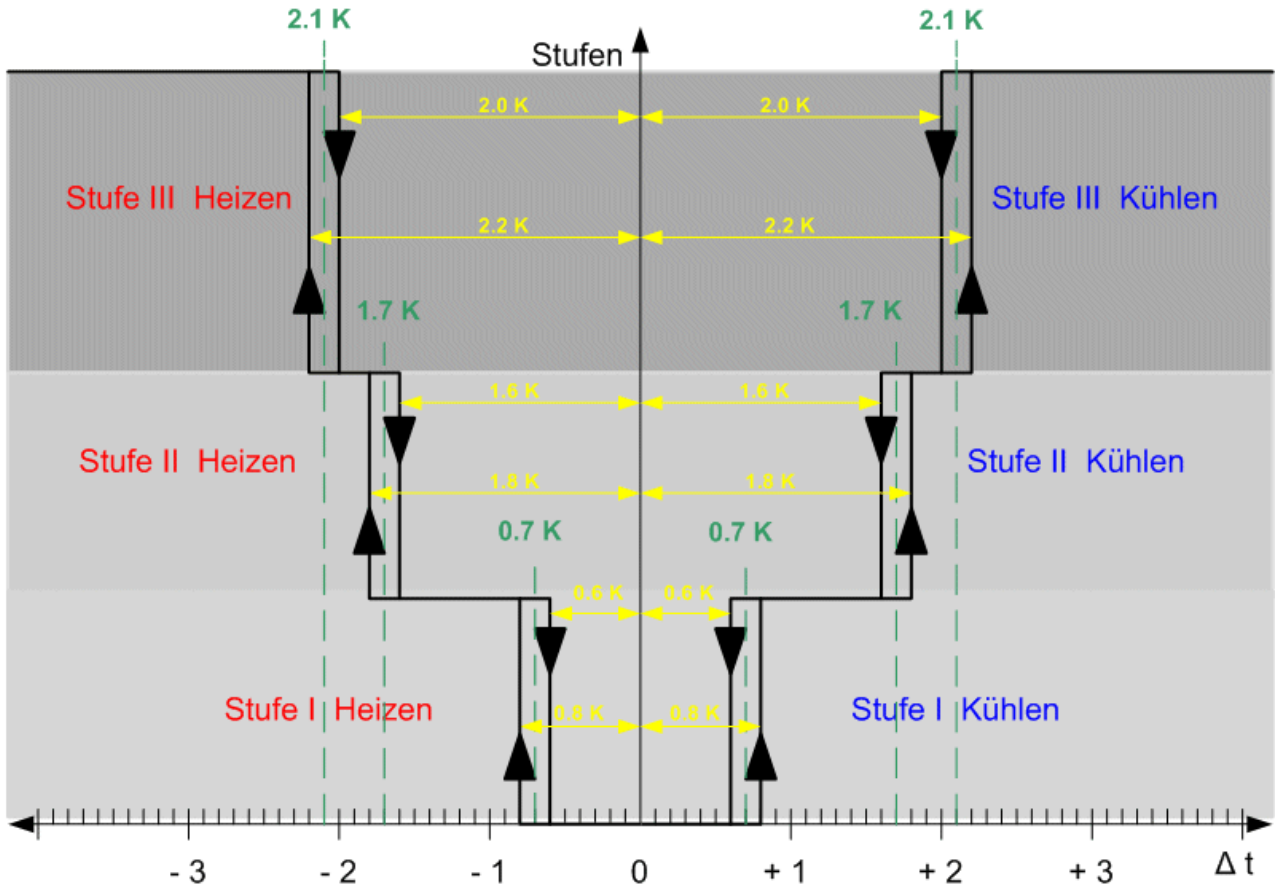
udiSecMINPowerOnTime: Mindesteinschaltzeit, die ein Ventilator in einer Stufe laufen muss bis er auf eine andere Stufe schaltet bzw. ausschaltet. Eingabe in Sekunden (z.B. 120 entspricht 120s). Die Variable wird persistent gespeichert. Voreingestellt auf 120s.

iFctModeFanCoil: Über die Wertigkeit der Variable hat der Anwender die Möglichkeit die Ventilatorsteuerung für den Heizbetrieb oder Kühlbetrieb oder für beide Betriebe zu aktivieren. Gültige Werte sind 1,2 oder 3 andere Werte sind ungültig und bInvalidParameter wird auf TRUE gesetzt. Die Variable wird persistent gespeichert. Voreingestellt auf 3.

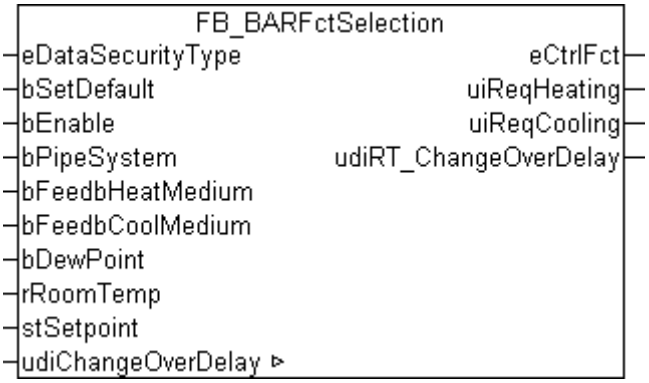
Cooling	Heating	Wertigkeit
0	1	1 (entspricht Ventilatorsteuerung im Heizbetrieb aktiv)
1	0	2 (entspricht Ventilatorsteuerung im Kühlbetrieb aktiv)
1	1	3 (entspricht Ventilatorsteuerung im Heizbetrieb und Kühlbetrieb aktiv)

udiTimePeriod: Zeitfenster indem die manuelle Übersteuerung aktiv ist bei Präsenz. Angabe in Minuten

Bild1: Darstellung der Ventilatoransteuerung mit den Default-Parametern



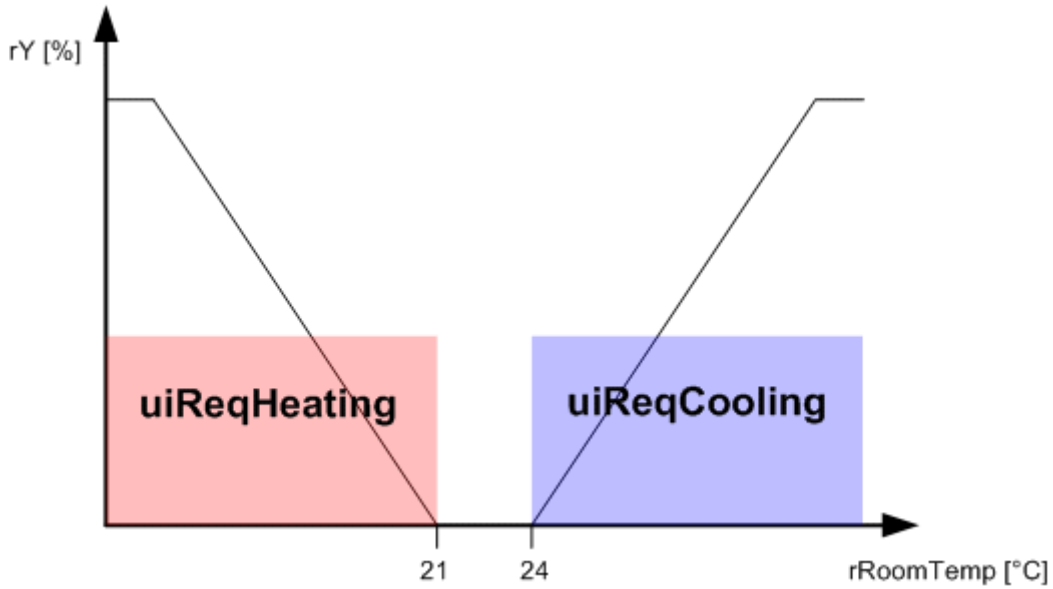
3.3.1.3 FB_BARFctSelection



Anwendung

Der Funktionsbaustein dient zur Freigabe einer Raumheizung oder Raumkühlung. Er kann für 2-Leiter Rohrleitungssysteme (change over) oder 4-Leiter Rohrleitungssysteme verwendet werden.
Bei einem 4-Leiter Rohrleitungssystem erfolgt die Umschaltung vom Heiz- in den Kühlbetrieb automatisch anhand des Vergleiches von dem Raumtemperatursollwert und dem Raumtemperaturistwert.

Skizze:



Bei einem 2-Leiter Rohrleitungssystem darf die Freigabe des Heizbetriebes oder des Kühlbetriebes nur dann erfolgen, wenn Heizmedium bzw. Kühlmedium ansteht. diese Information bekommt die Raumtemperaturregelung von den Primäranlagen.

Sowohl in 2-Leiter Rohrleitungssystemen als auch in 4-Leiter Rohrleitungssystemen kann das Umschalten zwischen dem Heizbetrieb und dem Kühlbetrieb durch einen Timer verzögert werden. Dafür muss die Eingangsvariable *udiChangeOverDelay* größer Null sein.

Folgende Tabellen beschreiben den Zusammenhang zwischen den Eingängen und dem Ausgang eCtrlFct des Funktionsbausteins FB_BARFctSelection.

im 2-Leiter Rohrleitungssystem

bEnable	bPipeSystem	bFeedbHeat Medium	bFeedbCool Medium	interim Result	bDewPoint	eCtrlFct
0	0	0	0	OFF	TRUE / FALSE	OFF
1	0	0	0	Heating	TRUE / FALSE	Heating
1	0	0	1	Cooling	TRUE	OFF
					FALSE	Cooling
1	0	1	0	Heating	TRUE / FALSE	Heating
1	0	1	1	Heating	TRUE / FALSE	Heating

im 4-Leiter Rohrleitungssystem

bEnable	bPipeSystem	T_Room < = Tsetpoint	T_Room > Tsetpoint	interim Result	bDewPoint	eCtrlFct
0	1	0	0	OFF	TRUE / FALSE	OFF
1	1	0	1	Cooling	TRUE	OFF
					FALSE	Cooling
1	1	1	0	Heating	TRUE / FALSE	Heating
1	1	1	1	Heating	TRUE / FALSE	Heating

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
bEnable          : BOOL;
bPipeSystem      : BOOL;
bFeedbHeatMedium : BOOL;
bFeedbCoolMedium : BOOL;
bDewPoint        : BOOL;
rRoomTemp        : REAL;
stSetpoint       : ST_BARSetpointRoom;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: TRUE, Freigabe des Bausteines. Liegt ein FALSE an, ist der Baustein gesperrt und *eCtrlFct := eHVACCtrlFct_Off*.

bPipeSystem: FALSE, bedeutet ein 2-Leitersystem ist vorhanden. TRUE, bedeutet ein 4-Leitersystem ist vorhanden.

bFeedbHeatMedium: Meldung von der Energieerzeugung bzw. Verteilung, dass Heizmedium zur Verfügung steht.

bFeedbCoolMedium: Meldung von der Energieerzeugung bzw. Verteilung, dass Kühlmedium zur Verfügung steht.

bDewPoint: An diesen Eingang wird der Taupunktfühler angeschlossen, löst dieser aus wird die Kühlregelfunktion deaktiviert und *eCtrlFct := eHVACCtrlFct_Off* gesetzt.

rRoomTemp: Über diese Eingangsvariable wird dem Funktionsbaustein die aktuelle Raumtemperatur übergeben.

stSetpoint: STRUKTUR, die die Sollwerte der einzelnen Energieniveaus beinhaltet.

VAR_OUTPUT

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

eCtrlFct: Dieser Ausgang enthält die aktuelle Regelfunktion.

uiReqHeating: Ist 1, wenn der Raum/Zone Heizenergie anfordert. Ist 0, wenn kein Heizbedarf vorhanden ist.

uiReqCooling: Ist 1, wenn der Raum/Zone Kühlenergie anfordert. Ist 0, wenn kein Kühlbedarf vorhanden ist.

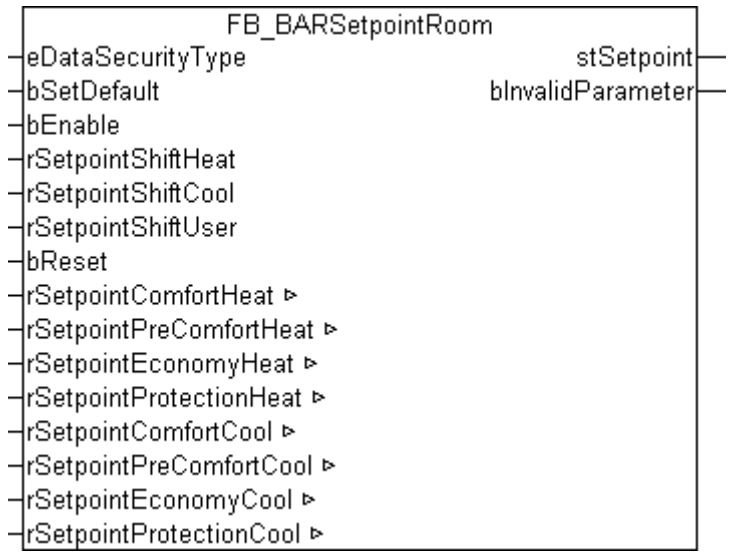
udiRT_ChangeOverDelay: Zeigt die verbleibende Zeit an, bis die aktive Regelfunktion umgeschaltet wird.

VAR_IN_OUT

```
uiChangeOverDelay  : UINT;
```

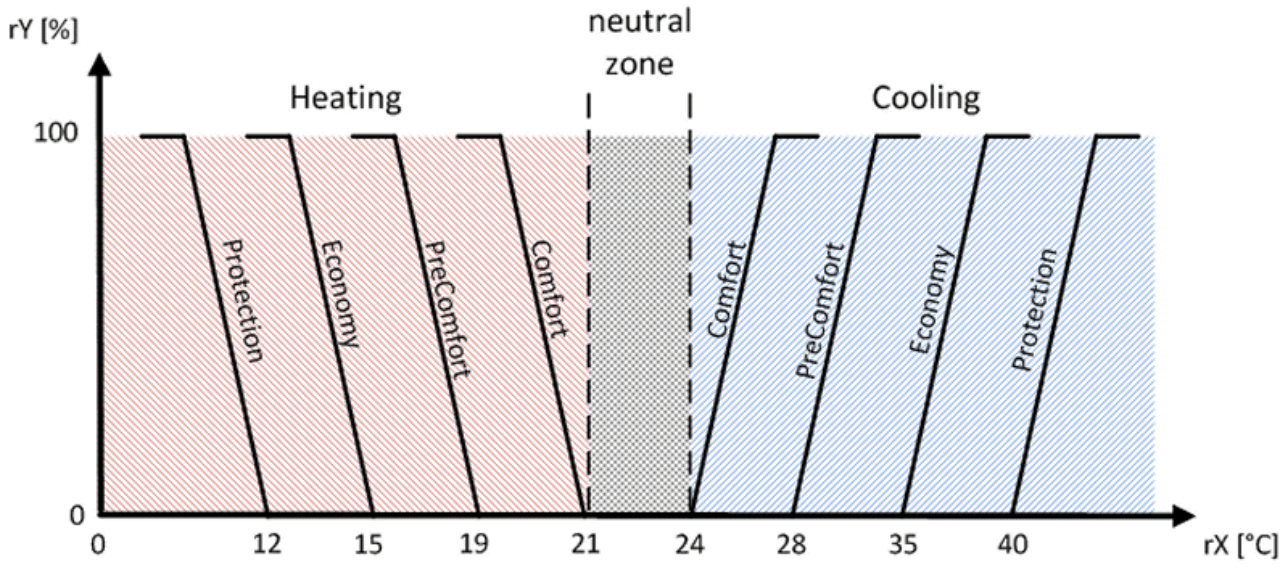
uiChangeOverDelay: Umschaltzeit zw. den Regelfunktionen. Die Angabe muss in Sekunden erfolgen. Ist die Eingabe größer 0, wird diese immer beachtet. Soll zw. den Regelfunktionen keine Umschaltzeit sein, muss die Variable 0 sein. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

3.3.1.4 FB_BARSetpointRoom



Anwendung

Der Funktionbaustein FB_BARSetpointRoom weist den Energieniveaus Protection, Economy, PreComfort und Comfort jeweils einen Sollwert für den Kühl- und Heizbetrieb zu. In Verbindung mit dem Funktionsbaustein FB_BAREnergyLevel werden der Raumtemperaturregelung entsprechend der Raumnutzung und der Funktionsauswahl Heiz- oder Kühlbetrieb energetisch optimalen Sollwert zugewiesen.



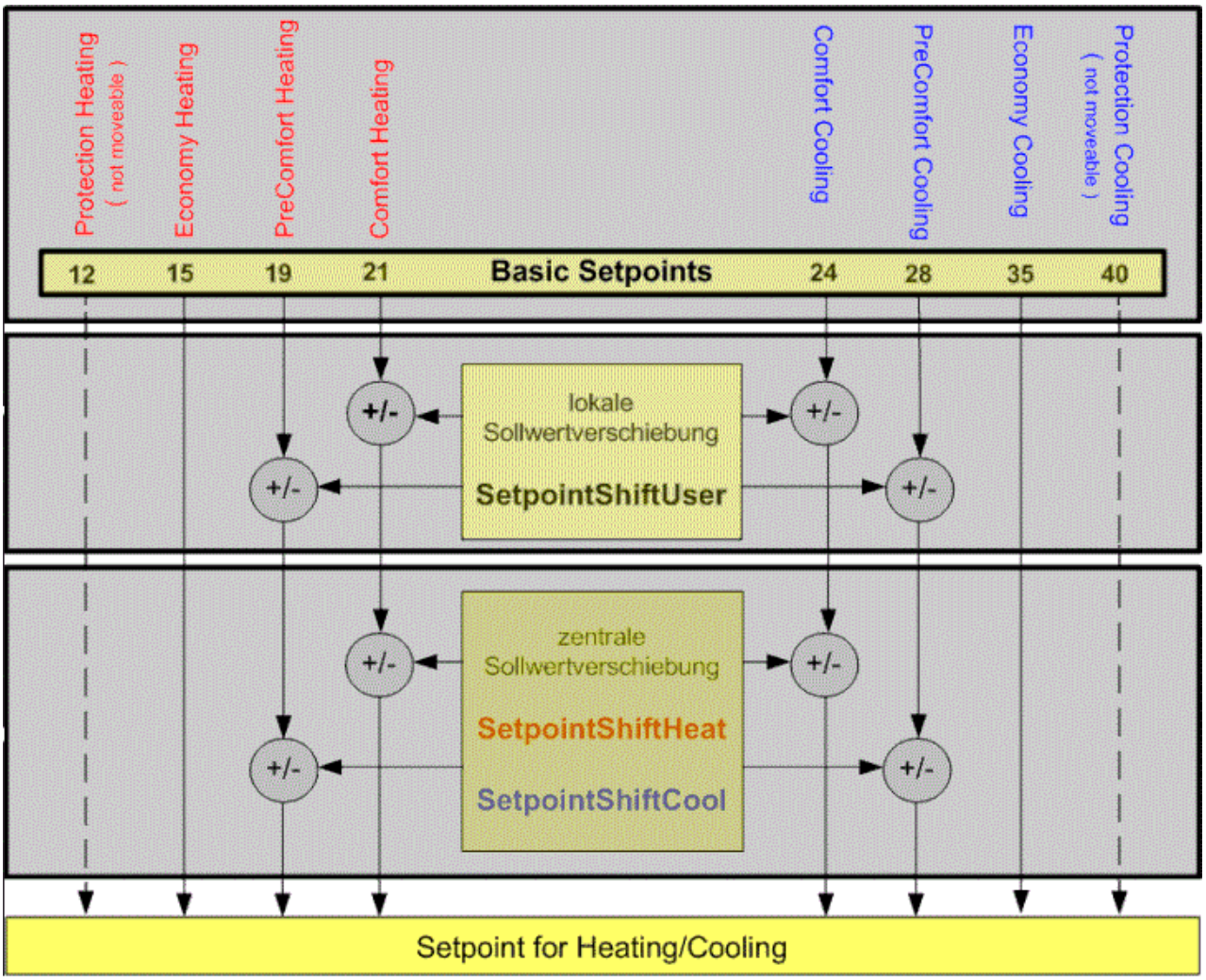
Der resultierende Sollwert für die verschiedenen Energieniveaus setzt sich zusammen aus:

1. dem Basissollwert

2. der lokalen Sollwertverschiebung (nicht bei den Protection Sollwerte)

3. der zentralen Sollwertverschiebung (nicht bei den Protection Sollwerte)

Die lokale Verschiebung durch einen Raumsollwertsteller wie auch die Fernverstellung der Sollwerte über eine Gebäudeleittechnik wirken nur auf die Energieniveaus Comfort und PreComfort.



VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
rSetpointShiftHeat: REAL;
rSetpointShiftCool: REAL;
rSetpointShiftUser: REAL;
bReset            : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Ist die Variable *bEnable* TRUE, dann ist der Funktionsbaustein aktiviert.

rSetpointShiftHeat: Zur Anpassung des ComfortHeating Sollwertes von der Gebäudeleittechnik dient die Variable rSetpointShiftHeat.

Bei einer Erhöhung des ComfortHeating Sollwertes wird der Sollwert des ComfortCooling und PreComfortCooling mit erhöht.
Beispiel:

Energieniveaus		ProtectionHeating	EconomyHeating	PreComfortHeating	ComfortHeating	ComfortCooling	PreComfortCooling	EconomyCooling	ProtectionCooling
Basissollwert [°C]		12	15	19	21	24	28	35	40
SetpointShiftHeat [K]	+3	-	-	-	+3	+3	+3	-	-
resultierender Sollwert [°C]		12	15	19	24	27	31	35	40

Eine Senkung des ComfortHeating Sollwertes bezieht sich nur auf den ComfortHeating und PreComfortHeating Sollwert.
Beispiel:

Energieniveaus		ProtectionHeating	EconomyHeating	PreComfortHeating	ComfortHeating	ComfortCooling	PreComfortCooling	EconomyCooling	ProtectionCooling
Basissollwert [°C]		12	15	19	21	24	28	35	40
SetpointShiftHeat [K]	-3	-	-	-3	-3	-	-	-	-
resultierender Sollwert [°C]		12	15	16	18	24	28	35	40

rSetpointShiftCool: Zur Anpassung des ComfortCooling Sollwertes von der Gebäudeleittechnik dient die Variable rSetpointShiftCool.

Bei einer Erhöhung des ComfortCooling Sollwertes wird der Sollwert des PreComfortCooling mit erhöht.
Beipiel:

Energieniveaus		ProtectionHeating	EconomyHeating	PreComfortHeating	ComfortHeating	ComfortCooling	PreComfortCooling	EconomyCooling	ProtectionCooling
Basissollwert [°C]		12	15	19	21	24	28	35	40
SetpointShiftCool [K]	+3	-	-	-	-	+3	+3	-	-

resultierender Sollwert [°C]	12	15	19	21	27	31	35	40

Eine Senkung des ComfortCooling Sollwertes bezieht sich nur auf den ComfortCooling. Der PreComfortCooling wird dabei nicht verändert.
Beispiel:

Energieniveaus		ProtectionHeating	EconomyHeating	PreComfortHeating	ComfortHeating	ComfortCooling	PreComfortCooling	EconomyCooling	ProtectionCooling
Basissollwert [°C]		12	15	19	21	24	28	35	40
SetpointShiftCool [K]	-3	-	-	-	-	-3	-	-	-
resultierender Sollwert [°C]		12	15	19	21	21	28	35	40

Falls der Sollwert der Energieniveaus PreComfort über den Sollwert der Niveaus Economy hinaus verschoben wird, nimmt der Sollwert des Niveaus Economy den Wert des Niveaus PreComfort an.
Beispiel:

Energieniveaus		ProtectionHeating	EconomyHeating	PreComfortHeating	ComfortHeating	ComfortCooling	PreComfortCooling	EconomyCooling	ProtectionCooling
Basissollwert [°C]		12	15	19	21	24	28	35	40
SetpointShiftCool [K]	+8	-	-	-	-	+8	+8	-	-
resultierender Sollwert [°C]		12	15	19	21	32	36	36	40

rSetpointShiftUser: Zur lokalen Sollwertverschiebung des Nutzers dient die Variable rSetpointShiftUser. Ein positiver Wert von rSetpointShiftUser wirkt sich auf den Sollwert von ComfortHeating,ComfortCooling und PreComfortColling aus.
Beispiel:

Energieniveaus		ProtectionHeating	EconomyHeating	PreComfortHeating	ComfortHeating	ComfortCooling	PreComfortCooling	EconomyCooling	ProtectionCooling
Basissollwert [°C]		12	15	19	21	24	28	35	40
SetpointShiftUser [K]	+3	-	-	-	+3	+3	+3	-	-
resultierender Sollwert [°C]		12	15	19	24	27	31	35	40

Ein negativer Wert von rSetpointShiftUser wirkt sich auf den Sollwert von PreComfortHeating, ComfortHeating und ComfortCooling.
Beispiel:

Energieniveaus		ProtectionHeating	Economy Heating	PreComfortHeating	Comfort Heating	Comfort Cooling	PreComfortCooling	Economy Cooling	ProtectionCooling
Basissollwert [°C]		12	15	19	21	24	28	35	40
SetpointShiftUser [K]	-3	-	-	-3	-3	-3	-	-	-
resultierender Sollwert [°C]		12	15	16	18	21	28	35	40

bReset: Quittierungseingang bei einer Störung oder bei einem falschen Parameter.

VAR_OUTPUT

```
stSetpoint       : ST_BARSetpointRoom;
bInvalidParameter: BOOL;
```

stSetpoint: Struktur, die die Sollwerte für alle Energieniveaus beinhaltet.

bInvalidParameter: Zeigt an, dass ein falscher Eingangsparameter anliegt. *bInvalidParameter* muss mit *bReset* quittiert werden.

VAR_IN_OUT

```
rSetpointComfortHeat        : REAL;
rSetpointPreComfortHeat     : REAL;
rSetpointEconomyHeat        : REAL;
rSetpointProtectionHeat     : REAL;
rSetpointComfortCool        : REAL;
rSetpointPreComfortCool     : REAL;
rSetpointEconomyCool        : REAL;
rSetpointProtectionCool     : REAL;
```

rSetpointComfortHeat: Sollwert für das Energieniveau Comfort-Heizen. Die Variable wird persistent gespeichert. Voreingestellt auf 21,0.

rSetpointPreComfortHeat: Sollwert für das Energieniveau PreComfort-Heizen. Die Variable wird persistent gespeichert. Voreingestellt auf 19,0.

rSetpointEconomyHeat: Sollwert für das Energieniveau Economy-Heizen. Die Variable wird persistent gespeichert. Voreingestellt auf 15,0.

rSetpointProtectionHeat: Sollwert für das Energieniveau Protection-Heizen. Die Variable wird persistent gespeichert. Voreingestellt auf 12,0.

rSetpointComfortCool: Sollwert für das Energieniveau Comfort-Kühlen. Die Variable wird persistent gespeichert. Voreingestellt auf 24,0.

rSetpointPreComfortCool: Sollwert für das Energieniveau PreComfort-Kühlen. Die Variable wird persistent gespeichert. Voreingestellt auf 28,0.

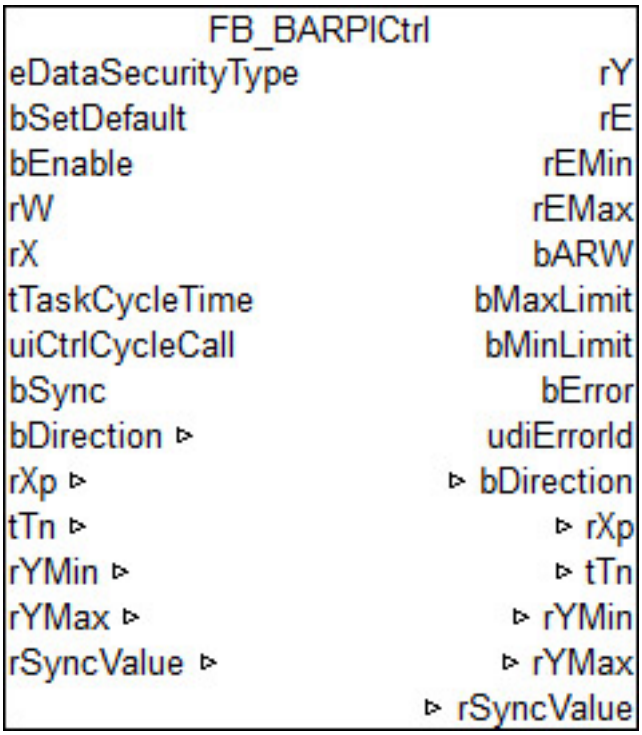
rSetpointEconomyCool: Sollwert für das Energieniveau Economy-Kühlen. Die Variable wird persistent gespeichert. Voreingestellt auf 35,0.

rSetpointProtectionCool: Sollwert für das Energieniveau Protection-Kühlen. Die Variable wird persistent gespeichert. Voreingestellt auf 40.0.

3.3.2 Regler

3.3.2.1 FB_BARPICtrl

Einfacher PI-Regler. Die Regelverstärkung hat keinen Einfluss auf den I-Anteil.



Dieser PI-Regler arbeitet nicht direkt mit einem einstellbarem Verstärkungsfaktor K_p sondern mit dem so genannten Proportionalband (Eingang *rXp*) bezogen auf die Stellgrößengrenzen (*rYmin* und *rYmax*). Daraus wird dann intern der K_p ermittelt.

Ein-Ausgänge

VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen. Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistentist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Regleraktivierung

rW : Sollwert.

rX : Istwert.

tTaskCycleTime: Zykluszeit, mit der der Funktionsbaustein aufgerufen wird. Diese entspricht der Task-Zykluszeit der aufrufenden Task, wenn der Baustein in jedem Zyklus aufgerufen wird.

uiCtrlCycleCall : Aufrufzyklus des Bausteines als Vielfaches der Zykluszeit. . Ein Nulleintrag wird automatisch als *uiCtrlCycleCall* =1 gewertet.
Beispiel: *tTaskCycleTime* = 20ms, *uiCtrlCycleCall* =10 -> Der Regelalgorithmus wird alle 200ms aufgerufen. Damit werden aber auch nur alle 200ms die Ausgänge aktualisiert.

bSync: Eine steigende Flanke an diesem Eingang setzt den (internen) I-Anteils so, dass am Stellgrößenausgang *rSyncValue* ausgegeben wird. Ist der I-Anteil hingegen durch *tTn*=0ms deaktiviert, so wird dieses Kommando ignoriert.

VAR_OUTPUT

```
rY         : REAL;
rE         : REAL;
bARW       : BOOL;
bMaxLimit  : BOOL;
bMinLimit  : BOOL;
bError     : BOOL;
udiErrorId: UDINT;
```

rY : Stellgröße.

rE : Regelabweichung (Berechnung abhängig vom Wirksinn [▶ 119])

rEMin : unterer Regelabweichungs-Grenzwert, welcher sich aus dem eingegebenen Proportionalband ergibt.

rEMax : oberer Regelabweichungs-Grenzwert, welcher sich aus dem eingegebenen Proportionalband ergibt.

bARW : Anti-Reset-Windup-Funktion ist aktiv.

bMaxLimit : Die Stellgröße hat ihren oberen Grenzwert erreicht.

bMinLimit : Die Stellgröße hat ihren unteren Grenzwert erreicht.

bError : Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId : Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

VAR_IN_OUT

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

bDirection: Mit dem Parameter *bDirection* kann der Wirksinn [▶ 119] des Reglers verändert werden. Ist *bDirection* TRUE**,** ist der direkte Wirksinn für einen Kühlbetrieb des Reglers aktiv. Wenn *bDirection* FALSE ist, ist der indirekte Wirksinn des Reglers für den Heizbetrieb aktiviert. Die Variable wird persistent gespeichert. Voreingestellt auf FALSE.

rXp: Proportionalband. Dieses definiert den internen Proportionalfaktor, siehe unten. Der Proportionalitätsfaktor, bzw die Verstärkung wirkt nur auf den P-Anteil. Die Variable wird persistent gespeichert. Voreingestellt auf 100.0.

tTn: Nachstellzeit (Integrierzeit) in Sekunden. Der I-Anteil korrigiert die verbleibende Regelabweichung nach der Korrektur des P-Anteils. Je kleiner die *tTi* eingestellt wird, desto schneller korrigiert der Regler. Ist die Zeit zu kurz, wird der Regelkreis instabil. Um den Integrationsanteil zu vermindern, sind größere *tTi*-Zeiten einzugeben. Die Nachstellzeit sollte größer als die Verfahrzeit des Ventil- oder Klappenantriebes gewählt werden. Ein Nullwert an diesem Eingang schaltet den I-Anteil ab. Die Variable wird persistent gespeichert. Voreingestellt auf 30s.

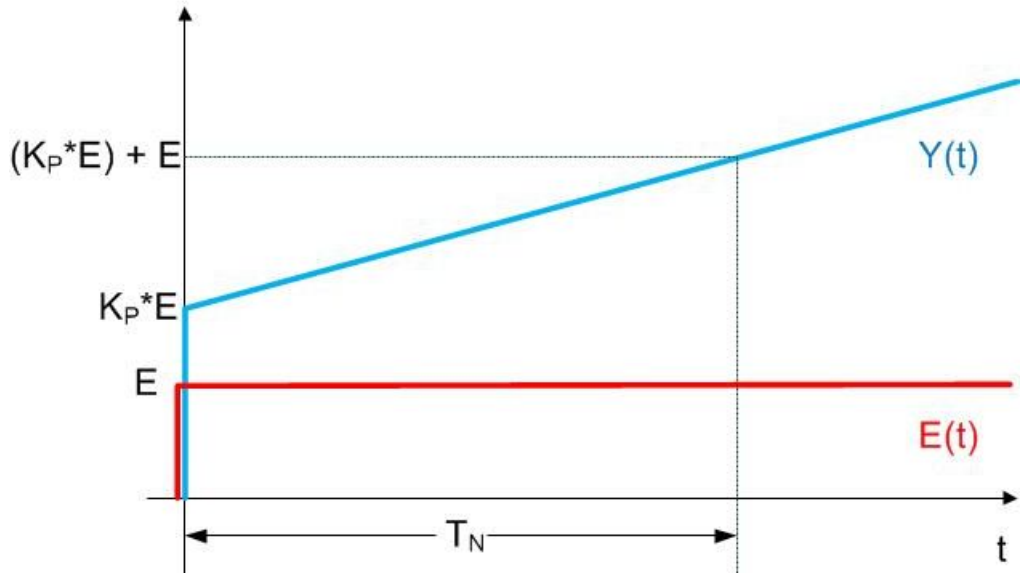
rYMin / rYMax: Begrenzen den Arbeitsbereich des Reglers. Einige andere Funktionsbausteine z.B. Sequenzer erfordern einen symmetrischen (- 100 bis +100) Stellbereich. Bei einer Kaskadenstruktur bestimmt der Arbeitsbereich des Führungsreglers den Sollwert des Folgereglers. Zum Beispiel 15° bis 25° als Begrenzung des Zulufttempertursollwerts einer Abluft- Zuluftkaskadenregelung. Die Variable wird persistent gespeichert. Voreingestellt auf 0.0 bzw. 100.0.

rSyncValue: Mit einer steigenden Flanke am Eingang *bSync* wird die Stellgröße *rY* auf diesen Wert gesetzt. Dazu wird intern der I-Anteil verändert. Ist der I-Anteil nicht gegeben (PD-Regler), so wird der D-Anteil verändert. Die Variable wird persistent gespeichert. Voreingestellt auf 0.0.

Funktionsbeschreibung

Sprungantwort eines einfachen PI-Reglers, wobei die Regelverstärkung keinen Einfluss auf den Integralanteil hat.
Reaktion des Ausganges Y(t) auf einen Regelabweichungssprung um E: Bei einem Sprung der Regelabweichung um E springt der Ausgang Y durch den Proportionalanteil zunächst auf K_P*E und wächst dann in jedem Intervall T_N um ein weiteres E an.

Der Regler ist so konzipiert, dass bei einem SPS-Reset bzw. Neustart der Regler bei 0, also ohne den KP*E-Sprung anfängt.



Grundlegende Funktion

Ein TRUE-Signal am Eingang *bEnable* aktiviert den Baustein. Der interne Regelalgorithmus wird nun abgearbeitet. Der Eingangswert *uiCtrlCycleCall* gibt dabei vor, nach wie vielen SPS-Zyklen der interne Regelalgorithmus abgearbeitet wird. Ist *uiCtrlCycleCall* =1, so erfolgt die Neuberechnung in jedem SPS-Zyklus, ist er hingegen auf 100 eingestellt, so erfolgt eine erneute Berechnung der Ausgangsgrößen nur alle hundert SPS-Zyklen. Die SPS-Zykluszeit geht mit in die Stellgrößenberechnung ein. Ein falscher Eingabewert führt zu fehlerhafter Berechnung.
Die Eingänge *rW* (Sollwert), *rX* (Istwert), *rXp* (Proportionalband) und *rTn* (Nachstellzeit) sind die Eingangsgrößen des PI-Reglers. Mit Ihnen werden in jedem Berechnungszyklus die Ausgangsgrößen rY (Stellgröße) und *rE* (Regelabweichung) ermittelt. Die Stellgröße lässt sich zusätzlich durch die Eingänge *rYMin* und *rYMax* begrenzen.

Einstellung über das Proportionalband

Die Einstellung des Verstärkungsfaktors K_p eines Reglers birgt oft die Schwierigkeit für den Anwender, dass der Größenbezug zur Anwendung fehlt. Arbeitet eine Heizungsregelung normalerweise im zweistelligen Bereich, so kann eine Volumenstromregelung Werte im fünfstelligen Bereichen annehmen. Es macht daher Sinn, den K_p-Faktor so darzustellen, dass er einen Bezug zur möglichen Regelabweichung und Stellgrößenänderung hat. Zur Dimensionierung des K_p-Faktors betrachtet man den P-Anteil des Reglers. Für diesen gilt:

- Stellgröße = Regelabweichung x Verstärkungsfaktor→ $Y=E*K_p$

dieser Zusammenhang gilt auch für die Änderungen der Regelabweichung und der Stellgröße:

- Stellgrößen-Änderung =Regelabweichungs-Änderung • Verstärkungsfaktor→ $\Delta Y=\Delta E*K_p$

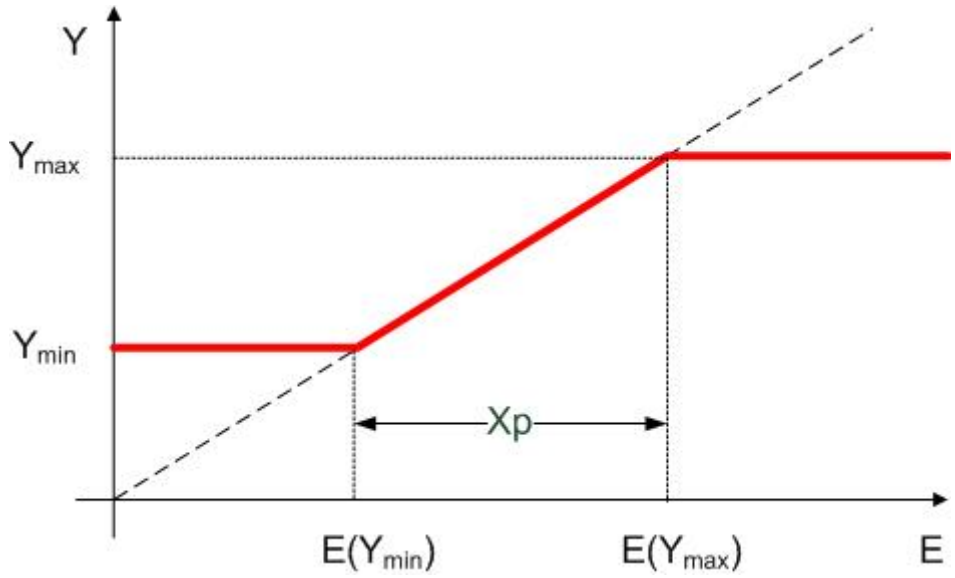
Bezogen auf den Minimal- und Maximalwert der Stellgröße, Y_{min} und Y_{max}:

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

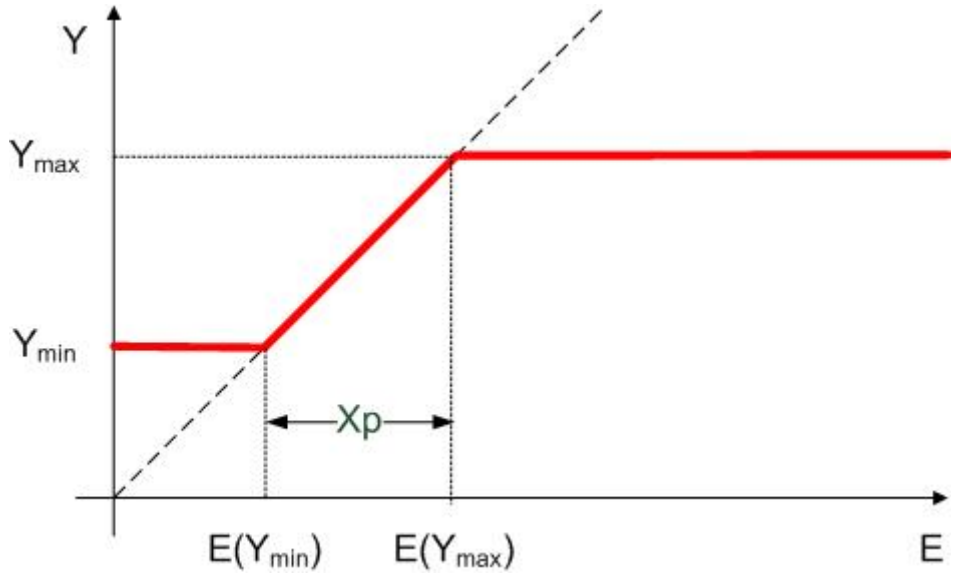
Diese Differenz, $E(Y_{max})-E(Y_{min})$, wird Proportionalband (Xp) genannt. Umgestellt lautet die Gleichung dann:

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

Das folgende Diagramm verdeutlicht den funktionalen Zusammenhang:



Das Proportionalband X_p gibt demnach die Größe des Bereiches der Regelabweichung an, die am Regler ausgangsseitig zu einer Ausgabe von Y_{min} bis Y_{max} führen.
Ein kleineres $X_p$führt zu einer steileren Funktion und damit zu einer Erhöhung des Kp-Faktors. Allerdings verschieben sich die Regelabweichungs-Grenzwerte $E(Y_{max})-E(Y_{min})$:



Wirksinn

Wirksinn

Mit *bDirection* = FALSE wird der Wirksinn des Reglers so umgekehrt , dass eine Regelabweichung kleiner als 0 eine Stellgrößenänderung ins Positive bewirkt. Dies wird dadurch erreicht, daß die Regelabweichung negativ berechnet wird:

bDirection	**rE (Regelabweichung)**	Wirksinn
TRUE	lrX-lrW (Istwert-Sollwert)	direkt (Kühlen)
FALSE	lrW-lrX (Sollwert-Istwert)	indirekt (Heizen)

Anti-Reset-Windup (ARW)

"Läuft" der Regler in diese Begrenzung, so wird der I-Anteil intern auf dem letzten Wert festgehalten. Würde dies nicht gemacht werden, so könnte der I-Anteil während des Begrenzungsfalles ungehindert sehr große Werte annehmen, die im Falle einer Wirksinn-Umkehrung des Reglers zunächst wieder eliminiert werden müssten. Diese Funktion wird "Anti-Reset-Windup" (ARW). Ist diese Funktion im Eingriff, so ist der Ausgang *bARW* gesetzt.

Sonderfall Tn=0 als Abschaltung des I-Anteiles

Aus dem oben aufgeführten Sprungantwort-Diagramm geht hervor, dass der Einfluss des I-Anteiles umso schwächer ist, je größer die Nachstellzeit T_n eingestellt wurde. Geht die Nachstellzeit gegen Unendlich, so ist der Einfluss vom I-Anteil gleich Null. Umgekehrt würde eine immer kleiner werdende Nachstellzeit den Einfluss des I-Anteiles wachsen lassen, mit T_n=0 ginge die Stellgröße gegen unendlich. Dieser Sonderfall jedoch dazu genutzt, den I-Anteil **abzuschalten**. Es handelt sich hierbei um eine intern gebildete Ausnahme, da die Nachstellzeit direkt zum I-Anteil gehört und durch den Nulleintrag auch bildlich eine Abschaltung zur Folge haben soll.

Synchronisation

Eine positive Flanke an *bSync* setzt den Reglerausgang *rY* unmittelbar auf *rSyncValue*, sofern der Regler durch ein TRUE.-Signal an *bEnable* aktiviert ist. Ist dies nicht der Fall, bleibt die positive Flanke an *bSync* unberücksichtigt.

Fehlerfall/Baustein nicht aktiviert

Ist der Regler fehlerhaft parametriert, so wird die Abarbeitung gestoppt, der Ausgang *bError* gesetzt und an *udiErrorID* die entsprechende Fehlerkennung ausgegeben, siehe Fehlercodes [▶ 243]. Der Baustein ist ebenfalls gestoppt, wenn der Eingang *bEnable* nicht gesetzt ist. In beiden Fällen sind die Ausgänge wie folgt gesetzt:

rY	0.0
rE	0.0
bARW	FALSE
bMaxLimit	FALSE
bMinLimit	FALSE

Dokumente hierzu

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

3.3.3 Beleuchtung

3.3.3.1 Beleuchtungsfunktionen - Übersicht

Die Beleuchtung untergliedert sich in zwei verschiedene Funktionsgruppen, die sich zu beliebigen Lösungen kombinieren lassen:

- Anwenderfunktionen
- Aktorfunktionen

Die **Anwenderfunktionen** sind Steuer- und Regelkreise, welche bedient durch die Sensorfunktionen jeweils einen Lichtwert ausgeben.

FB_BARLightCircuit [▶ 137] - Einfacher Lichtschaltkreis ohne Dimmfunktionalität.

FB_BARLightCircuitDim [▶ 138] - Einfacher Lichtschaltkreis mit Dimmfunktionalität.

FB_BARAutomaticLight [▶ 120] - Präsenzgesteuertes Automatiklicht mit Ausschaltverzögerung.

FB_BARStairwellAutomatic [▶ 142] - Treppenhausbeleuchtung mit Vorwarnsequenz.

FB_BARTwilightAutomatic [▶ 145] - Dämmerungsschaltung.

FB_BARDaylightControl [▶ 130] - Tageslichtsteuerung ohne Dimmvorgänge

FB_BARConstantLightControl [▶ 123] - Konstantlichtregelung mit kontinuierlicher Ausgabe von Analogwerten.

Die Gruppe der **Aktorfunktionen** wird zur Zeit von nur einem Baustein repräsentiert.

FB_BARLightActuator [▶ 134] - Ausgabe eines vorgegebenen prozentualen Dimmwertes über eine Rampenfunktion. Die Ausgabe erfolgt wahlweise in Prozent, INTEGER oder BOOL. Dieser Baustein umfasst ebenfalls einen Lichtszenenspeicher von 21 einstellbaren Dimmwerten.

3.3.3.2 FB_BARAutomaticLight

Funktionsbaustein für eine Automatiklicht-Schaltung, wie sie in Fluren oder sanitären Anlagen zur Anwendung kommen.

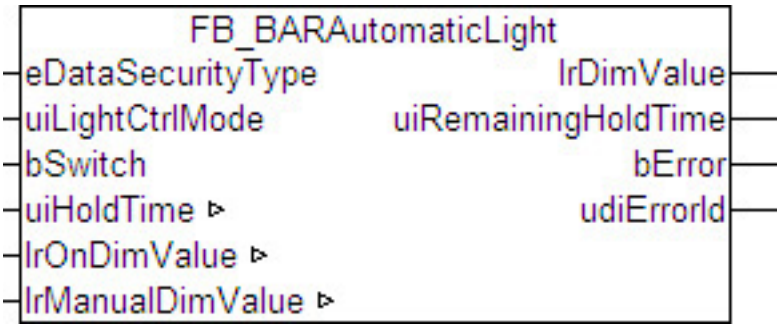
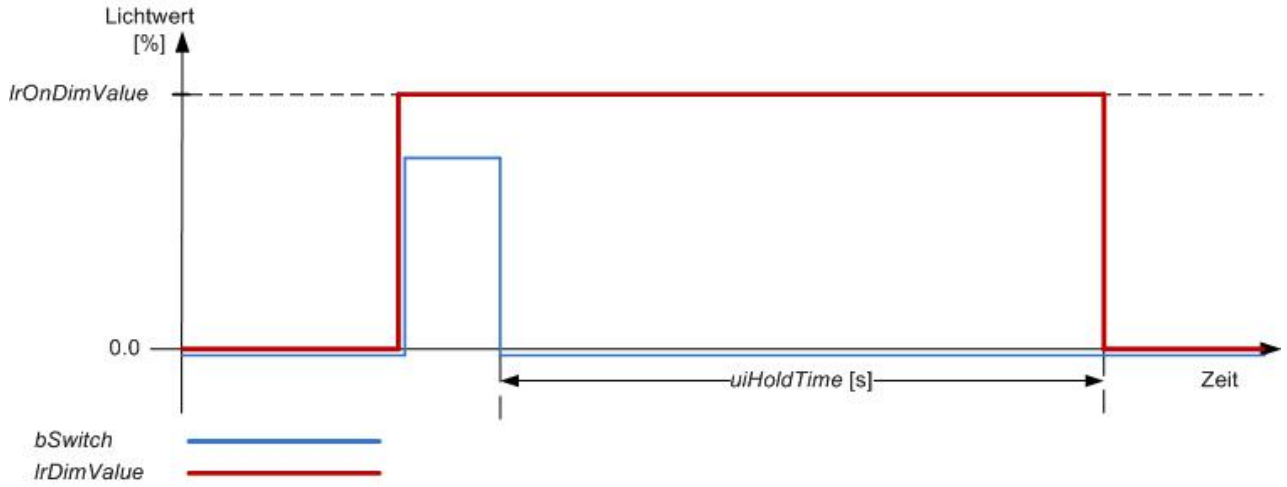


Abb. 2: FB_BARAutomaticLight

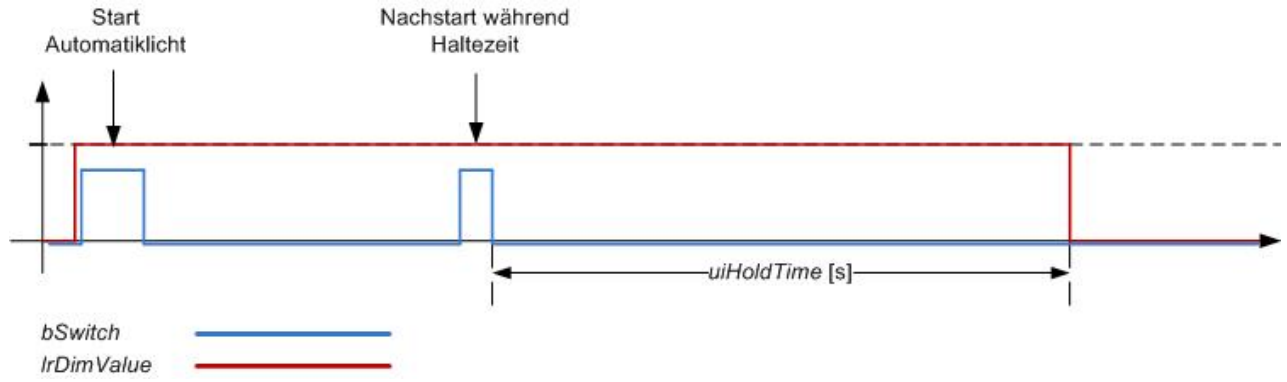
Der Baustein kennt 3 verschiedene Modi, welche über den Eingang *uiLightCtrlMode* eingestellt werden können:

- Automatikbetrieb
- Hand-Ein-Betrieb
- Hand-Aus-Betrieb

Im Automatikbetrieb (*uiLightCtrlMode*=0) ist die Automatiklicht-Schaltung aktiv. Eine positive Flanke an *bSwitch* setzt den Ausgang *lrDimValue* auf den unter *lrManualDimValue* eingetragenen Wert. Mit einer negativen Flanke wird der Haltezeitgeber gestartet. Ist die Haltezeit *uiHoldTime* [s] abgelaufen, wird der Ausgang *lrDimValue* wieder auf 0.0 gesetzt.



Die Sequenz kann dabei jederzeit nachgestartet werden:



In den Hand-Betriebsmodi ist der Eingang *bSwitch* ohne Funktion: bei *uiLightCtrlMode*=1 wird der Ausgabewert *lrDimValue* konstant auf *lrManualDimValue* und bei *uiLightCtrlMode*=2 konstant auf 0.0 gesetzt.
Ein Wechsel in den Handmodus setzt eine bis dahin gestartete Haltezeit zurück.

VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Andernfalls wird der instanziierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

uiLightCtrlMode : Betriebsmodus.

- 0: Automatikbetrieb, die Automatiklicht-Schaltung ist aktiv und reagiert auf den Eingang *bSwitch*.
- 1: Hand-Ein, die Automatiklicht-Schaltung ist nicht aktiv - der Ausgang *lrDimValue* wird konstant auf *lrManualDimValue* gesetzt.
- 2: Hand-Aus, die Automatiklicht-Schaltung ist nicht aktiv - der Ausgang *lrDimValue* wird konstant auf 0.0 gesetzt.

bSwitch: Eine steigende Flanke schaltet im Automatikbetrieb (*uiLightCtrlMode*=0) das Licht ein, eine fallende Flanke startet den Haltezeitgeber. Im Handbedienmodus (*uiLightCtrlMode*=1 oder 2) hat dieser Eingang keine Funktion.

VAR_OUTPUT

```
lrDimValue          : LREAL;
uiRemainingHoldTime: UINT;
bError              : BOOL;
udiErrorId          : UDINT;
```

lrDimValue : Ausgabedimmwert für die Beleuchtung in Prozent.

uiRemainingTimeHold : Verbleibende Haltezeit in in Sekunden. Ist das Licht aus oder der Hand-Bedienmodus aktiv, so steht dieser Ausgang auf "0". Mit einer steigenden Flanke an *bSwitch* im Automatikmodus zeigt dieser Ausgang zunächst die komplette Anzahl der Sekunden der Haltezeit (*uiHoldTime*) an, um, beginnend mit einer fallenden Flanke an *bSwitch,* das Ablaufen der Haltezeit darzustellen. Solange kein Herunterzählen der Zeit stattfindet, steht dieser Ausgang auf 0.

bError : Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId : Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

VAR_IN_OUT

Damit die eingetragenen Parameter über einen Steuerungsausfall hinweg erhalten bleiben ist es erforderlich, sie als In-Out-Variablen zu deklarieren. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variablen wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

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

uiHoldTime : Haltezeit [s] der Automatiklicht-Steuerung nach fallender Flanke an *bSwitch.*

lrOnDimValue : Ein-Dimmwert im Automatik-Modus (uiLightCtrlMode=0).

lrManualDimValue : Ausgabe-Dimmwert im Hand-Ein-Modus (uiLightCtrlMode=1).

3.3.3.3 FB_BARConstantLightControl

Der Funktionsbaustein Konstantlichtregelung regelt die Beleuchtung bei Raumbelegung so, dass eine eingestellte Mindestbeleuchtungsstärke nicht unterschritten wird. Dabei wird eine ausreichende Beleuchtung bei gleichzeitig minimalem Energieverbrauch sichergestellt.

Die Freigabe der Konstantlichtregelung erfolgt beim Betreten des Raums bzw. bei einer steigenden Flanke an den Eingang *bPresence*. Optional kann die Konstantlichtregelung auch über einen Taster bedient werden. Bei einem Kurzimpuls an dem Eingang *bToggle* wird die Konstantlichtregelung je nach aktuellem Zustand frei gegeben oder ausgeschaltet. Falls viele Konstantlichtregler mit einem Zentralein-Kommando eingeschaltet werden sollen zum Beispiel in einer Büroetage, kann dieses mit dem Eingang *bCentralOn* realisiert werden. Ein Zentralbefehl zur Ausschaltung kann am Eingang *bCentralOff* angeschlossen werden.

Bei langem Betätigen der Taste an *bToggle* wird der Sollwert für die Raumhelligkeit manuell verändert. Bei einer fallenden Flanke von *bToggle* wird der aktuelle Wert der Raumhelligkeit als Sollwert abgespeichert. Die Änderungsgeschwindigkeit beim manuellen Erhöhen bzw. Absenken der Raumhelligkeit kann mittels eines Parameters eingestellt werden.

Der manuell eingestellte Sollwert für die Konstantlichtregelung bleibt bis zum nächsten Abschalten der Beleuchtung erhalten. Beim Neustart der Regelung bedingt durch die Ereignisse, steigende Flanke von *bPresence*, steigende Flanke an *bCentralOn* oder einem Impuls an *bToggle* wird je nach Einstellung des Parameter *bInitialMode* der zuletzt manuell gewählte Sollwert oder der Wert von *uiSetpointlValue* übernommen.

Im Automatikbetrieb erfolgt die Einschaltung der Beleuchtung nur dann wenn die Raumhelligkeit unter einem einstellbare Hysteresewert (*uiTargetRange*) liegt. Bei steigender Außenhelligkeit reduziert der Konstantlichtregler den Kunstlichtanteil so lange bis ein minimaler Laststellwert *lrMinDimValue* am Ausgang des Reglers lrDimValue ansteht. Anschließend schaltet der Regler die Beleuchtung zeitverzögert mit dem Timer *uiOffDelay* ab.
Ein erneutes Einschalten der Beleuchtung bei abnehmender Außenhelligkeit wird mit dem Timer *uiOnDealy* verzögert.

Um unangenehme sichtbare Änderungen der Helligkeit zu vermeiden, wird die Änderungsgeschwindigkeit des Stellsignals mit dem Parameter *uiControlRampTime* verzögert.

Für Wartungs- und Testwecke kann die Automatik der Konstantlichtregelung deaktiviert und die Beleuchtung im Handbetrieb ein- und ausgeschaltet werden. Die Flankenerkennung der Eingänge *bCentrolOn,bCentalOff* und *bToggle* ist damit deaktiviert.

Abb. 3: FB_BARConstantLightControl

Der Baustein kennt drei verschiedene Modi, welche über den Eingang *uiLightCtrlMode* eingestellt werden können:

- Automatikbetrieb
- Hand-Ein-Betrieb
- Hand-Aus-Betrieb

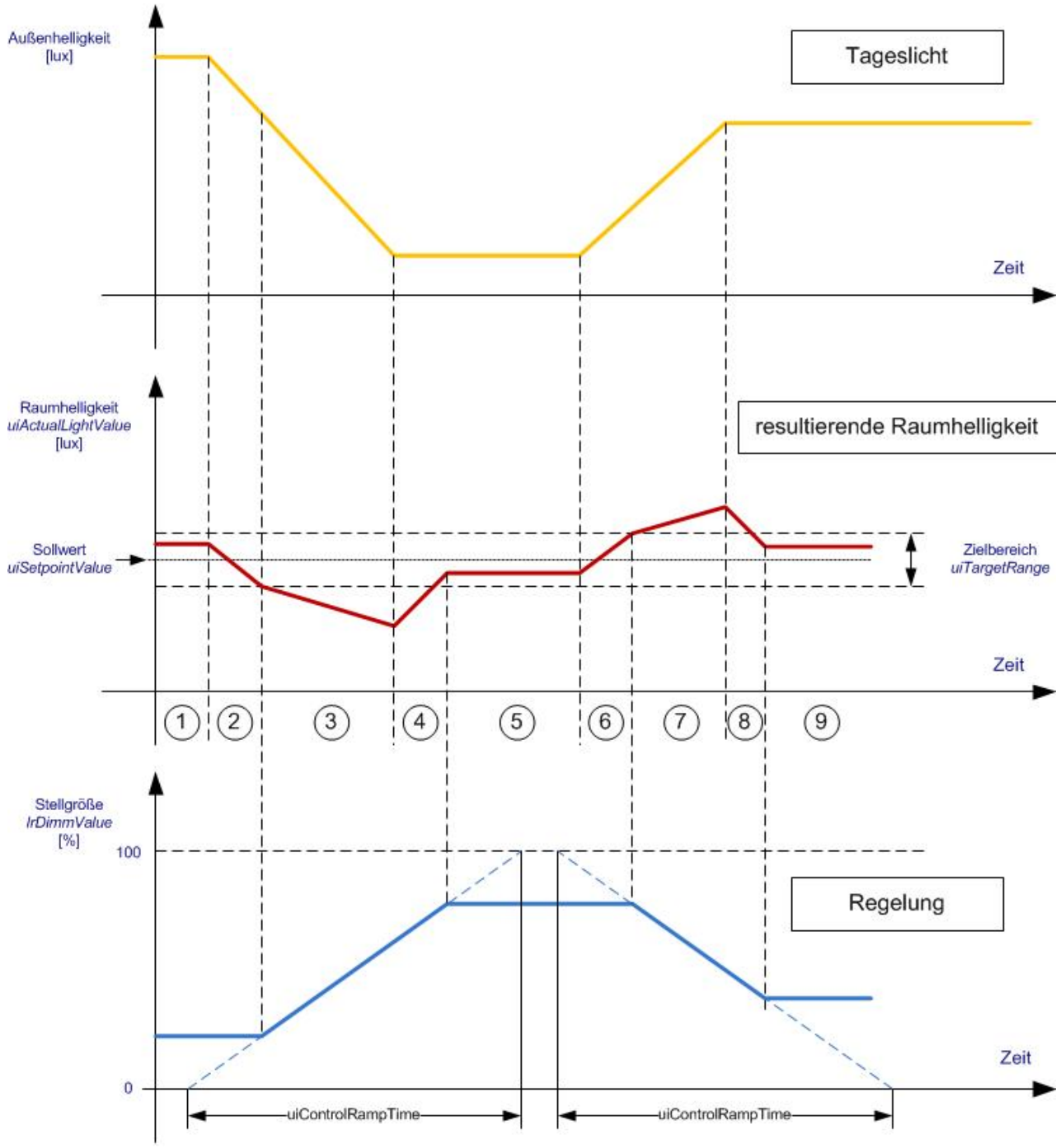
Automatikbetrieb

Im Automatikbetrieb kann die Regelung auf drei verschiedene Weisen aktiviert bzw. deaktiviert werden:

1. Über den Eingang *bSwitch*: Ein kurzes TRUE-Signal (kürzer als *uiSwitchOverTime* in Millisekunden) an *bSwitch* schaltet die Konstantlichtregelung aktiv, sofern sie vorher inaktiv war. Eine erneutes Kurzsignal an *bSwitch* deaktiviert die Konstantlichtregelung wieder. Ein langes TRUE-Signal an *bSwitch* lässt den Baustein in die Sollwert-Nachführung wechseln, dies ist weiter unten beschrieben.
2. Über den Eingang *bPresence*: Die Einschaltmöglichkeit über diesen Eingang muss explizit durch ein TRUE-Signal an *bPresenceOnActive* (VAR_IN_OUT) freigegeben werden. Dann aktiviert eine steigende Flanke die Regelung, während eine fallende Flanke die Regelung immer deaktiviert, unabhängig vom Zustand von *bPresenceOnActive* .
3. Über TRUE-Flanken an den Eingängen **bCentralOn** und *bCentralOff* wird die Regelung explizit ein- und ausgeschaltet, unabhängig vom vorherigen Zustand der Regelung. Diese Eingänge wirken nicht übersteuernd: ist beispielsweise über *bCentralOff* ausgeschaltet worden kann über *bSwitch* jederzeit auch wieder eingeschaltet werden.

Der Lichtausgabewert *lrDimValue* wird im aktiven Zustand zunächst auf den Startwert *lrStartDimValue* gesetzt. Darauf hin findet ein kontinuierlicher Ist-Sollwert-Vergleich statt. Läuft der Raum-Istwert *uiBrightness* [lux] dabei aus einem Zielbereich *uiTargetRange* [lux] um den Sollwert *uiSetpointValue* [lux] heraus, so wird dem durch Auf- bzw. Abdimmen der Beleuchtung entgegen geregelt. Die Regelung arbeitet dabei immer mit

einer konstanten, parametrierbaren Dimmrampe *uiRamptime* [s], welche die Zeit eines kompletten Dimmvorgangs von 0% bis 100% angibt. Ebenfalls konstant sind die Zielwerte der Regelung: 0% für ab- und 100% für Aufdimmen. Im inaktiven Zustand wird der Lichtausgabewert *lrDimValue* auf 0.0 gesetzt.

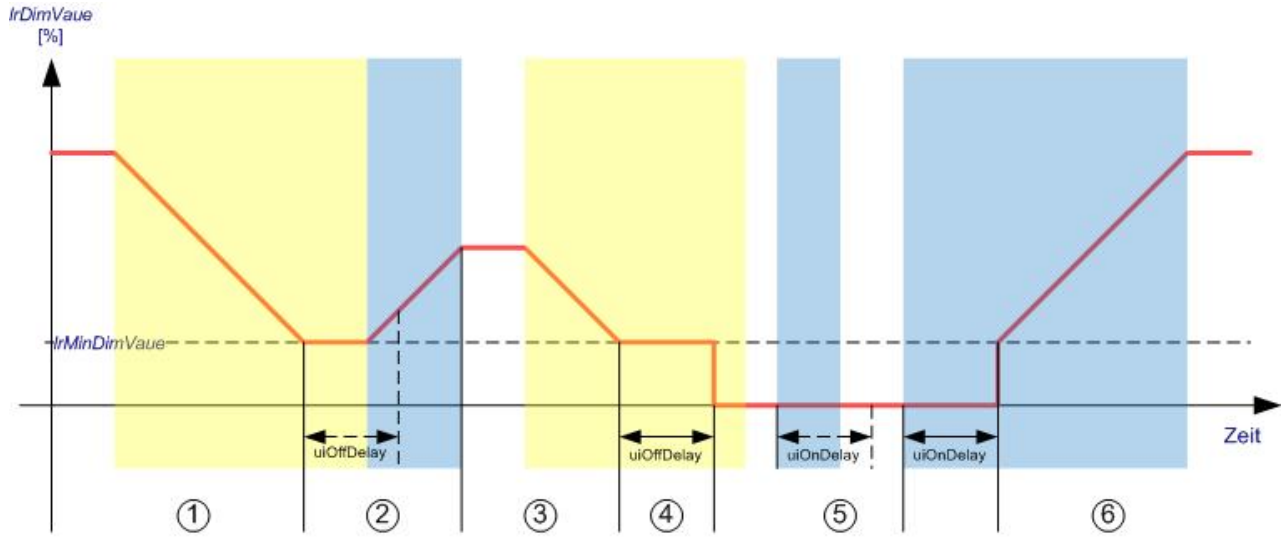


1. Ausgangssituation: Die Raumbeleuchtung ist ausgeregelt.
2. Die Außenhelligkeit sinkt, damit auch die des Raumes, jedoch noch im Zielbereich. Die Stellgröße ändert sich daher noch nicht.
3. Die Außenhelligkeit sinkt weiter, die Raumhelligkeit verlässt den Zielbereich des Sollwertes. Die Regelung startet nun ihre Rampe, schafft es jedoch nur die Steiheit der Verdunkelung abzumildern
4. Die Außenhelligkeit sinkt nicht weiter. Die vorgegebene Rampe der Regelung schafft es nun, der Verdunkelung entgegen zu wirken.
5. Der Zielbereich ist wieder erreicht und die Raumbeleuchtung ist ausgeregelt.
6. Die Außenhelligkeit steigt wieder, damit auch die des Raumes, jedoch noch im Zielbereich. Die Stellgröße ändert sich daher noch nicht.
7. Die Außenhelligkeit steigt weiter, die Raumhelligkeit verlässt den Zielbereich des Sollwertes. Die Regelung startet nun ihre Rampe, schafft es jedoch nur die Steiheit der Aufhellung abzumildern
8. Die Außenhelligkeit steigt nicht weiter. Die vorgegebene Rampe der Regelung schafft es nun, der Aufhellung des Raumes entgegen zu wirken.
9. Der Zielbereich ist wieder erreicht und die Raumbeleuchtung ist ausgeregelt.
   Da jetzt weniger Tageslicht zur Verfügung steht wie noch im Bereich 1, ist hier der Anteil künstlichen Lichts größer.

Ein- und Ausschaltverzögerung, Mindestausgabewert

Steigt die Außenhelligkeit, so wird immer weniger künstliches Licht notwendig um die gewünschte Gesamthelligkeit zu erreichen. Bei ausreichender Außenhelligkeit kann die Beleuchtung auch ganz abgeschaltet werden.
Ein- und Ausschaltvorgänge können jedoch als störend empfunden werden, genauso wie sehr niedrige Ausgabe-Dimmwerte. Daher können an dem Baustein eine Ein- und Ausschaltverzögerung *uiOnDelay*/ *uiOffDelay* um einen unteren Grenzwert *lrMinDimValue* definiert werden. Sinkt der intern ermittelte Ausgabewert unter diesen Mindestwert, so verharrt die Ausgabe auf diesem Minimum für die Zeit *uiOffDelay* [s]. Erst danach wird auf am Ausgang *lrDimValue* 0.0 ausgegeben. Ist im umgekehrten Fall einmal auf 0.0 ausgeschaltet worden, so wird bei Bedarf an künstlichem Licht dieses erst nach Ablauf von *uiOnDelay* [s] und dann auf den Wert *lrMinDimValue* eingeschaltet. Das folgende Diagramm soll das Verhalten verdeutlichen:

Die gelben Flächen zeigen die Bereiche, wo die Außenhelligkeit ein Abdimmen der Beleuchtung zulässt, bei den blauen Bereichen hingegen reicht die Außenhelligkeit alleine nicht aus um die gewünschte Raumhelligkeit zu erzielen. In den weißen Flächen ist die ausgegebene Stellgröße *lrDimValue* den Lichtverhältnissen angemessen.



1. Das Außenlicht reicht zunächst aus und der Ausgabewert wird über die Rampe gemindert.
2. Am Beginn dieses Abschnitts beginnt die Wartezeit, ganz auf 0 schalten zu können. Noch Vor Ablauf der Zeit jedoch wird es wieder dunkel und der Baustein hebt den Ausgabewert unverzögert wieder an.
3. Das Außenlicht reicht zunächst wieder aus. Bei zunehmender Aussenhelligkeit – gelber Bereich - wird der Ausgabewert über die Rampe gemindert.
4. Hier ist die Außenhelligkeit so lange ausreichend, dass ein Abschalten auf 0.0 zugelassen wird.
5. Eine kurze Phase geringen Außenlichts reicht nun nicht aus um den Lichtausgabewert lrDimmValue wieder anzuheben.
6. Erst bei einer längeren Dunkel-Phasewird das Licht wieder eingeschaltet und hochgestellt.

Manuelle Nachführung des Sollwertes

Um die Lichtregelung persönlichen Helligkeitsbedürfnissen anpassen zu können, ist die Möglichkeit gegeben, den Sollwert zu erhöhen oder zu erniedrigen. Ein langes TRUE-Signal an *bSwitch* (länger als *uiSwitchOverTime* in Millisekunden) lässt den Baustein in den Dimmmodus wechseln und das Licht wird aufgedimmt, wenn im vorhergehenden Dimmmodus abgedimmt wurde und umgekehrt. Fällt *bSwitch* wieder auf FALSE, so wird der nun gemessene Helligkeitswert an *uiBrightness* als neuer Sollwert angenommen, auf den es zu regeln gilt.

Handbetrieb

In den Hand-Betriebsmodi sind die Eingänge *bSwitch*, *bCentralOn* und *bCentralOff* ohne Funktion: bei *uiLightCtrlMode*=1 wird der Ausgabewert *lrDimValue* konstant auf *lrManualDimValue* und bei *uiLightCtrlMode*=2 konstant auf 0.0 gesetzt.

Ein Wechsel in den Handmodus setzt eine zuvor gestartete Regelung zurück. Beim Wiedereintritt in den Automatik-Modus ist der Ausgangswert 0.0 und die Regelung muss neu gestartet werden.*lrDimValue*

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
uiLightCtrlMode  : UINT;
bSwitch          : BOOL;
bPresence        : BOOL;
bCentralOn       : BOOL;
bCentralOff      : BOOL;
uiBrightness     : UINT; (*lux*)
uiSetpoint       : UINT; (*lux*)
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Andernfalls wird der instanziierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

uiLightCtrlMode : Betriebsmodus.

- 0: Automatikbetrieb, die Befehle *bSwitch*, *bCentralOn* und *bCentralOff* sind ausführbar und die Regelung lässt sich damit ein- und ausschalten.
- 1: Hand-Ein-Betrieb, die Konstantlichtregelung ist nicht aktiv - der Wert *lrManualDimValue* wird unverzögert an den Ausgang *lrDimValueOut* ausgegeben, alle anderen Befehlseingänge sind unwirksam.
- 2: Hand-Aus-Betrieb, die Konstantlichtregelung ist nicht aktiv - der Wert 0.0 wird unverzögert an den Ausgang *lrDimValueOut* ausgegeben, alle anderen Befehlseingänge sind unwirksam.

bSwitch : Positive Flanken an diesem Eingang schalten die Regelung wechselseitig ein und aus. Beim Ausschalten wird der Ausgang *lrDimValue* auf 0.0 gesetzt. Dieser Befehlseingang ist nur im Automatik-Modus (*uiLightCtrlMode*=0) aktiv.

bPresence : Eine dauerhaftes TRUE-Signal an diesem Eingang aktiviert die Regelung dann, wenn die Präsenz-Funktion durch *bPresenceOnActive*=TRUE (VAR_IN_OUT) aktiviert ist. Eine fallende Flanke an diesem Eingang hingegen deaktiviert die Regelung immer. Dieser Befehlseingang ist nur im Automatik-Modus (*uiLightCtrlMode*=0) aktiv.

bCentralOn : Eine positive Flanke an diesem Eingang schaltet die Regelung ein. Dieser Befehlseingang ist nur im Automatik-Modus (*uiLightCtrlMode*=0) aktiv.

bCentralOff : Eine positive Flanke an diesem Eingang schaltet die Regelung aus und den Ausgang *lrDimValue* auf 0.0. Dieser Befehlseingang ist nur im Automatik-Modus (*uiLightCtrlMode*=0) aktiv.

uiBrightness : Licht-Istwert [lux].

uiSetpointValue : Licht-Sollwert [lux].

VAR_OUTPUT

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

lrDimValue : Lichtausgabewert, 0..100%.

bControlActive : Dieser Ausgang ist TRUE, wenn der Baustein sich im Automatikbetrieb befindet und die Regelung eingeschaltet ist. Dies soll als zusätzliche Rückmeldung dienen, wenn eingeschaltet wurde, die Regelung jedoch einen Lichtwert von *lrDimValue*=0.0 ausgibt.

bAdjustedSetpointActive : Ist die Regelung aktiv und der Sollwert wurde manuell nachgeführt (s.o.), so wechselt der Zustand dieses Ausganges auf TRUE, um anzuzeigen, dass der Sollwert am Eingang *uiSetpointValue* nicht mehr aktiv ist.

uiAdjustedSetpoint : Dieser Ausgang zeigt den aktiven Sollwert an, wenn dieser manuell nachgeführt wurde (*bAdjustedSetpointActive* = TRUE). Ist kein nach geführter Sollwert aktiv, so wird dieser Ausgang auf 0 gesetzt.

diActDeviation : Aktuelle Regelabweichung in Lux. Dieser Ausgang zeigt nur dann einen gültigen Wert an, wenn der Baustein im Automatikmodus und eingeschaltet ist. Andernfalls wird 0.0 ausgegeben.

bError : Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId : Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

VAR_IN_OUT

Damit die eingetragenen Parameter über einen Steuerungsausfall hinweg erhalten bleiben ist es erforderlich, sie als In-Out-Variablen zu deklarieren. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variablen wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel:  https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

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

uiSwitchOverTime : Umschaltzeit in Millisekunden für den Eingang *bSwitch* zu Erkennung von Kurz- und Langsignal. (Kurzsignal: Ein-Ausschaltfunktion, Langsignal: Dimmfunktion)

uiAdjustRampTime : Rampenzeit in Sekunden mit welcher der Sollwert bei manueller Nachführung verstellt wird.

bPresenceOnActive : Steht dieser Eingang auf TRUE, so wird die Regelung durch eine positive Flanke an *bPresence* aktiviert werden, sofern der Baustein im Automatikmodus (*uiLightCtrlMode* = 0) ist.

bInitialMode : Ein TRUE-Signal an diesem Eingang lässt den Baustein nach jeder Aktivierung mit dem Sollwert an *uiSetpoint* beginnen. Steht dieser Eingang hingegen auf FALSE, so wird bei Aktivierung des Bausteines der Sollwert genommen, welcher zuletzt aktiv war, also auch manuell nachgeführte Sollwerte.

uiTargetRange : Zielbereich um den Sollwert in dem nicht weiter ausgeregelt wird.

uiControlRampTime : Rampenzeit in Sekunden (benötigte Zeit, um von 0% auf 100% zu dimmen).

uiOnDelay : Einschaltverzögerung in Sekunden um den Minimalwert *lrMinDimValue*.

uiOffDelay : Ausschaltverzögerung in Sekunden um den Minimalwert *lrMinDimValue*.

lrMinDimValue : Unterer Grenzwert, auf den gedimmt wird, siehe Einleitung [▶ 123].

lrMaxDimValue : Obere Grenze, auf die der Ausgang *lrDimValue* geregelt werden kann.

lrStartDimValue : Wert, auf den das Licht beim Start der Regelung springen soll.

lrManualDimValue : Ausgabe-Dimmwert im Hand-Ein-Modus (uiLightCtrlMode=1).

3.3.3.4 FB_BARDaylightControl

Tageslichtschaltung. Diese Automatikschaltung arbeitet im Gegensatz zur Konstantlichtregelung [▶ 123] nicht mit Dimmwerten, sondern schaltet das Licht in Abhängigkeit der gemessene Helligkeit nur ein und aus.

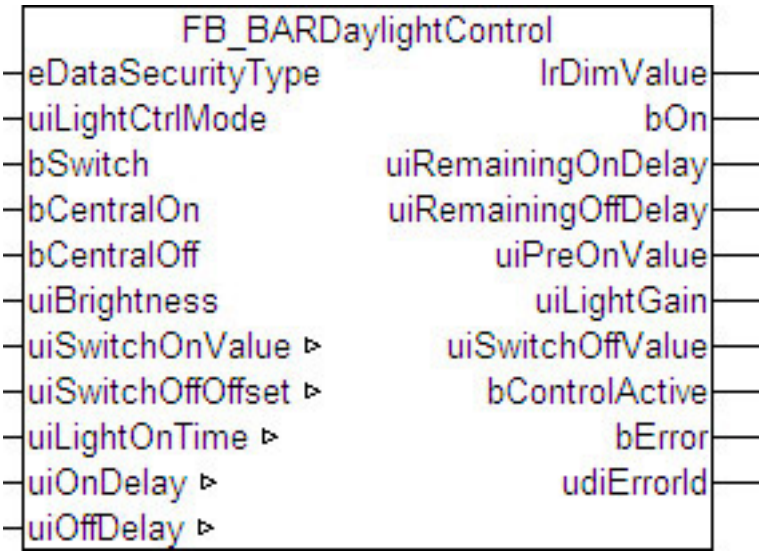


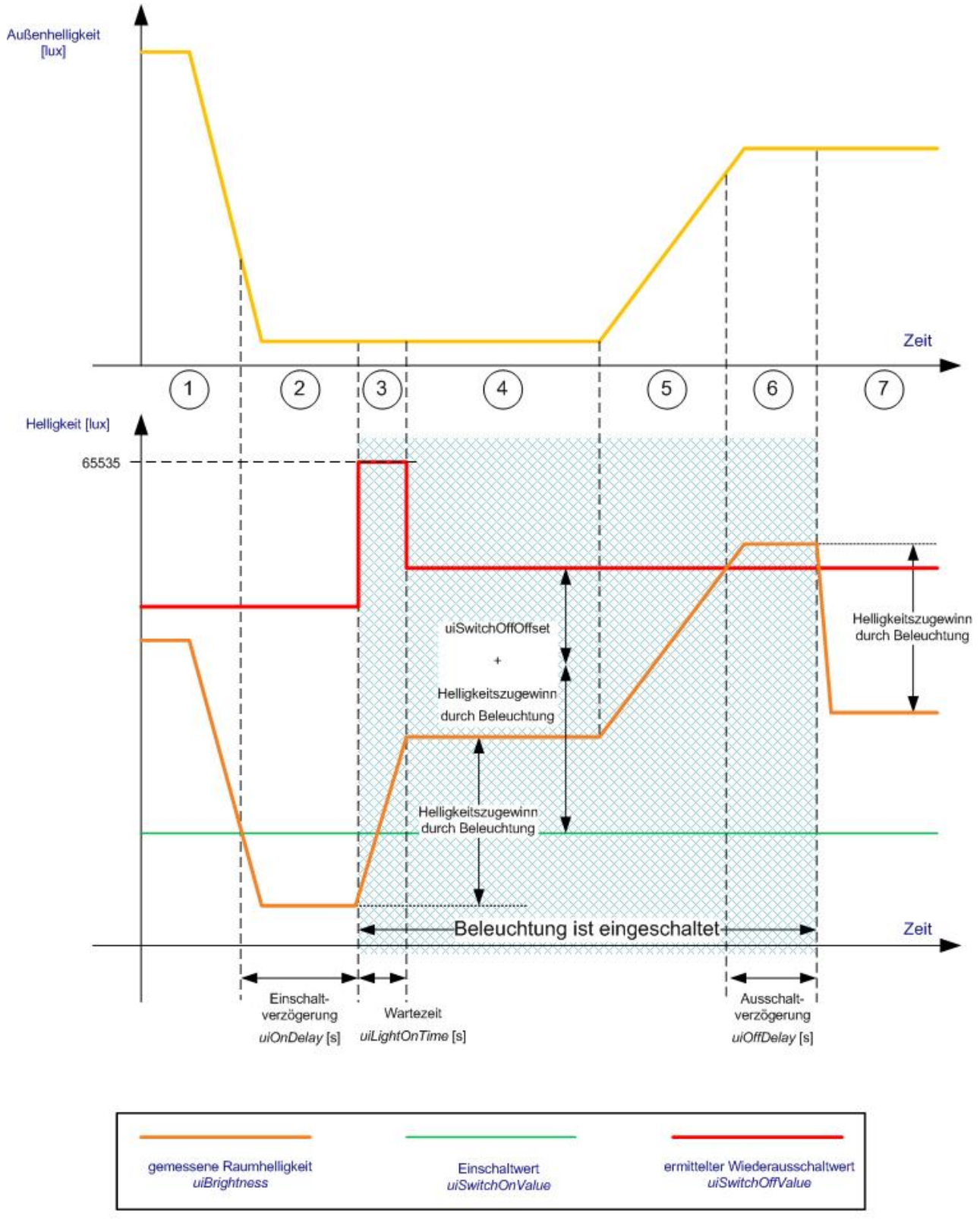
Abb. 4: FB_BARDaylightControl

Der Baustein kennt drei verschiedene Modi, welche über den Eingang *uiLightCtrlMode* eingestellt werden können:

- Automatikbetrieb
- Hand-Ein-Betrieb
- Hand-Aus-Betrieb

Automatikbetrieb

Unterschreitet die Helligkeit im Raum *uiBrightness* den Wert *uiSwitchOnValue* [lux] für die Zeit *uiOnDelay* [s], so wird das Licht eingeschaltet. Nun wird für die Zeit *uiLightOnTime* gewartet, bis die Leuchtmittel ihre volle Leuchtkraft erreichen. Die dann gemessene Helligkeit im Raum ist dann - **bei angenommen gleichbleibender Außenhelligkeit** - um den Helligkeitszugewinn höher als vor dem Einschalten. Ein Wiederausschalten der Beleuchtung macht erst dann Sinn, wenn die Außenhelligkeit wieder deutlich ansteigt. Dieser Grenzwert errechnet sich aus dem Lichteinschaltwert plus dem Helligkeitszugewinn plus dem Parameter *uiSwitchOffOffset*. Überschreitet die Raumhelligkeit durch wieder ansteigende Außenhelligkeit diesen ermittelten Wiederausschaltwert für die Dauer von *uiOffDelay* [s], so wird die Beleuchtung wieder ausgeschaltet.


Außenhelligkeit
[lux]
Zeit
1
2
3
4
5
6
7
Helligkeit [lux]
65535
uiSwitchOffOffset
+
Helligkeitszugewinn
durch Beleuchtung
Helligkeitszugewinn
durch Beleuchtung
Helligkeitszugewinn
durch Beleuchtung
Beleuchtung ist eingeschaltet
Zeit
Einschalt-
verzögerung
uiOnDelay [s]
Wartezeit
uiLightOnTime [s]
Ausschalt-
verzögerung
uiOffDelay [s]
gemessene Raumhelligkeit
uiBrightness
Einschaltwert
uiSwitchOnValue
ermittelter Wiederausschaltwert
uiSwitchOffValue

Handbetrieb

In den Hand-Betriebsmodi sind die Eingänge bSwitch, bCentralOn und bCentralOff ohne Funktion: bei *uiLightCtrlMode*=1 wird der Ausgabewert *bOn* auf *TRUE* und bei *uiLightCtrlMode*=2 auf FALSE gesetzt. **Ein Wechsel in den Handmodus setzt eine zuvor gestartete Regelung zurück. Beim Wiedereintritt in den Automatik-Modus ist der Ausgangswert bOn=FALSE und die Regelung muss neu gestartet werden.**

VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Andernfalls wird der instanziierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

uiLightCtrlMode : Betriebsmodus.

- 0: Automatikbetrieb, die Befehle *bSwitch*, *bCentralOn* und *bCentralOff* sind ausführbar und die Regelung lässt sich damit ein- und ausschalten.
- 1: Hand-Ein-Betrieb, die Konstantlichtregelung ist nicht aktiv - der Wert *lrManualDimValue* wird unverzögert an den Ausgang *lrDimValueOut* ausgegeben, alle anderen Befehlseingänge sind unwirksam.
- 2: Hand-Aus-Betrieb, die Konstantlichtregelung ist nicht aktiv - der Wert 0.0 wird unverzögert an den Ausgang *lrDimValueOut* ausgegeben, alle anderen Befehlseingänge sind unwirksam.

bSwitch : Positive Flanken an diesem Eingang schalten die Regelung wechselseitig ein und aus. Beim Ausschalten wird der Ausgang *lrDimValue* auf 0.0 gesetzt. Dieser Befehlseingang ist nur im Automatik-Modus (*uiLightCtrlMode*=0) aktiv.

bCentralOn : Eine positive Flanke an diesem Eingang schaltet die Regelung ein. Dieser Befehlseingang ist nur im Automatik-Modus (*uiLightCtrlMode*=0) aktiv.

bCentralOff : Eine positive Flanke an diesem Eingang schaltet die Regelung aus und den Ausgang *lrDimValue* auf 0.0. Dieser Befehlseingang ist nur im Automatik-Modus (*uiLightCtrlMode*=0) aktiv.

uiBrightness : Licht-Ist-Wert [lux].

VAR_OUTPUT

```
lrDimValue        : LREAL;
bOn               : BOOL;
uiRemainingOnDelay: UINT;
uiRemainingOffDel : UINT;
uiPreOnValue      : UINT;
uiLightGain       : UINT;
uiSwitchOffValue  : UINT;
bControlActive    : BOOL;
bError            : BOOL;
udiErrorId        : UDINT;
```

lrDimValue : Um diese Funktion zu den anderen Licht-Anwenderfunktionen gleich zu halten, existiert auch hier ein Lichtausgabewert als Fließkommazahl in Prozent, obwohl das Licht nur ein- oder ausgeschaltet wird. Das bedeutet: Licht aus: *lrDimValue* = 0.0, Licht an: *lrDimValue* = 100.0.

bOn : Schaltausgang für die Beleuchtung.

uiRemainingOnDelay : Countdown der Einschaltverzögerung in Sekunden. Solange kein Herunterzählen der Zeit stattfindet, steht dieser Ausgang auf 0.

uiRemainingOffDelay : Countdown der Ausschaltverzögerung in Sekunden. Solange kein Herunterzählen der Zeit stattfindet, steht dieser Ausgang auf 0.

uiPreOnValue : Gemessener Lichtwert unmittelbar vor dem Einschalten der Beleuchtung. Bei ausgeschalteter Steuerung oder im Handmodus seht dieser Ausgang auf "0".

uiLightGain : Errechneter Helligkeitszugewinn nach Einschalten der Beleuchtung und Ablauf der Wartezeit *uiFullLightTime*. Bei ausgeschalteter Steuerung oder im Handmodus seht dieser Ausgang auf "0".

uiSwitchOffValue : Ermittelter Wiederausschaltwert, wobei die gemessene Helligkeit größer sein muss. Während der Wartephase (*uiLightOnTime*) springt dieser Wert auf 65535, um ein Ausschalten des Lichts in dieser Zeit zu vermeiden. Bei ausgeschalteter Steuerung oder im Handmodus seht dieser Ausgang auf "0".

bError : Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId : Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

VAR_IN_OUT

Damit die eingetragenen Parameter über einen Steuerungsausfall hinweg erhalten bleiben ist es erforderlich, sie als In-Out-Variablen zu deklarieren. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variablen wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

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

uiSwitchOnValue : Lichteinschaltwert. Bei einer Außenhelligkeit unter diesem Wert wird - nach Ablauf der Einschaltverzögerung - die Beleuchtung eingeschaltet.

uiSwitchOffOffset : Ist das Licht für die Zeit *uiLightOnTime* eingeschaltet (siehe oben) so berechnet sich der Lichtausschaltwert aus der aktuell gemessenen Lichtstärke plus diesem Wert.

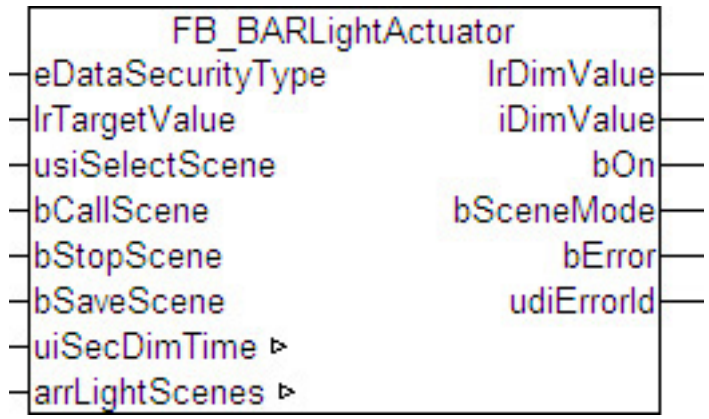
uiLightOnTime : Die Beleuchtung erreicht nicht sofort ihren wahren Einschaltwert. Um den Zugewinn an Lichtintensität zu beurteilen wird nach dem Einschalten um diese Zeit in Millisekunden gewartet, bevor der Zugewinn und der davon abhängige Lichtausschaltwert berechnet wird.

uiOnDelay : Einschaltverzögerung in Sekunden. Ständige Ein- und Ausschaltvorgänge der Beleuchtung werden als sehr störend empfunden. Sinkt die Außenhelligkeit derart, dass die Konstantlichtregelung zur Unterstützung eingeschaltet werden soll, so soll dies erst nach dieser Verzögerungszeit geschehen.

uiOffDelay : Ausschaltverzögerung in Sekunden. Steigt die Außenhelligkeit derart, dass die Konstantlichtregelung ausgeschaltet werden soll, so soll dies erst nach dieser Verzögerungszeit geschehen um kurzzeitige Schwankungen auszublenden.

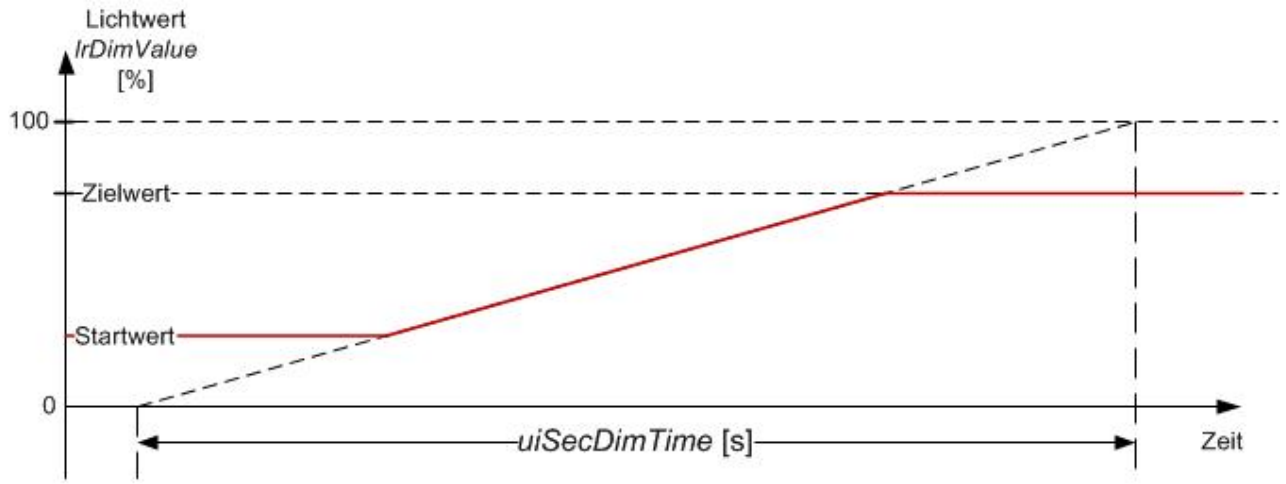
3.3.3.5 FB_BARLightActuator

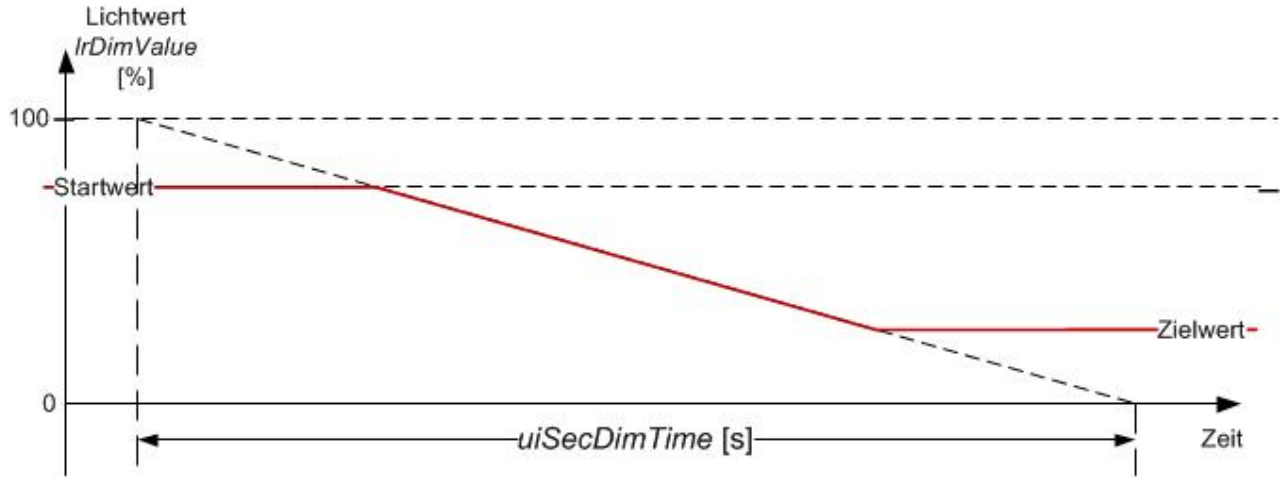
Dieser Baustein dient zur Ansteuerung eines konventionellen Lichtaktors. Die Ausgänge decken die Wertebereiche 0..100%, 0..32767 und Ein/Aus ab.
Darüber hinaus enthält der Baustein einen Szenenspeicher, in denen bis zu 21 verschiedene Lichtwerte hinterlegt werden können.



Der Funktionsbaustein gibt grundsätzlich die Werte, welche am Eingang *lrTargetValue* anliegen, an den Ausgang *lrDimValue* durch. Eine positive Flanke am Eingang *bCallScene* setzt den Ausgang hingegen auf den Lichtwert, welcher in der Szenentabelle *arrLightScenes* unter dem Index *usiSelectedScene* hinterlegt ist. Der Ausgang *bSceneMode* wechselt dann auf TRUE. Wechselt der Szenenindex *usiSelectedScene*, so wird der neu gewählte Szenenwert erst durch eine erneute positive Flanke an *bCallScene* übernommen. Durch eine Wertänderung am Eingang *lrTargetValue* oder einer positiven Flanke an *bStopScene* wird der Szenenmodus wieder verlassen. Der Ausgang *bSceneMode* geht wieder auf FALSE und der Ausgang *lrDimValue* folgt wieder dem Eingang *lrTargetValue*. Ein TRUE-Signal an bSaveScene speichert den aktuellen Lichtausgabewert in die Szenentabelle *arrLightScenes* unter dem Index *usiSelectedScene.*

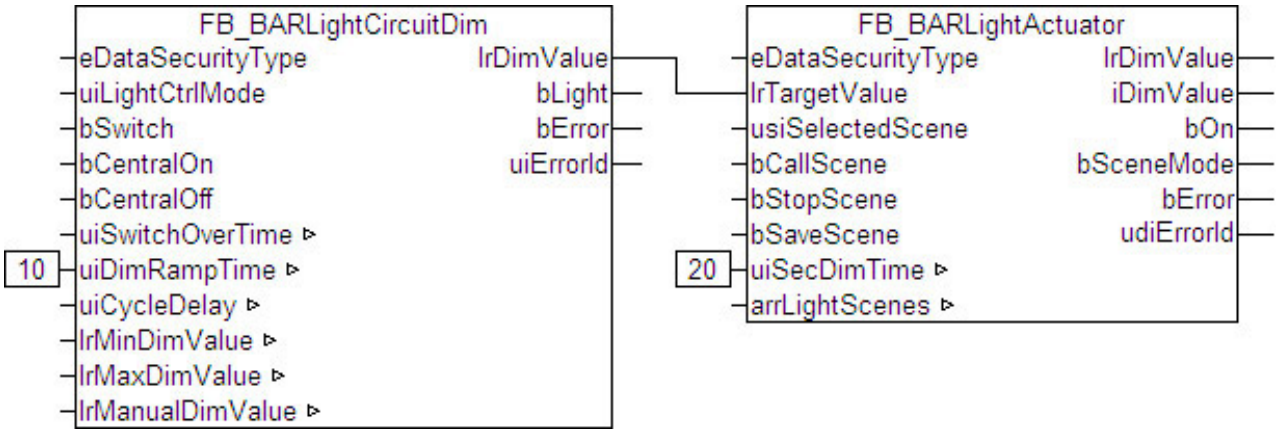
Der Lichtausgabewert *lrDimValue* folgt den ihm vorgegebenen Zielwerten immer über eine Rampe. Diese ist wie bei der Anwenderfunktion FB_BARLightCircuitDim [▶ 138] definiert durch eine Rampenzeit - hier *uiSecDimTime* - welche die Zeitspanne in Sekunden angibt, die der Lichtausgabewert benötigen soll, um sich um 100% zu ändern:





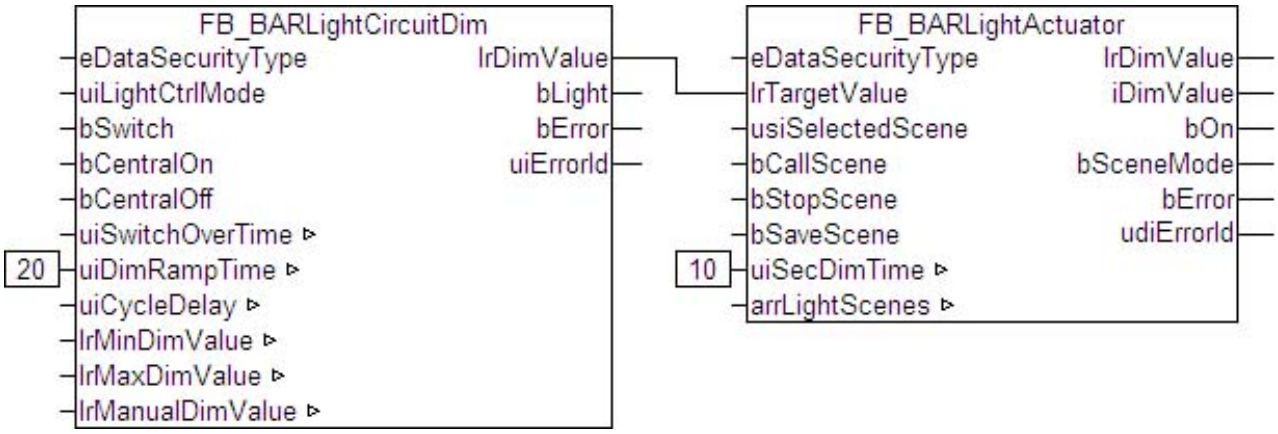
Da auch die Lichtsensor-Funktion FB_BARLightCircuitDim [▶ 138] eine Rampenfunktion zur Zielwertvorgabe beinhaltet, wird die der hier beschriebenen Lichtaktorfunktion "secondary dimtime" - *uiSecDimTime* - genannt.
Haben beide unterschiedliche Werte, so ist immer diejenige Rampenzeit die maßgebliche, welche größer ist:

Beispiel1:



In dieser Konstellation wird der Baustein FB_BARLightCircuitDim [▶ 138] seinen Ausgabewert *lrDimValue* in 10s von 0% auf 100% ändern. Der Baustein FB_BARLightActuator kann diesen sich kontinuierlich ändernden Sollwerten jedoch nur mit einer Rampenzeit von 20s (bezogen auf eine Änderung von 0% bis 100%) folgen. Daher diese Rampenzeit letztendlich die resultierende.

Beispiel2:



Anders herum könnte der Baustein FB_BARLightActuator hier mit einer Rampenzeit von 10s viel schneller folgen. Da der Dimmbaustein FB_BARLightCircuitDim [▶ 138] seine Sollwerte in diesem Beispiel jedoch nur mit einer Rampenzeit von 20s vorgibt, ist auch hier diese längere Zeit maßgeblich.
Zu beachten ist in diesem Beispiel, dass ein Kurztastendruck am Baustein FB_BARLightCircuitDim den Ausgang *lrDimValue* unverzögert auf den intern gespeicherten Wert ändert (siehe FB_BARLightCircuitDim [▶ 138]). Damit ist dann die Rampenzeit am FB_BARLightActuator maßgebend.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
lrTargetValue    : LREAL;
usiSelectedScene : USINT;
bCallScene       : BOOL;
bStopScene       : BOOL;
bSaveScene       : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Andernfalls wird der instanziierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

lrTargetValue : Zielwert der Lichtausgabe in 0..100%;

usiSelectedScene : Gewählte Lichtszene, 0..20;

bCallScene : Setzt den Ausgabewert rampengesteuert auf den unter dem Index usiSelectedScene (*arrLightScenes*) eingetragenen Lichtwert.

bStopScene : Setzt den Ausgabewert wieder auf den Wert *lrTargetValue* zurück. Die Änderung erfolgt ebenfalls rampengeführt..

bSaveScene : Speichert den aktuellen Lichtwert am Ausgang *lrDimvalue* in der Lichtwerttabelle*arrLightScenes* unter dem Index *usiSelectedScene* .

VAR_OUTPUT

```
lrDimValue: LREAL;
iDimValue : INT;
bOn       : BOOL;
bSceneMode: BOOL;
bError    : BOOL;
udiErrorId: UDINT;
```

lrDimValue : Ausgabe-Lichtwert in 0..100%.

iDimValue : Ausgabe-Lichtwert in 0..32767.

bOn : Ausgabe-Lichtstatus: *lrDimmValue*=0.0 => *bOn*=FALSE - *lrDimmValue*>0.0 => *bOn*=TRUE.

bSceneMode : Der Baustein gibt derzeit einen Szenenwert aus und nicht den Wert *lrTargetValue* am Eingang aus.

bError : Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId : Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

VAR_IN_OUT

Damit die eingetragenen Parameter über einen Steuerungsausfall hinweg erhalten bleiben ist es erforderlich, sie als In-Out-Variablen zu deklarieren. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variablen wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

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

uiSecDimTime : Rampenzeit in Sekunden. Dies ist die Zeit, die die Lichtaktor-Funktion benötigt, um von 0..100% zu regeln.

arrLightScenes : Tabelle von gespeicherten Lichtwerten.

3.3.3.6 FB_BARLightCircuit

Dieser Baustein stellt einen einfachen Lichtschaltkreis ohne Dimmfunktionalität dar.

Der Baustein kennt drei verschiedene Modi, welche über den Eingang *uiLightCtrlMode* eingestellt werden können:

- Automatikbetrieb
- Hand-Ein-Betrieb

- Hand-Aus-Betrieb

Im Automatikbetrieb (*uiLightCtrlMode*=0) lässt sich der Baustein über die Eingänge *bSwitch*, *bCentralOff* und *bCentralOn* steuern. Eine steigende Flanke an *bCentralOff* schaltet den Ausgang *lrDimValue* auf 0.0 und durch eine steigende Flanke an *bCentralOn* wird der Ausgang auf 100.0 gesetzt. Steigende Flanken an *bSwitch* lassen den Ausgang *lrDimValue* jeweils zwischen 0.0 und 100.0 wechseln.

In den Hand-Betriebsmodi wird bei *uiLightCtrlMode*=1 der Ausgabewert *lrDimValueOut* konstant auf den Wert *100.0* und bei *uiLightCtrlMode*=2 konstant auf den Wert 0.0 gesetzt.

VAR_INPUT

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

uiLightCtrlMode : Betriebsmodus.

- 0: Automatikbetrieb, der Ausgabewert *lrDimmValue* kann durch die die Befehlseingänge *bSwitch*, *bCentralOn* und *bCentralOff* beeinflusst werden
- 1: Hand-Ein-Betrieb, der Wert 100.0 wird unverzögert an den Ausgang *lrDimValueOut* ausgegeben, alle anderen Befehlseingänge sind unwirksam.
- 2: Hand-Aus-Betrieb, der Wert 0.0 wird unverzögert an den Ausgang *lrDimValueOut* ausgegeben, alle anderen Befehlseingänge sind unwirksam.

bSwitch: Steigende Flanken an *bSwitch* lassen den Ausgang *lrDimValue* jeweils zwischen 0.0 und 100.0 wechseln.

bCentralOn: Schaltet den Ausgang *lrDimValueOut* auf 100.0.

bCentralOff: Schaltet den Ausgang *lrDimValueOut* auf 0.0.

Alle Schaltbefehle, bSwitch, bCentralOn und bCentralOff sind nur im Automatik-Modus wirksam.

VAR_OUTPUT

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

lrDimValue : Lichtausgabewert in Prozent 0.0, wenn das Licht ausgeschaltet und 100.0, wenn das Licht eingeschaltet ist.

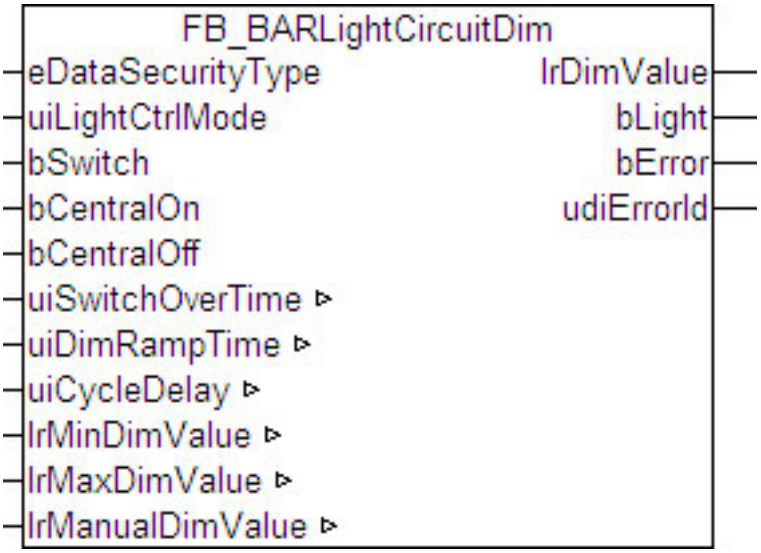
bLight : Ist FALSE, wenn *lrDimmValue* = 0.0, andernfallsTRUE.

bError : Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId : Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

3.3.3.7 FB_BARLightCircuitDim

Dieser Baustein stellt einen Lichtschaltkreis mit Dimmfunktionalität dar.

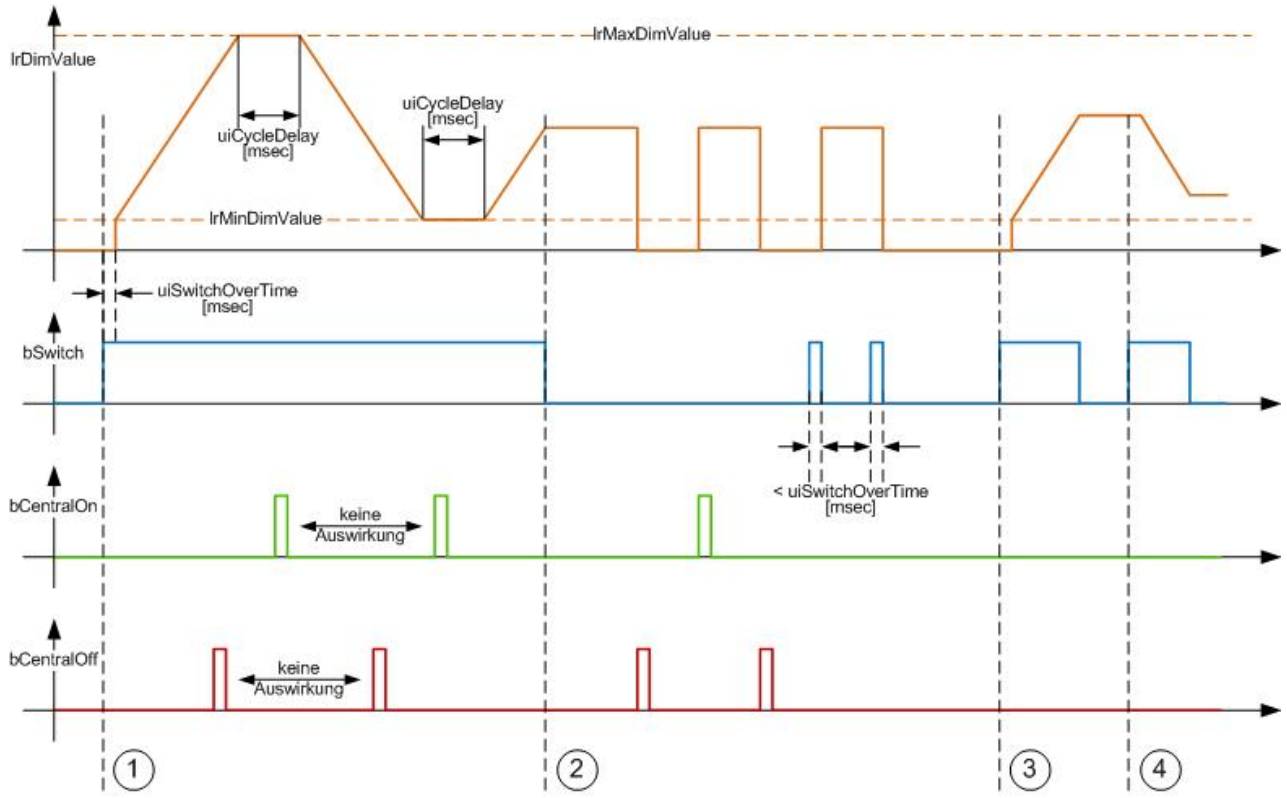


Der Baustein kennt drei verschiedene Modi, welche über den Eingang *uiLightCtrlMode* eingestellt werden können:

- Automatikbetrieb
- Hand-Ein-Betrieb
- Hand-Aus-Betrieb

Im Automatikbetrieb (*uiLightCtrlMode*=0) lässt sich der Baustein über die Eingänge *bSwitch*, *bCentralOff* und *bCentralOn* steuern. Während durch steigende Flanken an *bCentralOff* und *bCentralOn* der Ausgang *lrDimValue* auf 0.0 bzw. auf den vor dem letzten Ausschalten erreichten Wert gestellt wird, hängt das Verhalten des Einganges *bSwitch* von der Signaldauer ab. Ein kurzes TRUE-Signal unterhalb *uiSwitchOverTime* in Millisekunden schaltet Ausgang *lrDimValue.* Ist der Ausgang lrDimValue zunächst größer als 0.0, so wird er auf 0 geschaltet und der vorherige Wert abgespeichert. Ist er hingegen 0.0, so wird der Ausgang auf den zuvor intern gespeicherten Wert gestellt. Dieser gespeicherte Wert ist im Einschaltzustand des Programms auf den Maximalwert, siehe unten, gesetzt. Anders als bei den Eingängen *bCentralOff* und *bCentralOn* löst hier die fallende Flanke das Schaltereignis aus. Ein langes TRUE-Signal an *bSwitch* startet den Dimmvorgang. Dabei wird grundsätzlich nur zwischen den eingestellten Minimal- und Maximalwerten (*lrMinDimValue* und *lrMaxDimValue*) gedimmt. Die Dimmrampe ist dabei durch *uiRampTime* in Sekunden *definiert*. Diese Dimmzeit bezieht sich auf den Dimmbereich von 0 bis 100 Prozent, auch wenn die Grenzen *lrMinDimValue* und *lrMaxDimValue* anders gewählt sind. Ist der Ausgabewert *lrDimmValue* an eine der Grenzen angelangt, ohne dass *bSwitch* wieder auf FALSE zurückfällt, so verharrt er dort für die Zeit *uiCycleDelay* in Millisekunden, bevor wieder zum anderen Grenzwert hin gedimmt wird.

Ausnahme: Ein Wert *uiCycleDelay* von 0 wird nicht als sofortige Dimmrichtungsumkehr interpretiert, sondern deaktiviert diese. Erst ein erneutes langes TRUE-Signal an *bSwitch* startet das Dimmen in entgegen gesetzter Richtung.



(1) **Dimmen:**
Nach einem Lang-Signal größer *uiSwitchOverTime* in Millisekunden an *bSwitch* wird der Ausgang *lrDimmValue* zunächst auf den Wert *lrMinValue* geschaltet. Da *bSwitch* weiterhin auf TRUE bleibt wird der Dimm-Vorgang fortgesetzt und *lrDimmValue* steigt stetig an. Am Maximalwert *lrMaxValue* angelangt verharrt der Wert für die Zeit *uiCycleDelay* in Millisekunden und fällt dann wieder stetig bis auf den Minimalwert *lrMinValue*. Der Minimalwert bleibt am Ausgang *lrDimmValue* wiederum für die Zeit *uiCycleDelay* in Millisekunden erhalten, bis dieser wieder ansteigt. Eine fallende Flanke an *bSwitch* beendet den Dimmvorgang. Während des gesamten Dimmvorganges, d.h. solange *bSwitch* auf TRUE steht, sind die Eingänge *bCentralOn* und *bCentralOff* ohne Funktion.

(2) **Schalten:**
Steigende Flanken an bCentralOff und bCentralOn schalten den Ausgang lrDimValue auf 0 bzw auf den zuvor eingestellten Wert. Mit dem Eingang bSwitch lässt sich der Ausgang ebenfalls abwechselnd auf 0 und den zuvor gestellten Wert schalten, jedoch lösen hier die fallenden Flanken des Kurzsignals (kleiner *uiSwitchOverTime* in Millisekunden) das Schaltereignis aus.

(3) Ein erneutes Lang-Signal an bSwitch (größer *uiSwitchOverTime* in Millisekunden) startet den Dimm-Vorgang erneut: Da das Licht vorher aus war (*lrDimmValue*=0.0) wird auf den Minimalwert-gesprungen und dann hochgedimmt. Der Dimmvorgang endet mit der fallenden Flanke an *bDimm*.

(4) Wird darauf hin noch einmal gedimmt, so startet der Vorgang ab dem letzten Wert, jedoch in entgegen gesetzter Richtung.

In den Hand-Betriebsmodi wird bei *uiLightCtrlMode*=1 der Ausgabewert *lrDimValueOut* konstant auf den Wert *lrManualDimValue* und bei *uiLightCtrlMode*=2 konstant auf den Wert 0.0 gesetzt.

VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Andernfalls wird der instanziierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

uiLightCtrlMode : Betriebsmodus.

- 0: Automatikbetrieb, der Ausgabewert *lrDimmValue* kann durch die die Befehlseingänge *bSwitch*, *bCentralOn* und *bCentralOff* beeinflusst werden
- 1: Hand-Ein-Betrieb, der Wert *lrManualDimValue* wird unverzögert an den Ausgang *lrDimValueOut* ausgegeben, alle anderen Befehlseingänge sind unwirksam.
- 2: Hand-Aus-Betrieb, der Wert 0.0 wird unverzögert an den Ausgang *lrDimValueOut* ausgegeben, alle anderen Befehlseingänge sind unwirksam.

bSwitch: Ein kurzes TRUE-Signal unterhalb *uiSwitchOverTime* in Millisekunden schaltet Ausgang *lrDimValue.* Ist der Ausgang lrDimValue zunächst größer als 0.0, so wird er auf 0 geschaltet und der vorherige Wert abgespeichert. Ist er hingegen 0.0, so wird der Ausgang auf den zuvor intern gespeicherten Wert gestellt. Dieser gespeicherte Wert ist im Einschaltzustand des Programms auf den Maximalwert, siehe unten, gesetzt.
Ein langes TRUE-Signal größer *uiSwitchOverTime* in Millisekunden startet den Dimmvorgang, wobei sich die Dimmrichtung verändert, wenn zwei Lang-Signale am Eingang bSwitch aufeinander folgen.

bCentralOn: Schaltet den Ausgang *lrDimValueOut* auf den zuvor (beim letzten Ausschalten) gespeicherten Dimmwert, siehe *bSwitch*.

bCentralOff: Schaltet den Ausgang *lrDimValueOut* auf 0.0.

Alle Schaltbefehle, bSwitch, bCentralOn und bCentralOff sind nur im Automatik-Modus wirksam.

VAR_OUTPUT

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

lrDimValue : Lichtausgabewert, kann Werte zwischen *lrMinDimValue* und *lrMaxDimValue* annehmen, siehe VAR_IN_OUT, maximal jedoch 0..100%.

bLight : Ist FALSE, wenn *lrDimmValue* = 0.0, andernfalls TRUE.

bError : Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId : Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

VAR_IN_OUT

Damit die eingetragenen Parameter über einen Steuerungsausfall hinweg erhalten bleiben ist es erforderlich, sie als In-Out-Variablen zu deklarieren. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variablen wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

```
uiSwitchOverTime : UINT;
uiDimRampTime    : UINT;
uiCycleDelay     : UINT;
lrMinDimValue    : LREAL;
lrMaxDimValue    : LREAL;
lrManualDimValue : LREAL;
```

uiSwitchOverTime : Umschaltzeit in Millisekunden vom Tast- in den Dimmmodus.

uiDimRampTime : Rampenzeit in Sekunden (benötigte Zeit, um von 0% auf 100% zu dimmen).

uiCycleDelay : Umschaltzeit in Millisekunden vom automatischem Wechsel zwischen Auf- und Abdimmen. Ist dieser Wert gleich Null, so ist der automatische Wechsel inaktiv.

lrDimMinValue : Minimaler Lichtwert in %, der durch Dimmen erreicht werden kann.

lrDimMaxValue : Maximaler Lichtwert in %.

lrManualDimValue : Ausgabe-Dimmwert im Hand-Ein-Modus (uiLightCtrlMode=1).

.

3.3.3.8 FB_BARStairwellAutomatic

Funktionsbaustein für eine Treppenhauslichtschaltung.

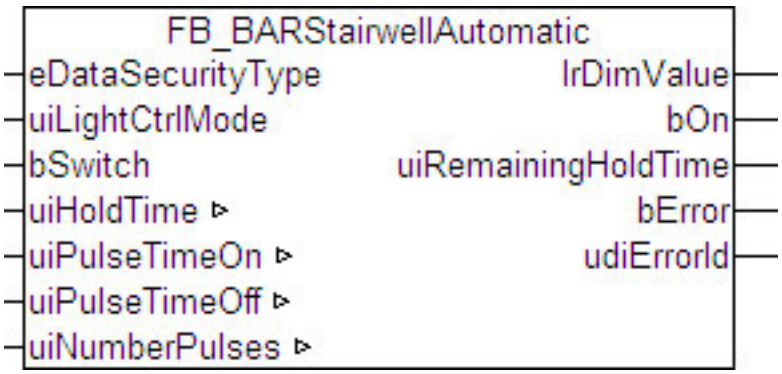
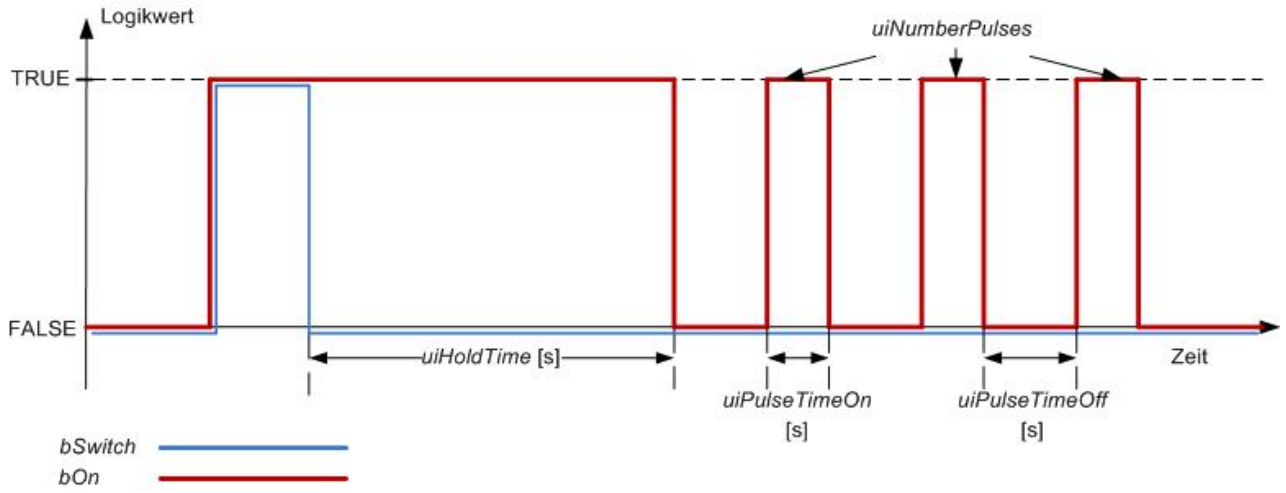


Abb. 5: FB_BARStairwellAutomatic

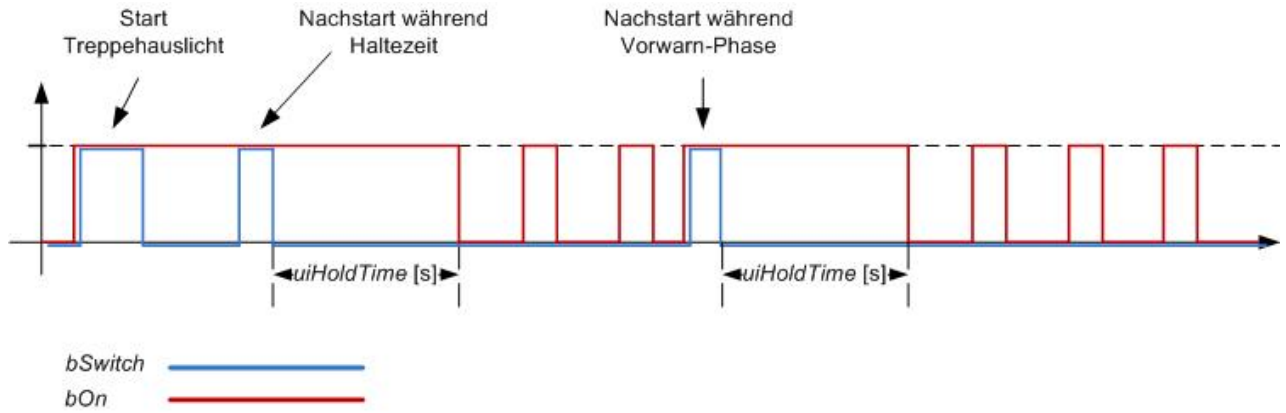
Der Baustein kennt 3 verschiedene Modi, welche über den Eingang *uiLightCtrlMode* eingestellt werden können:

- Automatikbetrieb
- Hand-Ein-Betrieb
- Hand-Aus-Betrieb

Im Automatikbetrieb (*uiLightCtrlMode*=0) ist die Treppenhaussteuerung aktiv. Eine positive Flanke an *bSwitch* schaltet das Licht (Ausgang *bOn*) zunächst nur ein. Mit einer negativen Flanke wird der Haltezeitgeber gestartet. Ist die Haltezeit *uiHoldTime* [s] abgelaufen, beginnt als Hinweis für das bevorstehende Ausschalten eine Blinksequenz mit einer durch *uiNumberPulses* programmierbaren Anzahl von Blinkimpulsen. Diese Impulse haben eine Ein-Zeit von *uiPulseTimeOn* [ms] und eine Aus-Zeit von *uiPulseTimeOff* [ms].



Die Sequenz kann dabei jederzeit nachgestartet werden:



In den Hand-Betriebsmodi ist der Eingang *bSwitch* ohne Funktion: bei *uiLightCtrlMode*=1 wird der Ausgabewert *bOn* konstant auf TRUE und bei *uiLightCtrlMode*=2 konstant auf FALSE gesetzt.
Ein Wechsel in den Handmodus setzt eine bis dahin gestartete Beleuchtungssequenz zurück.

VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Andernfalls wird der instanziierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

uiLightCtrlMode : Betriebsmodus.

- 0: Automatikbetrieb, die Treppenhausschaltung ist aktiv und reagiert auf den Eingang *bSwitch*.
- 1: Hand-Ein-Betrieb, die Treppenhausschaltung ist nicht aktiv - der Ausgang *bOn* wird konstant auf TRUE gesetzt.
- 2: Hand-Aus-Betrieb, die Treppenhausschaltung ist nicht aktiv - der Ausgang *bOn* wird konstant auf FALSE gesetzt.

bSwitch: Eine steigende Flanke schaltet im Automatikbetrieb (*uiLightCtrlMode*=0) das Licht ein, eine fallende Flanke startet den Haltezeitgeber. Im Handbedienmodus (*uiLightCtrlMode*=1 oder 2) hat dieser Eingang keine Funktion.

VAR_OUTPUT

```
lrDimValue          : LREAL;
bOn                 : BOOL;
uiRemainingHoldTime: UINT;
bError              : BOOL;
udiErrorId          : UDINT;
```

lrDimValue : Um diese Funktion zu den anderen Licht-Anwenderfunktionen gleich zu halten, existiert auch hier ein Lichtausgabewert als Fließkommazahl in Prozent, obwohl das Licht nur ein- oder ausgeschaltet wird. Das bedeutet: Licht aus: *lrDimValue* = 0.0, Licht an: *lrDimValue* = 100.0.

bOn : Schaltausgang für die Beleuchtung.

uiRemainingTimeHold: Verbleibende Haltezeit in in Sekunden. Ist das Licht aus oder der Hand-Bedienmodus aktiv, so steht dieser Ausgang auf "0". Mit einer steigenden Flanke an *bSwitch* im Automatikmodus zeigt dieser Ausgang zunächst die komplette Anzahl der Sekunden der Haltezeit (*uiHoldTime*) an um, beginnend mit einer fallenden Flanke an *bSwitch,* das Ablaufen der Haltezeit darzustellen. Solange kein Herunterzählen der Zeit stattfindet, steht dieser Ausgang auf 0.

bError : Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId : Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

VAR_IN_OUT

Damit die eingetragenen Parameter über einen Steuerungsausfall hinweg erhalten bleiben ist es erforderlich, sie als In-Out-Variablen zu deklarieren. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variablen wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel:  https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

```
uiHoldTime      : UINT;
uiPulseTimeOn   : UINT;
uiPulseTimeOff  : UINT;
uiNumberPulses  : UINT;
```

uiHoldTime: Haltezeit [s] der Treppenhaussteuerung nach fallender Flanke an *bSwitch.*

uiPulseTimeOn: EIN-Zeit der Vorwarnimpulse in Millisekunden.

uiPulseTimeOff: AUS-Zeit der Vorwarnimpulse in Millisekunden.

uiNumberPulses: Anzahl der Vorwarnimpulse.

3.3.3.9 FB_BARTwilightAutomatic

Dämmerungsautomatik

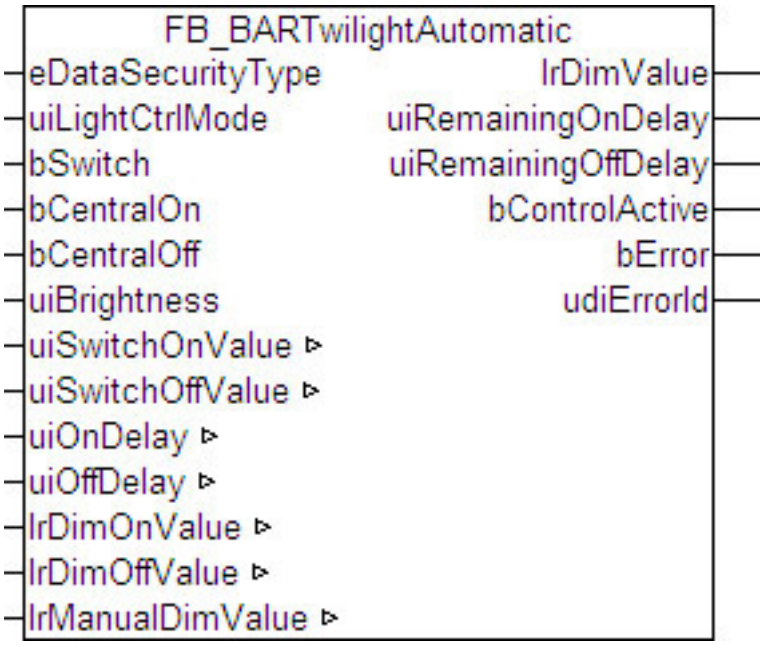


Abb. 6: FB_BARTwilightAutomatic

Der Baustein kennt drei verschiedene Modi, welche über den Eingang *uiLightCtrlMode* eingestellt werden können:

- Automatikbetrieb
- Hand-Ein-Betrieb
- Hand-Aus-Betrieb

Automatikbetrieb

Im Automatikbetrieb schaltet eine positive Flanke an *bSwitch* die Dämmerungsautomatik aktiv, sofern sie vorher inaktiv war. Eine erneute Flanke an *bSwitch* deaktiviert die Dämmerungsautomatik wieder. Mit *bCentralOn* und *bCentralOff* wird die Automatik explizit ein- und ausgeschaltet, unabhängig vom vorherigen Zustand der Automatik. Ist die Dämmerungsautomatik aktiviert, so schaltet der Baustein bei Unterschreiten eines Einschaltschwellwertes *uiSwitchOnValue* für eine eingetragene Verzögerungszeit *uiOnDelay* den Ausgang *lrDimValue* auf den Einschaltwert *lrDimValueOn*. Anders herum wird bei Überschreiten eines Ausschaltschwellwertes *uiSwitchOffValue* für eine eingetragene Verzögerungszeit *uiOffDelay* der Ausgang auf den Ausschaltwert *lrDimValueOff* geschaltet. Im inaktiven Zustand wird der Lichtausgabewert *lrDimValue* auf 0.0 gesetzt.

Handbetrieb

In den Hand-Betriebsmodi sind die Eingänge bSwitch, bCentralOn und bCentralOff ohne Funktion: bei *uiLightCtrlMode*=1 wird der Ausgabewert *lrDimValue* konstant auf *lrManualDimValue* und bei *uiLightCtrlMode*=2 konstant auf 0.0 gesetzt.

Ein- und Ausschaltverzögerung

Die unter dem Automatikbetrieb beschriebenen Ein- und Ausschaltverzögerungen werden unabhängig vom Zustand der Automatik (aktiv oder inaktiv) und dem Betriebsmodus immer durchlaufen, das heißt die Zeitgeber werden durch diese Betriebszustände nicht zurück gesetzt.

VAR_INPUT

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

```
bCentralOn      : BOOL;
bCentralOff     : BOOL;
uiBrightness    : UINT;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Andernfalls wird der instanziierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

uiLightCtrlMode : Betriebsmodus.

- 0: Automatikbetrieb, die Befehle *bSwitch*, *bCentralOn* und *bCentralOff* sind ausführbar und die Dämmerungsautomatik lässt sich damit ein- und ausschalten.
- 1: Hand-Ein-Betrieb, die Dämmerungsautomatik ist nicht aktiv - der Wert *lrManualDimValue* wird unverzögert an den Ausgang *lrDimValueOut* ausgegeben, alle anderen Befehlseingänge sind unwirksam.
- 2: Hand-Aus-Betrieb, die Dämmerungsautomatik ist nicht aktiv - der Wert 0.0 wird unverzögert an den Ausgang *lrDimValueOut* ausgegeben, alle anderen Befehlseingänge sind unwirksam.

bSwitch : Positive Flanken an diesem Eingang schalten die Dämmerungsautomatik wechselseitig ein und aus. Beim Ausschalten wird der Ausgang *lrDimValue* auf 0.0 gesetzt. Dieser Befehlseingang ist nur im Automatik-Modus (*uiLightCtrlMode*=0) aktiv.

bCentralOn : Eine positive Flanke an diesem Eingang schaltet die Dämmerungsautomatik ein. Dieser Befehlseingang ist nur im Automatik-Modus (*uiLightCtrlMode*=0) aktiv.

bCentralOff : Eine positive Flanke an diesem Eingang schaltet die Dämmerungsautomatik aus und den Ausgang *lrDimValue* auf 0.0. Dieser Befehlseingang ist nur im Automatik-Modus (*uiLightCtrlMode*=0) aktiv.

uiBrightness : Licht-Ist-Wert [lux].

VAR_OUTPUT

```
lrDimValue          : LREAL;
uiRemainingOnDelay  : UINT;
uiRemainingOffDelay: UINT;
bControlActive      : BOOL;
bError              : BOOL;
udiErrorId          : UDINT;
```

lrDimValue : Lichtausgabewert, 0..100%.

uiRemainingOnDelay : Countdown der Einschaltverzögerung in Sekunden. Solange kein Herunterzählen der Zeit stattfindet, steht dieser Ausgang auf 0.

uiRemainingOffDelay : Countdown der Ausschaltverzögerung in Sekunden. Solange kein Herunterzählen der Zeit stattfindet, steht dieser Ausgang auf 0.

bControlActive : Dieser Ausgang ist TRUE, wenn sich der Baustein im Automatikbetrieb befindet und die Dämmerungsautomatik eingeschaltet ist. Dies soll als zusätzliche Rückmeldung dienen, wenn eingeschaltet wurde, die Steuerung jedoch einen Lichtwert von *lrDimValue*=0.0 ausgibt.

bError : Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId : Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

HinweisSollte ein Fehler anstehen, so wird diese Automatik deaktiviert und Position und Winkel auf 0 gesetzt. Das bedeutet, dass bei Verwendung einer Prioritätssteuerung automatisch eine andere Funktion niedrigerer Priorität (siehe Übersicht) die Steuerung der Jalousie übernimmt. Bei direkter Beschaltung hingegen wird die Jalousie auf Position/Winkel 0 fahren.

VAR_IN_OUT

Damit die eingetragenen Parameter über einen Steuerungsausfall hinweg erhalten bleiben ist es erforderlich, sie als In-Out-Variablen zu deklarieren. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variablen wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

```
uiSwitchOnValue  : UINT;
uiSwitchOffValue : UINT;
uiOnDelay        : UINT;
uiOffDelay       : UINT;
lrDimOnValue     : LREAL;
lrDimOffValue    : LREAL;
lrManualDimValue : LREAL;
```

uiSwitchOnValue: Einschalt-Schwellwert. Wird mit dem Helligkeitswert am Eingang *uiBrightness* verglichen. Dieser Wert muss größer sein als der Ausgangsschwellwert *uiSwitchOffValue*.

uiSwitchOffValue: Ausschalt-Schwellwert. Wird mit dem Helligkeitswert am Eingang *uiBrightness* verglichen.

uiOnDelay: Einschaltverzögerung in Sekunden.

uiOffDelay: Ausschaltverzögerung in Sekunden.

lrDimOnValue : Einschalt-Lichtwert in %.

lrDimOffValue : Ausschalt-Lichtwert in %.

lrManualDimValue : Ausgabe-Dimmwert im Hand-Ein-Modus (uiLightCtrlMode=1).

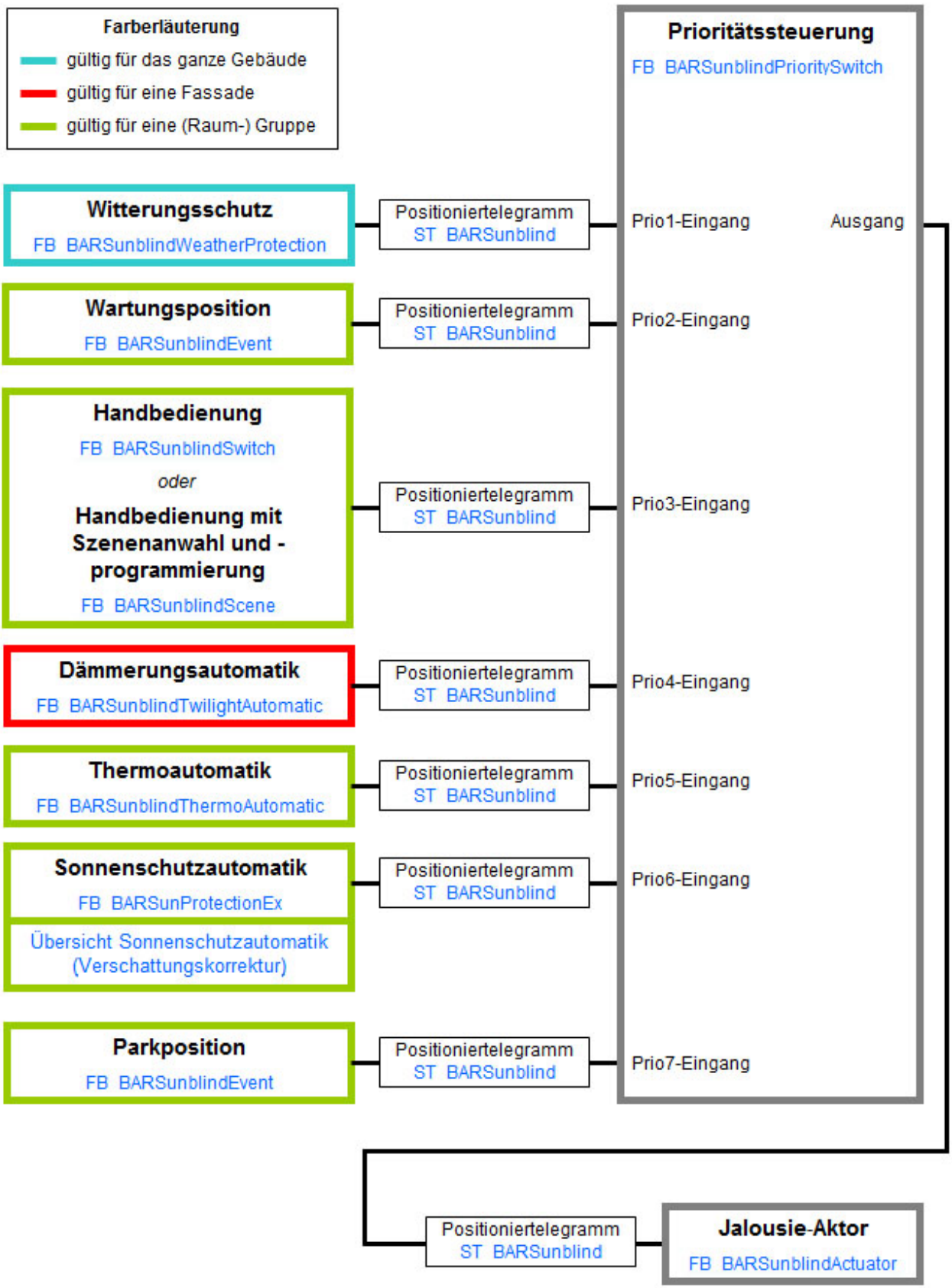
Dokumente hierzu

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

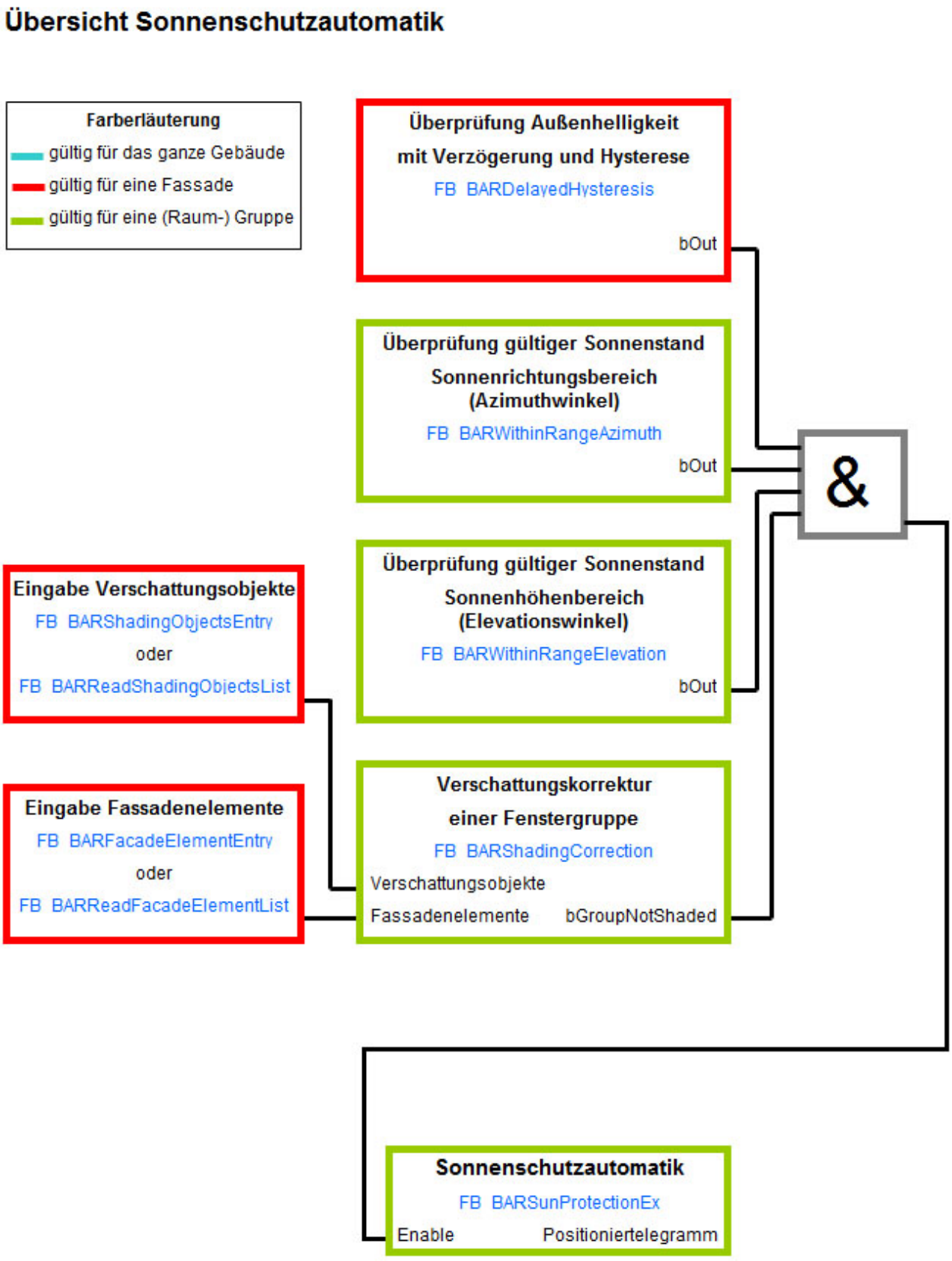
3.3.4 Sonnenschutz

3.3.4.1 Übersicht Verschattung

Verschattung - Übersicht



3.3.4.2 Übersicht Sonnenschutz



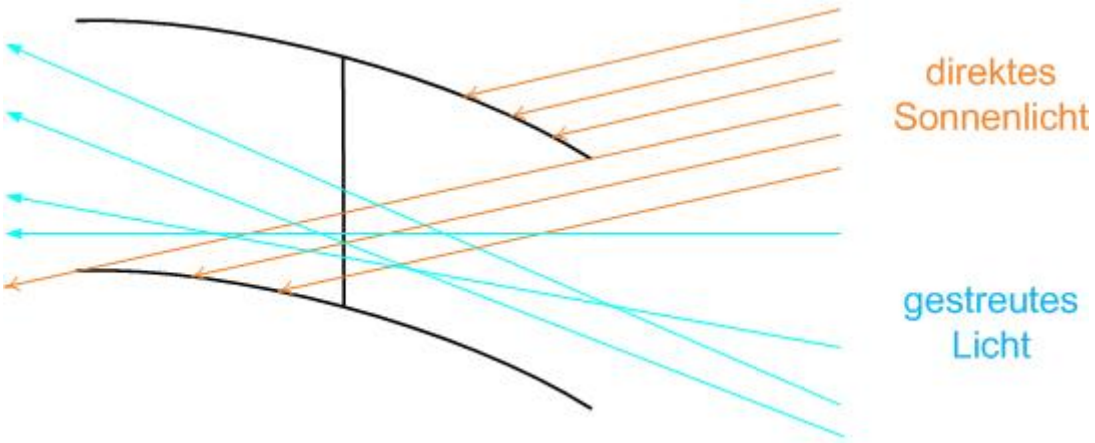
3.3.4.3 Sonnenschutz: Grundlagen und Definitionen

Direkter Einfall von Tageslicht wird von in Räumen befindlichen Personen als störend angesehen. Auf der anderen Seite jedoch empfindet der Mensch das natürliche Licht als angenehmer im Vergleich zum künstlichen Licht. Zwei Möglichkeiten des Blendschutzes sollen hier vorgestellt werden:

- Lamellennachführung
- Höhenverstellung

Lamellennachführung

Eine Jalousie mit Lamellen, welche sich nachführen lassen bietet hier die Möglichkeit eines intelligenten Sonnenschutzes. Dabei wird die Stellung der Lamellen zyklisch dem aktuellen Sonnenstand angepasst, so dass kein direktes Tageslicht durch die Jalousien fällt, jedoch möglichst viel diffuses Tageslicht genutzt werden kann.



Die Abbildung zeigt, dass gestreutes Licht von unten noch einfallen kann, während vom direkten Tageslicht gerade nichts mehr, bzw. theoretisch nur noch ein Strahl hindurch tritt. Zur Berechnung des Lamellenwinkels sind folgende Parameter notwendig:

- die aktuelle Sonnenhöhe (Elevationswinkel)
- der Sonnenstand , d.h. der Azimutwinkel
- die Fassadenausrichtung
- die Lamellenbreite
- die Lamellenabstand

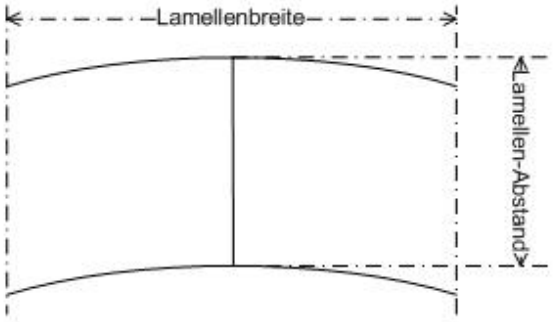
Aus den ersten drei der oben genannten Parametern errechnet sich der effektive Elevationswinkel:

Sind Fassadenausrichtung und Sonnenstand (Azimut) gleich, so ist der effektive Winkel gleich dem aktuellen Elevationswinkel.

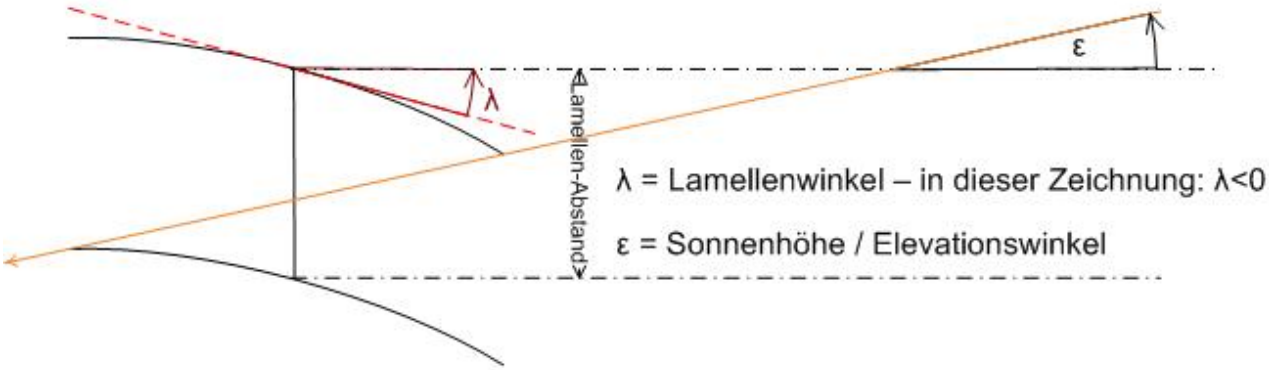
Fällt die Sonne jedoch von der Sonnenrichtung aus gesehen schräg auf die Fassade, so ist bei gleichem Elevationswinkel der effektive Winkel größer.

Die folgenden drei Bilder zeigen den Zusammenhang zwischen effektivem Elevationswinkel und den Jalousiemaßen und wie sich der resultierende Lamellenwinkel λ ändert:

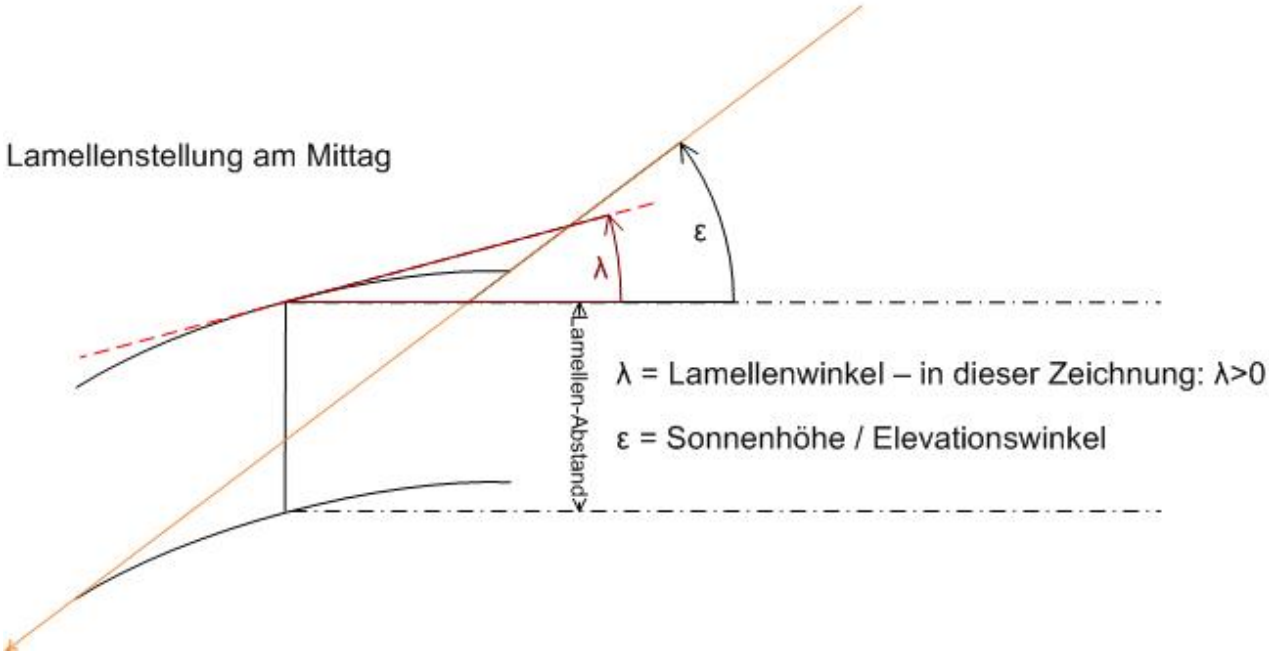
Lamellen bei einen Winkel von λ=0



Lamellenstellung am Morgen und am Abend

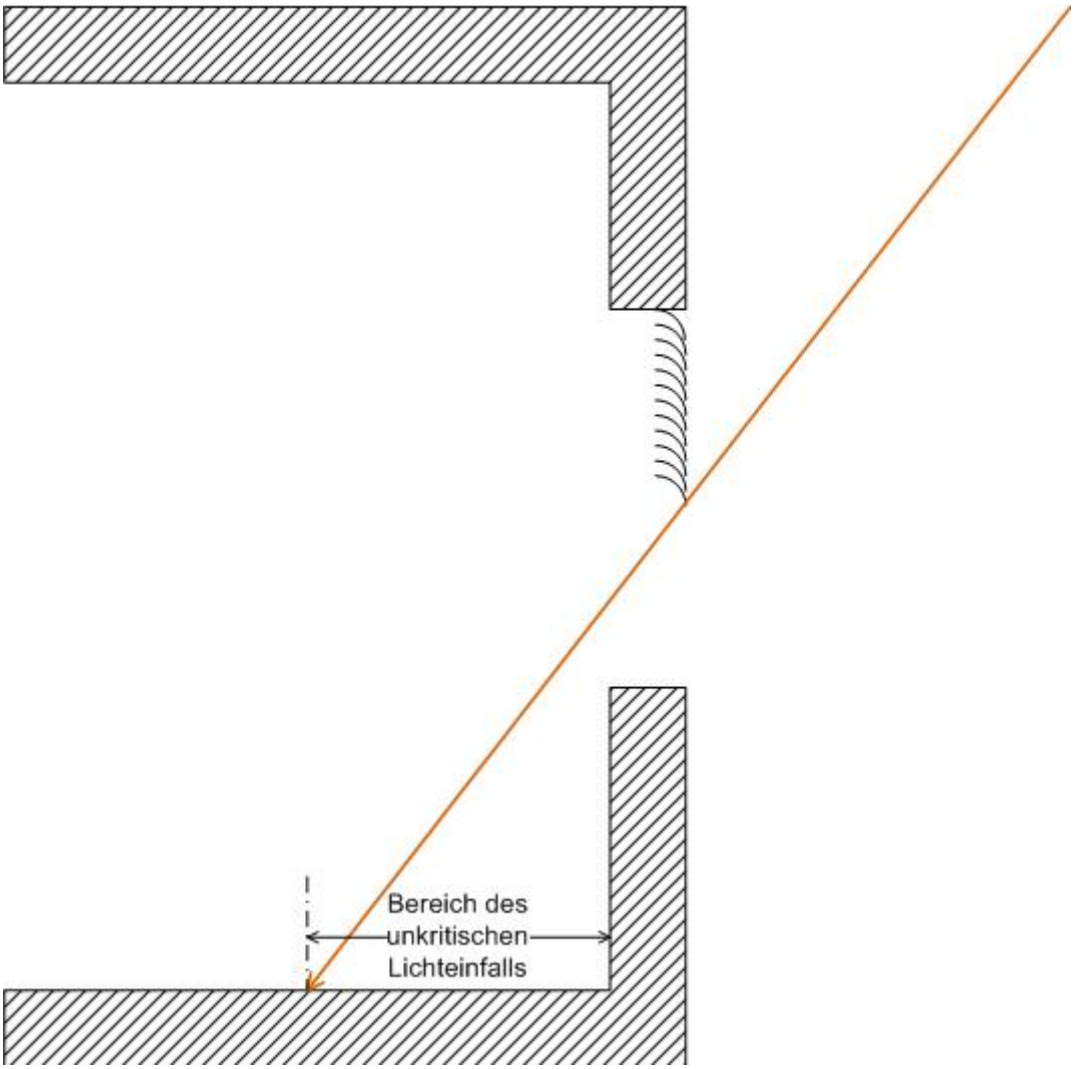


Lamellenstellung am Mittag



Höhenverstellung

Bei einem hohen Sonnenstand in der Mittagszeit dringen die direkten Sonnenstrahlen nicht in die volle Tiefe des Raumes ein. Wenn direkte Sonnenstrahlen im Bereich der Fensterbrüstung als unkritisch betrachtet werden, kann die Höhe des Sonnenschutzes automatisch so angepasst werden, dass die Sonnenstrahlen immer nur bis zu einer unkritischen Tiefe in den Raum eindringen.

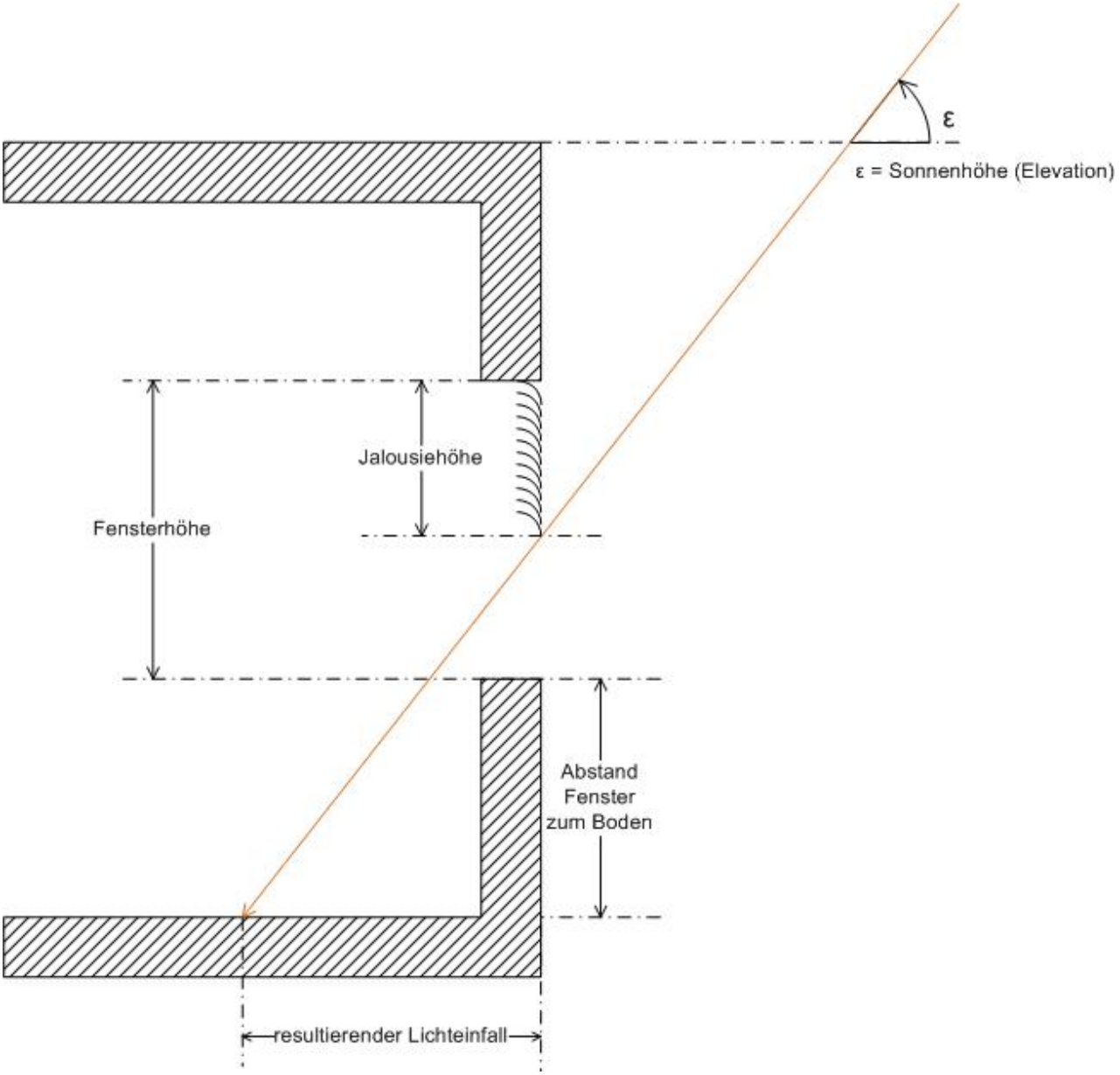


Um zu jedem Zeitpunkt die entsprechende Jalousiehöhe berechnen zu können, die sicherstellt, dass die Sonneneinstrahlung einen gewissen Wert nicht überschreitet, sind folgende Werte nötig.

Zur Berechnung der jeweiligen Jalousiehöhe erforderlich:

- Sonnenhöhe (Elevation)
- Fensterhöhe
- Abstand Fenster zum Boden

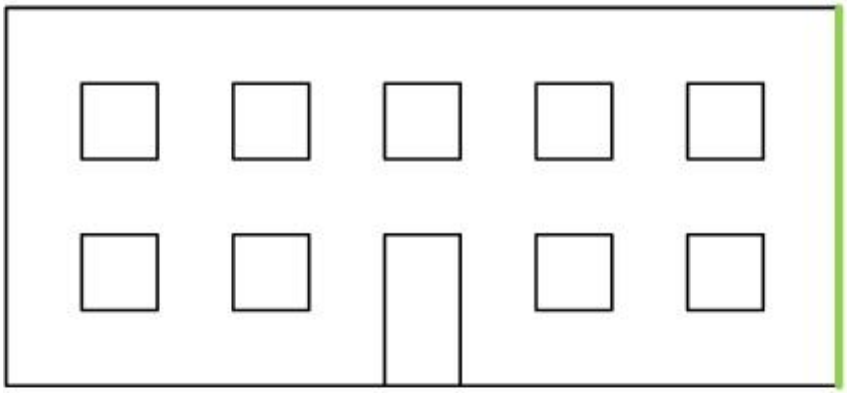
Die folgende Abbildung veranschaulicht, wo diese Parameter einzuordnen sind:



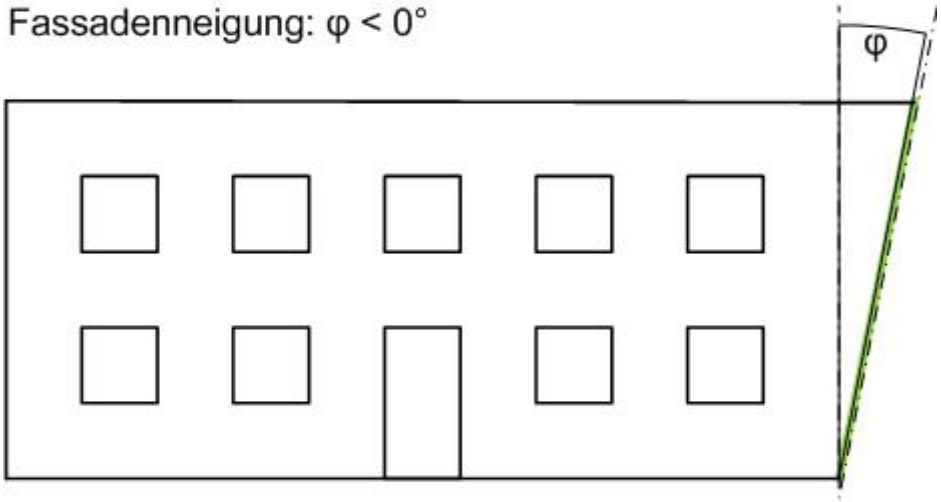
Einfluss der Fassadenneigung

Bei den beiden beschriebenen Methoden des Sonnenschutzes wurde davon ausgegangen, dass die Fassade und damit die Fenster senkrecht zum Boden stehen. Bei einer geneigten Fassade jedoch ändert sich der Lichteinfall, so dass dieser Einfluss mit berücksichtigt wird. Die Fassadenneigung wird wie folgt definiert:

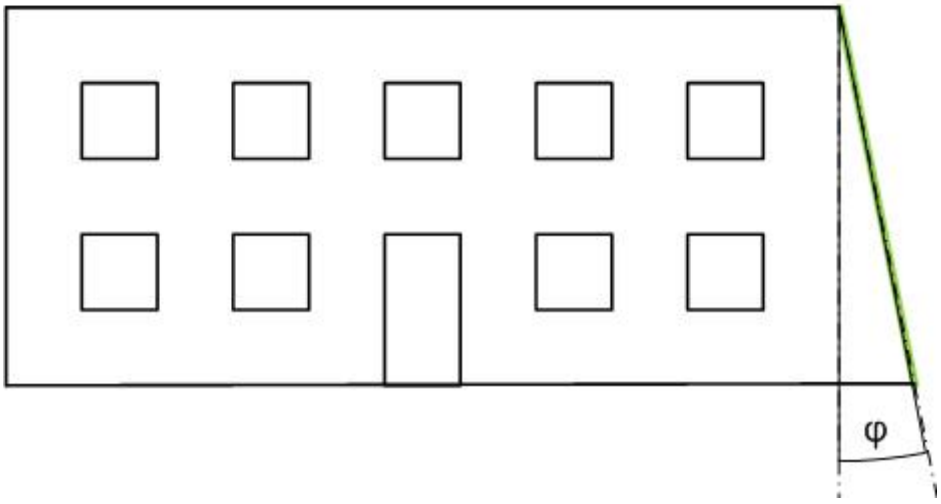
Fassadenneigung: φ = 0°



Fassadenneigung: φ < 0°



Fassadenneigung: φ > 0°



3.3.4.4 Verschattungskorrektur: Grundlagen und Definitionen

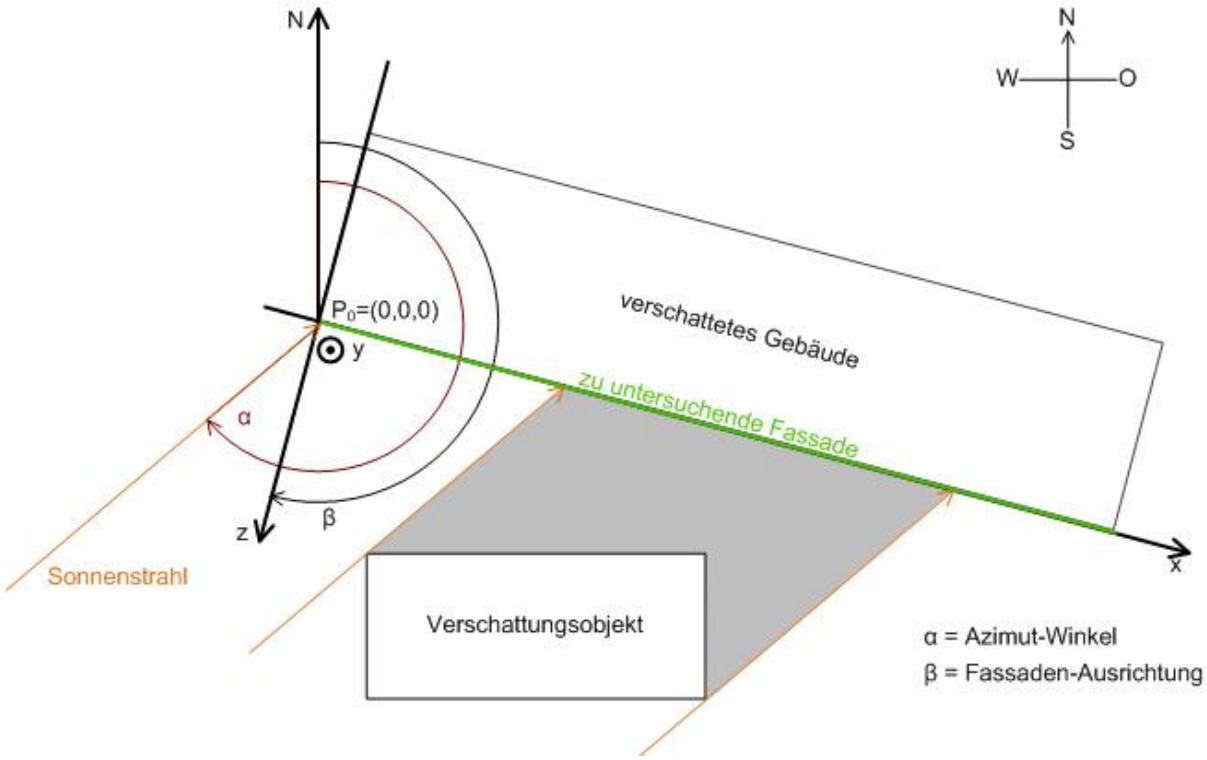
Die Verschattungskorrektur ist in Verbindung mit der Sonnenautomatik oder Lamellennachführung nutzbar. Die Funktion prüft, ob ein Fenster oder eine Gruppe von Fenstern, die z.B. einem Raum zugeordnet sind, temporär durch umliegende Bebauung oder eigene Gebäudeteile verschattet werden. Für Fenster, welche im Schatten umliegender Gebäude oder Bäume stehen ist ein Sonnenschutz nicht notwendig, unter Umständen sogar störend. Die Verschattungskorrektur ermittelt anhand von eingetragenen Daten der Fassade und ihrer Umgebung, welche Teile der Fassade verschattet werden. Damit ist es dann möglich, für einzelne Fenster oder Fenstergruppen zu entscheiden, ob der Sonnenschutz aktiv sein soll.
Neben dem aktuellen Sonnenstand hängt die Verschattung der einzelnen Fenster von drei Dingen ab:

- der Ausrichtung der Fassade
- der Lage der Fenster
- der Positionierung der Verschattungsobjekte

Die folgenden Abbildungen sollen diese Zusammenhänge erläutern und die einzutragenden Parameter vorstellen.

Ausrichtung der Fassade

Betrachtung von oben



Für die reine Betrachtung des Schattenwurfes auf die Fassade ist letztendlich ein zweidimensionales Koordinatensystem erforderlich, daher wurden die x- und y-Achse auf die Fassade gelegt. Der Nullpunkt liegt dabei im Fußpunkt links unten, so als würde man die Fassade von vorne betrachten. Zur Bemessung der verschattenden Objekte kommt dann noch die Z-Komponente hinzu. Deren Achse weist von der Fassade weg und hat denselben Nullpunkt, wie die x- und y-Achse.

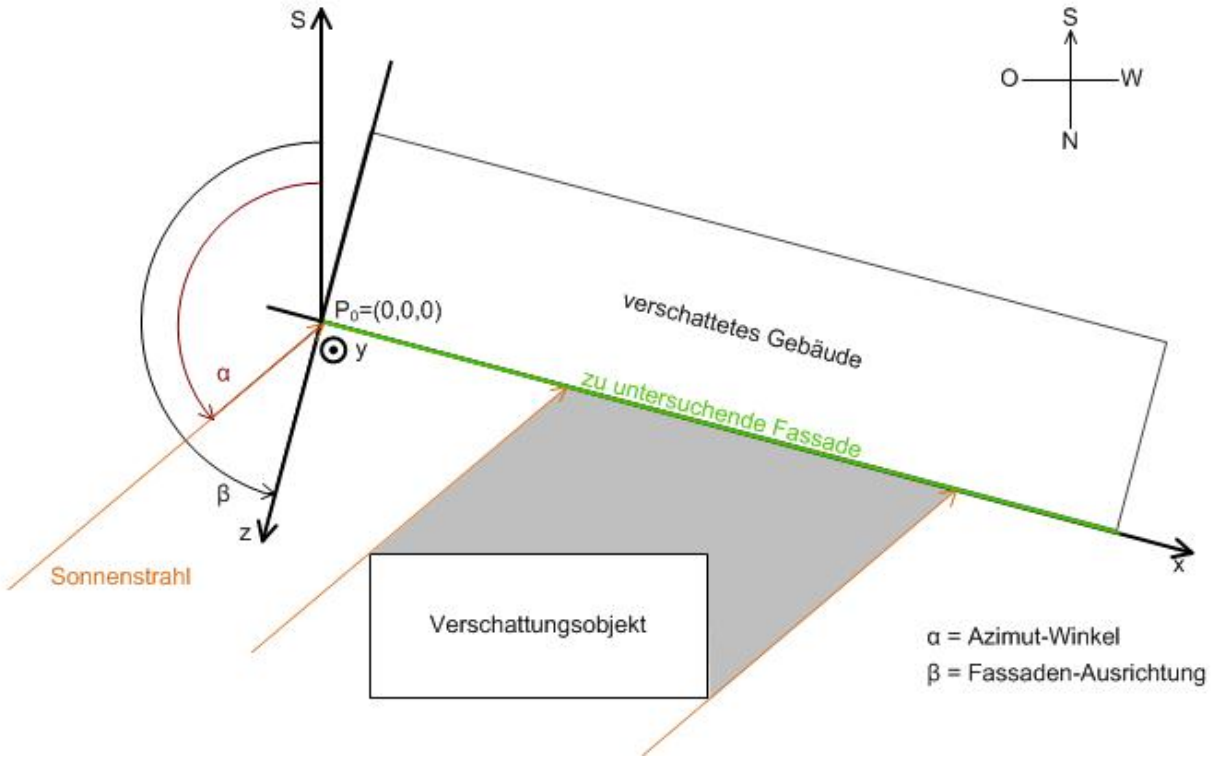
Der horizontale Sonnenstand (Azimutwinkel) ist auf der Nordhalbkugel per Definition von der Nordrichtung aus bemessen. Die Fassadenausrichtung richtet sich ebenfalls nach der Nordrichtung, wobei die Blickrichtung aus einem Fenster der Fassade gilt:

Blickrichtung	Fassadenausrichtung
Nord	β=0°
Ost	β=90°

Blickrichtung	Fassadenausrichtung
Süd	β=180°
West	β=270°

Auf der Südhalbkugel ist der Sonnenverlauf anders herum: Sie geht zwar auch im Osten auf, hat ihren Mittagsstand jedoch im Norden. Die Fassadenausrichtung wird diesem Verlauf angepasst:

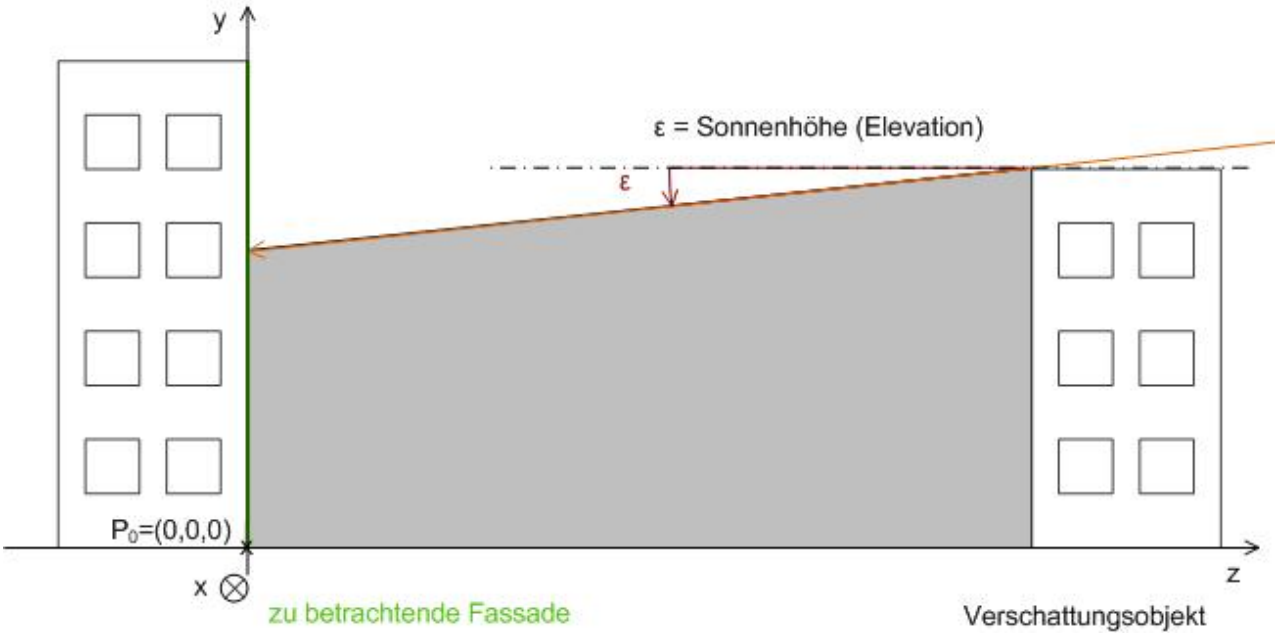
Blickrichtung	Fassadenausrichtung
Süd	β=0°
Ost	β=90°
Nord	β=180°
West	β=270°



Die weiteren Ausführungen beziehen sich jedoch der Einfachheit halber nur auf die Anwendung auf der Nordhalbkugel. Für die spätere Parametrierung (FB_BARShadingCorrection [▶ 180] / FB_BARShadingCorrectionSouth [▶ 183]) ist ohnehin nur die entsprechende Fassadenausrichtung nötig, welche der jeweils gültigen Tabelle - Nord- oder Südhalbkugel - zu entnehmen ist.

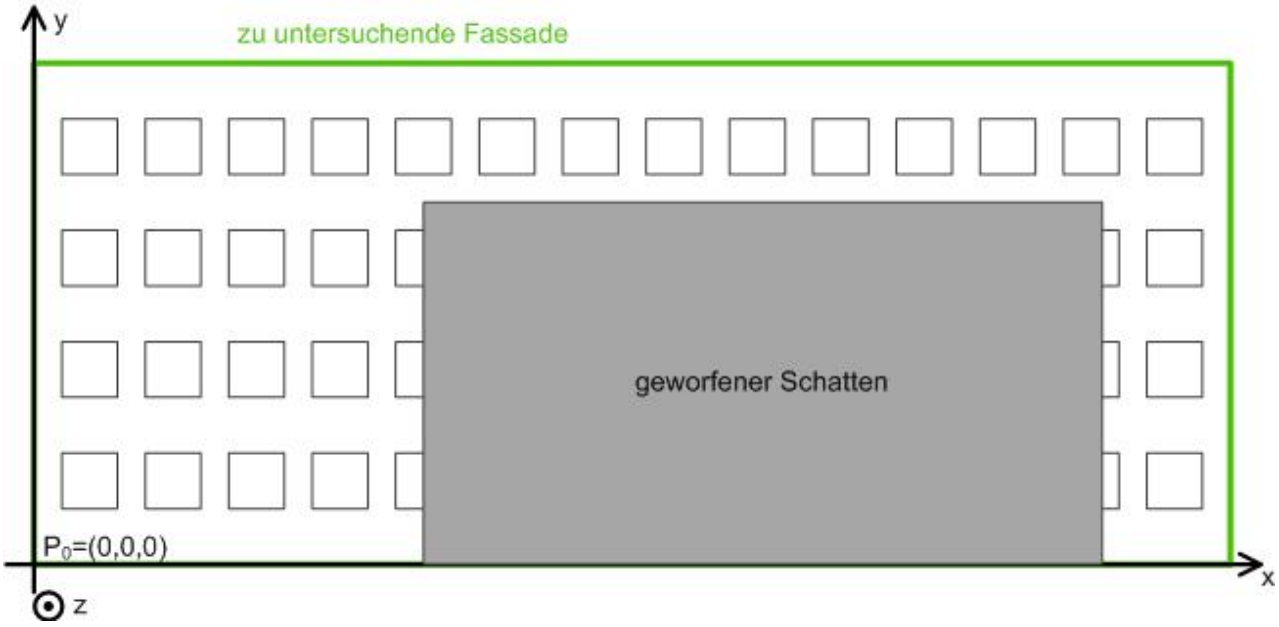
Die beiden folgenden Abbildungen sollen die Lage des Ursprungspunktes P_0 sowie die Ausrichtung des Koordinatensystems weiter verdeutlichen:

Betrachtung von der Seite



Anhand dieser Abbildung lässt sich auch der Elevationswinkel (Sonnenhöhe) darstellen: per Definition ist dieser bei Sonnenaufgang 0° (horizontaler Lichteinfall) und kann maximal 90° erreichen, dies jedoch nur an Orten innerhalb des nördlichen und südlichen Wendekreises.

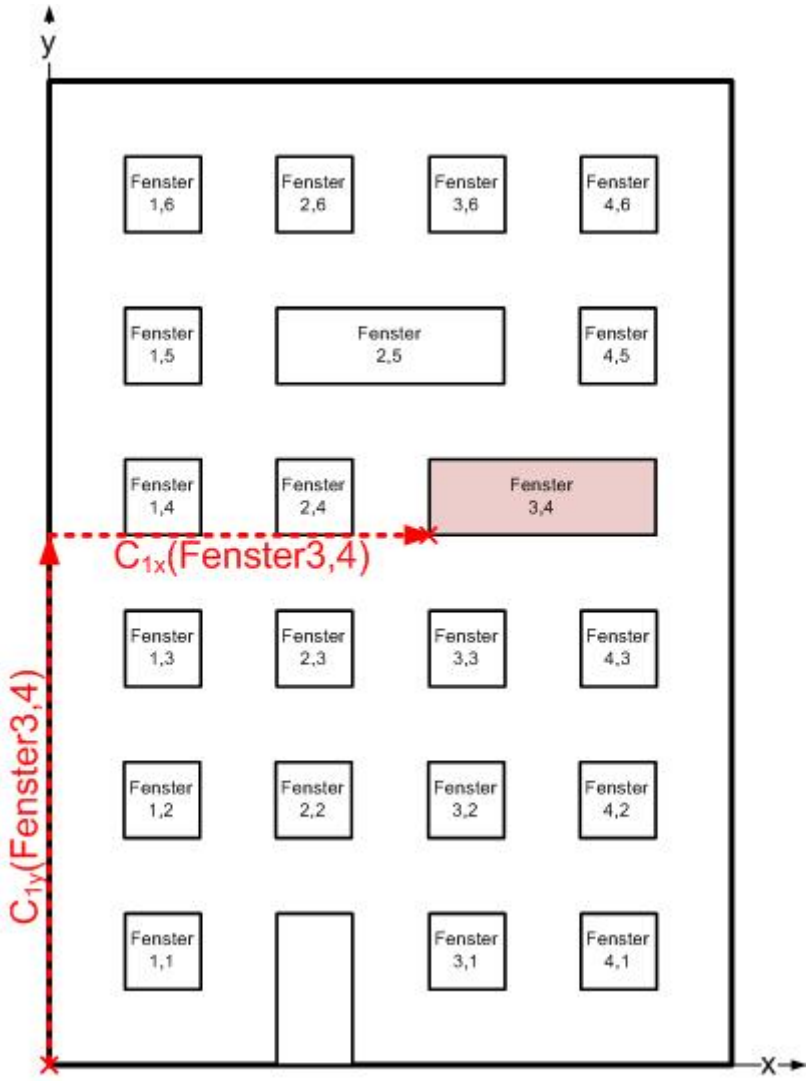
Betrachtung von vorne



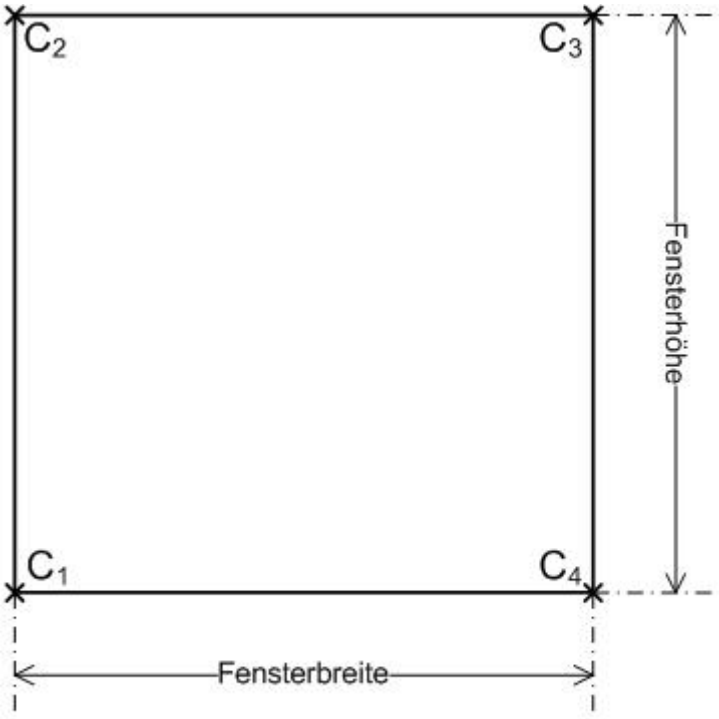
Hier ist die Lage des Koordinatenursprungs, P_0, am linken unteren Fußpunkt der Fassade noch einmal besonders deutlich. Darüber hinaus ist die x-y-Ausrichtung dargestellt, die später für den Eintrag der Fensterelemente wichtig ist.

Lage der Fenster

Die Lage der Fenster wird durch die Angabe ihres linken unteren Eckpunktes in Bezug auf das Fassaden-Koordinatensystems definiert. Da ein Fenster plan auf der Fassade liegt, ist die Eingabe auf die x- und die y-Koordinate beschränkt.



Zusätzlich sind die Breite und die Höhe anzugeben.



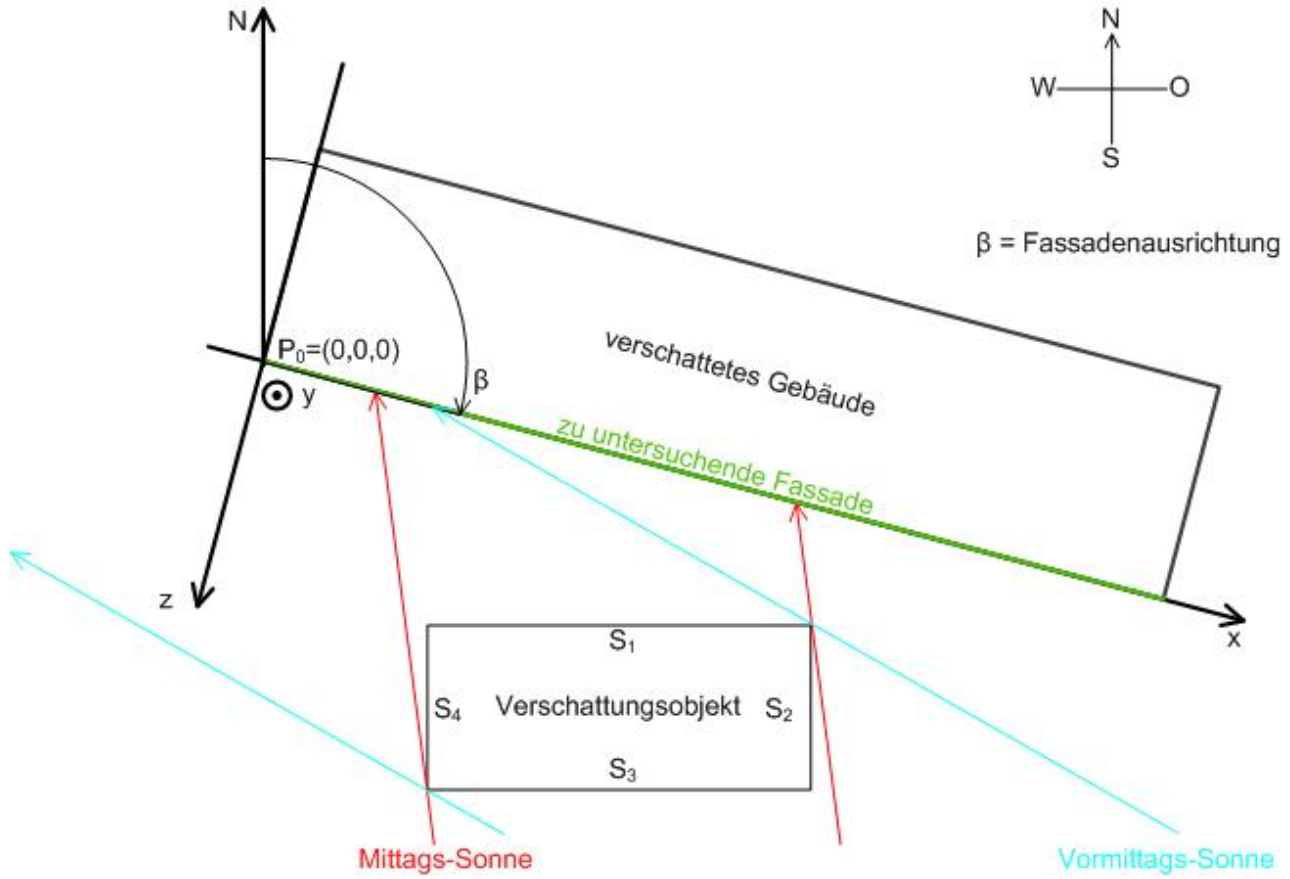
C_n=Ecke n (Corner)

Aus den eingetragenen Werten wird intern die Lage jedes Fenstereckpunktes auf der Fassade ermittelt. Ein Fenster gilt dann als verschattet, wenn alle Eckpunkte im Schatten liegen.

Positionierung der Verschattungsobjekte

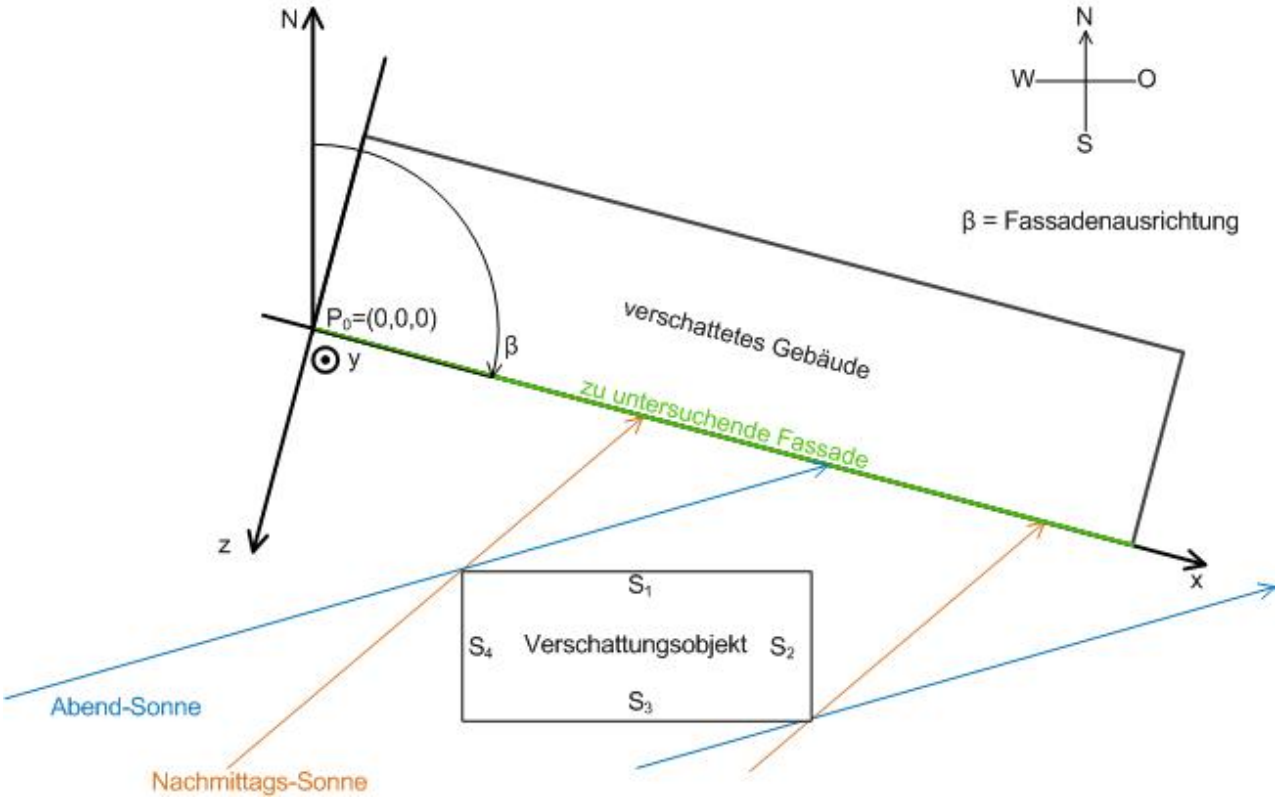
Bei der Beschreibung der Verschattungsobjekte wird zwischen eckigen Objekten (Gebäude, Pfeiler) und Objekten, die annähernd kugelförmig sind (z.B. Bäume), unterschieden. Eckige Objekte lassen sich ihrem Schattenwurf nach in viereckige Schatten werfende Fassaden unterteilen, wobei überlegt werden muss, welche über den Tag hinweg den Hauptschatten werfen:

Morgens/Mittags

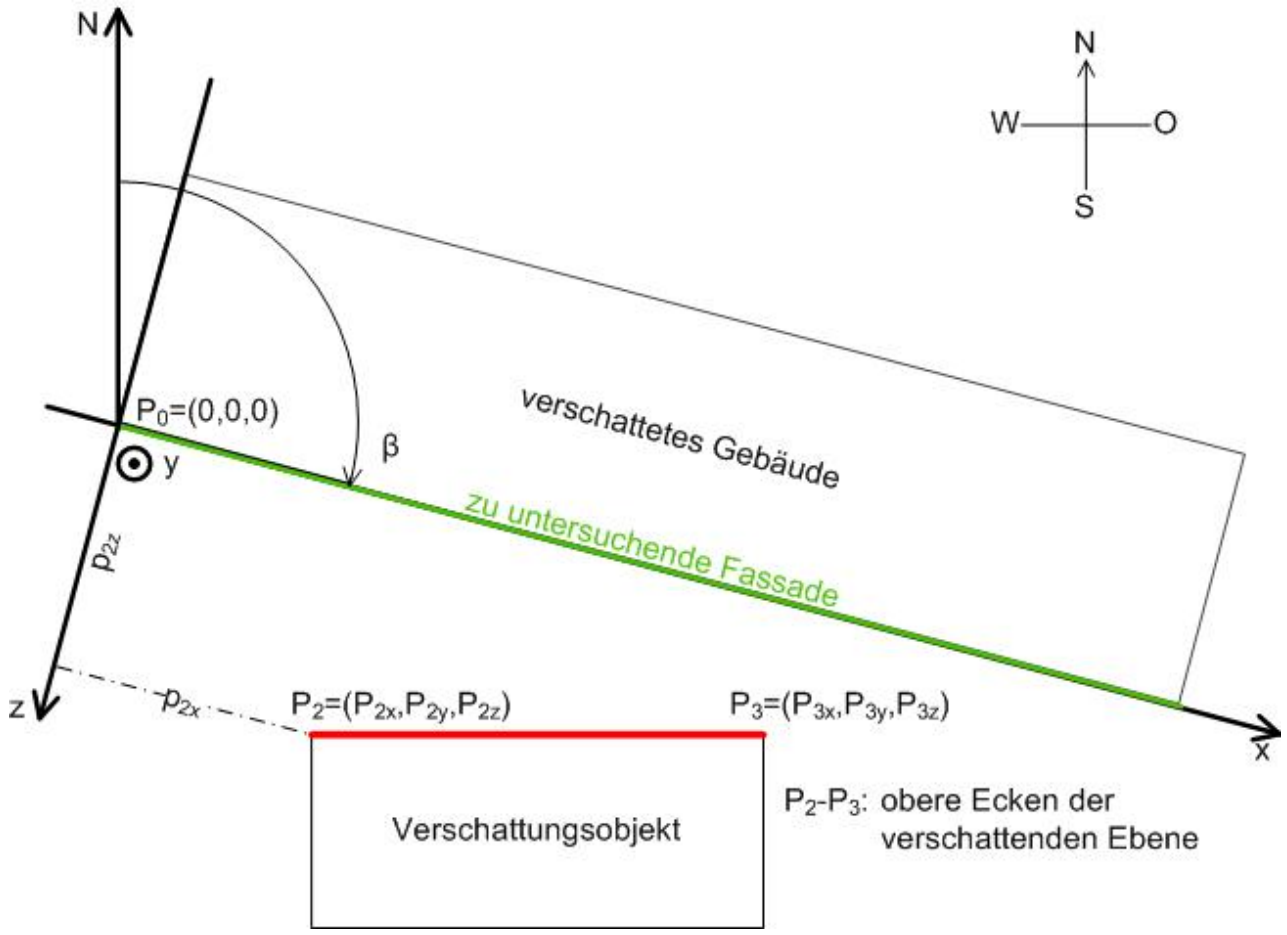


Morgens und Mittags würde der geworfene Schatten hauptsächlich durch die Seiten S_1 und S_4 gegeben sein, eine Betrachtung von S_2 und S_3, sollten sie nicht höher sein, wäre nicht nötig.

Nachmittags/Abends



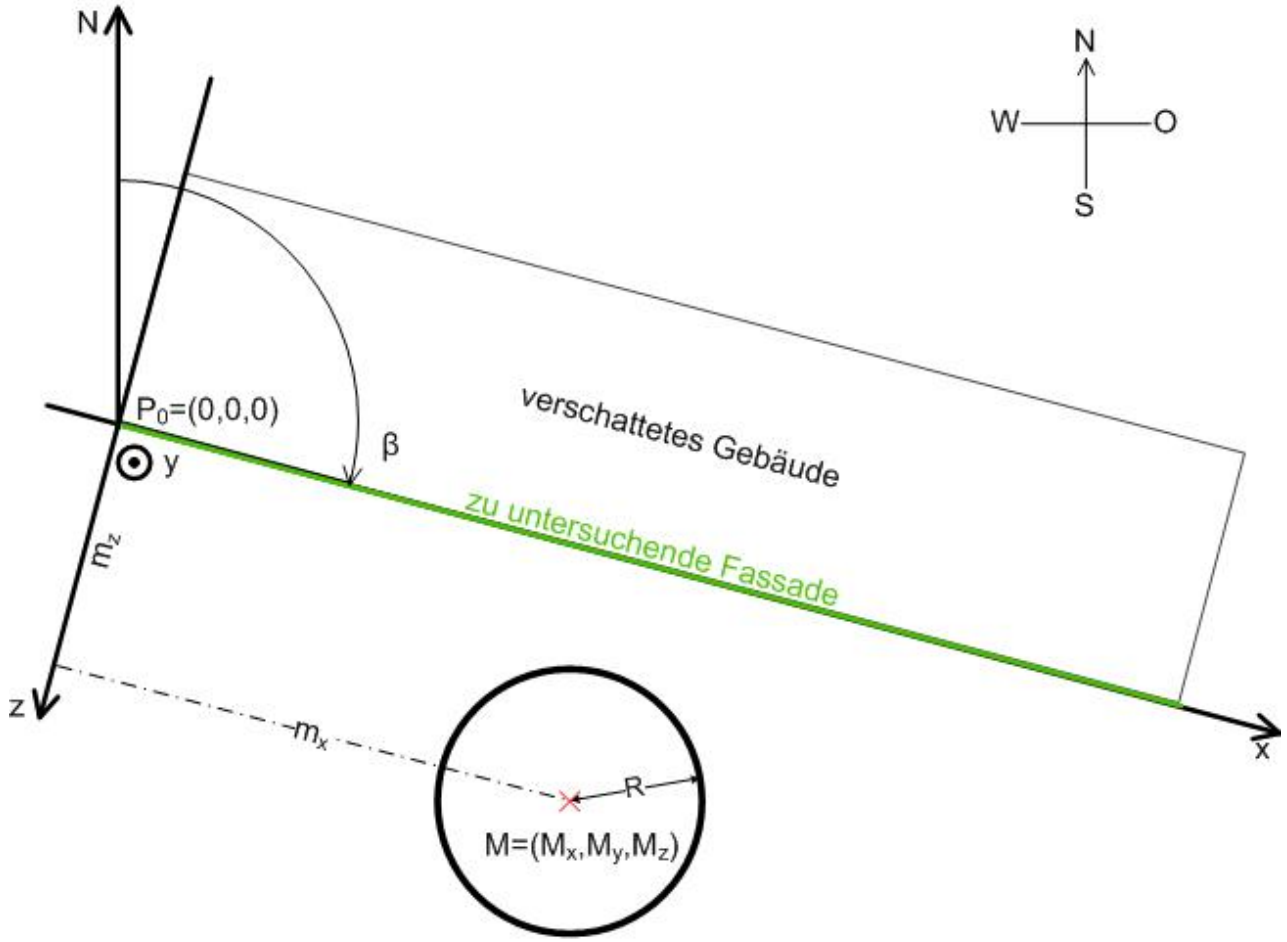
Auch am Nachmittag und am Abend lässt sich der Summen-Schatten allein durch die Betrachtung von S_1 und S_3 ermitteln. Es reicht also in diesem Fall S_1 und S_3 als Schattenwerfer anzugeben. Die Eingabe erfolgt dabei anhand der vier Eckpunkte bzw. deren Koordinaten in Bezug auf den Fassaden-Nullpunkt:



In dieser Skizze sind wegen der Draufsicht nur die oberen Punkte, P_2 und P_3 dargestellt. Der untere Punkt P_1 liegt unter P_2 und P_4 unter P_3.

Die Eingabe von verschattenden Kugelelementen erfolgt durch die Eingabe des Kugelmittelpunktes und des Radius:

Kugelelemente



Eine "Einteilung" des Kugelelementes, wie beim eckigen Gebäude, ist freilich nicht notwendig, da der Schattenwurf einer Kugel sich nur in seiner Richtung, nicht aber in seiner Größe ändert.

3.3.4.5 FB_BARBlindPositionEntry

Dieser Baustein dient zur Eingabe von Stützstellen für den Baustein FB_BARSunProtectionEx [▶ 226], sollte dieser im Modus der Höhenpositionierung mit Hilfe einer Tabelle betrieben werden, siehe E_BARPosMode [▶ 239].

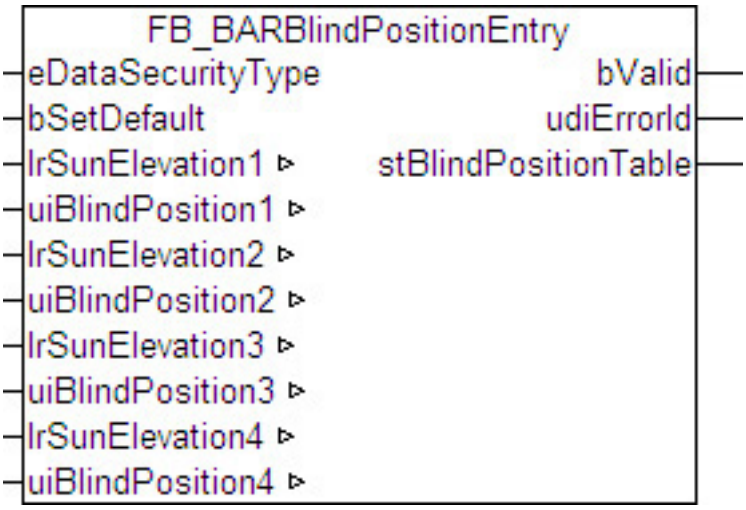
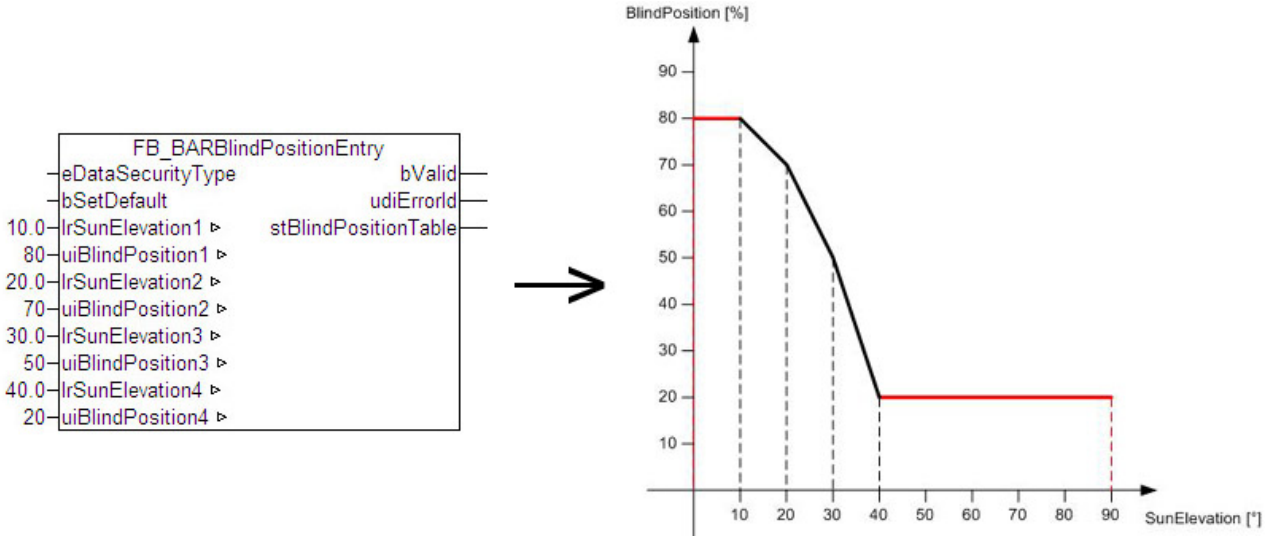


Abb. 7: FB_BARBlindPositionEntry

Der Baustein FB_BARSunProtectionEx [▶ 226] bietet neben den Betriebsarten "fixe Jalousiehöhe" und "maximaler Lichteinfall" auch die Möglichkeit, die Jalousiehöhe per Tabelleneinträge in Abhängigkeit von der Sonnenhöhe zu steuern. Durch die Eingabe mehrerer Stützpunkte wird linear interpoliert die betreffende Jalousiehöhe zum entsprechenden Sonnenstand errechnet. Da falsch eingetragene Werte jedoch zu Fehlfunktionen beim FB_BARSunProtectionEx [▶ 226] führen können ist diesem der Baustein FB_BARBlindPositionEntry voranzustellen. Es lassen sich vier Stützpunkte an diesem Baustein parametrieren, wobei ein fehlender Eintrag als Nulleintrag gewertet wird.
Der Baustein sortiert die eingegebenen Werte nicht selbstständig, sondern achtet darauf, dass die eingetragenen Sonnenstände der jeweiligen Stützstellen in aufsteigender Reihenfolge eingegeben wurden. Unbeabsichtigt fehlerhafte Einträge fallen dadurch schneller auf.
Die gewählten Werte für *rSunElevation1* .. *rSunElevation4* müssen auch eindeutig sein, es darf beispielsweise nicht gelten:
[*rSunElevation1* = 10 ; *uiBlindPosition1* = 50] und gleichzeitig [*rSunElevation2* = 10 *; uiBlindPosition2* = 30].
Das würde bedeuten, dass für ein und denselben Wert zwei verschiedene Zielwerte bestünden, was keinen eindeutigen funktionalen Zusammenhang bilden lässt.
Darüber hinaus müssen die Einträge für Sonnenstand und Jalousiehöhe im gültigen Bereich liegen. Das bedeutet mathematisch, dass folgende Bedingungen erfüllt sein müssen:

- rSunElevation1 < *rSunElevation2* < *rSunElevation3* < *rSunElevation4* - (Werte aufsteigend und ungleich)
- 0 ≤*rSunElevation*≤ 90 (in Grad - Gültigkeitsbereich Quellwerte)
- 0 ≤*uiBlindPosition*≤ 100 (in Prozent - Gültigkeitsbereich Zielwerte)

Der Baustein überprüft die eingetragenen Werte auf diese Bedingungen hin und gibt bei Nicht-Erfüllen einen Fehlercode [▶ 243] aus. Zusätzlich wird der Ausgang *bValid* auf FALSE gesetzt.
Des Weiteren sorgt der Baustein selbstständig für ein Ausfüllen der Randbereiche: Intern wird ein weiterer Stützpunkt bei *rSunElevation* = 0 mit *uiBlindPosition1*und ein weiterer oberhalb von *rSunElevation4* bei *rSunElevation* = 90 mit *uiBlindPosition4*aufgestellt. Damit wird sichergestellt, dass für alle gültigen Eingabewerte 0 ≤*rSunElevation*≤ 90 ein sinnvoller Zielwert vorhanden ist **ohne** dass ein Eintrag für *rSunElevation* = 0 und *rSunElevation* = 90 vom Anwender zwingend vergeben werden muss:



Die tatsächliche Anzahl an Stützstellen, welche an den Baustein FB_BARSunProtectionEx [▶ 226] übergeben wird, erhöht sich damit auf 6, siehe ST_BARBlindPositionTable [▶ 240].

Die Interpolation der Werte erfolgt im Blendschutz-Baustein.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Andernfalls wird der instanziierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

VAR_OUTPUT

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

bValid: Solange die Einträge den oben genannten Kriterien entsprechen, wird dieser Ausgang auf TRUE stehen.

uiErrorId: Enthält den Fehlercode, sollten die eingetragenen Werte nicht den oben aufgeführten Kriterien entsprechen. Siehe Fehlercodes [▶ 243].

stBlindPositionTable : Übergabestruktur der Stützstellen, siehe ST_BARBlindPositionTable [▶ 240].

VAR_IN_OUT

Damit die eingetragenen Parameter über einen Steuerungsausfall hinweg erhalten bleiben ist es erforderlich, sie als In-Out-Variablen zu deklarieren. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variablen wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

```
rSunElevation1   : REAL;;
uiBlindPosition1 : UINT;
rSunElevation2   : REAL;
uiBlindPosition2 : UINT;
rSunElevation3   : REAL;
uiBlindPosition3 : UINT;
rSunElevation4   : REAL;
uiBlindPosition4 : UINT;
```

rSunElevation1: Sonnenstand des 1. Stützpunktes (0°..90°).

uiBlindPosition1: Jalousieposition (Grad der Schließung) des 1. Stützpunktes (0%..100%).

rSunElevation2: Sonnenstand des 2. Stützpunktes (0°..90°).

uiBlindPosition2: Jalousieposition (Grad der Schließung) des 2. Stützpunktes (0%..100%).

rSunElevation3: Sonnenstand des 3. Stützpunktes (0°..90°).

uiBlindPosition3: Jalousieposition (Grad der Schließung) des 3. Stützpunktes (0%..100%).

rSunElevation4: Sonnenstand des 4. Stützpunktes (0°..90°).

uiBlindPosition4: Jalousieposition (Grad der Schließung) des 4. Stützpunktes (0%..100%).

3.3.4.6 FB_BARDelayedHysteresis

Dieser Baustein stellt einen Schwellwertschalter für Helligkeit dar. Das Ein- bzw. Ausschaltverhalten kann zusätzlich zeitlich verzögert werden.

VAR_INPUT

```
eDataSecurityType  : E_HVACDataSecurityType;
uiBrightness       : UINT;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Andernfalls wird der instanziierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

uiBrightness: Außenhelligkeit in Lux.

VAR_OUTPUT

```
bOut       : BOOL;
bError     : BOOL;
udiErrorId : UDINT;
```

bOut: Binärer verzögerter Ausgang des Schwellwertschalters

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId: Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

VAR_IN_OUT

Damit die eingetragenen Parameter über einen Steuerungsausfall hinweg erhalten bleiben ist es erforderlich, sie als In-Out-Variablen zu deklarieren. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variablen wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

```
uiOnValue  : UINT;
uiOffValue : UINT;
uiOnDelay  : UINT;
uiOnDelay  : UINT;
```

uiOnValue: Einschaltschwellwert in Lux. Dieser muss größer sein als der Ausschaltschwellwert *usiOffValue.*

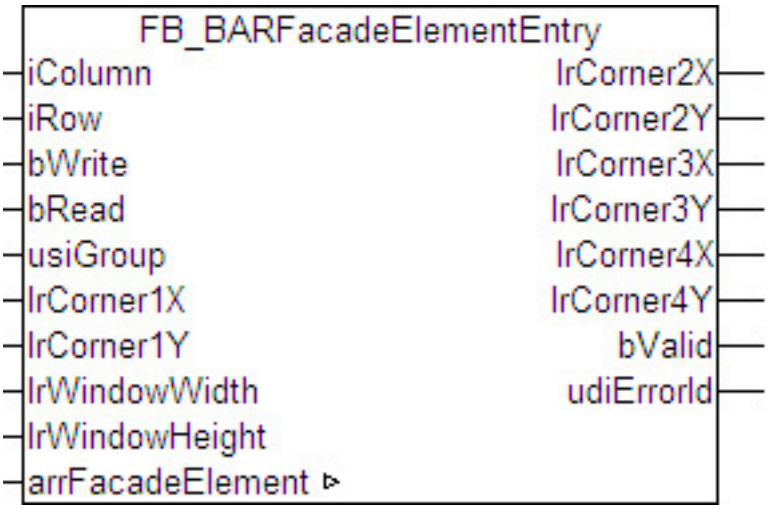
uiOffValue: Ausschaltschwellwert in Lux. Dieser muss kleiner sein als der Einschaltschwellwert *usiOnValue.*

uiOnDelay: Einschaltverzögerung in Sekunden.

uiOffDelay: Ausschaltverzögerung in Sekunden.

3.3.4.7 FB_BARFacadeElementEntry

Dieser Baustein dient zur Verwaltung aller Fassadenelemente (Fenster) einer Fassade, welche global in einer Liste von Fassadenelementen [▶ 243] hinterlegt ist. Er soll die Eingabe der Elementinformationen - auch im Hinblick auf die Nutzung der Target-Visualisierung - erleichtern. Eine schematische Darstellung der Objekte mit Beschreibung der Koordinaten ist unter Verschattungskorrektur: Grundlagen und Definitionen [▶ 156] gegeben.

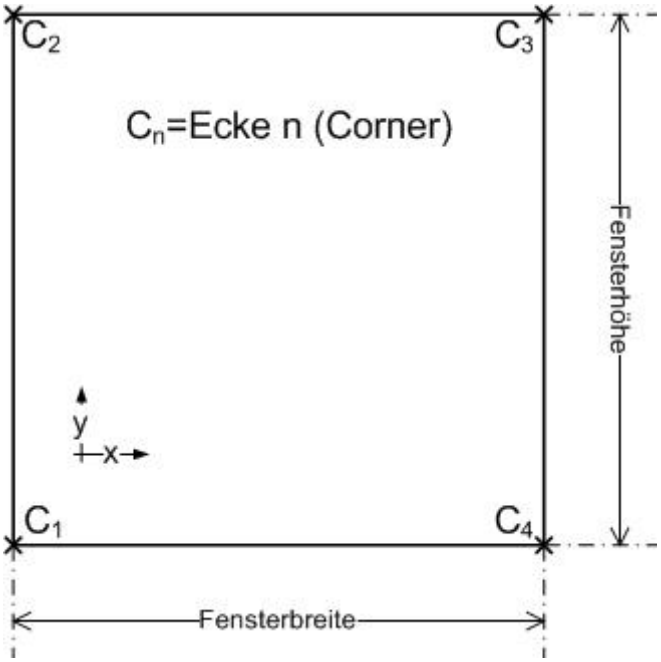


Die Deklaration der Fassadenelemente erfolgt in den globalen Variablen als zweidimensionales Feld über die Fensterspalten (Columns) und -reihen (Rows):

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

Jedes einzelne Element *arrFacadeElement* trägt die Informationen für jeweils ein Fassadenelement (ST_BARFacadeElement [▶ 241]). Dazu gehören die Gruppenzugehörigkeit, die Abmessungen (Breite, Höhe) und die Koordinaten der Eckpunkte. Der Baustein greift dabei über die IN-OUT-Variable *arrFacadeElement* direkt auf dieses Feld zu.

Die Tatsache, dass die Koordinaten der Eckpunkte 2 bis 4 Ausgangswerte sind, ergibt sich daraus, dass sie aus den Eingabeparametern gebildet werden und für die Verwendung in einer Visualisierung zur Verfügung stehen sollen.



Alle Angaben in Meter!

lrCorner2X = lrCorner1X
lrCorner2Y = lrCorner1Y + lrWindowHeight (Fensterhöhe)
lrCorner3X = lrCorner1X + lrWindowWidth (Fensterbreite)
lrCorner3Y = lrCorner2Y
lrCorner4X = lrCorner1X + lrWindowWidth (Fensterbreite)
lrCorner4Y = lrCorner1Y

Die Verwendung des Bausteines erfolgt in drei Schritten:

- Auslesen
- Ändern
- Schreiben

Auslesen

Mit den Einträgen an *iColumn* und *iRow* wird das entsprechende Element aus der Liste, *arrFacadeElement[iColumn,iRow]*, ausgewählt. Eine steigende Flanke an *bRead* liest folgende Daten aus dem Listenelement aus:

- usiGroup Gruppenzugehörigkeit,
- lrCorner1X X-Koordinate des Eckpunktes1 in Meter
- lrCorner1Y Y-Koordinate des Eckpunktes1 in Meter
- lrWindowWidth Fensterbreite in Meter
- lrWindowHeight Fensterhöhe in Meter

Diese werden dann den entsprechenden Eingangsvariablen des Bausteines zugewiesen, der daraus nach dem oben erläuterten Zusammenhang die Koordinaten der Eckpunkte 2-4 als Ausgangsvariablen errechnet. Wichtig ist hierbei, dass im Schritt des Auslesens die Eingangswerte nicht überschrieben werden. So lassen sich alle Werte zunächst in einer Visualisierung anzeigen.

Ändern

In einem nächsten Programmschritt können dann die aufgeführten Eingabewerte verändert werden. Die eingegebenen Werte werden dabei ständig auf Plausibilität überprüft. Der Ausgang *bValid* zeigt an, ob die Werte gültig sind (*bValid*=TRUE). Sollte dies nicht der Fall sein, wird am Ausgang *udiErrorId* ein entsprechender Fehlercode [▶ 243] ausgegeben. Siehe auch unten "Fehler (bValid=FALSE)".

Schreiben

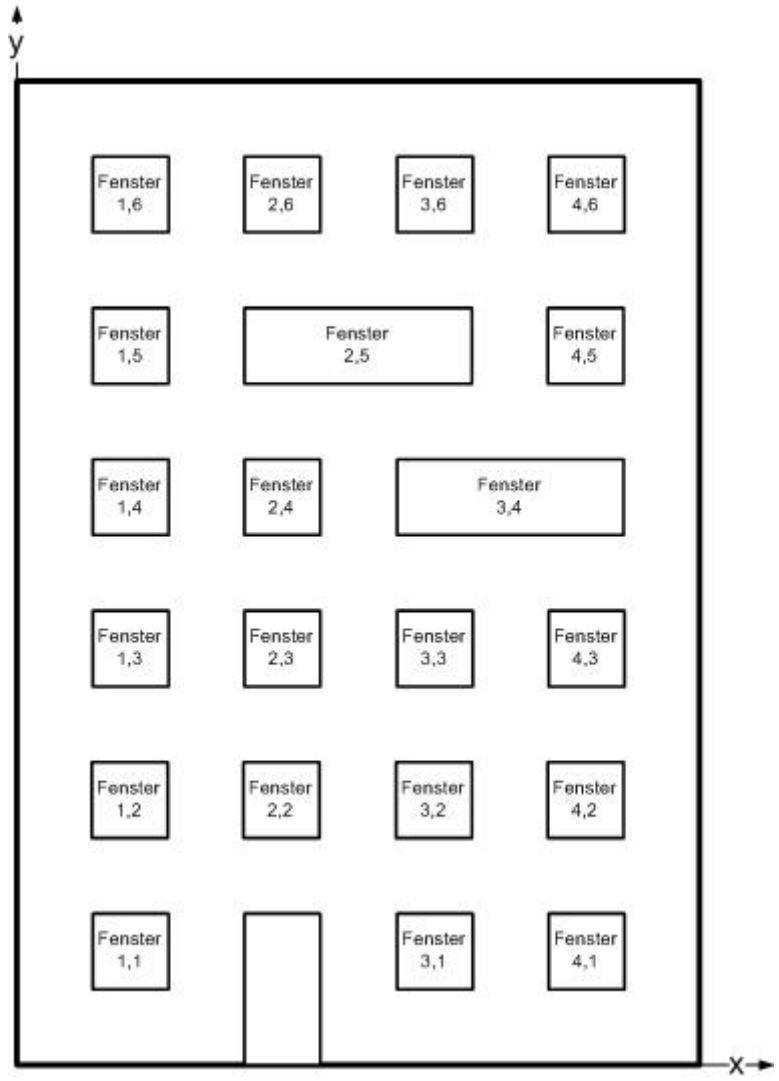
Mit einer positiven Flanke an *bWrite* werden die parametrierten Daten in das Listenelement mit dem Index *nId* geschrieben, unabhängig davon, ob sie gültige Werte darstellen oder nicht. Daher ist innerhalb der Elementstruktur ST_BARFacadeElement [▶ 241] ebenfalls ein Plausibilitätsbit *bValue* vorhanden, das genau diese Information an den Baustein FB_BARShadingCorrection [▶ 180] / FB_BARShadingCorrectionSouth [▶ 183] weiterreicht und dort Fehlberechnungen vorbeugt.

Fehler (bValid=FALSE)

Der Baustein FB_BARShadingCorrection [▶ 180] / FB_BARShadingCorrectionSouth [▶ 183], welcher beurteilt, ob alle Fenster einer Gruppe verschattet sind, wird nur dann seine Aufgabe ausführen, wenn alle Fenster der betrachteten Gruppe gültige Einträge haben.
Das bedeutet:

- usiGroup muss größer als 0 sein
- lrCorner1X muss größer oder gleich 0.0 sein
- lrCorner1Y muss größer oder gleich 0.0 sein
- lrWindowWidth muss größer als 0 sein
- lrWindowHeight muss größer als 0 sein

Ist eines dieser Kriterien nicht erfüllt, so wird dies als Falscheingabe interpretiert und *bValid* am Bausteinausgang von FB_BARFacadeElementEntry sowie im Fensterelement ST_BARFacadeElement [▶ 241] auf FALSE gesetzt.
Sind hingegen alle Einträge eines Fassadenelementes Null, wird es als gültiges, gewollt ausgelassenes Fassadenelement angesehen:



Bei einer Fassade von 6x4 Fenstern wären hier die Elemente Fenster (2,1), Fenster (3,5) und Fenster (4,4) Leerelemente.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
iColumn          : INT;
iRow             : INT;
bWrite           : BOOL;
bRead            : BOOL;
usiGroup         : USINT;
lrCorner1X       : LREAL;
lrCorner1Y       : LREAL;
lrWindowWidth    : LREAL;
lrWindowHeight   : LREAL;
```

iColumn: Spalten-Index des gewählten Elementes auf der Fassade. Dies bezieht sich auf die Auswahl eines Feldelementes des unter der IN-OUT-Variablen *arrFacadeElement* angelegten Arrays.

iRow: dto. Reihen-Index. **iRow und iColumn dürfen nicht Null sein!** Das ergibt sich aus der Felddefinition, siehe oben.

bRead: Mit einer positiven Flanke an diesem Eingang werden die Informationen des gewählten Elementes, *arrFacadeElement[iColumn,iRow]* in den Baustein gelesen und den Eingangsvariablen *usiGroup* bis *rWindowHeight*zugewiesen. Daraus ergeben sich dann die Ausgangsvariablen rCorner2X bis rCorner4Y. Sind zu Zeitpunkt des Auslesens schon Daten an den Eingängen *usiGroup* bis *rWindowHeight* angelegt, so werden die zuvor ausgelesenen Daten sofort mit diesen überschrieben.

bWrite: Eine positive Flanke schreibt die eingegebenen sowie errechneten Werte in das gewählte Feldelement *arrFacadeElement[iColumn,iRow].*

usiGroup: Gruppenzugehörigkeit,

lrCorner1X: X-Koordinate des Eckpunktes1 in Meter.

lrCorner1Y: Y-Koordinate des Eckpunktes1 in Meter.

lrWindowWidth: Fensterbreite in Meter.

lrWindowHeight: Fensterhöhe in Meter.

VAR_OUTPUT

```
lrCorner2X : LREAL;
lrCorner2Y : LREAL;
lrCorner3X : LREAL;
lrCorner3Y : LREAL;
lrCorner4X : LREAL;
lrCorner4Y : LREAL;
bValid     : BOOL;
udiErrorId : UDINT;
```

lrCorner2X: Ermittelte X-Koordinate des Eckpunktes 2 des Fensters in Meter. Siehe "Info" oben.

lrCorner2Y: Ermittelte Y-Koordinate des Eckpunktes 2 des Fensters in Meter. Siehe " Info " oben.

lrCorner3X: Ermittelte X-Koordinate des Eckpunktes 3 des Fensters in Meter. Siehe " Info " oben.

lrCorner3Y: Ermittelte Y-Koordinate des Eckpunktes 3 des Fensters in Meter. Siehe " Info " oben.

lrCorner4X: Ermittelte X-Koordinate des Eckpunktes 4 des Fensters in Meter. Siehe " Info " oben.

lrCorner4Y: Ermittelte Y-Koordinate des Eckpunktes 4 des Fensters in Meter. Siehe " Info " oben.

bValid: Ergebnis Kontrolle für die eingegebenen Werte.

udiErrorId: Enthält den Fehlercode, sollten die eingetragenen Werte nicht in Ordnung sein. Siehe Fehlercodes [▶ 243].

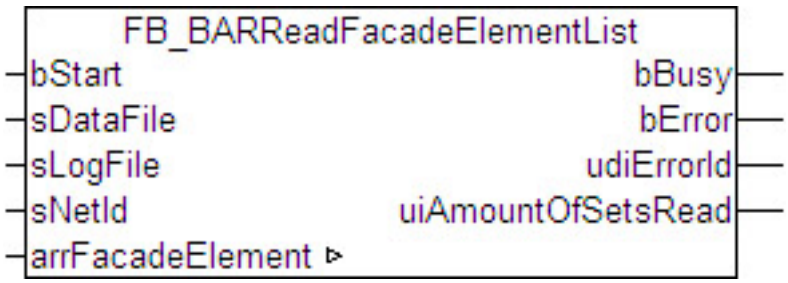
VAR_IN_OUT

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

arrFacadeElement: Liste von Fassadenelementen [▶ 243].

3.3.4.8 FB_BARReadFacadeElementList

Mit Hilfe dieses Bausteines lassen sich Daten für Fassadenelemente (Fenster) aus einer vordefinierten Excel-Tabelle im csv-Format in die Liste der Fassadenelemente [▶ 243] importieren. Zusätzlich werden die importierten Daten auf Plausibilität überprüft und Fehler in eine Log-Datei geschrieben.



VAR_INPUT

```
bStart    : BOOL;
sDataFile : STRING;
sLogFile  : STRING;
sNetId    : STRING;
```

bStart: Eine TRUE-Flanke an diesem Eingang startet den Lesevorgang.

sDataFile: Enthält den Pfad- und Dateinamen der zu öffnenden Daten-Datei. Diese muss in Excel als Dateityp "CSV (Trennzeichen-getrennt) (*.csv)" abgespeichert worden sein. Öffnet man die Datei mit einem einfachen Text-Editor, so müssen die Werte durch Semikolons getrennt dargestellt sein. Beispiel für einen Eintrag: *sDataFile:= 'C:\Projekte\FacadeElements.csv'*

sLogFile: dto. Log-File für die auflaufenden Fehler. Diese Datei wird bei jeder neuen Bausteinaktivierung überschrieben, so dass nur aktuelle Fehler enthalten sind.

sNetId: Hier kann ein String mit der AMS Net Id des TwinCAT-Rechners angegeben werden, auf dem die Dateien geschrieben/gelesen werden sollen. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

Die Speicherung der Daten kann nur auf dem Steuerungsrechner selbst und den Rechnern erfolgen, die mit dem Steuerungsrechner per ADS verbunden sind. Verweise auf lokale Festplatten dieser Rechner sind möglich, nicht jedoch auf verbundene Netzwerkfestplatten.

VAR_OUTPUT

```
bBusy             : BOOL;
bError            : BOOL;
udiErrorId        : UDINT;
uiAmountOfSetsRead: UINT;
```

bBusy: Dieser Ausgang steht auf TRUE, solange Elemente aus der Datei ausgelesen werden.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind oder wenn ein Datei Schreib- bzw. Lesefehler aufgetreten ist.

udiErrorId: Enthält den Fehlercode des zuletzt aufgetretenen Fehlers. Siehe Fehlercodes [▶ 243] bzw. ADS_Fehlercodes.

uiAmountOfSetsRead: Anzahl der gelesenen Datensätze

VAR_IN_OUT

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

arrFacadeElement: Liste von Fassadenelementen [▶ 243].

Excel-Tabelle

Das folgende Beispiel zeigt die Excel-Tabelle mit den Einträgen der Fensterelemente.
Alle Textfelder sind frei beschreibbar, wichtig sind die grün markierten Felder, wobei dort jede Zeile einen Datensatz kennzeichnet.
Folgende Regeln sind zu beachten:

- Ein Datensatz muss immer mit einem '@' beginnen.
- Die Indizes *IndexColumn* und *IndexRow* müssen innerhalb der definierten Grenzen liegen, siehe Liste von Fassadenelementen [▶ 243]. Mit diesen Indizes wird direkt beschrieben, in welches Fassadenelement der Liste *arrFacadeElements* die Daten des Satzes hineingelegt werden.
- Fensterbreite und Fensterhöhe müssen größer Null sein
- Die Eck-Koordinaten P1x und P1y müssen größer oder gleich Null sein.
- Jedes Fensterelement muss einer Gruppe 1..255 zugeordnet sein.
- Die Gesamtgröße der Tabelle darf systembedingt 65534 Bytes nicht überschreiten.
- Die Tabelle muss in Excel als Dateityp "CSV (Trennzeichen-getrennt) (*.csv)" abgespeichert worden sein.

Es ist nicht notwendig, alle Fensterelemente zu beschreiben, die per Definition bzw. Deklaration möglich wären. Bevor die neue Liste eingelesen wird, löscht der Baustein die gesamte alte Liste im Programm. Alle Elemente, welche dann nicht durch Einträge in der Excel-Tabelle beschrieben werden, haben reine Null-Einträge und sind dadurch als nicht-vorhanden gekennzeichnet und auch nicht auswertbar, da der Baustein zur Verschattungskorrektur, FB_BARSunProtectionEx [▶ 226], Elemente mit Gruppeneintrag '0' nicht annimmt.

DE_FacadeElements.xls [Kompatibilitätsmodus]

	A	B	C	D	E	F	G	H	I	J
1	Nummer	Beschreibung		IndexColumi	IndexRow	Fensterbreit	Fensterhoeh	P1x	P1y	Gruppe
2				(Achse)	(Stockwerk)	[m]	[m]	[m]	[m]	
3		Text								
4	1	Beschreibung	@	1	1	1,2	1,3	1,5	1	2
5	2	Beschreibung	@	0	1	1,2	1,3	2,7	1	2
6	3	Beschreibung	@	3	1	1,2	1,3	4,4	1	2
7	4	Beschreibung	@	4	1	1,2	1,3	6,1	1	2
8	5	Beschreibung	@	5	1	1,2	1,3	7,8	1	2
9	6	Beschreibung	@	6	1	1,2	1,3	9,5	1	2
10	7	Beschreibung	@	7	1	1,2	1,3	11,2	1	2
11	8	Beschreibung	@	8	1	1,2	1,3	12,9	1	2
12	9	Beschreibung	@	9	1	1,2	1,3	14,6	1	2
13	10	Beschreibung	@	10	1	1,2	1,3	16,3	1	2
14	11	Beschreibung	@	1	1	1,2	1,3	1,5	4	2
15	12	Beschreibung	@	0	1	1,2	1,3	2,7	4	2
16	13	Beschreibung	@	3	1	1,2	1,3	4,4	4	2
17	14	Beschreibung	@	4	1	1,2	1,3	6,1	4	2
18	15	Beschreibung	@	5	1	1,2	1,3	7,8	4	2
19	16	Beschreibung	@	6	1	1,2	1,3	9,5	4	2
20	17	Beschreibung	@	7	1	1,2	1,3	11,2	4	2
21	18	Beschreibung	@	8	1	1,2	1,3	12,9	4	2
22	19	Beschreibung	@	9	1	1,2	1,3	14,6	4	2
23	20	Beschreibung	@	10	1	1,2	1,3	16,3	4	2
24	21	Beschreibung	@	1	1	1,2	1,3	1,5	7	2
25	22	Beschreibung	@	0	1	1,2	1,3	2,7	7	2
26	23	Beschreibung	@	3	1	1,2	1,3	4,4	7	2
27	24	Beschreibung	@	4	1	1,2	1,3	6,1	7	2
28	25	Beschreibung	@	5	1	1,2	1,3	7,8	7	2
29	26	Beschreibung	@	6	1	1,2	1,3	9,5	7	2
30	27	Beschreibung	@	7	1	1,2	1,3	11,2	7	2
31	28	Beschreibung	@	8	1	1,2	1,3	12,9	7	2
32	29	Beschreibung	@	9	1	1,2	1,3	14,6	7	2
33	30	Beschreibung	@	10	1	1,2	1,3	16,3	7	2
34	31	Beschreibung	@	1	1	1,2	1,3	1,5	10	2
35	32	Beschreibung	@	0	1	1,2	1,3	2,7	10	2
36	33	Beschreibung	@	3	1	1,2	1,3	4,4	10	2
37	34	Beschreibung	@	4	1	1,2	1,3	6,1	10	2
38	35	Beschreibung	@	5	1	1,2	1,3	7,8	10	2
39	36	Beschreibung	@	6	1	1,2	1,3	9,5	10	2
40	37	Beschreibung	@	7	1	1,2	1,3	11,2	10	2
41	38	Beschreibung	@	8	1	1,2	1,3	12,9	10	2
42	39	Beschreibung	@	9	1	1,2	1,3	14,6	10	2
43	40	Beschreibung	@	10	1	1,2	1,3	16,3	10	2
44										

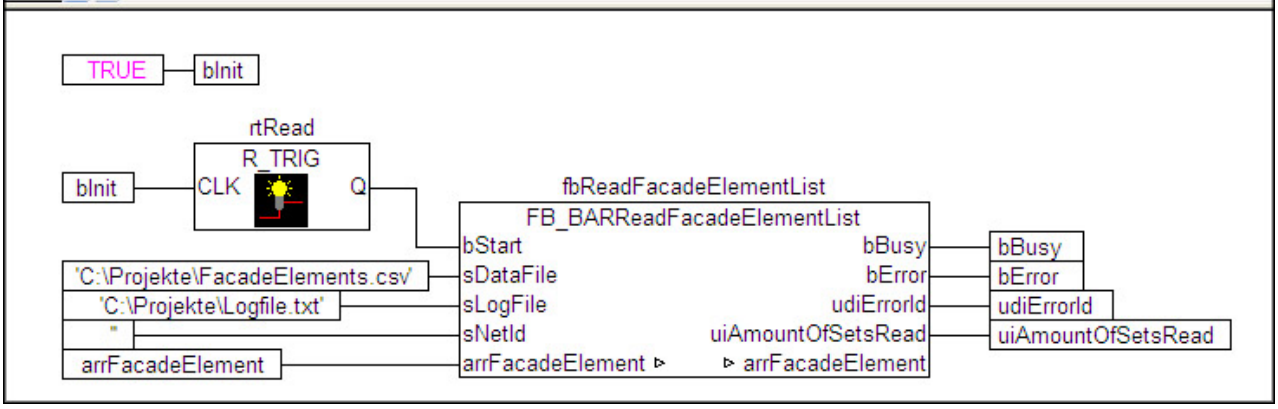
Log-Datei

Bei jedem Start des Lesebausteines wird die Logdatei neu beschrieben und der alte Inhalt gelöscht. Ist die Log-Datei nicht vorhanden, so wird sie zunächst automatisch erstellt. Die Log-Datei enthält dann entweder eine OK Meldung oder eine Auflistung aller aufgetretenen Fehler. Fehler, welche mit dem Öffnen, Beschreiben oder Schließen der Logdatei selbst in Zusammenhang stehen, können nicht mitgeschrieben

werden. Daher ist immer auch der Ausgang *udiErrorId* des Lesebausteines zu beachten, welcher den letzten Fehlercode anzeigt. Da im Ablauf des Lesens die Log-Datei immer als Letztes geschlossen wird, ist ein entsprechender Hinweis im Fehlerfall gewährleistet.

Programmbeispiel

```
0001 PROGRAM ReadFacadeElements
0002 VAR
0003     bInit                      : BOOL;
0004     rtRead                     : R_TRIG;
0005     fbReadFacadeElementList    : FB_BARReadFacadeElementList;
0006     arrFacadeElement           : ARRAY[1..iColumnsPerFacade, 1..iRowsPerFacade] OF ST_BARFacadeElement;
0007
0008     bBusy                      : BOOL;
0009     bError                     : BOOL;
0010     udiErrorId                 : UDINT;
0011     uiAmountOfSetsRead         : UINT;
0012 END_VAR
```

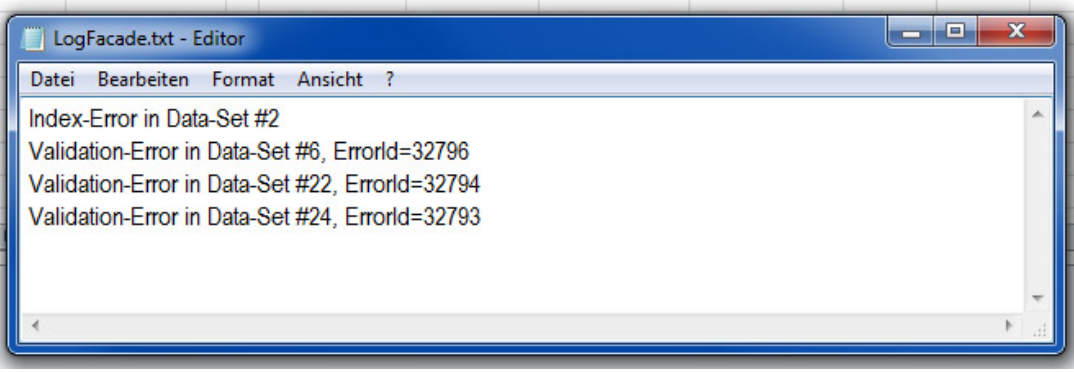


In diesem Beispiel wird bei PLC-Start die Variable *bInit* zunächst auf TRUE gesetzt. Damit erhält der Eingang *bStart* am Baustein fbReadFacadeElementList einmalig eine steigende Flanke, welcher den Lesevorgang auslöst. Gelesen wird die Datei "FacadeElements.csv", welche sich im Ordner "C:\Projekte\" befindet. In demselben Ordner wird dann die Logdatei "Logfile.txt" hinterlegt. Ist diese Logdatei noch nicht vorhanden wird sie erstellt, andernfalls wird der bestehende Inhalt überschrieben. Das Lesen und Schreiben erfolgt auf demselben Rechner, auch dem sich auch die PLC befindet. Das wird durch den Eingang *sNetID* = '' (=lokal) definiert. Alle Daten werden in die im Programm deklarierte Liste *arrFacadeElement* geschrieben. Solange gelesen und geschrieben wird ist der Ausgang *bBusy* auf TRUE. Der zuletzt auftretende Fehler wird an *udiErrorId* angezeigt, *bError* steht dann auf TRUE. Die Anzahl der gefundenen und gelesenen Datenzeilen wird an *uiAmountOfSetsRead* zur Kontrolle angezeigt.

In die folgende Excel Liste wurden die markierten Fehler "eingebaut". Dadurch ergibt sich das gezeigte Log-File:

DE_FacadeElements.csv

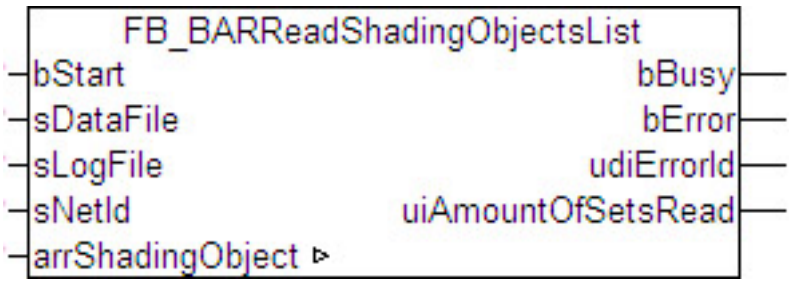
	A	B	C	D	E	F	G	H	I	J
1	Nummer	Beschreibung		IndexColumn	IndexRow	Fensterbreite	Fensterhoehe	P1x	P1y	Gruppe
2				(Achse)	(Stockwerk)	[m]	[m]	[m]	[m]	
3		Text								
4	1	Beschreibung	@	1	1	1,2	1,3	1,5	1,4	2
5	2	Beschreibung	@	0	1	1,2	1,3	2,7	1	2
6	3	Beschreibung	@	3	1	1,2	1,3	4,4	1	2
7	4	Beschreibung	@	4	1	1,2	1,3	6,1	1	2
8	5	Beschreibung	@	5	1	1,2	1,3	7,8	1	2
9	6	Beschreibung	@	6	1	0	1,3	9,5	1	2
10	7	Beschreibung	@	7	1	1,2	1,3	11,2	1	2
11	8	Beschreibung	@	8	1	1,2	1,3	12,9	1	2
12	9	Beschreibung	@	9	1	1,2	1,3	14,6	1	2
13	10	Beschreibung	@	10	1	1,2	1,3	16,3	1	5
14	11	Beschreibung	@	1	2	1,2	1,3	1	1	5
15	12	Beschreibung	@	2	2	1,2	1,3	2,7	3	5
16	13	Beschreibung	@	3	2	1,2	1,3	4,4	4	5
17	14	Beschreibung	@	4	2	1,2	1,3	4,4	4	5
18	15	Beschreibung	@	5	2	1,2	1,3	7,8	4	5
19	16	Beschreibung	@	6	2	1,2	1,3	9,5	4	5
20	17	Beschreibung	@	7	2	1,2	1,3	11,2	4	5
21	18	Beschreibung	@	8	2	1,2	1,3	12,9	4	5
22	19	Beschreibung	@	9	2	1,2	1,3	14,6	4	3
23	20	Beschreibung	@	10	2	1,2	1,3	16,3	4	3
24										
25	31	Beschreibung	@	1	3	1,2	1,3	1	7	3
26	32	Beschreibung	@	2	3	1,2	1,3	-1	6	3
27	33	Beschreibung	@	3	3	1,2	1,3	4,4	7	3
28	34	Beschreibung	@	4	3	1,2	1,3	6,1	7	0
29	35	Beschreibung	@	5	3	1,2	1,3	7,8	7	3
30	36	Beschreibung	@	6	3	1,2	1,3	9,5	7	3
31	37	Beschreibung	@	7	3	1,2	1,3	11,2	7	3
32	38	Beschreibung	@	8	3	1,2	1,3	12,9	7	7
33	39	Beschreibung	@	9	3	1,2	1,3	14,6	7	7
34	40	Beschreibung	@	10	3	1,2	1,3	16,3	7	7
35										



Der erste Fehler ist im Datensatz 2 und ein Index-Fehler, da "0" nicht erlaubt ist.
Der nächste Fehler in Datensatz 6 wurde nach Validierung der Daten mit dem intern genutzten Baustein FB_BARShadingObjectsEntry [▶ 186] gefunden und daher näher mit einer Fehlernummer versehen, welche unter Fehlercodes [▶ 243] näher aufgeschlüsselt wird. Der dritte und der vierte Fehler trat ebenfalls nach der internen Validierung auf. Wichtig hierbei ist, dass sich die Datensatz-Nummern (hier 22 und 24) nicht nach den eingetragenen Nummern in der Liste, sondern nach den tatsächlichen laufenden Nummern richten: eingelesen wurden hier lediglich 30 Datensätze.

3.3.4.9 FB_BARReadShadingObjectsList

Mit Hilfe dieses Bausteines lassen sich Daten für Verschattungsobjekte aus einer vordefinierten Excel-Tabelle im csv-Format in die Liste der Verschattungsobjekte [▶ 243] importieren. Zusätzlich werden die importierten Daten auf Plausibilität überprüft und Fehler in eine Log-Datei geschrieben.



VAR_INPUT

```
bStart    : BOOL;
sDataFile : STRING;
sLogFile  : STRING;
sNetId    : STRING;
```

bStart: Eine TRUE-Flanke an diesem Eingang startet den Lesevorgang.

sDataFile: Enthält den Pfad- und Dateinamen der zu öffnenden Daten-Datei. Diese muss in Excel als Dateityp "CSV (Trennzeichen-getrennt) (*.csv)" abgespeichert worden sein. Öffnet man die Datei mit einem einfachen Text-Editor, so müssen die Werte durch Semikolons getrennt dargestellt sein. Beispiel für einen Eintrag: *sDataFile:= 'C:\Projekte\ShadingObjects.csv'*

sNetId: Hier kann ein String mit der AMS Net Id des TwinCAT-Rechners angegeben werden, auf dem die Dateien geschrieben/gelesen werden sollen. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

> Die Speicherung der Daten kann nur auf dem Steuerungsrechner selbst und den Rechnern erfolgen, die mit dem Steuerungsrechner per ADS verbunden sind. Verweise auf lokale Festplatten dieser Rechner sind möglich, nicht jedoch auf verbundene Netzwerkfestplatten.

VAR_OUTPUT

```
bBusy             : BOOL;
bError            : BOOL;
udiErrorId        : UDINT;
uiAmountOfSetsRead: UINT;
```

bBusy: Dieser Ausgang steht auf TRUE, solange Elemente aus der Datei ausgelesen werden.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind oder wenn ein Datei Schreib- bzw. Lesefehler aufgetreten ist.

udiErrorId: Enthält den Fehlercode des zuletzt aufgetretenen Fehlers. Siehe Fehlercodes [▶ 243] bzw. ADS_Fehlercodes.

uiAmountOfSetsRead: Anzahl der gelesenen Datensätze.

VAR_IN_OUT

```
arrShadingObject : ARRAY[1..iShadingObjects] OF ST_BARShadingObject;
```

arrShadingObject: Liste der Verschattungsobjekte [▶ 243].

Excel-Tabelle

Das folgende Beispiel zeigt die Excel-Tabelle mit den Einträgen der Fensterelemente.
Alle Textfelder sind frei beschreibbar, wichtig sind die grün markierten Felder, wobei dort jede Zeile einen Datensatz kennzeichnet. Je nachdem ob es sich um den Typ Viereck oder Kugel handelt, haben die Spalten G bis J eine unterschiedliche Bedeutung. Die Spalten K bis M sind bei Kugeln freizulassen. Bezüglich der

Vierecks-Koordinaten werden nur die relevanten Daten eingegeben und die restlichen intern berechnet, siehe FB_BARShadingObjectsEntry [▶ 186].
Folgende Regeln sind zu beachten:

- Ein Datensatz muss immer mit einem '@' beginnen.
- Die Monatseinträge dürfen nicht 0 und größer 12 sein, andernfalls sind alle Kombinationen möglich.
  Beispiele:
  Beginn=1, Ende=1: Verschattung im Januar.
  Beginn=1, Ende=5: Verschattung von Anfang Januar bis Ende Mai.
  Beginn=11, Ende=5: Verschattung von Anfang November bis Ende Mai (des folgenden Jahres).
- Fensterbreite und Fensterhöhe müssen größer Null sein
- Die z-Koordinaten P1z und P3z bzw. Mz müssen größer Null sein.
- Der Radius muss größer Null sein.
- Die Gesamtgröße der Tabelle darf systembedingt 65534 Bytes nicht überschreiten.
- Die Tabelle muss in Excel als Dateityp "CSV (Trennzeichen-getrennt) (*.csv)" abgespeichert worden sein.

Es ist nicht notwendig, alle Verschattungsobjekte, welche pro Fassade möglich sind, zu beschreiben. Nur die, die in der Liste aufgeführt werden, kommen letztendlich zum Tragen.

DE_ShadingObjects.csv

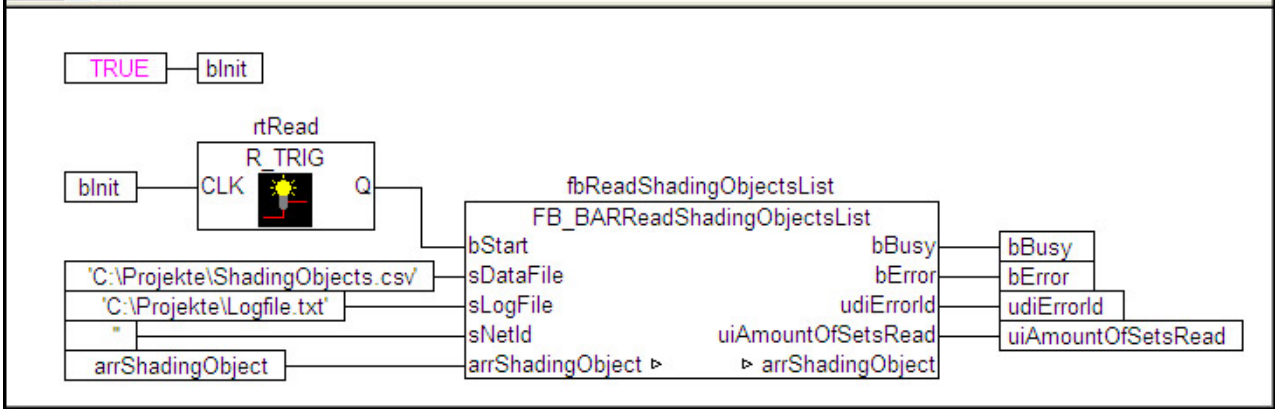
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Nummer	Beschreibung		Typ	Beginn	Ende	P1x/Mx	P1y/My	P1z/Mz	P2y/R	P3x	P3y	P3z
2				0 - Viereck	(Monat)	(Monat)	[m]	[m]	[m]	[m]	[m]	[m]	[m]
3				1 - Kugel									
4		Text											
5	1	Beschreibung	@	0	1	2	-94,75	0	36,06	11	-70,71	11	68,59
6	2	Beschreibung	@	0	1	2	-23,33	0	9,9	10,5	-3,54	10,5	22,62
7	3	Beschreibung	@	0	1	2	62,23	0	0	14,47	62,23	14,47	8
8	4	Beschreibung	@	0	1	2	46	0	13	14,47	62,23	14,47	8
9	5	Beschreibung	@	0	1	2	46	0	13	14,47	46	14,47	38,89
10	6	Beschreibung	@	0	1	2	0	0	14	9	35	9	14
11	7	Beschreibung	@	0	1	2	0	0	14	9,8	16	9,8	14
12	8	Beschreibung	@	0	1	2	23,6	0	14	9,8	25	9,8	14
13	9	Beschreibung	@	0	1	2	27,8	0	14	9,8	35	9,8	14
14													
15	10	Beschreibung	@	1	1	2	27	15	40	6			
16	11	Beschreibung	@	1	1	2	38	15	36	6			
17	12	Beschreibung	@	1	1	2	-14	4	4	1,5			
18	13	Beschreibung	@	1	1	2	-6,5	6	6	3,2			
19	14	Beschreibung	@	1	1	2	-7	9	6	1,2			
20	15	Beschreibung	@	1	1	2	-1	6	8	3,2			
21	16	Beschreibung	@	1	1	2	-1	9	8	1,2			

Log-Datei

Bei jedem Start des Lesebausteines wird die Logdatei neu beschrieben und der alte Inhalt gelöscht. Ist die Log-Datei nicht vorhanden, so wird sie zunächst automatisch erstellt. Die Log-Datei enthält dann entweder eine OK Meldung oder eine Auflistung aller aufgetretenen Fehler. Fehler, welche mit dem Öffnen, Beschreiben oder Schließen der Logdatei selbst in Zusammenhang stehen, können jedoch nicht mitgeschrieben werden. Daher ist immer auch der Ausgang *udiErrorId* des Lesebausteines zu beachten, welcher den letzten Fehlercode anzeigt. Da im Ablauf des Lesens die Log-Datei immer als Letztes geschlossen wird, ist ein entsprechender Hinweis im Fehlerfall gewährleistet.

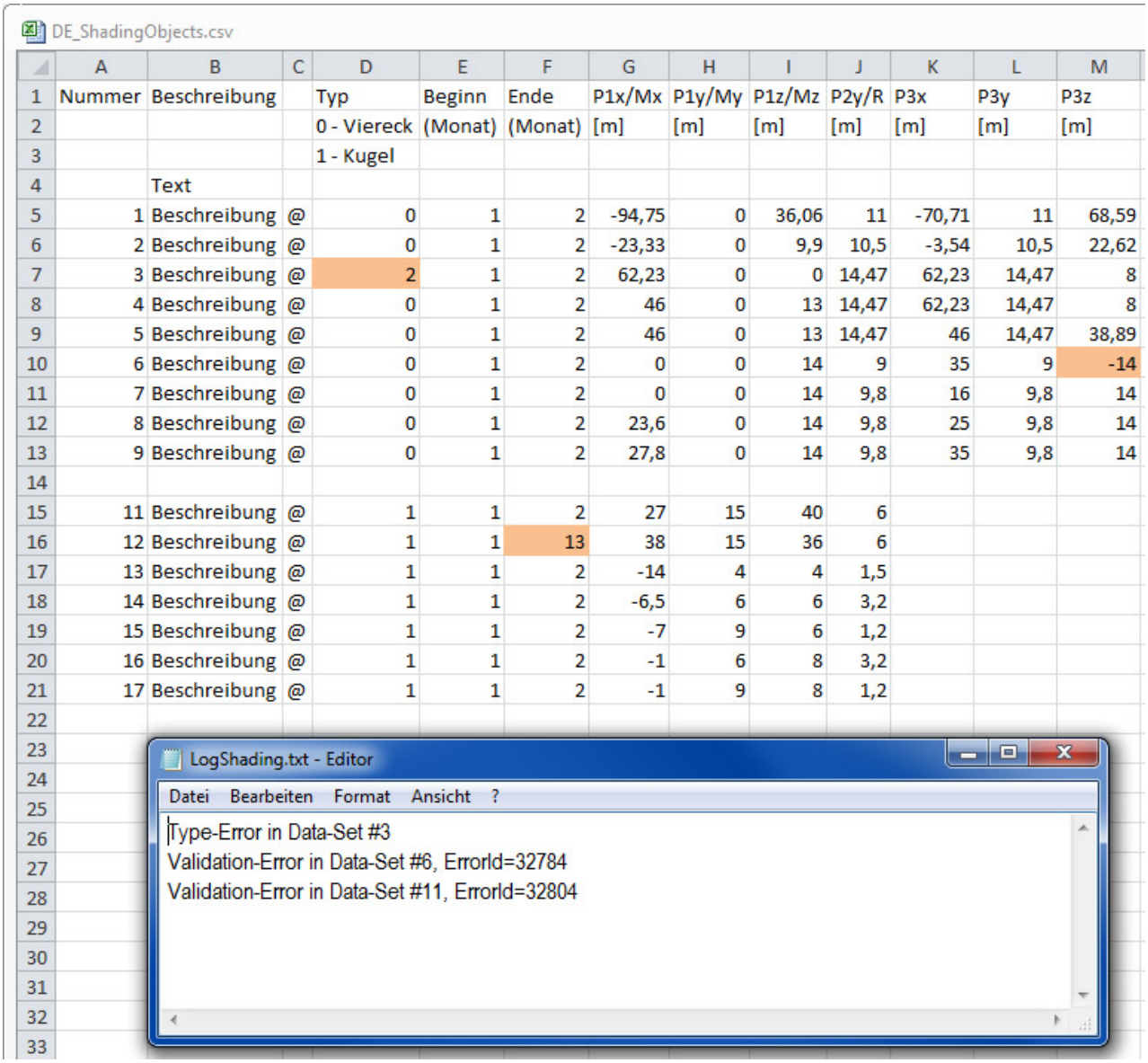
Programmbeispiel

```
0001 PROGRAM ReadShadingObjects
0002 VAR
0003     bInit                      : BOOL;
0004     rtRead                     : R_TRIG;
0005     fbReadShadingObjectsList   : FB_BARReadShadingObjectsList;
0006     arrShadingObject           : ARRAY[1..iShadingObjects] OF ST_BARShadingObject;
0007
0008     bBusy                      : BOOL;
0009     bError                     : BOOL;
0010     udiErrorId                 : UDINT;
0011     uiAmountOfSetsRead         : UINT;
0012 END_VAR
```



In diesem Beispiel wird bei PLC-Start die Variable *bInit* zunächst auf TRUE gesetzt. Damit erhält der Eingang *bStart* am Baustein fbReadShadingObjectsList einmalig eine steigende Flanke, welche den Lesevorgang auslöst. Gelesen wird die Datei "ShadingObjects.csv", welche sich im Ordner "C:\Projekte\" befindet. In demselben Ordner wird dann die Logdatei "Logfile.txt" hinterlegt. Ist diese Logdatei noch nicht vorhanden wird sie erstellt, andernfalls wird der bestehende Inhalt überschrieben. Das Lesen und Schreiben erfolgt auf demselben Rechner, auf dem sich auch die PLC befindet. Das wird durch den Eingang *sNetID* = '' (=lokal) definiert. Alle Daten werden in die im Programm deklarierte Liste *arrShadingObject* geschrieben. Solange gelesen und geschrieben wird ist der Ausgang *bBusy* auf TRUE. Der zuletzt auftretende Fehler wird an *udiErrorId* angezeigt, *bError* steht dann auf TRUE. Die Anzahl der gefundenen und gelesenen Datenzeilen wird an *uiAmountOfSetsRead* zur Kontrolle angezeigt.

In die folgende Excel Liste wurden die markierten Fehler eingebaut. Dadurch ergibt sich das gezeigte Log-File:



DE_ShadingObjects.csv

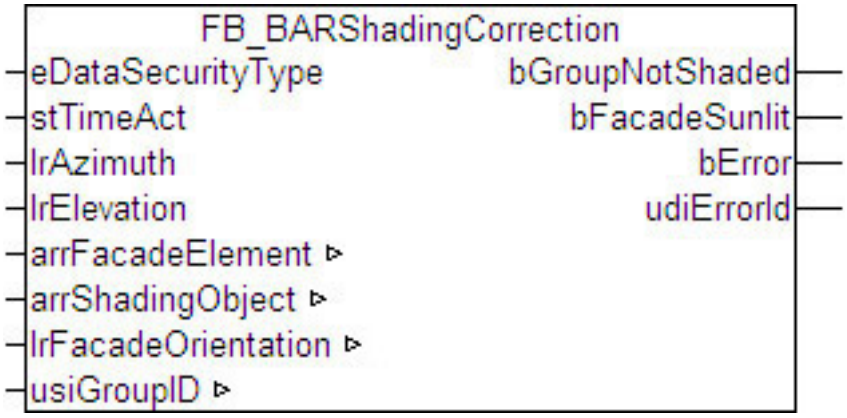
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Nummer	Beschreibung		Typ	Beginn	Ende	P1x/Mx	P1y/My	P1z/Mz	P2y/R	P3x	P3y	P3z
2				0 - Viereck	(Monat)	(Monat)	[m]	[m]	[m]	[m]	[m]	[m]	[m]
3				1 - Kugel									
4		Text											
5	1	Beschreibung	@	0	1	2	-94,75	0	36,06	11	-70,71	11	68,59
6	2	Beschreibung	@	0	1	2	-23,33	0	9,9	10,5	-3,54	10,5	22,62
7	3	Beschreibung	@	2	1	2	62,23	0	0	14,47	62,23	14,47	8
8	4	Beschreibung	@	0	1	2	46	0	13	14,47	62,23	14,47	8
9	5	Beschreibung	@	0	1	2	46	0	13	14,47	46	14,47	38,89
10	6	Beschreibung	@	0	1	2	0	0	14	9	35	9	-14
11	7	Beschreibung	@	0	1	2	0	0	14	9,8	16	9,8	14
12	8	Beschreibung	@	0	1	2	23,6	0	14	9,8	25	9,8	14
13	9	Beschreibung	@	0	1	2	27,8	0	14	9,8	35	9,8	14
14													
15	11	Beschreibung	@	1	1	2	27	15	40	6			
16	12	Beschreibung	@	1	1	13	38	15	36	6			
17	13	Beschreibung	@	1	1	2	-14	4	4	1,5			
18	14	Beschreibung	@	1	1	2	-6,5	6	6	3,2			
19	15	Beschreibung	@	1	1	2	-7	9	6	1,2			
20	16	Beschreibung	@	1	1	2	-1	6	8	3,2			
21	17	Beschreibung	@	1	1	2	-1	9	8	1,2			
22													
23													
24													
25													
26													
27													
28													
29													
30													
31													
32													
33													

Der erste Fehler ist im Datensatz 3 und ein Typ-Fehler, da "2" nicht definiert ist.
Der nächste Fehler in Datensatz 6 wurde nach Validierung der Daten mit dem intern genutzten Baustein FB_BARShadingObjectsEntry [▶ 186] gefunden und daher näher mit einer Fehlernummer versehen, welche unter Fehlercodes [▶ 243] näher aufgeschlüsselt wird. Der dritte Fehler trat ebenfalls nach der internen Validierung auf. Wichtig hierbei ist, dass sich die Datensatz-Nummer (hier 11) nicht nach der eingetragenen Nummer in der Liste, sondern nach der tatsächlichen laufenden Nummer richtet: eingelesen wurden hier lediglich 16 Datensätze.

3.3.4.10 FB_BARShadingCorrection

Funktionsbaustein zur Verschattungsbeurteilung einer Fenstergruppe auf einer Fassade.

Dieser Baustein ist nur für die Nordhalbkugel gültig. Der für die Südhalbkugel gültige Baustein ist FB_BARShadingCorrectionSouth [▶ 183].



Der Baustein FB_BARShadingCorrection berechnet für eine Gruppe von Fenstern, ob sich diese im Schatten von umliegenden Objekten befindet. Mit dem Ergebnis, welches am Ausgang *bGroupNotShaded* ausgegeben wird, kann beurteilt werden, ob ein Sonnenschutz für diese Fenstergruppe sinnvoll ist.
Dabei greift der Baustein auf zwei zu definierende Listen zu:

- Die Daten der Elemente (Fenster) der Fassade, in der sich die zu betrachtende Gruppe befindet. Auf diese Liste der Fassadenelemente [▶ 243] wird über die IN-OUT-Variable *arrFacadeElement* zugegriffen, welche ihrerseits global definiert ist.
- Die Parameter, welche die Verschattungselemente beschreiben, die für die Fassade maßgeblich sind, auf der sich die Fenstergruppe befindet.. Diese Liste der Verschattungsobjekte [▶ 243] ist ebenfalls global definiert. Die IN-OUT-Variable *arrShadingObject* greift direkt darauf zu.

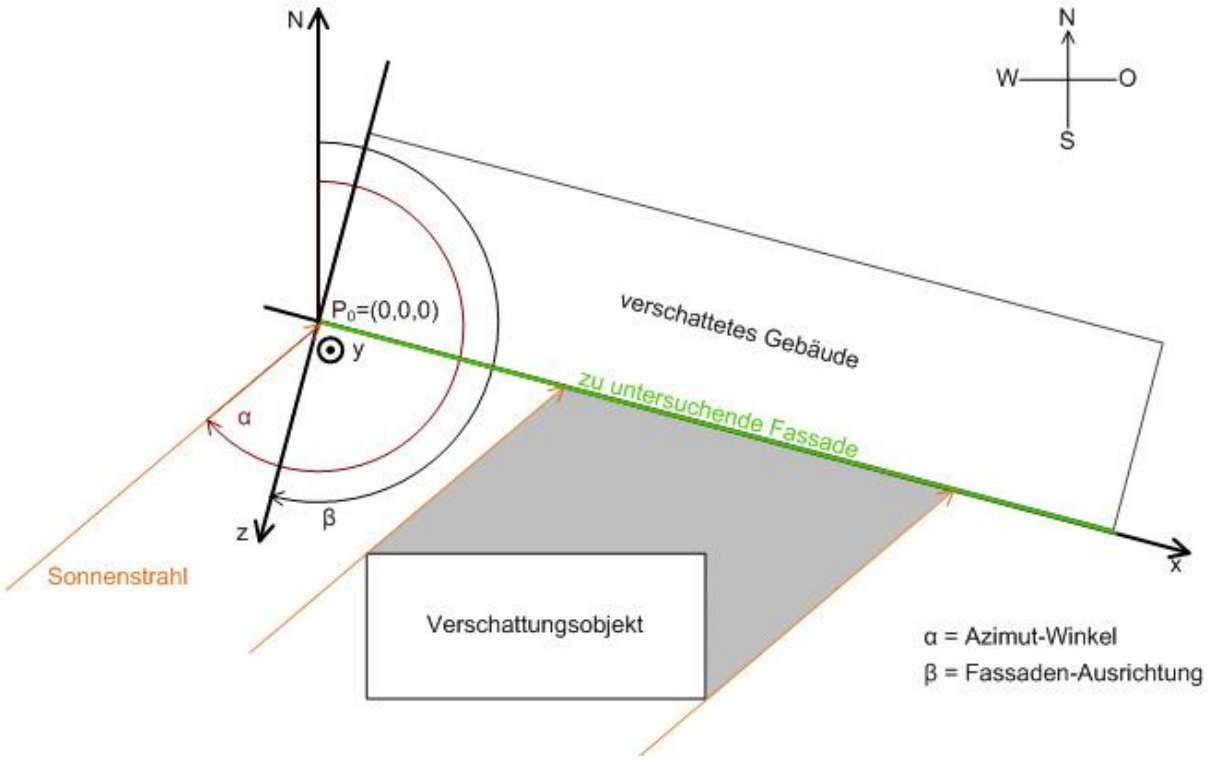
Anhand der Fassadenausrichtung (*lrFacadeOrientation*), der Sonnenrichtung (*lrAzimuth*) und der Sonnenhöhe (*lrElevation*) kann dann für jeden Eckpunkt eines Fensters errechnet werden, ob sich dieser im Bereich eines Schattens befindet. Eine Gruppe von Fenstern gilt dann als komplett verschattet, wenn alle Eckpunkte verschattet sind.

Dabei gilt auf der Nordhalbkugel für die Fassadenausrichtung (Blick aus dem Fenster):

Blickrichtung	Fassadenausrichtung
Nord	β=0°
Ost	β=90°
Süd	β=180°
West	β=270°

Der Baustein führt seine Berechnungen nur dann durch, wenn die Sonne tatsächlich auf die Fassade scheint. Betrachtet man die in der Einleitung vorgestellte Zeichnung, so ist dies gegeben wenn gilt:

Fassadenausrichtung < Azimutwinkel < Fassadenausrichtung + 180°



Darüber hinaus ist eine Berechnung auch dann nicht nötig, wenn die Sonne noch nicht aufgegangen ist, die Sonnenhöhe (Elevation) also unter 0° liegt. In beiden Fällen wird der Ausgang *bFacadeSunlit* auf FALSE gesetzt.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
stTimeAct        : TIMESTRUCT;
lrAzimuth        : LREAL;
lrElevation      : LREAL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Andernfalls wird der instanziierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

stTimeAct: Eingabe der aktuellen Uhrzeit in GMT (Greenwich-Mean-Time).

rAzimuth: Sonnenrichtung zum Betrachtungszeitpunkt in Grad.

rElevation: Sonnenhöhe zum Betrachtungszeitpunkt in Grad.

VAR_OUTPUT

```
bGroupNotShaded: BOOL;
bFacadeSunlit  : BOOL;
bError         : BOOL;
udiErrorId     : UDINT;
```

bGroupNotShaded : Steht auf TRUE, solange die Fenstergruppe als nicht verschattet errechnet wird.

bFacadeSunlit: Dieser Ausgang ist dann auf TRUE gesetzt, wenn die Sonne auf die Fassade scheint. Siehe Beschreibung oben.

bError : Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung des Bausteines ein Fehler erkannt wird.

udiErrorId : Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

VAR_IN_OUT

Damit die eingetragenen Parameter über einen Steuerungsausfall hinweg erhalten bleiben ist es erforderlich, sie als In-Out-Variablen zu deklarieren. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variablen wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

```
arrFacadeElement     : ARRAY[1..iColumnsPerFacade, 1..iRowsPerFacade] OF ST_BARFacadeElement;
arrShadingObject     : ARRAY[1..iShadingObjects] OF ST_BARShadingObject;
lrFacadeOrientation  : LREAL;
usiGroupID           : USINT;
```

arrFacadeElement: Liste der Fassadenelemente [▶ 243].

arrShadingObject: Liste der Verschattungsobjekte [▶ 243].

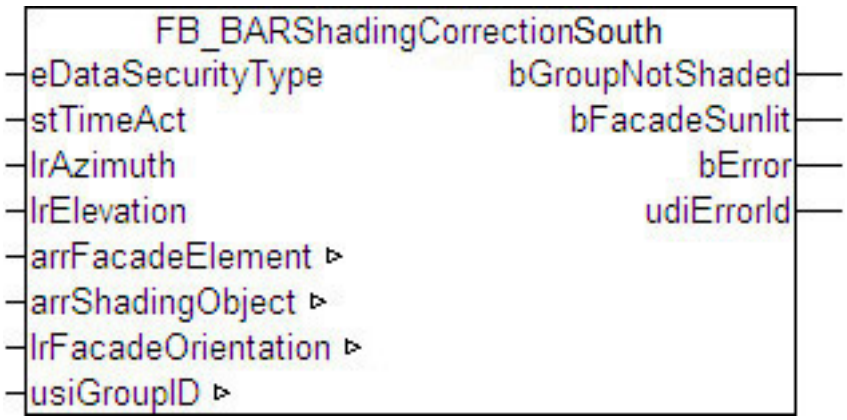
lrFacadeOrientation: Fassadenausrichtung, siehe Abbildung oben.

usiGroupId: Betrachtete Fenstergruppe. Die Gruppe 0 ist hierbei für nicht verwendete Fensterelemente reserviert, siehe FB_BARFacadeElementEntry [▶ 168]. Ein 0-Eintrag würde zu einer Fehlerausgabe führen (bError=TRUE). Der Baustein wird dann nicht weiter abgearbeitet und *bGroupNotShaded* auf FALSE gesetzt.

3.3.4.11 FB_BARShadingCorrectionSouth

Funktionsbaustein zur Verschattungsbeurteilung einer Fenstergruppe auf einer Fassade.

Dieser Baustein ist nur für die Südhalbkugel gültig. Der für die Nordhalbkugel gültige Baustein ist FB_BARShadingCorrection [▶ 180].



Der Baustein FB_BARShadingCorrectionSouth berechnet für eine Gruppe von Fenstern, ob sich diese im Schatten von umliegenden Objekten befindet. Mit dem Ergebnis, welches am Ausgang *bGroupNotShaded* ausgegeben wird, kann beurteilt werden, ob ein Sonnenschutz für diese Fenstergruppe sinnvoll ist.
Dabei greift der Baustein auf zwei zu definierende Listen zu:

- Die Daten der Elemente (Fenster) der Fassade, in der sich die zu betrachtende Gruppe befindet. Auf diese Liste der Fassadenelemente [▶ 243] wird über die IN-OUT-Variable *arrFacadeElement* zugegriffen, welche ihrerseits global definiert ist.
- Die Parameter, welche die Verschattungselemente beschreiben, die für die Fassade maßgeblich sind, auf der sich die Fenstergruppe befindet.. Diese Liste der Verschattungsobjekte [▶ 243] ist ebenfalls global definiert. Die IN-OUT-Variable *arrShadingObject* greift direkt darauf zu.

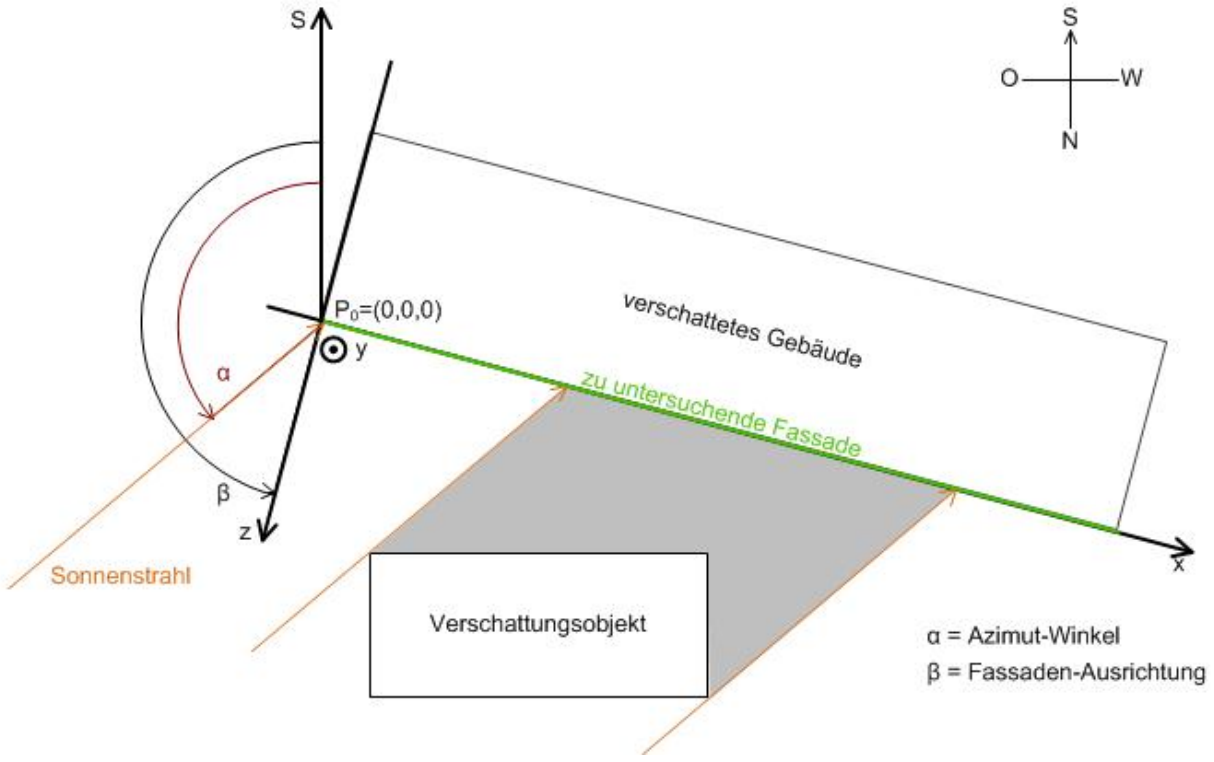
Anhand der Fassadenausrichtung (*lrFacadeOrientation*), der Sonnenrichtung (*lrAzimuth*) und der Sonnenhöhe (*lrElevation*) kann dann für jeden Eckpunkt eines Fensters errechnet werden, ob sich dieser im Bereich eines Schattens befindet. Eine Gruppe von Fenstern gilt dann als komplett verschattet, wenn alle Eckpunkte verschattet sind.

Dabei gilt auf der Südhalbkugel für die Fassadenausrichtung (Blick aus dem Fenster):

Blickrichtung	Fassadenausrichtung
Süd	β=0°
Ost	β=90°
Nord	β=180°
West	β=270°

Der Baustein führt seine Berechnungen nur dann durch, wenn die Sonne tatsächlich auf die Fassade scheint. Betrachtet man die in der Einleitung vorgestellte Zeichnung, so ist dies gegeben wenn gilt:

Fassadenausrichtung < Azimutwinkel < Fassadenausrichtung + 180°



Darüber hinaus ist eine Berechnung auch dann nicht nötig, wenn die Sonne noch nicht aufgegangen ist, die Sonnenhöhe (Elevation) also unter 0° liegt. In beiden Fällen wird der Ausgang *bFacadeSunlit* auf FALSE gesetzt.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
stTimeAct        : TIMESTRUCT;
lrAzimuth        : LREAL;
lrElevation      : LREAL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Andernfalls wird der instanziierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

stTimeAct: Eingabe der aktuellen Uhrzeit in GMT (Greenwich-Mean-Time).

rAzimuth: Sonnenrichtung zum Betrachtungszeitpunkt in Grad.

rElevation: Sonnenhöhe zum Betrachtungszeitpunkt in Grad.

VAR_OUTPUT

```
bGroupNotShaded: BOOL;
bFacadeSunlit  : BOOL;
bError         : BOOL;
udiErrorId     : UDINT;
```

bGroupNotShaded : Steht auf TRUE, solange die Fenstergruppe als nicht verschattet errechnet wird.

bFacadeSunlit: Dieser Ausgang ist dann auf TRUE gesetzt, wenn die Sonne auf die Fassade scheint. Siehe Beschreibung oben.

bError : Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung des Bausteines ein Fehler erkannt wird.

udiErrorId : Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

VAR_IN_OUT

Damit die eingetragenen Parameter über einen Steuerungsausfall hinweg erhalten bleiben ist es erforderlich, sie als In-Out-Variablen zu deklarieren. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variablen wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

```
arrFacadeElement     : ARRAY[1..iColumnsPerFacade, 1..iRowsPerFacade] OF ST_BARFacadeElement;
arrShadingObject     : ARRAY[1..iShadingObjects] OF ST_BARShadingObject;
lrFacadeOrientation  : LREAL;
usiGroupID           : USINT;
```

arrFacadeElement: Liste der Fassadenelemente [▶ 243].

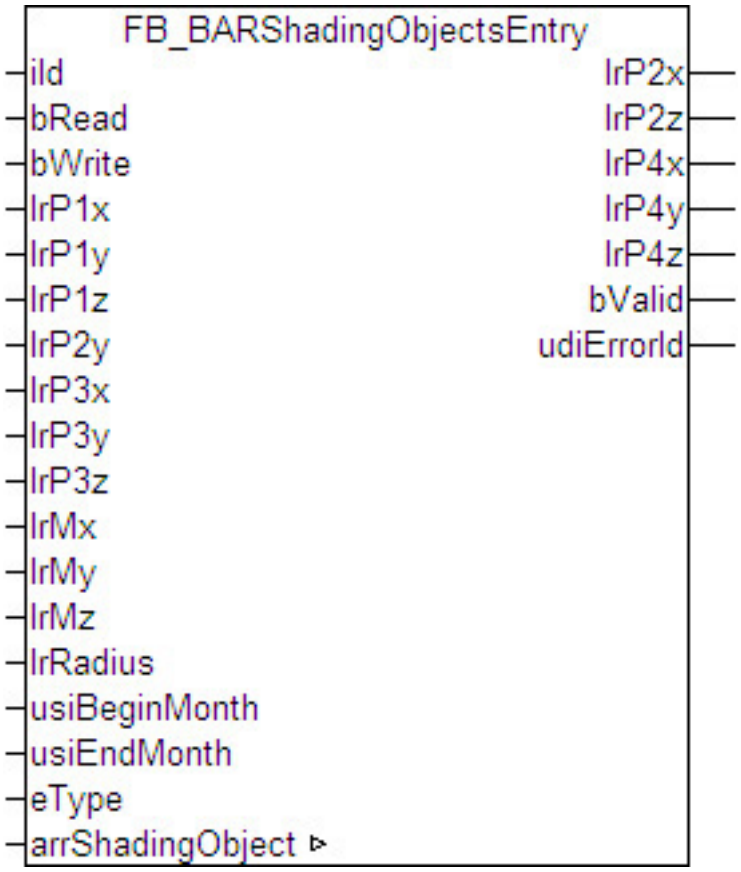
arrShadingObject: Liste der Verschattungsobjekte [▶ 243].

lrFacadeOrientation: Fassadenausrichtung, siehe Abbildung oben.

usiGroupId: Betrachtete Fenstergruppe. Die Gruppe 0 ist hierbei für nicht verwendete Fensterelemente reserviert, siehe FB_BARFacadeElementEntry [▶ 168]. Ein 0-Eintrag würde zu einer Fehlerausgabe führen (bError=TRUE). Der Baustein wird dann nicht weiter abgearbeitet und *bGroupNotShaded* auf FALSE gesetzt.

3.3.4.12 FB_BARShadingObjectsEntry

Dieser Baustein dient zur Verwaltung aller Verschattungselemente einer Fassade, welche global in einer Liste von Verschattungselementen [▶ 243] hinterlegt ist. Er soll die Eingabe der Elementinformationen - auch im Hinblick auf die Nutzung der Target-Visualisierung - erleichtern. Eine schematische Darstellung der Objekte mit Beschreibung der Koordinaten ist unter Verschattungskorrektur: Grundlagen und Definitionen [▶ 156] gegeben.



Die Deklaration der Verschattungselemente erfolgt in der globalen Variable:

```
VAR_GLOBAL
     arrShadingObject : ARRAY[1..iShadingObjects] OF ST_BARShadingObject;
END_VAR
```

Jedes einzelne Element *arrShadingObject[1]* bis *arrShadingObject[iShadingObjects]* trägt die Informationen für jeweils ein Verschattungselement (ST_BARShadingObject [▶ 241]). Dies sind neben dem gewählten Verschattungstyp (Viereck oder Kugel) die jeweils dazugehörigen Koordinaten. Im Falle eines Vierecks sind es die Eckpunkte *(lrP1x*, *lrP1y*, *lrP1z), (lrP2x*, *lrP2y*, *lrP2z),(lrP3x*, *lrP3y*, *lrP3z)* und *(lrP4x*, *lrP4y*, *lrP4z)* und im Falle einer Kugel der Mittelpunkt *(lrMx*, *lrMy*, *lrMz)* und der Radius *lrRadius*. Zusätzlich kann über die Eingänge *usiBeginMonth* und *usiEndMonth* die Phase der Verschattung definiert werden, was bei Objekten, wie z.B. Bäume, die im Winter kein Laub tragen, wichtig ist.

Der Baustein greift dabei über die IN-OUT-Variable *arrShadingObject* direkt auf das Feld dieser Informationen zu.

Die Tatsache, dass die Viereckskoordinaten lrP2x, lrP2z, lrP4x, lrP4y, und lrP4z Ausgangswerte sind, ergibt sich daraus, dass sie aus den Eingabeparametern gebildet werden.

lrP2x = lrP1x; *lrP2z = lrP1z*; *lrP4x = lrP3x*; *lrP4y = lrP1y*; *lrP4z = lrP3z*;

Das schränkt die Eingabe eines Vierecks so weit ein, dass die seitlichen Kanten senkrecht auf dem Boden stehen (*lrP2x = lrP1x* und *lrP4x = lrP3x*), dass das Viereck keine Neigung hat (*lrP2z = lrP1z* und *lrP4z = lrP3z*) und nur nach "oben" also in positive y-Richtung eine unterschiedliche Höhe haben kann (*lrP4y = lrP1y*).

Die Verwendung des Bausteines erfolgt in drei Schritten:

- Auslesen
- Ändern
- Schreiben

Auslesen

Mit dem Eintrag an *iId* wird das entsprechende Element aus der Liste, *arrShadingObject[iId]*, ausgewählt. Eine steigende Flanke an *bRead* liest die Daten aus. Diese Werte werden den Ein- und Ausgangsvariablen des Bausteines zugewiesen. Es handelt sich hierbei um die Eingangswerte *lrP1x*, *lrP1y*, *lrP1z*, *lrP2y*, *lrP3x*, *lrP3y*, *lrP3z*, *lrMx*, *lrMy*, *lrMz*, *rRadius* und den Objekt-Enumerator *eType* und die Ausgangswerte *lrP2x*, *lrP2z*, *lrP4x*, *lrP4y*, und *lrP4z.* Wichtig ist hierbei, dass im Schritt des Auslesens die Eingangswerte nicht überschrieben werden. So lassen sich alle Werte zunächst in einer Visualisierung anzeigen.

Ändern

In einem nächsten Programmschritt können dann die aufgeführten Eingabewerte verändert werden. Sollte am EingangeType [▶ 240] über den Wert *"eObjectTypeTetragon"* die Verwendung eines Vierecks vorgewählt sein, ergeben sich die Ausgangswerte *rP2x, rP2z, rP4x, rP4y*, und *rP4z* aus den eingegebenen Koordinaten des Vierecks, siehe oben.

Die eingegebenen Werte werden ständig auf Plausibilität überprüft. Der Ausgang *bValid* zeigt an, ob die Werte gültig sind (*bValid*=TRUE). Sollte dies nicht der Fall sein, wird am Ausgang *udiErrorId* ein entsprechender Fehlercode [▶ 243] ausgegeben.
Wird ein Viereck definiert, müssen lediglich die Eingänge *lrP1x*, *lrP1y*, *lrP1z*, *lrP2y*, *lrP3x*, *lrP3y* und *lrP3z* beschrieben werden, die Eingänge *lrMx*, *lrMy*, *lrMz* und *lrRadius* brauchen nicht verknüpft zu werden. Im Falle einer Kugeldefinition müssen nur *lrMx*, *lrMy*, *lrMz* und *lrRadius* beschrieben werden und die Viereckkoordinaten können unverknüpft bleiben

Mit einer positiven Flanke an *bWrite* werden die parametrierten Daten in das Listenelement mit dem Index *iId* geschrieben, unabhängig davon, ob sie gültige Werte darstellen oder nicht. Daher ist innerhalb der Elementstruktur ST_BARShadingObject [▶ 241] ebenfalls ein Plausibilitätsbit *bValid* vorhanden, das genau diese Information an den Baustein FB_BARShadingCorrection [▶ 180] / FB_BARShadingCorrectionSouth [▶ 183] weiterreicht und dort Fehlberechnungen vorbeugt.

Diese Vorgehensweise soll nur als Vorschlag angesehen werden. Es ist natürlich auch möglich, den Baustein ganz normal in einem Schritt zu parametrieren und die eingetragenen Werte mit einer steigenden Flanke an *bWrite* in das entsprechende Listenelement zu schreiben.

VAR_INPUT

```
iId          : INT;
bRead        : BOOL;
bWrite       : BOOL;
lrP1x        : LREAL;
lrP1y        : LREAL;
lrP1z        : LREAL;
lrP2y        : LREAL;
lrP3x        : LREAL;
```

```
lrP3y        : LREAL;
lrP3z        : LREAL;
lrMx         : LREAL;
lrMy         : LREAL;
lrMz         : LREAL;
lrRadius     : LREAL;
usiBeginMonth: USINT;
usiEndMonth  : USINT;
eType        : E_BARShadingObjectType;
```

iId: Index des gewählten Elementes. Dies bezieht sich auf die Auswahl eines Feldelementes des unter der IN-OUT-Variablen *arrShadingObject* angelegten Arrays. **iId darf nicht Null sein!** Das ergibt sich aus der Felddefinition [▶ 186], *arrShadingObject : ARRAY[1..iShadingObjects] OF ST_BARShadingObject;*

bRead: Mit einer positiven Flanke an diesem Eingang werden die Informationen des gewählten Elementes, *arrShadingObject[iId]* in den Baustein gelesen und den Eingangsvariablen *lrP1x* bis *eType* sowie den Ausgangsvariablen *lrP2x* bis *lrP4z* zugewiesen. Sind zu diesem Zeitpunkt schon Daten an den Eingängen *lrP1x* bis *eType* angelegt, so werden die zuvor ausgelesenen Daten sofort mit diesen überschrieben.

bWrite: Eine positive Flanke schreibt die an den Eingängen *rP1x* bis *eType* angelegten sowie die ermittelten und den Ausgängen *lrP2x* bis *lrP4z* zugewiesenen Werte in das gewählte Feldelement *arrShadingObject[iId].*

lrP1x: X-Koordinate des Punktes 1 des Verschattungselementes (Viereck) in Meter.

lrP1y: Y-Koordinate des Punktes 1 des Verschattungselementes (Viereck) in Meter.

lrP1z: Z-Koordinate des Punktes 1 des Verschattungselementes (Viereck) in Meter.

lrP2y: Y-Koordinate des Punktes 2 des Verschattungselementes (Viereck) in Meter.

lrP3x: X-Koordinate des Punktes 3 des Verschattungselementes (Viereck) in Meter.

lrP3y: Y-Koordinate des Punktes 3 des Verschattungselementes (Viereck) in Meter.

lrP3z: Z-Koordinate des Punktes 3 des Verschattungselementes (Viereck) in Meter.

lrMx: X-Koordinate des Mittelpunktes des Verschattungselementes (Kugel) in Meter.

lrMy: Y-Koordinate des Mittelpunktes des Verschattungselementes (Kugel) in Meter.

lrMz: Z-Koordinate des Mittelpunktes des Verschattungselementes (Kugel) in Meter.

lrRadius: Radius des Verschattungselementes (Kugel) in Meter.

usiBeginMonth: Anfang der Verschattungsperiode (Monatszahl).

usiEndMonth: Ende der Verschattungsperiode (Monatszahl).

eType: Gewählter Elementtyp: Viereck oder Kugel. Siehe E_BARShadingObjectType [▶ 240].

Bemerkung zur Verschattungsperiode:

Die Monatseinträge dürfen nicht 0 und größer 12 sein, andernfalls sind alle Kombinationen möglich.
Beispiele:
Beginn=1, Ende=1: Verschattung im Januar.
Beginn=1, Ende=5: Verschattung von Anfang Januar bis Ende Mai.
Beginn=11, Ende=5: Verschattung von Anfang November bis Ende Mai (des folgenden Jahres).

VAR_OUTPUT

```
lrP2x      : LREAL;
lrP2z      : LREAL;
lrP4x      : LREAL;
lrP4y      : LREAL;
lrP4z      : LREAL;
bValid     : BOOL;
udiErrorId : UDINT;
```

lrP2x: Ermittelte X-Koordinate des Punktes 2 des Verschattungselementes (Viereck) in Meter. Siehe "Info" oben.

lrP2z: Ermittelte Z-Koordinate des Punktes 2 des Verschattungselementes (Viereck) in Meter. Siehe " Info " oben.

lrP4x: Ermittelte X-Koordinate des Punktes 4 des Verschattungselementes (Viereck) in Meter. Siehe " Info " oben.

lrP4y: Ermittelte Y-Koordinate des Punktes 4 des Verschattungselementes (Viereck) in Meter. Siehe " Info " oben.

lrP4z: Ermittelte Z-Koordinate des Punktes 4 des Verschattungselementes (Viereck) in Meter. Siehe " Info " oben.

bValid: Ergebnis der Plausibilitätskontrolle für die eingegebenen Werte. In Bezug auf ein Viereck ist verlangt, dass der Innenwinkel 360° beträgt und die Punkte in einer Ebene liegen und *vor* der betrachteten Fassade liegen. Bei einer Kugel muss der Mittelpunkt ebenfalls vor der Fassade liegen und der Radius muss größer Null sein.

udiErrorId: Enthält den Fehlercode, sollten die eingetragenen Werte nicht den erwähnten Kriterien entsprechen. Siehe Fehlercodes [▶ 243].

VAR_IN_OUT

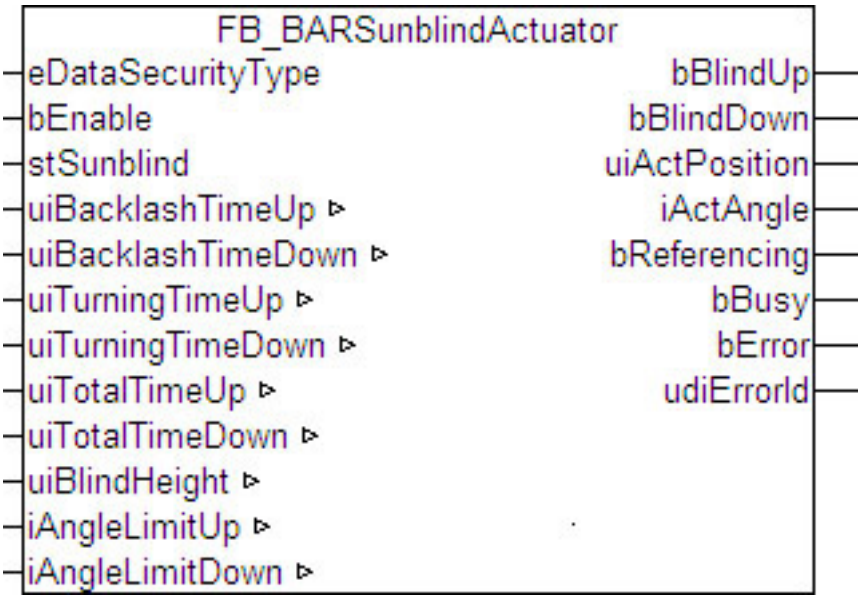
```
arrShadingObject : ARRAY[1..iShadingObjects] OF ST_BARShadingObject;
```

arrShadingObject: Liste der Verschattungsobjekte [▶ 243]. Werte werden persistent gespeichert.

3.3.4.13 FB_BARSunblindActuator

Dieser Baustein dient zur Positionierung einer Lamellen-Jalousie über zwei Ausgänge: hoch- und herunterfahren. Über das Positioniertelegramm stSunblind [▶ 242] kann die Jalousie auf eine beliebige (Höhen-) Position und einen Lamellenwinkel gefahren werden. Darüber hinaus beinhaltet das Positioniertelegramm stSunblind [▶ 242] auch Handbefehle, mit denen die Jalousie individuell auf bestimmte Stellungen bewegt werden kann. Diese Handbefehle werden von dem Baustein FB_BARSunblindSwitch [▶ 211] angesteuert.

Die gesamt-Verfahrzeit in Millisekunden ist hier durch das UINT-Format auf 65535 ms beschränkt. Für längere Verfahrzeiten ist der Baustein FB_BARSunblindActuatorEx [▶ 194] zu verwenden, der diese Zeiten im UDINT-Format darstellt(*udiTotalTimeUp/udiTotalTimeDown*).



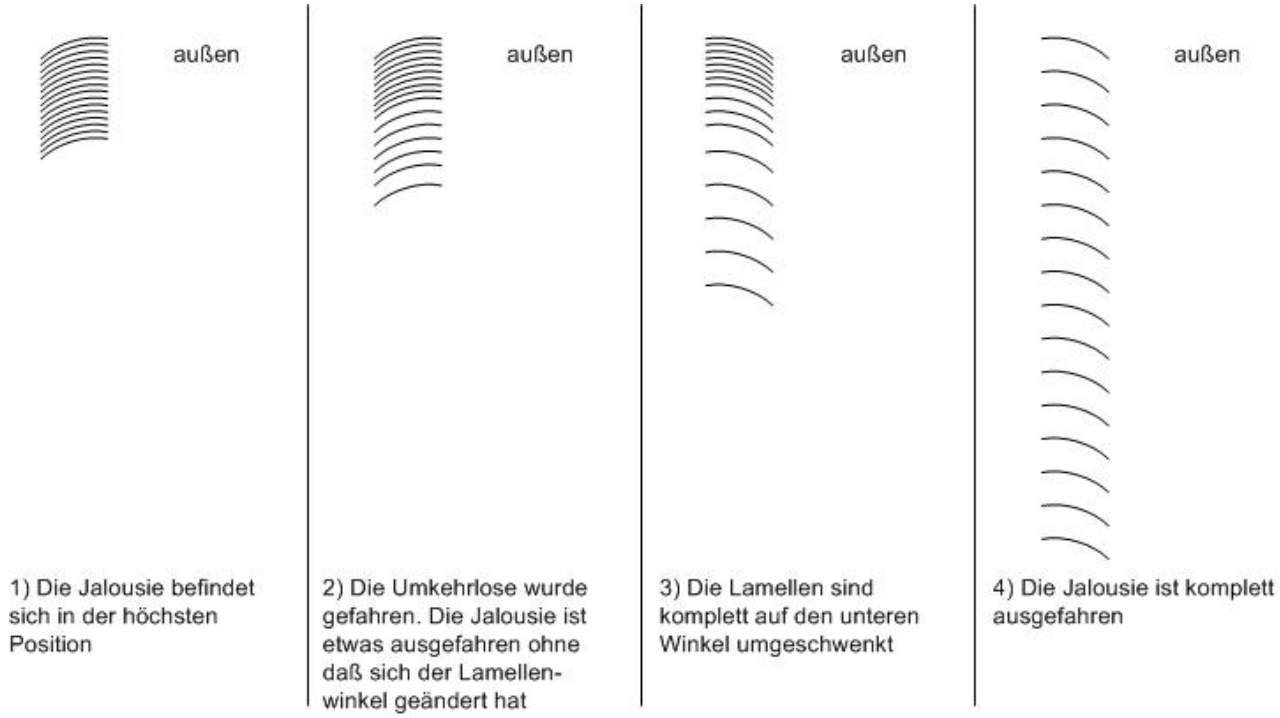
Struktur des Jalousie-Positioniertelegramms stSunblind [▶ 242].

```
TYPE ST_BARSunblind:
STRUCT
     uiPosition  : UINT;
```

```
    iAngle      : INT;
    bManUp      : BOOL;
    bManDown    : BOOL;
    bManualMode : BOOL;
    bActive     : BOOL;
END_STRUCT
END_TYPE
```

Die aktuelle Höhenposition und der Lamellenwinkel werden dabei nicht durch einen zusätzlichen Encoder eingelesen, sondern intern durch die Laufzeit der Jalousie ermittelt. Der Berechnung liegt folgendes Fahrprofil zugrunde (von der höchsten und niedrigsten Position der Jalousie aus betrachtet):

Fahrprofil abwärts:

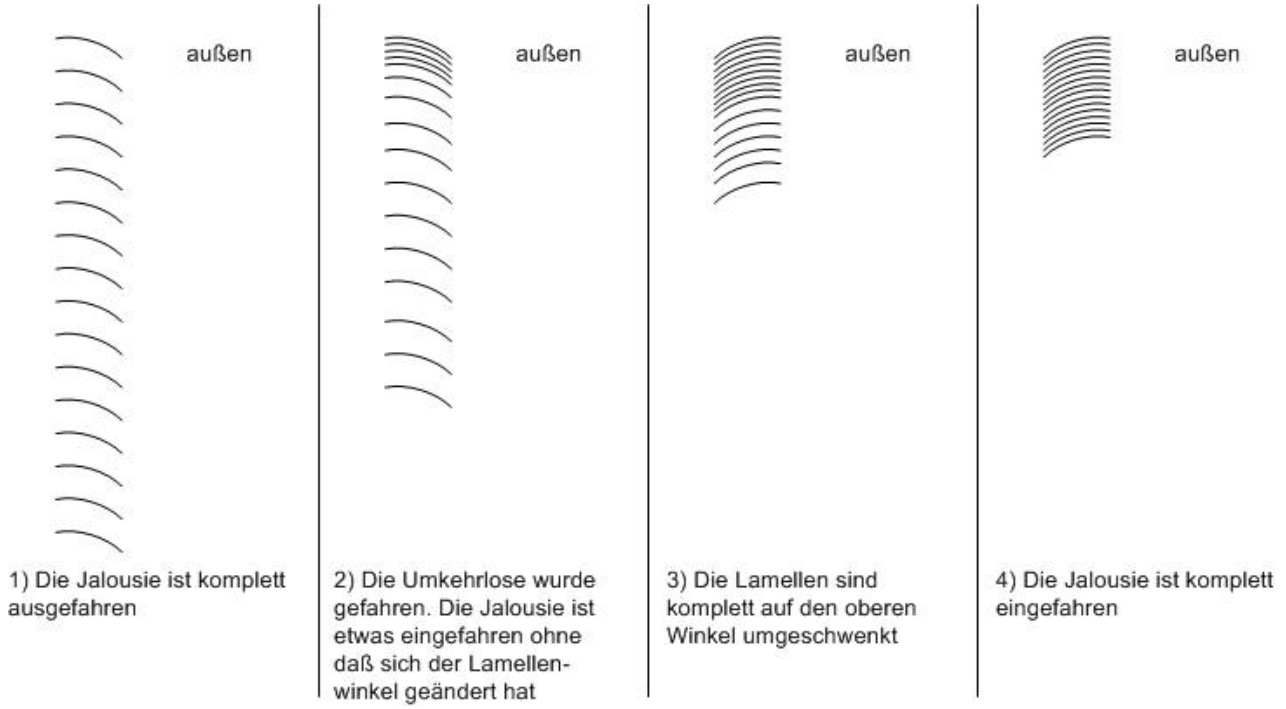


Nähere Erläuterung zu den Begriffen "Umkehrlose" und "Umschwenken", hier in der Abwärtsbewegung:

Die Jalousie beschreibt ihre Abwärtsbewegung normalerweise mit dem Lamellentiefpunkt nach außen gerichtet, wie in Bild 3.
Befindet sich die Jalousie in einer Ausgangsstellung mit dem Tiefpunkt nach innen (d.h. nach Abschluss einer Aufwärtsbewegung), so vergeht bei einer erneuten Abwärtsfahrt eine gewisse Zeit bis die Lamellen vom "Tiefpunkt innen" bis zu "Tiefpunkt außen" zu schwenken beginnen. Während dieser Zeit ändert sich der Lamellenwinkel nicht, die Jalousie fährt nur herunter (Bild1 und Bild2). Diese Zeit ist ein wichtiger Parameter zur Bewegungsberechnung und wird am Baustein unter *uiBacklashTimeDown* in ms eingetragen. Da man an einem x-beliebigen Punkt nach einer beliebig langen Jalousiebewegung nicht weiß, ob schon etwas von der Umkehrlosen ausgefahren wurde, misst man die Umkehrlose der Abwärtsbewegung bzw. deren Ausfahrzeit am Sichersten, wenn die

Jalousie zunächst ganz nach oben gefahren wurde. Ein weiterer wichtiger Parameter ist die Zeitspanne des darauf folgenden Umschwenkens der Lamellen von "Tiefpunkt innen" bis zu "Tiefpunkt außen". Diese Zeit ist als *uiTurningTimeDown* in ms am Baustein einzutragen.

Fahrprofil aufwärts:



Nähere Erläuterung zu den Begriffen "Umkehrlose" und "Umschwenken", hier in der Aufwärtsbewegung:

Dieser Sachverhalt ist analog zur oben beschriebenen Abwärtsbewegung: Die Jalousie beschreibt ihre Aufwärtsbewegung normalerweise mit dem Lamellentiefpunkt nach innen gerichtet, wie in Bild 3.
Befindet sich die Jalousie in einer Ausgangsstellung mit dem Tiefpunkt nach außen (d.h. nach Abschluss einer Abwärtsbewegung), so vergeht bei einer erneuten Aufwärtsfahrt eine gewisse Zeit bis die Lamellen vom "Tiefpunkt außen" bis zu "Tiefpunkt innen" zu schwenken beginnen. Während dieser Zeit ändert sich der Lamellenwinkel nicht, die Jalousie fährt nur herauf (Bild1 und Bild2). Auch diese Zeit ist ein wichtiger Parameter zur Bewegungsberechnung und wird am Baustein unter *uiBacklashTimeUp* in ms eingetragen. Da man an einem x-beliebigen Punkt nach einer beliebig langen Jalousiebewegung nicht weiß, ob schon etwas von der Umkehrlosen ausgefahren wurde, misst man die Umkehrlose der Aufwärtsbewegung bzw. deren Ausfahrzeit am Sichersten, wenn die Jalousie zunächst ganz nach unten gefahren wurde. Ein weiterer wichtiger Parameter ist die Zeitspanne des darauf folgenden Umschwenkens der Lamellen von "Tiefpunkt außen" bis zu "Tiefpunkt innen". Diese Zeit ist als *uiTurningTimeUp* in ms am Baustein einzutragen.

Parametrierung

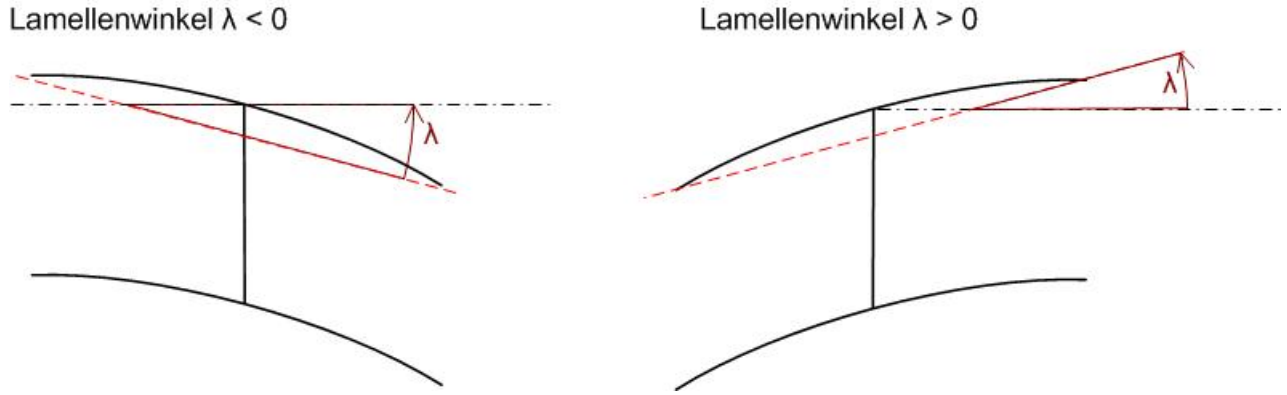
Zur Berechnung der (Höhen-)Position und des Lamellenwinkels sind nun jeweils für die Auf- und Abwärtsbewegung folgende Zeiten zu ermitteln:

- die Verfahrdauer der Umkehrlose (*uiBacklashTimeUp* / *uiBacklashTimeDown* in ms)
- die Verfahrdauer des Umschwenkens (*uiTurningTimeUp* / *uiTurningTimeDown* in ms)
- die Gesamt-Verfahrdauer (*uiTotalUpTime* / *uiTotalDownTime* in ms)

Des Weiteren sind zur Berechnung erforderlich:

- die ausgefahrene Jalousie-Gesamthöhe (*uiBlindHeight* in mm)
- der höchste Lamellenwinkel nach einem Umschwenken nach oben (*iAngleLimitUp* in Grad)
- der niedrigste Lamellenwinkel nach einem Umschwenken nach unten (*iAngleLimitDown* in Grad)

Der Lamellenwinkel λ ist dabei durch eine gedachte Gerade durch die Endpunkte der Lamelle zur Horizontalen definiert.



Funktionsweise

Der Baustein steuert die Jalousie grundsätzlich über die Informationen, aus dem Positioniertelegramm stSunblind [▶ 242]. Ist der Automatikmodus aktiv (*bManualMode*=FALSE), so wird immer die aktuelle Position und Lamellenwinkel angefahren, wobei Änderungen sofort berücksichtigt werden. Die Positionierung auf die Höhe hat dabei Vorrang: Es wird zunächst die eingegebene Höhe und danach der Lamellenwinkel angefahren. Aus Gründen der Einfachheit bleibt dabei der Positionsfehler durch das Winkel-Verfahren unberücksichtigt. Im Handbetrieb (*bManualMode*=TRUE) steuern die Befehle *bManUp* und *bManDown* die Jalousie.

Referenzieren

Ein sicheres Referenzieren ist gegeben, wenn die Jalousie länger als ihre komplette Hochlaufzeit nach oben hin angesteuert wird. Die Position ist dann auf jeden Fall "0" und der Lamellenwinkel auf seinem Maximum. Da eine Jalousiepositionierung ohne Encoder naturgemäß immer fehlerbehaftet ist, ist es wichtig möglichst oft automatisch zu referenzieren: jedes mal, wenn die Position "0" angefahren werden soll (der Winkel spielt dabei keine Rolle) fährt die Jalousie zunächst ganz normal mit kontinuierlicher Positionsberechnung nach oben. Erreicht sie den errechneten Positionswert 0%, so wird der Ausgang *bBlindUp* noch einmal für die weiterhin gehalten und zwar noch einmal für die komplette Hochlaufzeit + 5s.
Aus Gründen der Flexibilität gibt es nun zwei Möglichkeiten, den Referenziervorgang zu unterbrechen: Bis zum Erreichen der errechneten 0%-Position wird eine Positionsänderung immer noch angenommen und ausgeführt, nach Erreichen dieser 0%-Position kann die Jalousie noch mit dem Handbefehl "herunterfahren" anders bewegt werden. Diese beiden sinnvollen Einschränkungen machen es nötig, dass der Nutzer selbst dafür Sorge trägt, die Jalousie so oft wie möglich sicher referenzieren zu lassen.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bEnable          : BOOL;
stSunblind       : ST_BARSunblind;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Andernfalls wird der instanziierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bEnable: Freigabeeingang für den Baustein. Solange dieser Eingang auf TRUE steht, nimmt der Aktorbaustein Befehle entgegen und arbeitet, wie oben beschrieben. Ein FALSE-Signal an diesen Eingang setzt die Steuerausgänge *bBlindUp* und *bBlindDown* zurück und der Funktionsbaustein verharrt in einem Ruhezustand.

stSunblind: Positioniertelegramm, siehe ST_BARSunblind [▶ 242].

VAR_OUTPUT

```
bBlindUp     : BOOL;
bBlindDown   : BOOL;
uiActPosition: UINT;
uiActAngle   : UINT;
bReferencing : BOOL;
bBusy        : BOOL;
bError       : BOOL;
udiErrorId   : UDINT;
```

bBlindUp: Steuerausgang Jalousie hoch.

bBlindDown: Steuerausgang Jalousie herunter.

uiActPosition: Aktuelle Position in Prozent.

uiActAngle: Aktueller Lamellenwinkel in Grad.

bReferencing: Die Jalousie befindet sich in der Referenzierung, d.h. für die die komplette Hochlaufzeit + 5s wird der Ausgang *bBlindUp* gesetzt. Nur ein Handbefehl "herunter" kann die Jalousie in Gegenrichtung bewegen und diesen Modus beenden.

bBusy: Ein Positionier- oder Referenziervorgang findet statt.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId: Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

VAR_IN_OUT

Damit die eingetragenen Parameter über einen Steuerungsausfall hinweg erhalten bleiben ist es erforderlich, sie als In-Out-Variablen zu deklarieren. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variablen wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

```
uiBacklashTimeUp   : UINT;
uiBacklashTimeDown : UINT;
uiTurningTimeUp    : UINT;
uiTurningTimeDown  : UINT;
uiTotalTimeUp      : UINT;
uiTotalTimeDown    : UINT;
uiBlindHeight      : UINT;
iAngleLimitUp      : INT;
iAngleLimitDown    : INT;
```

uiBacklashTimeUp: Zeit zum Ausfahren der Umkehrlose in obere Richtung in ms.

uiBacklashTimeDown: Zeit zum Ausfahren der Umkehrlose in untere Richtung in ms.

uiTurningTimeUp: Zeit zum Umschwenken der Lamellen in obere Richtung in ms.

uiTurningTimeDown: Zeit zum Umschwenken der Lamellen in untere Richtung in ms.

uiTotalTimeUp: Komplette Hochfahrzeit in ms.

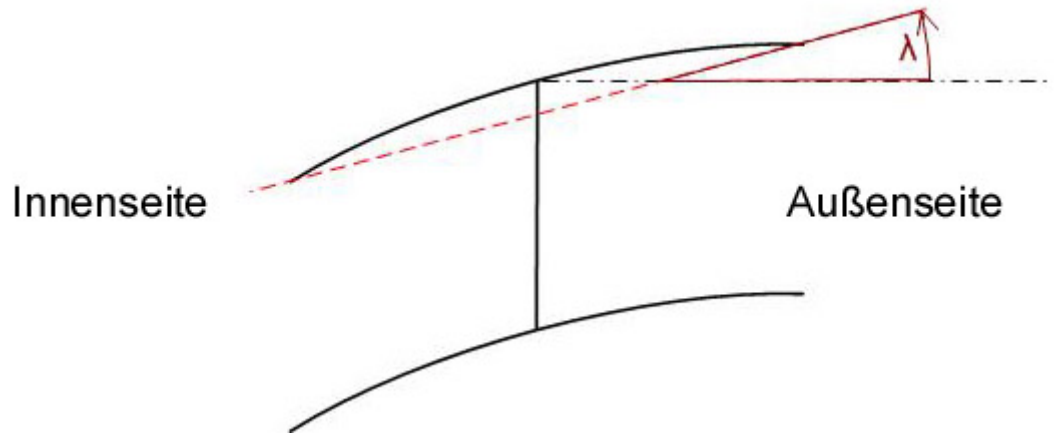
uiTotalTimeDown: Komplette Zeit zum Herunterfahren in ms.

uiBlindHeight: Jalousiehöhe in mm.

iAngleLimitUp: Höchste Stellung der Lamellen in Grad.

Diese Stellung ist erreicht, wenn die Jalousie ganz hochgefahren ist.

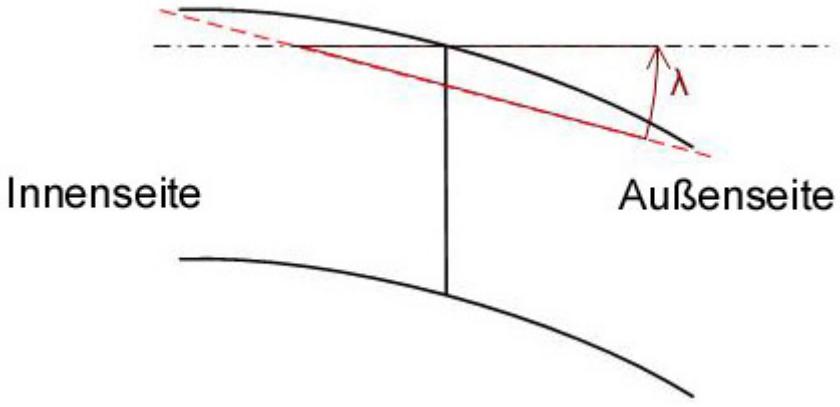
Der Lamellenwinkel λ, so wie er einleitend definiert ist, ist dann typischerweise größer Null.



iAngleLimitDown: Niedrigste Stellung der Lamellen in Grad.

Diese Stellung ist erreicht, wenn die Jalousie ganz heruntergefahren ist.

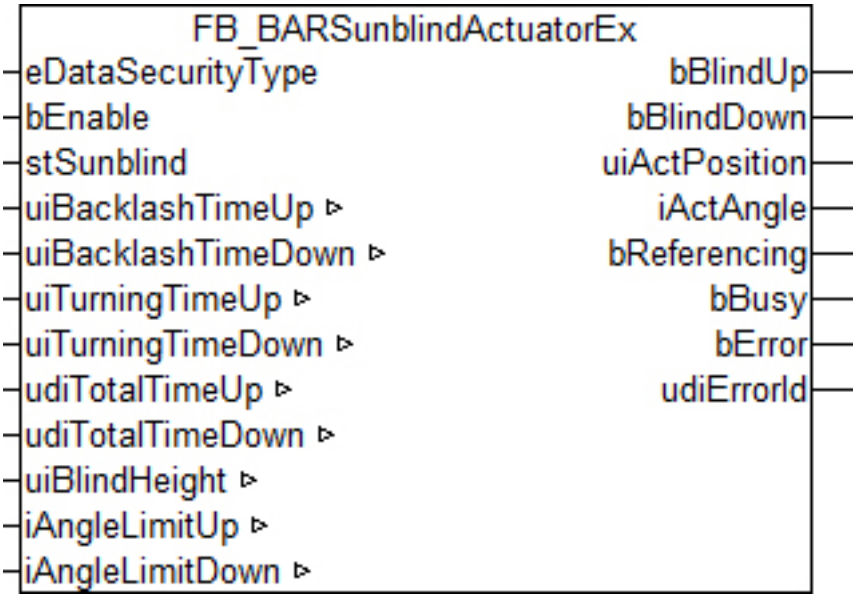
Der Lamellenwinkel λ, so wie er einleitend definiert ist, ist dann typischerweise kleiner Null.



3.3.4.14 FB_BARSunblindActuatorEx

Dieser Baustein dient zur Positionierung einer Lamellen-Jalousie über zwei Ausgänge: hoch- und herunterfahren. Über das Positioniertelegramm stSunblind [▶ 242] kann die Jalousie auf eine beliebige (Höhen-) Position und einen Lamellenwinkel gefahren werden. Darüber hinaus beinhaltet das Positioniertelegramm stSunblind [▶ 242] auch Handbefehle, mit denen die Jalousie individuell auf bestimmte Stellungen bewegt werden kann. Diese Handbefehle werden von dem Baustein FB_BARSunblindSwitch [▶ 211] angesteuert.

Im Gegensatz zum Baustein FB_BARSunblindActuator [▶ 189] erlaubt dieser die Eingabe längerer Verfahrzeiten, da diese hier im nicht im UINT, sondern UDINT-Format dargestellt sind (*udiTotalTimeUp/ udiTotalTimeDown*).

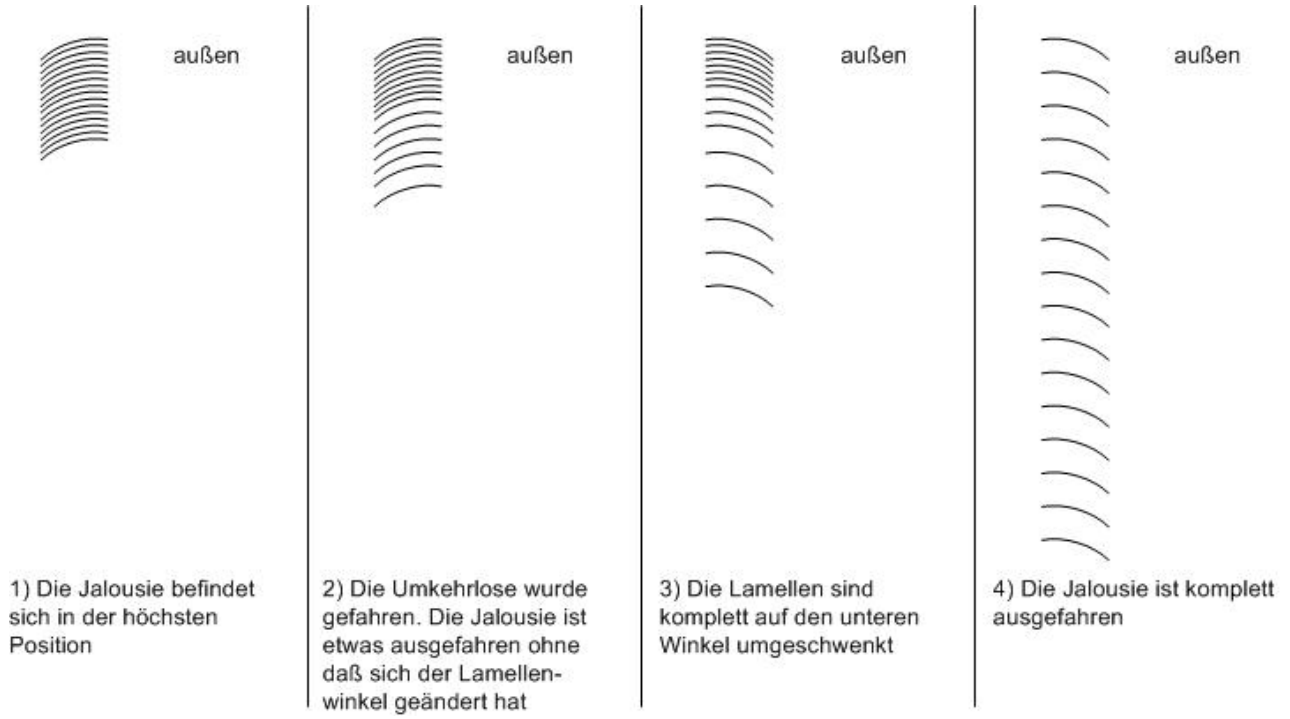


Struktur des Jalousie-Positioniertelegramms stSunblind [▶ 242].

```
TYPE ST_BARSunblind:
STRUCT
     uiPosition  : UINT;
     iAngle      : INT;
     bManUp      : BOOL;
     bManDown    : BOOL;
     bManualMode : BOOL;
     bActive     : BOOL;
END_STRUCT
END_TYPE
```

Die aktuelle Höhenposition und der Lamellenwinkel werden dabei nicht durch einen zusätzlichen Encoder eingelesen, sondern intern durch die Laufzeit der Jalousie ermittelt. Der Berechnung liegt folgendes Fahrprofil zugrunde (von der höchsten und niedrigsten Position der Jalousie aus betrachtet):

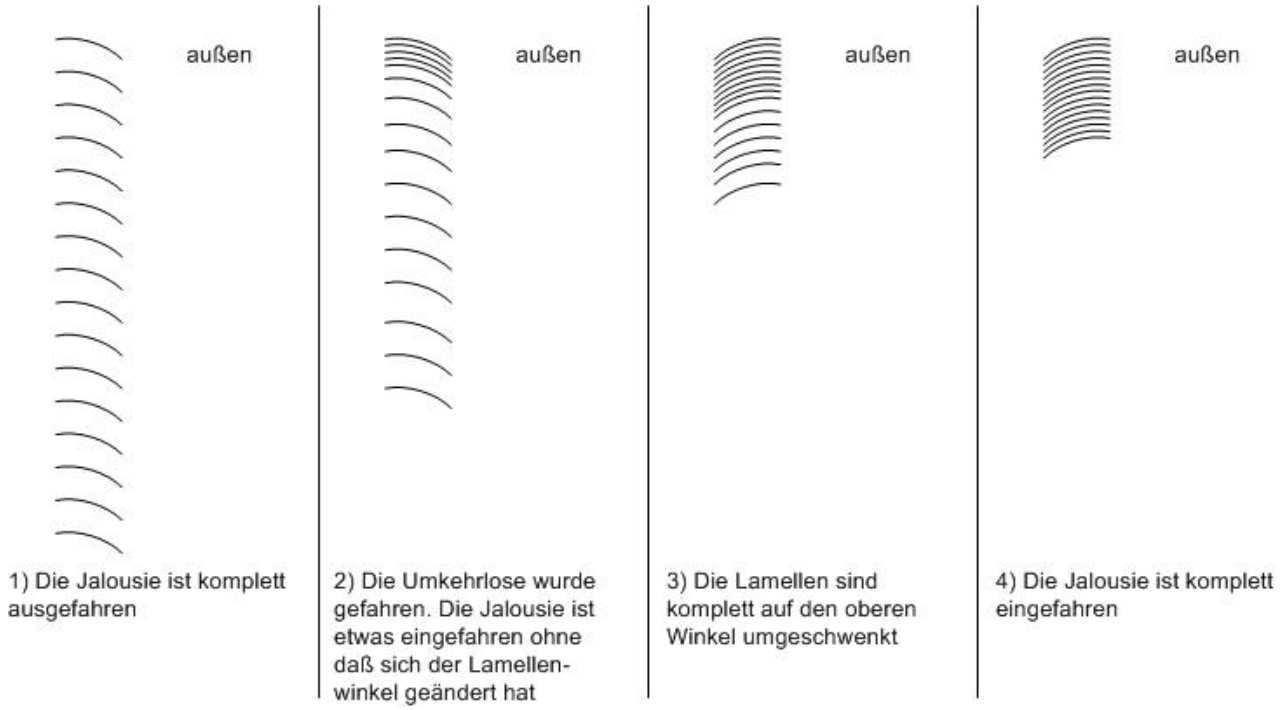
Fahrprofil abwärts:



Nähere Erläuterung zu den Begriffen "Umkehrlose" und "Umschwenken", hier in der Abwärtsbewegung:

Die Jalousie beschreibt ihre Abwärtsbewegung normalerweise mit dem Lamellentiefpunkt nach außen gerichtet, wie in Bild 3.
Befindet sich die Jalousie in einer Ausgangsstellung mit dem Tiefpunkt nach innen (d.h. nach Abschluss einer Aufwärtsbewegung), so vergeht bei einer erneuten Abwärtsfahrt eine gewisse Zeit bis die Lamellen vom "Tiefpunkt innen" bis zu "Tiefpunkt außen" zu schwenken beginnen. Während dieser Zeit ändert sich der Lamellenwinkel nicht, die Jalousie fährt nur herunter (Bild1 und Bild2). Diese Zeit ist ein wichtiger Parameter zur Bewegungsberechnung und wird am Baustein unter *uiBacklashTimeDown* in ms eingetragen. Da man an einem x-beliebigen Punkt nach einer beliebig langen Jalousiebewegung nicht weiß, ob schon etwas von der Umkehrlosen ausgefahren wurde, misst man die Umkehrlose der Abwärtsbewegung bzw. deren Ausfahrzeit am Sichersten, wenn die Jalousie zunächst ganz nach oben gefahren wurde. Ein weiterer wichtiger Parameter ist die Zeitspanne des darauf folgenden Umschwenkens der Lamellen von "Tiefpunkt innen" bis zu "Tiefpunkt außen". Diese Zeit ist als *uiTurningTimeDown* in ms am Baustein einzutragen.

Fahrprofil aufwärts:



Nähere Erläuterung zu den Begriffen "Umkehrlose" und "Umschwenken", hier in der Aufwärtsbewegung:

Dieser Sachverhalt ist analog zur oben beschriebenen Abwärtsbewegung: Die Jalousie beschreibt ihre Aufwärtsbewegung normalerweise mit dem Lamellentiefpunkt nach innen gerichtet, wie in Bild 3.
Befindet sich die Jalousie in einer Ausgangsstellung mit dem Tiefpunkt nach außen (d.h. nach Abschluss einer Abwärtsbewegung), so vergeht bei einer erneuten Aufwärtsfahrt eine gewisse Zeit bis die Lamellen vom "Tiefpunkt außen" bis zu "Tiefpunkt innen" zu schwenken beginnen. Während dieser Zeit ändert sich der Lamellenwinkel nicht, die Jalousie fährt nur herauf (Bild1 und Bild2). Auch diese Zeit ist ein wichtiger Parameter zur Bewegungsberechnung und wird am Baustein unter *uiBacklashTimeUp* in ms eingetragen. Da man an einem x-beliebigen Punkt nach einer beliebig langen Jalousiebewegung nicht weiß, ob schon etwas von der Umkehrlosen ausgefahren wurde, misst man die Umkehrlose der Aufwärtsbewegung bzw. deren Ausfahrzeit am Sichersten, wenn die

Jalousie zunächst ganz nach unten gefahren wurde. Ein weiterer wichtiger Parameter ist die Zeitspanne des darauf folgenden Umschwenkens der Lamellen von "Tiefpunkt außen" bis zu "Tiefpunkt innen". Diese Zeit ist als *uiTurningTimeUp* in ms am Baustein einzutragen.

Parametrierung

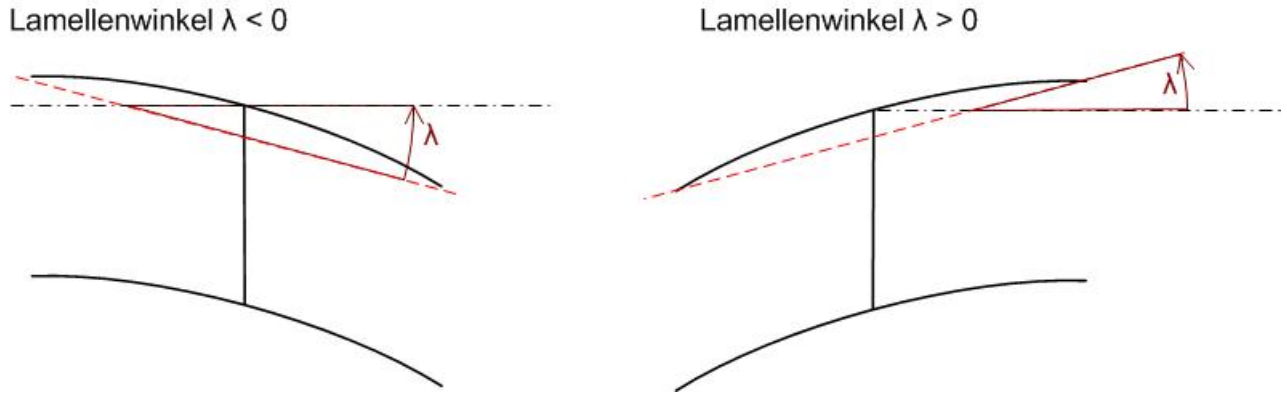
Zur Berechnung der (Höhen-)Position und des Lamellenwinkels sind nun jeweils für die Auf- und Abwärtsbewegung folgende Zeiten zu ermitteln:

- die Verfahrdauer der Umkehrlose (*uiBacklashTimeUp* / *uiBacklashTimeDown* in ms)
- die Verfahrdauer des Umschwenkens (*uiTurningTimeUp* / *uiTurningTimeDown* in ms)
- die Gesamt-Verfahrdauer (*udiTotalTimeUp* / *udiTotalTimeDownin* ms)

Des Weiteren sind zur Berechnung erforderlich:

- die ausgefahrene Jalousie-Gesamthöhe (*uiBlindHeight* in mm)
- der höchste Lamellenwinkel nach einem Umschwenken nach oben (*iAngleLimitUp* in Grad)
- der niedrigste Lamellenwinkel nach einem Umschwenken nach unten (*iAngleLimitDown* in Grad)

Der Lamellenwinkel λ ist dabei durch eine gedachte Gerade durch die Endpunkte der Lamelle zur Horizontalen definiert.



Funktionsweise

Der Baustein steuert die Jalousie grundsätzlich über die Informationen, aus dem Positioniertelegramm stSunblind [▶ 242]. Ist der Automatikmodus aktiv (*bManualMode*=FALSE), so wird immer die aktuelle Position und Lamellenwinkel angefahren, wobei Änderungen sofort berücksichtigt werden. Die Positionierung auf die Höhe hat dabei Vorrang: Es wird zunächst die eingegebene Höhe und danach der Lamellenwinkel angefahren. Aus Gründen der Einfachheit bleibt dabei der Positionsfehler durch das Winkel-Verfahren unberücksichtigt. Im Handbetrieb (*bManualMode*=TRUE) steuern die Befehle *bManUp* und *bManDown* die Jalousie.

Referenzieren

Ein sicheres Referenzieren ist gegeben, wenn die Jalousie länger als ihre komplette Hochlaufzeit nach oben hin angesteuert wird. Die Position ist dann auf jeden Fall "0" und der Lamellenwinkel auf seinem Maximum. Da eine Jalousiepositionierung ohne Encoder naturgemäß immer fehlerbehaftet ist, ist es wichtig möglichst oft automatisch zu referenzieren: jedes mal, wenn die Position "0" angefahren werden soll (der Winkel spielt dabei keine Rolle) fährt die Jalousie zunächst ganz normal mit kontinuierlicher Positionsberechnung nach oben. Erreicht sie den errechneten Positionswert 0%, so wird der Ausgang *bBlindUp* noch einmal für die weiterhin gehalten und zwar noch einmal für die komplette Hochlaufzeit + 5s.
Aus Gründen der Flexibilität gibt es nun zwei Möglichkeiten, den Referenziervorgang zu unterbrechen: Bis zum Erreichen der errechneten 0%-Position wird eine Positionsänderung immer noch angenommen und ausgeführt, nach Erreichen dieser 0%-Position kann die Jalousie noch mit dem Handbefehl "herunterfahren" anders bewegt werden. Diese beiden sinnvollen Einschränkungen machen es nötig, dass der Nutzer selbst dafür Sorge trägt, die Jalousie so oft wie möglich sicher referenzieren zu lassen.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bEnable          : BOOL;
stSunblind       : ST_BARSunblind;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Andernfalls wird der instanziierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bEnable: Freigabeeingang für den Baustein. Solange dieser Eingang auf TRUE steht, nimmt der Aktorbaustein Befehle entgegen und arbeitet, wie oben beschrieben. Ein FALSE-Signal an diesen Eingang setzt die Steuerausgänge *bBlindUp* und *bBlindDown* zurück und der Funktionsbaustein verharrt in einem Ruhezustand.

stSunblind: Positioniertelegramm, siehe ST_BARSunblind [▶ 242].

VAR_OUTPUT

```
bBlindUp     : BOOL;
bBlindDown   : BOOL;
uiActPosition: UINT;
uiActAngle   : UINT;
bReferencing : BOOL;
bBusy        : BOOL;
bError       : BOOL;
udiErrorId   : UDINT;
```

bBlindUp: Steuerausgang Jalousie hoch.

bBlindDown: Steuerausgang Jalousie herunter.

uiActPosition: Aktuelle Position in Prozent.

uiActAngle: Aktueller Lamellenwinkel in Grad.

bReferencing: Die Jalousie befindet sich in der Referenzierung, d.h. für die die komplette Hochlaufzeit + 5s wird der Ausgang *bBlindUp* gesetzt. Nur ein Handbefehl "herunter" kann die Jalousie in Gegenrichtung bewegen und diesen Modus beenden.

bBusy: Ein Positionier- oder Referenziervorgang findet statt.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId: Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

VAR_IN_OUT

Damit die eingetragenen Parameter über einen Steuerungsausfall hinweg erhalten bleiben ist es erforderlich, sie als In-Out-Variablen zu deklarieren. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variablen wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

```
uiBacklashTimeUp   : UINT;
uiBacklashTimeDown : UINT;
uiTurningTimeUp    : UINT;
uiTurningTimeDown  : UINT;
udiTotalTimeUp     : UDINT;
udiTotalTimeDown   : UDINT;
uiBlindHeight      : UINT;
iAngleLimitUp      : INT;
iAngleLimitDown    : INT;
```

uiBacklashTimeUp: Zeit zum Ausfahren der Umkehrlose in obere Richtung in ms.

uiBacklashTimeDown: Zeit zum Ausfahren der Umkehrlose in untere Richtung in ms.

uiTurningTimeUp: Zeit zum Umschwenken der Lamellen in obere Richtung in ms.

uiTurningTimeDown: Zeit zum Umschwenken der Lamellen in untere Richtung in ms.

udiTotalTimeUp: Komplette Hochfahrzeit in ms.

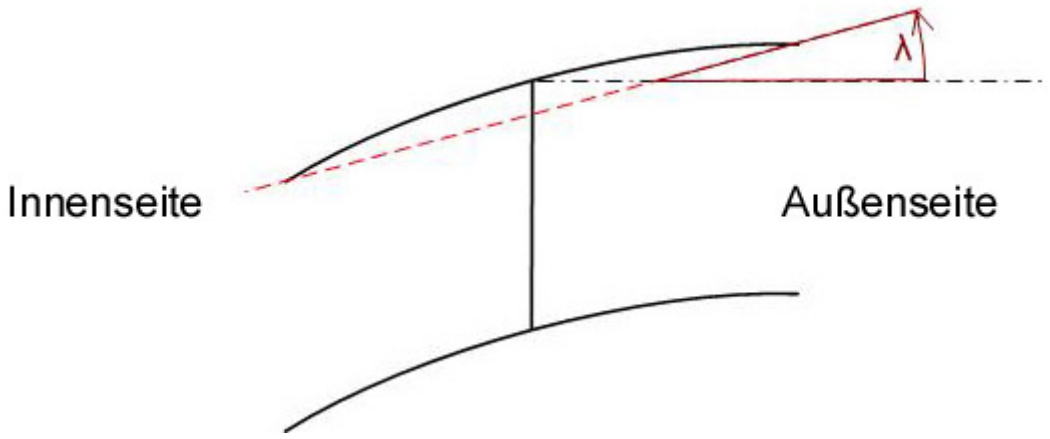
udiTotalTimeDown: Komplette Zeit zum Herunterfahren in ms.

uiBlindHeight: Jalousiehöhe in mm.

iAngleLimitUp: Höchste Stellung der Lamellen in Grad.

Diese Stellung ist erreicht, wenn die Jalousie ganz hochgefahren ist.

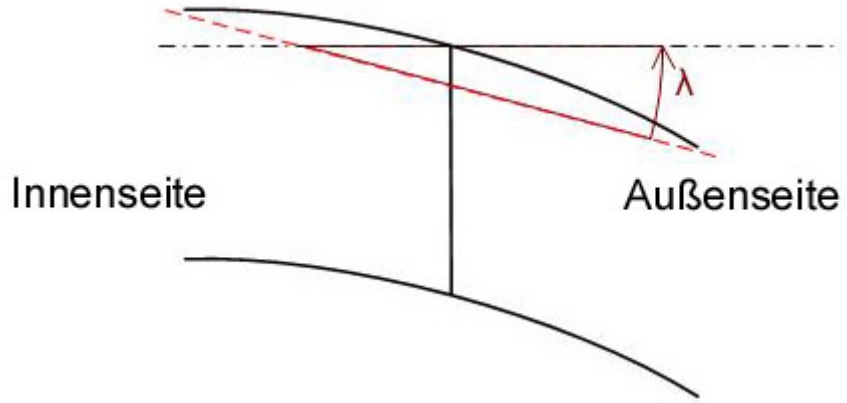
Der Lamellenwinkel λ, so wie er einleitend definiert ist, ist dann typischerweise größer Null.



iAngleLimitDown: Niedrigste Stellung der Lamellen in Grad.

Diese Stellung ist erreicht, wenn die Jalousie ganz heruntergefahren ist.

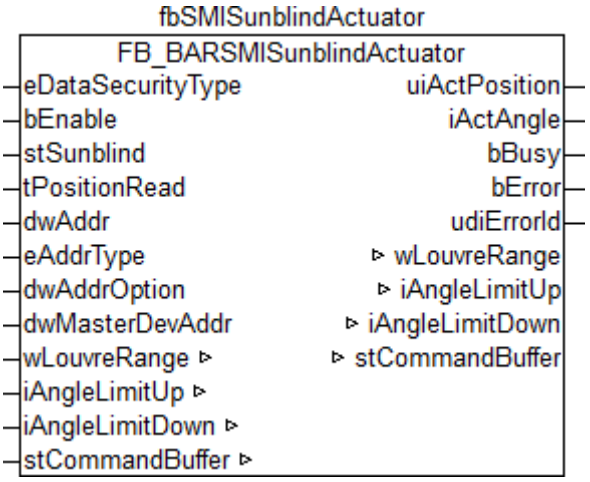
Der Lamellenwinkel λ, so wie er einleitend definiert ist, ist dann typischerweise kleiner Null.



3.3.4.15 FB_BARSMISunblindActuator

Dieser Baustein arbeitet wie der FB_BARSunblindActuator [▶ 189] nur mit dem Unterschied, dass ein SMI-Motor direkt angesteuert wird.

Benötigte Bibliothek ist die TwinCAT 2 PLC Lib: TcSMI.



Download SPS-Exportdatei: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/

11659719179.zip https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659719179.zip

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bEnable          : BOOL;
stSunblind       : ST_BARSunblind;
tPositionRead    : TIME;
dwAddr           : DWORD;
eAddrType        : E_SMIAddrType;
dwAddrOption     : DWORD;
dwMasterDevAddr  : DWORD;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Andernfalls wird der instanziierte FB intern nicht freigegeben.

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bEnable: Freigabeeingang für den Baustein. Solange dieser Eingang auf TRUE steht, nimmt der Aktorbaustein Befehle entgegen und arbeitet, wie oben beschrieben. Ein FALSE-Signal an diesen Eingang lässt den Funktionsbaustein in einem Ruhezustand verharren.

stSunblind: Positioniertelegramm, siehe ST_BARSunblind [▶ 242].

tPositionRead: Intervall, in dem die aktuelle Position vom SMI-Motor ausgelesen wird, wenn Positionierungsbefehle verarbeitet werden.

dwAddr: Herstellercode (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode, Adresse eines Teilnehmers, zur Gruppenadressierung oder als Slave-Id ausgewertet werden soll.

dwAddrOption: Wird das SMI-Gerät per Slave-Id adressiert (*eAddrType* = *eSMIAddrTypeSlaveId*), so muss über diesen Eingang der Herstellercode angegeben werden.

dwMasterDevAddr: Adresse (0-15) vom SMI-Motor, von dem die aktuelle Position ausgelesen werden soll, wenn *eAddrType* != *eSMIAddrTypeAddress* ist.

VAR_OUTPUT

```
uiActPosition: UINT;
iActAngle    : INT;
bBusy        : BOOL;
bError       : BOOL;
udiErrorId   : UDINT;
```

uiActPosition: Aktuelle Position in Prozent.

iActAngle: Aktueller Lamellenwinkel in Grad.

bBusy: Ein Positioniervorgang findet statt.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId: Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes.

VAR_IN_OUT

```
wLouvreRange       : WORD;
iAngleLimitUp      : INT;
iAngleLimitDown    : INT;
stCommandBuffer    : ST_SMICommandBuffer;
```

wLouvreRange: Schrittanzahl des SMI-Motors vom Lamellen "Tiefpunkt innen" zum "Tiefpunkt außen".

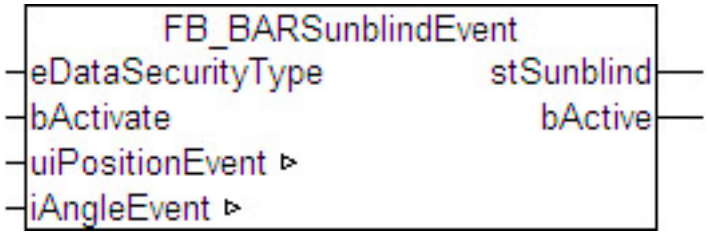
iAngleLimitUp: Höchste Stellung der Lamellen in Grad (-360..360).

iAngleLimitDown: Niedrigste Stellung der Lamellen in Grad (-360..360).

stCommandBuffer: Verweis auf die Struktur zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication()-Baustein.

3.3.4.16 FB_BARSunblindEvent

Dieser Baustein dient zur Positions- und Winkelvorgabe bei einem beliebigen Ereignis. Sie kann beispielsweise genutzt werden, um eine Parkposition anzufahren oder im Wartungsfall die Jalousie hochfahren zu lassen.



Die Funktion wird über den Eingang *bActivate* aktiviert. Ist dies der Fall, so wird der Aktiv-Merker im Positioniertelegramm (*bActive* in *stSunblind*) am Ausgang stSunblind [▶ 242] gesetzt und die an den In-Out-Variablen eingetragenen Werte *uiPositionEvent* für die Jalousiehöhe in % und *iAngleEvent* für den Lamellenwinkel in Grad in diesem Telegramm weiter gereicht. Ist die Funktion durch Rücksetzen von *bActive* nicht mehr aktiv, so wird der Aktiv-Merker im Positioniertelegramm stSunblind [▶ 242] wird zurück und die Positionen für Höhe und Winkel auf "0" gesetzt. Nutzt man den Prioritätenbaustein, so kann dann eine Funktion niedrigerer Priorität die Steuerung übernehmen.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bActivate         : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Andernfalls wird der instanziierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bActivate: Ein TRUE-Signal an diesem Eingang aktiviert den Baustein und übergibt die eingetragenen Sollwerte im Positioniertelegramm ST_BARSunblind [▶ 242] zusammen mit dem Aktivmerker. Ein FALSE-Signal setzt den Aktivmerker wieder zurück, sowie Position und Winkel auf Null.

VAR_OUTPUT

```
stSunblind : ST_BARSunblind;
bActive    : BOOL;
```

bActive : Entspricht dem Boole´schen Wert *bActive* im Jalousie-Telegramm ST_BARSunblind [▶ 242] und dient zur reinen Anzeige, ob der Baustein ein aktives Telegramm sendet.

stSunblind: Ausgabestruktur der Jalousiestellungen, siehe ST_BARSunblind [▶ 242]

VAR_IN_OUT

Damit die eingetragenen Parameter über einen Steuerungsausfall hinweg erhalten bleiben ist es erforderlich, sie als In-Out-Variablen zu deklarieren. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variablen wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die

Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

```
uiPositionPark : UINT;
iAnglePark     : INT;
```

uiPositionEvent: Höhenposition der Jalousie in % im Falle einer Aktivierung.

iAngleEvent: Lamellenwinkel der Jalousie in Grad im im Falle einer Aktivierung.

Dokumente hierzu

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

3.3.4.17 FB_BARSunblindPrioritySwitch

Prioritätssteuerung für bis zu 9 Positioniertelegramme (*stSunblind_Prio1* ... *stSunblind_Prio9*) des Typs ST_BARSunblind [▶ 242].

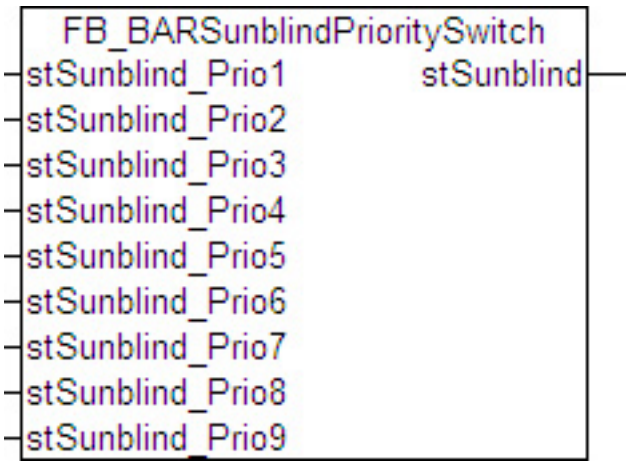


Abb. 8: FB_BARSunblindPrioritySwitch

Struktur des Jalousie-Positioniertelegramms ST_BARSunblind [▶ 242].

```
TYPE ST_BARSunblind:
STRUCT
     uiPosition  : UINT;
     iAngle      : INT;
     bManUp      : BOOL;
     bManDown    : BOOL;
     bManualMode : BOOL;
     bActive     : BOOL;
END_STRUCT
END_TYPE
```

An diesem Baustein lassen sich bis zu 9 Positioniertelegramme verschiedener Steuerbausteine anlegen. Dabei hat das Telegramm an *stSunblind_Prio1* die höchste und das an *stSunblind_Prio9* die niedrigste Priorität. Das aktive Telegramm der höchsten Priorität wird am Ausgang *stSunblind* ausgegeben. "Aktiv" bedeutet, dass innerhalb der Struktur des Positioniertelegramms die Variable *bActive* gesetzt ist.

Dieser Baustein ist so zu programmieren, dass immer eines der angelegten Telegramme aktiv ist. Sollte dies nicht der Fall sein, so wird am Ausgang ein Leertelegramm ausgegeben, d.h. *uiPosition*=0, *iAngle*=0, *bManUp*=FALSE, *bManDown*=FALSE, *bManualMode*=FALSE, *bActive*=FALSE. Da der Jalousiebaustein FB_BARSunblindActuator [▶ 189] bzw. der Rollladenbaustein FB_BARRollerblind [▶ 204] seinerseits nicht auf den Merker *bActive* achtet, würde dieses Telegramm als Fahrbefehl auf Position "0", also vollständig geöffnet, gewertet werden. Das Fehlen eines aktiven Telegramms stellt damit kein Sicherheitsrisiko für die Jalousie dar.

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 : Zur Auswahl stehende Positioniertelegramme. Dabei hat *stSunblind_Prio1* die höchste und *stSunblind_Prio9* die niedrigste Priorität.

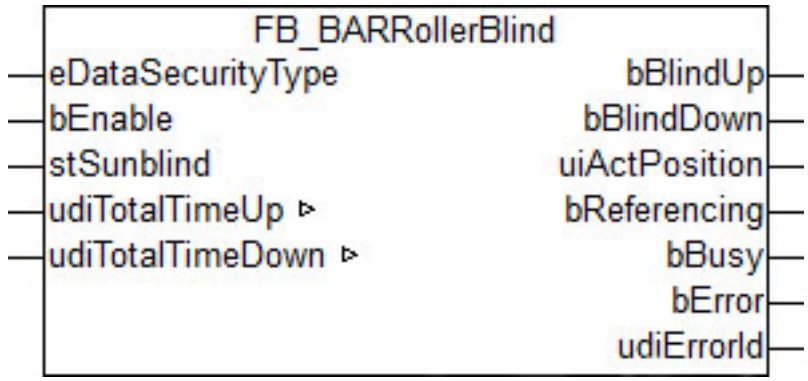
VAR_OUTPUT

```
stSunblind    : ST_BARSunblind;
```

stSunblind : Resultierendes Positioniertelegramm.

3.3.4.18 FB_BARRollerBlind

Dieser Baustein dient zur Positionierung einer Rolladen-Jalousie über zwei Ausgänge: Hoch- und Herunterfahren. Über das Positioniertelegramm stSunblind [▶ 242] kann die Jalousie auf eine beliebige Position gesteuert werden. Darüber hinaus beinhaltet das Positioniertelegramm stSunblind [▶ 242] auch Handbefehle, mit denen die Jalousie individuell auf bestimmte Positionen bewegt werden kann. Diese Handbefehle werden von dem Baustein FB_BARSunblindSwitch [▶ 211] angesteuert.



Struktur des Jalousie-Positioniertelegramms stSunblind [▶ 242].

```
TYPE ST_BARSunblind:
STRUCT
     uiPosition      : UINT;
     iAngle      : INT;
     bManUp      : BOOL;
     bManDown    : BOOL;
     bManualMode     : BOOL;
     bActive     : BOOL;
END_STRUCT
END_TYPE
```

Die aktuelle Höhenposition und der Lamellenwinkel werden dabei nicht durch einen zusätzlichen Encoder eingelesen, sondern intern durch die Laufzeit der Jalousie ermittelt.
Durch die beiden unterschiedlichen Laufzeitparameter *udiTotalTimeUp* (Laufzeit Jalousie hoch in ms) und *udiTotalTime*Down (Laufzeit Jalousie herunter in ms) wird den unterschiedlichen Fahrcharakteristiken Rechnung getragen.

Funktionsweise

Der Baustein steuert die Jalousie grundsätzlich über die Informationen, aus dem Positioniertelegramm stSunblind [▶ 242]. Ist der Automatikmodus aktiv (*bManualMode*=FALSE), so wird immer die aktuelle Position angefahren, wobei Änderungen sofort berücksichtigt werden. Im Handbetrieb (*bManualMode*=TRUE) steuern die Befehle *bManUp* und *bManDown* die Jalousie.

Referenzieren

Ein sicheres Referenzieren ist gegeben, wenn die Jalousie länger als ihre komplette Hochlaufzeit nach oben hin angesteuert wird. Die Position ist dann auf jeden Fall "0" . Da eine Jalousiepositionierung ohne Encoder naturgemäß immer fehlerbehaftet ist, ist es wichtig möglichst oft automatisch zu referenzieren: jedes mal, wenn die Position "0" angefahren werden soll, fährt die Jalousie zunächst ganz normal mit kontinuierlicher Positionsberechnung nach oben. Erreicht sie den errechneten Positionswert 0%, so wird der Ausgang *bBlindUp* noch einmal für die weiterhin gehalten und zwar noch einmal für die komplette Hochlaufzeit + 5s. Aus Gründen der Flexibilität gibt es nun zwei Möglichkeiten, den Referenziervorgang zu unterbrechen: Bis zum Erreichen der errechneten 0%-Position wird eine Positionsänderung immer noch angenommen und ausgeführt, nach Erreichen dieser 0%-Position kann die Jalousie noch mit dem Handbefehl "herunterfahren" anders bewegt werden. Diese beiden sinnvollen Einschränkungen machen es nötig, dass der Nutzer selbst dafür Sorge trägt, die Jalousie so oft wie möglich sicher referenzieren zu lassen.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bEnable          : BOOL;
stSunblind       : ST_BARSunblind;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Andernfalls wird der instanziierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bEnable: Freigabeeingang für den Baustein. Solange dieser Eingang auf TRUE steht, nimmt der Aktorbaustein Befehle entgegen und arbeitet, wie oben beschrieben. Ein FALSE-Signal an diesen Eingang setzt die Steuerausgänge *bBlindUp* und *bBlindDown* zurück und der Funktionsbaustein verharrt in einem Ruhezustand.

stSunblind: Positioniertelegramm, siehe ST_BARSunblind [▶ 242].

VAR_OUTPUT

```
bBlindUp     : BOOL;
bBlindDown   : BOOL;
uiActPosition: UINT;
bReferencing : BOOL;
bBusy        : BOOL;
bError       : BOOL;
udiErrorId   : UDINT;
```

bBlindUp: Steuerausgang Jalousie hoch.

bBlindDown: Steuerausgang Jalousie herunter.

uiActPosition: Aktuelle Position in Prozent.

bReferencing: Die Jalousie befindet sich in der Referenzierung, d.h. für die die komplette Hochlaufzeit + 5s wird der Ausgang *bBlindUp* gesetzt. Nur ein Handbefehl "herunter" kann die Jalousie in Gegenrichtung bewegen und diesen Modus beenden.

bBusy: Ein Positionier- oder Referenziervorgang findet statt.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId: Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

VAR_IN_OUT

Damit die eingetragenen Parameter über einen Steuerungsausfall hinweg erhalten bleiben ist es erforderlich, sie als In-Out-Variablen zu deklarieren. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variablen wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

```
udiTotalTimeUp      : UDINT;
udiTotalTimeDown    : UDINT;
```

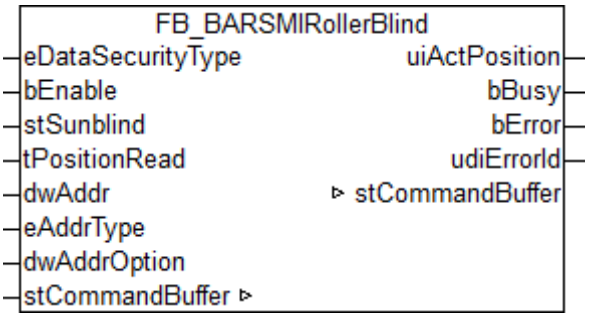
udiTotalTimeUp: Komplette Hochfahrzeit in ms.

udiTotalTimeDown: Komplette Zeit zum Herunterfahren in ms.

3.3.4.19 FB_BARSMIRollerBlind

Dieser Baustein arbeitet wie der FB_BARRollerBlind [▶ 204] nur mit dem Unterschied, dass ein SMI-Motor direkt angesteuert wird.

Benötigte Bibliothek ist die TwinCAT 2 PLC Lib: TcSMI.



Download SPS-Exportdatei: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659720587.zip https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659720587.zip

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bEnable          : BOOL;
stSunblind       : ST_BARSunblind;
tPositionRead    : TIME;
dwAddr           : DWORD;
eAddrType        : E_SMIAddrType;
dwAddrOption     : DWORD;
dwMasterDevAddr  : DWORD;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Andernfalls wird der instanziierte FB intern nicht freigegeben.

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bEnable: Freigabeeingang für den Baustein. Solange dieser Eingang auf TRUE steht, nimmt der Aktorbaustein Befehle entgegen und arbeitet, wie oben beschrieben. Ein FALSE-Signal an diesen Eingang lässt den Funktionsbaustein in einem Ruhezustand verharren.

stSunblind: Positioniertelegramm, siehe ST_BARSunblind [▶ 242].

tPositionRead: Intervall, in dem die aktuelle Position vom SMI-Motor ausgelesen wird, wenn Positionierungsbefehle verarbeitet werden.

dwAddr: Herstellercode (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode, Adresse eines Teilnehmers, zur Gruppenadressierung oder als Slave-Id ausgewertet werden soll.

dwAddrOption: Wird das SMI-Gerät per Slave-Id adressiert (*eAddrType* = *eSMIAddrTypeSlaveId*), so muss über diesen Eingang der Herstellercode angegeben werden.

dwMasterDevAddr: Adresse (0-15) vom SMI-Motor, von dem die aktuelle Position ausgelesen werden soll, wenn *eAddrType* != *eSMIAddrTypeAddress* ist.

VAR_OUTPUT

```
uiActPosition: UINT;
bBusy        : BOOL;
bError       : BOOL;
udiErrorId   : UDINT;
```

uiActPosition: Aktuelle Position in Prozent.

bBusy: Ein Positioniervorgang findet statt.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId: Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes.

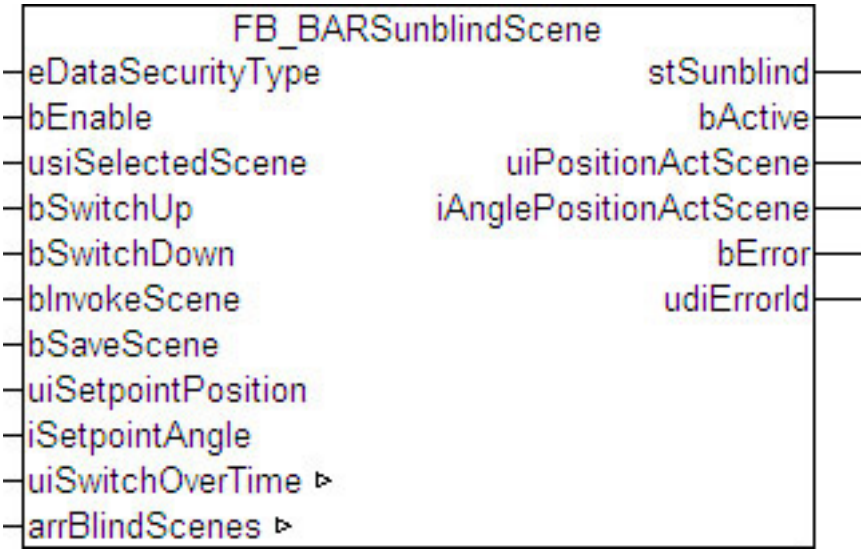
VAR_IN_OUT

```
stCommandBuffer    : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication()-Baustein.

3.3.4.20 FB_BARSunblindScene

Dieser Baustein stellt eine Erweiterung der Handbedienung FB_BARSunblindSwitch [▶ 211] um eine Szenen-Speicher- und Aufruf-Funktionalität dar. Damit lässt sich die Jalousieansteuerung FB_BARSunblindActuator [▶ 189] bzw. die Rollladenansteuerung FB_BARRollerblind [▶ 204] sowohl im Handbedienmodus ansteuern, als auch zuvor gespeicherte Positionen (Szenen) direkt anfahren. Es können bis zu 21 Szenen gespeichert werden.



Struktur des Jalousie-Positioniertelegramms stSunblind [▶ 242].

```
TYPE ST_BARSunblind:
STRUCT
     uiPosition  : UINT;
     iAngle      : INT;
     bManUp      : BOOL;
     bManDown    : BOOL;
     bManualMode : BOOL;
     bActive     : BOOL;
END_STRUCT
END_TYPE
```

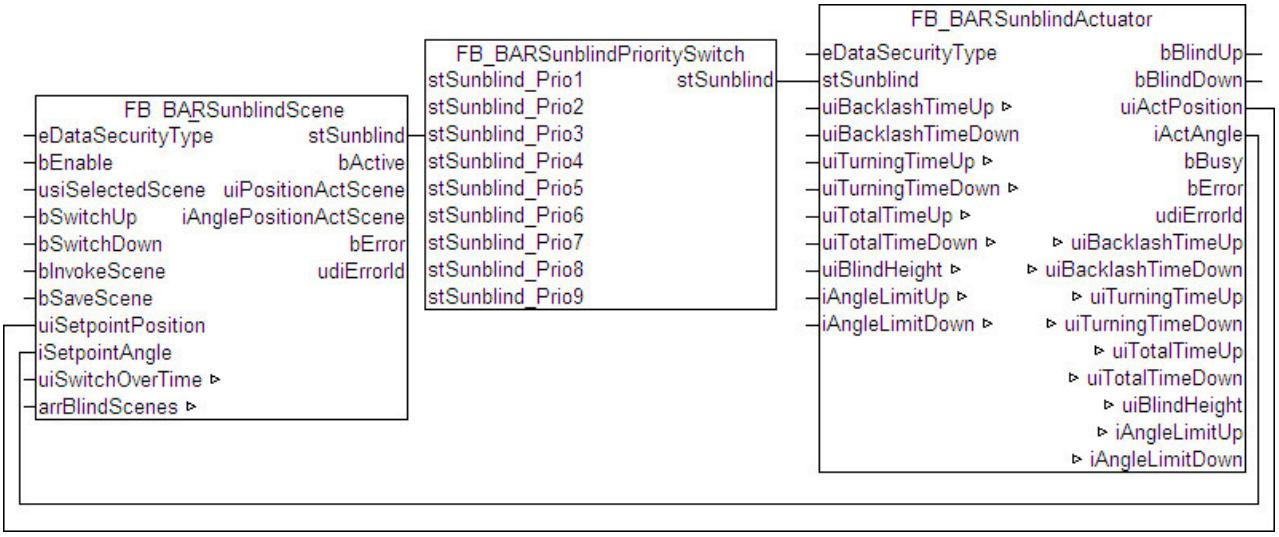
Betriebsweise

Der Funktionsbaustein steuert über die Befehlseingänge *bSwitchUp* und *bSwitchDown* den Jalousiebaustein FB_BARSunblindActuator [▶ 189] bzw. den Rollladenbaustein FB_BARRollerblind [▶ 204] im Handbetrieb an, wobei *bSwitchUp* Vorrang hat. Die Befehle werden an die jeweiligen Kommandos *bManUp* und *bManDown* des Positioniertelegramms weiter gereicht. Ist ein Befehlseingang länger als die eingetragene Zeit *uiSwitchOverTime* (in ms) aktiviert, so geht der entsprechende Steuerbefehl in Selbsthaltung. Ein erneutes Aktivieren eines Befehlseinganges löscht diese Selbsthaltung wieder.
Eine steigende Flanke an *bSaveScene* speichert die aktuelle Position und den Lamellenwinkel in die unter *usiSelectedScene* angewählte Szene. Dieser Vorgang ist jederzeit möglich, auch während einer aktiven Positionierung. Mit *bInvokeScene* wird die angewählte Szene aufgerufen, das heißt, die gespeicherten Werte von Position und Winkel angefahren.

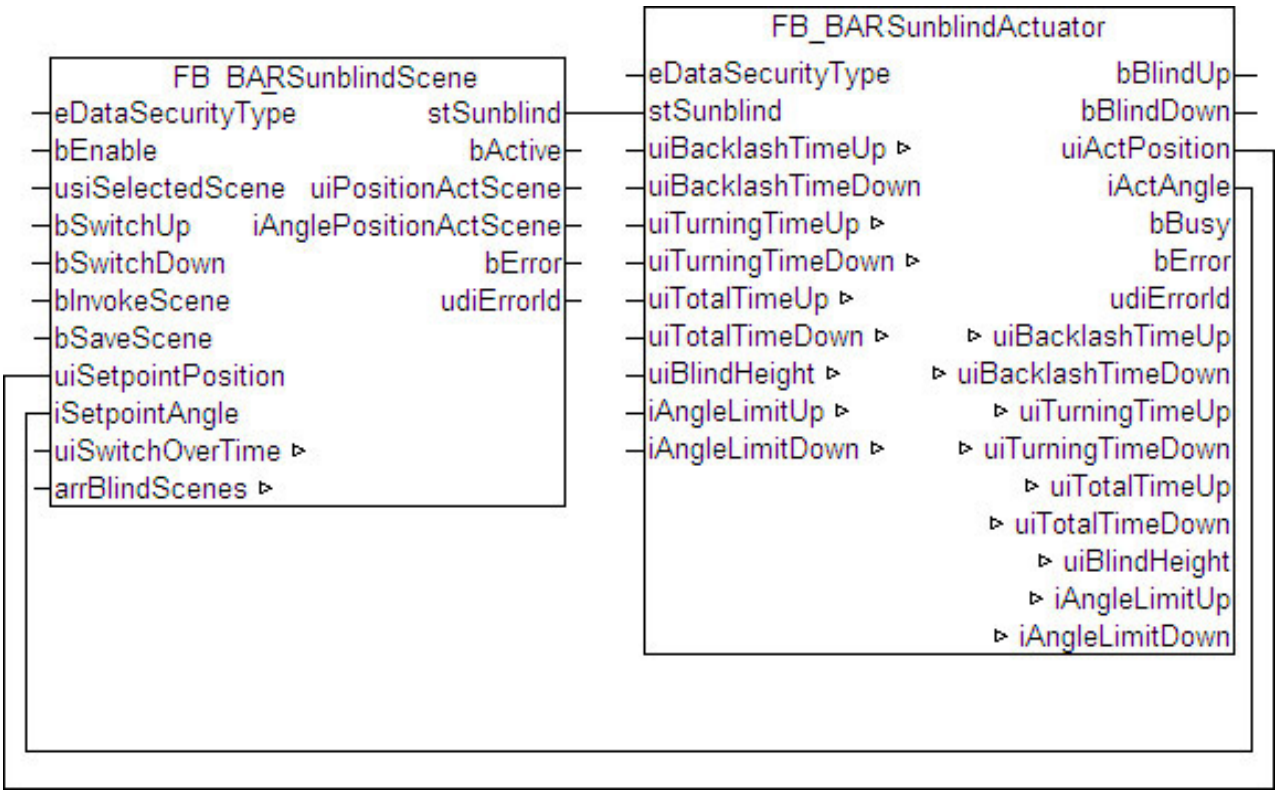
Verknüpfung an Jalousiebaustein

Der Szenenanwahlbaustein kann, wie der "normale" Handbedienbaustein FB_BARSunblindSwitch [▶ 211], entweder über eine voran gestellte Prioritätssteuerung FB_BARSunblindPrioritySwitch [▶ 203] oder aber direkt an den Jalousiebaustein angeschlossen werden. Die Verbindung erfolgt dabei über das Positioniertelegramm stSunblind [▶ 242]. Des weiteren benötigt der Szenenbaustein die aktuelle Positionen aus dem Jalousiebaustein der Referenz-Jalousie:

Verwendung einer Prioritätssteuerung:



Direkte Beschaltung:



VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
usiSelectedScene  : USINT;
bSwitchUp         : BOOL;
bSwitchDown       : BOOL;
bInvokeScene      : BOOL;
bSaveScene        : BOOL;
uiSetpointPosition: UINT;
iSetpointAngle    : INT;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Andernfalls wird der instanziierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bEnable : Steht dieser Eingang auf FALSE, so ist der Baustein ohne Funktion. Im Positioniertelegramm stSunblind [▶ 242] werden für Position und Winkel jeweils 0 ausgegeben - *bManualMode* und *bActive* stehen jeweils auf FALSE. Das bedeutet für eine Beschaltung mit Prioritätssteuerung, dass eine andere Funktionalität die Jalousieansteuerung übernimmt. Eine direkte Beschaltung hingegen lässt die Jalousie direkt auf Position 0, also ganz nach oben fahren, da der Aktorbaustein das Bit *bActive* selbst nicht auswertet.

usiSelectedScene : Angewählte Szene, welche entweder gespeichert (*bSaveScene*) oder aufgerufen (*bInvokeScene*) werden soll.

bSwitchUp : Befehlseingang Jalousie hoch.

bSwitchDown : Befehlseingang Jalousie herunter.

bInvokeScene : Angewählte Szene aufrufen.

bSaveScene : Angewählte Szene speichern.

uiSetpointPosition : Sollposition in %, welche in der gewählten Szene gesichert werden soll. Ist mit der Ist-Position des Aktorbausteins FB_BARSunblindActuator [▶ 189] bzw. FB_BARRollerblind [▶ 204] der Referenz-Jalousie/Rolllade zu verknüpfen, um dadurch eine vorher manuell angefahrene Position speichern zu können.

iSetpointAngle : dto. Lamellenwinkel in Grad.

VAR_OUTPUT

```
stSunblind            : ST_BARSunblind;
bActive               : BOOL;
uiPositionActScene    : UINT;
iAnglePositionActScene: INT;
bError                : BOOL;
udiErrorId            : UDINT;
```

stSunblind : Positioniertelegramm, siehe ST_BARSunblind [▶ 242].

bActive : Entspricht dem Boole´schen Wert *bActive* im Jalousie-Telegramm ST_BARSunblind [▶ 242] und dient zur reinen Anzeige, ob der Baustein ein aktives Telegramm sendet.

uiPositionActScene : Zeigt die gespeicherte relative Jalousiehöhenposition in % der aktuell angewählten Szene an.

iAnglePositionActScene : dto. Lammellenwinkel in Grad.

bError : Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId : Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

Sollte ein Fehler anstehen, so wird diese Automatik deaktiviert und Position und Winkel auf 0 gesetzt. Das bedeutet, dass bei Verwendung einer Prioritätssteuerung automatisch eine andere Funktion niedrigerer Priorität (siehe Übersicht) die Steuerung der Jalousie übernimmt. Bei direkter Beschaltung hingegen wird die Jalousie auf Position/Winkel 0 fahren.

VAR_IN_OUT

Damit die eingetragenen Parameter über einen Steuerungsausfall hinweg erhalten bleiben ist es erforderlich, sie als In-Out-Variablen zu deklarieren. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variablen wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

```
uiSwitchOverTime : UINT;
arrBlindScenes   : ARRAY[0..20] OF ST_BARSunblindScene;
```

uiSwitchOverTime : Zeit in Millisekunden bis bei dauerhaft aktiviertem Befehlseingang der entsprechende Handbefehl im Positioniertelegramm stSunblind [▶ 242] in Selbsthaltung geht.

arrBlindScenes : Tabelle mit den Szeneneinträgen vom Typ ST_BARSunblindScene [▶ 242]. Es können bis zu 21 Szenen gespeichert werden: 0..20.

Dokumente hierzu

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

3.3.4.21 FB_BARSunblindSwitch

Mit Hilfe dieses Bausteines lassen sich die Jalousieansteuerung FB_BARSunblindActuator [▶ 189] bzw. die Rollladensteuerung FB_BARRollerblind [▶ 204] im Handbedienmodus ansteuern. Die Verbindung erfolgt dabei über das Positioniertelegramm stSunblind [▶ 242] entweder direkt oder mit einer zusätzlichen Prioritätssteuerung.

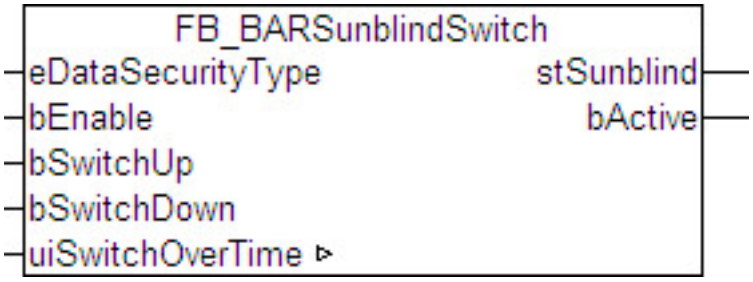


Abb. 9: FB_BARSunblindSwitch

Struktur des Jalousie-Positioniertelegramms stSunblind [▶ 242].

```
TYPE ST_BARSunblind:
STRUCT
     uiPosition  : UINT;
     iAngle      : INT;
     bManUp      : BOOL;
     bManDown    : BOOL;
     bManualMode : BOOL;
     bActive     : BOOL;
END_STRUCT
END_TYPE
```

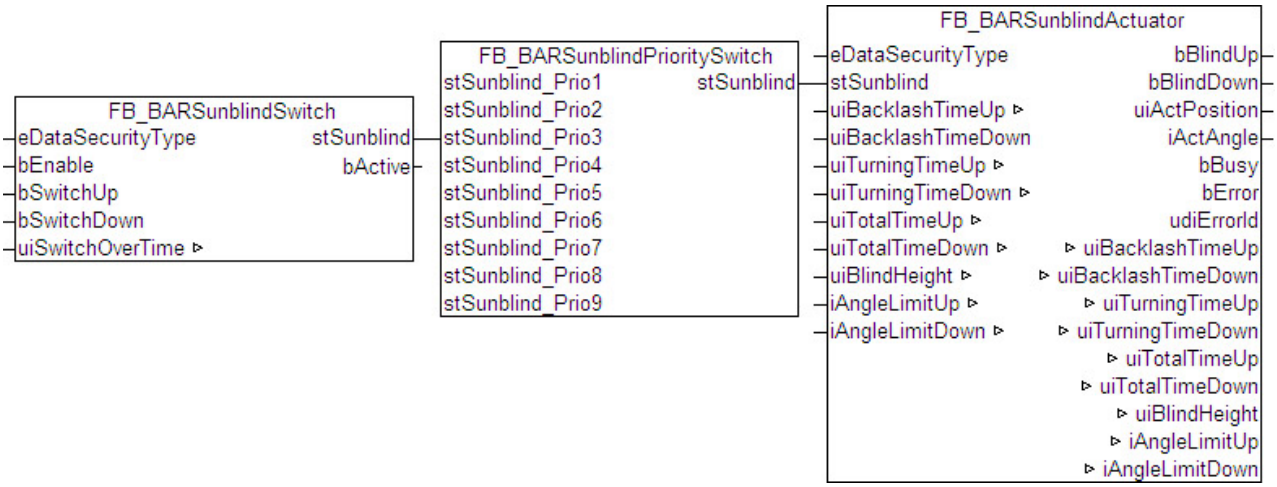
Betrieb

Der Funktionsbaustein steuert über die Befehlseingänge *bSwitchUp* und *bSwitchDown* den Jalousiebaustein FB_BARSunblindActuator [▶ 189] bzw. den Rollladenbaustein FB_BARRollerblind [▶ 204] im Handbetrieb an, wobei *bSwitchUp* Vorrang hat. Die Befehle werden an die jeweiligen Kommandos *bManUp* und *bManDown* des Positioniertelegramms weiter gereicht. Ist ein Befehlseingang länger als die eingetragene Zeit *uiSwitchOverTime* (in ms) aktiviert, so geht der entsprechende Steuerbefehl in Selbsthaltung. Ein erneutes Aktivieren eines Befehlseinganges löscht diese Selbsthaltung wieder.

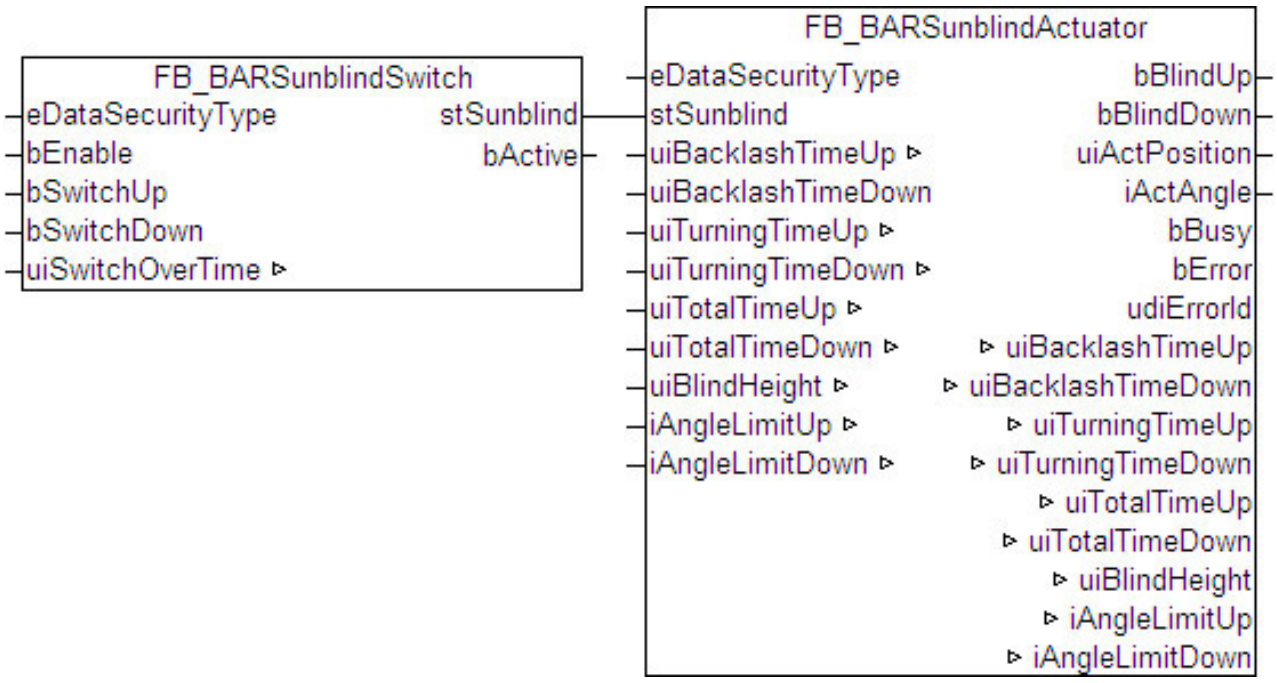
Verknüpfung an Jalousiebaustein

Der Handbedienbaustein kann entweder über eine voran gestellte Prioritätssteuerung FB_BARSunblindPrioritySwitch [▶ 203] oder aber direkt an den Jalousiebaustein angeschlossen werden. Die Verbindung erfolgt dabei über das Positioniertelegramm stSunblind [▶ 242].

Verwendung einer Prioritätssteuerung:



Direkte Beschaltung:



VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bEnable          : BOOL;
bSwitchUp        : BOOL;
bSwitchDown      : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Andernfalls wird der instanziierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bEnable : Steht dieser Eingang auf FALSE, so ist der Baustein ohne Funktion. Im Positioniertelegramm stSunblind [▶ 242] werden für Position und Winkel jeweils 0 ausgegeben - *bManualMode* und *bActive* stehen jeweils auf FALSE. Das bedeutet für eine Beschaltung mit Prioritätssteuerung, dass eine andere Funktionalität die Jalousieansteuerung übernimmt. Eine direkte Beschaltung hingegen lässt die Jalousie direkt auf Position 0, also ganz nach oben fahren, da der Aktorbaustein das Bit *bActive* selbst nicht auswertet.

bSwitchUp: Befehlseingang Jalousie hoch.

bSwitchDown: Befehlseingang Jalousie herunter.

VAR_OUTPUT

```
stSunblind : ST_BARSunblind;
bActive    : BOOL;
```

stSunblind : Positioniertelegramm, siehe ST_BARSunblind [▶ 242].

bActive : Entspricht dem Boole´schen Wert *bActive* im Jalousie-Telegramm ST_BARSunblind [▶ 242] und dient zur reinen Anzeige, ob der Baustein ein aktives Telegramm sendet.

VAR_IN_OUT

Damit die eingetragenen Parameter über einen Steuerungsausfall hinweg erhalten bleiben ist es erforderlich, sie als In-Out-Variablen zu deklarieren. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variablen wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

```
uiSwitchOverTime : UINT;
```

uiSwitchOverTime : Zeit in Millisekunden bis bei dauerhaft aktiviertem Befehlseingang der entsprechende Handbefehl im Positioniertelegramm stSunblind [▶ 242] in Selbsthaltung geht.

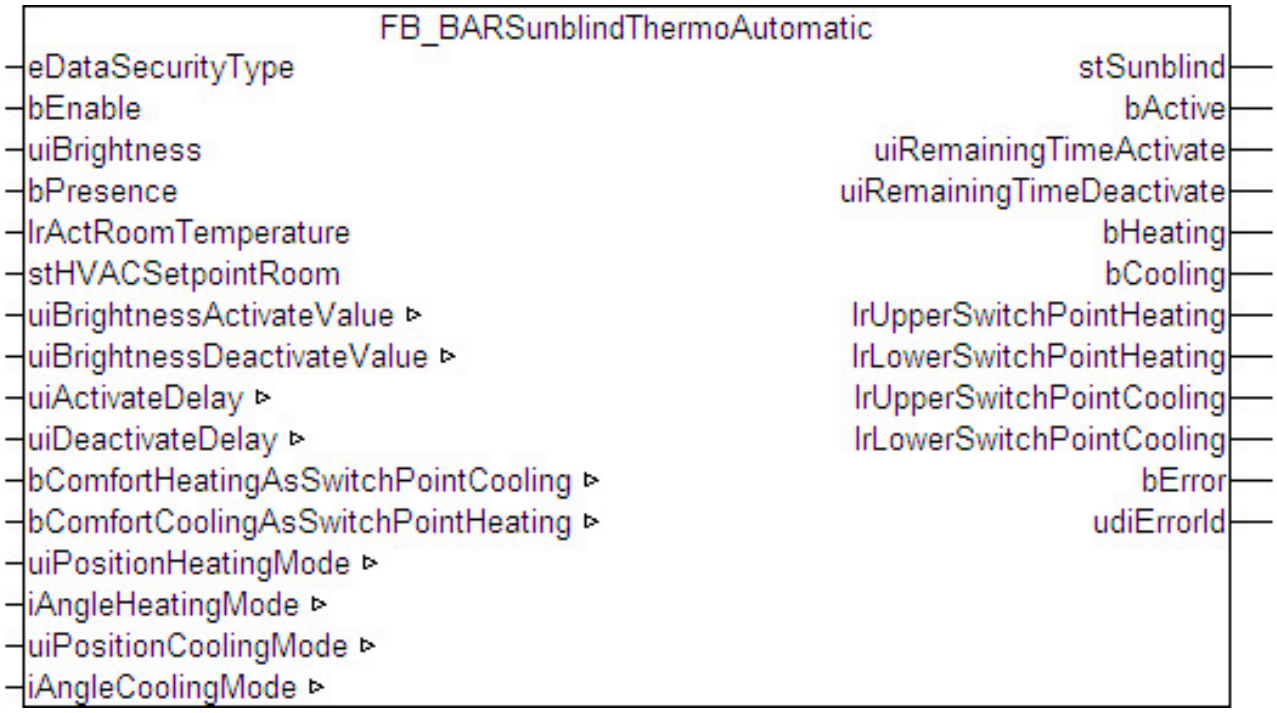
Dokumente hierzu

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

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

3.3.4.22 FB_BARSunblindThermoAutomatic

Dieser Baustein steuert die Jalousie bei **Nicht-Belegung** des Raumes in Abhängigkeit von der Raum-Innentemperatur.
Ziel ist es im Kühlfall Wärmestrahlung abzuhalten und im Heizfall durch Öffnen der Jalousien den Wärmeeintrag zu erhöhen.



Die Thermoautomatik ist nur dann aktiv, wenn genügend Sonneneinstrahlung vorhanden und der betreffende Raum verlassen ist. Dazu muss zum einen der Außenhelligkeitswert *uiBrightness* [lux] den Aktivierungs-Grenzwert *uiBrightnessActivateValue* für die Zeit *uiActivateDelay* (in Sekunden) überschritten haben und der Präsenz-Eingang *bPresence* auf FALSE stehen. Die Automatik wird dann wieder inaktiv, wenn entweder Präsenz gemeldet wird (*bPresence* = TRUE) oder für die Zeit *uiDeactivateDelay* die Außenhelligkeitswert *uiBrightness* unter den Wert *uiBrightnessDeactivateValue* absinkt.

Am Eingang *stBARSetpointRoom* ist eine Struktur vom Typ *ST_BARSetpointRoom* anzulegen, welche die Informationen über die Temperaturen-Schaltpunkte für die Klimatisierung des Raumes in sich trägt:

```
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
```

Einmal aktiviert, unterscheidet die Themoautomatik zwischen 3 Fällen:

- Verharren in der aktuellen Stellung, wenn sich die Raumtemperatur *rActRoomTemperature* zwischen den Werten *stBARSetpointRoom_ComfortHeat* und *stBARSetpointRoom_ComfortCool* bewegt.
- Heizfall, wenn die Raumtemperatur *rActRoomTemperature* den Wert *stBARSetpointRoom_ComfortHeat* unterschreitet.
- Kühlfall, wenn die Raumtemperatur *rActRoomTemperature* den Wert *stBARSetpointRoom_Comfort*Cool überschreitet.

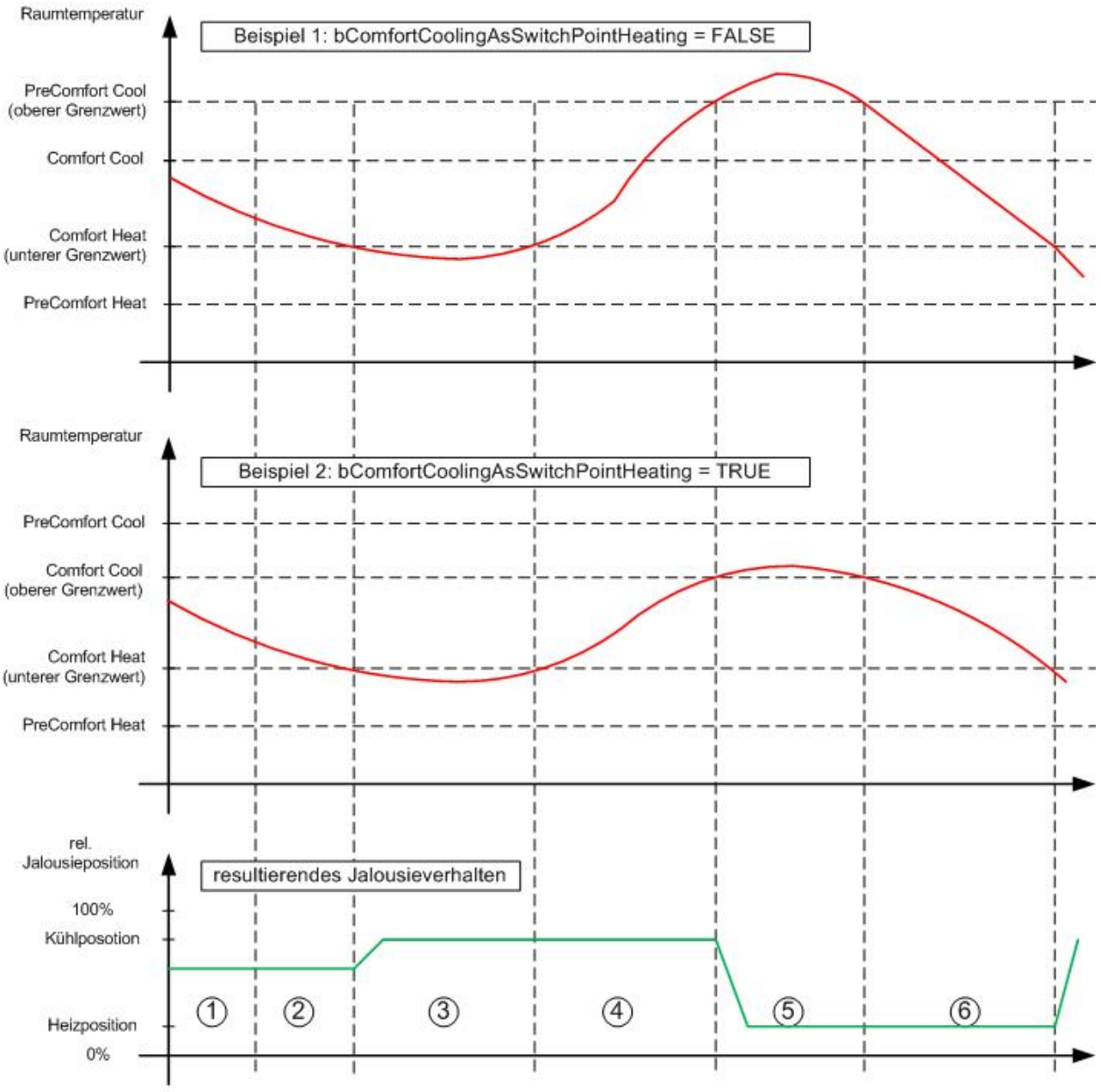
Wird einmal der Heiz- oder der Kühlfall erreicht, so merkt sich die Thermoautomatik dies so lange, bis sie selbst durch erneute Präsenzerkennung oder zu niedriger Außenhelligkeit (siehe oben) wieder inaktiv wird. In beiden Fällen jedoch - Heizen oder Kühlen - arbeitet die Automatik wie ein Zweipunktregler.

Heizfall

Im Heizfall fährt die Jalousie auf die Heizposition *uiPositionHeatingMode* und *iAngleHeatingMode,* wenn die Raumtemperatur den unteren Grenzwert unterschreitet. Überschreitet die Raumtemperatur den oberen Grenzwert, so fährt die Jalousie auf Kühlposition *uiPositionCoolingMode* und *iAngleCoolingMode*.
Der obere Grenzwert ist durch den Parameter *bComfortCoolingAsSwitchPointHeating* [BOOL] wählbar, der untere Grenzwert ist nicht wählbar:

	bComfortCoolingAsSwitchPointHeating = FALSE	bComfortCoolingAsSwitchPointHeating = TRUE
oberer Grenzwert	stBARSetpointRoom_PreComfortCool	stBARSetpointRoom_ComfortCool
unterer Grenzwert	stBARSetpointRoom_ComfortHeat	stBARSetpointRoom_ComfortHeat

Diagrammdarstellung



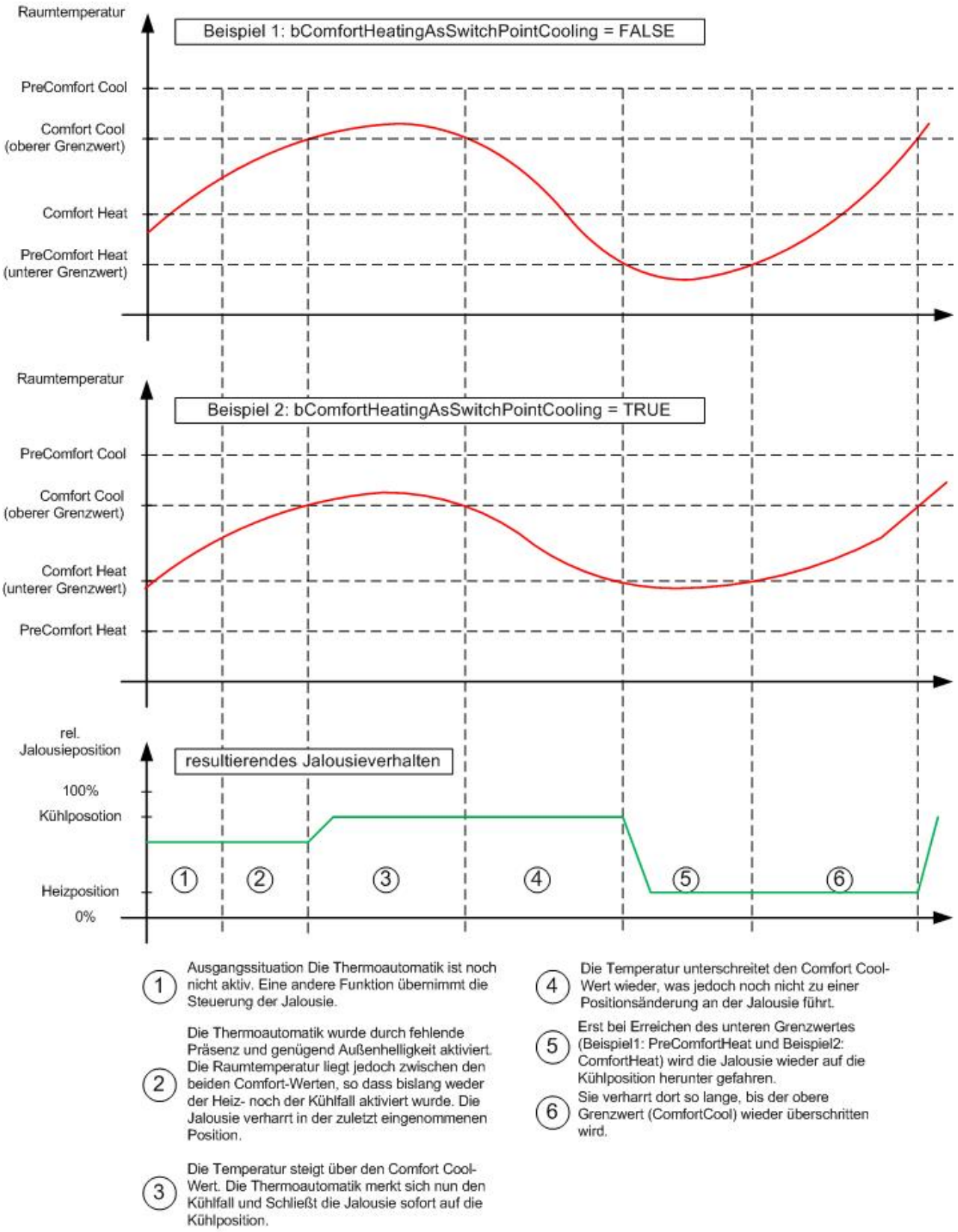
1. Ausgangssituation Die Thermoautomatik ist noch nicht aktiv. Eine andere Funktion übernimmt die Steuerung der Jalousie.
2. Die Thermoautomatik wurde durch fehlende Präsenz und genügend Außenhelligkeit aktiviert. Die Raumtemperatur liegt jedoch zwischen den beiden Comfort-Werten, so dass bislang weder der Heiz- noch der Kühlfall aktiviert wurde. Die Jalousie verharrt in der zuletzt eingenommenen Position.
3. Die Temperatur fällt unter den Comfort Heat-Wert. Die Thermoautomatik merkt sich nun den Heizfall und öffnet die Jalousie sofort auf die Heizposition.
4. Die Temperatur überschreitet den Comfort Heat-Wert wieder, was jedoch noch nicht zu einer Positionsänderung an der Jalousie führt.
5. Erst bei Erreichen des oberen Grenzwertes (Beispiel1: PreComfortCool und Beispiel2: ComfortCool) wird die Jalousie wieder auf die Kühlposition herunter gefahren.
6. Sie verharrt dort so lange, bis der untere Grenzwert (ComfortHeat) wieder unterschritten wird.

Kühlfall

Im Kühlfall fährt die Jalousie auf die Kühlposition *uiPositionCoolingMode* und *iAngleCoolingMode,* wenn die Raumtemperatur den oberen Grenzwert überschreitet. Unterschreitet die Raumtemperatur den unteren Grenzwert, so fährt die Jalousie auf Heizposition *uiPositionHeatingMode* und *iAngleHeatingMode*.
Der untere Grenzwert ist durch den Parameter *bComfortHeatingAsSwitchPointCooling* [BOOL] wählbar, der obere Grenzwert ist nicht wählbar:

	bComfortHeatingAsSwitchPointCooling = FALSE	bComfortHeatingAsSwitchPointCooling = TRUE
oberer Grenzwert	stBARSetpointRoom_ComfortCool	stBARSetpointRoom_ComfortCool
unterer Grenzwert	stBARSetpointRoom_PreComfortHeat	stBARSetpointRoom_ComfortHeat

Diagrammdarstellung



1. Ausgangssituation Die Thermoautomatik ist noch nicht aktiv. Eine andere Funktion übernimmt die Steuerung der Jalousie.
2. Die Thermoautomatik wurde durch fehlende Präsenz und genügend Außenhelligkeit aktiviert. Die Raumtemperatur liegt jedoch zwischen den beiden Comfort-Werten, so dass bislang weder der Heiz- noch der Kühlfall aktiviert wurde. Die Jalousie verharrt in der zuletzt eingenommenen Position.
3. Die Temperatur steigt über den Comfort Cool-Wert. Die Thermoautomatik merkt sich nun den Kühlfall und Schließt die Jalousie sofort auf die Kühlposition.
4. Die Temperatur unterschreitet den Comfort Cool-Wert wieder, was jedoch noch nicht zu einer Positionsänderung an der Jalousie führt.
5. Erst bei Erreichen des unteren Grenzwertes (Beispiel1: PreComfortHeat und Beispiel2: ComfortHeat) wird die Jalousie wieder auf die Kühlposition herunter gefahren.
6. Sie verharrt dort so lange, bis der obere Grenzwert (ComfortCool) wieder überschritten wird.

Durch die Wahl des *Precomfort Cool*-Wertes statt des *Comfort Cool*-Wertes als oberen Hysteresepunkt erreicht man eine durchschnittlich tiefere Raumtemperatur. Dieses könnte beim Wiederbetreten des Raumes als angenehmer empfunden werden.

VAR_INPUT

```
eDataSecurityType    : E_HVACDataSecurityType;
bEnable              : BOOL;
uiBrightness         : UINT;
bPresence            : BOOL;
lrActRoomTemperature: LREAL;
stBARSetpointRoom    : ST_BARSetpointRoom;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bEnable : Steht dieser Eingang auf FALSE, so ist der Baustein ohne Funktion. Im Positioniertelegramm stSunblind [▶ 242] werden für Position und Winkel jeweils 0 ausgegeben und *bActive* steht auf FALSE. Das bedeutet, dass über die Prioritätssteuerung eine andere Funktionalität die Jalousieansteuerung übernimmt.

uiBrightness: Außenhelligkeit in lux.

bPresence: Die Automatik ist auch nur dann aktiv, wenn der Raum unbelegt ist, da anderenfalls der Sonnenschutz Vorrang hätte. Ist dieser Eingang FALSE, gilt der Raum als nicht belegt und die Thermoautomatik ist aktiv. Eine TRUE-Signal an *bPresence* deaktiviert die Thermoautomatik **unmittelbar**.

lrActRoomTemperature: Eingang für die Raumtemperatur in °C.

stBARSetpointRoom: Struktur vom Typ ST_BARSetpointRoom [▶ 214], welche die Informationen über die Temperaturen-Schaltpunkte für die Klimatisierung des Raumes innehat, siehe oben.

VAR_OUTPUT

```
stSunblind               : ST_BARSunblind;
bActive                  : BOOL;
uiRemainingTimeActivate  : UINT;
uiRemainingTimeDeactivate: UINT;
bHeating                 : BOOL;
bCooling                 : BOOL;
lrUpperSwitchPointHeating: LREAL;
lrLowerSwitchPointHeating: LREAL;
lrUpperSwitchPointCooling: LREAL;
lrLowerSwitchPointCooling: LREAL;
bError                   : BOOL;
udiErrorId               : UDINT;
```

stSunblind: Ausgabestruktur der Jalousiestellungen, siehe ST_BARSunblind [▶ 242].

bActive : Entspricht dem BOOL´schen Wert *bActive* im Jalousie-Telegramm ST_BARSunblind [▶ 242] und dient zur reinen Anzeige, ob der Baustein ein aktives Telegramm sendet.

uiRemainingTimeActivate: Die Thermoautomatik ist aktiv, wenn keine Präsenz erkannt wird UND die Helligkeit für die Zeit *uiActivateDelay* (Schaltverzögerung) über dem Grenzwert *uiBrightnessActivateValue* liegt. Dieser Ausgang zeigt die verbleibende Dauer der Schaltverzögerung in Sekunden an. Solange kein Herunterzählen der Zeit stattfindet, steht dieser Ausgang auf 0.

uiRemainingTimeDeactivate: Die Thermoautomatik ist nicht aktiv, wenn Präsenz erkannt wird ODER die Helligkeit für die Zeit *uiDeactivateDelay* (Schaltverzögerung) unter dem Grenzwert *uiBrightnessDeactivateValue* liegt. Dieser Ausgang zeigt die verbleibende Dauer der Schaltverzögerung in Sekunden an. Solange kein Herunterzählen der Zeit stattfindet, steht dieser Ausgang auf 0.

bHeating : Zeigt an, ob der Heizfall aktiv ist.

bCooling : Zeigt an, ob der Kühlfall aktiv ist.

lrUpperSwitchPointHeating : Zeigt den gewählten oberen Umschaltpunkt im Heizbetrieb an, bei der die Jalousie zum Kühlen auf die Position *uiPositionCoolingMode* gefahren wird. Dies ist je nach Vorwahl entweder *stBARSetpointRoom_ComfortCool* oder *stBARSetpointRoom_PreComfortCool* (siehe Diagramme oben).

lrLowerSwitchPointHeating : Zeigt den unteren Umschaltpunkt im Heizbetrieb an, bei der die Jalousie zum Heizen auf die Position *uiPositionHeatingMode* gefahren wird. Dies ist fest eingestellt der Wert *stBARSetpointRoom_ComfortHeat* (siehe Diagramme oben).

lrUpperSwitchPointCooling : Zeigt den oberen Umschaltpunkt im Kühlbetrieb an, bei der die Jalousie zum Kühlen auf die Position *uiPositionCoolingMode* gefahren wird. Dies ist fest eingestellt der Wert *stBARSetpointRoom_ComfortCool* (siehe Diagramme oben).

lrLowerSwitchPointCooling : Zeigt den gewählten unteren Umschaltpunkt im Kühlbetrieb an, bei der die Jalousie zum Heizen auf die Position *uiPositionHeatingMode* gefahren wird. Dies ist je nach Vorwahl entweder *stBARSetpointRoom_ComfortHeat* oder *stBARSetpointRoom_PreComfortHeat* (siehe Diagramme oben).

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId: Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

Sollte ein Fehler anstehen, so wird diese Automatik deaktiviert und Position und Winkel auf 0 gesetzt. Das bedeutet, dass bei Verwendung einer Prioritätssteuerung automatisch eine andere Funktion niedrigerer Priorität (siehe Übersicht) die Steuerung der Jalousie übernimmt. Bei direkter Beschaltung hingegen wird die Jalousie auf Position/Winkel 0 fahren.

VAR_IN_OUT

Die Notwendigkeit, dass eingetragene Parameter über einen Steuerungsausfall hinweg erhalten bleiben, macht es erforderlich, dass sie als In-Out-Variablen deklariert werden. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variable wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

```
uiBrightnessActivateValue          : UINT;(*lux*)
uiBrightnessDeactivateValue        : UINT;(*lux*)
uiActivateDelay                    : UINT;(*sec*)
uiDeactivateDelay                  : UINT;(*sec*)
bComfortHeatingAsSwitchPointCooling: BOOL;
bComfortCoolingAsSwitchPointHeating: BOOL;
uiPositionHeatingMode              : UINT;
iAngleHeatingMode                  : INT;
uiPositionCoolingMode              : UINT;
iAngleCoolingMode                  : INT;
```

uiBrightnessActivateValue / uiActivateDelay : Aktivierungsgrenzwert. Die Thermoautomatik ist nur bei genügend Außenhelligkeit aktiv. Anderenfalls wäre keine Wärmestrahlung durch die Sonne gegeben die entweder nutzbar gemacht werden kann oder abgehalten werden muss. Überschreitet die Außenhelligkeit *uiBrightness* [lux] den Wert *uiBrightnessActivateValue* [lux] für die Zeit *uiActivateDelay* [s] , so wird die Automatik bei zusätzlich erkannter Nicht-Präsenz *(bPresence* = FALSE) aktiv.

uiBrightnessDeactivateValue / uiDeactivateDelay : Deaktivierungsgrenzwert der Helligkeit: Unterschreitet die Außenhelligkeit *uiBrightness* [lux] den Wert *uiBrightnessDeactivateValue* [lux] für die Zeit *uiDeactivateDelay* [s] , so wird die Automatik inaktiv. Der Wert *uiBrightnessDeactivateValue* muß kleiner sein als der Wert *uiBrightnessActivateValue*. Anderenfalls wird ein Fehler ausgegeben.

bComfortHeatingAsSwitchPointCooling : Steht dieser Eingang auf TRUE, so gilt der Wert *stBARSetpointRoom_ComfortHeat* als unterer Wert im Kühlmodus, ab dem die Jalousien wieder geöffnet werden. Ist der Eingang auf FALSE gesetzt, so ist der Wert *stBARSetpointRoom_PreComfortHeat* gültig (siehe einleitende Beschreibung).

bComfortCoolingAsSwitchPointHeating : Steht dieser Eingang auf TRUE, so gilt der Wert *stBARSetpointRoom_ComfortCool* als oberer Wert im Heizmodus, ab dem die Jalousien wieder geschlossen werden. Ist der Eingang auf FALSE gesetzt, so ist der Wert *stBARSetpointRoom_PreComfortCool* gültig (siehe einleitende Beschreibung)

uiPositionHeatingMode : Höhenposition der Jalousie in % im Heizfall (gewollte Wärmeeinstrahlung). Für die Höhenposition gilt: 0% = ganz geöffnet, 100% = ganz geschlossen.

iAngleHeatingMode : Lamellenwinkel der Jalousie in Grad im Heizfall (gewollte Wärmeeinstrahlung).

uiPositionCoolingMode : Höhenposition der Jalousie in % im Kühlfall (Sonnenschutz). Für die Höhenposition gilt: 0% = ganz geöffnet, 100% = ganz geschlossen. Der Wert der Höhenposition muss höher sein als die im Heizfall, d.h. die Jalousie muss im Kühlfall weiter geschlossen sein als im Heizfall. Ansonsten könnte die Wärmezufuhr durch die Sonne nach obiger Beschreibung nicht sinnvoll geregelt werden. In diesem Fall wird ein Fehler ausgegeben.

iAngleCoolingMode : Lamellenwinkel der Jalousie in Grad im Kühlfall (Sonnenschutz).

Dokumente hierzu

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

3.3.4.23 FB_BARSunblindTwilightAutomatic

Dieser Baustein steuert die Jalousie, sollte die Außenhelligkeit einen Grenzwert unterschritten haben.

Die Dämmerungsautomatik arbeitet mit einer Werte- und einer zeitlichen Hysterese: Unterschreitet der Außenhelligkeitswert *usiBrightness* [lux] für die Zeit *uiActivateDelay* [s] den Wert *usiActivateValue* [lux], so ist der Baustein aktiv und wird die an den IN-OUT-Variablen angegebenen Jalousiepositionen uiPositionActiveMode (Höhe in %) und uiAngleActiveMode (Lamellenwinkel in Grad) am Ausgang im Positioniertelegramm stSunblind [▶ 242] bereitstellen. Überschreitet die Außenhelligkeit hingegen für die Zeit *uiDeactivateDelay* [s] den Wert *usiDeactivateValue* [lux], so ist die Automatik nicht mehr aktiv. Der Aktiv-Merker im Positioniertelegramm stSunblind [▶ 242] wird zurück und die Positionen für Höhe und Winkel auf "0" gesetzt. Eine Funktion niedrigerer Priorität kann dann die Steuerung übernehmen.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bEnable          : BOOL;
uiBrightness     : UINT;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bEnable : Steht dieser Eingang auf FALSE, so ist der Baustein ohne Funktion. Im Positioniertelegramm stSunblind [▶ 242] werden für Position und Winkel jeweils 0 ausgegeben und *bActive* steht auf FALSE. Das bedeutet, dass über die Prioritätssteuerung eine andere Funktionalität die Jalousieansteuerung übernimmt.

uiBrightness: Außenhelligkeit in lux.

VAR_OUTPUT

```
stSunblind               : ST_BARSunblind;
bActive                  : BOOL;
uiRemainingTimeActivate  : UINT;
uiRemainingTimeDeactivate: UINT;
bError                   : BOOL;
udiErrorId               : UDINT;
```

stSunblind: Ausgabestruktur der Jalousiestellungen, siehe ST_BARSunblind [▶ 242].

bActive : Entspricht dem BOOL´schen Wert *bActive* im Jalousie-Telegramm ST_BARSunblind [▶ 242] und dient zur reinen Anzeige, ob der Baustein ein aktives Telegramm sendet.

uiRemainingTimeActivate: Zeigt die verbleibende Zeit an nach Unterschreitung des Schaltwertes *usiActivateValue* bis zur Aktivierung der Automatik in Sekunden an. Solange kein Herunterzählen der Zeit stattfindet, steht dieser Ausgang auf 0.

uiRemainingTimeDeactivate: Zeigt die verbleibende Zeit an nach Überschreitung des Schaltwertes *usiDeactivateValue* bis zur Abschaltung der Automatik in Sekunden an. Solange kein Herunterzählen der Zeit stattfindet, steht dieser Ausgang auf 0.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId: Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

Sollte ein Fehler anstehen, so wird diese Automatik deaktiviert und Position und Winkel auf 0 gesetzt. Das bedeutet, dass bei Verwendung einer Prioritätssteuerung automatisch eine andere Funktion niedrigerer Priorität (siehe Übersicht) die Steuerung der Jalousie übernimmt. Bei direkter Beschaltung hingegen wird die Jalousie auf Position/Winkel 0 fahren.

VAR_IN_OUT

Die Notwendigkeit, dass eingetragene Parameter über einen Steuerungsausfall hinweg erhalten bleiben, macht es erforderlich, dass sie als In-Out-Variablen deklariert werden. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variable wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

```
uiActivateValue      : UINT;
uiDeactivateValue    : UINT;
uiActivateDelay      : UINT;
uiDeactivateDelay    : UINT;
uiPositionActiveMode : UINT;
iAngleActiveMode     : INT;
```

uiActivateValue: Aktivierungsgrenzwert in lux.

uiDeactivateValue: Deaktivierungsgrenzwert in lux.

uiActivateDelay: Aktivierungsverzögerung in Sekunden.

uiDeactivateDelay: Deaktivierungsverzögerung in Sekunden.

uiPositionActiveMode: Höhenposition der Jalousie in %, sollte die Dämmerungsautomatik aktiv sein.

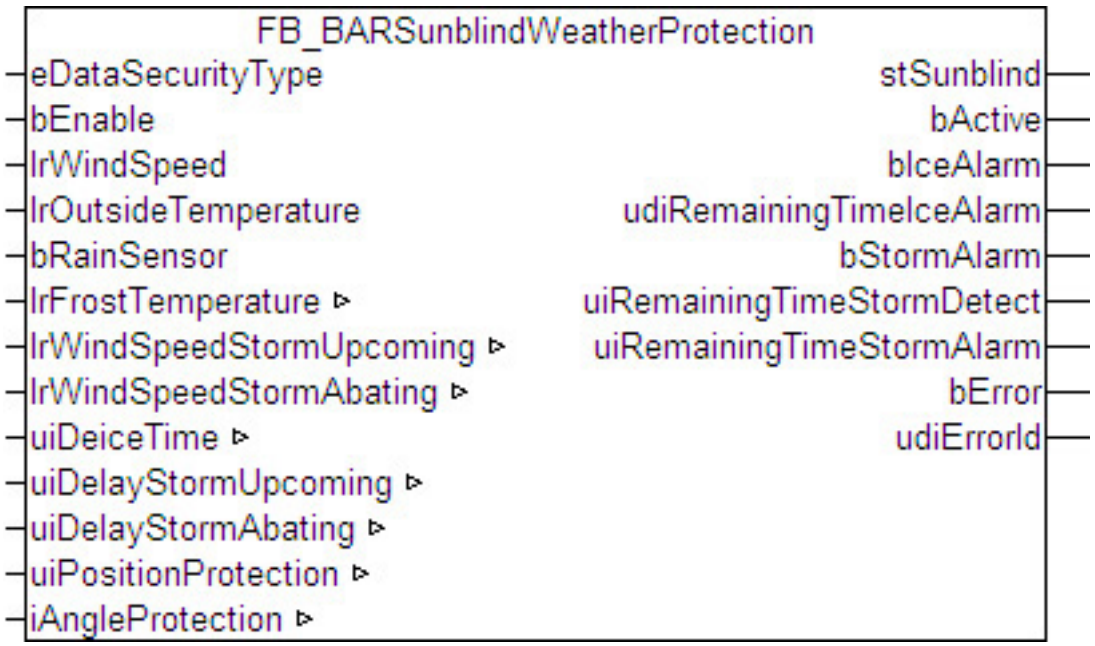
iAngleActiveMode: Lamellenwinkel der Jalousie in Grad, sollte die Dämmerungsautomatik aktiv sein.

Dokumente hierzu

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

3.3.4.24 FB_BARSunblindWeatherProtection

Der Witterungsschutz hat bei der Jalousiesteuerung die höchste Priorität (siehe Übersicht) und soll sicherstellen, dass die Jalousie weder durch Eis, noch durch Wind beschädigt wird.



Die Witterungsschutzautomatik hat die Aufgabe, die Jalousie vor zwei unmittelbar bevorstehenden Gefahren zu schützen und sie dafür in eine sichere Position zu fahren:

- **Vereisung:** Eine bevorstehende Vereisung wird dadurch erkannt, dass während einer Niederschlagserkennung an *bRainSensor* die gemessene Außentemperatur *rOutsideTemperature* den Frost-Grenzwert *rFrostTemperature* unterschreitet. Dieses Ereignis wird intern gespeichert und bleibt dann so lange bestehen, bis sichergestellt ist, dass das Eis wieder abgetaut ist. Dazu muss die Außentemperatur den Frost-Grenzwert für die eingetragene Enteisungszeit *uiDeiceTime* (in Minuten) überschritten haben. Aus Sicherheitsgründen wird das Vereisungsereignis persistent, also über einen SPS-Ausfall hinweg gespeichert. Fällt die Steuerung also während einer Ver- bzw. Enteisungsperiode aus, so gilt die Jalousie nach Wiederanlauf der Steuerung als neu vereist und der Enteisungszeitmesser startet von Neuem.
- **Sturm :** Liegt die gemessene Windgeschwindigkeit für die Zeit *uiDelayStormUpcoming* [s] über dem Wert *rWindSpeedStormUpcoming*, so wird davon ausgegangen, dass ein Sturm unmittelbar bevor steht. Erst wenn die Windgeschwindigkeit den Wert *rWindSpeedStormAbating* für die Zeit *uiDelayStormAbating* [s] unterschreitet, gilt der Sturm als abgeflaut und das Fahren der Jalousie als sicher. Aus Sicherheitsgründen wird auch das Sturm-Ereignis persistent gespeichert. Fällt die Steuerung also während eines Sturmes aus, so wird nach Wiederanlauf der Steuerung Abflau-Zeitgeber von Neuem Gestartet.

In beiden Gefahr-Fällen wird die Jalousie in die Schutzposition gefahren die durch *uiPositionProtection* (Höhenposition in Prozent) und *iAngleProtection* (Lamellenwinkel in Grad) vorgegeben wird.

VAR_INPUT

```
eDataSecurityType    : E_HVACDataSecurityType;
bEnable              : BOOL;
lrWindSpeed          : LREAL;
lrOutsideTemperature: LREAL;
bRainSensor          : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bEnable : Steht dieser Eingang auf FALSE, so ist der Baustein ohne Funktion. Im Positioniertelegramm stSunblind [▶ 242] werden für Position und Winkel jeweils 0 ausgegeben und *bActive* steht auf FALSE. Das bedeutet, dass über die Prioritätssteuerung eine andere Funktionalität die Jalousieansteuerung übernimmt.

lrWindSpeed: Windgeschwindigkeit. Die Einheit der Eingabe ist beliebig, jedoch ist es wichtig, dass es keine Werte kleiner als 0 gibt, und die Werte mit zunehmender Geschwindigkeit größer werden.

rOutsideTemperature: Außentemperatur in Grad Celsius.

bRainSensor: Eingang für einen Niederschlagssensor.

VAR_OUTPUT

```
stSunblind                : ST_BARSunblind;
bIceAlarm                 : BOOL;
udiRemainingTimeIceAlarm  : UDINT;
bStormAlarm               : BOOL;
```

```
uiRemainingTimeStormDetect: UINT;
uiRemainingTimeStormAlarm : UINT;
bError                    : BOOL;
udiErrorId                : UDINT;
```

stSunblind: Ausgabestruktur der Jalousiestellungen, siehe ST_BARSunblind [▶ 242].

bActive : Entspricht dem BOOL´schen Wert *bActive* im Jalousie-Telegramm ST_BARSunblind [▶ 242] und dient zur reinen Anzeige, ob der Baustein ein aktives Telegramm sendet.

bIceAlarm: Zeigt den Vereisungsalarm an.

udiRemainingTimeIceAlarm: Bei aufkommenden Vereisungsfall (*bIceAlarm*=TRUE) wird dieser Sekundenzähler auf die Enteisungszeit gesetzt. Sobald die Temperatur über dem eingetragenen Frostpunkt (*rFrostTemperature*) liegt, werden hier die verbleibenden Sekunden bis zur Entwarnung (*bIceAlarm*=FALSE) angezeigt. Solange kein Herunterzählen der Zeit stattfindet, steht dieser Ausgang auf 0.

bStormAlarm: Zeigt den Sturmalarm an.

uiRemainingTimeStormDetect: Im unkritischen Fall zeigt dieser Sekundenzähler konstant die Alarmverzögerungszeit *uiDelayStormUpcoming* an. Liegt die gemessene Windstärke *rWindSpeed* über dem Aktivierungsgrenzwert *rWindSpeedStormUpcoming*, so werden die Sekunden bis zum Alarm herunter gezählt. Solange kein Herunterzählen der Zeit stattfindet, steht dieser Ausgang auf 0.

uiRemainingTimeStormAlarm: Sobald der Sturmalarm ausgelöst wird, zeigt dieser Sekundenzähler zunächst konstant die Deaktivierungsverzögerungszeit *uiDelayStormAbating* des Sturmalarmes an. Sinkt die gemessene Windstärke *rWindSpeed* unter den Deaktivierungsgrenzwert *rWindSpeedStormAbating*, so werden die Sekunden bis zur Entwarnung (*bStormAlarm*=FALSE) herunter gezählt. Solange kein Herunterzählen der Zeit stattfindet, steht dieser Ausgang auf 0.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId: Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

Sollte ein Fehler anstehen, so wird diese Automatik deaktiviert und Position und Winkel auf 0 gesetzt. Das bedeutet, dass bei Verwendung einer Prioritätssteuerung automatisch eine andere Funktion niedrigerer Priorität (siehe Übersicht) die Steuerung der Jalousie übernimmt. Bei direkter Beschaltung hingegen wird die Jalousie auf Position/Winkel 0 fahren.

VAR_IN_OUT

Die Notwendigkeit, dass eingetragene Parameter über einen Steuerungsausfall hinweg erhalten bleiben, macht es erforderlich, dass sie als In-Out-Variablen deklariert werden. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variable wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

```
lrFrostTemperature       : LREAL;
lrWindSpeedStormUpcoming : LREAL;
lrWindSpeedStormAbating  : LREAL;
uiDeiceTime              : UINT;
uiDelayStormUpcoming     : UINT;
uiDelayStormAbating      : UINT;
uiPositionProtection     : UINT;
iAngleProtection         : INT;
```

lrFrostTemperature: Vereisungstemperatur-Grenzwert in Grad Celsius. Dieser Wert darf nicht größer als 0 sein. Anderenfalls wird ein Fehler ausgegeben.

lrWindSpeedStormUpcoming: Windgeschwindigkeits-Grenzwert zur Aktivierung des Sturmalarmes. Dieser Wert darf nicht kleiner als 0 sein und muss oberhalb des Wertes für die Deaktivierung liegen. Anderenfalls wird ein Fehler ausgegeben. Die Einheit der Eingabe muss die gleiche sein wie die des Einganges *rWindSpeed*. Ein Wert größer als dieser Grenzwert löst nach der eingetragenen Zeit *uiDelayStormUpcoming* den Alarm aus.

lrWindSpeedStormAbating: Windgeschwindigkeits-Grenzwert zur Deaktivierung des Sturmalarmes. Dieser Wert darf nicht kleiner als 0 sein und muss unterhalb des Wertes für die Aktivierung liegen. Anderenfalls wird ein Fehler ausgegeben. Die Einheit der Eingabe muss die gleiche sein wie die des Einganges *rWindSpeed*. Ein Wert kleiner oder gleich diesem Grenzwert setzt nach der eingetragenen Zeit *uiDelayStormAbating* den Alarm zurück.

uiDeiceTime: Zeit zum Abtauen der Jalousie nach Vereisung (in Minuten). Danach wird der Vereisungsalarm zurück gesetzt.

uiDelayStormUpcoming: Verzögerungszeit zur Auslösung des Sturmalarmes in Sekunden.

uiDelayStormUpcoming: Verzögerungszeit zum Rücksetzen des Sturmalarmes in Sekunden.

uiPositionProtection: Höhenposition der Jalousie in % im Schutzfall.

iAngleProtection: Lamellenwinkel der Jalousie in Grad im Schutzfall.

Dokumente hierzu

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

3.3.4.25 FB_BARSunProtectionEx

Baustein zur Blendschutzsteuerung mit Hilfe einer Lamellen-Jalousie.

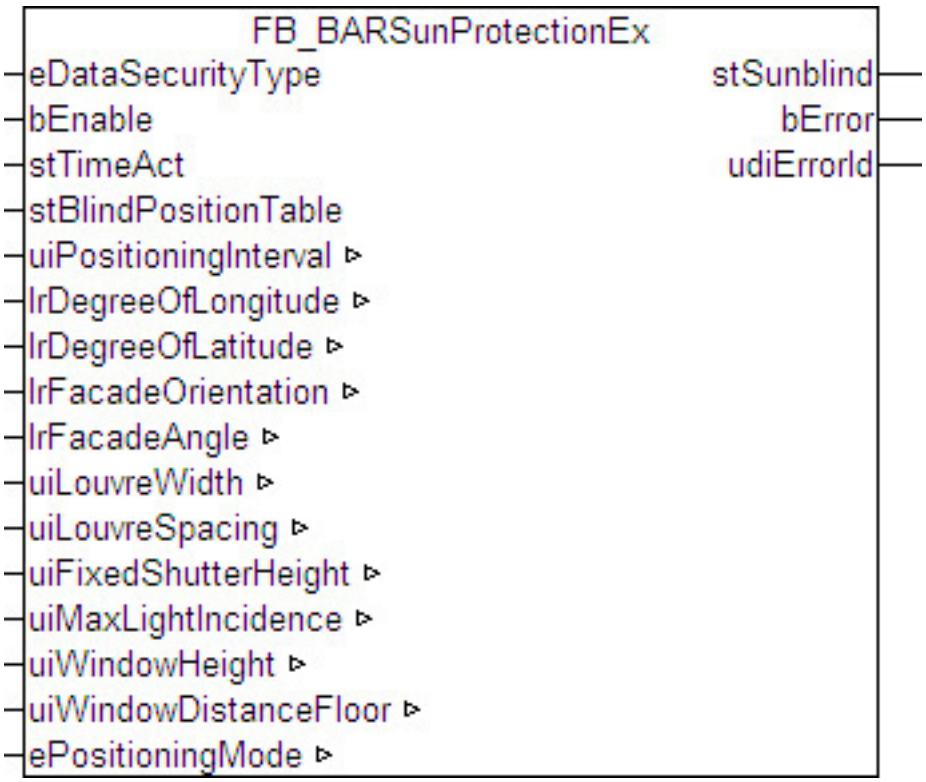


Abb. 10: FB_BARSunProtectionEx

Der Baustein FB_BARSunProtectionEx ermöglicht einen Blendschutz auf 2 verschiede parallel arbeitende Weisen:

- Lamellennachführung, so dass das direkt einfallende Licht durch den heruntergelassenen Teil der Jalousie gerade nicht mehr hindurch treten kann.

- Steuerung der Jalousiehöhe, mit 3 verschiedenen Möglichkeiten (Einstellbar über den Enumerator an *ePositioningMode*) :
  1) fixe Jalousiehöhe, d.h. keine Änderung (voreingestellt)
  2) Jalousiehöhe in Abhängigkeit des Sonnenstandes, definiert über eine Tabelle (stBlindPosTable [▶ 240]), siehe auch Beschreibung des FB_BARBlindPositionEntry [▶ 164]
  3) maximal erwünschter Lichteinfall

Anhand der eingetragenen Parameter, welche weiter unten beschrieben werden, errechnet der Baustein die notwendige Lamellenstellung und Jalousie-Position und übergibt sie in die Ausgabestruktur stSunblind [▶ 242]. Die Ausgabe erfolgt freilich nicht kontinuierlich, da eine ständige Jalousiebewegung als extrem störend empfunden werden würde. Am Eingang *uiPositioningInterval* lässt sich in Sekunden einstellen, in welchem Abstand neue Positionswerte ausgegeben werden sollen.
Jedoch muss zwischen zwei Positionierzeitpunkten die Verschattungskriterien immer erfüllt sein: kein direktes Licht darf durch die Lamellen treten und der erwünschte Lichteinfall durch die Jalousiehöhe muss begrenzt bleiben, geht man zunächst davon aus, dass die Jalousienhöhe über den Modus "maximal erwünschter Lichteinfall" gesteuert wird. Daher werden intern zwei Jalousie- und Lamellenstellungen errechnet: die für den jetzigen und die für den nächsten Schaltpunkt. Diejenige Stellung, bei der die Jalousie mehr geschlossen ist, ist dann die gültige.

Die Positionierung in Intervallen beginnt genau dann, wenn folgende 3 Bedingungen erfüllt sind:

- der Eingang *bEnable* muss auf TRUE stehen.
- der Baustein darf nicht durch falsche Parametrierung im Fehlerzustand sein (*bError*=TRUE).
- die Sonne muss aufgegangen sein, d.h. die Sonnenhöhe (Elevation) muss größer 0° sein. Hierbei handelt es sich um eine interne Sicherheitsabfrage, da die Begrenzung auf min. 0° vom Anwender eigentlich über die Programmierung des Einganges *bEnable* erfolgen soll, siehe Übersicht Sonnenschutzautomatik (Verschattungskorrektur).

Sind diese 3 Bedingungen nicht erfüllt, so wird in der Positionierstruktur das Aktiv-Bit (*bActive*) auf FALSE, die Jalousiehöhe auf 0% und der Lamellenwinkel auf 0% gestellt.

VAR_INPUT

```
eDataSecurityType    : E_HVACDataSecurityType;
bEnable              : BOOL;
stTimeAct            : TIMESTRUCT;
stBlindPositionTable: ST_BARBlindPositionTable;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eDataSecurityType_Persistentist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bEnable: Wenn dieser Eingang auf FALSE gesetzt wird, ist die Positionierung nicht aktiv, d.h. in der Positionier-Struktur stSunblind vom Typ ST_BARSunblind [▶ 242] wird das aktiv-Bit (*bActive*) zurückgesetzt und der Baustein selbst verharrt in einem Stillstands-Modus. Ist der Baustein hingegen aktiviert, so ist das aktiv-Bit auf TRUE und der Baustein gibt in der Positionierstruktur zu den entsprechenden Zeiten seine Stellwerte durch (*uiShutterHeight* ,*iLouvreAngle*).

stTimeAct: Eingabe der aktuellen Uhrzeit in GMT (Greenwich-Mean-Time).

stBlindPositionTable: Tabelle von 6 Stützpunkten, davon 4 parametrierbar, aus denen dann durch lineare Interpolation eine Jalousieposition in Abhängigkeit des Sonnenstandes gegeben wird. Gültig, wenn *ePositioningMode* = *ePosModeFixed* (siehe Enumerator E_BARPosMode [▶ 239]). Weitere Beschreibung siehe FB_BARBlindPositionEntry [▶ 164].

VAR_OUTPUT

```
stSunblind: ST_BARSunblind;
bError    : BOOL;
udiErrorId: UDINT;
```

stSunblind: Ausgabestruktur der Jalousiestellungen, siehe ST_BARSunblind

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId: Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

Sollte ein Fehler anstehen, so wird diese Automatik deaktiviert und Position und Winkel auf 0 gesetzt. Das bedeutet, dass bei Verwendung einer Prioritätssteuerung automatisch eine andere Funktion niedrigerer Priorität (siehe Übersicht) die Steuerung der Jalousie übernimmt. Bei direkter Beschaltung hingegen wird die Jalousie auf Position/Winkel 0 fahren.

VAR_IN_OUT

Die Notwendigkeit, dass eingetragene Parameter über einen Steuerungsausfall hinweg erhalten bleiben, macht es erforderlich, dass sie als In-Out-Variablen deklariert werden. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variable wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

```
uiPositioningInterva : UINT;
lrDegreeOfLongitude  : LREAL;
lrDegreeOfLatitude   : LREAL;
lrFacadeOrientation  : LREAL;
lrFacadeAngle        : LREAL;
uiLouvreWidth        : UINT;
uiLouvreSpacing      : UINT;
uiFixedShutterHeigh  : UINT;
uiMaxLightIncidence  : UINT;
uiWindowHeight       : UINT;
uiWindowDistanceFloor: UINT;
ePositioningMode     : E_BARPOSMODE;
```

uiPositioningInterval: Positionierintervall in Sekunden - Zeitspanne zwischen zwei Ausgaben von Jalousiestellungen. Gültiger Bereich: 1s..7200s.

lrDegreeOfLongitude: Geographische Länge (Längengrad) in Grad. Gültiger Bereich: -180°..180°.

lrDegreeOfLatitude: Geographische Breite (Breitengrad) in Grad. Gültiger Bereich: -90°..90°.

lrFacadeOrientation: Fassadenausrichtung in Grad:

Dabei gilt auf der Nordhalbkugel für die Fassadenausrichtung (Blick aus dem Fenster):

Blickrichtung	Fassadenausrichtung
Nord	β=0°
Ost	β=90°
Süd	β=180°
West	β=270°

Für die Südhalbkugel gilt hingegen:

Blickrichtung	Fassadenausrichtung
Süd	β=0°
Ost	β=90°
Nord	β=180°
West	β=270°

lrFacadeAngle: Fassadenneigung in Grad. Siehe Fassadenneigung [▶ 154].

uiLouvreWidth: Breite der Lamellen in mm, siehe Skizze [▶ 151].

uiLouvreSpacing: Lamellenabstand in mm, siehe Skizze [▶ 151].

uiFixedShutterHeight: Fixe (konstante) Jalousiehöhe [0..100%]. Gültig, wenn *ePositioningMode* = *ePosModeFixed* (siehe Enumerator E_BARPosMode [▶ 239]).

uiMaxLightIncidence: Maximal gewünschter Lichteinfall in mm gemessen ab Außenseite der Wand (siehe Höhenverstellung [▶ 151]) Mit Hilfe der Parameter *uiWindowHeight* und *uiWindowDistanceFloor* wird in Abhängigkeit des Sonnenstandes errechnet, wie hoch die Jalousie stehen muss, damit der Lichteinfall den Wert *uiMaxLightIncidence* nicht ´überschreitet. Gültig, wenn *ePositioningMode* = *ePosModeMaxIncidence* (siehe Enumerator E_BARPosMode [▶ 239]).

uiWindowHeight: Fensterhöhe in mm zur Errechnung des der Jalousiehöhe, wenn der Modus des "Maximal gewünschter Lichteinfalls" gewählt ist.

uiWindowDistanceFloor: Abstand Boden - Fenstersims in mm zur Errechnung des der Jalousiehöhe, wenn der Modus des "Maximal gewünschter Lichteinfalls" gewählt ist.

ePositioningMode: Auswahl des Positioniermodus, siehe Enumerator E_BARPosMode [▶ 239].

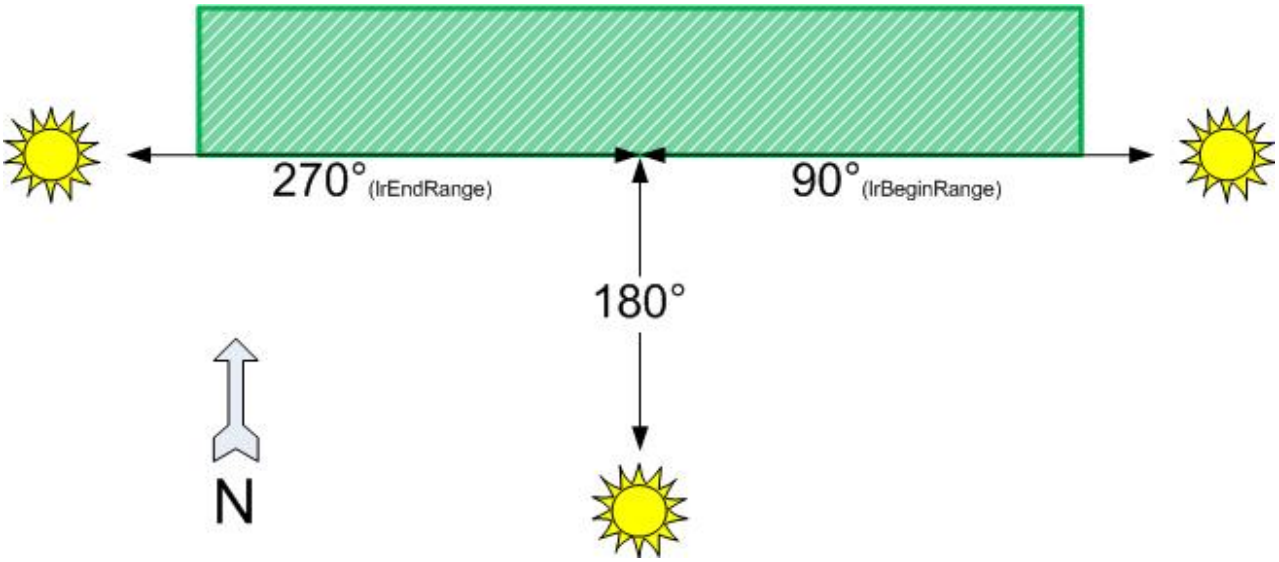
Dokumente hierzu

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

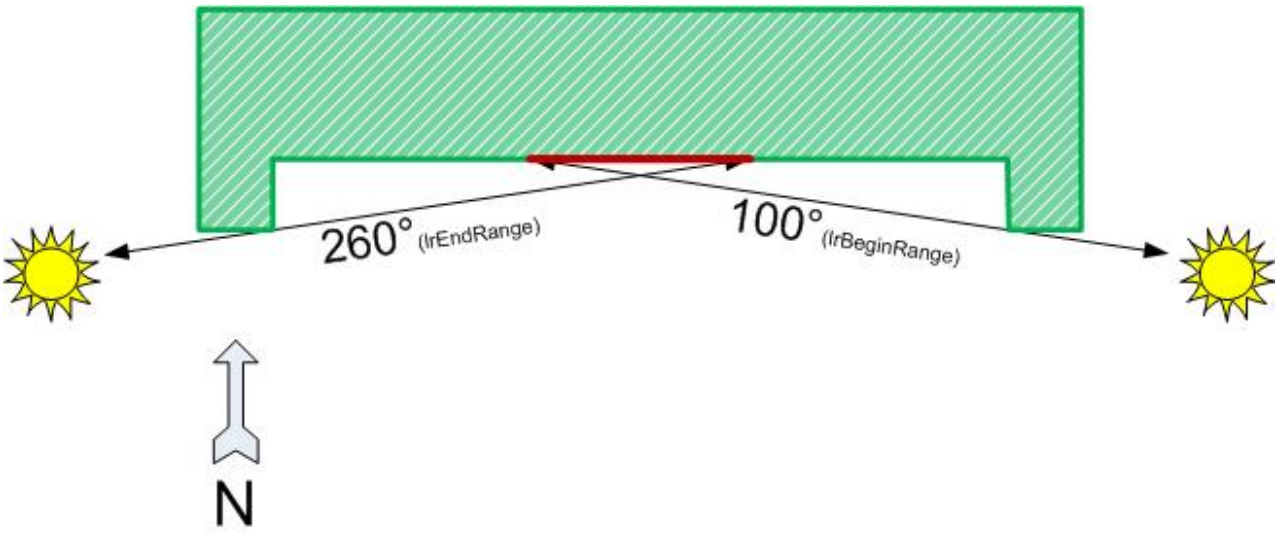
3.3.4.26 FB_BARWithinRangeAzimuth

Dieser Baustein prüft, ob der aktuelle Azimutwinkel (horizontaler Sonnenstand) innerhalb der eingetragenen Grenzen liegt. Wie in der Übersicht [▶ 150] erkennbar, gibt der Baustein eine zusätzliche Bewertung, ob der Sonnenschutz einer Fenstergruppe aktiviert werden soll. Daher gelten die Betrachtungen im weiteren Text immer für eine Fenstergruppe.

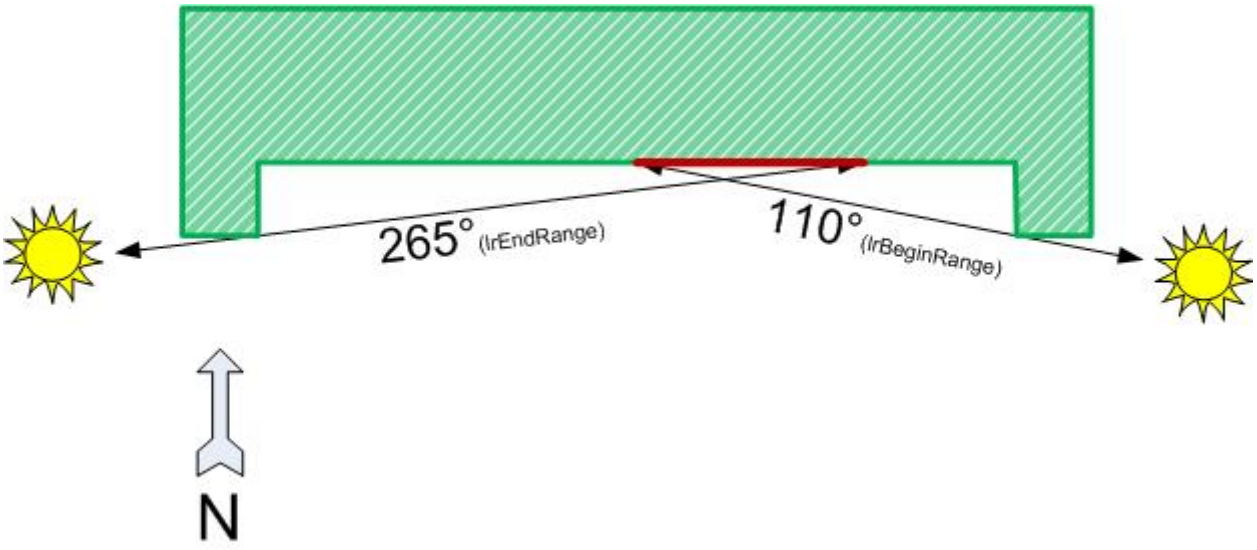
Eine glatte Fassade wird von der Sonne immer in einem Azimutwinkel von *Fassadenausrichtung-90°... Fassadenausrichtung+90°* bestrahlt.



Hat die Fassade jedoch seitliche Vorsprünge, so wird dieser Bereich eingeschränkt. Diese Einschränkung lässt sich mit Hilfe dieses Bausteines überprüfen. Dabei spielt aber auch die Lage der Fenstergruppe auf der Fassade eine Rolle. Liegt sie zentral, so ergibt sich folgende Situation (Die Werte sind dabei nur beispielhaft):



Für eine Gruppe am Rand andern sich die Werte:



Der Anfang des Bereiches *rBeginRange* darf dabei größer sein als das Ende *rEndRange*, es wird dann über 0° hinaus betrachtet:.

Beispiel:

lrAzimuth	10.0°
lrBeginRange	280.0°
lrEndRange	20.0°
bOut	**TRUE**

Der betrachtete Bereich darf jedoch nicht größer als 180° oder gleich 0° sein, dieses wäre unrealistisch. Derartige Eingaben ergeben einen Fehler am Ausgang *bError* - der Prüfausgang *bOut* wird dabei zusätzlich auf FALSE gesetzt.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
lrAzimuth        : LREAL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

lrAzimuth: Aktueller Azimutwinkel.

VAR_OUTPUT

```
bOut       : BOOL;
bError     : BOOL;
udiErrorId : UDINT;
```

bOut: Binärer verzögerter Ausgang des Schwellwertschalters

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId: Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

VAR_IN_OUT

Damit die eingetragenen Parameter über einen Steuerungsausfall hinweg erhalten bleiben ist es erforderlich, sie als In-Out-Variablen zu deklarieren. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variablen wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

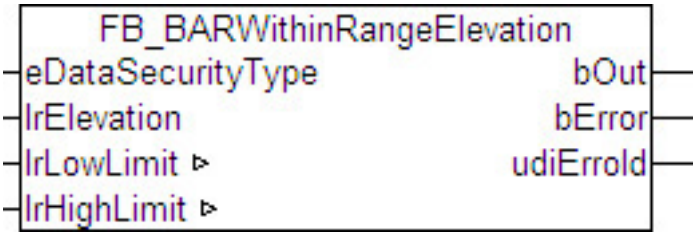
```
lrBeginRange : LREAL;
lrEndRange   : LREAL;
```

lrBeginRange: Bereichsanfang in Grad.

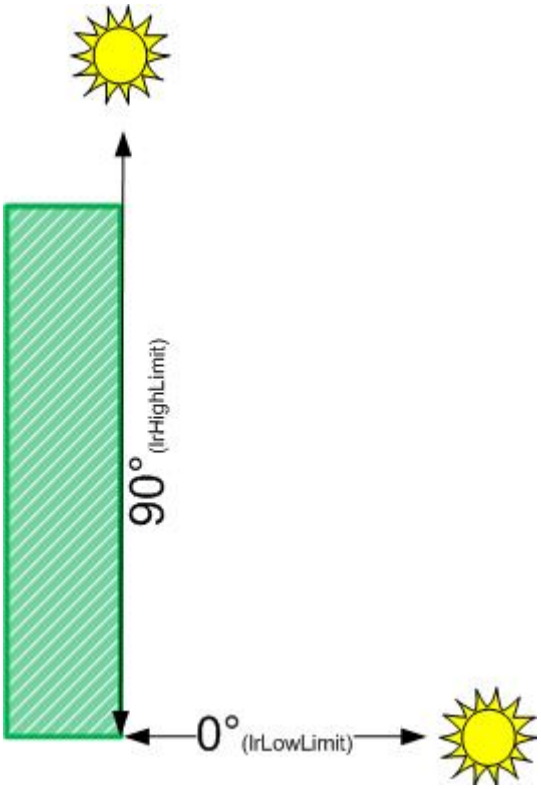
lrEndRange: Bereichsende in Grad.

3.3.4.27 FB_BARWithinRangeElevation

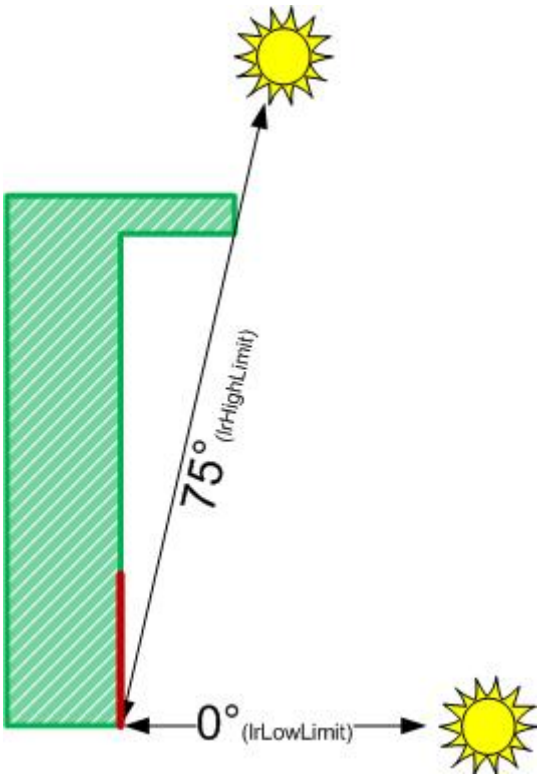
Dieser Baustein prüft, ob der aktuelle Elevationswinkel (vertikaler Sonnenstand) innerhalb der eingetragenen Grenzen liegt. Wie in der Übersicht [▶ 150] erkennbar, gibt der Baustein eine zusätzliche Bewertung, ob der Sonnenschutz einer Fenstergruppe aktiviert werden soll. Daher gelten die Betrachtungen im weiteren Text immer für eine Fenstergruppe.



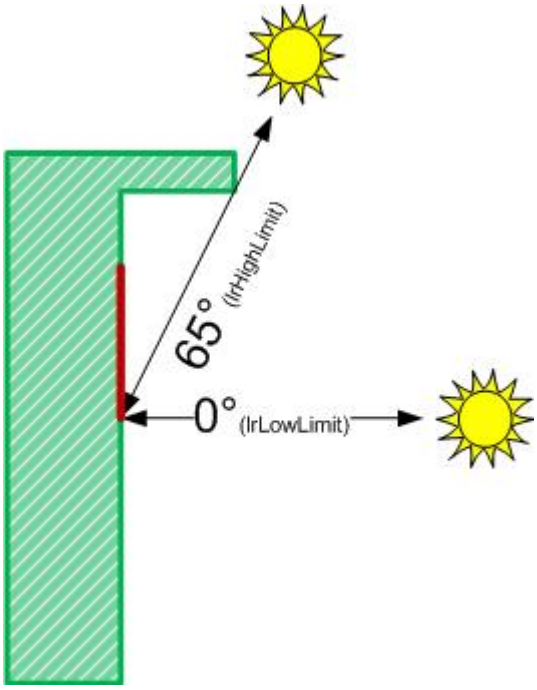
Eine normale senkrechte Fassade wird von der Sonne immer in einem Elevationswinkel von 0° bis maximal 90° bestrahlt.



Hat die Fassade jedoch Vorsprünge, so wird dieser Bereich eingeschränkt. Diese Einschränkung lässt sich mit Hilfe dieses Bausteines überprüfen. Dabei spielt aber auch die Lage der Fenstergruppe auf der Fassade eine Rolle. Liegt sie im unteren Bereich, so ergibt sich folgende Situation (Die Werte sind dabei nur beispielhaft):



Für eine Gruppe unterhalb des Vorsprunges ändern sich die Werte:



Die untere Betrachtungsgrenze, *rLowLimit*, darf dabei nicht größer oder gleich der oberen, *rHighLimit*, sein. Derartige Eingaben ergeben einen Fehler am Ausgang *bError* - der Prüfausgang *bOut* wird dabei zusätzlich auf FALSE gesetzt.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
lrElevation      : LREAL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

lrElevation: Aktueller Elevationswinkel in Grad.

VAR_OUTPUT

```
bOut       : BOOL;
bError     : BOOL;
udiErrorId : UDINT;
```

bOut: Binärer verzögerter Ausgang des Schwellwertschalters

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId: Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

VAR_IN_OUT

Damit die eingetragenen Parameter über einen Steuerungsausfall hinweg erhalten bleiben ist es erforderlich, sie als In-Out-Variablen zu deklarieren. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variablen wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

```
lrLowLimit   : LREAL;
lrHighLimit  : LREAL;
```

lrLowLimit: Unterer Grenzwert in Grad.

lrHighLimit: Oberer Grenzwert in Grad.

3.3.5 Programmbeispiele

3.3.5.1 Programmbeispiel FB_BARSunblindSwitch

Dieses Beispiel soll zeigen, wie eine mögliche Aktivierung (Eingang *bEnable*) der manuellen Jalousiesteuerung aussehen kann. Die Ein- und Ausgangsvariablen (umrandet) haben die folgende Bedeutung:

tAutoResetTime : Zeit, nach der die Aktivierung automatisch wieder zurück gesetzt werden soll.

bSwitchUp : Taste Jalousie hoch.

bSwitchDown : Taste Jalousie herunter.

bPauseClock : Pausensignal der Liegenschaft, z.B. Mittagspause: 12:00-13:00 Uhr. *bPauseClock* = TRUE: Pause.

bPresenceDetection : Belegt-Meldung des Raumes, in dem die Jalousien gesteuert werden. *bPresenceDetection*= TRUE: Raum ist belegt.

uiSwitchOverTime : Zeit in Millisekunden bis der entsprechende Handbefehl bei dauerhaft aktiviertem Befehlseingang in Selbsthaltung geht.

stSunblind : Positioniertelegramm, siehe ST_BARSunblind [▶ 242]. Zur weiteren Verknüpfung an eine Prioritätenauswahl, siehe FB_BARSunblindPrioritySwitch [▶ 203].

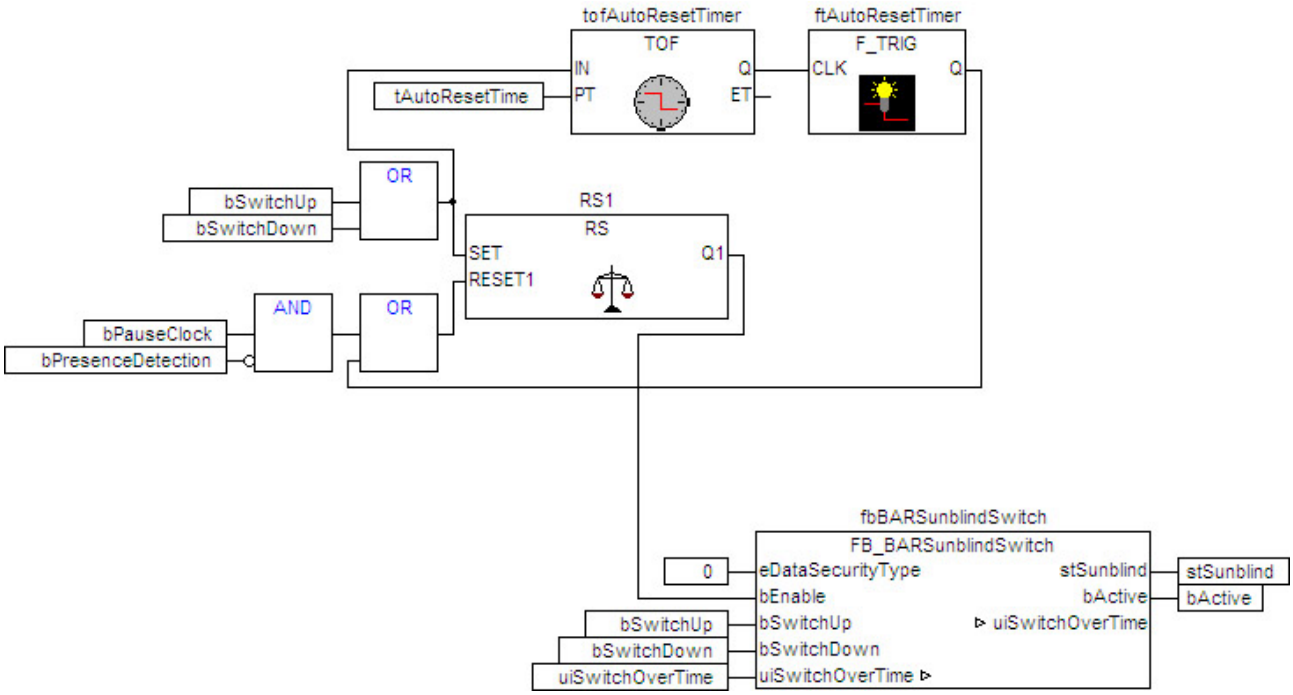
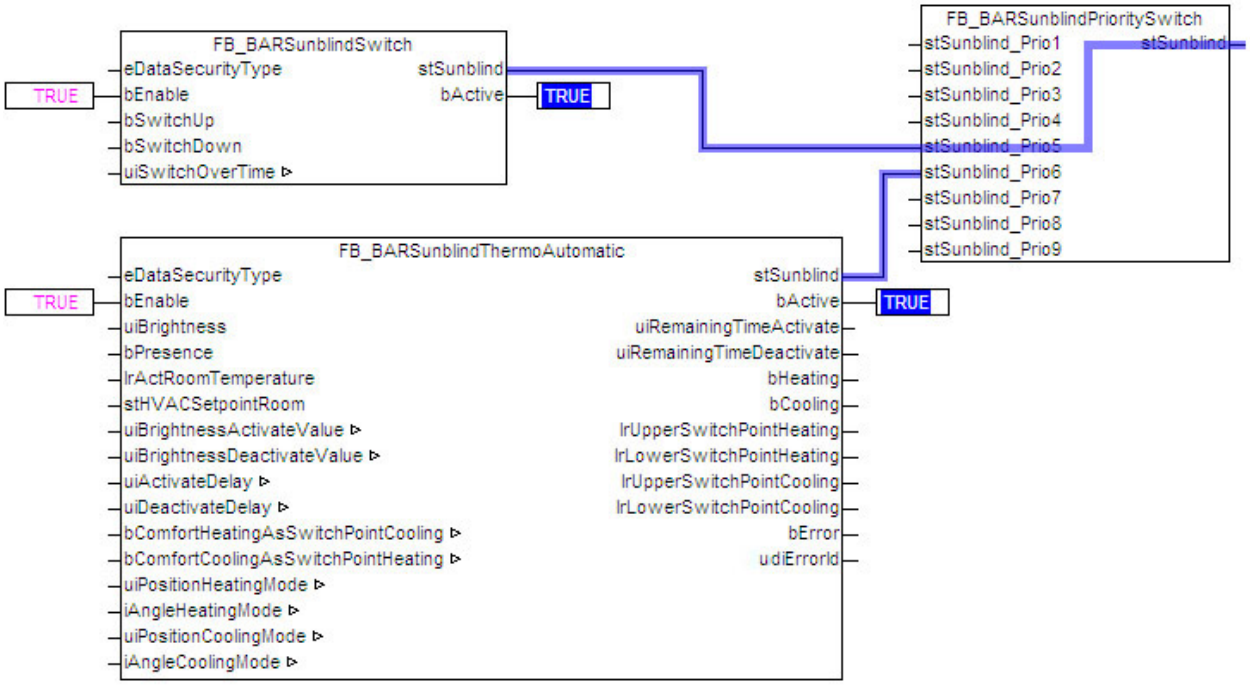


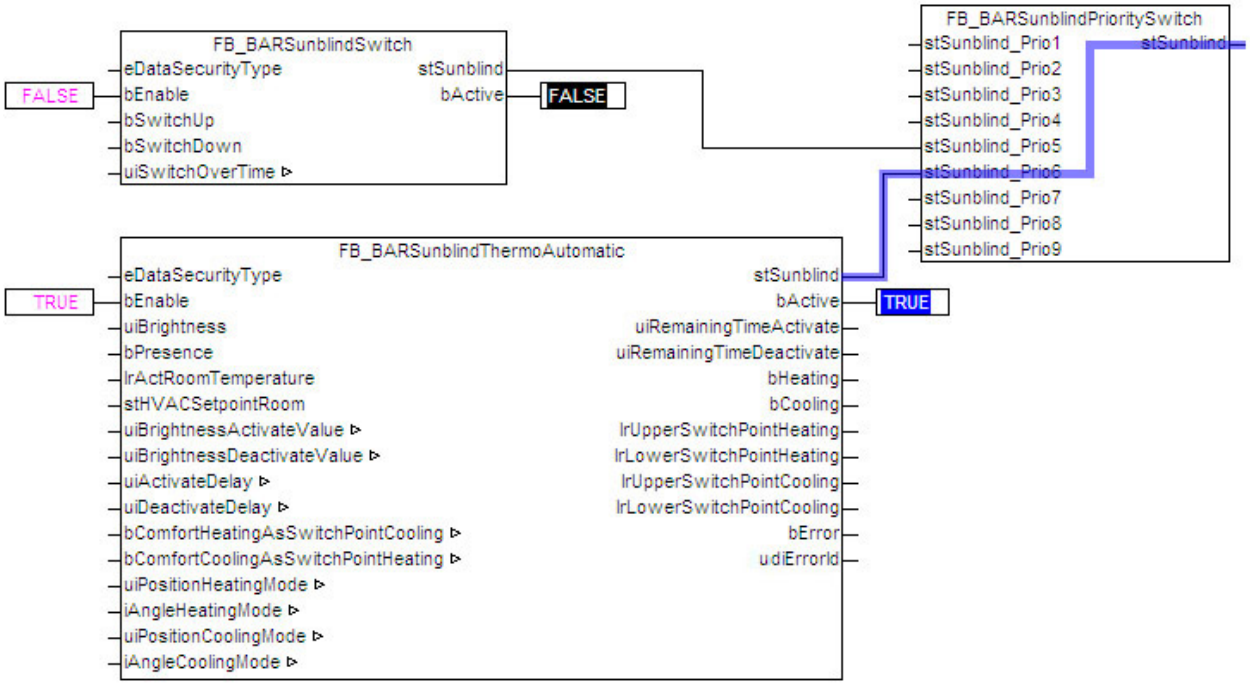
Abb. 11: SwitchWithResetEnable

Funktionsweise

Wird einer der Jalousietaster, *bSwitchUp* oder *bSwitchDown* betätigt, so wird zunächst der Ausgang des Flip-Flops auf TRUE geschaltet, jedoch nur dann, wenn nicht gerade ein Pausensignal (*bPauseClock*) anliegt bzw. der Raum belegt ist (*bPresenceDetection*). Damit ist der Eingang *bEnable* des Bausteins *fbBARSunblindSwitch* TRUE, was diesen aktiviert. Das jeweilige Tastersignal, welches zur Aktivierung geführt hat kann durch den Baustein unmittelbar im Befehlstelegramm *stSunblind* weiter gereicht werden. Gleichzeitig liegt am Eingang der Ausschaltverzögerung *tofAutoResetTimer* ein TRUE-Signal an und dessen Ausgang *Q* wird unmittelbar auf TRUE gesetzt. Mit dem Loslassen der Jalousietaste wird die Ausschaltverzögerung gestartet und nach Ablauf von *tAutoResetTime* fällt der Ausgang *Q* wieder auf FALSE. Diese fallende Flanke wird wiederum von *ftAutoResetTimer* erkannt und es wird ein Triggerimpuls gesendet, welcher das Flip-Flop wieder zurück setzt. Damit ist der Aktivierungseingang *bEnable* wieder auf FALSE und der Baustein *fbBARSunblindSwitch* passiv.

Ist dieser Baustein über das Befehlstelegramm an eine Prioritätenauswahl FB_BARSunblindPrioritySwitch [▶ 203] verknüpft, so wird dort das Telegramm der nächsten Priorität durchgereicht:





3.3.6 Strukturen und Aufzählungen

3.3.6.1 E_BARCtrlFct

```
TYPE E_BARCtrlFct:
(
    eBARCtrlFct_Off          := 0,
    eBARCtrlFct_Heating      := 1,
```

```
    eBARCtrlFct_Cooling     := 2
);
END_TYPE
```

eBARCtrlFct_Off: Function selection OFF acitve.

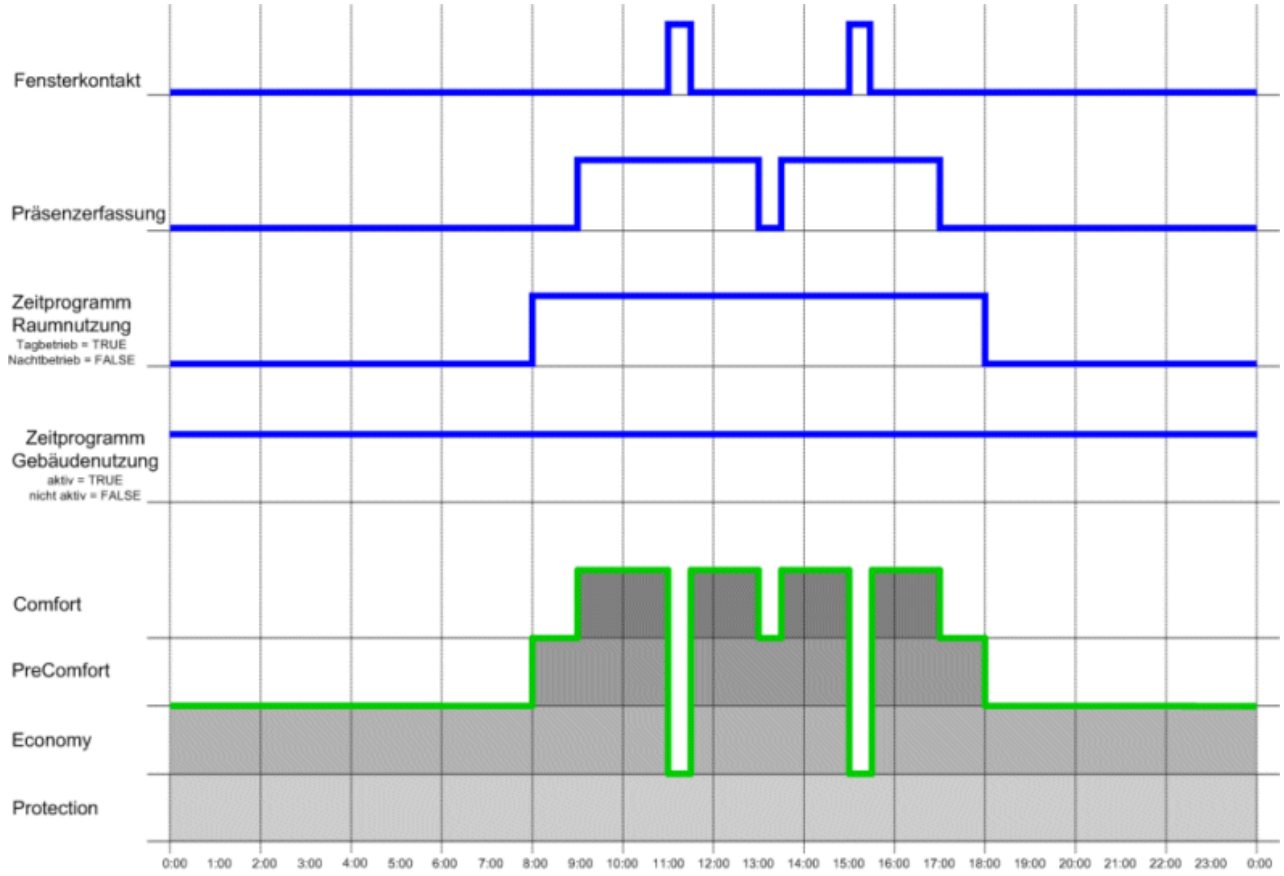
eBARCtrlFct_Heating: Function selection for heating operation active.

eBARCtrlFct_Cooling: Function selection for cooling operation active.

3.3.6.2 E_BAREnergyLevel

```
TYPE E_BAREnergyLevel:
(
    eBAREnergyLevel_AUTO_I     := 0,
    eBAREnergyLevel_Protection     := 1,
    eBAREnergyLevel_Economy    := 2,
    eBAREnergyLevel_PreComfort     := 3,
    eBAREnergyLevel_Comfort    := 4,
    eBAREnergyLevel_AUTO_II    := 5
);
END_TYPE
```

eBAREnergyLevel_AUTO_I: Automatik-Modus-I bedeutet, dass in dieser Betriebsart erst in das nächst höhere Energieniveau geschaltet werden kann, wenn die darunter liegenden Energieniveaus bereits TRUE sind. In dieser Betriebsart werden die Zustände der Eingänge *bProtection*, *bEconomy*, *bPreComfort* und *bComfort* ausgewertet für die Wahl des Energieniveaus. Siehe bitte Diagramm.



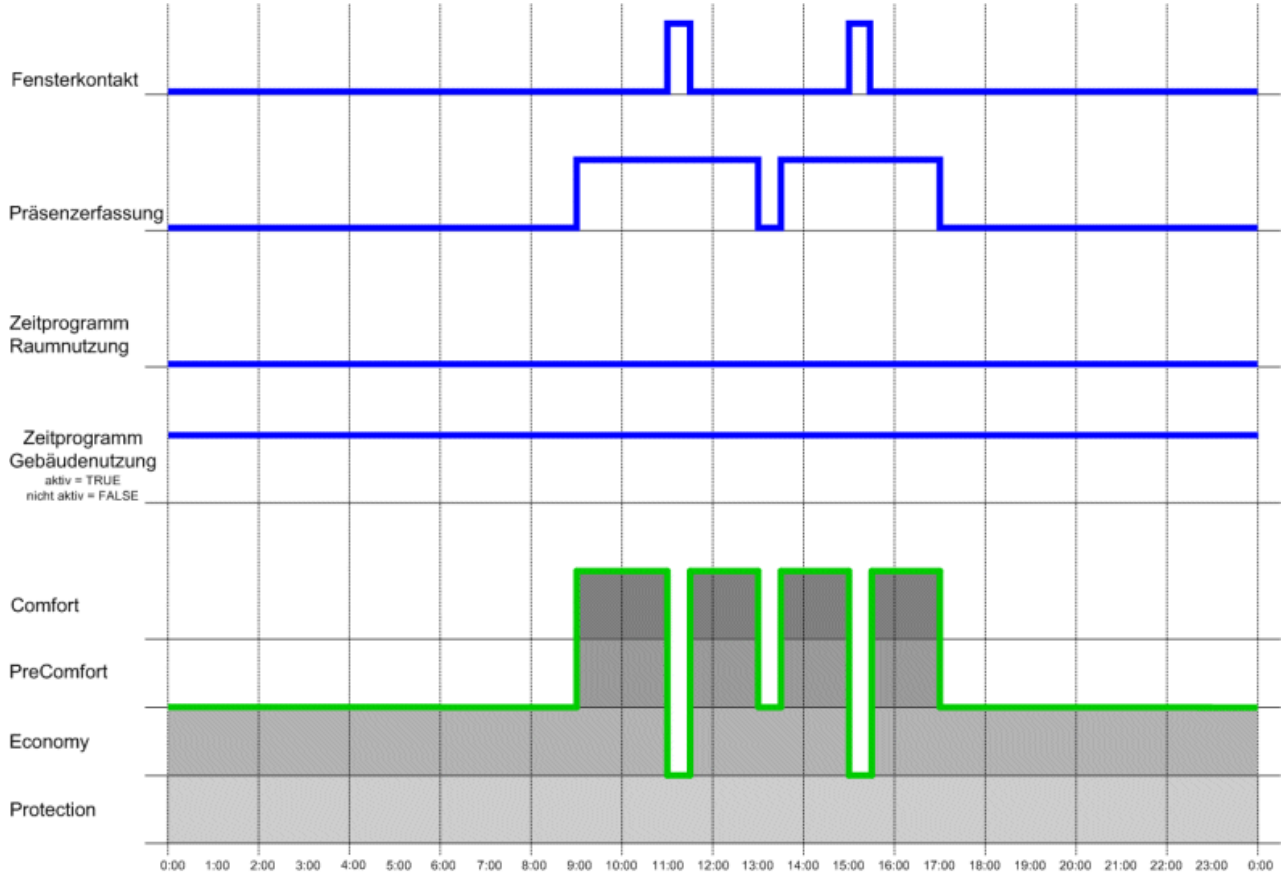
eBAREnergyLevel_Protection: In dieser Betriebsart, die von der Gebäudeleitebene manuell einstellbar ist, wird der Raum/Zone/Bereich in einem minimalen Energiezustand gehalten. Diese Betriebsart ist ein reiner Gebäudeschutzbetrieb.

eBAREnergyLevel_Economy: In dieser Betriebsart, die von der Gebäudeleitebene manuell einstellbar ist, wird der Raum/Zone/Bereich im Absenkbetrieb gehalten. Diese Betriebsart ist bei längeren Abwesenheitszeiten oder Nachts einzustellen.

eBAREnergyLevel_PreComfort: In dieser Betriebsart, die von der Gebäudeleitebene manuell einstellbar ist, wird der Raum/Zone/Bereich im Bereitschaftsbetrieb gehalten. Diese Betriebsart ist bei kurzen Abwesenheitszeiten einzustellen.

eBAREnergyLevel_Comfort: In dieser Betriebsart, die von der Gebäudeleitebene manuell einstellbar ist, wird der Raum/Zone/Bereich im Komfortbetrieb gehalten. Diese Betriebsart ist einzustellen bei Anwesenheit.

eBAREnergyLevel_AUTO_II: Automatik-Modus-II bedeutet, dass in dieser Betriebsart bei Präsenzerkennung direkt aus dem Energieniveau Protection, Economy oder PreComfort in das Energieniveau Comfort geschaltet wird. Siehe bitte Diagramm.



3.3.6.3 E_BARPosMode

Enumerator zur Definition des Positioniermodus.

```
TYPE E_BARPosMode :
(
     ePosModeFixed := 0,
     ePosModeTable,
     ePosModeMaxIncidence
);
END_TYPE
```

ePosModeFixed: Die Jalousiehöhe nimmt einen festen Wert ein, welcher am Baustein FB_BARSunProtectionEx [▶ 226] über den Wert *uiFixedShutterHeight* eingestellt wird (in %);

ePosModeTable: Die Höhenpositionierung erfolgt mit Hilfe einer Tabelle von 6 Stützpunkten, davon 4 parametrierbar. Aus diesen Punkten wird dann durch lineare Interpolation eine Jalousieposition in Abhängigkeit des Sonnenstandes errechnet. Weitere Beschreibung siehe FB_BARBlindPositionEntry [▶ 164].

ePosModeMaxIncidence: Die Positionierung erfolgt unter Angabe des maximal gewünschten Lichteinfalles.

3.3.6.4 E_BARShadingObjectType

Enumerator zur Auswahl des Verschattungsobjekt-Typs.

```
TYPE E_BARShadingObjectType :
(
     eObjectTypeTetragon  := 0,
     eObjectTypeGlobe     := 1
);
END_TYPE
```

eObjectTypeTetragon: Objekttyp ist ein Viereck.

eObjectTypeGlobe: Objekttyp ist eine Kugel.

3.3.6.5 E_HVACDataSecurityType

Enumeration zur Definition der Speicherung eingetragener Parameter.

```
TYPE E_HVACDataSecurityType :
(
eDataSecurityType_Persistent := 0,
eDataSecurityType_Idle       := 1
);
END_TYPE
```

eDataSecurityType_Persistent: Nach jeder Änderung werden die eingetragenen Parameter persistent gespeichert. Die Werte stehen an den Ein- bzw. Ausgabevariablen nach einem Neustart zur Verfügung.

eDataSecurityType_Idle: Die Werte werden nicht gespeichert. Nach einem Neustart stehen nur - falls vorhanden - Defaultwerte zur Verfügung.

3.3.6.6 ST_BARBlindPositionTable

Struktur der Stützpunkteinträge für die Höhenverstellung der Jalousie.

```
TYPE ST_BARBlindPositionTable:
STRUCT
     lrSunElevation   : ARRAY[0..5] OF LREAL;
     uiBlindPosition  : ARRAY[0..5] OF UINT;
     bValid       : BOOL;
END_STRUCT
END_TYPE
```

lrSunElevation / uiBlindPosition : Die 6 Stützstellen welche übergeben werden, wobei die Array-Elemente 0 und 5 die oben erwähnten automatisch generierten Rand-Elemente darstellen.

bValid : Gültigkeitsflag für den Baustein FB_BARSunProtectionEx [▶ 226]. Wird vom Baustein FB_BARBlindPositionEntry [▶ 164] auf TRUE gesetzt, wenn die eingegebenen Daten den beschriebenen Gültigkeitskriterien entsprechen.

3.3.6.7 ST_BARCorner

Information über Fenster-Eckpunkte

```
TYPE ST_BARCorner :
STRUCT
     lrX      : LREAL;
     lrY      : LREAL;
     bShaded  : BOOL;
END_STRUCT
END_TYPE
```

lrX: X-Koordinate des Fensters (auf der Fassade).

lrY: Y-Koordinate des Fensters (auf der Fassade).

bShaded: Information, ob dieser Eckpunkt verschattet ist: *bShaded*=TRUE: Eckpunkt ist verschattet.

3.3.6.8 ST_BARFacadeElement

Listeneintrag eines Fassadenelementes (Fenster).

```
TYPE ST_BARFacadeElement :
STRUCT
     usiGroup       : USINT;
     lrWindowWidth  : LREAL;
     lrWindowHeight : LREAL;
     stCorner       : ARRAY [1..4] OF ST_BARCorner;
     bValid     : BOOL;
END_STRUCT
END_TYPE
```

usiGroup: Gruppenzugehörigkeit des Elementes

lrWindowWidth: Breite des Fensters.

lrWindowHeight: Höhe des Fensters.

stCorner: Koordinaten der Fenster-Eckpunkte und Information, ob dieser Eckpunkt verschattet ist, siehe ST_BARCorner [▶ 240].

bValid: Plausibilität der eingetragenen Daten: *bValid*=TRUE: Daten sind plausibel.

3.3.6.9 ST_BARSetpointRoom

```
TYPE ST_BARSetpointRoom :
STRUCT
  stBARSetpointRoom_ComfortHeat       : REAL:= 21.0;
  stBARSetpointRoom_PreComfortHeat    : REAL:= 19.0;
  stBARSetpointRoom_EconomyHeat       : REAL:= 15.0;
  stBARSetpointRoom_ProtectionHeat    : REAL:= 12.0;
  stBARSetpointRoom_ComfortCool       : REAL:= 24.0;
  stBARSetpointRooom_PreComfortCool   : REAL:= 28.0;
  stBARSetpointRoom_EconomyCool       : REAL:= 35.0;
  stBARSetpointRoom_ProtectionCool    : REAL:= 40.0;
END_STRUCT
END_TYPE
```

Die Werte in der Struktur sind mit den Voreinstellwerten deklariert.

3.3.6.10 ST_BARShadingObject

Listeneintrag eines Verschattungsobjektes.

```
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 : Eckkoordinaten. Nur von Bedeutung, wenn das Element ein Viereck ist.

lrMx .. lrMz: Mittelpunktkoordinaten. Nur von Bedeutung, wenn das Element eine Kugel ist.

lrRadius: Radius der Kugel. Nur von Bedeutung, wenn das Element eine Kugel ist.

usiBeginMonth: Anfang der Verschattungsperiode (Monatszahl).

usiEndMonth: Ende der Verschattungsperiode (Monatszahl).

eType: Objekttyp, siehe E_BARShadingObjectType [▶ 240].

bValid: Plausibilität der Daten: bValid=TRUE: Daten sind plausibel.

Bemerkung zur Verschattungsperiode:

Die Monatseinträge dürfen nicht 0 und größer 12 sein, andernfalls sind alle Kombinationen möglich.
Beispiele:
Beginn=1, Ende=1: Verschattung im Januar.
Beginn=1, Ende=5: Verschattung von Anfang Januar bis Ende Mai.
Beginn=11, Ende=5: Verschattung von Anfang November bis Ende Mai (des folgenden Jahres).

3.3.6.11 ST_BARSunblind

Struktur des Jalousie-Positioniertelegramms.

```
TYPE ST_BARSunblind:
STRUCT
     uiPosition      : UINT;
     iAngle      : INT;
     bManUp      : BOOL;
     bManDown    : BOOL;
     bManualMode     : BOOL;
     bActive     : BOOL;
END_STRUCT
END_TYPE
```

uiPosition: Übergebene Jalousiehöhe in %.

iAngle: Übergebene Lamellenstellung in Grad.

bManUp: Handbefehl: Jalousie hoch.

bManDown: Handbefehl: Jalousie herunter.

bManualMode: TRUE: Der Handbedienmodus ist aktiv.

FALSE: Der Automatikmodus ist aktiv.

bActive: Der Absender des Telegramms ist aktiv. Dieses Bit wird nur von der Prioritätssteuerung FB_BARSunblindPrioritySwitch [▶ 203] ausgewertet. Die Sonnenschutzaktoren FB_BARSunblindActuator [▶ 189] und FB_BARRollerblind [▶ 204] beachteten es nicht.

3.3.6.12 ST_BARSunblindScene

Tabelleneintrag einer Jalousie-Szene.

```
TYPE ST_BARSunblindScene:
STRUCT
     uiPosition      : UINT;
     iAngle      : INT;
END_STRUCT
END_TYPE
```

uiPosition: Jalousiehöhe in %.

iAngle: Lamellenstellung in Grad.

3.3.7 Listenbeschreibungen

3.3.7.1 Liste der Fassadenelemente

Die Daten aller Fenster (Fassadenelemente) pro Fassade werden innerhalb des Programms in einem Feld von Strukturelementen des Typs ST_BARFacadeElement [▶ 241] hinterlegt.
Die Deklaration ist global, da der Verwaltungsbaustein FB_BARFacadeElementEntry [▶ 168] sowie die Verschattungskorrektur FB_BARShadingCorrection [▶ 180] / FB_BARShadingCorrectionSouth [▶ 183] direkt per Ein-/Ausgangsvariable auf dieses Feld zugreifen:

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

Die Variablen *iColumnsPerFacade* und *iRowsPerFacade* definieren dabei die Obergrenze der zur Verfügung stehenden Elemente und sind global als Konstante zu deklarieren:

```
VAR_GLOBAL CONSTANT
     iRowsPerFacade    : INT :=10;
     iColumnsPerFacade : INT :=20;
END_VAR
```

3.3.7.2 Liste der Verschattungselemente

Die Verschattungselemente pro Fassade werden innerhalb des Programms in einem Feld von Strukturelementen des Typs ST_BARShadingObject [▶ 241] hinterlegt.
Die Deklaration ist global, da der Verwaltungsbaustein FB_BARShadingObjectsEntry [▶ 186] sowie die Verschattungskorrektur FB_BARShadingCorrection [▶ 180] / FB_BARShadingCorrectionSouth [▶ 183] direkt per Ein-/Ausgangsvariable auf dieses Feld zugreifen:

```
VAR_GLOBAL
     arrShadingObject : ARRAY[1..iShadingObjects] OF ST_BARShadingObject;
END_VAR
```

Die Variable *iShadingObjects* stellt dabei die Obergrenze der zur Verfügung stehenden Elemente dar und ist global als Konstante zu definieren:

```
VAR_GLOBAL CONSTANT
     iShadingObjects  : INT := 20;
END_VAR
```

3.3.7.3 Fehlercodes

udiErrorId (hex)	udiErrorId (dez)	Fehler
0x0	0	Kein Fehler
0x8001	32769	FB_BARBlindPositionEntry [▶ 164]: Werte für *rSunElevation* sind nicht aufsteigend bzw. ungleich
0x8002	32770	FB_BARBlindPositionEntry [▶ 164]: Werte für *rSunElevation* sind nicht im gültigen Bereich von 0°..90°
0x8003	32771	FB_BARBlindPositionEntry [▶ 164]: Werte für *uiBlindPosition* sind nicht im gültigen Bereich von 0%..100%
0x8004	32772	FB_BARSunProtectionEx [▶ 226]: Die Dauer des Positionierintervalls ist gleich Null oder überschreitet 7200s.

udiErrorId (hex)	udiErrorId (dez)	Fehler
0x8005	32773	FB_BARSunProtectionEx [▶ 226]: Der eingetragene Längengrad ist nicht im gültigen Bereich von -180°..180°.
0x8006	32774	FB_BARSunProtectionEx [▶ 226]: Der eingetragene Breitengrad ist nicht im gültigen Bereich von -90°..90°.
0x8007	32775	FB_BARSunProtectionEx [▶ 226]: Der Wert für den Lamellenabstand (*uiLouvreSpacing)* ist größer oder gleich dem Wert für die Lamellenbreite (*uiLouvreWidth*).
0x8008	32776	FB_BARSunProtectionEx [▶ 226]: Das Bit "Werte gültig" (*bValid*) in der Positioniertabelle *stBlindPositionTable* ist nicht gesetzt - ungültige Werte, sieheFB_BARBlindPositionEntry [▶ 164].
0x8009	32777	FB_BARSunProtectionEx [▶ 226]: Der eingetragene Wert für die fixe Jalousiehöhe (*uiFixedShutterHeight*) überschreitet 100%
0x801A	32778	FB_BARSunProtectionEx [▶ 226]: Der eingetragene Wert für die Fensterhöhe ist gleich Null
0x801B	32779	FB_BARSunProtectionEx [▶ 226]: Der eingetragene Wert für den Lamellenabstand ist gleich Null
0x801C	32780	FB_BARSunProtectionEx [▶ 226]: Der eingetragene Wert für die Lamellenbreite ist gleich Null
0x801D	32781	FB_BARShadingObjectsEntry [▶ 186]: Die Summe der Winkel des Vierecks ist nicht 360°. Das bedeutet, die Eckpunkte sind nicht in der Reihenfolge P1, P2, P3 und P4 sondern P1, P3, P2 und P4. Dies ergibt ein über Kreuz geschlagenes Viereck.
0x801E	32782	FB_BARShadingObjectsEntry [▶ 186]: Die Eckpunkte des Vierecks liegen nicht auf der selben Ebene.
0x801F	32783	FB_BARShadingObjectsEntry [▶ 186]: Die z-Komponente von P1 ist kleiner Null. Damit läge dieser Eckpunkt hinter der Fassade.
0x8010	32784	FB_BARShadingObjectsEntry [▶ 186]: Die z-Komponente von P3 ist kleiner Null. Damit läge dieser Eckpunkt hinter der Fassade.

udiErrorId (hex)	udiErrorId (dez)	Fehler
0x8011	32785	FB_BARShadingObjectsEntry [▶ 186]: P1 ist gleich P2. Damit ist das eingetragene Objekt kein Viereck.
0x8012	32786	FB_BARShadingObjectsEntry [▶ 186]: P1 ist gleich P3. Damit ist das eingetragene Objekt kein Viereck.
0x8013	32787	FB_BARShadingObjectsEntry [▶ 186]: P1 ist gleich P4. Damit ist das eingetragene Objekt kein Viereck.
0x8014	32788	FB_BARShadingObjectsEntry [▶ 186]: P2 ist gleich P3. Damit ist das eingetragene Objekt kein Viereck.
0x8015	32789	FB_BARShadingObjectsEntry [▶ 186]: P2 ist gleich P4. Damit ist das eingetragene Objekt kein Viereck.
0x8016	32790	FB_BARShadingObjectsEntry [▶ 186]: P3 ist gleich P4. Damit ist das eingetragene Objekt kein Viereck.
0x8017	32791	FB_BARShadingObjectsEntry [▶ 186]: Der eingetragene Radius ist gleich Null
0x8018	32792	FB_BARShadingObjectsEntry [▶ 186]: Die z-Komponente des Kugelmittelpunktes ist kleiner Null. Damit läge dieser Punkt hinter der Fassade.
0x8019	32793	FB_BARFacadeElementEntry [▶ 168]: Der Gruppenindex ist 0, gleichzeitig ist jedoch ein anderer Eintrag des Fassadenelementes ungleich Null. Nur wenn alle Einträge eines Fassadenelementes Null sind, wird es als gültiges, gewollt ausgelassenes Fassadenelement angesehen, ansonsten jedoch als Fehleingabe interpretiert.
0x801A	32794	FB_BARFacadeElementEntry [▶ 168]: Die X-Komponente ersten Eckpunktes (Corner1) ist kleiner Null.
0x801B	32795	FB_BARFacadeElementEntry [▶ 168]: Die Y-Komponente ersten Eckpunktes (Corner1) ist kleiner Null.
0x801C	32796	FB_BARFacadeElementEntry [▶ 168]: Die Fensterbreite ist kleiner oder gleich Null.

udiErrorId (hex)	udiErrorId (dez)	Fehler
0x801D	32797	FB_BARFacadeElementEntry [▶ 168]: Die Fensterhöhe ist kleiner oder gleich Null.
0x801E	32798	FB_BARDelayedHysteresis [▶ 167]: Der Einschaltwert *bOnValue* ist kleiner als der Ausschaltwert *bOffValue*.
0x801F	32799	FB_BARWithinRangeAzimuth [▶ 229]: Bereichsanfang *rBeginRange* und Bereichsende *rEndRange* sind gleich.
0x8020	32800	FB_BARWithinRangeAzimuth [▶ 229]: Eine der eingetragenen Bereichsgrenzen ist größer als 360° oder kleiner als 0°
0x8021	32801	FB_BARWithinRangeAzimuth [▶ 229]: Der eingetragene Bereich ist größer als 180°.
0x8022	32802	FB_BARWithinRangeElevation [▶ 232]: Der eingetragene untere Grenzwert*rLowLimit* ist größer oder gleich dem oberen Grenzwert *rHighLimit* .
0x8023	32803	FB_BARWithinRangeElevation [▶ 232]: Der obere Grenzwert*rHighLimit* ist größer als 90° oder der untere Grenzwert *rLowLimit* kleiner als 0°.
0x8024	32804	FB_BARShadingObjectsEntry [▶ 186]: Die Monatszahl des Verschattungsbeginns oder des Verschattungsendes ist gleich Null oder größer als 12.
0x8025	32805	FB_BARShadingObjectsEntry [▶ 186]: Indexfehler! iId liegt außerhalb der zulässigen Grenzen *1..iShadingObjects*. Siehe Liste der Verschattungselemente [▶ 243].
0x8026	32806	FB_BARSunblindActuator [▶ 189]/ FB_BARSunblindActuatorEx [▶ 194]: Die eingetragene Jalousiehöhe ist 0.
0x8027	32807	FB_BARSunblindActuator [▶ 189]/ FB_BARSunblindActuatorEx [▶ 194]: Die gesamte Verfahrzeit "hoch" (*uiTotalTimeUp/udiTotalTimeUp*) bzw. die gesamte Verfahrzeit "herunter" (*uiTotalTimeDown/ udiTotalTimeDown*) ist 0ms.
0x8028	30808	FB_BARSunblindActuator [▶ 189]/ FB_BARSunblindActuatorEx [▶ 194]: Die Lamellen-Schwenkzeit "hoch"

udiErrorId (hex)	udiErrorId (dez)	Fehler
		(*uiTurningTimeUp*) bzw. die Lamellen-Schwenkzeit "herunter" (*uiTurningTimeDown*) ist 0ms.
0x8029	32809	FB_BARSunblindActuator [▶ 189]/ FB_BARSunblindActuatorEx [▶ 194]: Der obere Lamellenwinkel-Grenzwert (*iAngleLimitUp*) ist kleiner oder gleich dem unteren Lamellenwinkel-Grenzwert (*iAngleLimitDown*)
0x802A	32810	FB_BARSunblindScene [▶ 207]: Die ausgewählte Szenennummer überschreitet die Höchstzahl von 20.
0x802B	32811	FB_BARSunblindThermoAutomatic [▶ 214]: Die Rautemperatur-Vorgabewerte sind in einer falschen Reihenfolge.
0x802C	32812	FB_BARSunblindThermoAutomatic [▶ 214]: Der Helligkeits-Aktivierungs-Grenzwert ist kleiner oder gleich dem Deaktivierungs-Grenzwert.
0x802D	32813	FB_BARSunblindThermoAutomatic [▶ 214]: Die Jalousieposition (Höhe in %) vom Heizbetrieb ist größer oder gleich der des Kühlbetriebes.
0x802E	32814	FB_BARSunblindTwilightAutomatic [▶ 214]: Helligkeits-Grenzwert für die Aktivierung der Dämmerungsautomatik ist größer oder gleich dem Deaktivierungs-Wert.
0x802F	32815	FB_BARSunblindWeatherProtection [▶ 223]: Der Frost-Grenzwert ist größer als 10°C.
0x8030	32816	FB_BARSunblindWeatherProtection [▶ 223]: Der Grenzwert für die Aktivierung des Sturmalarms ist kleiner oder gleich dem Deaktivierungs-Wert.
0x8031	32817	FB_BARSunblindWeatherProtection [▶ 223]: Die Eingabe von Windstärke-Grenzwerten kleiner 0 ist nicht zulässig.
0x8032	32818	FB_BARShadingCorrection [▶ 180] / FB_BARShadingCorrectionSouth [▶ 183]: Der Gruppenindex ist 0. Dieser Index ist nur für Fassadenelemente vorgesehen, die als nicht existent markiert werden sollen, zum Beispiel wenn Fenster in einem gleichmäßigem

udiErrorId (hex)	udiErrorId (dez)	Fehler
		Fassadenraster ausgelassen werden oder eine Tür statt eines Fensters verbaut ist.
0x8033	32819	FB_BARShadingCorrection [▶ 180] / FB_BARShadingCorrectionSouth [▶ 183]: Ein Fensterelement der gewählten Gruppe ist als ungültig (bValid=FALSE) markiert.
0x8034	32820	FB_BARFacadeElementEntry [▶ 168]: Indexfehler! iColumn und/oder iRow liegen außerhalb der zulässigen Grenzen *1..iColumnsPerFacade* bzw. *1..iRowsPerFacade*. Siehe Liste der Fassadenelemente [▶ 243].
0x8035	32821	FB_BARSunProtectionEx [▶ 226]: Die eingetragene Fassadenneigung (lrFacadeAngle) ist nicht im gültigen Bereich von -90°..90°.
0x8036	32822	FB_BARSunblindActuator [▶ 189]/ FB_BARSunblindActuatorEx [▶ 194]: Die Zeit zum Ausfahren der Umkehrlose (*uiBacklashTimeUp*) bzw. (*uiBacklashTimeDown*) ist 0ms.
0x8037	32823	FB_BARRollerBlind [▶ 204]: Die gesamte Verfahrzeit "hoch" (*udiTotalTimeUp*) bzw. die gesamte Verfahrzeit "herunter" (*udiTotalTimeDown*) ist 0ms.
0x8038	32824	FB_BARSMISunblindActuator [▶ 200]: Der Wert für *wLouvreRange* darf nicht 0 betragen.
0x8039	32825	FB_BARSMISunblindActuator [▶ 200]: Verhältnis von *wLouvreRange* / (*iAngleLimitUp* + *iAngleLimitDown*) muss größer 0 sein.
0x803A.. 0x80FF	32826.. 33023	reserviert für Verschattungsbausteine
0x8100	33024	FB_BARLightCircuit [▶ 137]: Der Betriebsmodus,*uiLightCtrlMode*, ist größer 2. Es sind jedoch nur 0,1, 2 zulässig.
0x8101	33025	FB_BARLightCircuitDim [▶ 138]: Der Betriebsmodus,*uiLightCtrlMode*, ist größer 2. Es sind jedoch nur 0,1, 2 zulässig.

udiErrorId (hex)	udiErrorId (dez)	Fehler
0x8102	33026	FB_BARLightCircuitDim [▶ 138]: Der untere Grenzwert, *lrMinDimValue*, ist kleiner Null oder größer als 100%.
0x8103	33027	FB_BARLightCircuitDim [▶ 138]: Der obere Grenzwert, *lrMaxDimValue*, ist kleiner Null oder größer als 100%.
0x8104	33028	FB_BARLightCircuitDim [▶ 138]: Der untere Grenzwert, *lrMinDimValue*, ist größer als der obere Grenzwert, *lrMaxDimValue*.
0x8105	33029	FB_BARLightCircuitDim [▶ 138]: Der Wert für "Hand Ein", *lrManualDimValue*, ist kleiner Null oder größer als 100%.
0x8106	33030	FB_BARAutomaticLight [▶ 120]: Der Betriebsmodus,*uiLightCtrlMode*, ist größer 2. Es sind jedoch nur 0,1, 2 zulässig.
0x8107	33031	FB_BARAutomaticLight [▶ 120]: Der Wert für "Hand Ein", *lrManualDimValue*, ist kleiner Null oder größer als 100%.
0x8108	33032	FB_BARAutomaticLight [▶ 120]: Der Licht-Einschaltwert, *lrOnDimValue*, ist kleiner Null oder größer als 100%.
0x8109	33033	FB_BARStairwellAutomatic [▶ 142]: Der Betriebsmodus,*uiLightCtrlMode*, ist größer 2. Es sind jedoch nur 0,1, 2 zulässig.
0x810A	33034	FB_BARTwilightAutomatic [▶ 145]: Der Betriebsmodus,*uiLightCtrlMode*, ist größer 2. Es sind jedoch nur 0,1, 2 zulässig.
0x810B	33035	FB_BARTwilightAutomatic [▶ 145]: Der Wert für "Hand Ein", *lrManualDimValue*, ist kleiner Null oder größer als 100%.
0x810C	33036	FB_BARTwilightAutomatic [▶ 145]: Der EIN-Wert für der Automatik, *lrDimOnValue*, ist kleiner Null oder größer als 100%.
0x810D	33037	FB_BARTwilightAutomatic [▶ 145]: Der AUS-Wert für der Automatik, *lrDimOffValue*, ist kleiner Null oder größer als 100%.
0x810E	33038	FB_BARTwilightAutomatic [▶ 145]: Der Einschaltschwellwert, *uiSwitchOnValue*, ist größer oder gleich dem Ausschaltschwellwert, *uiSwitchOffValue*.

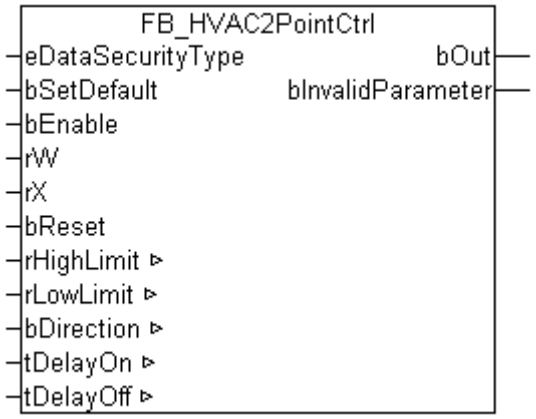
udiErrorId (hex)	udiErrorId (dez)	Fehler
0x810F	33039	FB_BARDaylightControl [▶ 130]: Der Betriebsmodus,*uiLightCtrlMode*, ist größer 2. Es sind jedoch nur 0,1, 2 zulässig.
0x8110	33040	FB_BARConstantLightControl [▶ 123]: Der Betriebsmodus,*uiLightCtrlMode*, ist größer 2. Es sind jedoch nur 0,1, 2 zulässig.
0x8111	33041	FB_BARConstantLightControl [▶ 123]: Der untere Grenzwert, *lrMinDimValue*, ist kleiner Null oder größer als 100%.
0x8112	33042	FB_BARConstantLightControl [▶ 123]: Der obere Grenzwert, *lrMaxDimValue*, ist kleiner Null oder größer als 100%.
0x8113	33043	FB_BARConstantLightControl [▶ 123]: Der untere Grenzwert, *lrMinDimValue*, ist größer als der obere Grenzwert, *lrMaxDimValue*.
0x8114	33044	FB_BARConstantLightControl [▶ 123]: Der Startwert der Regelung, lrStartDimValue, ist kleiner Null oder größer als 100%.
0x8115	33045	FB_BARConstantLightControl [▶ 123]: Der Wert für "Hand Ein", *lrManualDimValue*, ist kleiner Null oder größer als 100%.
0x8116.. 0x81FF	33046.. 33279	reserviert für Lichtbausteine
0x8200	33280	FB_BARPICtrl [▶ 115]: Die Ausführungszyklusnummer *uiCtrlCycleCall* ist gleich Null
0x8201	33281	FB_BARPICtrl [▶ 115]: Das Proportionalband *rXp* ist gleich Null .
0x8202	33282	FB_BARPICtrl [▶ 115]: Der untere Stellgrößen-Grenzwert *rYMin*, ist größer oder gleich dem oberen Grenzwert *rYMax*.
0x8203	33283	FB_BARPICtrl [▶ 115]: Die Task-Zykluszeit *tTaskCycleTime* ist auf 0 eingestellt.
0x8204	33284	FB_HVACOptimizedOn [▶ 439]: Die Außentemperaturwerte *rOutsideTemp[i]* der Vorstartfunktion *stTempChangeFunction* sind nicht in aufsteigender Reihenfolge oder 2 Werte sind gleich.

udiErrorId (hex)	udiErrorId (dez)	Fehler
0x8205	33285	FB_HVACOptimizedOn [▶ 439]: Ein oder mehrere Werte der Raumtemperaturänderung *rRoomTempChange[i]* in der Vorstartfunktion *stTempChangeFunction* sind größer als der obere Grenzwert *rMaxTempChange*.
0x8206	33286	FB_HVACOptimizedOn [▶ 439]: Ein oder mehrere Werte der Raumtemperaturänderung *rRoomTempChange[i]* in der Vorstartfunktion *stTempChangeFunction* sind kleiner als der untere Grenzwert *rMinTempChange*.
0x8207	33287	FB_HVACOptimizedOn [▶ 439]: Der Steuereingang *eCtrlFct* ist weder auf Heiz- noch auf Kühlmodus eingestellt (*eBARCtrlFct_Heating* oder *eBARCtrlFct_Cooling*).
0x8208	33288	FB_HVACOptimizedOff [▶ 448]: Die Außentemperaturwerte *rOutsideTemp[i]* der Vorstartfunktion *stTempChangeFunction* sind nicht in aufsteigender Reihenfolge oder 2 Werte sind gleich.
0x8209	33289	FB_HVACOptimizedOff [▶ 448]: Ein oder mehrere Werte der Raumtemperaturänderung *rRoomTempChange[i]* in der Vorstartfunktion *stTempChangeFunction* sind größer als der obere Grenzwert *rMaxTempChange*.
0x820A	33290	FB_HVACOptimizedOff [▶ 448]: Ein oder mehrere Werte der Raumtemperaturänderung *rRoomTempChange[i]* in der Vorstartfunktion *stTempChangeFunction* sind kleiner als der untere Grenzwert *rMinTempChange*.
0x820B	33291	FB_HVACOptimizedOff [▶ 448]: Der Steuereingang *eCtrlFct* ist weder auf Heiz- noch auf Kühlmodus eingestellt (*eBARCtrlFct_Heating* oder *eBARCtrlFct_Cooling*).
0x820C	33292	FB_HVACTempChangeFunctionEntry [▶ 459]: Die Außentemperaturwerte *rOutsideTemp[i]* der Vorstartfunktion *stTempChangeFunction* sind nicht in aufsteigender Reihenfolge oder 2 Werte sind gleich.

udiErrorId (hex)	udiErrorId (dez)	Fehler
0x8C00.. 0x8CFF	35840.. 36095	Siehe SMI Fehlercodes 0x0C00, um den SMI spezifischen Fehlercode zu erhalten.

3.4 HLK Regler

3.4.1 FB_HVAC2PointCtrl



Anwendung

Dieser Funktionsbaustein stellt einen 2-Punkt Regler dar. Über *bEnable* = TRUE wird der Regler freigegeben und ist aktiv.

VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Über ein TRUE wird der Regler freigegeben und ist aktiv.

rW: Mit der Variablen *rW* wird der Sollwert übergeben.

rX: Mit der Variablen *rX* wird der Istwert des Regelkreises übergeben.

bReset: Quittierungseingang bei einer Störung.

VAR_OUTPUT

```
bOut                : BOOL;
bInvalidParameter   : BOOL;
```

bOut: Schaltausgang des Zweipunktreglers.

bInvalidParameter: TRUE, wenn bei der Plausibilitätsüberprüfung ein Fehler aufgetreten ist. Die Meldung muss mit *bReset* quittiert werden.

VAR_IN_OUT

```
rHighLimit      : REAL;
rLowLimit       : REAL;
bDirection      : BOOL;
tDelayOn        : TIME;
tDelayOff       : TIME;
```

rHighLimit: Obere Grenze der Regelabweichung (0..32767). Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rLowLimit: Untere Grenze der Regelabweichung (0..32767). Die Variable wird persistent gespeichert. Voreingestellt auf 0.

bDirection: Mit *bDirection* wird der Wirksinn des Reglers bestimmt. FALSE = Heizbetrieb; TRUE = Kühlbetrieb. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

tDelayOn: Einschaltverzögerung [s]. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

tDelayOff: Ausschaltverzögerung [s]. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

Zeitliches Verhalten

Abbildung 1:

Verlauf des Schaltausgangs *bOut,* bei einer Zweipunktregelung, in Abhängigkeit der Ober- / Untergrenze der Regelabweichung ohne Ein- / Ausschaltverzögerung.

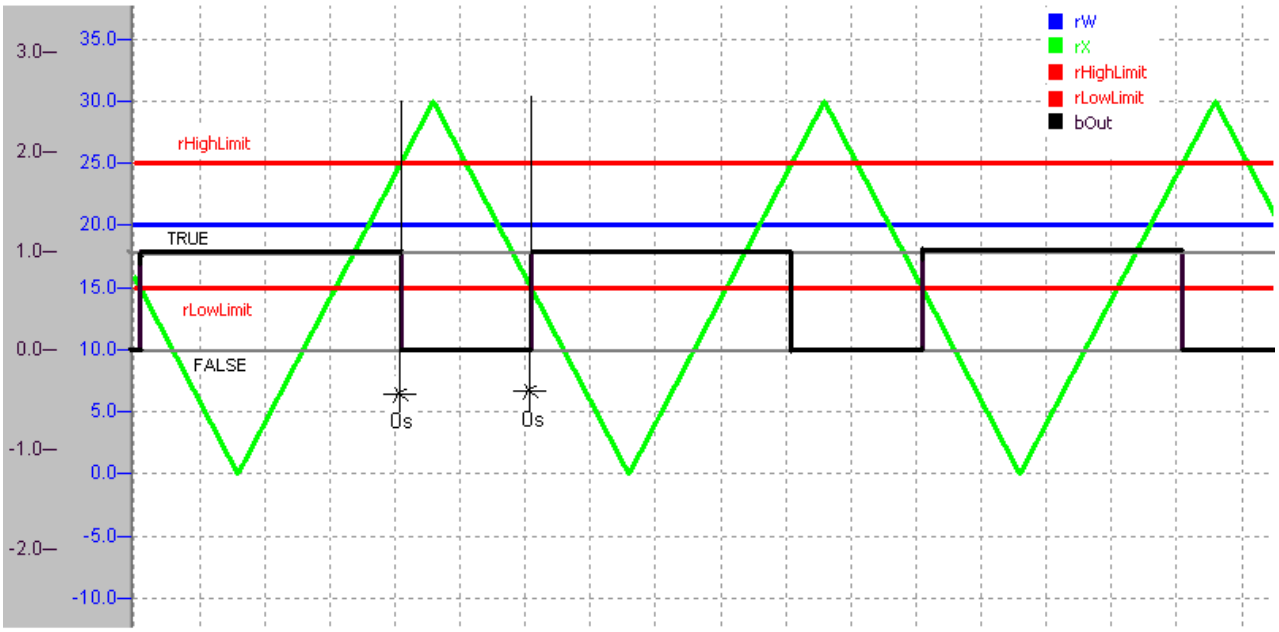


Figure: FB_HVAC2PointCtrl timing diagram

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

Abbildung 2:

Verlauf des Schaltausganges *bOut* bei einer Zweipunktregelung in Abhängigkeit der Obergrenze *rHighLimit* = 3 und der Untergrenze *rLowLimit* = 8 mit einer Einschaltverzögerung von *tDelayOn* = 4s und einer Ausschaltverzögerung von *tDelayOff* = 6s.

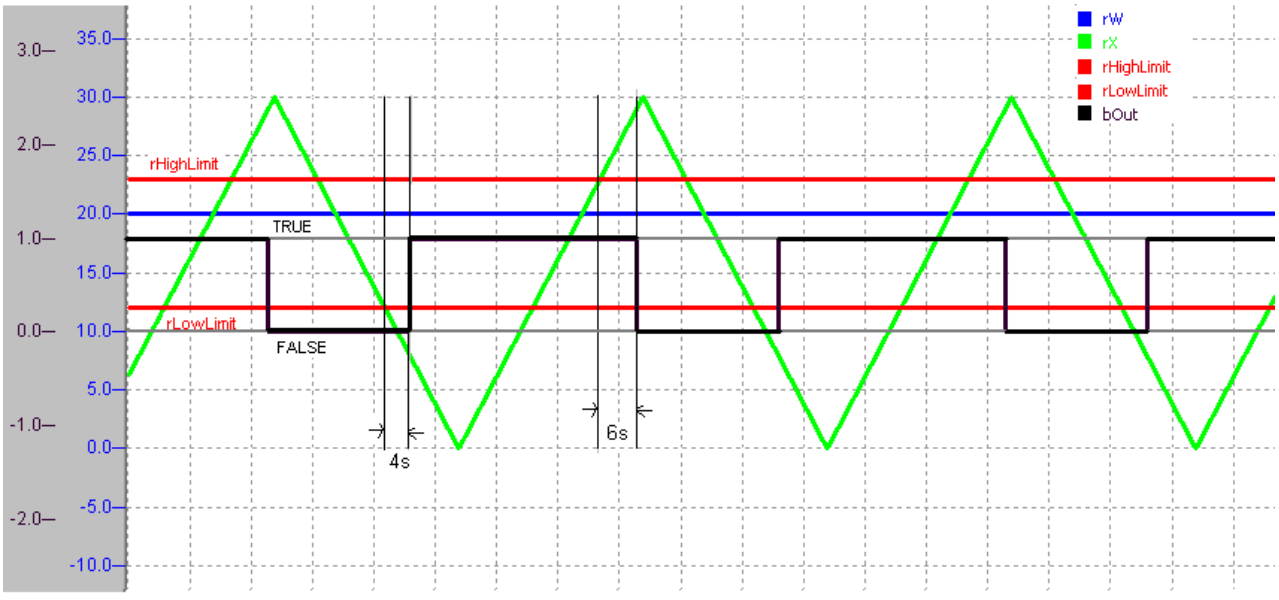


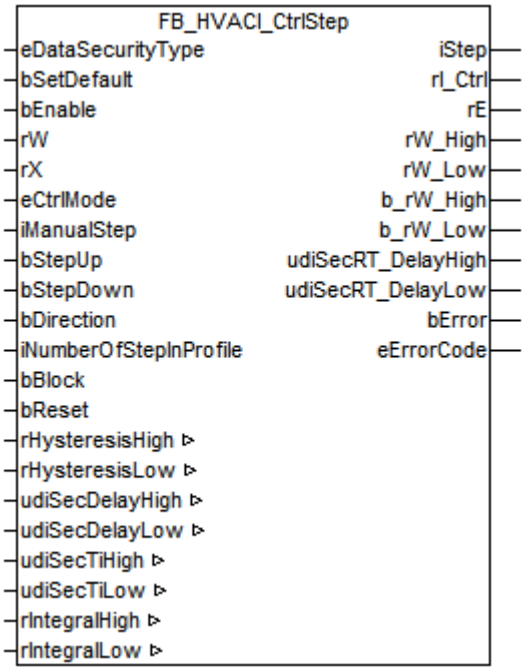
Figure: FB_HVAC2PointCtrl timing diagram

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

Dokumente hierzu

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

3.4.2 FB_HVACI_CtrlStep



Anwendung

Anwendung

Der Funktionsbaustein dient zur sequentiellen Folgesteuerung von Leistungserzeugern.

In Verbindung mit der Leistungsstufentabelle FB_HVACPowerRangeTable [▶ 281] kann der Leistungsstufenregler für die stufige Regelung mehrerer Heizkessel, Kältemaschinen oder Rückkühlwerke eingesetzt werden.

Das Hoch- oder Runterschalten in die nächst höhere oder niedrigere Leistungsstufe erfolgt über ein Integral (rW - rX) und einer Zeitverzögerung. Zuerst muss der Temperaturistwert *rX* einen Schwellwert *rW_High* / *rW_Low* über- oder unterschritten haben. Anschliessend startet das Integral. Wird an dem Ausgang *rI_Ctrl* der obere oder untere Grenzwert *(rIntegralHigh / rIntegralLow)* erreicht, so wird ein Zeitglied gestartet, nach dessen Ablauf (*udiSecRT_DelayHigh / udiSecRT_DelayLow)*der Ausgang *iStep* in- oder dekrementiert wird, siehe Programmablaufdarstellung als Diagramm [▶ 258]

Eine eingestellte Integrationszeit udiSecTiHigh/udiSecTiLow = 60 Sekunden heisst, dass das I-Übertragungsglied sich um ein Kelvin pro Minute ändert. Bei einer Regelabweichung rE = 5 hat der Ausgang des I-Übertragungsgliedes rI_Ctrl nach einer Minute den Wert 5. Beispiel: bDirection = FALSE; udiSecTiLow = 60; rE = 2; rIntegralLow = 10 Bei der bleibenden Regelabweichung von 2 und der unteren Integralgrenze von 10 ist nach 5 Minuten rI_Ctrl = rIntegralLow. Dieses hat zur Folge, dass ein Zeitglied mit der Verzögerungszeit udiSecDelayLow gestartet wird. Nach dessen Ablauf (udiSecRT_DelayHigh / udiSecRT_DelayLow) wird der Ausgang iStep um 1 inkrementiert und das I-Übertragungsglied und die Zeitverzögerungen werden zurück gesetzt.

iStep wird für einen SPS-Zyklus auf 1 gesetzt, wenn bEnable = TRUE, bError = FALSE, eCtrlMode = eHVACCtrlMode_AutoANDiNumberOfStepInProfile > 0 ist.

Bedingungen

Das I-Übertragungsglied wird freigegeben, wenn

bEnable = TRUEAND NOT*bError*AND*eCtrlMode* = *eHVACCtrlMode_Auto*AND NOT*bBlock*
(
(*rX* >= *rW_High*AND ((*iStep* > 0 ANDNOT*bDirection*) OR (*iStep* < *iNumberOfStepInProfile*AND*bDirection*))
OR
(*rX* < *rW_Low*AND ((*iStep* < *iNumberOfStepInProfile*ANDNOT*bDirection*) OR (*iStep* > 0 AND*bDirection*))
)

ist. Ist *rI_Ctrl =rIntegralHigh / rIntegralLow*, so werden die Zeitglieder der Verzögerungszeiten *udiSecDelayHigh / udiSecDelayLo*w aktiviert. Nach Ablauf einer der Verzögerungszeiten *udiSecRT_DelayHigh* / *udiSecRT_DelayLow* werden die Stufen der Leistungsstufensequenz folgendermaßen gesteuert:

iStep = *iStep* + 1 wenn

(*rX* < *rW_Low*AND*iStep* < *iNumberOfStepInProfile*ANDNOT*bDirection*)
OR
(*rX* >= *rW_High*AND*iStep* < *iNumberOfStepInProfile*AND*bDirection*)

iStep = *iStep* - 1 wenn

(*rX* < *rW_Low*AND*iStep* > 0 AND*bDirection*)
OR
(*rX* >= *rW_High*AND*iStep* > 0 ANDNOT*bDirection*)

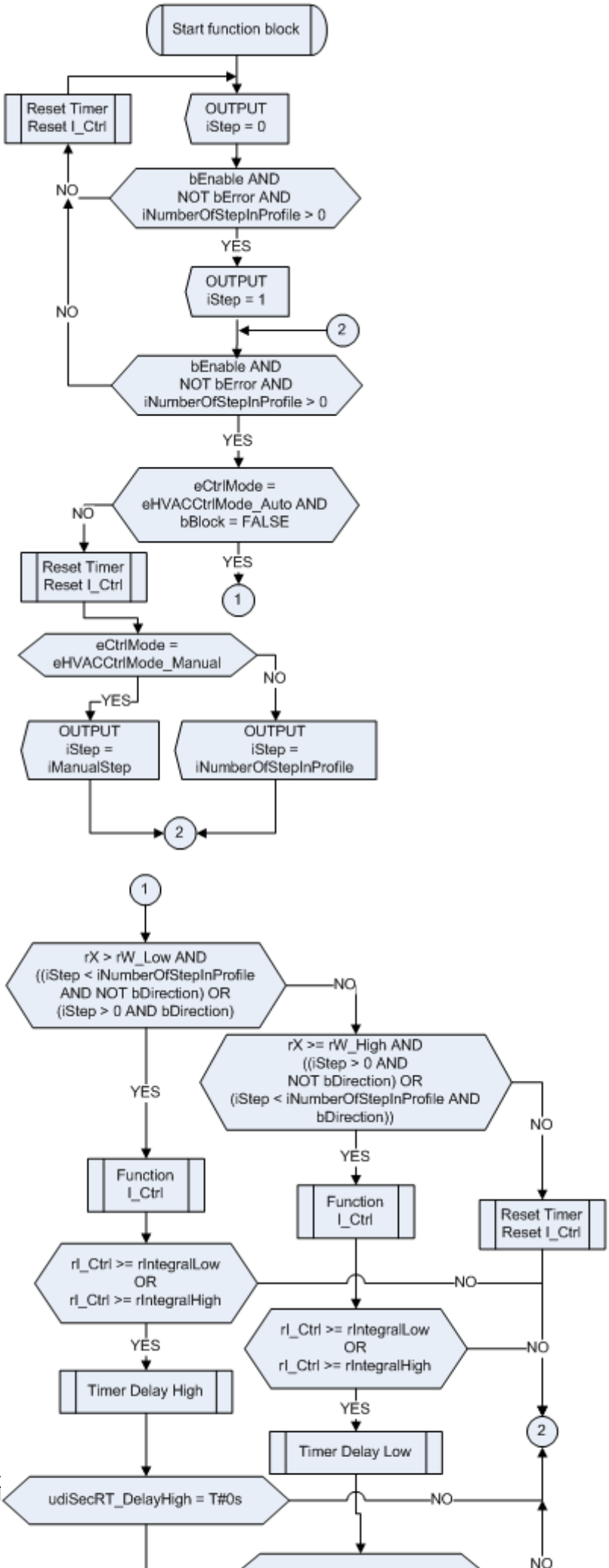
Nach jeder Änderung von iStep wird das I-Übertragungsglied und die internen Zeitverzögerungen (udiSecDelayLow, udiSecDelayHigh) zurück gesetzt.

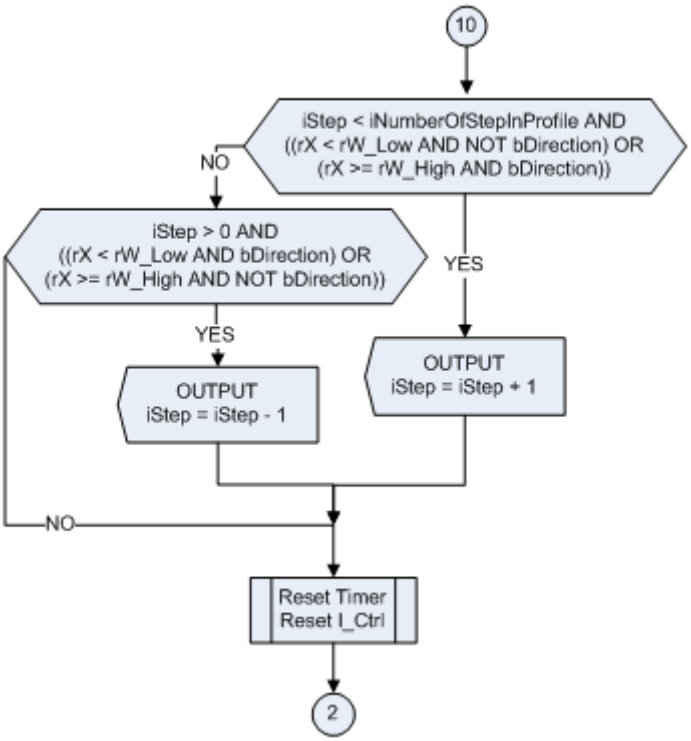
Übertragungsfunktion des I-Übertragungsglied (I_Ctrl)

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

Programmablaufplan

Programmablaufplan


Start function block
Reset Timer
Reset I_Ctrl
OUTPUT
iStep = 0
bEnable AND
NOT bError AND
iNumberOfStepInProfile > 0
NO
YES
OUTPUT
iStep = 1
2
bEnable AND
NOT bError AND
iNumberOfStepInProfile > 0
NO
YES
eCtrlMode =
eHVACCtrlMode_Auto AND
bBlock = FALSE
NO
YES
Reset Timer
Reset I_Ctrl
1
eCtrlMode =
eHVACCtrlMode_Manual
YES
NO
OUTPUT
iStep =
iManualStep
OUTPUT
iStep =
iNumberOfStepInProfile
2
1
rX > rW_Low AND
((iStep < iNumberOfStepInProfile
AND NOT bDirection) OR
(iStep > 0 AND bDirection)
NO
rX >= rW_High AND
((iStep > 0 AND
NOT bDirection) OR
(iStep < iNumberOfStepInProfile AND
bDirection))
YES
NO
Function
I_Ctrl
YES
Function
I_Ctrl
Reset Timer
Reset I_Ctrl
rI_Ctrl >= rIntegralLow
OR
rI_Ctrl >= rIntegralHigh
NO
YES
rI_Ctrl >= rIntegralLow
OR
rI_Ctrl >= rIntegralHigh
NO
Timer Delay High
YES
2
Timer Delay Low
udiSecRT_DelayHigh = T#0s
NO
NO



Programmablaufdarstellung als Diagramm

Programmablaufdarstellung als Diagramm

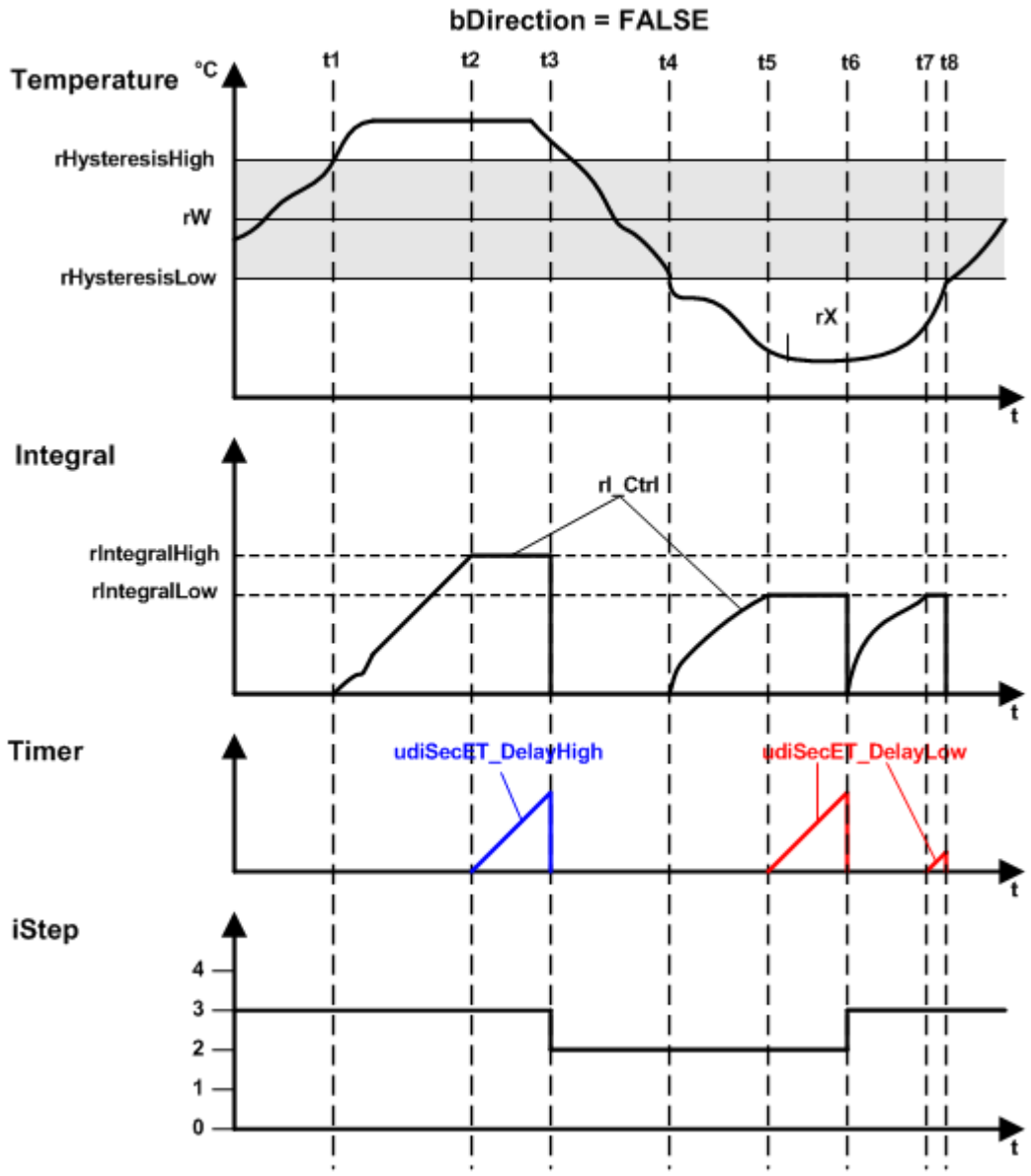


Abb. 12: FB_HVACI_CtrlStepDiagramm

Verhalten der verschiedenen Variablen

Verhalten der verschiedenen Variablen

Im Bild ist dargestellt wann welche Variablen wie zum Einsatz kommen.

Anwendungsbeispiel

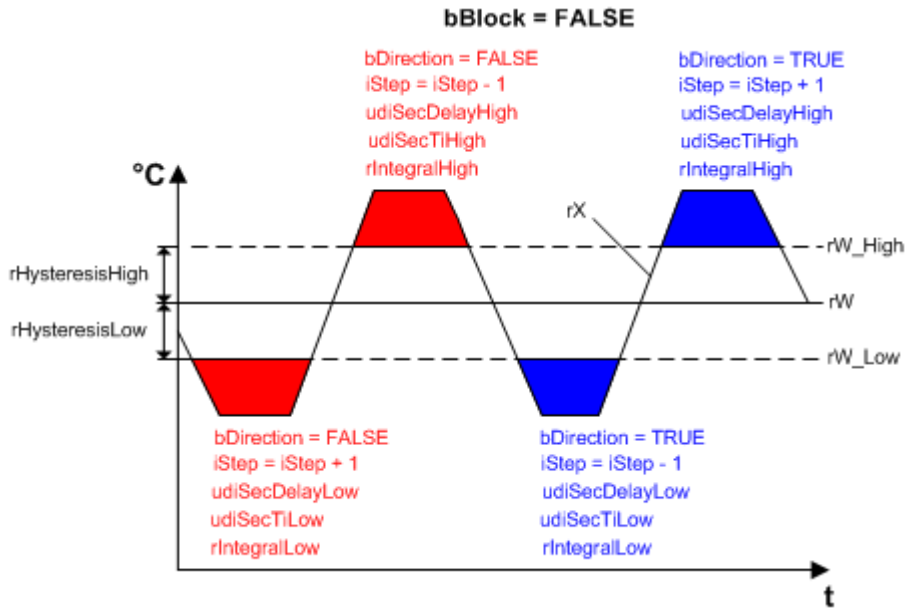


Abb. 13: FB_HVACI_CtrlStepParameter

Das Anwendungsbeispiel zeigt den Funktionsbaustein FB_HVACI_CtrlStep in Verbindung mit der Leistungsstufentabelle FB_HVACPowerRangeTable [▶ 281]. Das Beispiel liegt in den Programmiersprachen ST und CFC vor. Das Programmbeispiel **P_CFC_I_CtrlStep.PRG** in CFC ist im Ordner **Language CFC** > **Controller**, das Programmbeispiel **P_ST_I_CtrlStep.PRG** in ST ist im Ordner **Language Structure Text** > **Controller**.

Download	Benötige Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

```
eDataSecurityType     : E_HVACDataSecurityType;
bSetDefault           : BOOL;
bEnable               : BOOL;
rW                    : REAL;
rX                    : REAL;
eCtrlMode             : E_HVACCtrlMode;
iManualStep           : INT;             0..iNumberOfStepInProfile
bStepUp               : BOOL;
bStepDown             : BOOL;
bDirection            : BOOL;
iNumberOfStepInProfile: INT;             0..g_iMaxNumberOfSteps
bBlock                : BOOL;
bReset                : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Über ein TRUE wird der Funktionsbaustein freigegeben. Ist *bEnable* = FALSE, so wird *iStep* konstant auf 0 gesetzt. Die Überprüfung der Variable *iNumberOfStepInProfile* ist weiterhin aktiv. Falls dort ein Fehler auftritt, so wird dieses mit *bError* = TRUE angezeigt und kann nach Behebung des Fehlers mit *bReset* quittiert werden.

rW: Mit der Variablen *rW* wird der Sollwert übergeben.

rX: Mit der Variablen *rX* wird der Istwert übergeben.

eCtrlMode: Enum, welches die Betriebsart des Funktionsbausteins vorgibt. Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND*iNumberOfStepInProfile* > 0, so kann der Wert des Ausgangs *iStep* über *iManualStep* vorgeben werden. Beim Starten der PLC ist *eCtrlMode* = *eHVACCtrlMode_Auto*.

iManualStep: Ist die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual*AND*bEnable* = TRUEAND*bError* = FALSEAND*iNumberOfStepInProfile* > 0, so kann der Wert des Ausgangs *iStep* über *iManualStep* vorgeben werden. Der Eingabebereich von *iManualStep* kann maximal den Wert von *iNumberOfStepInProfile* haben und minimal 0 sein.

bStepUp: Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bBlock*=FALSE, so kann über eine steigende Flanke an dem Eingang *bStepUp* der Wert des Ausgangs *iStep* um 1 erhöht werden. Dieses kann so oft wiederholt werden bis *iStep* = *iNumberOfStepInProfile* ist.

bStepDown: Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bBlock*=FALSE, so kann über eine steigende Flanke an dem Eingang *bStepDown* der Wert des Ausgangs *iStep* um 1 erniedrigt werden. Dieses kann so oft wiederholt werden bis *iStep* = 0 ist.

bDirection: Mit *bDirection* wird der Wirksinn des Funktionsbausteins bestimmt. FALSE = Heizbetrieb; TRUE = Kühlbetrieb

iNumberOfStepInProfile: Anzahl der Stufen in der Leistungsstufensequenz. *iNumberOfStepInProfile* kann minimal 0 und maximal g_iMaxNumberOfSteps [▶ 541] sein. Werden die Grenzwerte nicht eingehalten, so wird ein Fehler ausgegeben und mit *bError* = TRUE angezeigt und *iStep* wird = 0.

bBlock: Mittels der Eingangsvariable *bBlock* kann die Regelung des Funktionsbausteins entweder frei gegeben werden oder der Ausgang *iStep* wird auf den Wert der Variablen *iNumberOfStepInProfile* fixiert.

Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bBlock*=FALSE, so ist das I-Übertragungsglied und die internen Zeitglieder zur Steuerung des Ausgangs *iStep* frei gegeben.

Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bBlock*=TRUE, so ist das I-Übertragungsglied und die internen Zeitglieder gesperrt und *iStep* = *iNumberOfStepInProfile*.

bReset: Eingang zur Quittierung der Störungen nach deren Behebung. Intern wird auf eine steigende Flanke reagiert.

VAR_OUTPUT

```
iStep                : INT;          0..iNumberOfStepInProfile
rI_Ctrl              : REAL;
rE                   : REAL;
rW_High              : REAL;
rW_Low               : REAL;
b_rW_High            : BOOL;
b_rW_Low             : BOOL;
udiSecRT_DelayHigh   : UDINT;        Second Remaining Time Delay High
```

```
udiSecRT_DelayLow   : UDINT;         Second Remaining Time Delay Low
bError              : BOOL;
eErrorCode         : E_HVACErrorCodes;
```

iStep: Mit der Ausgangsvariablen *iStep* wird die Stufe in einer Leistungsstufensequenz angegeben. Beim Aufstarten des Funktionsbaustein ist *iStep* = 1, wenn *bEnable* = TRUE, *bError* = FALSE,*eCtrlMode* = *eHVACCtrlMode_Auto*AND*iNumberOfStepInProfile* > 0 ist. *iStep* kann maximal den Wert von *iNumberOfStepInProfile* erreichen und minimal 0 sein.

Der Wert von *iStep* kann manuell über *iManualStep* vorgegeben werden, wenn *bEnable* = TRUE, *bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Manual* ist.

Über die Eingangsvariablen *bStepUp* / *bStepDown* kann der Wert von *iStep* in- oder dekrementiert werden, wenn *bEnable* = TRUE, *bError* = FALSE, *eCtrlMode* = *eHVACCtrlMode_Auto*AND*bBlock* = FALSE ist.

Ist *bEnable* = TRUE, *bError* = FALSE, *eCtrlMode* = *eHVACCtrlMode_Auto*AND*bBlock* = FALSE und der Wert von *iStep* hat sich geändert, so werden die Verzögerungszeitglieder und das I-Übertragungsglied zurück gesetzt.

Ist *bEnable* = TRUE, *bError* = FALSE, *eCtrlMode* = *eHVACCtrlMode_Auto*, *bBlock* = FALSEAND*iNumberOfStepInProfile* > 0, der Ausgang des I-Übertragungsglieds *rI_Ctrl* =*rIntegralHigh* oder *rIntegralLow* und eine der Verzögerungszeiten *tRemainingTimeDelayHigh* / *tRemainingTimeDelayLow* =T#0s, so wird der Ausgang *iStep* folgendermaßen gesteuert:

iStep = *iStep* + 1 wenn

(*rX* < *rW_Low*AND*iStep* < *iNumberOfStepInProfile* ANDNOT*bDirection*)
OR
(*rX* >= *rW_High*AND*iStep* < *iNumberOfStepInProfile* AND*bDirection*) ist.

iStep = *iStep* - 1 wenn

(*rX* < *rW_Low*AND*iStep* > 0 AND*bDirection*)
OR
(*rX* >= *rW_High*AND*iStep* > 0 ANDNOT*bDirection*) ist, siehe Programmablaufplan [▶ 256]

Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bBlock*=TRUE, so sind die internen Zeitglieder und das I-Übertragungsglied gesperrt und *iStep* = *iNumberOfStepInProfile*.

rI_Ctrl: Ausgang des I-Übertragungsgliedes.
Ist *rI_Ctrl* >= *rIntegralHigh* oder*rIntegralLow* und *tRemainingTimeDelayHigh* oder *tRemainingTimeDelayLow* = T#0s, so wird über *iStep* die Nummer der Stufen um eins erhöht oder erniedrigt. Anschliessend wird das I-Übertragungsglied und die Zeitglieder für das Hoch- und Runterschalten zurück gesetzt, so dass das Hoch- oder Runterschalten neu gestartet werden, siehe Programmablaufplan [▶ 256].

rE: Regelabweichung mit der das interne I-Übertragungsglied arbeitet: *rE* = *rW* - *rX*

rW_High: *rW_High* := *rW* + *rHysteresisHigh* - obere Sollwertgrenze nach dessen Überschreitung durch *rX* zuerst das I-Übertragungsglied aktiviert wird und anschliessend das Zeitglied der Verzögerungszeit *udiSecDelayHigh*, siehe Anwendung [▶ 255] oder Verhalten verschiedener Variablen [▶ 259].

rW_Low: *rW_Low* := *rW* - *rHysteresisLow* - untere Sollwertgrenze nach dessen Unterschreitung durch *rX* zuerst das I-Übertragungsglied aktiviert wird und anschliessend das Zeitglied der Verzögerungszeit *udiSecDelayLow*, siehe Anwendung [▶ 255] oder Verhalten verschiedener Variablen [▶ 259].

b_rW_High: *b_rW_High* wird TRUE, wenn *rX* > *rW_High* ist.

b_rW_Low: *b_rW_Low* wird TRUE, wenn *rX* < *rW_Low* ist.

udiSecRT_DelayHigh: Verbleibende Zeit der Verzögerungszeit *udiSecDelayHigh.*

udiSecRT_DelayLow: Verbleibende Zeit der Verzögerungszeit *udiSecDelayLow.*

bError: Der Ausgang signalisiert mit einem TRUE, dass ein Fehler anliegt und ein falscher Parameter an der Variable *iNumberOfStepInProfile* anliegt. *iStep* wird konstant auf 0 gesetzt und das Enum *eErrorCode* zeigt die Fehlernummer an. Nach Behebung des Fehlers muss die Meldung *bError* mit *bReset* quittiert werden.

eErrorCode: Liefert bei einem gesetzten *bError*-Ausgang die Fehlernummer [▶ 529]. Folgender Fehler kann in diesem Funktionsbaustein vorkommen: *eHVACErrorCodes_InvalidParam_NumberOfStepInProfile*

Um in der SPS an die Fehlernummern des Enums zu gelangen, kann eErrorCode einer Variablen vom Datentyp WORD zugewiesen werden.eHVACErrorCodes_Error_iNumberOfStepInProfil= 32

VAR_IN_OUT

```
rHysteresisHigh       : REAL;
rHysteresisLow        : REAL;
udiSecDelayHigh       : UDINT;
udiSecDelayLow        : UDINT;
udiSecTiHigh          : UDINT;
udiSecTiLow           : UDINT;
rIntegralHigh         : REAL;
rIntegralLow          : REAL;
```

rHysteresisHigh: Positiver Wert der oberen Grenze der Regelabweichung, siehe Anwendung [▶ 255] oder Verhalten verschiedener Variablen [▶ 259]. *rW_High* := *rW* + *rHysteresisHigh.* Die Variable wird persistent gespeichert. Voreingestellt auf 5.

rHysteresisLow: Positiver Wert der unteren Grenze der Regelabweichung, siehe Anwendung [▶ 255] oder Verhalten verschiedener Variablen [▶ 259]. *rW_Low* := *rW* – *rHysteresisLow.* Die Variable wird persistent gespeichert. Voreingestellt auf 5.

udiSecDelayHigh: Verzögerungszeit nach deren Ablauf *iStep* in- oder dekrementiert, siehe Anwendung [▶ 255] oder Verhalten verschiedener Variablen [▶ 259]. Die Variable wird persistent gespeichert. Voreingestellt auf 300s.

udiSecDelayLow: Verzögerungszeit nach deren Ablauf *iStep* in- oder dekrementiert, siehe Anwendung [▶ 255] oder Verhalten verschiedener Variablen [▶ 259]. Die Variable wird persistent gespeichert. Voreingestellt auf 300s.

udiSecTiHigh: Integrationszeit für das obere Limit des I-Übertragungsgliedes in Sekunden, siehe Anwendung [▶ 255] oder Verhalten verschiedener Variablen [▶ 259]. *udiSecTiHigh* muss > 0 sein. Ansonsten wird intern mit dem Default-Wert oder dem letzten gültigen Variablenwert gearbeitet. Die Variable wird persistent gespeichert. Voreingestellt auf 60s.

udiSecTiLow: Integrationszeit für das untere Limit des I-Übertragungsgliedes, siehe Anwendung [▶ 255] oder Verhalten verschiedener Variablen [▶ 259]. *udiSecTiLow* muss > 0 sein. Ansonsten wird intern mit dem Default-Wert oder dem letzten gültigen Variablenwert gearbeitet. Die Variable wird persistent gespeichert. Voreingestellt auf 60s.

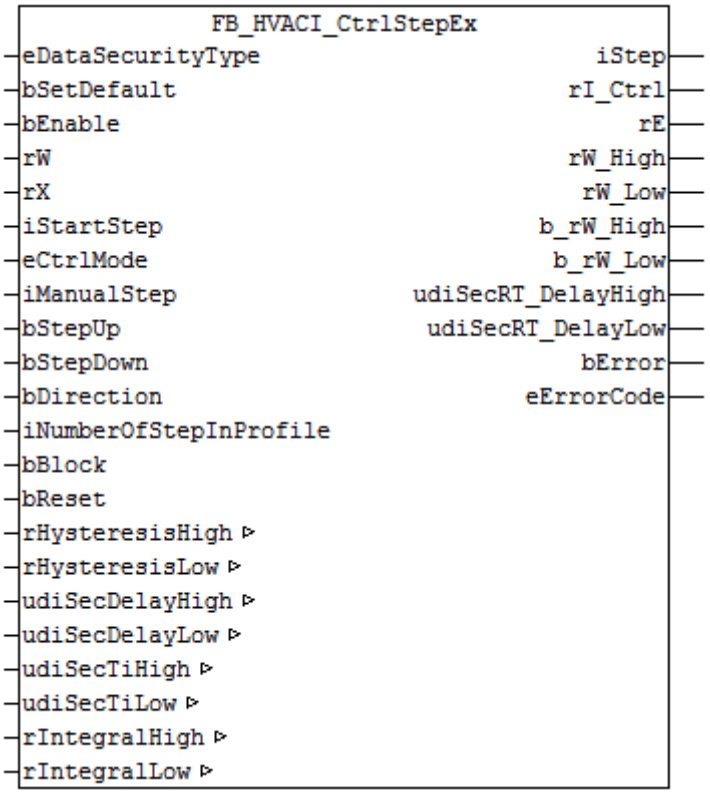
rIntegralHigh: Positiver Wert für das obere Limit an dem die Integration des I-Übertragungsgliedes angehalten wird (ARW-Maßnahme, anti-reset-windup) und eine der Verzögerungszeiten startet, siehe Anwendung [▶ 255] oder Verhalten verschiedener Variablen [▶ 259]. Die Variable wird persistent gespeichert. Voreingestellt auf 15.

rIntegralLow: Positiver Wert für das untere Limit an dem die Integration des I-Übertragungsgliedes angehalten wird (ARW-Maßnahme, anti-reset-windup) und eine der Verzögerungszeiten startet, siehe Anwendung [▶ 255] oder Verhalten verschiedener Variablen [▶ 259]. Die Variable wird persistent gespeichert. Voreingestellt auf 15.

Dokumente hierzu

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

3.4.3 FB_HVACI_CtrlStepEx



Anwendung

Anwendung

Der Funktionsbaustein dient zur sequentiellen Folgesteuerung von Leistungserzeugern. Der Unterschied zum FB_HVACI_CtrlStep [▶ 255] liegt darin, dass der Wert von *iStep* beim Aufstarten des Funktionsbausteins über die Variable *iStartStep* vorgegeben wird.

In Verbindung mit der Leistungsstufentabelle FB_HVACPowerRangeTable [▶ 281] kann der Leistungsstufenregler für die stufige Regelung mehrerer Heizkessel, Kältemaschinen oder Rückkühlwerke eingesetzt werden.

Das Hoch- oder Runterschalten in die nächst höhere oder niedrigere Leistungsstufe erfolgt über ein Integral (rW - rX) und einer Zeitverzögerung. Zuerst muss der Temperaturistwert *rX* einen Schwellwert *rW_High* / *rW_Low* über- oder unterschritten haben. Anschliessend startet das Integral. Wird an dem Ausgang *rI_Ctrl* der obere oder untere Grenzwert (*rIntegralHigh / rIntegralLow)* erreicht, so wird ein Zeitglied gestartet, nach dessen Ablauf (*udiSecRT_DelayHigh / udiSecRT_DelayLow)*der Ausgang *iStep* in- oder dekrementiert wird, siehe Programmablaufdarstellung als Diagramm [▶ 267]

Eine eingestellte Integrationszeit udiSecTiHigh/udiSecTiLow = 60 Sekunden heisst, dass das I-Übertragungsglied sich um ein Kelvin pro Minute ändert. Bei einer Regelabweichung rE = 5 hat der Ausgang des I-Übertragungsgliedes rI_Ctrl nach einer Minute den Wert 5. Beispiel:bDirection = FALSE; udiSecTiLow = 60; rE = 2; rIntegralLow = 10 Bei der bleibenden Regelabweichung von 2 und der unteren Integralgrenze von 10 ist nach 5 Minuten rI_Ctrl = rIntegralLow. Dieses hat zur Folge, dass ein Zeitglied mit der Verzögerungszeit udiSecDelayLow gestartet wird. Nach dessen Ablauf (udiSecRT_DelayHigh / udiSecRT_DelayLow) wird der Ausgang iStep um 1 inkrementiert und das I-Übertragungsglied und die Zeitverzögerungen werden zurück gesetzt.

iStep wird für einen SPS-Zyklus auf iStartStep gesetzt, wenn bEnable = TRUEANDbError = FALSE ist.

Bedingungen

Das I-Übertragungsglied wird freigegeben, wenn

bEnable = TRUEAND NOT*bError*AND*eCtrlMode* = *eHVACCtrlMode_Auto*AND NOT*bBlock*
(
(*rX* >= *rW_High*AND ((*iStep* > 0 ANDNOT*bDirection*) OR (*iStep* < *iNumberOfStepInProfile*AND*bDirection*))
OR
(*rX* < *rW_Low*AND ((*iStep* < *iNumberOfStepInProfile*ANDNOT*bDirection*) OR (*iStep* > 0 AND*bDirection*))
)

ist. Ist *rI_Ctrl =rIntegralHigh / rIntegralLow*, so werden die Zeitglieder der Verzögerungszeiten *udiSecDelayHigh / udiSecDelayLo*w aktiviert. Nach Ablauf einer der Verzögerungszeiten *udiSecRT_DelayHigh* / *udiSecRT_DelayLow* werden die Stufen der Leistungsstufensequenz folgendermaßen gesteuert:

iStep = *iStep* + 1 wenn

(*rX* < *rW_Low*AND*iStep* < *iNumberOfStepInProfile*ANDNOT*bDirection*)
OR
(*rX* >= *rW_High*AND*iStep* < *iNumberOfStepInProfile*AND*bDirection*)

iStep = *iStep* - 1 wenn

(*rX* < *rW_Low*AND*iStep* > 0 AND*bDirection*)
OR
(*rX* >= *rW_High*AND*iStep* > 0 ANDNOT*bDirection*)

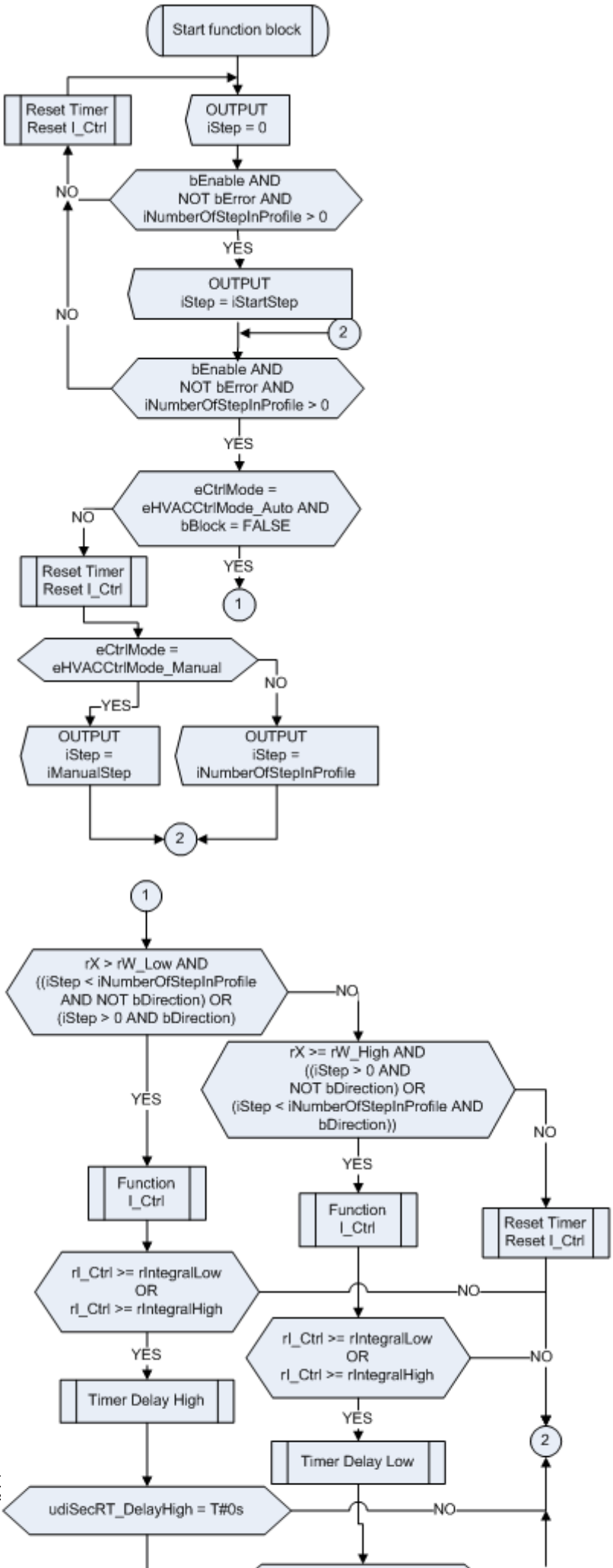
Nach jeder Änderung von iStep wird das I-Übertragungsglied und die internen Zeitverzögerungen (udiSecDelayLow, udiSecDelayHigh) zurück gesetzt.

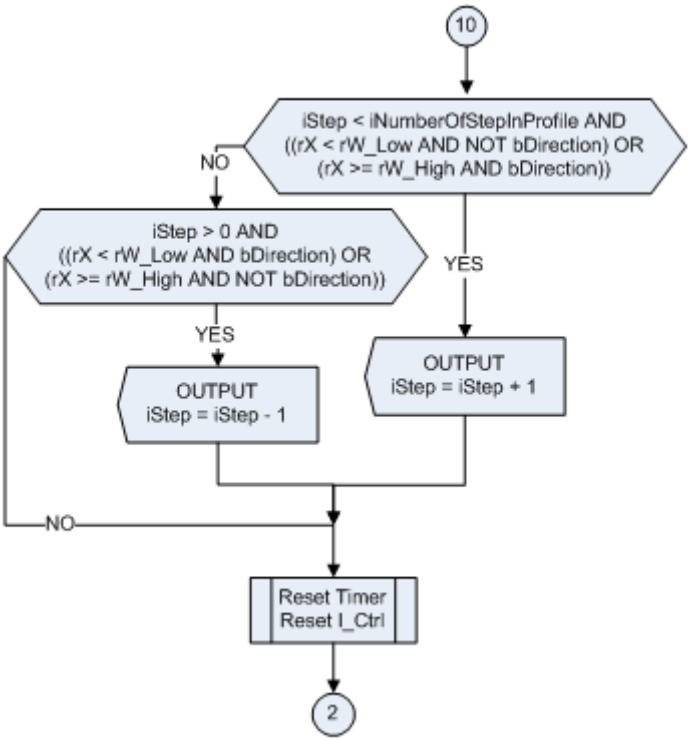
Übertragungsfunktion des I-Übertragungsglied (I_Ctrl)

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

Programmablaufplan

Programmablaufplan


Start function block
Reset Timer
Reset I_Ctrl
OUTPUT
iStep = 0
bEnable AND
NOT bError AND
iNumberOfStepInProfile > 0
NO
YES
OUTPUT
iStep = iStartStep
2
bEnable AND
NOT bError AND
iNumberOfStepInProfile > 0
NO
YES
eCtrlMode =
eHVACCtrlMode_Auto AND
bBlock = FALSE
NO
YES
Reset Timer
Reset I_Ctrl
1
eCtrlMode =
eHVACCtrlMode_Manual
NO
YES
OUTPUT
iStep =
iManualStep
OUTPUT
iStep =
iNumberOfStepInProfile
2
1
rX > rW_Low AND
((iStep < iNumberOfStepInProfile
AND NOT bDirection) OR
(iStep > 0 AND bDirection)
NO
YES
rX >= rW_High AND
((iStep > 0 AND
NOT bDirection) OR
(iStep < iNumberOfStepInProfile AND
bDirection))
NO
YES
Function
I_Ctrl
Function
I_Ctrl
Reset Timer
Reset I_Ctrl
rI_Ctrl >= rIntegralLow
OR
rI_Ctrl >= rIntegralHigh
NO
YES
rI_Ctrl >= rIntegralLow
OR
rI_Ctrl >= rIntegralHigh
NO
YES
Timer Delay High
Timer Delay Low
2
udiSecRT_DelayHigh = T#0s
NO



Programmablaufdarstellung als Diagramm

Programmablaufdarstellung als Diagramm

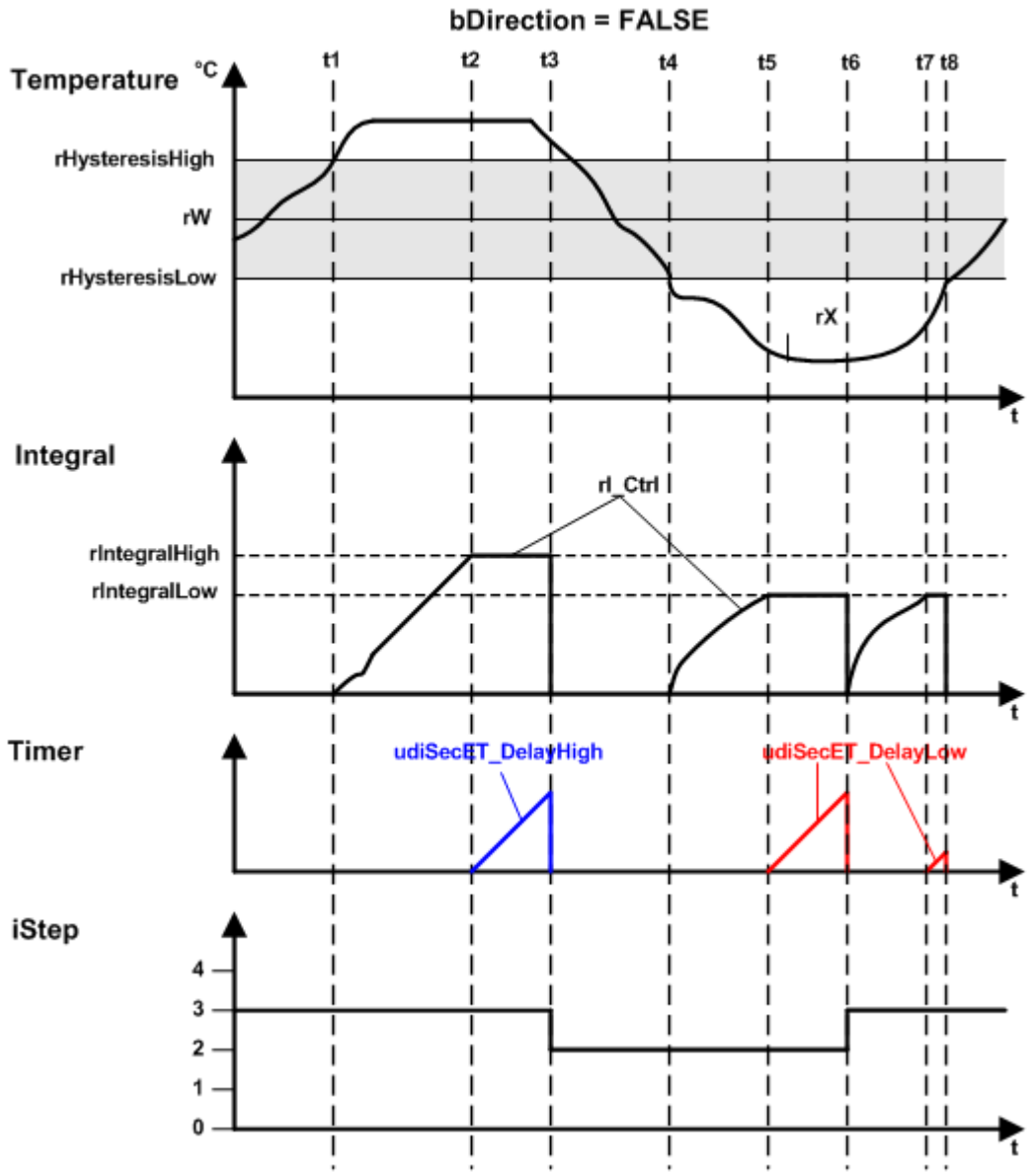


Abb. 14: FB_HVACI_CtrlStepDiagramm

Verhalten der verschiedenen Variablen

Verhalten der verschiedenen Variablen

Im Bild ist dargestellt wann welche Variablen wie zum Einsatz kommen.

Anwendungsbeispiel

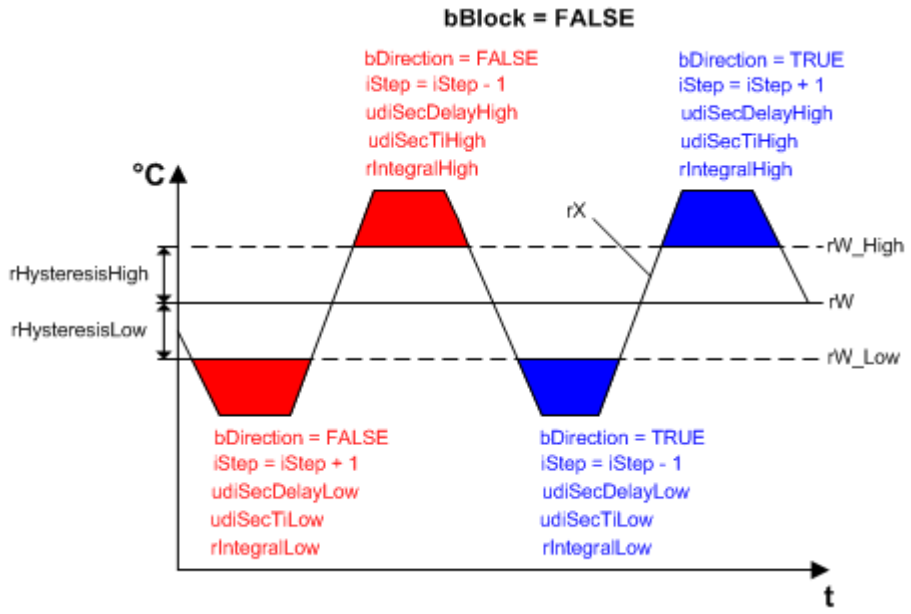


Abb. 15: FB_HVACI_CtrlStepParameter

Das Anwendungsbeispiel zeigt den Funktionsbaustein FB_HVACI_CtrlStep in Verbindung mit der Leistungsstufentabelle FB_HVACPowerRangeTable [▶ 281]. Das Beispiel liegt in den Programmiersprachen ST und CFC vor. Das Programmbeispiel **P_CFC_I_CtrlStep.PRG** in CFC ist im Ordner **Language CFC** > **Controller**, das Programmbeispiel **P_ST_I_CtrlStep.PRG** in ST ist im Ordner **Language Structure Text** > **Controller**.

Download	Benötige Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

```
eDataSecurityType     : E_HVACDataSecurityType;
bSetDefault           : BOOL;
bEnable           : BOOL;
rW            : REAL;
rX            : REAL;
iStartStep        : INT;             0..iNumberOfStepInProfile
eCtrlMode         : E_HVACCtrlMode;
iManualStep           : INT;             0..iNumberOfStepInProfile
bStepUp           : BOOL;
bStepDown         : BOOL;
bDirection        : BOOL;
iNumberOfStepInProfile    : INT;             0..g_iMaxNumberOfSteps
bBlock            : BOOL;
bReset            : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen. Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Über ein TRUE wird der Funktionsbaustein freigegeben. Ist *bEnable* = FALSE, so wird *iStep* konstant auf 0 gesetzt. Die Überprüfung der Variable *iNumberOfStepInProfile* ist weiterhin aktiv. Falls dort ein Fehler auftritt, so wird dieses mit *bError* = TRUE angezeigt und kann nach Behebung des Fehlers mit *bReset* quittiert werden.

rW: Mit der Variablen *rW* wird der Sollwert übergeben.

rX: Mit der Variablen *rX* wird der Istwert übergeben.

iStartStep: Ist der Funktionsbaustein freigegeben (*bEnable* = TRUE) und es liegt kein Fehler an (bError = FALSE), so ist für ein SPS-Zyklus *iStep* = *iStartStep*.

eCtrlMode: Enum, welches die Betriebsart des Funktionsbausteins vorgibt. Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND*iNumberOfStepInProfile* > 0, so kann der Wert des Ausgangs *iStep* über *iManualStep* vorgeben werden. Beim Starten der PLC ist *eCtrlMode* = *eHVACCtrlMode_Auto*.

iManualStep: Ist die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual*AND*bEnable* = TRUEAND*bError* = FALSEAND*iNumberOfStepInProfile* > 0, so kann der Wert des Ausgangs *iStep* über *iManualStep* vorgeben werden. Der Eingabebereich von *iManualStep* kann maximal den Wert von *iNumberOfStepInProfile* haben und minimal 0 sein.

bStepUp: Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bBlock*=FALSE, so kann über eine steigende Flanke an dem Eingang *bStepUp* der Wert des Ausgangs *iStep* um 1 erhöht werden. Dieses kann so oft wiederholt werden bis *iStep* = *iNumberOfStepInProfile* ist.

bStepDown: Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bBlock*=FALSE, so kann über eine steigende Flanke an dem Eingang *bStepDown* der Wert des Ausgangs *iStep* um 1 erniedrigt werden. Dieses kann so oft wiederholt werden bis *iStep* = 0 ist.

bDirection: Mit *bDirection* wird der Wirksinn des Funktionsbausteins bestimmt. FALSE = Heizbetrieb; TRUE = Kühlbetrieb

iNumberOfStepInProfile: Anzahl der Stufen in der Leistungsstufensequenz. *iNumberOfStepInProfile* kann minimal 0 und maximal g_iMaxNumberOfSteps [▶ 541] sein. Werden die Grenzwerte nicht eingehalten, so wird ein Fehler ausgegeben und mit *bError* = TRUE angezeigt und *iStep* wird = 0.

bBlock: Mittels der Eingangsvariable *bBlock* kann die Regelung des Funktionsbausteins entweder frei gegeben werden oder der Ausgang *iStep* wird auf den Wert der Variablen *iNumberOfStepInProfile* fixiert.

Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bBlock*=FALSE, so ist das I-Übertragungsglied und die internen Zeitglieder zur Steuerung des Ausgangs *iStep* frei gegeben.

Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bBlock*=TRUE, so ist das I-Übertragungsglied und die internen Zeitglieder gesperrt und *iStep* = *iNumberOfStepInProfile*.

bReset: Eingang zur Quittierung der Störungen nach deren Behebung. Intern wird auf eine steigende Flanke reagiert.

VAR_OUTPUT

```
iStep               : INT;          0..iNumberOfStepInProfile
rI_Ctrl             : REAL;
rE                  : REAL;
rW_High             : REAL;
rW_Low              : REAL;
b_rW_High              : BOOL;
b_rW_Low            : BOOL;
```

```
udiSecRT_DelayHigh          : UDINT;                        Second Remaining Time Delay High
udiSecRT_DelayLow           : UDINT;                        Second Remaining Time Delay Low
bError              : BOOL;
eErrorCode              : E_HVACErrorCodes;
```

iStep: Mit der Ausgangsvariablen *iStep* wird die Stufe in einer Leistungsstufensequenz angegeben. Beim Aufstarten des Funktionsbaustein ist *iStep* = *iStartStep*, wenn *bEnable* = TRUEAND*bError* = FALSE. *iStep* kann maximal den Wert von *iNumberOfStepInProfile* erreichen und minimal 0 sein.

Der Wert von *iStep* kann manuell über *iManualStep* vorgegeben werden, wenn *bEnable* = TRUE, *bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Manual* ist.

Über die Eingangsvariablen *bStepUp* / *bStepDown* kann der Wert von *iStep* in- oder dekrementiert werden, wenn *bEnable* = TRUE, *bError* = FALSE, *eCtrlMode* = *eHVACCtrlMode_Auto*AND*bBlock* = FALSE ist.

Ist *bEnable* = TRUE, *bError* = FALSE, *eCtrlMode* = *eHVACCtrlMode_Auto*AND*bBlock* = FALSE und der Wert von *iStep* hat sich geändert, so werden die Verzögerungszeitglieder und das I-Übertragungsglied zurück gesetzt.

Ist *bEnable* = TRUE, *bError* = FALSE, *eCtrlMode* = *eHVACCtrlMode_Auto*, *bBlock* = FALSEAND*iNumberOfStepInProfile* > 0, der Ausgang des I-Übertragungsglieds *rI_Ctrl* =*rIntegralHigh* oder *rIntegralLow* und eine der Verzögerungszeiten *tRemainingTimeDelayHigh* / *tRemainingTimeDelayLow* =T#0s, so wird der Ausgang *iStep* folgendermaßen gesteuert:

iStep = *iStep* + 1 wenn

(*rX* < *rW_Low*AND*iStep* < *iNumberOfStepInProfile*ANDNOT*bDirection*)
OR
(*rX* >= *rW_High*AND*iStep* < *iNumberOfStepInProfile*AND*bDirection*) ist.

iStep = *iStep* - 1 wenn

(*rX* < *rW_Low*AND*iStep* > 0 AND*bDirection*)
OR
(*rX* >= *rW_High*AND*iStep* > 0 ANDNOT*bDirection*) ist, siehe Programmablaufplan [▶ 265]

Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bBlock*=TRUE, so sind die internen Zeitglieder und das I-Übertragungsglied gesperrt und *iStep* = *iNumberOfStepInProfile*.

rI_Ctrl: Ausgang des I-Übertragungsgliedes.
Ist *rI_Ctrl* >= *rIntegralHigh* oder*rIntegralLow* und *tRemainingTimeDelayHigh* oder *tRemainingTimeDelayLow* = T#0s, so wird über *iStep* die Nummer der Stufen um eins erhöht oder erniedrigt. Anschliessend wird das I-Übertragungsglied und die Zeitglieder für das Hoch- und Runterschalten zurück gesetzt, so dass das Hoch- oder Runterschalten neu gestartet werden, siehe Programmablaufplan [▶ 265].

rE: Regelabweichung mit der das interne I-Übertragungsglied arbeitet: *rE* = *rW* - *rX*

rW_High: *rW_High* := *rW* + *rHysteresisHigh* - obere Sollwertgrenze nach dessen Überschreitung durch *rX* zuerst das I-Übertragungsglied aktiviert wird und anschliessend das Zeitglied der Verzögerungszeit *udiSecDelayHigh*, siehe Anwendung [▶ 264] oder Verhalten verschiedener Variablen [▶ 268].

rW_Low: *rW_Low* := *rW* - *rHysteresisLow* - untere Sollwertgrenze nach dessen Unterschreitung durch *rX* zuerst das I-Übertragungsglied aktiviert wird und anschliessend das Zeitglied der Verzögerungszeit *udiSecDelayLow*, siehe Anwendung [▶ 264] oder Verhalten verschiedener Variablen [▶ 268].

b_rW_High: *b_rW_High* wird TRUE, wenn *rX* > *rW_High* ist.

b_rW_Low: *b_rW_Low* wird TRUE, wenn *rX* < *rW_Low* ist.

udiSecRT_DelayHigh: Verbleibende Zeit der Verzögerungszeit *udiSecDelayHigh.*

udiSecRT_DelayLow: Verbleibende Zeit der Verzögerungszeit *udiSecDelayLow.*

bError: Der Ausgang signalisiert mit einem TRUE, dass ein Fehler anliegt und ein falscher Parameter an der Variable *iNumberOfStepInProfile* anliegt. *iStep* wird konstant auf 0 gesetzt und das Enum *eErrorCode* zeigt die Fehlernummer an. Nach Behebung des Fehlers muss die Meldung *bError* mit *bReset* quittiert werden.

eErrorCode: Liefert bei einem gesetzten *bError*-Ausgang die Fehlernummer [▶ 529]. Folgender Fehler kann in diesem Funktionsbaustein vorkommen: *eHVACErrorCodes_InvalidParam_NumberOfStepInProfile*

Um in der SPS an die Fehlernummern des Enums zu gelangen, kann eErrorCode einer Variablen vom Datentyp WORD zugewiesen werden.eHVACErrorCodes_Error_iNumberOfStepInProfil= 32

VAR_IN_OUT

```
rHysteresisHigh       : REAL;
rHysteresisLow        : REAL;
udiSecDelayHigh       : UDINT;
udiSecDelayLow        : UDINT;
udiSecTiHigh          : UDINT;
udiSecTiLow           : UDINT;
rIntegralHigh         : REAL;
rIntegralLow          : REAL;
```

rHysteresisHigh: Positiver Wert der oberen Grenze der Regelabweichung, siehe Anwendung [▶ 264] oder Verhalten verschiedener Variablen [▶ 268]. *rW_High* := *rW* + *rHysteresisHigh* Die Variable wird persistent gespeichert. Voreingestellt auf 5.

rHysteresisLow: Positiver Wert der unteren Grenze der Regelabweichung, siehe Anwendung [▶ 264] oder Verhalten verschiedener Variablen [▶ 268]. *rW_Low* := *rW* – *rHysteresisLow* Die Variable wird persistent gespeichert. Voreingestellt auf 5.

udiSecDelayHigh: Verzögerungszeit nach deren Ablauf *iStep* in- oder dekrementiert, siehe Anwendung [▶ 264] oder Verhalten verschiedener Variablen [▶ 268]. Die Variable wird persistent gespeichert. Voreingestellt auf 300s.

udiSecDelayLow: Verzögerungszeit nach deren Ablauf *iStep* in- oder dekrementiert, siehe Anwendung [▶ 264] oder Verhalten verschiedener Variablen [▶ 268]. Die Variable wird persistent gespeichert. Voreingestellt auf 300s.

udiSecTiHigh: Integrationszeit für das obere Limit des I-Übertragungsgliedes in Sekunden, siehe Anwendung [▶ 264] oder Verhalten verschiedener Variablen [▶ 268]. *udiSecTiHigh* muss > 0 sein. Ansonsten wird intern mit dem Default-Wert oder dem letzten gültigen Variablenwert gearbeitet. Die Variable wird persistent gespeichert. Voreingestellt auf 60s.

udiSecTiLow: Integrationszeit für das untere Limit des I-Übertragungsgliedes, siehe Anwendung [▶ 264] oder Verhalten verschiedener Variablen [▶ 268]. *udiSecTiLow* muss > 0 sein. Ansonsten wird intern mit dem Default-Wert oder dem letzten gültigen Variablenwert gearbeitet. Die Variable wird persistent gespeichert. Voreingestellt auf 60s.

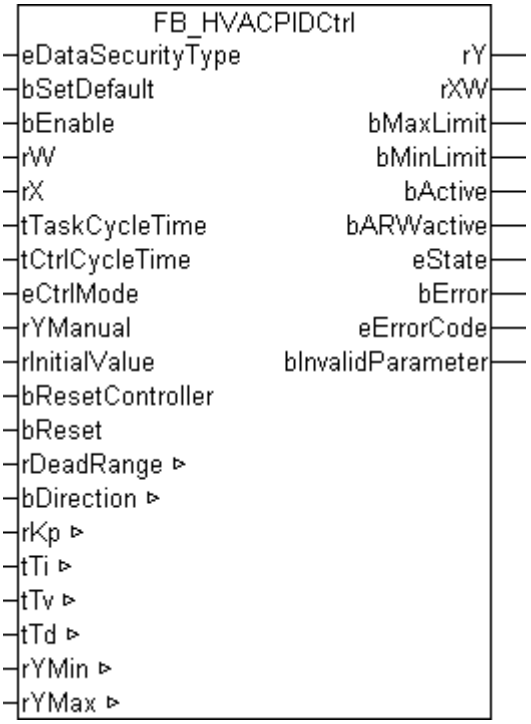
rIntegralHigh: Positiver Wert für das obere Limit an dem die Integration des I-Übertragungsgliedes angehalten wird (ARW-Maßnahme, anti-reset-windup) und eine der Verzögerungszeiten startet, siehe Anwendung [▶ 264] oder Verhalten verschiedener Variablen [▶ 268]. Die Variable wird persistent gespeichert. Voreingestellt auf 15.

rIntegralLow: Positiver Wert für das untere Limit an dem die Integration des I-Übertragungsgliedes angehalten wird (ARW-Maßnahme, anti-reset-windup) und eine der Verzögerungszeiten startet, siehe Anwendung [▶ 264] oder Verhalten verschiedener Variablen [▶ 268]. Die Variable wird persistent gespeichert. Voreingestellt auf 15.

Dokumente hierzu

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

3.4.4 FB_HVACPIDCtrl



Anwendung

Der PID-Regler ist ein Standardregler für Applikationen der Heizungs- und Klimatechnik. Er kann zur Regelung von Temperaturen, Drücken, Volumenströmen oder der Luftfeuchte verwendet werden. Die Werte für Ti und Tv können auf Null gesetzt werden. Dadurch ist eine P, PI oder PD Charakteristik des Reglers einstellbar. Der Regler besitzt eine ARW-Funktion (Anti-Reset-Windup). Diese verhindert bei einer fortwährend anstehenden Regelabweichung das andauernde Aufintegrieren.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
bEnable          : BOOL;
rW               : REAL;
rX               : REAL;
tTaskCycleTime   : TIME;
tCtrlCycleTime   : TIME;
eCtrlMode        : E_HVACCtrlMode;
rYManual         : REAL;
rInitialValue    : REAL;
bResetController : BOOL;
bReset           : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Eingangsvariable zur Freigabe des Reglers. Der Regler ist aktiv, wenn *bEnable* TRUE ist.

rW: Mit der Variablen *rW* wird dem Regler der Sollwert übergeben.

rX: *rX* erfasst den Istwert des Regelkreises.

tTaskCycleTime: Zykluszeit, mit der der Funktionsbaustein aufgerufen wird. Diese entspricht der Task-Zykluszeit der aufrufenden Task, wenn der Baustein in jedem Zyklus aufgerufen wird.

tCtrlCycleTime: Mit der Variablen *tCtrlCycleTime* wird die Zykluszeit vorgegeben mit welcher der PID-Regler abgearbeitet wird. Die kleinste mögliche Zykluszeit ist die der Steuerung. Da die Regelstrecken in der Gebäudeautomation überwiegend langsam sind, kann die Zykluszeit des Reglers ein mehrfaches der Steuerungszykluszeit betragen.

eCtrlMode: Über dieses Enum wird der Betriebsmodus ausgewählt.

rYManual: Für Testzwecke kann der Ausgang *rY* des Reglers übersteuert werden. Bei Handbetrieb ist am Ausgang des Reglers der eingestellte Wert von *rYManual* zu finden.

rInitialValue: Mit dem *rInitialValue* wird das Neustartverhalten des Reglers beeinflusst. Die Initialwerte werden bei einer steigenden Flanke der Reglerfreigabe *bEnable* übernommen.
0 = Start mit dem Wert Null am Ausgang *rY*
<>0 = Start mit dem Wert von *rInitialValue*

bResetController: Eine positive Flanke am Eingang *bResetController* bewirkt einen Neustart des PID-Reglers.

bReset: Quittierungseingang bei einer Störung.

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: Stellsignalausgang des PID-Reglers.

rXW: Regelabweichung.

bMaxLimit: Der Ausgang *bMaxLimit* ist TRUE, wenn der Ausgang *rY* den Wert *rYMax*erreicht hat.

bMinLimit: Der Ausgang *bMinLimit* ist TRUE**,** wenn der Ausgang *rY* den Wert *rYMin*erreicht hat.

bActive: *bActive* ist TRUE**,** wenn der Regler aktiv und freigegeben ist.

bARWactive: *bARWactive*ist TRUE**,** wenn der Integralanteil des Reglers die untere oder obere Stellgrößenlimitierung erreicht hat.

eState: Status vom Regler. Siehe E_HVACState. [▶ 532]

bError: Der Ausgang signalisiert mit einem TRUE, dass ein Fehler anliegt.

eErrorCode: Enthält den befehlsspezifischen Fehlercode. Siehe E_HVACErrorCodes [▶ 529].

bInvalidParameter: TRUE, wenn bei der Plausibilitätsüberprüfung ein Fehler aufgetreten ist. Die Meldung muss mit *bReset* quittiert werden.

VAR_IN_OUT

```
rDeadRange      : REAL;
bDirection      : BOOL;
rKp             : REAL;
tTi             : TIME;
tTv             : TIME;
tTd             : TIME;
rYMin           : REAL;
rYMax           : REAL;
```

rDeadRange: Um unnötiges Verfahren und damit frühzeitiges Verschleißen der Ventile oder Klappenantriebe zu vermeiden, kann für das Ausgangssignal *rY* des Reglers eine Totzone (0..32767) eingestellt werden. Eine Stellsignaländerung wird damit erst aktiv, wenn die Wertänderung größer als die Totzone ist. Eine stetige Änderung des Stellsignals *rY* wird bei Angabe einer Totzone in ein pulsierendes Verfahren des Stellorgans umgewandelt. Je größer die Totzone, desto größer sind die Pausen und Stellsignalsprünge. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

bDirection: Mit dem Parameter *bDirection* kann der Wirksinn des Reglers verändert werden. Ist *bDirection* TRUE**,** ist der direkte Wirksinn für einen Kühlbetrieb des Reglers aktiv. Wenn *bDirection* FALSE ist, ist der indirekte Wirksinn des Reglers für den Heizbetrieb aktiviert. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rKp: Proportionalfaktor (0,01..100) Verstärkung. Die Variable wird persistent gespeichert. Voreingestellt auf 1,0.

tTi: Integrierzeit [s]. Der I-Anteil korrigiert die verbleibende Regelabweichung nach der Korrektur des P-Anteils. Je kleiner die *tTi*-Zeit eingestellt wird, desto schneller korrigiert der Regler. Ist die Zeit zu kurz, wird der Regelkreis instabil. Um den Integrationsanteil zu vermindern, sind größere *tTi*-Zeiten einzugeben. Die Nachstellzeit sollte größer als die Verfahrzeit des Ventil- oder Klappenantriebes gewählt werden. Die Variable wird persistent gespeichert. Voreingestellt auf 30s.

tTv: Vorhaltezeit [s]. Je größer *tTv*ist, desto stärker korrigiert der Regler. Eine zu große Zeit führt zu einem instabilen Regelkreis. In normalen Anwendungen der Gebäudeautomation wird häufig nur ein PI-Regler verwendet. In diesem Fall muss für *tTv* Null eingegeben werden. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

tTd: Dämpfungszeit [s]. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

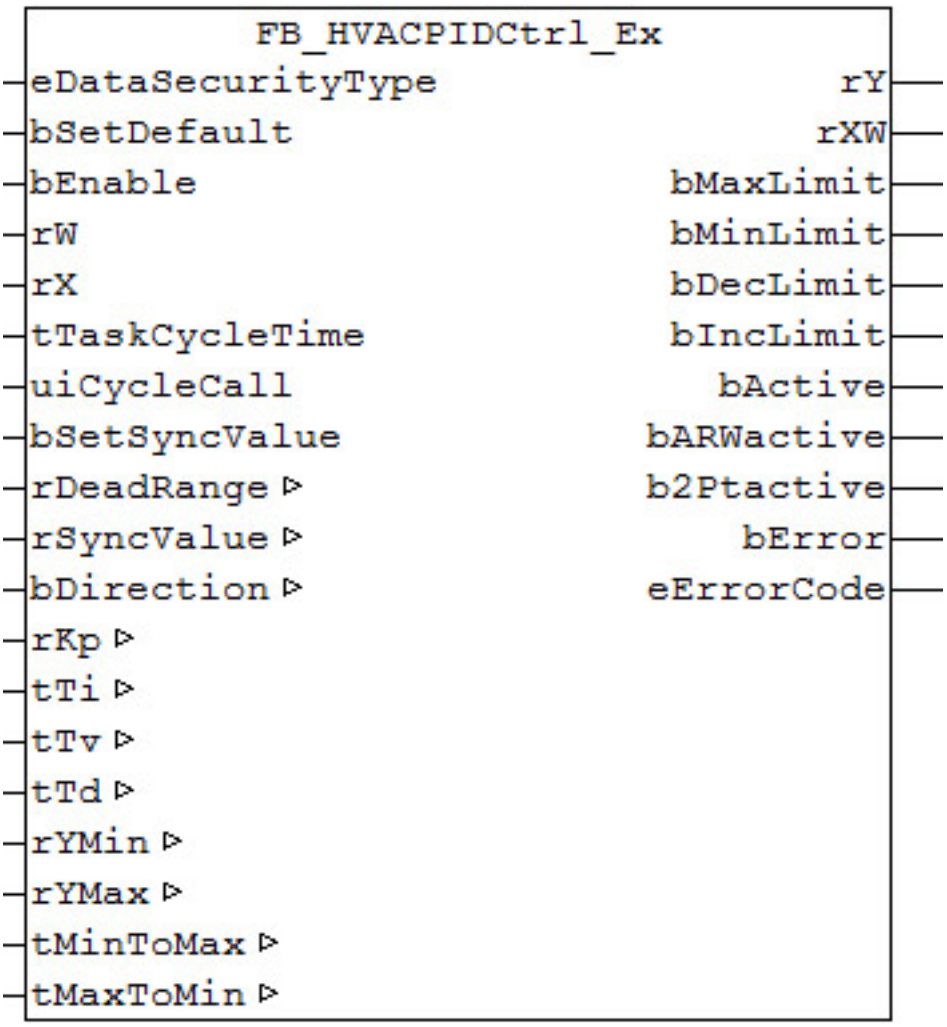
rYMin / rYMax: Begrenzen den Arbeitsbereich des Reglers (0..32767). Einige andere Funktionsbausteine z.B. Sequenzer erfordern einen symmetrischen (- 100 bis +100) Stellbereich. Bei einer Kaskadenstruktur bestimmt der Arbeitsbereich des Führungsreglers den Sollwert des Folgereglers. Zum Beispiel 15° bis 25° als Begrenzung des Zulufttemperturs ollwerts einer Abluft- Zuluftkaskadenregelung. Die Variablen werden persistent gespeichert. *rYMin* voreingestellt auf 0. *rYMax* voreingestellt auf 100.

Dokumente hierzu

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

3.4.5 FB_HVACPIDCtrl_Ex

Universeller PID-Regler.



Ein-Ausgänge

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
bEnable          : BOOL;
rW               : REAL;
rX               : REAL;
tTaskCycleTime   : TIME;
uiCycleCall      : UINT;
bSetSyncValue    : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistentist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Regleraktivierung. **Im Moment der Aktivierung reagiert der Regler unmittelbar auf die Regelabweichung, ohne interne Synchronisation auf einen Wert.**

rW : Sollwert

rX : Istwert

tTaskCycleTime: Zykluszeit, mit der der Funktionsbaustein aufgerufen wird. Diese entspricht der Task-Zykluszeit der aufrufenden Task, wenn der Baustein in jedem Zyklus aufgerufen wird.

uiCycleCall : Aufrufzyklus des Bausteines als Vielfaches der Zykluszeit. Ein Nulleintrag wird automatisch als *uiCycleCall* =1 gewertet.
Beispiel: *tTaskCycleTime* = 20ms, *uiCycleCall* =10 -> Der Regelalgorithmus wird alle 200ms aufgerufen. Damit werden aber auch nur alle 200ms die Ausgänge aktualisiert.

bSetSyncValue: Mit einer steigenden Flanke am Eingang *bSetSyncValue* wird die Stellgröße *rY* auf den Wert *rSyncValue* (VAR_IN_OUT) gesetzt. Dazu wird intern der I-Anteil verändert. Ist der I-Anteil nicht gegeben (PD-Regler), so wird der D-Anteil verändert.

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 : Stellgröße. Bereich durch *rYMin* und *rYMax* eingeschränkt.

rXW : Regelabweichung (Berechnung abhängig vom Wirksinn [▶ 279])

bMaxLimit : Die Stellgröße hat ihren oberen Grenzwert erreicht.

bMinLimit : Die Stellgröße hat ihren unteren Grenzwert erreicht.

bDecLimit : Die Stellgrößen-Steigung hat ihren Grenzwert für das maximale Abfallen erreicht, siehe *tMaxToMin* (VAR_IN_OUT).

bIncLimit : Die Stellgrößen-Steigung hat ihren Grenzwert für den maximale Anstieg erreicht, siehe *tMinToMax* (VAR_IN_OUT).

bActive : Der Regler ist aktiv, das heißt freigegeben (bEnable = TRUE) und nicht im Fehlerzustand (bError = FALSE).

bARWactive : Anti-Reset-Windup-Funktion ist aktiv.

b2Ptactive : Das Zweipunktverhalten des Reglers ist aktiv.

bError : Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

eErrorCode : Enthält den befehlsspezifischen Fehlercode. Siehe E_HVACErrorCodes [▶ 529].

VAR_IN_OUT

```
rDeadRange      : REAL;
rSyncValue      : REAL;
bDirection      : BOOL;
rKp             : REAL;
tTi             : TIME;
tTv             : TIME;
tTd             : TIME;
rYMin           : REAL;
rYMax           : REAL;
tMinToMax       : TIME;
tMaxToMin       : TIME;
```

rDeadRange: Um unnötiges Verfahren und damit frühzeitiges Verschleißen der Ventile oder Klappenantriebe zu vermeiden, kann für Die Regelabweichung eine Totzone eingestellt werden. Ist der Betrag der Regelabweichung kleiner als die Hälfte der Totzone *rDeadRange*, so wird die P-I-D-Berechnung und damit der Stellausgang *rY* "eingefroren". Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rSyncValue: Mit einer steigenden Flanke am Eingang *bSync* wird die Stellgröße *rY* auf diesen Wert gesetzt. Dazu wird intern der I-Anteil verändert. Ist der I-Anteil nicht gegeben (PD-Regler), so wird der D-Anteil verändert. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

bDirection: Mit dem Parameter *bDirection* kann der Wirksinn [▶ 279] des Reglers verändert werden. Ist *bDirection* TRUE**,** ist der direkte Wirksinn für einen Kühlbetrieb des Reglers aktiv. Wenn *bDirection* FALSE ist, ist der indirekte Wirksinn des Reglers für den Heizbetrieb aktiviert. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rKp: Proportionalfaktor Verstärkung. Wirkt nur auf den P-Anteil. Die Variable wird persistent gespeichert. Voreingestellt auf 1,0.

tTi: Integrierzeit [s]. Der I-Anteil korrigiert die verbleibende Regelabweichung nach der Korrektur des P-Anteils. Je kleiner die *tTi* eingestellt wird, desto schneller korrigiert der Regler. Ist die Zeit zu kurz, wird der Regelkreis instabil. Um den Integrationsanteil zu vermindern, sind größere *tTi*-Zeiten einzugeben. Die Nachstellzeit sollte größer als die Verfahrzeit des Ventil- oder Klappenantriebes gewählt werden. Ein Nullwert an diesem Parameter schaltet den I-Anteil ab. Die Variable wird persistent gespeichert. Voreingestellt auf 30s.

tTv: Vorhaltezeit [s]. Je größer *tTv* ist, desto stärker korrigiert der Regler. Eine zu große Zeit führt zu einem instabilen Regelkreis. In normalen Anwendungen der Gebäudeautomation wird häufig nur ein PI-Regler verwendet. Ein Nullwert an diesem Parameter schaltet den D-Anteil ab. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

tTd: Dämpfungszeit [s]. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

rYMin / rYMax: Begrenzen den Arbeitsbereich des Reglers. Einige andere Funktionsbausteine z.B. Sequenzer erfordern einen symmetrischen (- 100 bis +100) Stellbereich. Bei einer Kaskadenstruktur bestimmt der Arbeitsbereich des Führungsreglers den Sollwert des Folgereglers. Zum Beispiel 15° bis 25° als Begrenzung des Zulufttempertursollwerts einer Abluft- Zuluftkaskadenregelung. Die Variablen werden persistent gespeichert. *rYMin* voreingestellt auf 0. *rYMax* voreingestellt auf 100.

tMinToMax: Steigungsbegrenzung [s] des Reglerausgangs für Anstieg: *tMinToMax* in Sekunden bezogen auf eine Änderung von *rYMin* auf *rYMax*. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

tMaxToMin: Steigungsbegrenzung [s] des Reglerausgangs für Abfall: *tMaxToMin* in Sekunden bezogen auf eine Änderung von *rYMax* auf *rYMin*. Die Variable wird persistent gespeichert. Voreingestellt auf 0s

Funktionsbeschreibung

Passiv-Verhalten (bEnable = FALSE oder bError = TRUE)

Die Ausgänge werden wie folgt gesetzt:

rY	0,0
rXW	0,0
bMaxLimit	FALSE
bMinLimit	FALSE

bDecLimit	FALSE
bIncLimit	FALSE
bActive	FALSE
bARWactive	FALSE
b2Ptactive	FALSE

Im Fehlerfall steht *bError* auf TRUE - *eErrorCode* gibt den aktuellen Fehlercode an. Die internen Werte für P-, I-, und D-Anteil werden auf 0 gesetzt, ebenso die Werte für den I- und D-Anteil vom vorhergehenden Zyklus. Damit wird die Stellgröße bei einem Neustart im ersten Zyklus "sauber", das heißt ohne Vergangenheitswerte berechnet.

Aktiv-Verhalten (bEnable = TRUE und bError = FALSE)

Im ersten Zyklus werden I- und D-Anteil wie bereits erwähnt ohne Vergangenheitswerte berechnet und somit "sauber" aufgestartet. Ein positives Signal an *bSetSyncValue* setzt den I-Anteil so, daß die Stellgröße den Wert *rSyncValue* annimmt. Diese Methode kann, wenn *bEnable* und *bSetSyncValue* gleichzeitig gesetzt werden, zum Setzen eines Initialwertes genutzt werden, von dem aus die Regelung "losläuft". Ist der I-Anteil nicht aktiv, so wird der D-Anteil entsprechend gesetzt. Zu beachten ist, daß intern nur die steigende Flanke von *bSetSyncValue* ausgewertet wird, handelt es sich doch um eine Setz-Aktion. Für ein erneutes Synchronisieren, etwa auf einen Übergabewert, muß am Eingang *bSetSyncValue* ein erneutes TRUE-Signal angelegt werden. Ist der I-Anteil aktiv so sorgt der Regler dafür, daß dieser fest gehalten wird, sollte sich der Reglerausgang *rY* an den Grenzen *rYMin* oder *rYMax* befinden und noch weiter fallen bzw steigen wollen. Dieses Verfahren wird Anti-Wind-Up genannt unt trägt dafür Sorge, dass der I-Anteil immer nur so groß ist, dass die Stellgröße bei entsprechender Regelabweichung sofort wieder Werte innerhalb der Grenzen annehmen kann, ohne dass ein zu groß gewordener Integralanteich zunächst noch abgebaut werden muss.

Wirksinn

Wirksinn

Mit *bDirection* = FALSE wird der Wirksinn des Reglers so umgekehrt , dass eine Regelabweichung kleiner als 0 eine Stellgrößenänderung ins Positive bewirkt. Dies wird dadurch erreicht, daß die Regelabweichung negativ berechnet wird:

bDirection	**rXW (Regelabweichung)**	Wirksinn
TRUE	lrX-lrW (Istwert-Sollwert)	direkt (Kühlen)
FALSE	lrW-lrX (Sollwert-Istwert)	indirekt (Heizen)

Anti-Reset-Windup bei Erreichen des Maximal- bzw Minimalwertes

Erreicht der Regler am Ausgang seine Obergrenze und ist die Regelabweichung weiterhin positiv, so wird der Integralanteil sich so lange weiter erhöhen, bis die Regelabweichung wieder kleiner oder gleich Null ist. Dies kann dazu führen, daß sich unnötigerweise ein sehr hoher Integralanteil aufbaut, der bei einer Vorzeichenänderung der Regelabweichung erst wieder abgebaut werden muss und das Regelverhaten träge macht. Das gleiche gilt auch bei Erreichen des Minimalwertes am Ausgang bei einer weiterhin negativen Regelabweichung.
Um diesem vorzubeugen wird bei einem Austritt aus dem Regelbereich der I-Anteil so gesetzt, daß er in Addition mit dem P- und D-Anteil den jeweiligen Grenzwert *lrYmin* oder *lrYMax* am Stellausgang erreicht. Die weitere Berechnung der P- I- und D-Werte wird dann so lange unterbrochen, bis das Vorzeichen der Regelabweichung wieder einen Eintritt in den Regelbereich zuläßt, d.h. bei einem Verharren an der Maximalgrenze und eine Regelabweichung kleiner als 0.0 bzw. bei einem Verharren an der Minimalgrenze und eine Regelabweichung größer als 0.0.

Auch im SPS-Zyklus des Wiedereintrittes wird der Ausgang *lrY* durch Manipulation des I-Anteiles so gesetzt, dass er sich nicht sprunghaft in den Regelbereich bewegt, sondern von der Grenze des vorher gehenden Verharrens aus anfängt sich zu ändern.

Steigungsbegrenzung

Ist der Regler schneller eingestellt, als das Stellglied, so kann dieses dem Regler nicht folgen, was zu eimem Schwingverhalten führen kann. Daher gibt es die Möglichkeit die Steigung der Stellgröße zu begrenzen.

Als Maß dafür gelten die Zeiten:

tMinToMax : Steigungsbegrenzung des Reglerausgangs für Anstieg: *tMinToMax* in Sekunden bezogen auf eine Änderung von *rYMin* auf *rYMax*.

tMaxToMin : Steigungsbegrenzung des Reglerausgangs für Abfall: *tMaxToMin* in Sekunden bezogen auf eine Änderung von *rYMax* auf *rYMin*.

Hieraus läßt sich jeweils die Maximale Änderung pro SPS-Zyklus - das maximale Inkrement oder Dekrement - errechnen.

Ist die berechnete Änderung des Stellsignals über einen SPS-Zyklus nun höher als die unter *tMinToMax* bzw. *tMaxToMin* eingestellte, so wird das Stellsignal lediglich um das maximale Inkrement oder Dekrement erhöht bzw erniedrigt.

Der I-Anteil wird dabei intern auf dieselbe Art nachgeführt (I-Anteil des letztenSPS-Zyklus + maximales Inkrement bzw. I-Anteil des letztenSPS-Zyklus - maximales Dekrement).

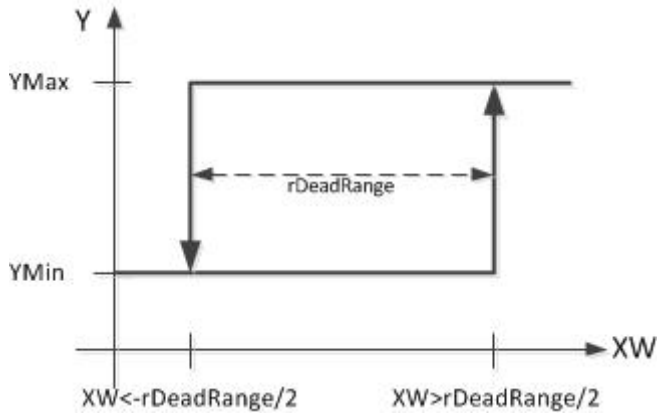
Totzone

Ein Wert von *rDeadRange*> 0.0 gibt die Funktion der Totzone frei. Ist der Absolutwert der Regelabweichung dann kleiner als 1/2 *rDeadRange*, so ist die Neutrale Zone aktiv und P-, I- und D-Anteil werden nicht weiter berechnet, was zu einem gleich bleibenden Stellsignal führt. Erst wenn die Regelabweichung wieder größer wird, werden die Anteile neu berechnet und das Ausgangssignal des Reglers ändert dich wieder.

Durch diese Funktion sollen unnötig viele Stellimpulse vermieden werden.

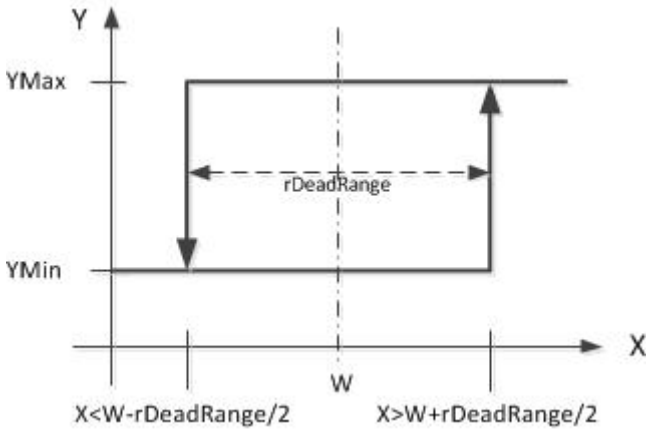
Zweipunkt-Regelverhalten

Sind die Regelparameterlr *rKp*, *tTn*, *tTv* und *tTd* auf 0.0 bzw. t#0s gesetzt, so hat der Regler per Definition ein Zweipunkt-Verhalten.
Die Totzone *rDeadRange* definiert dabei die Hysterese. Das Schalten auf den Maximalwert am Ausgang *rY* erfolgt grundsätzlich, wenn die Regelabweichung *rXW* größer als die Hälfte des Hysteresewertes *rDeadRange* ist - auf den Minimalwert wird immer dann geschaltet, wenn die Regelabweichung *rXW* kleiner als die negative Hälfte der Hysterese *rDeadRange* wird:

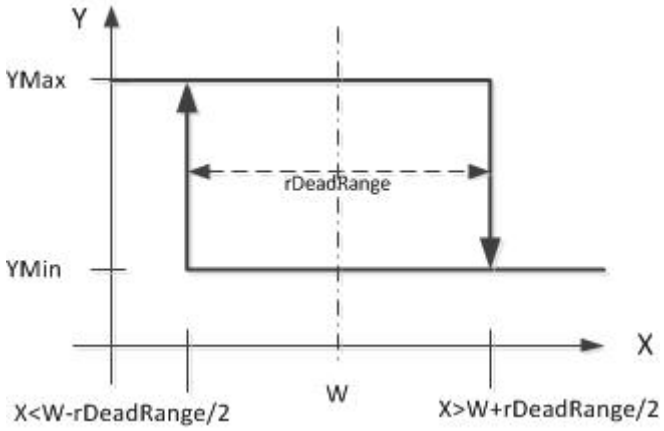


Aus der unterschiedlichen Berechnung der Regelabweichung bei direktem und indirektem Wirksinn ergibt sich dann folgendes Schaltverhalten bezogen auf den Istwert *rX*:

- direkter Wirksinn (Kühlen): Regelabweichung = Istwert-Sollwert *rXW* = *rX-rW*



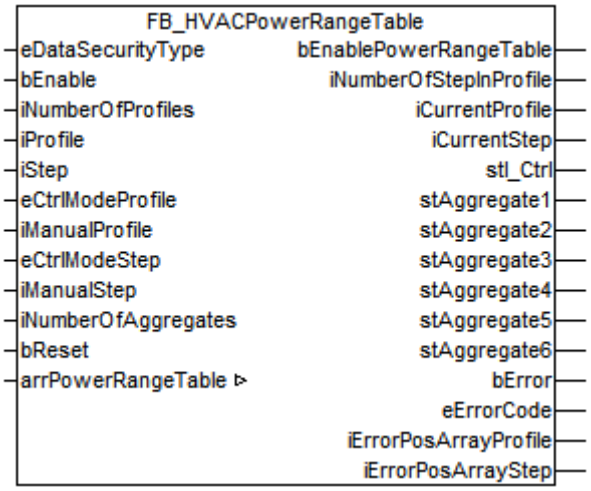
- indirekter Wirksinn (Heizen): Regelabweichung = Istwert-Sollwert *rXW* = *rW-rX*
-



Dokumente hierzu

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

3.4.6 FB_HVACPowerRangeTable



Anwendung

Anwendung

Der Funktionsbaustein stellt eine Leistungsstufentabelle dar und dient zur sequentiellen Regelung von Leistungserzeugern wie beispielsweise Heizkesseln oder Kältemaschinen. Die Leistungsstufen werden von dem vorgelagerten Regler FB_HVACI_Ctrl_Step [▶ 255] bestimmt und der Leistungsstufentabelle über den Eingang *iStep* übergeben.

Alle für die Anlagensteuerung relevanten Informationen bzw. Parameter sind in dem Leistungsstufentabellen-ARRAY *arrPowerRangeTable* zusammengefasst. Die Leistungsstufentabelle ist ein zweidimensionales Array mit der Struktur ST_HVACPowerRange [▶ 537]. Die Leistungsstufentabelle kann mit einer normalen Tabelle verglichen werden. Ein einzelnes Element ist durch Nennung von Zeile und Spalte eindeutig bezeichnet. Die Spalten sind durch den Feldbereich 1..g_iMaxNumberOfProfiles [▶ 541] gekennzeichnet. Dieser Bereich wird als Profil bezeichnet und wird über die Eingangsvariable *iProfile* angesprochen. Der Feldbereich 0..g_iMaxNumberOfSteps [▶ 541] stellt die Zeilen dar. Dieser Bereich wird als Stufe bezeichnet und wird über die Eingangsvariable *iStep* angesprochen.

Die Leistungsstufentabelle besteht aus 16 Profilen. Jedes Profil kann bis zu 33 Stufen haben, von 0 bis 32. Jede einzelne Stufe beinhaltet die für die Anlagensteuerung benötigte Parameterstruktur ST_HVACPowerRange [▶ 537]. Die hier angegebenen Parameter werden über die Ausgangsstrukturen *stI_Ctrl* und *stAggregate1-6* dem Funktionsbaustein FB_HVACI_CtrlStep [▶ 255] zur Steuerung der Stufen und den Funktionsbausteinen zur Energieerzeugung übergeben.

Wieviele Stufen sich in einem Profil befinden, wird über die AusgangsvariableiNumberOfStepInProfileangegeben.Die Anzahl ist abhängig von den Einträgen in der Leistungsstufentabelle und von der Angabe des ausgewählte ProfilsiCurrentProfile.Intern wird das ausgewählte Profil von der Stufe 1 bis zu der Stufe überprüft, in der alle Variablen der Struktur ST_HVACPowerRange [▶ 537] den Wert 0 haben. *iNumberOfStepInProfile* wird immer nur für das vorgegebene Profil bestimmt. *iNumberOfStepInProfile* kann somit als Begrenzung der Stufen in einem Profil für andere Funktionsbausteine verwendet werden wie z.B. für den Eingang iNumberOfStepInProfile des FB_HVACI_CtrlStep [▶ 255], der über seinen Ausgang iStep die einzelnen Stufen des FB_HVACPowerRangeTable steuern könnte. Die Stufe 0 wird bei der Auswertung der Anzahl der Stufen über *iNumberOfStepInProfile* nicht berücksichtigt, weil in dieser Stufe die Leistungserzeuger ausgeschaltet sind, sie kann als Bereitschaftsstufe gesehen werden. Die Werte der Variablen der Ausgangsstrukturen *stAggregate1-6* geben dann den Wert 0 aus. In der Bereitschaftsstufe werden dem vorgelagerten Regler FB_HVACI_Ctrl_Step [▶ 255] über die Ausgangsstruktur *stI_Ctrl* die Parameter übergeben, die für das Starten der Leistungsstufensequenz der Energieerzeuger notwendig sind.

Tabellenbeispiel zur Leistungsstufentabelle arrPowerRangeTable

Innerhalb eines Profils (Tabellenspalten) wird jeder Leistungsstufe ein Aggregat zugeordnet.
Damit wird die Reihenfolge bestimmt mit der die Leistungserzeuger innerhalb einer Sequenz zu- oder ausgeschaltet werden.
Die Leistung der Aggregate wird bei stufigen Leistungserzeugern von 0 bis 6 angegeben, bei stetigen Leistungserzeugern von 0 bis 100%.
Nach dem Umschalten in eine höhere oder niedrigere Leistungsstufe wird der Integralanteil des vorgelagerten Funktionsbausteins FB_HVACI_Ctrl_Step [▶ 255] neu initialisiert. Dem Regler werden dann mittels der Struktur *stI_Ctrl* die aktuellen Werte für das Hoch- und Rückschaltintegral übergeben.
Für einen Folgewechsel der Aggregate innerhalb der Sequenz kann auf ein anderes Profil umgeschaltet werden. Durch geschicktes Eintragen unterschiedlicher Reihenfolgen der Aggregate in den Profilen kann äußerst flexibel auf alle Anforderungen bezüglich der Führung von Leistungserzeugern reagiert werden.
Das Umschalten der Leistungsprofile (*iCurrentProfile*) erfolgt in der Regel zeit- oder betriebsstundenabhängig damit alle Leistungserzeuger einer Anlage gleichmäßig ausgelastet sind.
Bei Leistungserzeugern unterschiedlicher Nennleistung wird das Leistungsprofil lastabhängig geändert.

Ist ein Aggregat innerhalb einer Erzeugergruppe gestört kann auf ein Profil umgeschaltet werden in dem der gestörte Erzeuger an das Ende der Leistungsfolge gestellt wird.

	ST_HVACPowerRange	iCurrentProfile = 1	iCurrentProfile = 2	iCurrentProfile = 3
iCurrentStep = 0	iAggregate	0	0	0
	iAggregateStep	0	0	0
	rY_Max	0	0	0
	rY_Min	0	0	0
	bBlock	FALSE	FALSE	FALSE
	rIntegralHigh	2	3	4
	rIntegralLow	2	3	4
	udiSecDelayHigh	300	305	295
	udiSecDelayLow	30	35	25
iCurrentStep = 1	iAggregate	1	3	5
	iAggregateStep	1	1	1
	rY_Max			
	rY_Min			
	bBlock	FALSE	FALSE	FALSE
	rIntegralHigh	15	16	17
	rIntegralLow	5	6	7
	udiSecDelayHigh	300	305	295
	udiSecDelayLow	30	35	25
iCurrentStep = 2	iAggregate	2	3	4
	iAggregateStep	1	2	1
	rY_Max		60	
	rY_Min		30	
	bBlock	TRUE	FALSE	TRUE
	rIntegralHigh	10	11	7
	rIntegralLow	10	10	7
	udiSecDelayHigh	300	88	400
	udiSecDelayLow	300	150	400
iCurrentStep = 3	iAggregate	0	1	3
	iAggregateStep	0	1	1
	rY_Max	0	0	
	rY_Min	0	0	
	bBlock	FALSE	FALSE	FALSE
	rIntegralHigh	0	8	7
	rIntegralLow	0	8	7
	udiSecDelayHigh	0	23	400
	udiSecDelayLow	0	123	200
iCurrentStep = 4	iAggregate	2	0	0
	iAggregateStep	2	0	0
	rY_Max	70	0	0
	rY_Min	40	0	0
	bBlock	FALSE	FALSE	FALSE
	rIntegralHigh	10	0	0
	rIntegralLow	10	0	0
	udiSecDelayHigh	320	0	0
	udiSecDelayLow	233	0	0

Die Leistungsstufentabelle zeigt drei Leistungsprofile. In jedem Leistungsprofil ist die Folge der Leistungserzeuger eingetragen.

Profil 1:
In dem Profil 1 wird in der Leistungsstufe 1 das Aggregat 1 in die erste Leistungsstufe geschaltet. In der 2. Leistungsstufe wird das Aggregat 2 mit seiner 1. Stufe hinzu geschaltet.

Profil 2:
In dem Profil 2 wird in der Leistungsstufe 1 das Aggregat 3 in die erste Leistungsstufe geschaltet. In der 2. Leistungsstufe wird das Aggregat 3 in seiner 2. Stufe geschaltet. In der 3. Leistungsstufe wird das Aggregat 1 mit seiner 1. Stufe hinzu geschaltet.

Profil 3:
In dem Profil 3 wird in der Leistungsstufe 1 das Aggregat 5 in die erste Leistungsstufe geschaltet. In der 2. Leistungsstufe wird das Aggregat 4 mit seiner 1. Stufe hinzu geschaltet. In der 3. Leistungsstufe wird das Aggregat 3 mit seiner 1. Stufe hinzu geschaltet.

Durch geschicktes Anordnen der Leistungsprofile kann auf die Störung einzelner Aggregate, dem betriebsstundenabhängigen Folgewechsel oder auf den lastoptimierten Wechsel von Leistungsfolgen reagiert werden.

Anhand des Tabellenbeispiels soll der Zusammenhang zwischen der Leistungsstufentabelle *arrPowerRange* und den Ausgangsstrukturen *stAggregate1-6* und *stI_Ctrl* näher erläutert werden:

1. Die Spalte **iCurrentProfile= 1** soll das 1. Profil der Leistungsstufentabelle *arrPowerRange* darstellen. In diesem Profil hat der Ausgang *iNumberOfStepInProfile* den Wert 2, weil in der Zeile **iCurrentStep= 3** kein Variablenwert der Struktur ST_HVACPowerRange [▶ 537](rot makiert) größer 0 ist.
   Nimmt man an, dass die aktuelle Leistungsstufe *iCurrentStep* den Wert 2 hat, so werden die Variablenwerte wie folgt an den Ausgangsstrukturen ausgegeben:

   arrPowerRangeTable[] **iCurrentProfile,iCurrentStep**;

   stAggregate1.iAggregateStep :=*arrPowerRangeTable[,].iAggregateStep***11**;
   stAggregate1.rY_Max := *arrPowerRangeTable[,].rY_Max***11**;
   stAggregate1.rY_Min :=*arrPowerRangeTable[,].rY_Min***11**;
   stAggregate1.bBlock := *arrPowerRangeTable[,].bBlock***11**;

   stAggregate2.iAggregateStep :=*arrPowerRangeTable[,].iAggregateStep***12**;
   stAggregate2.rY_Max := *arrPowerRangeTable[,].rY_Max***12**;
   stAggregate2.rY_Min :=*arrPowerRangeTable[,].rY_Min***12**;
   stAggregate2.bBlock := *arrPowerRangeTable[,].bBlock***12**;
   stI_Ctrl.rIntegralHigh := *arrPowerRangeTable[,].rIntegralHigh***12**;
   stI_Ctrl.rIntegralLow := *arrPowerRangeTable[,].rIntegralLow***12**;
   stI_Ctrl.udiSecDelayHigh :=*arrPowerRangeTable[,].udiSecDelayHigh***12**;
   stI_Ctrl.udiSecDelayLow :=*arrPowerRangeTable[,].udiSecDelayLow***12**;

   Inhalte der Ausgangsstrukturen *stAggregate1, stAggregate2* und *stI_Ctrl*. Die Variablen der Ausgangstrukturen *stAggregate3-6* haben den Wert 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. Die Spalte **iCurrentProfile= 2** stellt das 2. Profil der Leistungsstufentabelle *arrPowerRange* dar. In diesem Profil hat der Ausgang *iNumberOfStepInProfile* den Wert 3, weil in der Zeile **iCurrentStep= 4** kein Variablenwert der Struktur ST_HVACPowerRange [▶ 537](rot makiert) größer 0 ist.
   Nimmt man an, dass die aktuelle Leistungsstufe *iCurrentStep* den Wert 3 hat, so werden die Variablenwerte wie folgt an den Ausgangsstrukturen ausgegeben:

   arrPowerRangeTable[] **iCurrentProfile,iCurrentStep**;

   stAggregate1.iAggregateStep :=*arrPowerRangeTable[,].iAggregateStep***23**;
   stAggregate1.rY_Max := *arrPowerRangeTable[,].rY_Max***23**;

stAggregate1.rY_Min :=*arrPowerRangeTable[,].rY_Min***23**;
stAggregate1.bBlock := *arrPowerRangeTable[,].bBlock***23**;

stAggregate3.iAggregateStep :=*arrPowerRangeTable[,.iAggregateStep***22]**;
stAggregate3.rY_Max := *arrPowerRangeTable[,].rY_Max***22**;
stAggregate3.rY_Min :=*arrPowerRangeTable[,].rY_Min***22**;
stAggregate3.bBlock := *arrPowerRangeTable[,].bBlock***22**;

stI_Ctrl.rIntegralHigh := *arrPowerRangeTable[,].rIntegralHigh***23**;
stI_Ctrl.rIntegralLow := *arrPowerRangeTable[,].rIntegralLow***23**;
stI_Ctrl.udiSecDelayHigh :=*arrPowerRangeTable[,].udiSecDelayHigh***23**;
stI_Ctrl.udiSecDelayLow :=*arrPowerRangeTable[,].udiSecDelayLow***23**;

Inhalte der Ausgangsstrukturen *stAggregate1, stAggregate3* und *stI_Ctrl.* Die Variablen der Ausgangstrukturen *stAggregate2* und *stAggregate4-6* haben den Wert 0.

stAggregate1.iAggregateStep :=1;
stAggregate1.rY_Max := 0;
stAggregate1.rY_Min :=0;
stAggregate1.bBlock := FALSE;

stAggregate3.iAggregateStep :=2;
stAggregate3.rY_Max := 60;
stAggregate3.rY_Min :=30;
stAggregate3.bBlock := FALSE;

stI_Ctrl.rIntegralHigh := 8;
stI_Ctrl.rIntegralLow := 8;
stI_Ctrl.udiSecDelayHigh :=23;
stI_Ctrl.udiSecDelayLow :=123;

3. Die Spalte **iCurrentProfile= 3** stellt das 3. Profil der Leistungsstufentabelle *arrPowerRange* dar. In diesem Profil hat der Ausgang *iNumberOfStepInProfile* den Wert 3, weil in der Zeile **iCurrentStep= 4** kein Variablenwert der Struktur ST_HVACPowerRange [▶ 537](rot makiert) größer 0 ist.
   Nimmt man an, dass die aktuelle Leistungsstufe *iCurrentStep* den Wert 3 hat, so werden die Variablenwerte wie folgt an den Ausgangsstrukturen ausgegeben:

arrPowerRangeTable[] **iCurrentProfile,iCurrentStep**;

stAggregate3.iAggregateStep :=*arrPowerRangeTable[,].iAggregateStep***33**;
stAggregate3.rY_Max := *arrPowerRangeTable[,].rY_Max***33**;
stAggregate3.rY_Min :=*arrPowerRangeTable[,].rY_Min***33**;
stAggregate3.bBlock := *arrPowerRangeTable[,].bBlock***33**;

stAggregate4.iAggregateStep :=*arrPowerRangeTable[,.iAggregateStep***32]**;
stAggregate4.rY_Max := *arrPowerRangeTable[,].rY_Max***32**;
stAggregate4.rY_Min :=*arrPowerRangeTable[,].rY_Min***32**;
stAggregate4.bBlock := *arrPowerRangeTable[,].bBlock***32**;

stAggregate5.iAggregateStep :=*arrPowerRangeTable[,.iAggregateStep***31]**;
stAggregate5.rY_Max := *arrPowerRangeTable[,].rY_Max***31**;
stAggregate5.rY_Min :=*arrPowerRangeTable[,].rY_Min***31**;
stAggregate5.bBlock := *arrPowerRangeTable[,].bBlock***31**;

stI_Ctrl.rIntegralHigh := *arrPowerRangeTable[,].rIntegralHigh***33**;
stI_Ctrl.rIntegralLow := *arrPowerRangeTable[,].rIntegralLow***33**;
stI_Ctrl.udiSecDelayHigh :=*arrPowerRangeTable[,].udiSecDelayHigh***33**;
stI_Ctrl.udiSecDelayLow :=*arrPowerRangeTable[,].udiSecDelayLow***33**;

Inhalte der Ausgangsstrukturen *stAggregate3, stAggregate4, stAggregate5* und *stI_Ctrl.* Die Variablen der Ausgangstrukturen *stAggregate1, stAggregate2* und *stAggregate6* haben den Wert 0.

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

```
stAggregate4.iAggregateStep :=1;
stAggregate4.rY_Max := 0;
stAggregate4.rY_Min :=0;
stAggregate4.bBlock := TRUE;

stAggregate5.iAggregateStep :=1;
stAggregate5.rY_Max := 0;
stAggregate5.rY_Min :=0;
stAggregate5.bBlock := FALSE;

stI_Ctrl.rIntegralHigh  := 7;
stI_Ctrl.rIntegralLow  := 7;
stI_Ctrl.udiSecDelayHigh  :=400;
stI_Ctrl.udiSecDelayLow  :=200;
```

Ausgangsstruktur stI_Ctrl abgestimmt auf das Beispiel

		iCurrentProfile = 1	iCurrentProfile = 2	iCurrentProfile = 3
		stI_Ctrl	stI_Ctrl	stI_Ctrl
iCurrentStep = 0	rIntegralHigh	2	3	4
	rIntegralLow	2	3	4
	udiSecDelayHigh	300	305	295
	udiSecDelayLow	30	35	25
iCurrentStep = 1	rIntegralHigh	15	16	17
	rIntegralLow	5	6	7
	udiSecDelayHigh	300	305	295
	udiSecDelayLow	30	35	25
iCurrentStep = 2	rIntegralHigh	10	11	7
	rIntegralLow	10	10	7
	udiSecDelayHigh	300	88	400
	udiSecDelayLow	300	150	400
iCurrentStep = 3	rIntegralHigh	0	8	7
	rIntegralLow	0	8	7
	udiSecDelayHigh	0	23	400
	udiSecDelayLow	0	123	200
iCurrentStep = 4	rIntegralHigh	0	0	0
	rIntegralLow	0	0	0
	udiSecDelayHigh	0	0	0
	udiSecDelayLow	0	0	0

Ausgangsstrukturen stAggregate1-6 abgestimmt auf das Beispiel

		iCurrentProfile = 1						iCurrentProfile = 2						iCurrentProfile = 3					
		stAggregate						stAggregate						stAggregate					
		1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6
iCurrentStep = 0	iAggregateStep	0	0					0		0						0	0	0	
	rY_Max	0	0					0		0						0	0	0	
	rY_Min	0	0					0		0						0	0	0	
	bBlock	FALSE	FALSE					FALSE		FALSE						FALSE	FALSE	FALSE	
iCurrentStep = 1	iAggregateStep	1	0					0		1						0	0	1	
	rY_Max	0	0					0		0						0	0	0	
	rY_Min	0	0					0		0						0	0	0	
	bBlock	FALSE	FALSE					FALSE		FALSE						FALSE	FALSE	FALSE	
iCurrentStep = 2	iAggregateStep	1	1					0		2						0	1	1	
	rY_Max	0	0					0		60						0	0	0	
	rY_Min	0	0					0		30						0	0	0	
	bBlock	FALSE	TRUE					FALSE		FALSE						FALSE	TRUE	FALSE	
iCurrentStep = 3	iAggregateStep	0	0					1		2						1	1	1	
	rY_Max	0	0					0		60						0	0	0	
	rY_Min	0	0					0		30						0	0	0	
	bBlock	FALSE	FALSE					FALSE		FALSE						FALSE	TRUE	FALSE	
iCurrentStep = 4	iAggregateStep	0	0					0		0						0	0	0	
	rY_Max	0	0					0		0						0	0	0	
	rY_Min	0	0					0		0						0	0	0	
	bBlock	FALSE	FALSE					FALSE		FALSE						FALSE	FALSE	FALSE	

Anwendungsbeispiel

Das Anwendungsbeispiel zeigt den Funktionsbaustein FB_HVACPowerRangeTable in Verbindung mit dem I-Übertragungsglied FB_HVACI_CtrlStep [▶ 255]. Die Darstellung des Beispiels gibt es in den Programmiersprachen ST und CFC. Das Programmbeispiel **P_CFC_I_CtrlStep.PRG** für die Programmiersprachen CFC ist im Ordner **Language CFC** > **Controller** zu finden, das Programmbeispiel **P_ST_I_CtrlStep.PRG** für die Programmiersprachen ST im Ordner **Language Structured Text** > **Controller**.

Download	Benötigte Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

```
eDataSecurityType     : E_HVACDataSecurityType;
bEnable               : BOOL;
iNumberOfProfiles     : INT;
iProfile              : INT;
iStep                 : INT;
eCtrlModeProfile      : E_HVACCtrlMode;
iManualProfile        : INT;
eCtrlModeStep         : E_HVACCtrlMode;
iManualStep           : INT;
iNumberOfAggregates   : INT;
bReset                : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bEnable: Über ein TRUE wird der Funktionsbaustein freigegeben. Ist *bEnable* = FALSE, so werden alle Ausgangsvariablen und Ausgangsstrukturen konstant auf 0 gesetzt. Die Überprüfung der Variablen *iNumberOfProfiles*, *iProfile*, *iStep*, *iNumberOfStepInProfile*, *iNumberOfAggregates*, *arrPowerRangeTable[X,X].iAggregate, arrPowerRangeTable[X,X].iAggregateSteps, arrPowerRangeTable[iP,iS].udiSecDelayHigh,* und *arrPowerRangeTable[iP,iS].udiSecDelayLow* sind weiterhin aktiv. Falls dort ein Fehler auftritt, so wird dieses mit *bError* = TRUE angezeigt und kann nach Behebung des Fehlers mit *bReset* quittiert werden.

iNumberOfProfiles: Anzahl der parametrierten Profile in der Leistungsstufentabelle *arrPowerRangeTable*. Die Angabe der Anzahl der Profile darf g_iMinNumberOfProfiles [▶ 541] nicht unterschreiten und g_iMaxNumberOfProfiles [▶ 541] nicht überschreiten. Ansonsten wird mit *bError* = TRUE ein Fehler angezeigt und die Abarbeitung des Funktionsbausteins wird gestoppt.

iProfile: Angabe des aktuellen Profils mit welchem der Funktionsbaustein aus der Leistungsstufentabelle *arrPowerRangeTable[***iCurrentProfile***,iCurrentStep]* arbeitet. Ist *eCtrlModeProfile* = *eHVACCtrlMode_Auto*, dann ist *iCurrentProfile* = *iProfile*. Die Angabe des Profils darf g_iMinNumberOfProfiles [▶ 541] nicht unterschreiten und *iNumberOfProfiles* nicht überschreiten. Ansonsten wird mit *bError* = TRUE ein Fehler angezeigt und die Abarbeitung des Funktionsbausteins wird gestoppt.

iStep: Angabe der aktuellen Stufe mit welcher der Funktionsbaustein aus der Leistungsstufentabelle *arrPowerRangeTable[iCurrentProfile,]***iCurrentStep** arbeitet. Ist *eCtrlModeStep* = *eHVACCtrlMode_Auto,* dann ist *iCurrentStep* = *iStep*. Die Angabe der Stufen darf g_iMinNumberOfSteps [▶ 541] nicht unterschreiten und *iNumberOfStepInProfile* nicht überschreiten. Ansonsten wird mit *bError* = TRUE ein Fehler angezeigt und die Abarbeitung des Funktionsbausteins wird gestoppt.

eCtrlModeProfile: Über dieses Enum wird entschieden über welche Eingangsvariable die Vorgabe des Profils aus der Leistungsstufentabelle *arrPowerRangeTable[***iCurrentProfile***,iCurrentStep]* angegeben wird. Ist *eCtrlModeProfile* = *eHVACCtrlMode_Auto*, dann ist *iCurrentProfile* = *iProfile*. Ist *eCtrlModeProfile* = *eHVACCtrlMode_Manual*, dann ist *iCurrentProfile* = *iManualProfile.*

iManualProfile: Angabe mit welchem Profil aus der Leistungsstufentabelle *arrPowerRangeTable[***iCurrentProfile***,iCurrentStep]* gearbeitet wird. Ist *eCtrlModeProfile* = *eHVACCtrlMode_Manual*, dann ist *iCurrentProfile* = *iManualProfile*. Die Angabe des Profils darf g_iMinNumberOfProfiles [▶ 541] nicht unterschreiten und *iNumberOfProfiles* nicht überschreiten. Ansonsten wird intern *iManualProfile* =g_iMinNumberOfProfiles [▶ 541] bei Unterschreitung von g_iMinNumberOfProfiles [▶ 541] oder *iManualProfile* = *iNumberOfProfiles* bei Überschreitung von *iNumberOfProfiles* gesetzt.

eCtrlModeStep: Über dieses Enum wird entschieden über welche Eingangsvariable die Vorgabe der Stufen aus der Leistungsstufentabelle *arrPowerRangeTable[iCurrentProfile,]***iCurrentStep** angegeben wird. Ist *eCtrlModeStep* = *eHVACCtrlMode_Auto*, dann ist *iCurrentStep* = *iStep*. Ist *eCtrlModeStep* = *eHVACCtrlMode_Manual*, dann ist *iCurrentStep* = *iManualStep.*

iManualStep: Angabe mit welcher Stufe aus der Leistungsstufentabelle *arrPowerRangeTable[iCurrentProfile,]***iCurrentStep** gearbeitet wird. Ist *eCtrlModeStep* = *eHVACCtrlMode_Manual,* dann ist *iCurrentStep* = *iManualStep*. Die Angabe der Stufen darf g_iMinNumberOfSteps [▶ 541] nicht unterschreiten und *iNumberOfStepInProfile* nicht überschreiten. Ansonsten wird intern *iManualStep* =g_iMinNumberOfSteps [▶ 541] bei Unterschreitung von g_iMinNumberOfProfiles [▶ 541] oder *iManualStep* = *iNumberOfStepInProfile* bei Überschreitung von *iNumberOfStepInProfile* gesetzt.

iNumberOfAggregates: *iNumberOfAggregates* bestimmt die Anzahl der Aggregate in der Leistungserzeugersequenz. Ist z.B. *iNumberOfAggregates* = 4, so werden die Ausgangsstrukturen *stAggregate1-4* je nach Vorgabe von *iCurrentStep/iCurrentProfile* mit den Parametern aus dem Array

arrPowerRangeTable beschrieben. Die Angabe darf g_iMinNumberOfAggregates [▶ 541] nicht unterschreiten und g_iMaxNumberOfAggregates [▶ 541] nicht überschreiten. Ansonsten wird mit bError = TRUE ein Fehler angezeigt und die Abarbeitung des Funktionsbausteins wird gestoppt.

bReset: Eingang zur Quittierung der Störungen nach deren Behebung. Intern wird auf eine steigende Flanke reagiert.

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: Anzeige, dass der Funktionsbaustein frei gegeben ist. *bEnablePowerRangeTable* ist TRUE, wenn *bEnable* = TRUEAND*bError* = FALSE sind.

iNumberOfStepInProfile: Zeigt die Anzahl der parametrierten Stufen aus dem vorgegebenen Profil *iCurrentProfile* der Leistungsstufentabelle *arrPowerRangeTable* an. Jede einzelne Stufe beinhaltet die für die Anlagensteuerung benötigte Parameterstruktur ST_HVACPowerRange [▶ 537]. Jedes Profil kann bis zu 33 Stufen haben. Das Feld für die Stufen in der Leistungsstufentabelle *arrPowerRangeTable* fängt bei 0 an und endet bei 32. Die Anzahl der parametrieten Stufen in einem Profil *iNumberOfStepInProfile* wird von der Stufe 1 aufwärts bis zur Stufe 32 bestimmt. Die Zählung der parametrieten Stufen endet in der Stufe in der alle Variablen der Parameterstruktur ST_HVACPowerRange [▶ 537] den Wert 0 haben. Minimal kann *iNumberOfStepInProfile* den Wert 1 haben, maximal 32. Die Stufe 0 wird nicht berücksichtigt, weil in dieser Stufe die Leistungserzeuger ausgeschaltet werden. Die Stufe kann als Bereitschaftsstufe gesehen werden. Die Werte der Variablen der Ausgangsstrukturen *stAggregate1-6* geben den Wert 0 aus. In der Bereitschaftsstufe werden dem vorgelagerten Regler FB_HVACI_Ctrl_Step [▶ 255] über die Ausgangsstruktur *stI_Ctrl* die Parameter übergeben, die für das Starten der Leistungsstufensequenz der Energieerzeuger verantwortlich sind.

> iNumberOfStepInProfile kann als Begrenzung der Stufen für andere Funktionsbausteine verwendet werden wie z.B. für den Eingang iNumberOfStepInProfile des Funktionsbausteines FB_HVACI_CtrlStep. Über die Ausgangsvariable iStep des FB_HVACI_CtrlStep kann wiederum die Stufenvorgabe der Leistungsstufentabelle FB_HVACPowerRangeTable über den Eingang iStep vorgegeben werden.

iCurrentProfile: Angabe des aktuellen Profils mit welchem der Funktionsbaustein aus der Leistungsstufentabelle *arrPowerRangeTable[***iCurrentProfile***,iCurrentStep]* arbeitet. Die Angabe ist abhängig von den Eingangsvariablen *iProfile*, *iManualProfile* und *eCtrlModeProfile*. Ist *eCtrlModeProfile* = *eHVACCtrlMode_Auto*, dann ist *iCurrentProfile* = *iProfile*. Ist *eCtrlModeProfile* = *eHVACCtrlMode_Manual*, dann ist *iCurrentProfile* = *iManualProfile*.

iCurrentStep: Angabe der aktuellen Leistungsstufe mit welcher der Funktionsbaustein aus der Leistungsstufentabelle *arrPowerRangeTable[iCurrentProfile,***iCurrentStep***]* arbeitet. Die aktuellste Stufe ist immer höchste Stufe von 1 an aufwärts gesehen. Die Angabe ist abhängig von den Eingangsvariablen *iStep*, *iManualStep* und *eCtrlModeStep*. Ist *eCtrlModeStep* = *eHVACCtrlMode_Auto*, dann ist *iCurrentStep* = *iStep*. Ist *eCtrlModeStep* = *eHVACCtrlMode_Manual*, dann ist *iCurrentStep* = *iManualStep*.

stI_Ctrl: Ausgabestruktur der Parameter ST_HVACI_Ctrl [▶ 535] für den Funktionsbaustein FB_HVACI_CtrlStep [▶ 255].
Welche Werte die Variablen der Ausgangsstruktur *stI_Ctrl* aus der Leistungsstufentabelle *arrPowerRangeTable* annehmen, ist abhängig vom ausgewählten Profil und der aktuellen Stufe. Ist das ausgewählte Profil **= 2***iCurrentProfile* und die Stufe **= 3***iCurrentStep*, dann wird die Struktur *stI_Ctrl* mit den Inhalten der folgenden Variablen aus der Leistungsstufentabelle beschrieben:

stI_Ctrl.rIntegralHigh := *arrPowerRangeTable[,].rIntegralHigh***iCurrentProfileiCurrentStep**;
stI_Ctrl.rIntegralLow := *arrPowerRangeTable[,].rIntegralLow***iCurrentProfileiCurrentStep**;
stI_Ctrl.udiSecDelayHigh :=*arrPowerRangeTable[,].udiSecDelayHigh***iCurrentProfileiCurrentStep**;
stI_Ctrl.udiSecDelayLow :=*arrPowerRangeTable[,].udiSecDelayLow***iCurrentProfileiCurrentStep**;

stI_Ctrl.rIntegralHigh := *arrPowerRangeTable[,].rIntegralHigh***23**;
stI_Ctrl.rIntegralLow := *arrPowerRangeTable[,].rIntegralLow***23**;
stI_Ctrl.udiSecDelayHigh :=*arrPowerRangeTable[,].udiSecDelayHigh***23**;
stI_Ctrl.udiSecDelayLow :=*arrPowerRangeTable[,].udiSecDelayLow***23**;

stI_Ctrl.rIntegralHigh: Positiver Wert für das obere Limit an dem die Integration des I-Übertragungsgliedes angehalten wird.

stI_Ctrl.rIntegralLow: Positiver Wert für das untere Limit an dem die Integration des I-Übertragungsgliedes angehalten wird.

stI_Ctrl.udiSecDelayHigh: Verzögerungszeit nach deren Ablauf das I-Übertragungsglied aktiviert wird.

stI_Ctrl.udiSecDelayLow: Verzögerungszeit nach deren Ablauf das I-Übertragungsglied aktiviert wird.

stAggregate1-6: Ausgabestrukturen der Parameter ST_HVACAggregate [▶ 534] für das Ansteuern der Aggregate 1 bis 6. *iNumberOfAggregates* bestimmt die Anzahl der Aggregate in der Leistungserzeugersequenz. Welche Werte die Variablen der Ausgangsstrukturen *stAggregate1-6* aus der Leistungsstufentabelle *arrPowerRangeTable* annehmen, ist abhängig vom ausgewählten Profil und der aktuellen Stufe. Ist das ausgewählte Profil **= 8***iCurrentProfile* und die Stufe **= 5***iCurrentStep*, dann ist entscheidend, dass die Variable*arrPowerRangeTable[,].iAggregate***85** den Wert 1 hat. Die in der Leistungsstufentabelle parametrierten Werte werden dann über die Ausgangsstruktur *stAggregate1* ausgegeben.

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

stAggregate1.rY_Max := *arrPowerRangeTable[,].rY_Max***85**;
stAggregate1.rY_Min :=*arrPowerRangeTable[,].rY_Min***85**;
stAggregate1.iAggregateStep :=*arrPowerRangeTable[,].iAggregateStep***83**;
stAggregate1.bBlock := *arrPowerRangeTable[,].bBlock***85**;

Wenn der Wert der Variable *arrPowerRangeTable[,].iAggregate***83** ebenfalls 1 ist, so werden in die dementsprechende Ausgangsstruktur *stAggregate1* die Werte der höchsten Leistungsstufe *iCurrentStep* = 5 geschrieben, weil auch hier *arrPowerRangeTable[,].iAggregate***85** = 1 ist. Erst wenn *iCurrentStep* = 3 ist, bedeutet das über die Ausgangsstruktur *stAggregate1* folgende Werte ausgegeben werden:

stAggregate1.rY_Max := *arrPowerRangeTable[,].rY_Max***83**;
stAggregate1.rY_Min :=*arrPowerRangeTable[,].rY_Min***83**;
stAggregate1.iAggregateStep :=*arrPowerRangeTable[,].iAggregateStep***83**;
stAggregate1.bBlock := *arrPowerRangeTable[,].bBlock***83**;

Ist *iCurrentStep* = 4 und*arrPowerRangeTable[,].iAggregate***84** = 2, dann bedeutet dies, dass über die Ausgangsstruktur *stAggregate1* und *stAggregate2* folgende Werte ausgegeben werden:

stAggregate2.rY_Max := *arrPowerRangeTable[,].rY_Max***84**;
stAggregate2.rY_Min :=*arrPowerRangeTable[,].rY_Min***84**;
stAggregate2.iAggregateStep :=*arrPowerRangeTable[,].iAggregateStep***84**;
stAggregate2.bBlock := *arrPowerRangeTable[,].bBlock***84**;

stAggregate1.rY_Max := *arrPowerRangeTable[,].rY_Max***83**;
stAggregate1.rY_Min :=*arrPowerRangeTable[,].rY_Min***83**;
stAggregate1.iAggregateStep :=*arrPowerRangeTable[,].iAggregateStep***83**;
stAggregate1.bBlock := *arrPowerRangeTable[,].bBlock***83**;

bError: Der Ausgang signalisiert mit einem TRUE, dass ein Fehler anliegt. Die Abarbeitung des Funktionsbausteins wird gestoppt. Das Enum *eErrorCode* zeigt die Fehlernummer an.
Nach Behebung des Fehlers muss die Meldung *bError* mit *bReset* quittiert werden.

eErrorCode: Liefert bei einem gesetzten *bError*-Ausgang die Fehlernummer [▶ 529].

Folgende Fehler können in diesem Funktionsbaustein vorkommen:

eHVACErrorCodes_InvalidParam_iStep: Fehler bei der Überprüfung der angegebenen Stufen. Der Wert von *iStep* muss größer gleich g_iMinNumberOfSteps [▶ 541] oder kleiner gleich *iNumberOfStepInProfile* sein.

eHVACErrorCodes_InvalidParam_iNumberOfStepInProfile: Fehler bei der Überprüfung der Anzahl der Profile, ob jedes der angegebenen Profile parametrierte Stufen hat. Die Überprüfung der Profile wird in Stufe 0 durchgeführt. Ist dort kein Variablenwert der Struktur ST_HVACPowerRange [▶ 537] größer 0, so liegt ein Fehler vor.
Mit der Variablen *iErrorPosArrayProfile* wird das fehlerhafte Profil in der Leistungsstufentabelle *arrPowerRangeTable***[x1**,**x2]** angegeben: **x1 =** *iErrorPosArrayProfile;*

eHVACErrorCodes_InvalidParam_iNumberOfAggregates: Fehler bei der Überprüfung der angegebenen Anzahl der Aggregate. Der Wert von *iNumberOfAggregates* muss kleiner als g_iMinNumberOfAggregates [▶ 541] und größer als g_iMaxNumberOfAggregates [▶ 541] sein.

eHVACErrorCodes_InvalidParam_iAggregateSteps: Fehler bei der Überprüfung der angegebenen Stufen für das Aggregate. Der Wert von *arrPowerRangeTable*[**x1**,**x2**].*iAggregateStep* muss größer gleich g_iAggregateMinNumberOfSteps [▶ 541] und kleiner gleich g_iAggregateMaxNumberOfSteps [▶ 541] sein.
Mit den Variablen *iErrorPosArrayProfile* und *iErrorPosArrayStep* wird die fehlerhafte Stelle in der Leistungsstufentabelle *arrPowerRangeTable***[x1**,**x2]** angegeben: **x1 = x2 =** *iErrorPosArrayProfile;iErrorPosArrayStep*

eHVACErrorCodes_InvalidParam_iNumberOfProfiles: Fehler bei der Überprüfung der Anzahl der Profile. Der Wert von *iNumberOfProfiles* muss größer gleich g_iMinNumberOfProfiles [▶ 541] und kleiner gleich g_iMaxNumberOfProfiles [▶ 541] sein.

eHVACErrorCodes_InvalidParam_iProfile: Fehler bei der Überprüfung der angegebenen Profile. Der Wert von *iProfile* muss größer gleich g_iMinNumberOfProfiles [▶ 541] und kleiner gleich *iNumberOfProfiles* sein.

eHVACErrorCodes_InvalidParam_iAggregate: Fehler bei der Überprüfung der angegebenen Aggregate. Der Wert von *arrPowerRangeTable*[**x**1,**x2**].*iAggregate* muss größer gleich g_iMinNumberOfAggregates [▶ 541] und kleiner gleich g_iMaxNumberOfAggregates [▶ 541] sein.
Mit den Variablen *iErrorPosArrayProfile* und *iErrorPosArrayStep* wird die fehlerhafte Stelle in der Leistungsstufentabelle *arrPowerRangeTable***[x1**,**x2]** angegeben: **x1 = x2 =** *iErrorPosArrayProfile;iErrorPosArrayStep*

eHVACErrorCodes_InvalidParam_udiSecDelayUp: Fehler bei der Überprüfung von *arrPowerRangeTable*[**x1**,**x2**].*udiSecDelayUp*. Der Wert von *arrPowerRangeTable*[**x**1,**x2**].*udiSecDelayUp* darf nicht größer als g_udiMaxSec [▶ 541] sein.
Mit den Variablen *iErrorPosArrayProfile* und *iErrorPosArrayStep* wird die fehlerhafte Stelle in der Leistungsstufentabelle *arrPowerRangeTable***[x1**,**x2]** angegeben: **x1 = x2 =** *iErrorPosArrayProfile;iErrorPosArrayStep*

eHVACErrorCodes_InvalidParam_udiSecDelayDown: Fehler bei der Überprüfung von *arrPowerRangeTable*[**x1**,**x2**].*udiSecDelayDown*. Der Wert von *arrPowerRangeTable*[**x**1,**x2**].*udiSecDelayDown* darf nicht größer als g_udiMaxSec [▶ 541] sein.
Mit den Variablen *iErrorPosArrayProfile* und *iErrorPosArrayStep* wird die fehlerhafte Stelle in der Leistungsstufentabelle *arrPowerRangeTable***[x1**,**x2]** angegeben: **x1 = x2 =** *iErrorPosArrayProfile;iErrorPosArrayStep*

Um in der SPS an die Fehlernummern des Enums zu gelangen, kann eErrorCode einer Variablen vom Datentyp WORD zugewiesen werden.eHVACErrorCodes_InvalidParam_iStep = 31 eHVACErrorCodes_InvalidParam_iNumberOfStepInProfil = 32eHVACErrorCodes_InvalidParam_iNumberOfAggregates = 33eHVACErrorCodes_InvalidParam_iAggregateSteps = 34eHVACErrorCodes_InvalidParam_iNumberOfProfiles = 35eHVACErrorCodes_InvalidParam_iProfile = 36eHVACErrorCodes_InvalidParam_iAggregate = 39 eHVACErrorCodes_InvalidParam_udiSecDelayUp = 40 eHVACErrorCodes_InvalidParam_udiSecDelayDown = 41

iErrorPosArrayProfile: Mit der Variablen *iErrorPosArrayProfile* wird das Profil angegeben, in der die fehlerhafte Stelle in der Leistungsstufentabelle *arrPowerRangeTable*[*iErrorPosArrayProfile*,*iErrorPosArrayStep*] zu finden ist.
Ist *iErrorPosArrayProfile* > 0, dann liegt einer der folgenden Fehler an *eErrorCode* an:
- *eHVACErrorCodes_InvalidParam_iNumberOfStepInProfile*
- *eHVACErrorCodes_InvalidParam_iAggregateSteps*
- *eHVACErrorCodes_InvalidParam_iAggregate*
- *eHVACErrorCodes_InvalidParam_udiSecDelayUp*
- *eHVACErrorCodes_InvalidParam_udiSecDelayDown*

Mit Hilfe der Variablen *iErrorPosArrayStep* kann die genaue Position der Stufe ermittelt werden.

Beispiel 1 zur Lokalisierung eines Fehlers:
Folgende Werte stehen an den Ausgangsvariablen an:

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

In der Leistungsstufentabelle *arrPowerRangeTable***[2,x]**ist in dem Profil 2in Stufe 0 kein Variablenwert der Struktur ST_HVACPowerRange [▶ 537] größer 0. Bei diesem Fehler wird *iErrorPosArrayStep* nicht berücksichtigt.

Beispiel 2 zur Lokalisierung eines Fehlers:
Folgende Werte stehen an den Ausgangsvariablen an:

bError =TRUE*;*
eErrorCode= eHVACErrorCodes_InvalidParam_iAggregateSteps;
iErrorPosArrayProfile = **2**;
iErrorPosArrayStep = **3**;

In der Leistungsstufentabelle *arrPowerRangeTable* steht an der folgenden Variable ein falscher Wert an:

arrPowerRangeTable[**2,3**].*iAggregateStep*

iErrorPosArrayStep: Ist *iErrorPosArrayProfile* > 0, dann wird mit der Variablen *iErrorPosArrayStep* die Stufe im Profil angegeben, in der die fehlerhafte Stelle in der Leistungsstufentabelle zu finden ist.

Beispiel zur Lokalisierung eines Fehlers:
Folgende Werte stehen an den Ausgangsvariablen an:

bError =TRUE*;*
eErrorCode= eHVACErrorCodes_InvalidParam_iAggregate;
iErrorPosArrayProfile = **5**;
iErrorPosArrayStep = **15**;

In der Leistungsstufentabelle *arrPowerRangeTable* steht an der folgenden Variable ein falscher Wert an:

arrPowerRangeTable[**5,15**].*iAggregate*

VAR_IN_OUT

```
arrPowerRangeTable    : ARRAY [1..g_iMaxNumberOfProfiles,0..g_iMaxNumberOfSteps] OF ST_HVACPowerRang
e;         X
```

arrPowerRangeTable: Alle für die Anlagensteuerung relevanten Informationen bzw. Parameter sind in der Leistungsstufentabelle *arrPowerRangeTable* zusammengefasst. Die Leistungsstufentabelle ist ein zweidimensionales Feld (Array) der Struktur ST_HVACPowerRange [▶ 537]. Die Leistungsstufentabelle kann mit einer normalen Tabelle verglichen werden, in der die waagerechten Einträge als Zeilen, die Senkrechten als Spalten bezeichnet werden. Ein einzelnes Element ist also durch Nennung von Zeile und Spalte eindeutig bezeichnet. Der Feldbereich 1..g_iMaxNumberOfProfiles [▶ 541] der Leistungsstufentabelle wäre der senkrechte Teil, also die Spalten. Dieser Bereich wird als Profil bezeichnet und anhand von *iCurrentProfile* wird angegeben, welches Profil angesprochen wird. Der Feldbereich 0..g_iMaxNumberOfSteps [▶ 541] wird als der waagerechte Teil, also die Zeilen, angesehen. Dieser Bereich wird als Stufe bezeichnet und anhand von *iCurrentStep* wird angegeben, welche Stufe angesprochen wird.

Die Leistungsstufentabelle besteht aus 16 Profilen. Jedes Profil kann bis zu 33 Stufen besitzen. Jede einzelne Stufe beinhaltet die für die Anlagensteuerung benötigte Parameterstruktur ST_HVACPowerRange [▶ 537]. Die hier angegebenen Parameter werden über die Ausgangsstrukturen *stI_Ctrl* dem Funktionsbaustein FB_HVACI_CtrlStep [▶ 255] zur Steuerung der Stufen und über *stAggregate1-6* den Funktionsbausteinen zur Energieerzeugung übergeben.

Wieviele Stufen sich in einem Profil befinden, wird über die Ausgangsvariable *iNumberOfStepInProfile* angegeben. Die Anzahl ist abhängig von den Einträgen in der Leistungsstufentabelle und vom ausgewählten Profil *iCurrentProfile.* Es wird intern im Baustein das ausgewählte Profil von der Stufe 1 bis zu der Stufe überprüft, in der alle Variablen der Struktur ST_HVACPowerRange [▶ 537] den Wert 0 haben. *iNumberOfStepInProfile* wird immer nur für das vorgegebene Profil bestimmt. Jedes Profil kann bis zu 33 Stufen haben von 0 bis 32, die Stufe 0 wird bei der Auswertung der Anzahl der Stufen über *iNumberOfStepInProfile* nicht berücksichtigt.

Struktur ST_HVACPowerRange [▶ 537]

arrPowerRangeTable[x,x].iAggregate: Parameterangabe in welche Ausgangsstruktur *stAggregate1-6* des Funktionsbausteins FB_HVACPowerRangeTable [▶ 281] die Variablen *rY_Min*, *rY_Max*, *iAggregateStep* und *bBlock* geschrieben werden.

arrPowerRangeTable[x,x].iAggregateStep:Parameterangabe in welcher Stufe das angesprochene Aggregat fixiert oder regeln soll, siehe *bBlock*.
iAggregateStep wird über die Struktur *stAggregateX* ausgegeben.

arrPowerRangeTable[x,x].rY_Max: Parameterangabe für stetige Aggregate. *rY_Max* wird über die Struktur *stAggregateX* ausgegeben.

arrPowerRangeTable[x,x].rY_Min: Parameterangabe für stetige Aggregate. *rY_Min* wird über die Struktur *stAggregateX* ausgegeben.

arrPowerRangeTable[x,x].rIntegralHigh: Positiver Wert für das obere Limit an dem die Integration des I-Übertragungsgliedes angehalten wird, siehe der VAR_IN_OUT-Variable *rIntegralHigh* im FB_HVACI_CtrlStep [▶ 255].
rIntegralHigh wird über die Struktur *stI_Ctrl* ausgegeben.

arrPowerRangeTable[x,x].rIntegralLow **:** Positiver Wert für das untere Limit an dem die Integration des I-Übertragungsgliedes angehalten wird, siehe der VAR_IN_OUT-Variable *rIntegralHigh* im FB_HVACI_CtrlStep [▶ 255].
rIntegralLow wird über die Struktur *stI_Ctrl* ausgegeben.

arrPowerRangeTable[x,x].udiSecDelayHigh: Verzögerungszeit nach deren Ablauf das I-Übertragungsglied aktiviert wird, siehe der VAR_IN_OUT-Variable *udiSecDelayHigh* im FB_HVACI_CtrlStep [▶ 255].
udiSecDelayHigh wird über die Struktur *stI_Ctrl* ausgegeben.

arrPowerRangeTable[x,x].udiSecDelayLow: Verzögerungszeit nach deren Ablauf das I-Übertragungsglied aktiviert wird, siehe der VAR_IN_OUT-Variable *udiSecDelayHigh* im FB_HVACI_CtrlStep [▶ 255].
udiSecDelayLow wird über die Struktur *stI_Ctrl* ausgegeben.

arrPowerRangeTable[x,x].bBlock: Ist *bBlock* = FALSE, so wird das angesprochene Aggregat in der vorgegebenen Stufe über *iAggregateStep* fixiert. Ist *bBlock* = TRUE, so wird die Regelung des angesprochenen Aggregats frei gegeben von der Ausstufe (0) bis zur vorgegebenen Stufe über iAggregateStep.
bBlock wird über die Struktur *stAggregateX* ausgegeben.

Die Variable wird persistent gespeichert.

Dokumente hierzu

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

3.4.7 Sequenzregler

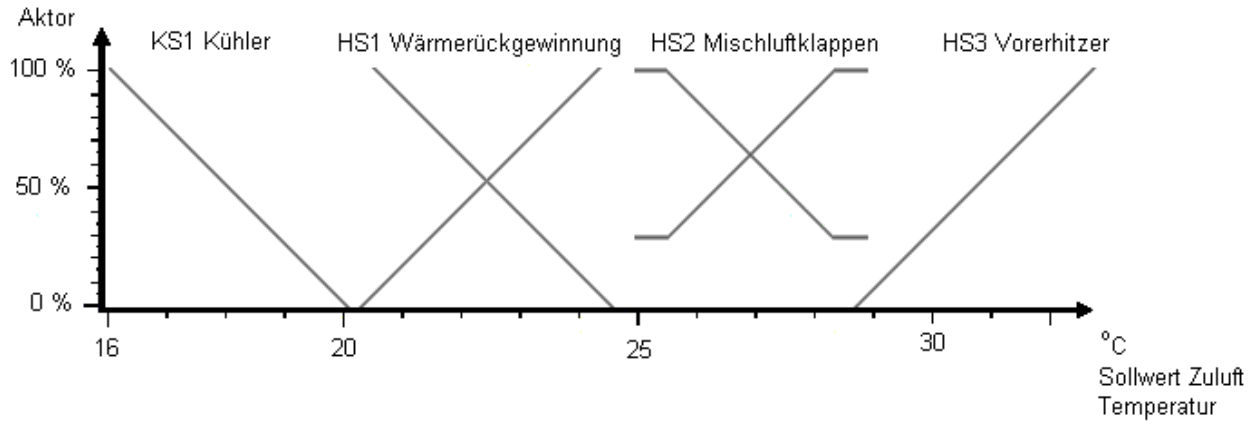
Einleitung Sequenzregler

In der Heizungs-, Lüftungs- und Klimatechnik kommt es häufig vor, dass zum Erreichen einer Regelgröße mehrere Stellglieder verwendet werden, die in einer so genannten Reglersequenz arbeiten. In der unten dargestellten Klimaanlage sind an der Zulufttemperaturregelung vier Stellorgane beteilig. In der HLK-Bibliothek wird für jedes dieser Stellorgane ein eigener Sequenzregler instanziert. Bei aktiver Regelung ist immer nur einer dieser Sequenzregler aktiv. Die anderen nicht aktiven Regler fixieren Ihr Stellsignal so wie es energetisch für die Temperierung der Zulufttemperatur optimal ist. Das bedeutet in Abhängigkeit des Wirksinns des einzelnen Reglers entweder das Maximum oder das Minimum für die Stellgröße *rY.*

Wenn die Wirkung des aktiven Stellgliedes (Reglers) bei dem Erreichen einer Endlage nicht ausreicht, schaltet der aktive Regler auf den in der Reglersequenz rechts oder links benachbarten Regler um. Dieser übernimmt damit die Regelung. Der zuvor aktive Regler verharrt je nach Wirksinn in der Endlage von *rYmax* oder *rYmin.*
Ebenso wird mit den weiteren Stellgliedern verfahren, bis der Sollwert oder das rechte oder linke Ende der Sequenz erreicht ist.

In der Sequenz der dargestellten Raumlufttechnischen Anlage sind alle Stellglieder, die die Regelgröße beeinflussen , von links nach
rechts dargestellt. Ganz links steht das Stellglied, das die größtmögliche Verringerung der Regelgröße ermöglicht, ganz rechts das Stellglied, das die größtmögliche Erhöhung der Regelgröße bewirkt.

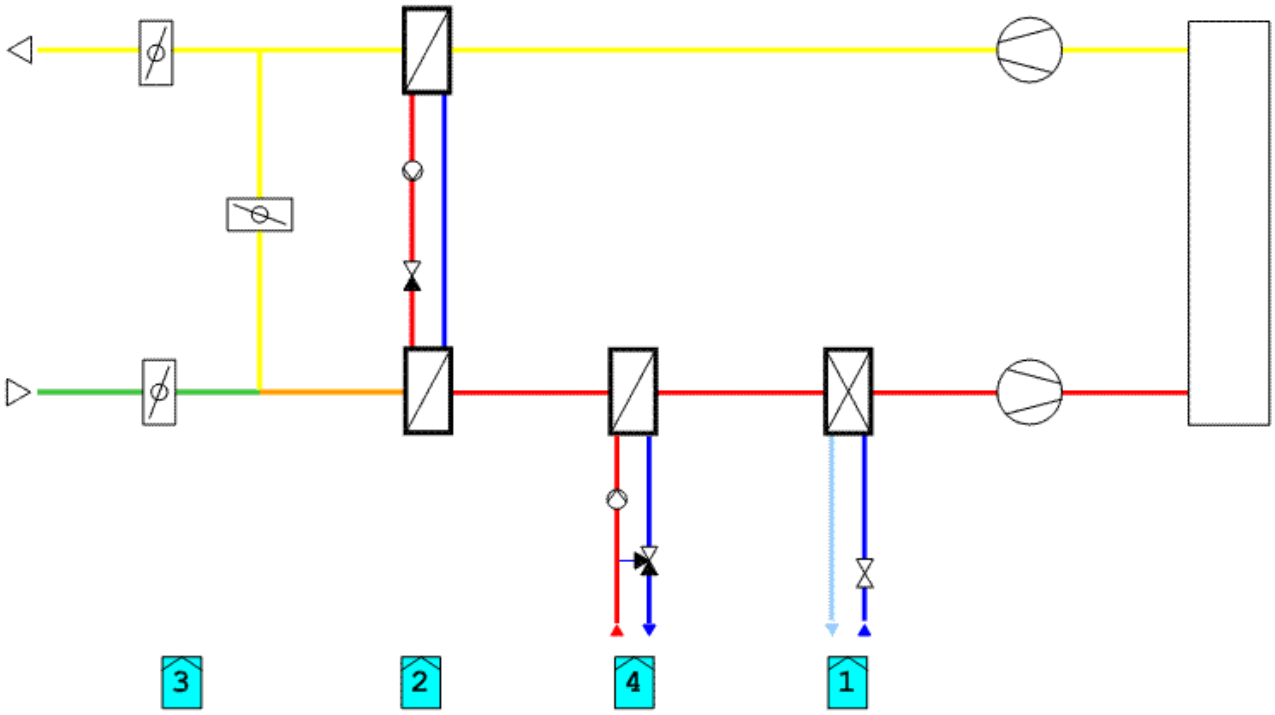
Stellglieder, die parallel oder gegensinnig fahren sollen, wie z.B. Außenluft- und Umluftklappe (HS2), werden nur einmal aufgeführt. Die einzelnen Stellglieder können positive Wirkrichtung (mehr Stellung ergibt mehr Regelgröße z.B. Erhitzer) oder negative Wirkrichtung (mehr Stellung ergibt weniger Regelgröße z.B. Kühler) haben. Manche Stellglieder, wie z.B. eine Umluftklappe (HS1 u. HS2), wechseln während des Betriebs ihre Wirkrichtung.



Beispiel: Zulufttemperaturregelung in einer Klimaanlage mit einem Luftkühler, einer Wärmerückgewinnung, einer Mischluftkammer und einem Lufterhitzer wie in der obigen Zeichnung.

```
iMyNumberInSequence
```

1. KS1 Kühler 1
2. HS1 Wärmerückgewinnung 2
3. HS2 Mischluftklappen 3
4. HS3 Vorerhitzer 4



Jeder Sequenzregler bekommt die Nummer der insgesamt in der Sequenz befindlichen Regler.
In diesem Beispiel 4. Die SPS-Variable heißt in den Funktionsbausteinen *iNumberOfSequences*. Zusätzlich bekommt jeder Funktionsbaustein seine eigene Nummer. In der Sequenz *iMyNumberInSequence*. Die Nummerierung der Regler erfolgt von links nach rechts aufsteigend. So ist jedem Regler bekannt, an welcher Stelle er sich innerhalb der Sequenz befindet. Den sequenzäußeren Reglern ist so bekannt, dass bei einer bleibenden Regelabweichung nicht auf einen weiteren Sequenzregler zu schalten ist.

Im Beispiel würde der Regler FB_HVACPIDEnergyRecovery [▶ 316] bei zu niedriger Regelgröße mit der WRG anfangen zu regeln. Alle anderen Stellglieder sind geschlossen. Wenn die maximale oder minimale Stellung der WRG erreicht ist, schaltet der Regler auf die Klappenregelung FB_HVACPIDMixedAir [▶ 323] um. Damit wird der Parameter*eHVACSequenceCtrlMode* auf 3 erhöht. Wenn die Stellwirkung dann noch nicht erreicht ist und die Mischluftklappe an Ihrem Limit (z.B. minimale Außenluftrate) angekommen ist, erhöht der Mischluftregler den Wert von *eHVACSequenceCtrlMode* auf 4. Damit wird der Vorerhitzer bzw. der Sequenzregler FB_HVACPIDPreHeating [▶ 326] aktiviert. Bei zu hoher Regelgröße geschieht dieses in umgekehrter Reihenfolge bis der Kühler mit der Freischaltung des Reglers FB_HVACPIDCooling [▶ 310] aktiviert wird.

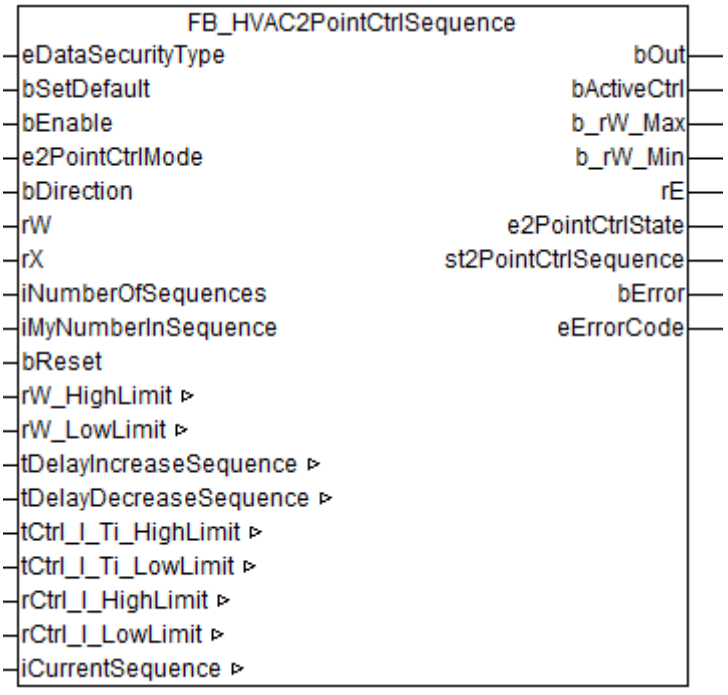
Falls die Stellgröße am unteren oder oberen Limit eines Reglers ist, der Istwert der Regelstrecke mit kleiner Amplitude um den Sollwert pendelt, kann ein häufiges Hin- und Herschalten zwischen zwei Sequenzreglern mit einem zusätzlichen Parameter für die Umschaltung gedämpft werden. Dazu wird nachdem der Sequenzregler sein unteres oder oberes Limit erreicht hat, die Abweichung zwischen dem Istwert und dem Sollwert der Regelstrecke aufintegriert. Eine Umschaltung auf die nächste Sequenz erfolgt erst, wenn der Betrag dieser Integration größer ist, als der Wert von *rDeadRange*.

Tabelle Betriebsarten

Eine weitere Besonderheit der Sequenzregler ist Ihre Steuerung mit dem Enum E_HVACSequenceCtrlMode [▶ 531]
Mittels des Enums E_HVACSequenceCtrlMode [▶ 531]erfolgt nicht nur die Freigabe der Regelung, sondern auch die Übertragung der Betriebsart der Raumlufttechnischen Anlage an die Regelungsbausteine in der Sequenz. Damit reagiert jeder Sequenzregler in Abhängigkeit der Betriebsarten speziell auf die Werte des Enums E_HVACSequenceCtrlMode [▶ 531] wie in der Tabelle dargestellt.

Wert von: E_HVACSequenceCtrlMode	0 (Stop)	1 (On)	2 (Nachtkühlung)	3 (Stützbetrieb)	4 (Überhitzungsschutz)	5 (Nachtkühlung und Überhitzungsschutz)
FB_HVACMasterSequenceCtrl Führungsregler	gesperrt 0%	Freigabe	gesperrt 0%	Freigabe rY=ZuluftMax.Temp	Freigabe rY=ZuluftMin.Temp	Freigabe rY=ZuluftMin.Temp
FB_HVACPIDPreHeating Vorerhitzer	gesperrt 0%	Freigabe	gesperrt 0%	Freigabe 0%-100%	gesperrt 0%	gesperrt 0%
FB_HVACPIDReHeating Nacherhitzer	gesperrt 0%	Freigabe	gesperrt 0%	gesperrt 0%	gesperrt 0%	gesperrt 0%
FB_HVACPIDCooling Kühler	gesperrt 0%	Freigabe	gesperrt 0%	gesperrt 0%	Freigabe 0%-100%	Freigabe 0%-100%
FB_HVACPIDEnergyRecovery Wärmerückgewinnung	gesperrt 0%	Freigabe	gesperrt 0%	gesperrt 0%	gesperrt 0%	gesperrt 0%
FB_HVACPIDMixedAir Mischluftklappen	gesperrt 0%	Freigabe	max. Außenluftrate 100%	0% reine Umluft	0% reine Umluft	max. Außenluftrate 100%

3.4.7.1 FB_HVAC2PointCtrlSequence



Anwendung

Dieser Funktionsbaustein stellt einen 2-Punkt-Sequenzregler dar. Er dient zur sequenziellen stufigen Regelung von unstetigen Aggregaten. Er kann z.B. bei Kesselkaskaden und bei Sequenzen von Kältemaschine oder Rückkühlwerken zur Regelung der Leistungsstufen verwendet werden. Für jede Leistungsstufe wird eine Instanz des Funktionsbausteins verwendet.

Der FB_HVAC2PointCtrlSequence kann z.B. in einer Reglersequenz mit den anderen Sequenzreglern aus der TcHVAC.lib eingesetzt werden.

Je nach Anwendung kann es vorkommen, dass zum Erreichen einer Regelgröße mehrere Stellglieder (Stufen) verwendet werden, die in einer so genannten Regelsequenz arbeiten. In der unten dargestellten Regelsequenz werden vier Stufen über den jeweiligen Ausgang *bOut* = TRUE des *FB_HVAC2PointCtrlSequence* freigegeben. Bei aktiver Regelung ist immer nur einer der 2Punkt-Sequenzregler aktiv. Die Ausgänge der nicht aktiven 2-Punkt Sequenzregler werden fixiert. Das bedeutet in Abhängigkeit des Wirksinns *bDirection* des einzelnen Reglers entweder das *bOut* = TRUE oder FALSE ist.

Wenn die Wirkung der aktiven Stufe im eingeschalteten Zustand nicht ausreicht, schaltet der aktive 2Punktregler auf den in der Reglersequenz rechts oder links benachbarten 2Punkt-Sequenzregler über *iCurrentSequence* um. Dieser übernimmt damit die Regelung/Steuerung der Stufen. Der zuvor aktive Regler verharrt je nach Wirksinn in der Endlage *bOut* = TRUE oder FALSE.

Übertragungsfunktion des internen I-Übertragungsglied

Übertragungsfunktion des internen I-Übertragungsglied

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

Mit der internen AND-Verknüpfung des I-Übertragungsglieds bestehend aus (*tCtrl_I_Ti_HighLimit, tCtrl_I_Ti_LowLimit*, *rCtrl_I_HighLimit, rCtrl_I_LowLimit, st2PointCtrlSequence.rCtrl_I_Out*) und den Verzögerungszeiten *tDelayIncreaseSequence/tDelayDecreaseSequence* wird das Umschalten in der Regelsequenz über *iCurrentSequence* nach rechts oder links gesteuert. Welcher Sequenzregler gerade aktiv ist, wird durch *bActiveCtrl* = TRUE (*iCurrentSequence* = *iMyNumberInSequence*) angezeigt.

Das I-Übertragungsglied und die Zeitglieder der Verzögerungszeiten *tDelayIncreaseSequence/ tDelayDecreaseSequence* werden aktiviert, wenn

bActiveCtrl = TRUEAND
(**(*Decrease*)**
(
rX > *st2PointCtrlSequence.rW_Max*AND ((*iMyNumberInSequence* <= 1) = FALSE) AND
(
(*bDirection* = TRUEAND*bOut* = TRUE) OR
(*bDirection* = FALSEAND*bOut* = FALSE)
)
)
OR
(**(*Increase*)**
rX < *st2PointCtrlSequence.rW_Min*AND(*(iMyNumberInSequence* >= *iNumberOfSequences*) = FALSE) AND
(
(*bDirection* = FALSEAND*bOut* = TRUE) OR
(*bDirection* = TRUEAND*bOut* = FALSE)
)
)
)

Nachdem das I-Übertragungsglied (*st2PointCtrlSequence.rCtrl_I_Out* Ausgang des internen I-Übertragungsgliedes) UND die Zeitglieder der Verzögerungszeiten *tDelayIncreaseSequence/ tDelayDecreaseSequence* aktiviert wurden, wird das Umschalten in der Regelsequenz über *iCurrentSequence* nach rechts oder links folgendermaßen gesteuert:

iCurrentSequence = *iCurrentSequence* - 1 wenn *st2PointCtrlSequence.rCtrl_I_Out* <= *st2PointCtrlSequence.rCtrl_I_LowLimit*UND*st2PointCtrlSequence.tRemainingTimeDecreaseSequence* = T#0s.

iCurrentSequence = *iCurrentSequence* + 1 wenn *st2PointCtrlSequence.rCtrl_I_Out* >= *st2PointCtrlSequence.rCtrl_I_HighLimit*UND*st2PointCtrlSequence.tRemainingTimeIncreaseSequence* = T#0s.

Verhalten der Ausgänge von vier FB_HVAC2PointCtrlSequence in einer Regelsequenz

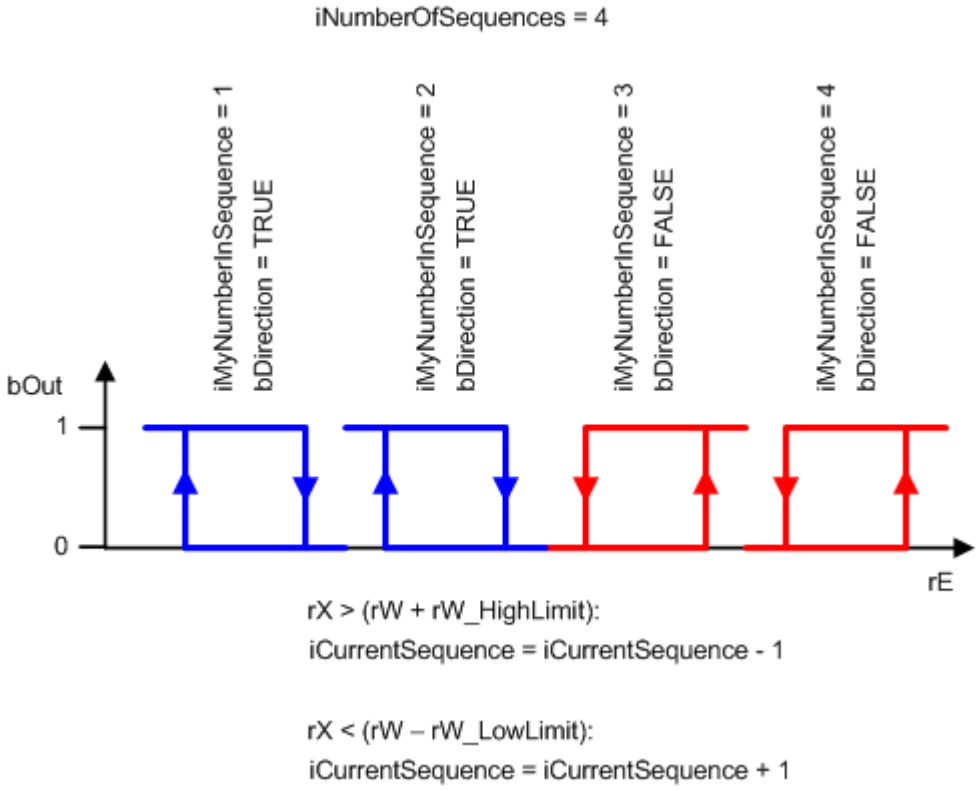


Abb. 16: FB_HVAC2PointCtrlSequence_1

In der oben dargestellten Regelsequenz von vier FB_HVAC2PointCtrlSequence-Reglern darf es bei der Vergabe von iMyNumberInSequence (1,2,3,4) und iNumberOfSequence (4) keine Lücke in der Folge geben, da ansonsten das automatische Umschalten der Regler vom aktiven auf den in der Regelsequenz rechts oder links benachbarten Regler nicht funktioniert.

Ist bEnable =FALSEODERbError = TRUEODER (e2PointCtrlMode =eHVAC2PointCtrlMode_On_BMSODEReHVAC2PointCtrlMode_On_OPODEReHVAC2PointCtrlMode_Off_BMSODEReHVAC2PointCtrlMode_Off_OP), so ist das automatische Umschalten der Regler vom aktiven auf den in der Regelsequenz rechts oder links benachbarten Regler weiterhin in Betrieb. Daher ist der Umschaltbetrieb immer aktiv. Mit bActiveCtrl = TRUE wird der aktive Funktionsbaustein angezeigt. Außerdem wird in Abhängigkeit der Regelabweichung rE das Rauf- oder Runterschalten der Sequenz über iCurrentSequence direkt ausgeführt. Wenn rX >= rW, dann ist iCurrentSequence = iCurrentSequence - 1. Wenn rX < rW, dann ist iCurrentSequence = iCurrentSequence + 1.

Anwendungsbeispiel

Download	Benötige Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

```
eDataSecurityType   : E_HVACDataSecurityType;
bSetDefault         : BOOL;
bEnable             : BOOL;
e2PointCtrlMode     : E_HVAC2PointCtrlMode;
bDirection          : BOOL;
rW                  : REAL;
rX                  : REAL;
iNumberOfSequences  : INT;              1..32
iMyNumberInSequence: INT;               1..32
bReset              : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen. Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Über ein TRUE wird der Funktionsbaustein freigegeben. Ist *bEnable* = FALSE, so ist der 2-Punkt-Sequenzregler deaktiviert. Die Überprüfung der Variablen *iNumberOfSequences*, *iMyNumberInSequence*, *iCurrentSequence* und *tTi_Ctrl_I* ist dennoch aktiv. Falls dort ein Fehler auftritt, so wird dieses mit *bError* = TRUE angezeigt und kann nach Behebung des Fehlers mit *bReset* quittiert werden.

Ist bEnable =FALSEODERbError = TRUEODER (e2PointCtrlMode =eHVAC2PointCtrlMode_On_BMSODEReHVAC2PointCtrlMode_On_OPODEReHVAC2PointCtrlMode_Off_BMSODEReHVAC2PointCtrlMode_Off_OP), so ist das automatische Umschalten der Regler vom aktiven auf den in der Regelsequenz rechts oder links benachbarten Regler weiterhin in Betrieb. Daher ist der Umschaltbetrieb immer aktiv. Mit bActiveCtrl = TRUE wird der aktive Funktionsbaustein angezeigt. Außerdem wird in Abhängigkeit der Regelabweichung rE das Rauf- oder Runterschalten der Sequenz über iCurrentSequence direkt ausgeführt. Wenn rX >= rW, dann ist iCurrentSequence = iCurrentSequence - 1. Wenn rX < rW, dann ist iCurrentSequence = iCurrentSequence + 1.

e2PointCtrlMode: Enum, welches die Betriebsart des 2-Punkt-Sequenzreglers vorgibt. Ist *bEnable* = TRUEAND*bError* = FALSE, so kann der Ausgang *bOut* direkt über das Enum ein- oder ausgeschaltet werden. Beim Starten der PLC ist *e2PointCtrlMode* = *eHVAC2PointCtrlMode_Auto_BMS*.
Das Rauf- oder Runterschalten der Sequenz über *iCurrentSequence* ist nicht abhängig von der Betriebsart *e2PointCtrlMode* des 2-Punkt-Sequenzreglers.

Ist bEnable =FALSEODERbError = TRUEODER (e2PointCtrlMode =eHVAC2PointCtrlMode_On_BMSODEReHVAC2PointCtrlMode_On_OPODEReHVAC2PointCtrlMode_Off_BMSODEReHVAC2PointCtrlMode_Off_OP), so ist das automatische Umschalten der Regler vom aktiven auf den in der Regelsequenz rechts oder links benachbarten Regler weiterhin in Betrieb. Daher ist der Umschaltbetrieb immer aktiv. Mit bActiveCtrl = TRUE wird der aktive Funktionsbaustein angezeigt. Außerdem wird in Abhängigkeit der Regelabweichung rE das Rauf- oder Runterschalten der Sequenz über iCurrentSequence direkt ausgeführt. Wenn rX >= rW, dann ist iCurrentSequence = iCurrentSequence - 1. Wenn rX < rW, dann ist iCurrentSequence = iCurrentSequence + 1.

bDirection: Mit *bDirection* wird der Wirksinn des internen 2 Punkt Reglers bestimmt. FALSE = Heizbetrieb; TRUE = Kühlbetrieb.

rW: Mit der Variablen *rW* wird der Sollwert übergeben.

rX: Mit der Variablen *rX* wird der Istwert übergeben.

iNumberOfSequences: Anzahl der Regler in der Sequenz. Ist *iNumberOfSequences* <= 0, so wird ein Fehler ausgegeben und mit *bError* = TRUE angezeigt. Die Anzahl der Teilnehmer in einer Sequenz wird von dem aktiven Sequenzregler über *iCurrentSequence* nicht überschritten.

iMyNumberInSequence: Die eigene Nummer des 2Punkt-Reglers in der Sequenz. Ist *iMyNumberInSequence* > *iNumberOfSequences*OR*iMyNumberInSequence* <= 0, so wird ein Fehler ausgegeben und mit *bError* = TRUE angezeigt.

bReset: Eingang zur Quittierung der Störungen nach deren Behebung. Intern wird auf eine steigende Flanke reagiert.

VAR_OUTPUT

```
bOut                : BOOL;
bActiveCtrl         : BOOL;
b_rW_Max            : BOOL;
b_rW_Min            : BOOL;
rE                  : REAL;
e2PointCtrlState    : E_HVAC2PointCtrlMode;
st2PointCtrlSequence: ST_HVAC2PointCtrlSequence;
bError              : BOOL;
eErrorCode          : E_HVACErrorCodes;
```

bOut: Ausgang des 2-Punkt-Sequenzreglers.

bOut wird TRUE, wenn

1.
bEnable = TRUEAND*bError* = FALSEAND
(*e2PointCtrlMode* = *eHVAC2PointCtrlMode_Auto_BMS*OR*e2PointCtrlMode* = *eHVAC2PointCtrlMode_Auto_OP*) AND
bActiveCtrl = TRUEAND
(
(bDirection = TRUEAND (rX > st2PointCtrlSequence.rW_Max)) OR (bDirection = FALSEAND (rX < st2PointCtrlSequence.rW_Min))
)

2.
bEnable = TRUEAND*bError* = FALSEAND
(*e2PointCtrlMode* = *eHVAC2PointCtrlMode_Auto_BMS*OR*e2PointCtrlMode* = *eHVAC2PointCtrlMode_Auto_OP*) AND
(
(*bDirection* = TRUEAND (*iCurrentSequence* < *iMyNumberInSequence*)) OR (*bDirection* = FALSEAND (*iCurrentSequence* > *iMyNumberInSequence*))
)

3.
bEnable = TRUEAND*bError* = FALSEAND
(*e2PointCtrlMode* = *eHVAC2PointCtrlMode_On_BMS*OR*e2PointCtrlMode* = *eHVAC2PointCtrlMode_On_OP*)

bOut wird FALSE, wenn

1.
bEnable = FALSEOR*bError* = TRUE

2.
bEnable = TRUEAND*bError* = FALSEAND
(*e2PointCtrlMode* = *eHVAC2PointCtrlMode_Auto_BMS*OR*e2PointCtrlMode* = *eHVAC2PointCtrlMode_Auto_OP*) AND
bActiveCtrl = TRUEAND
(
(*bDirection* = TRUEAND (*rX* < *st2PointCtrlSequence.rW_Min*)) OR (*bDirection* = FALSEAND (*rX* > *st2PointCtrlSequence.rW_Max*))
)

3.
bEnable = TRUEAND*bError* = FALSEAND
(*e2PointCtrlMode* = *eHVAC2PointCtrlMode_Auto_BMS*OR*e2PointCtrlMode* = *eHVAC2PointCtrlMode_Auto_OP*) AND
(
(*bDirection* = TRUEAND (*iCurrentSequence* > *iMyNumberInSequence*)) OR (*bDirection* = FALSEAND (*iCurrentSequence* < *iMyNumberInSequence*))
)

4.
bEnable = TRUEAND*bError* = FALSEAND
(*e2PointCtrlMode* = *eHVAC2PointCtrlMode_Off_BMS*OR*e2PointCtrlMode* = *eHVAC2PointCtrlMode_Off_OP*)

bActiveCtrl: *bActiveCtrl* zeigt mit einem TRUE an, dass der Funktionsbaustein der aktive in der Sequenz ist. *bActiveCtrl* wird TRUE, wenn *bEnable* = TRUE, *bError* = FALSEAND*iCurrentSequence* = *iMyNumberInSequence* ist.

b_rW_Max: *b_rW_Max* wird TRUE, wenn *rX* > *st2PointCtrlSequence.rW_Max* ist.

b_rW_Min: *b_rW_Min* wird TRUE, wenn *rX* < *st2PointCtrlSequence.rW_Min* ist.

rE: Regelabweichung: *rE* = *rW* - *rX*

e2PointCtrlState: Enum, welches den Status der Betriebsart *e2PointCtrlMode*des 2-Punkt-Sequenzreglers anzeigt

st2PointCtrlSequence: Die Struktur zeigt diverse Status, Ein- und Ausgänge des Funktionsbausteins an. Außerdem werden die verbleibenden Zeiten der VAR_IN_OUT-Variablen *tDelayIncreaseSequence* und *tDelayDecreaseSequence* ausgegeben, wenn deren Funktion aktiv ist. Über *st2PointCtrlSequence.rCtrl_I_Out* wird das Ausgangssignal des internes I-Übertragungsglied angezeigt.

st2PointCtrlSequence.tRemainingTimeIncreaseSequence: Verbleibende Zeit der Verzögerungszeit *tDelayIncreaseSequence.*

st2PointCtrlSequence.tRemainingTimeDecreaseSequence: Verbleibende Zeit der Verzögerungszeit *tDelayDecreaseSequence.*

st2PointCtrlSequence.rX: Status von *rX*

st2PointCtrlSequence.rW_Max: *st2PointCtrlSequence.rW_Max* := *rW* + *rW_HighLimit* - obere Sollwertgrenze nach dessen Überschreitung durch *rX* das interne I-Übertragungsglied und die Zeitglieder der Verzögerungszeiten *tDelayIncreaseSequence/tDelayDecreaseSequence* aktiviert bzw. deaktiviert werden können, siehe Übertragungsfunktion des internen I-Übertragungsglied [▶ 297] in diesem Dokument

st2PointCtrlSequence.rW_Min: *st2PointCtrlSequence.rW_Min* := *rW* - *rW_LowLimit* - untere Sollwertgrenze nach dessen Unterschreitung durch *rX* das interne I-Übertragungsglied und die Zeitglieder der Verzögerungszeiten *tDelayIncreaseSequence/tDelayDecreaseSequence* aktiviert bzw. deaktiviert werden können, siehe Übertragungsfunktion des internen I-Übertragungsglied [▶ 297] in diesem Dokument

st2PointCtrlSequence.rE: Regelabweichung: *rE* = *rW* - *rX*

st2PointCtrlSequence.rCtrl_I_HighLimit: Oberer Grenzwert an dem die Integration des internen I-Übertragungsgliedes angehalten wird. *st2PointCtrlSequence.rCtrl_I_HighLimit = rCtrl_I_HighLimit*

st2PointCtrlSequence.rCtrl_I_LowLimit: Unterer Grenzwert an dem die Integration des internen I-Übertragungsgliedes angehalten wird. *st2PointCtrlSequence.rCtrl_I_Low = rCtrl_I_LowLimit* * (-1)

st2PointCtrlSequence.rCtrl_I_Out: Ausgang des internen I-Übertragungsgliedes.
Ist *st2PointCtrlSequence.rCtrl_I_Out* = *st2PointCtrlSequence.rCtrl_I_HighLimit*ODER *st2PointCtrlSequence.rCtrl_I_LowLimit*UND
entweder *st2PointCtrlSequence.tRemainingTimeIncreaseSequence*ODER *st2PointCtrlSequence.tRemainingTimeDecreaseSequence* = T#0sUND
ist *bActive* = TRUE, so wird die Nummer des aktiven Reglers *iCurrentSequence* in Abhängigkeit von *bDirection* um eins erhöht oder erniedrigt.

st2PointCtrlSequence.e2PointCtrlState: siehe *e2PointCtrlState*

st2PointCtrlSequence.iNumberOfSequences: siehe *iNumberOfSequences*

st2PointCtrlSequence.iMyNumberInSequence: siehe *iMyNumberInSequence*

st2PointCtrlSequence.iCurrentSequence: siehe *iCurrentSequence*

st2PointCtrlSequence.bEnable: siehe *bEnable*

st2PointCtrlSequence.bError: siehe *bError*

st2PointCtrlSequence.bOut: siehe *bOut*

st2PointCtrlSequence.bActiveCtrl: siehe *bActiveCtrl*

st2PointCtrlSequence.b_rW_Max: siehe *b_rW_Max*

st2PointCtrlSequence.b_rW_Min: siehe *b_rW_Min*

bError: Der Ausgang signalisiert mit einem TRUE, dass ein Fehler anliegt und ein falscher Parameter an einer der Variablen *iNumberOfSequences, iMyNumberInSequence* oder *iCurrentSequence* anliegt. Nach Behebung des Fehlers muss die Meldung *bError* mit *bReset* quittiert werden. Das Enum *eErrorCode* zeigt die Fehlernummer an. Ist *bError* = TRUE, so wird der Ausgang *bOut* = FALSE.

Ist bEnable =FALSEODERbError = TRUEODER (e2PointCtrlMode =eHVAC2PointCtrlMode_On_BMSODEReHVAC2PointCtrlMode_On_OPODEReHVAC2PointCtrlMode_Off_BMSODEReHVAC2PointCtrlMode_Off_OP), so ist das automatische Umschalten der Regler vom aktiven auf den in der Regelsequenz rechts oder links benachbarten Regler weiterhin in Betrieb. Daher ist der Umschaltbetrieb immer aktiv. Mit bActiveCtrl = TRUE wird der aktive Funktionsbaustein angezeigt. Außerdem wird in Abhängigkeit der Regelabweichung rE das Rauf- oder Runterschalten der Sequenz über iCurrentSequence direkt ausgeführt. Wenn rX >= rW, dann ist iCurrentSequence = iCurrentSequence - 1. Wenn rX < rW, dann ist iCurrentSequence = iCurrentSequence + 1.

eErrorCode: Liefert bei einem gesetzten *bError*-Ausgang die Fehlernummer [▶ 529]. Folgende Fehler können in diesem Funktionsbaustein vorkommen: *eHVACErrorCodes_Error_iMyNumberInSequence*, *eHVACErrorCodes_Error_iNumberOfSequences*, *eHVACErrorCodes_Error_iCurrentSequences*

Um in der SPS an die Fehlernummern des Enums zu gelangen, kann eErrorCode einer Variablen vom Datentyp WORD zugewiesen werden.eHVACErrorCodes_Error_iNumberOfSequences = 27
eHVACErrorCodes_Error_iMyNumberInSequence = 28
eHVACErrorCodes_Error_iCurrentSequences = 29

VAR_IN_OUT

```
rW_HighLimit            : REAL;
rW_LowLimit             : REAL;
tDelayIncreaseSequence  : TIME;
tDelayDecreaseSequence  : TIME;
tCtrl_I_Ti_HighLimit    : TIME;
tCtrl_I_Ti_LowLimit     : TIME;
rCtrl_I_HighLimit       : REAL;
rCtrl_I_LowLimit        : REAL;
iCurrentSequence        : INT;
```

rW_HighLimit: Positiver Wert der oberen Grenze der Regelabweichung. *st2PointCtrlSequence.rW_Max =rW_Max := rW + rW_HighLimit.*
Die Variable wird persistent gespeichert. Voreingestellt auf 5.

rW_LowLimit: Positiver Wert der unteren Grenze der Regelabweichung. *st2PointCtrlSequence.rW_Min =rW_Min := rW - rW_LowLimit.*
Die Variable wird persistent gespeichert. Voreingestellt auf 5.

tDelayIncreaseSequence: Verzögerungszeit nach deren Ablauf *iCurrentSequence* um 1 erhöht wird, siehe Übertragungsfunktion des internen I-Übertragungsglied [▶ 297] in diesem Dokument. Die Variable wird persistent gespeichert. Voreingestellt auf 5min.

tDelayDecreaseSequence: Verzögerungszeit nach deren Ablauf *iCurrentSequence* um 1 erniedrigt wird, siehe Übertragungsfunktion des internen I-Übertragungsglied [▶ 297] in diesem Dokument. Die Variable wird persistent gespeichert. Voreingestellt auf 5min.

tCtrl_I_Ti_HighLimit: Integrationszeit für das obere Limit des internen I-Übertragungsgliedes, siehe Übertragungsfunktion des internen I-Übertragungsglied [▶ 297] in diesem Dokument. *tCtrl_I_Ti_HighLimit* muss > T#0s sein. Die Variable wird persistent gespeichert. Voreingestellt auf 10min.

tCtrl_I_Ti_LowLimit: Integrationszeit für das untere Limit des internen I-Übertragungsgliedes, siehe Übertragungsfunktion des internen I-Übertragungsglied [▶ 297] in diesem Dokument. *tCtrl_I_Ti_LowLimit* muss > T#0s sein. Die Variable wird persistent gespeichert. Voreingestellt auf 10min.

rCtrl_I_HighLimit: Positiver Wert für das obere Limit an dem die Integration des internen I-Übertragungsgliedes angehalten wird (ARW-Maßnahme, anti-reset-windup), siehe Übertragungsfunktion des internen I-Übertragungsglied [▶ 297] in diesem Dokument. Die Variable wird persistent gespeichert. Voreingestellt auf 10.

rCtrl_I_LowLimit: Negativer Wert für das untere Limit an dem die Integration des internen I-Übertragungsgliedes angehalten wird (ARW-Maßnahme, anti-reset-windup), siehe Übertragungsfunktion des internen I-Übertragungsglied [▶ 297] in diesem Dokument. Die Variable wird persistent gespeichert. Voreingestellt auf -10.

iCurrentSequence: Nummer des aktiven Reglers in der Sequenz (0..32). Die Anzahl der Teilnehmer in einer Sequenz *iNumberOfSequences* wird von dem aktiven Sequenzregler über *iCurrentSequence* nicht überschritten.
Ist *iCurrentSequence* > *iNumberOfSequences* oder *iCurrentSequence* < 0, so wird mit *bError* = TRUE ein Fehler angezeigt.
Das Rauf- oder Runterschalten der Sequenz über *iCurrentSequence* in Abhängigkeit der Regelabweichung *rE* findet dann statt, wenn der Funktionsbaustein aktiv in der Sequenz ist *bActiveCtrl* = TRUE.

1. iCurrentSequence = iCurrentSequence - 1 wenn st2PointCtrlSequence.rCtrl_I_Out >= st2PointCtrlSequence.rLimit_Ctrl_I_MinUNDst2PointCtrlSequence.tRemainingTimeDecreaseSequence = T#0sUNDbActiveCtrl = TRUE

2. iCurrentSequence = iCurrentSequence + 1 wenn st2PointCtrlSequence.rCtrl_I_Out >= st2PointCtrlSequence.rLimit_Ctrl_I_MaxUNDst2PointCtrlSequence.tRemainingTimeIncreaseSequence = T#0sUNDbActiveCtrl = TRUE

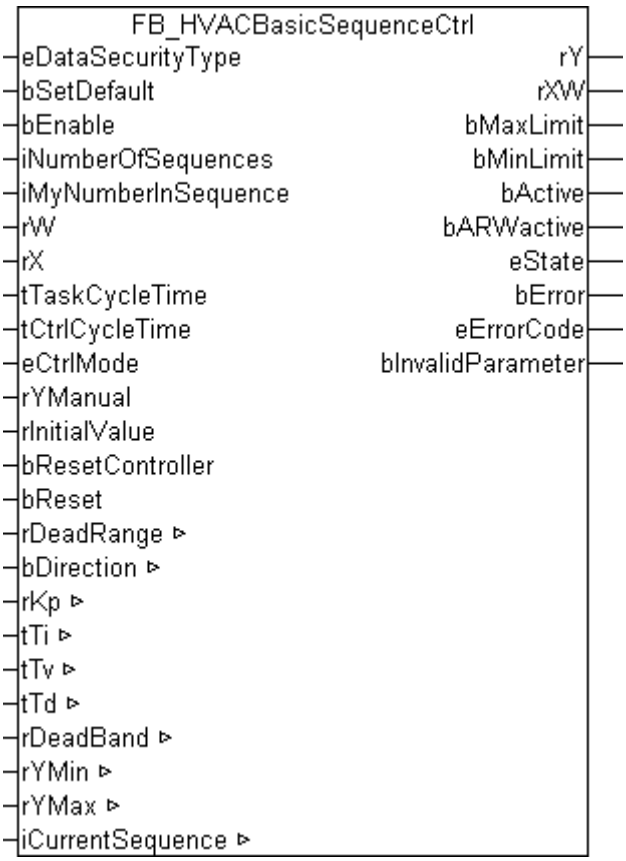
iCurrentSequence darf in einer Regelsequenz von außen nicht kontinuierlich beschrieben werden damit das automatische Umschalten der Regler vom aktiven auf den in der Regelsequenz rechts oder links benachbarten Regler funktioniert. Beim Starten einer Regelsequenz muss festgelegt werden welcher Sequenzregler aktiv ist. iCurrentSequence muss dazu für einen SPS-Zyklus beschrieben werden und > 0 und <= iNumberOfSequences sein.

Ist bEnable =FALSEODERbError = TRUEODER (e2PointCtrlMode =eHVAC2PointCtrlMode_On_BMSODEReHVAC2PointCtrlMode_On_OPODEReHVAC2PointCtrlMode_Off_BMSODEReHVAC2PointCtrlMode_Off_OP), so ist das automatische Umschalten der Regler vom aktiven auf den in der Regelsequenz rechts oder links benachbarten Regler weiterhin in Betrieb. Daher ist der Umschaltbetrieb immer aktiv. Mit bActiveCtrl = TRUE wird der aktive Funktionsbaustein angezeigt. Außerdem wird in Abhängigkeit der Regelabweichung rE das Rauf- oder Runterschalten der Sequenz über iCurrentSequence direkt ausgeführt. Wenn rX >= rW, dann ist iCurrentSequence = iCurrentSequence - 1. Wenn rX < rW, dann ist iCurrentSequence = iCurrentSequence + 1.

Dokumente hierzu

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

3.4.7.2 FB_HVACBasicSequenceCtrl



Anwendung

Der Funktionsbaustein FB_HVACBasicSequenceCtrl besitzt im Gegensatz zu den Sequenzreglern FB_HVACPIDCooling, FB_HVACDehumidify, FB_HVACEnergyRecovery, FB_HVACPIDHumidify, FB_HVACPIDMixedAir, FB_HVACPIDPreHeating und FB_HVACPIDReHeating keine besonderen anlagenspezifische Erweiterungen bzw. Applikation. Er ist allgemeiner gehalten. Der Wirksinn des Reglers wird nicht wie bei dem Funktionsbaustein für die Wärmerückgewinnung oder die Mischluftkammer automatisch in Abhängigkeit von Raum- und Außenluft bestimmt.

Mit diesem Funktionsbaustein kann im Raumautomationsbereich eine Sequenz aus statischer Heizung und Kühldecke realisiert werden.

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: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Eingangsvariable zur Freigabe des Reglers. Der Regler ist aktiv, wenn *bEnable* TRUE ist.

iNumberOfSequences: Anzahl der Sequenzregler in der Anlage.

iMyNumberInSequence: Eigene Nummer in des Reglers in der Sequenz.

rW: Mit der Variablen *rW* wird dem Regler der Sollwert übergeben.

rX: Istwert des Regelkreises.

tTaskCycleTime: Zykluszeit, mit der der Funktionsbaustein aufgerufen wird. Diese entspricht der Task-Zykluszeit der aufrufenden Task, wenn der Baustein in jedem Zyklus aufgerufen wird.

tCtrlCycleTime: Zykluszeit, mit der der Regelkreis bearbeitet wird. Diese muss größer oder gleich der TaskCycleTime sein. Der Funktionsbaustein berechnet mit dieser Eingangsgröße intern, ob die Zustands- und Ausgangsgrößen im aktuellen Zyklus aktualisiert werden müssen.

eCtrlMode: Über dieses Enum wird der Betriebsmodus ausgewählt.

rYManual: Manueller Wert.

rInitialValue: Mit dem *rInitialValue* wird das Neustartverhalten des Reglers beeinflusst.

bResetContoller: Eine positive Flanke am Eingang *bResetController* bewirkt einen Neustart des PID-Regler.

bReset: Quittierungseingang bei einer Störung.

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: Stellsignalausgang des PID-Reglers.

rXW: Regelabweichung

bMaxLimit: Der Ausgang *bMaxLimit* ist TRUE, wenn der Ausgang *rY* den Wert *rYMax*erreicht hat.

bMinLimit: Der Ausgang *bMinLimit* ist TRUE**,** wenn der Ausgang *rY* den Wert *rYMin*erreicht hat.

bActive: *bActive* ist TRUE**,** wenn der Regler aktiv und freigegeben ist.

bARWactive: *bARWactive*ist TRUE**,** wenn der Integralanteil des Regler die untere oder obere Stellgrößenlimitierung erreicht hat.

eState: Status vom Regler. Siehe E_HVACState. [▶ 532]

bError: Der Ausgang signalisiert mit einem TRUE, dass ein Fehler anliegt.

eErrorCode: Enthält den befehlsspezifischen Fehlercode. Siehe E_HVACErrorCodes. [▶ 529]

bInvalidParameter: TRUE, wenn bei der Plausibilitätsüberprüfung ein Fehler aufgetreten ist. Die Meldung muss mit *bReset* quittiert werden.

VAR_IN_OUT

```
rDeadRange       : REAL;
bDirection       : BOOL;
rKp              : REAL;
tTi              : TIME;
tTv              : TIME;
tTd              : TIME;
rDeadBand        : REAL;
rYMin            : REAL;
rYMax            : REAL;
iCurrentSequence: INT;
```

rDeadRange: Um unnötiges Verfahren und damit frühzeitiges Verschleißen der Ventile oder der Klappenantriebe zu vermeiden, kann für das Ausgangssignal *rY* des Reglers eine Totzone eingestellt werden (0..32767). Eine Stellsignaländerung wird damit erst aktiv, wenn die Wertänderung größer als die Totzone ist. Eine stetige Änderung des Stellsignals *rY* wird bei Angabe einer Totzone in ein pulsierendes Verfahren des Stellorganes umgewandelt. Je großer die Totzone desto größer sind die Pausen und Stellsignalsprünge. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

bDirection: Mit dem Parameter *bDirection* kann der Wirksinn des Reglers verändert werden. Ist *bDirection* TRUE ist der direkte Wirksinn für einen Kühlbetrieb des Reglers aktiv.
Wenn *bDirection* FALSE ist, ist der indirekte Wirksinn des Reglers für den Heizbetrieb aktiviert.
Die Variable wird persistent gespeichert. Voreingestellt auf FALSE.

rKp: Proportionalfaktor Verstärkung. Die Variable wird persistent gespeichert. Voreingestellt auf 1.

tTi: Integrierzeit. Der I-Anteil korrigiert die verbleibende Regelabweichung nach der Korrektur des P-Anteils. Je kleiner die *tTi*-Zeit eingestellt wird, desto schneller korrigiert der Regler. Ist die Zeit zu kurz wird der Regelkreis instabil. Um den Integrationsanteil zu vermindern, sind größere *tTi*-Zeiten einzugeben. Die Nachstellzeit sollte großer als die Verfahrzeit des Ventil- oder Klappenantriebes gewählt werden. Die Variable wird persistent gespeichert. Voreingestellt auf 30s.

tTv: Vorhaltezeit. Je größer *tTv*ist, desto stärker korrigiert der Regler. Eine zu große Zeit führt zu einem instabilen Regelkreis. In normalen Anwendungen der Gebäudeautomation wird häufig nur ein PI-Regler verwendet. In diesem Fall muss für *tTv* Null eingegeben werden. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

tTd: Dämpfungszeit. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

rDeadBand: Falls die Stellgröße am unteren oder oberen Limit eines Reglers ist, und der Istwert der Regelstrecke mit kleiner Amplitude um den Sollwert pendelt, kann ein häufiges Hin- und Herschalten zwischen zwei Sequenzreglern mit einem zusätzlichen Parameter für die Umschaltung gedämpft werden.

Dazu wird nachdem der Sequenzregler sein unteres oder oberes Limit erreicht hat, die Abweichung zwischen dem Istwert und dem Sollwert der Regelstrecke aufintegriert. Eine Umschaltung auf die nächste Sequenz erfolgt erst, wenn der Betrag dieser Integration größer ist als der Wert von *rDeadband* (Siehe Beispiel). [▶ 539] Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rYMin: Untere Begrenzung des Arbeitsbereiches vom Regler. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

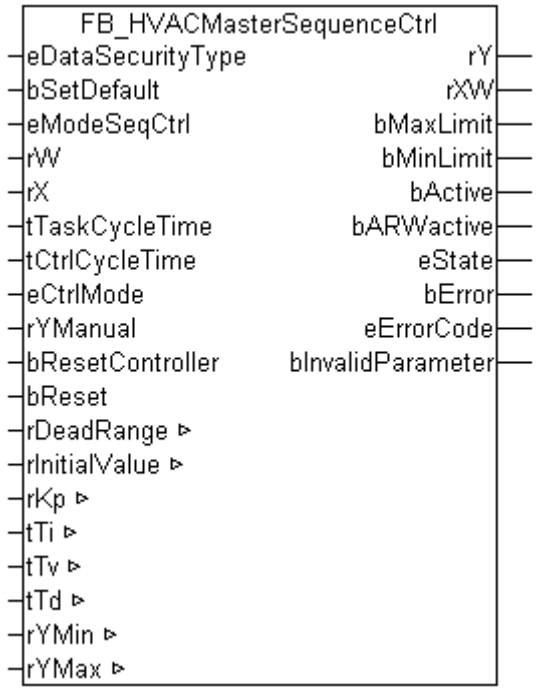
rYMax: Obere Begrenzung des Arbeitsbereiches vom Regler. Die Variable wird persistent gespeichert. Voreingestellt auf 100.

iCurrentSequence: Aktuell aktiver Regler in der Sequenz.

Dokumente hierzu

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

3.4.7.3 FB_HVACMasterSequenceCtrl



Anwendung

Eine verbesserte Regelbarkeit erreicht man mit Hilfe der Abluft- bzw. Raumtemperatur-Kaskadenregelung. Der Führungsregler misst die Raum- bzw. Ablufttemperatur und passt den Sollwert für die Zulufttemperatur den Verhältnissen im Raum an. Der Sollwert für die Zuluft wird durch einen Minimal- und einen Maximalwert begrenzt.

Im Stützbetrieb der Raumlufttechnischen Anlage*(eModeSeqCtrl = eHVACSequenceCtrlMode_FreezeProtection)* wird der Wert von *rYMax*direkt auf den Ausgang *rY* durchgeschaltet.Bei aktivem Überhitzungsschutz*(eModeSeqCtrl = eHVACSequenceCtrlMode_OverheatingProtection)* wird der Wert von *rYMin*direkt auf den Ausgang *rY* durchgeschaltet.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
eModeSeqCtrl     : E_HVACSequenceCtrlMode;
rW               : REAL;
rX               : REAL;
tTaskCycleTime   : TIME;
```

```
tCtrlCycleTime    : TIME;
eCtrlMode         : E_HVACCtrlMode;
rYManual          : REAL;
bResetController  : BOOL;
bReset            : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

eModeSeqCtrl: Mit dem Enum *eModeSeqCtrl* wird dem Regler die Anlagenbetriebsart übergeben.

rX: Erfasst den Istwert des Regelkreises.

rW: Mit der Variablen *rW* wird dem Regler der Sollwert übergeben.

tTaskCycleTime: Zykluszeit, mit der der Funktionsbaustein aufgerufen wird. Diese entspricht der Task-Zykluszeit der aufrufenden Task, wenn der Baustein in jedem Zyklus aufgerufen wird.

tCtrlCycleTime: Zykluszeit, mit der der Regelkreis bearbeitet wird. Diese muss größer oder gleich der *tTaskCycleTime* sein. Der Funktionsbaustein berechnet mit dieser Eingangsgröße intern, ob die Zustands- und Ausgangsgrößen im aktuellen Zyklus aktualisiert werden müssen.

eCtrlMode: Über dieses Enum wird der Betriebsmodus ausgewählt. Hand- oder Automatikbetrieb.

rYManual: Manueller Wert für den Handbetrieb.

bResetController: Die internen Variablen des PID-Regler werden zurückgesetzt.

bReset: Quittierungseingang bei einer Störung.

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: Temperatursollwert für die Zulufttemperaturregler.

rXW: Regelabweichung.

bMaxLimit: Der Ausgang *bMaxLimit* ist TRUE wenn der Ausgang *rY* den Wert *rYMax* erreicht hat.

bMinLimit: Der Ausgang *bMinLimit* ist TRUE wenn der Ausgang *rY* den Wert *rYMin*erreicht hat.

bActive: *bActive* ist TRUE**,** wenn der Regler aktiv und freigegeben ist.

bARWactive: *bARWactive* ist TRUE**,** wenn der Integralanteil vom Regler die untere oder obere Stellgrößenlimitierung erreicht hat.

eState: Status vom Regler. Siehe E_HVACState. [▶ 532]

bError: Der Ausgang signalisiert mit einem TRUE, dass ein Fehler anliegt.

eErrorCode: Enthält den befehlsspezifischen Fehlercode. Siehe E_HVACErrorCodes [▶ 529].

bInvalidParameter: TRUE, wenn bei der Plausibilitätsüberprüfung ein Fehler aufgetreten ist. Die Meldung muss mit *bReset* quittiert werden.

VAR_IN_OUT

```
rDeadRange          : REAL;
rInitialValue       : REAL;
rKp                 : REAL;
tTi                 : TIME;
tTv                 : TIME;
tTd                 : TIME;
rYMin               : REAL;
rYMax               : REAL;
```

rDeadRange: Um unnötige Verfahren und damit frühzeitiges Verschleißen der Ventile oder der Klappenantriebe zu vermeiden, kann für das Ausgangssignal *rY* des Reglers eine Totzone eingestellt werden (0..32767). Eine Stellsignaländerung wird damit erst aktiv, wenn die Wertänderung größer als die Totzone ist. Eine stetige Änderung des Stellsignals *rY* wird bei Angabe einer Totzone in ein pulsierendes Verfahren des Stellorgans umgewandelt. Je großer die Totzone desto größer sind die Pausen und Stellsignalsprünge. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rInitialValue: Mit dem *rInitialValue* wird das Neustartverhalten des Reglers beeinflusst (0..32767). Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rKp: Proportionalfaktor Verstärkung. Die Variable wird persistent gespeichert. Voreingestellt auf 1.

tTi: Integrierzeit. Der I-Anteil korrigiert die verbleibende Regelabweichung nach der Korrektur des P-Anteils. Je kleiner die *tTi*-Zeit eingestellt wird, desto schneller korrigiert der Regler. Ist die Zeit zu kurz, wird der Regelkreis instabil. Um den Integrationsanteil zu vermindern, sind größere *tTi*-Zeiten einzugeben. Die Nachstellzeit sollte großer als die Verfahrzeit des Ventil- oder Klappenantriebs gewählt werden. Die Variable wird persistent gespeichert. Voreingestellt auf 30s.

tTv: Vorhaltezeit. Je größer *tTv*ist, desto stärker korrigiert der Regler. Eine zu große Zeit führt zu einem instabilen Regelkreis. In normalen Anwendungen der Gebäudeautomation wird häufig nur ein PI-Regler verwendet. In diesem Fall muss für *tTv* Null eingegeben werden. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

tTd: Dämpfungszeit. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

rYMin: Untere Begrenzung des Arbeitsbereichs vom Regler. Die Variable wird persistent gespeichert. Voreingestellt auf 16.

rYMax: Obere Begrenzung des Arbeitsbereichs vom Regler. Die Variable wird persistent gespeichert. Voreingestellt auf 25.

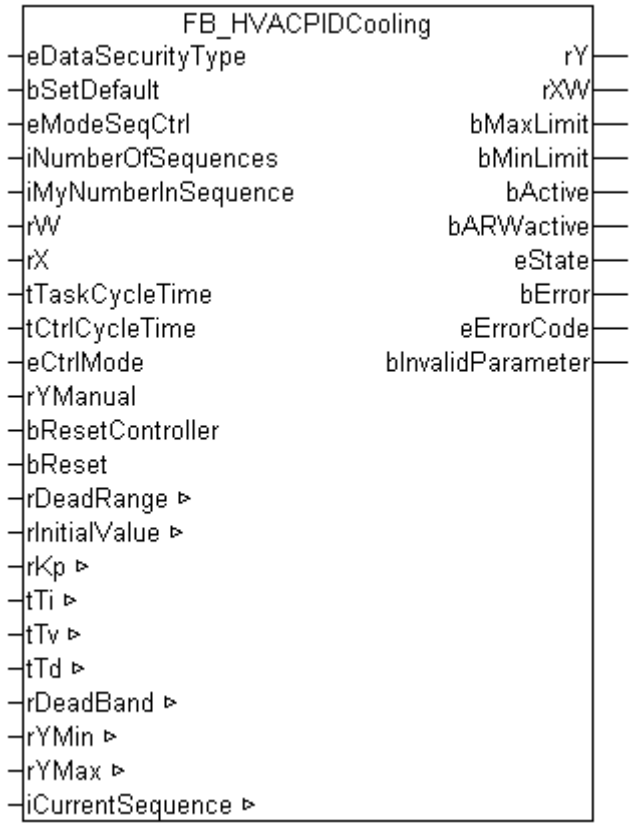
Dokumente hierzu

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

3.4.7.4 FB_HVACPIDCooling

PID-Regler Kühler

Der Wirksinn des Kühlreglers ist im Vergleich zum Heizregler invertiert. Bei einer Reduzierung des Zulufttemperatursollwertes erhöht der Sequenzregler sein Stellsignal. In der Betriebsart Überhitzungsschutz regelt der Kühler auf die minimale Zulufttemperatur. Außerhalb des normalen Sequenzregelbetriebs wird der Regler auch bei aktiven Überhitzungsschutz aktiviert.



VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

eModeSeqCtrl: Über diesen Enum erfolgt unter anderem die Freigabe der Regelung. Außerdem wird in der Sequenz die Betriebsart der RLT-Anlage an die Regelbausteine übertragen. Siehe auch Tabelle Betriebsarten. [▶ 541]

iNumberOfSequences: Anzahl der Sequenzregler in der Anlage.

iMyNumberInSequence: Die eigene Nummer des Reglers aus der Sequenz.

rW: Mit der Variablen rW wird dem Regler der Sollwert übergeben.

rX: Istwert des Regelkreises.

tTaskCycleTime: Zykluszeit, mit der der Funktionsbaustein aufgerufen wird. Diese entspricht der Task-Zykluszeit der aufrufenden Task, wenn der Baustein in jedem Zyklus aufgerufen wird.

tCtrlCycleTime: Mit der Variablen *tCtrlCycleTime* wird die Zykluszeit vorgegeben mit welcher der PID-Regler abgearbeitet wird. Die kleinste mögliche Zykluszeit ist die der Steuerung. Da die Regelstrecken in der Gebäudeautomation überwiegend langsam sind, kann die Zykluszeit des Reglers ein mehrfaches der Steuerungszykluszeit betragen.

eCtrlMode: Über dieses Enum wird der Betriebsmodus ausgewählt. Hand- oder Automatikbetrieb.

rYManual: Manueller Wert, der bei *eCtrlMode* = *eHVACCtrlMode_Manual* an den Ausgang *rY* gesetzt wird.

bResetController: Eine positive Flanke am Eingang *bResetController* bewirkt einen Neustart des PID-Regler.

bReset: Quittierungseingang bei einer Störung.

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: Stellsignalausgang des PID-Reglers.

rXW: Regelabweichung.

bMaxLimit: Der Ausgang *bMaxLimit* ist TRUE, wenn der Ausgang *rY* den Wert *rYMax* erreicht hat.

bMinLimit: Der Ausgang *bMinLimit* ist TRUE, wenn der Ausgang *rY* den Wert *rYMin* erreicht hat.

bActive: *bActive* ist TRUE, wenn der Regler aktiv und freigegeben ist.

bARWactive: *bARWactive* ist TRUE, wenn der Integralanteil des Reglers die untere oder obere Stellgrößenlimitierung erreicht hat.

eState: Status vom Regler. Siehe E_HVACState. [▶ 532]

bError: Der Ausgang signalisiert mit einem TRUE, dass ein Fehler anliegt.

eErrorCode: E_HVACErrorCodes. Siehe Fehlercodes. [▶ 529]

bInvalidParameter: TRUE, wenn bei der Plausibilitätsüberprüfung ein Fehler aufgetreten ist. Die Meldung muss mit *bReset* quittiert werden.

VAR_IN_OUT

```
rDeadRange       : REAL;
rInitialValue    : REAL;
rKp              : REAL;
tTi              : TIME;
tTv              : TIME;
tTd              : TIME;
rDeadBand        : REAL;
rYMin            : REAL;
rYMax            : REAL;
iCurrentSequence: INT;
```

rDeadRange: Um unnötiges Verfahren und damit frühzeitiges Verschleißen der Ventile oder Klappenantriebe zu vermeiden, kann für das Ausgangssignal *rY* des Reglers eine Totzone eingestellt werden. Eine Stellsignaländerung wird damit erst aktiv, wenn die Wertänderung größer als die Totzone ist. Eine stetige Änderung des Stellsignals *rY* wird bei Angabe einer Totzone in ein pulsierendes Verfahren des Stellorgans umgewandelt. Je größer die Totzone, desto größer sind die Pausen und Stellsignalsprünge. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rInitialValue: Mit dem *rInitialValue* wird das Neustartverhalten des Reglers beeinflusst. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rKp: Proportionalfaktor Verstärkung. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

tTi: Integrierzeit. Der I-Anteil korrigiert die verbleibende Regelabweichung nach der Korrektur des P-Anteils. Je kleiner die *tTi*-Zeit eingestellt wird, desto schneller korrigiert der Regler. Ist die Zeit zu kurz, wird der Regelkreis instabil. Um den Integrationsanteil zu vermindern, sind größere *tTi*-Zeiten einzugeben. Die Nachstellzeit sollte größer als die Verfahrzeit des Ventil- oder Klappenantriebes gewählt werden. Die Variable wird persistent gespeichert. Voreingestellt auf 30s.

tTv: Vorhaltezeit. Je größer *tTv* ist, desto stärker korrigiert der Regler. Eine zu große Zeit führt zu einem instabilen Regelkreis. In normalen Anwendungen der Gebäudeautomation wird häufig nur ein PI-Regler verwendet. In diesem Fall muss für *tTv* Null eingegeben werden. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

tTd: Dämpfungszeit. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

rDeadBand: Falls die Stellgröße am unteren oder oberen Limit eines Reglers ist, und der Istwert der Regelstrecke mit kleiner Amplitude um den Sollwert pendelt, kann ein häufiges Hin- und Herschalten zwischen zwei Sequenzreglern mit einem zusätzlichen Parameter für die Umschaltung gedämpft werden. Dazu wird nachdem der Sequenzregler sein unteres oder oberes Limit erreicht hat, die Abweichung zwischen dem Istwert und dem Sollwert der Regelstrecke aufintegriert. Eine Umschaltung auf die nächste Sequenz erfolgt erst, wenn der Betrag dieser Integration größer ist als der Wert von *rDeadband* (Siehe Beispiel). [▶ 539] Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rYMin: Untere Begrenzung des Arbeitsbereichs vom Regler. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rYMax: Obere Begrenzung des Arbeitsbereichs vom Regler. Die Variable wird persistent gespeichert. Voreingestellt auf 100.

iCurrentSequence: Nummer des aktiven Reglers aus der Sequenz.

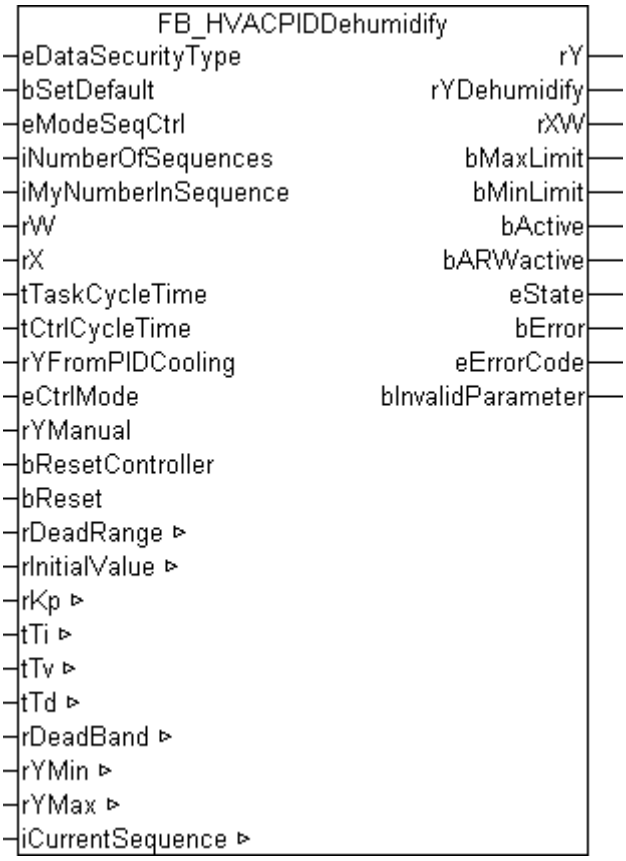
Dokumente hierzu

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

3.4.7.5 FB_HVACPIDDehumidify

PID-Regler Entfeuchtung

Zur Entfeuchtung wird die Zuluft mit dem Kühler herunter gekühlt. Durch die Dosierung des Kühlmediums am Kühlregister wird die Entfeuchtung stetig geregelt. Mit dem Regler FB_HVACPIDCooling [▶ 310]und dem Regler FB_HVACPIDDehumidify wirken zwei Regler auf das Ventil des Kühlers. Vom FB_HVACPIDCooling [▶ 310] wird das Stellsignal *rY* zunächst zum Sequenzregler Entfeuchten FB_HVACPIDDehumidify weitergeleitet. Innerhalb des Bausteins FB_HVACPIDDehumidify wird das größte der beiden Stellsignale an den Ausgang des Reglers weiter geleitet. Der Entfeuchtungsregler hat bei zu großer Luftfeuchte Vorrang vor dem Kühlregler. Um dennoch die richtige Zulufttemperatur erreichen zu können, wird der Vorerhitzer im Betriebsfall des Entfeuchtens gesperrt. Zur Nacherwärmung der Luft wird der Nacherhitzer in Betrieb genommen.



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: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

eModeSeqCtrl: Über dieses Enum erfolgt unter anderem die Freigabe der Regelung. Außerdem wird in der Sequenz die Betriebsart der RLT-Anlage an die Reglerbausteine übertragen. Siehe auch Tabelle Betriebsarten. [▶ 541]

iNumberOfSequences: Anzahl der Sequenzregler in der Anlage.

iMyNumberInSequence: Die eigene Nummer des Reglers in der Sequenz.

rW: Mit der Variablen rW wird dem Regler der Sollwert übergeben.

rX: Istwert des Regelkreises.

tTaskCycleTime: Zykluszeit, mit der der Funktionsbaustein aufgerufen wird. Diese entspricht der Task-Zykluszeit der aufrufenden Task, wenn der Baustein in jedem Zyklus aufgerufen wird.

tCtrlCycleTime: Mit der Variablen *tCtrlCycleTime* wird die Zykluszeit vorgegeben mit welcher der PID-Regler abgearbeitet wird. Die kleinste mögliche Zykluszeit ist die der Steuerung. Da die Regelstrecken in der Gebäudeautomation überwiegend langsam sind, kann die Zykluszeit des Reglers ein mehrfaches der Steuerungszykluszeit betragen.

rYFromPIDCooling: Bei einer RLT-Anlage mit Entfeuchtung wird an diesem Eingang der Ausgang des Kühlreglers angeschlossen. Innerhalb des Funktionsbausteins FB_HVACPIDDehumidify befindet sich eine MAX-Auswahl, die das größere Stellsignal der beiden Regler FB_HVACPIDDehumidify und FB_HVACPIDCooling an das Stellventil des Kühlers weiterleitet.

eCtrlMode: Über dieses Enum wird der Betriebsmodus ausgewählt. Hand- oder Automatikbetrieb.

rYManual: Manueller Wert, der bei *eCtrlMode* = *eHVACCtrlMode_Manual* an den Ausgang *rY* gesetzt wird.

bResetController: Eine positive Flanke am Eingang *bResetController* bewirkt einen Neustart des PID-Reglers.

bReset: Quittierungseingang bei einer Störung.

VAR_OUTPUT

```
rY                : REAL;
rYDehumidify      : REAL;
rXW               : REAL;
bMaxLimit         : BOOL;
bMinLimit         : BOOL;
bActive           : BOOL;
bARWactive        : BOOL;
eState            : E_HVACState;
bError            : BOOL;
eErrorCode        : E_HVACErrorCodes;
bInvalidParameter: BOOL;
```

rY: Stellsignalausgang des PID-Reglers.

rYDehumidify: Stellsignalausgang des Entfeuchtereglers.

rXW: Regelabweichung.

bMaxLimit: Der Ausgang *bMaxLimit* ist TRUE, wenn der Ausgang *rY* den Wert *rYMax* erreicht hat.

bMinLimit: Der Ausgang *bMinLimit* ist TRUE, wenn der Ausgang *rY* den Wert *rYMin* erreicht hat.

bActive: *bActive* ist TRUE, wenn der Regler aktiv und freigegeben ist.

bARWactive: *bARWactive* ist TRUE, wenn der Integralanteil des Regler die untere oder obere Stellgrößenlimitierung erreicht hat.

eState: Status vom Regler. Siehe E_HVACState. [▶ 532]

bError: Der Ausgang signalisiert mit einem TRUE, dass ein Fehler anliegt.

eErrorCode: Enthält den befehlsspezifischen Fehlercode. Siehe E_HVACErrorCodes. [▶ 529]

bInvalidParameter: TRUE, wenn bei der Plausibilitätsüberprüfung ein Fehler aufgetreten ist. Die Meldung muss mit *bReset* quittiert werden.

VAR_IN_OUT

```
rDeadRange       : REAL;
rInitialValue    : REAL;
rKp              : REAL;
tTi              : TIME;
tTv              : TIME;
tTd              : TIME;
rDeadBand        : REAL;
rYMin            : REAL;
rYMax            : REAL;
iCurrentSequence: INT;
```

rDeadRange: Um unnötiges Verfahren und damit frühzeitiges Verschleißen der Ventile oder Klappenantriebe zu vermeiden, kann für das Ausgangssignal *rY* des Reglers eine Totzone eingestellt werden. Eine Stellsignaländerung wird damit erst aktiv, wenn die Wertänderung größer als die Totzone ist. Eine stetige Änderung des Stellsignals *rY* wird bei Angabe einer Totzone in ein pulsierendes Verfahren des Stellorgans umgewandelt. Je größer die Totzone, desto größer sind die Pausen und Stellsignalsprünge. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rInitialValue: Mit dem *rInitialValue* wird das Neustartverhalten des Reglers beeinflusst. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rKp: Proportionalfaktor Verstärkung. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

tTi: Integrierzeit. Der I-Anteil korrigiert die verbleibende Regelabweichung nach der Korrektur des P-Anteils. Je kleiner die *tTi*-Zeit eingestellt wird, desto schneller korrigiert der Regler. Ist die Zeit zu kurz, wird der Regelkreis instabil. Um den Integrationsanteil zu vermindern, sind größere *tTi*-Zeiten einzugeben. Die Nachstellzeit sollte größer als die Verfahrzeit des Ventil- oder Klappenantriebes gewählt werden. Die Variable wird persistent gespeichert. Voreingestellt auf 30s.

tTv: Vorhaltezeit. Je größer *tTv* ist, desto stärker korrigiert der Regler. Eine zu große Zeit führt zu einem instabilen Regelkreis. In normalen Anwendungen der Gebäudeautomation wird häufig nur ein PI-Regler verwendet. In diesem Fall muss für *tTv* Null eingegeben werden. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

tTd: Dämpfungszeit. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

rDeadBand: Falls die Stellgröße am unteren oder oberen Limit eines Reglers ist, und der Istwert der Regelstrecke mit kleiner Amplitude um den Sollwert pendelt, kann ein häufiges Hin- und Herschalten zwischen zwei Sequenzreglern mit einem zusätzlichen Parameter für die Umschaltung gedämpft werden. Dazu wird nachdem der Sequenzregler sein unteres oder oberes Limit erreicht hat, die Abweichung zwischen dem Istwert und dem Sollwert der Regelstrecke aufintegriert. Eine Umschaltung auf die nächste Sequenz erfolgt erst, wenn der Betrag dieser Integration größer ist als der Wert von *rDeadband* (Siehe Beispiel). [▶ 539] Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rYMin: Untere Begrenzung des Arbeitsbereichs vom Regler. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rYMax: Obere Begrenzung des Arbeitsbereichs vom Regler. Die Variable wird persistent gespeichert. Voreingestellt auf 100.

iCurrentSequence: Nummer des aktiven Reglers aus der Sequenz.

Dokumente hierzu

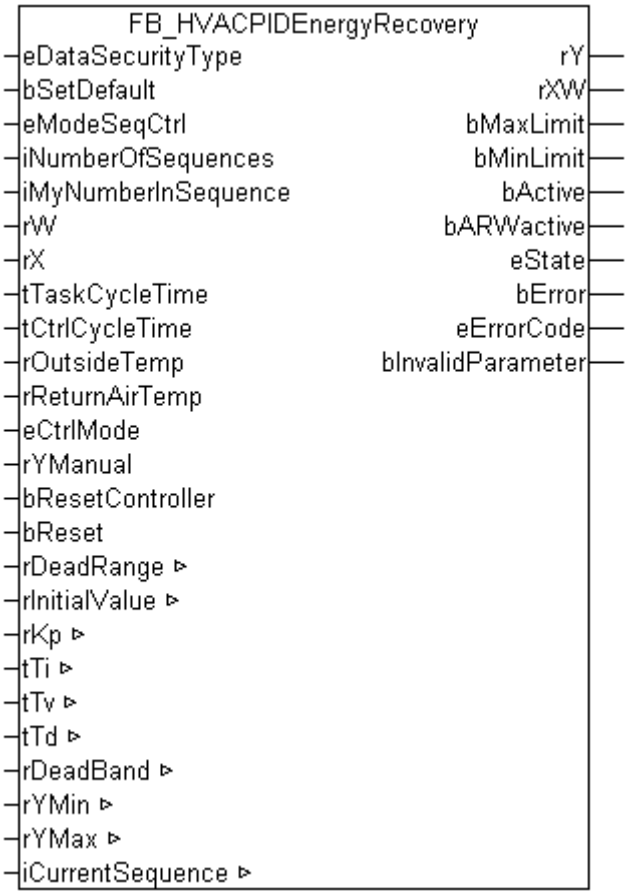
example_persistent_e.zip (Resources/zip/11659714827.zip)

3.4.7.6 FB_HVACPIDEnergyRecovery

PID-Regler Wärmerückgewinnung

Eine Besonderheit des Wärmerückgewinnungsreglers ist seine Wirksinnumkehr in Abhängigkeit der Außentemperatur und der Ablufttemperatur. Anstelle der Außen- und Ablufttemperaturen kann auch die Außen- und Abluftenthalpie verwendet werden.

Ist die Außentemperatur größer als die Ablufttemperatur kann das Wärmerückgewinnungssystem (WRG) benutzt werden, um die unter hohem Energieaufwand herunter gekühlte Abluft bzw. Raumluft zur Kühlung der Außenluft zu nutzen. Eine Herabsetzung der Zulufttemperatur führt im Kühlbetrieb zur einer Erhöhung der Stellgröße für die Wärmerückgewinnung (Kurve von links nach rechts fallend). Ist die Außentemperatur kleiner als die Ablufttemperatur, dann wird die WRG genutzt um die Wärme der Abluft in die Zuluft zurück zu führen. Damit befindet sich die WRG im Heizbetrieb (Kurve von links nach rechts steigend).



VAR_INPUT

```
eDataSecurityType  : E_HVACDataSecurityType;
bSetDefault        : BOOL;
eModeSeqCtrl       : E_HVACSequenceCtrlMode;
iNumberOfSequences : INT;
```

```
iMyNumberInSequence: INT;
rW                 : REAL;
rX                 : REAL;
tTaskCycleTime     : TIME;
tCtrlCycleTime     : TIME;
rOutsideTemp       : REAL;
rReturnAirTemp     : REAL;
eCtrlMode          : E_HVACCtrlMode;
rYManual           : REAL;
bResetController   : BOOL;
bReset             : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

eModeSeqCtrl: Über dieses Enum erfolgt unter anderem die Freigabe der Regelung. Außerdem wird in der Sequenz die Betriebsart der RLT-Anlage an die Reglerbausteine übertragen. Siehe auch Tabelle Betriebsarten. [▶ 541]

iNumberOfSequences: Anzahl der Sequenzregler in der Anlage.

iMyNumberInSequence: Die eigene Nummer des Reglers in der Sequenz.

rW: Mit der Variable rW wird dem Regler der Sollwert übergeben.

rX: Istwert des Regelkreises.

tTaskCycleTime: Zykluszeit, mit der der Funktionsbaustein aufgerufen wird. Diese entspricht der Task-Zykluszeit der aufrufenden Task, wenn der Baustein in jedem Zyklus aufgerufen wird.

tCtrlCycleTime: Mit der Variablen *tCtrlCycleTime* wird die Zykluszeit vorgegeben mit welcher der PID-Regler abgearbeitet wird. Die kleinste mögliche Zykluszeit ist die der Steuerung. Da die Regelstrecken in der Gebäudeautomation überwiegend langsam sind, kann die Zykluszeit des Reglers ein mehrfaches der Steuerungszykluszeit betragen.

rOutsideTemp: Außentemperatur.

rReturnAirTemp: Ablufttemperatur.

eCtrlMode: Über dieses Enum wird der Betriebsmodus ausgewählt. Hand- oder Automatikbetrieb.

rYManual: Manueller Wert, der bei *eCtrlMode* = *eHVACCtrlMode_Manual* an den Ausgang *rY* gesetzt wird.

bResetController: Eine positive Flanke am Eingang *bResetController* bewirkt einen Neustart des PID-Regler.

bReset: Quittierungseingang bei einer Störung.

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: Stellsignalausgang des PID-Reglers.

rXW: Regelabweichung

bMaxLimit: Der Ausgang *bMaxLimit* ist TRUE, wenn der Ausgang *rY* den Wert *rYMax*erreicht hat.

bMinLimit: Der Ausgang *bMinLimit* ist TRUE**,** wenn der Ausgang *rY* den Wert *rYMin*erreicht hat.

bActive: *bActive* ist TRUE**,** wenn der Regler aktiv und freigegeben ist.

bARWactive: *bARWactive*ist TRUE**,** wenn der Integralanteil des Reglers die untere oder obere Stellgrößenlimitierung erreicht hat.

eState: Status vom Regler. Siehe E_HVACState. [▶ 532]

bError: Der Ausgang signalisiert mit einem TRUE, dass ein Fehler anliegt.

eErrorCode: Enthält den befehlsspezifischen Fehlercode. Siehe E_HVACErrorCodes [▶ 529].

bInvalidParameter: TRUE, wenn bei der Plausibilitätsüberprüfung ein Fehler aufgetreten ist. Die Meldung muss mit *bReset* quittiert werden.

VAR_IN_OUT

```
rDeadRange       : REAL;
rInitialValue    : REAL;
rKp              : REAL;
tTi              : TIME;
tTv              : TIME;
tTd              : TIME;
rDeadBand        : REAL;
rYMin            : REAL;
rYMax            : REAL;
iCurrentSequence: INT;
```

rDeadRange: Um unnötiges Verfahren und damit frühzeitiges Verschleißen der Ventile oder Klappenantriebe zu vermeiden, kann für das Ausgangssignal *rY* des Reglers eine Totzone eingestellt werden. Eine Stellsignaländerung wird damit erst aktiv, wenn die Wertänderung größer als die Totzone ist. Eine stetige Änderung des Stellsignals *rY* wird bei Angabe einer Totzone in ein pulsierendes Verfahren des Stellorgans umgewandelt. Je größer die Totzone, desto größer sind die Pausen und Stellsignalsprünge. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rInitialValue: Mit dem *rInitialValue* wird das Neustartverhalten des Reglers beeinflusst. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rKp: Proportionalfaktor Verstärkung. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

tTi: Integrierzeit. Der I-Anteil korrigiert die verbleibende Regelabweichung nach der Korrektur des P-Anteils. Je kleiner die *tTi*-Zeit eingestellt wird, desto schneller korrigiert der Regler. Ist die Zeit zu kurz, wird der Regelkreis instabil. Um den Integrationsanteil zu vermindern, sind größere *tTi*-Zeiten einzugeben. Die Nachstellzeit sollte größer als die Verfahrzeit des Ventil- oder Klappenantriebes gewählt werden. Die Variable wird persistent gespeichert. Voreingestellt auf 30s.

tTv: Vorhaltezeit. Je größer *tTv* ist, desto stärker korrigiert der Regler. Eine zu große Zeit führt zu einem instabilen Regelkreis. In normalen Anwendungen der Gebäudeautomation wird häufig nur ein PI-Regler verwendet. In diesem Fall muss für *tTv* Null eingegeben werden. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

tTd: Dämpfungszeit. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

rDeadBand: Falls die Stellgröße am unteren oder oberen Limit eines Reglers ist, und der Istwert der Regelstrecke mit kleiner Amplitude um den Sollwert pendelt, kann ein häufiges Hin- und Herschalten zwischen zwei Sequenzreglern mit einem zusätzlichen Parameter für die Umschaltung gedämpft werden. Dazu wird nachdem der Sequenzregler sein unteres oder oberes Limit erreicht hat, die Abweichung zwischen dem Istwert und dem Sollwert der Regelstrecke aufintegriert. Eine Umschaltung auf die nächste Sequenz erfolgt erst, wenn der Betrag dieser Integration größer ist als der Wert von *rDeadband* (Siehe Beispiel). [▶ 539] Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rYMin: Untere Begrenzung des Arbeitsbereichs vom Regler. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rYMax: Obere Begrenzung des Arbeitsbereichs vom Regler. Die Variable wird persistent gespeichert. Voreingestellt auf 100.

iCurrentSequence: Nummer des aktiven Reglers aus der Sequenz.

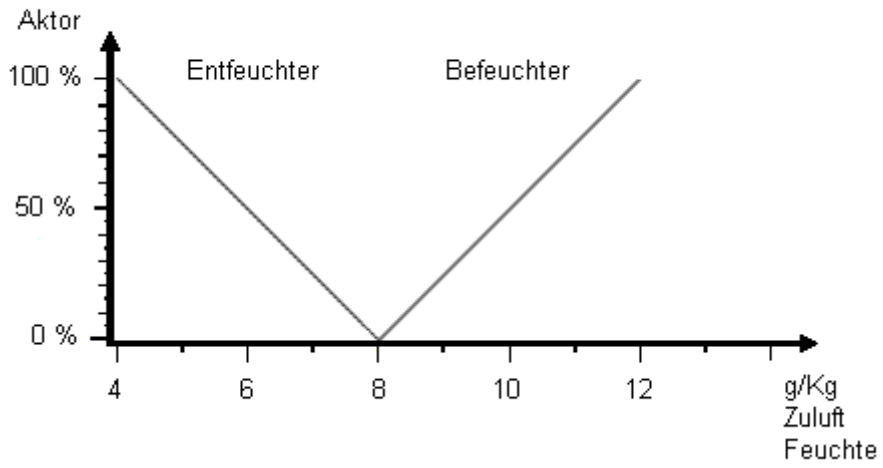
Dokumente hierzu

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

3.4.7.7 FB_HVACPIDHumidify

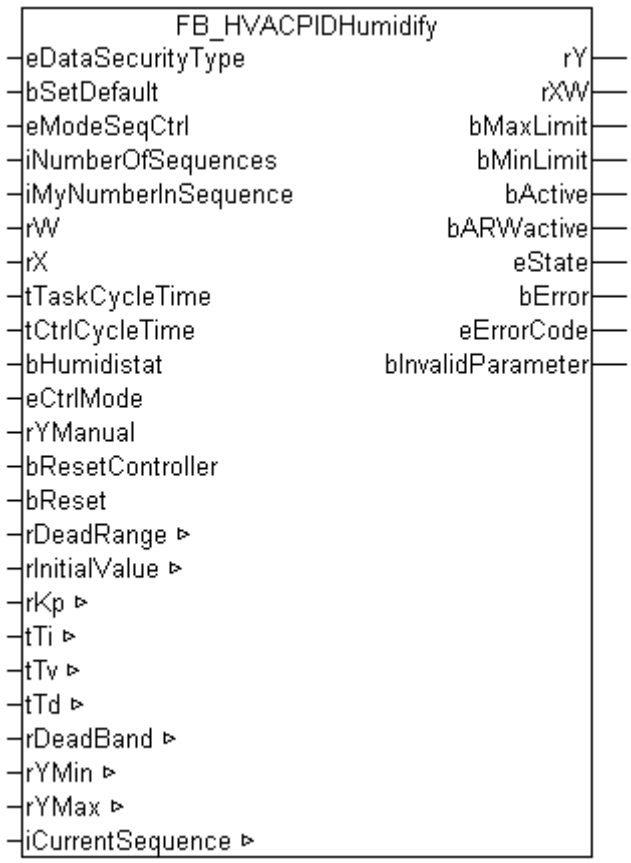
Feuchteregelung

Neben der Temperatur wird in einigen Raumlufttechnischen Anlagen auch die Luftfeuchte geregelt. Zum Befeuchten werden Dampfbefeuchter oder Luftwäscher verwendet. Zur Entfeuchtung kann die Zuluft mit einem Kühler bis zum Taupunkt herunter gekühlt werden. Damit kondensiert die Luftfeuchtigkeit am Luftkühler aus. Die Regelung zur Be- und Entfeuchtung wird mit einer Sequenz aus einem Befeuchtungs- und einem Entfeuchtungsregler realisiert.



PID-Regler Befeuchtung

Der Funktionsbaustein ist eine spezielle Applikation eines PID-Reglers. Die Feuchteregelung wird nur frei gegeben, wenn der Eingang *bHumidistat* des Funktionsbausteins TRUE ist.



VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

eModeSeqCtrl: Über dieses Enum erfolgt unter anderem die Freigabe der Regelung Außerdem wird in der Sequenz die Betriebsart der RLT-Anlage an die Regelbausteine übertragen. Siehe auch Tabelle Betriebsarten. [▶ 541]

iNumberOfSequences: Anzahl der Sequenzregler in der Anlage.

iMyNumberInSequence: Die eigene Nummer vom Regler in der Sequenz.

rW: Mit der Variable rW wird dem Regler der Sollwert übergeben.

rX: Istwert des Regelkreises.

tTaskCycleTime: Zykluszeit, mit der der Funktionsbaustein aufgerufen wird. Diese entspricht der Task-Zykluszeit der aufrufenden Task, wenn der Baustein in jedem Zyklus aufgerufen wird.

tCtrlCycleTime: Mit der Variablen *tCtrlCycleTime* wird die Zykluszeit vorgegeben mit der der PID-Regler abgearbeitet wird. Die kleinste mögliche Zykluszeit ist die der Steuerung. Da die Regelstrecken in der Gebäudeautomation überwiegend langsam sind, kann die Zykluszeit des Reglers ein mehrfaches der Steuerungszykluszeit betragen.

bHumidistat: Eingang, der die Feuchteregelung über ein TRUE freigibt. An diesen Eingang wird das Hygrostat geschaltet, damit keine Überbefeuchtung stattfinden kann.

eCtrlMode: Über dieses Enum wird der Betriebsmodus ausgewählt. Hand- oder Automatikbetrieb.

rYManual: Manueller Wert, der bei *eCtrlMode* = *eHVACCtrlMode_Manual* an den Ausgang *rY* gesetzt wird.

bResetController: Eine positive Flanke am Eingang *bResetController* bewirkt einen Neustart des PID-Reglers.

bReset: Quittierungseingang bei einer Störung.

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: Stellsignalausgang des PID-Reglers.

rXW: Regelabweichung.

bMaxLimit: Der Ausgang *bMaxLimit* ist TRUE, wenn der Ausgang *rY* den Wert *rYMax*erreicht hat.

bMinLimit: Der Ausgang *bMinLimit* ist TRUE**,** wenn der Ausgang *rY* den Wert *rYMin*erreicht hat.

bActive: *bActive* ist TRUE**,** wenn der Regler aktiv und freigegeben ist.

bARWactive: *bARWactive*ist TRUE**,** wenn der Integralanteil des Reglers die untere oder obere Stellgrößenlimitierung erreicht hat.

eState: Status vom Regler. Siehe E_HVACState. [▶ 532]

bError: Der Ausgang signalisiert mit einem TRUE, dass ein Fehler anliegt.

eErrorCode: Enthält den befehlsspezifischen Fehlercode. Siehe E_HVACErrorCodes [▶ 529].

bInvalidParameter: TRUE, wenn bei der Plausibilitätsüberprüfung ein Fehler aufgetreten ist. Die Meldung muss mit *bReset* quittiert werden.

VAR_IN_OUT

```
rDeadRange      : REAL;
rInitialValue   : REAL;
rKp             : REAL;
tTi             : TIME;
tTv             : TIME;
tTd             : TIME;
rDeadBand       : REAL;
rYMin           : REAL;
rYMax           : REAL;
iCurrentSequence: INT;
```

rDeadRange: Um unnötiges Verfahren und damit frühzeitiges Verschleißen der Ventile oder Klappenantriebe zu vermeiden, kann für das Ausgangssignal *rY* des Reglers eine Totzone eingestellt werden. Eine Stellsignaländerung wird damit erst aktiv, wenn die Wertänderung größer als die Totzone ist. Eine stetige Änderung des Stellsignals *rY* wird bei Angabe einer Totzone in ein pulsierendes Verfahren des Stellorgans umgewandelt. Je größer die Totzone, desto größer sind die Pausen und Stellsignalsprünge. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rInitialValue: Mit dem *rInitialValue* wird das Neustartverhalten des Reglers beeinflusst. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rKp: Proportionalfaktor Verstärkung. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

tTi: Integrierzeit. Der I-Anteil korrigiert die verbleibende Regelabweichung nach der Korrektur des P-Anteils. Je kleiner die *tTi*-Zeit eingestellt wird, desto schneller korrigiert der Regler. Ist die Zeit zu kurz, wird der Regelkreis instabil. Um den Integrationsanteil zu vermindern, sind größere *tTi*-Zeiten einzugeben. Die Nachstellzeit sollte größer als die Verfahrzeit des Ventil- oder Klappenantriebes gewählt werden. Die Variable wird persistent gespeichert. Voreingestellt auf 30s.

tTv: Vorhaltezeit. Je größer *tTv* ist, desto stärker korrigiert der Regler. Eine zu große Zeit führt zu einem instabilen Regelkreis. In normalen Anwendungen der Gebäudeautomation wird häufig nur ein PI-Regler verwendet. In diesem Fall muss für *tTv* Null eingegeben werden. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

tTd: Dämpfungszeit. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

rDeadBand: Falls die Stellgröße am unteren oder oberen Limit eines Reglers ist, und der Istwert der Regelstrecke mit kleiner Amplitude um den Sollwert pendelt, kann ein häufiges Hin- und Herschalten zwischen zwei Sequenzreglern mit einem zusätzlichen Parameter für die Umschaltung gedämpft werden. Dazu wird nachdem der Sequenzregler sein unteres oder oberes Limit erreicht hat, die Abweichung zwischen dem Istwert und dem Sollwert der Regelstrecke aufintegriert. Eine Umschaltung auf die nächste Sequenz erfolgt erst, wenn der Betrag dieser Integration größer ist als der Wert von *rDeadband* (Siehe Beispiel). [▶ 539] Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rYMin: Untere Begrenzung des Arbeitsbereichs vom Regler. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

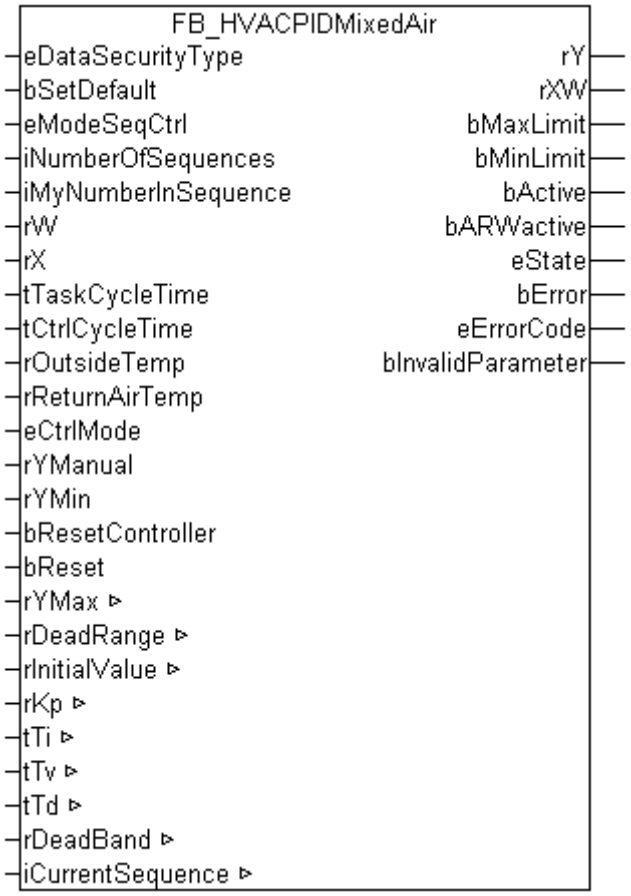
rYMax: Obere Begrenzung des Arbeitsbereichs vom Regler. Die Variable wird persistent gespeichert. Voreingestellt auf 100.

iCurrentSequence: Nummer des aktiven Reglers aus der Sequenz.

Dokumente hierzu

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

3.4.7.8 FB_HVACPIDMixedAir



PID-Regler Mischluftkammer

Eine Besonderheit des Mischluftreglers ist seine Wirksinnumkehr in Abhängigkeit der Außentemperatur und Ablufttemperatur. Anstelle der Außen- und Ablufttemperaturen können auch die Außen- und Abluftenthalpie verwendet werden.

Ist die Außentemperatur kleiner als die Ablufttemperatur so kann die Luft bei hohen thermischen Lasten im Gebäude zur Kühlung genutzt werden. Es steigt also die Außenluftrate mit einer Verringerung der Regelgröße bzw. Verringerung des Zulufttemperatursollwerts (Kurve von links nach rechts fallend).

Ist die Außentemperatur größer als die Ablufttemperatur, so kann der Außenluftanteil bei einer Erhöhung des Zulufttemperatursollwertes vergrößert werden (Kurve von links nach rechts steigend). Die Außenlufttemperatur wird an der Eingangsvariablen *rOutsideTemp* angelegt. Die Variable *rReturnAirTemp* ist für die Abluft oder Raumtemperatur. Die Betriebsart Nachtauskühlung *(eHVACSequenceCtrlMode_NightCooling)*wird dem Funktionsbaustein mittels des EnumseModeSeqCtr*l* übertragen. Während des nächtlichen Auskühlens wird die Regelung deaktiviert und am Ausgang *rY* werden 100% Stellgröße ausgegeben. Weitere Informationen zur Funktion der Sommernachtkühlung sind unter FB_HVACSummerNightCooling [▶ 468] zu finden.

Mit den Parametern *rYMax* und *rYMin* kann die maximale und die minimale Außenluftrate eingestellt werden.

VAR_INPUT

```
eDataSecurityType   : E_HVACDataSecurityType;
bSetDefault         : BOOL;
eModeSeqCtrl        : E_HVACSequenceCtrlMode;
iNumberOfSequences  : INT;
iMyNumberInSequence: INT;
rW                  : REAL;
rX                  : REAL;
tTaskCycleTime      : TIME;
tCtrlCycleTime      : TIME;
```

```
rOutsideTemp        : REAL;
rReturnAirTemp      : REAL;
eCtrlMode           : E_HVACCtrlMode;
rYManual            : REAL;
rYMin               : REAL;
bResetController    : BOOL;
bReset              : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

eModeSeqCtrl: Über dieses Enum erfolgt unter anderem die Freigabe der Regelung. Außerdem wird in der Sequenz die Betriebsart der RLT-Anlage an die Regelbausteine übertragen. Siehe auch Tabelle Betriebsarten. [▶ 541]

iNumberOfSequences: Anzahl der Sequenzregler in der Anlage.

iMyNumberInSequence: Die eigene Nummer vom Regler aus der Sequenz.

rW: Mit der Variablen *rW* wird dem Regler der Sollwert übergeben.

rX: Istwert des Regelkreises.

tTaskCycleTime: Zykluszeit, mit der der Funktionsbaustein aufgerufen wird. Diese entspricht der Task-Zykluszeit der aufrufenden Task, wenn der Baustein in jedem Zyklus aufgerufen wird.

tCtrlCycleTime: Mit der Variablen *tCtrlCycleTime* wird die Zykluszeit vorgegeben mit der der PID-Regler abgearbeitet wird. Die kleinste mögliche Zykluszeit ist die der Steuerung. Da die Regelstrecken in der Gebäudeautomation überwiegend langsam sind, kann die Zykluszeit des Reglers ein mehrfaches der Steuerungszykluszeit betragen.

rOutsideTemp: Außentemperatur

rReturnAirTemp: Ablufttemperatur

eCtrlMode: Über dieses Enum wird der Betriebsmodus ausgewählt. Hand- oder Automatikbetrieb.

rYManual: Manueller Wert, der bei *eCtrlMode* = *eHVACCtrlMode_Manual* an den Ausgang *rY* gesetzt wird.

rYMin: Untere Begrenzung des Arbeitsbereiches vom Regler.

bResetController: Eine positive Flanke am Eingang *bResetController* bewirkt einen Neustart des PID-Regler.

bReset: Quittierungseingang bei einer Störung.

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: Stellsignalausgang des PID-Reglers.

rXW: Regelabweichung

bMaxLimit: Der Ausgang *bMaxLimit* ist TRUE, wenn der Ausgang *rY* den Wert *rYMax*erreicht hat.

bMinLimit: Der Ausgang *bMinLimit* ist TRUE**,** wenn der Ausgang *rY* den Wert *rYMin*erreicht hat.

bActive: *bActive* ist TRUE**,** wenn der Regler aktiv und freigegeben ist.

bARWactive: *bARWactive*ist TRUE**,** wenn der Integralanteil des Reglers die untere oder obere Stellgrößenlimitierung erreicht hat.

eState: Status vom Regler. Siehe E_HVACState. [▶ 532]

bError: Der Ausgang signalisiert mit einem TRUE, dass ein Fehler anliegt.

eErrorCode: Enthält den befehlsspezifischen Fehlercode. Siehe E_HVACErrorCodes [▶ 529].

bInvalidParameter: TRUE, wenn bei der Plausibilitätsüberprüfung ein Fehler aufgetreten ist. Die Meldung muss mit *bReset* quittiert werden.

VAR_IN_OUT

```
rDeadRange       : REAL;
rInitialValue    : REAL;
rKp              : REAL;
tTi              : TIME;
tTv              : TIME;
tTd              : TIME;
rDeadBand        : REAL;
rYMax            : REAL;
iCurrentSequence: INT;
```

rDeadRange: Um unnötiges Verfahren und damit frühzeitiges Verschleißen der Ventile oder Klappenantriebe zu vermeiden, kann für das Ausgangssignal *rY* des Reglers eine Totzone eingestellt werden. Eine Stellsignaländerung wird damit erst aktiv, wenn die Wertänderung größer als die Totzone ist. Eine stetige Änderung des Stellsignals *rY* wird bei Angabe einer Totzone in ein pulsierendes Verfahren des Stellorgans umgewandelt. Je größer die Totzone, desto größer sind die Pausen und Stellsignalsprünge. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rInitialValue: Mit dem *rInitialValue* wird das Neustartverhalten des Reglers beeinflusst. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rKp: Proportionalfaktor Verstärkung. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

tTi: Integrierzeit. Der I-Anteil korrigiert die verbleibende Regelabweichung nach der Korrektur des P-Anteils. Je kleiner die *tTi*-Zeit eingestellt wird, desto schneller korrigiert der Regler. Ist die Zeit zu kurz, wird der Regelkreis instabil. Um den Integrationsanteil zu vermindern, sind größere *tTi*-Zeiten einzugeben. Die Nachstellzeit sollte größer als die Verfahrzeit des Ventil- oder Klappenantriebes gewählt werden. Die Variable wird persistent gespeichert. Voreingestellt auf 30s.

tTv: Vorhaltezeit. Je größer *tTv* ist, desto stärker korrigiert der Regler. Eine zu große Zeit führt zu einem instabilen Regelkreis. In normalen Anwendungen der Gebäudeautomation wird häufig nur ein PI-Regler verwendet. In diesem Fall muss für *tTv* Null eingegeben werden. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

tTd: Dämpfungszeit. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

rDeadBand: Falls die Stellgröße am unteren oder oberen Limit eines Reglers ist, und der Istwert der Regelstrecke mit kleiner Amplitude um den Sollwert pendelt, kann ein häufiges Hin- und Herschalten zwischen zwei Sequenzreglern mit einem zusätzlichen Parameter für die Umschaltung gedämpft werden. Dazu wird nachdem der Sequenzregler sein unteres oder oberes Limit erreicht hat, die Abweichung zwischen dem Istwert und dem Sollwert der Regelstrecke aufintegriert. Eine Umschaltung auf die nächste Sequenz erfolgt erst, wenn der Betrag dieser Integration größer ist als der Wert von *rDeadband* (Siehe Beispiel). [▶ 539] Die Variable wird persistent gespeichert. Voreingestellt auf 0.

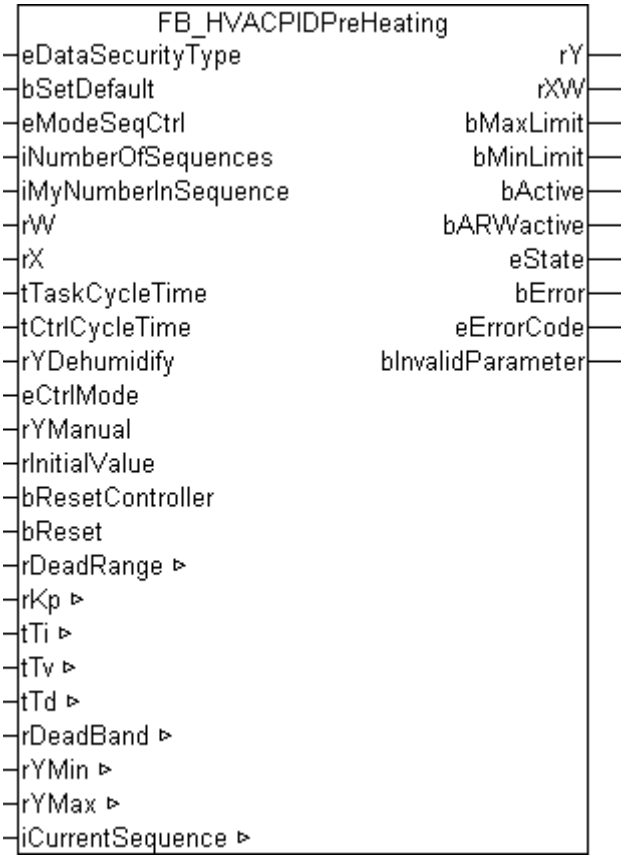
rYMax: Obere Begrenzung des Arbeitsbereichs vom Regler. Die Variable wird persistent gespeichert. Voreingestellt auf 100.

iCurrentSequence: Nummer des aktiven Reglers aus der Sequenz.

Dokumente hierzu

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

3.4.7.9 FB_HVACPIDPreHeating



PID-Regler Vorerhitzer

Der Funktionsbaustein ist eine spezielle Applikation eines PID-Reglers.
Neben dem PID-Regelalgorithmus sind die Bedingungen des Wechselns zur Sequenz des vor- oder nachgelagerten Sequenzreglers enthalten. Eine Besonderheit des Vorerhitzerreglers ist der Eingang *rYDehumidify*. Sobald das Signal von *rYDehumidify* größer als Null ist, wird der Vorerhitzer gesperrt und der Wert von*iCurrentSequence*um eins erhöht. Damit wird statt des Vorerhitzers der Nacherhitzer FB_HVACPIDReHeating [▶ 329] [▶ 329]aktiviert. Geht der Wert von *rYDehumidify* wieder auf Null, dann wird automatisch wieder auf den Vorerhitzer zurück geschaltet.Außerhalb des normalen Sequenzregelbetriebs wird der Regler auch bei aktiven Stützbetrieb (FreezeProtection) aktiviert.

VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel:  https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

eModeSeqCtrl: Über dieses Enum erfolgt unter anderem die Freigabe der Regelung. Außerdem wird in der Sequenz die Betriebsart der RLT-Anlage an die Reglerbausteine übertragen. Siehe auch Tabelle Betriebsarten. [▶ 541]

iNumberOfSequences: Anzahl der Sequenzregler in der Anlage.

iMyNumberInSequence: Die eigene Nummer des Reglers in der Sequenz.

rW: Mit der Variable rW wird dem Regler der Sollwert übergeben.

rX: Istwert des Regelkreises.

tTaskCycleTime: Zykluszeit, mit der der Funktionsbaustein aufgerufen wird. Diese entspricht der Task-Zykluszeit der aufrufenden Task, wenn der Baustein in jedem Zyklus aufgerufen wird.

tCtrlCycleTime: Mit der Variablen *rCtrlCycleTime* wird die Zykluszeit vorgegeben mit der der PID-Regler abgearbeitet wird. Die kleinste mögliche Zykluszeit ist die der Steuerung. Da die Regelstrecken in der Gebäudeautomation überwiegend langsam sind, kann die Zykluszeit des Reglers ein mehrfaches der Steuerungszykluszeit betragen.

rYDehumidify: Bei einer RLT-Anlage mit Entfeuchtung wird an diesen Eingang der Stellwert des Entfeuchtereglers FB_HVACPIDDehumidify [▶ 313] angeschlossen. Ist der Wert von *rYDehumidify* > 0, dann wird die VAR_IN_OUT Variable *iCurrentSequence* automatisch um eins erhöht. Damit wird im Entfeuchtebetrieb die Anlage vom Vorerhitzer **FB_HVACPIDPreHeating** auf den Nacherhitzer **FB_HVACPIDReHeating** umgeschaltet.

eCtrlMode: Über dieses Enum wird der Betriebsmodus ausgewählt. Hand- oder Automatikbetrieb.

rYManual: Manueller Wert, der bei *eCtrlMode* = *eHVACCtrlMode_Manual* an den Ausgang *rY* gesetzt wird.

rInitialValue: Mit dem *rInitialValue* wird das Neustartverhalten des Reglers beeinflusst.

bResetController: Eine positive Flanke am Eingang *bResetController* bewirkt einen Neustart des PID-Reglers.

bReset: Quittierungseingang bei einer Störung.

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: Stellsignalausgang des PID-Reglers.

rXW: Regelabweichung

bMaxLimit: Der Ausgang *bMaxLimit* ist TRUE, wenn der Ausgang *rY* den Wert *rYMax*erreicht hat.

bMinLimit: Der Ausgang *bMinLimit* ist TRUE**,** wenn der Ausgang *rY* den Wert *rYMin*erreicht hat.

bActive: *bActive* ist TRUE**,** wenn der Regler aktiv und freigegeben ist.

bARWactive: *bARWactive*ist TRUE**,** wenn der Integralanteil des Reglers die untere oder obere Stellgrößenlimitierung erreicht hat.

eState: Status vom Regler. Siehe E_HVACState. [▶ 532]

bError: Der Ausgang signalisiert mit einem TRUE, dass ein Fehler anliegt.

eErrorCode: Enthält den befehlsspezifischen Fehlercode. Siehe E_HVACErrorCodes [▶ 529].

bInvalidParameter: TRUE, wenn bei der Plausibilitätsüberprüfung ein Fehler aufgetreten ist. Die Meldung muss mit *bReset* quittiert werden.

VAR_IN_OUT

```
rDeadRange      : REAL;
rKp             : REAL;
tTi             : TIME;
tTv             : TIME;
tTd             : TIME;
rDeadBand       : REAL;
rYMin           : REAL;
rYMax           : REAL;
iCurrentSequence: INT;
```

rDeadRange: Um unnötiges Verfahren und damit frühzeitiges Verschleißen der Ventile oder Klappenantriebe zu vermeiden, kann für das Ausgangssignal *rY* des Reglers eine Totzone eingestellt werden. Eine Stellsignaländerung wird damit erst aktiv, wenn die Wertänderung größer als die Totzone ist. Eine stetige Änderung des Stellsignals *rY* wird bei Angabe einer Totzone in ein pulsierendes Verfahren des Stellorgans umgewandelt. Je größer die Totzone, desto größer sind die Pausen und Stellsignalsprünge. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rKp: Proportionalfaktor Verstärkung. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

tTi: Integrierzeit. Der I-Anteil korrigiert die verbleibende Regelabweichung nach der Korrektur des P-Anteils. Je kleiner die *tTi*-Zeit eingestellt wird, desto schneller korrigiert der Regler. Ist die Zeit zu kurz, wird der Regelkreis instabil. Um den Integrationsanteil zu vermindern, sind größere *tTi*-Zeiten einzugeben. Die Nachstellzeit sollte größer als die Verfahrzeit des Ventil- oder Klappenantriebes gewählt werden. Die Variable wird persistent gespeichert. Voreingestellt auf 30s.

tTv: Vorhaltezeit. Je größer *tTv* ist, desto stärker korrigiert der Regler. Eine zu große Zeit führt zu einem instabilen Regelkreis. In normalen Anwendungen der Gebäudeautomation wird häufig nur ein PI-Regler verwendet. In diesem Fall muss für *tTv* Null eingegeben werden. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

tTd: Dämpfungszeit. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

rDeadBand: Falls die Stellgröße am unteren oder oberen Limit eines Reglers ist, und der Istwert der Regelstrecke mit kleiner Amplitude um den Sollwert pendelt, kann ein häufiges Hin- und Herschalten zwischen zwei Sequenzreglern mit einem zusätzlichen Parameter für die Umschaltung gedämpft werden. Dazu wird nachdem der Sequenzregler sein unteres oder oberes Limit erreicht hat, die Abweichung zwischen dem Istwert und dem Sollwert der Regelstrecke aufintegriert. Eine Umschaltung auf die nächste Sequenz erfolgt erst, wenn der Betrag dieser Integration größer ist als der Wert von *rDeadband* (Siehe Beispiel). [▶ 539] Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rYMin: Untere Begrenzung des Arbeitsbereichs vom Regler. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

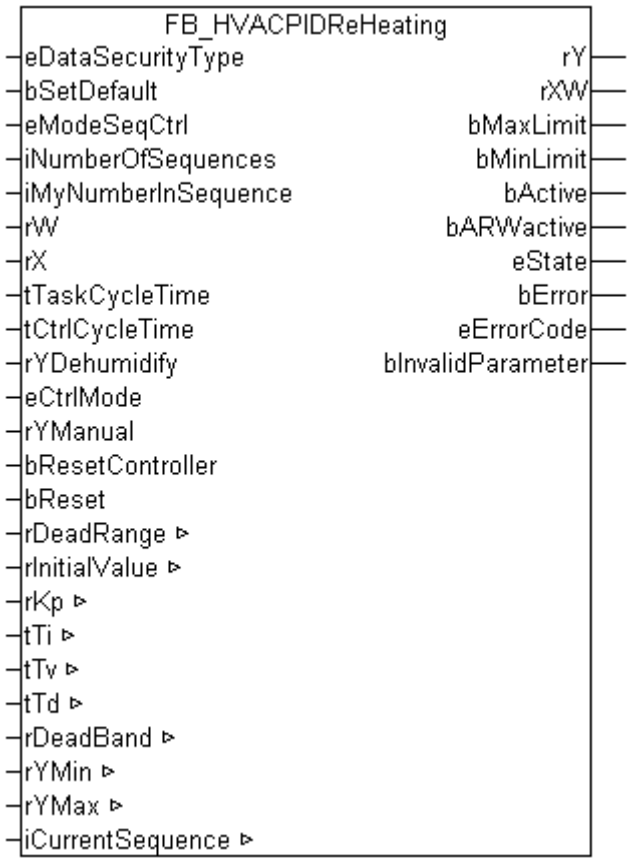
rYMax: Obere Begrenzung des Arbeitsbereichs vom Regler. Die Variable wird persistent gespeichert. Voreingestellt auf 100.

iCurrentSequence: Nummer des aktiven Reglers aus der Sequenz.

Dokumente hierzu

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

3.4.7.10 FB_HVACPIDReHeating



PID-Regler Nacherhitzer

Der Nacherhitzer wird zur Nacherwärmung, der vom Kühler zur Entfeuchtung herab gekühlten Zuluft verwendet. Eine Freigabe des Reglers erfolgt nur innerhalb des normalen **Sequenzregelbetriebes**.

VAR_INPUT

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

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

eModeSeqCtrl: Über diesen Enum erfolgt unter anderem die Freigabe der Regelung. Außerdem wird in der Sequenz die Betriebsart der RLT-Anlage wird an die Regelbausteine übertragen. Siehe auch Tabelle Betriebsarten. [▶ 541]

iNumberOfSequences: Anzahl der Sequenzregler in der Anlage.

iMyNumberInSequence: Die eigene Nummer des Reglers in der Sequenz.

rW: Mit der Variable rW wird dem Regler der Sollwert übergeben.

rX: Istwert des Regelkreises.

tTaskCycleTime: Zykluszeit, mit der der Funktionsbaustein aufgerufen wird. Diese entspricht der Task-Zykluszeit der aufrufenden Task, wenn der Baustein in jedem Zyklus aufgerufen wird.

tCtrlCycleTime: Mit der Variablen *tCtrlCycleTime* wird die Zykluszeit vorgegeben mit der der PID-Regler abgearbeitet wird. Die kleinste mögliche Zykluszeit ist die der Steuerung. Da die Regelstrecken in der Gebäudeautomation überwiegend langsam sind, kann die Zykluszeit des Reglers ein mehrfaches der Steuerungszykluszeit betragen.

rYDehumidify: Stetiger Wert.

eCtrlMode: Über dieses Enum wird der Betriebsmodus ausgewählt. Hand- oder Automatikbetrieb.

rYManual: Manueller Wert, der bei *eCtrlMode* = *eHVACCtrlMode_Manual* an den Ausgang *rY* gesetzt wird.

bResetController: Eine positive Flanke am Eingang *bResetController* bewirkt einen Neustart des PID-Regler.

bReset: Quittierungseingang bei einer Störung.

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: Stellsignalausgang des PID-Reglers.

rXW: Regelabweichung.

bMaxLimit: Der Ausgang *bMaxLimit* ist TRUE, wenn der Ausgang *rY* den Wert *rYMax*erreicht hat.

bMinLimit: Der Ausgang *bMinLimit* ist TRUE**,** wenn der Ausgang *rY* den Wert *rYMin*erreicht hat.

bActive: *bActive* ist TRUE**,** wenn der Regler aktiv und freigegeben ist.

bARWactive: *bARWactive*ist TRUE**,** wenn der Integralanteil des Reglers die untere oder obere Stellgrößenlimitierung erreicht hat.

eState: Status des Reglers. Siehe E_HVACState. [▶ 532]

bError: Der Ausgang signalisiert mit einem TRUE, dass ein Fehler anliegt.

eErrorCode: Enthält den befehlsspezifischen Fehlercode. Siehe E_HVACErrorCodes [▶ 529].

bInvalidParameter: TRUE, wenn bei der Plausibilitätsüberprüfung ein Fehler aufgetreten ist. Die Meldung muss mit *bReset* quittiert werden.

VAR_IN_OUT

```
rDeadRange       : REAL;
rInitialValue    : REAL;
rKp              : REAL;
tTi              : TIME;
tTv              : TIME;
tTd              : TIME;
rDeadBand        : REAL;
rYMin            : REAL;
rYMax            : REAL;
iCurrentSequence: INT;
```

rDeadRange: Um unnötiges Verfahren und damit frühzeitiges Verschleißen der Ventile oder Klappenantriebe zu vermeiden, kann für das Ausgangssignal *rY* des Reglers eine Totzone eingestellt werden. Eine Stellsignaländerung wird damit erst aktiv, wenn die Wertänderung größer als die Totzone ist. Eine stetige Änderung des Stellsignals *rY* wird bei Angabe einer Totzone in ein pulsierendes Verfahren des Stellorgans umgewandelt. Je größer die Totzone, desto größer sind die Pausen und Stellsignalsprünge. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rInitialValue: Mit dem *rInitialValue* wird das Neustartverhalten des Reglers beeinflusst. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rKp: Proportionalfaktor Verstärkung. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

tTi: Integrierzeit. Der I-Anteil korrigiert die verbleibende Regelabweichung nach der Korrektur des P-Anteils. Je kleiner die *tTi*-Zeit eingestellt wird, desto schneller korrigiert der Regler. Ist die Zeit zu kurz, wird der Regelkreis instabil. Um den Integrationsanteil zu vermindern, sind größere *tTi*-Zeiten einzugeben. Die Nachstellzeit sollte größer als die Verfahrzeit des Ventil- oder Klappenantriebes gewählt werden. Die Variable wird persistent gespeichert. Voreingestellt auf 30s.

tTv: Vorhaltezeit. Je größer *tTv* ist, desto stärker korrigiert der Regler. Eine zu große Zeit führt zu einem instabilen Regelkreis. In normalen Anwendungen der Gebäudeautomation wird häufig nur ein PI-Regler verwendet. In diesem Fall muss für *tTv* Null eingegeben werden. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

tTd: Dämpfungszeit. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

rDeadBand: Falls die Stellgröße am unteren oder oberen Limit eines Reglers ist, und der Istwert der Regelstrecke mit kleiner Amplitude um den Sollwert pendelt, kann ein häufiges Hin- und Herschalten zwischen zwei Sequenzreglern mit einem zusätzlichen Parameter für die Umschaltung gedämpft werden. Dazu wird nachdem der Sequenzregler sein unteres oder oberes Limit erreicht hat, die Abweichung zwischen dem Istwert und dem Sollwert der Regelstrecke aufintegriert. Eine Umschaltung auf die nächste Sequenz erfolgt erst, wenn der Betrag dieser Integration größer ist als der Wert von *rDeadband* (Siehe Beispiel). [▶ 539] Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rYMin: Untere Begrenzung des Arbeitsbereichs vom Regler. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rYMax: Obere Begrenzung des Arbeitsbereichs vom Regler. Die Variable wird persistent gespeichert. Voreingestellt auf 100.

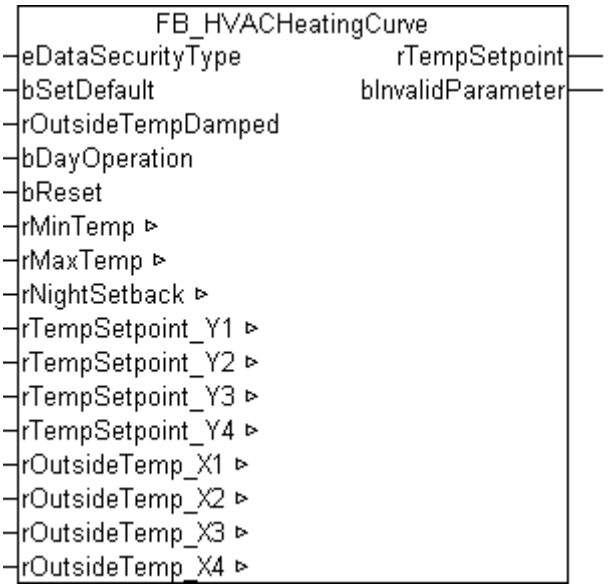
iCurrentSequence: Nummer des aktiven Reglers aus der Sequenz.

Dokumente hierzu

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

3.5 HLK Sollwertmodule

3.5.1 FB_HVACHeatingCurve



Anwendung

Die Heizkurve mit vier Punkten dient der Ermittlung des Sollwertes für die Vorlauftemperatur eines Heizkreises in Abhängigkeit der Außentemperatur.

Dieser Funktionsbaustein muss funktionsbedingt mit dem **FB_HVACSetpointHeating** zusammen verwendet werden. Der Grund dafür ist, dass der Betrag für die Nachtabsenkung beim **FB_HVACHeatingCurve** berücksichtigt wird.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
rOutsideTempDamped: REAL;
bDayOperation     : BOOL;
bReset            : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

rOutsideTempDamped: Mit dieser Eingangsvariablen wird dem Funktionsbaustein die aktuelle gedämpfte Außentemperatur übergeben.

bDayOperation: TRUE = Tagbetrieb, FALSE = Nachtbetrieb.

bReset: Quittierungseingang bei einer Störung.

VAR_OUTPUT

```
rTempSetpoint    : REAL;     0 .. 500      ° C
bInvalidParameter: BOOL;
```

rTempSetpoint: Der errechnete Sollwert für die Vorlauftemperatur.

bInvalidParameter: Zeigt an, dass ein falscher Eingangsparameter anliegt. *bInvalidParameter* muss mit *bReset* quittiert werden.

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: Mit dieser Variable wird der minimale Wert des Sollwertes für die Vorlauftemperatur definiert. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

rMaxTemp: Mit dieser Variable wird der maximale Wert des Sollwertes für die Vorlauftemperatur definiert. Die Variable wird persistent gespeichert. Voreingestellt auf 500.

rNightSetback: Mit dieser Variable wird der Betrag der Nachtabsenkung angegeben. Die Variable wird persistent gespeichert. Voreingestellt auf 20.

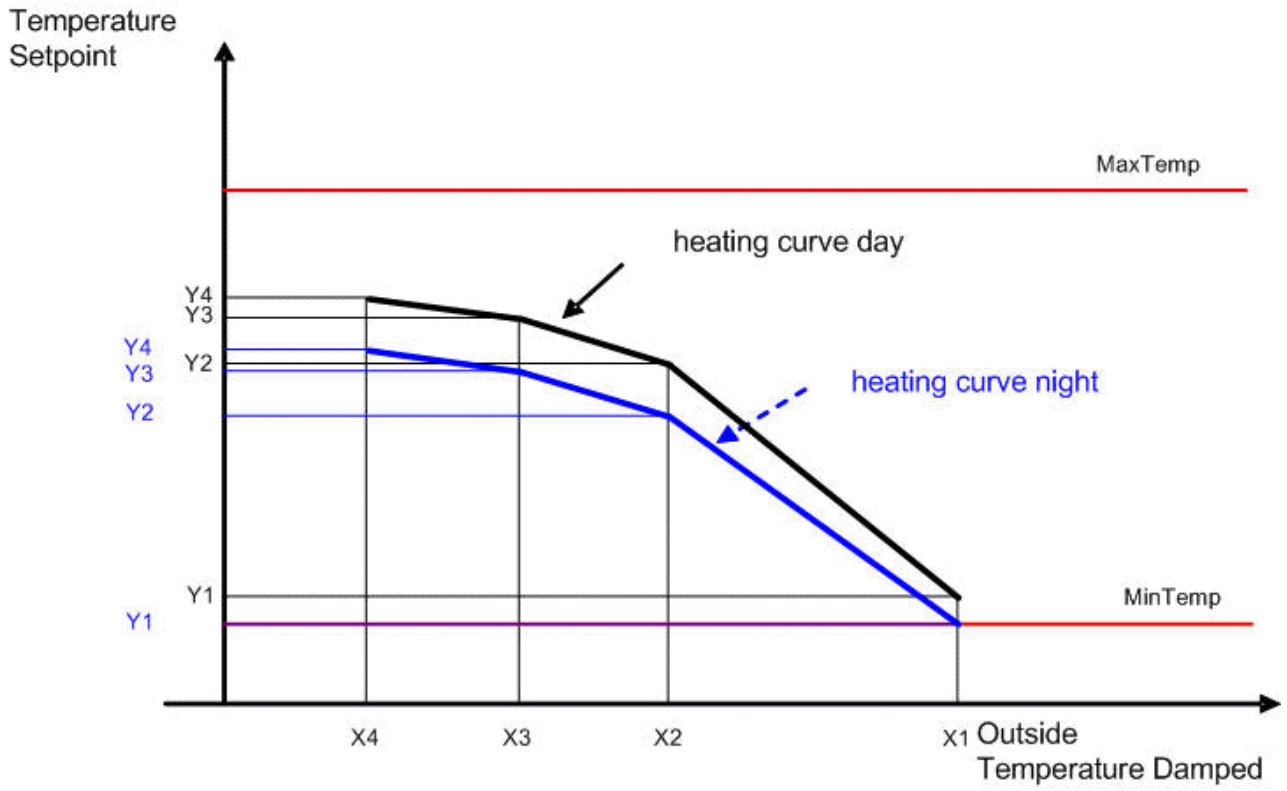
rTempSetpoint_Y1 / rOutsideTemp_X1: Mit diesem Wertepaar wird der Verlauf der Heizkurve Punkt 1 parametriert. Die Variable wird persistent gespeichert. Voreingestellt auf 20.

rTempSetpoint_Y2 / rOutsideTemp_X2: Mit diesem Wertepaar wird der Verlauf der Heizkurve Punkt 2 parametriert. Die Variable wird persistent gespeichert. Voreingestellt auf 65 und 0.

rTempSetpoint_Y3 / rOutsideTemp_X3: Mit diesem Wertepaar wird der Verlauf der Heizkurve Punkt 3 parametriert. Die Variable wird persistent gespeichert. Voreingestellt auf 74 und -10.

rTempSetpoint_Y4 / rOutsideTemp_X4: Mit diesem Wertepaar wird der Verlauf der Heizkurve Punkt 4 parametriert. Die Variable wird persistent gespeichert. Voreingestellt auf 80 und -20.

Verlauf von den Heizungskennlinien



Bedingungen

Für die Eingabe der Werte gilt folgendes: X1 > X 2 > X3 > X4 und Y1 < Y2 < Y3 <Y4.

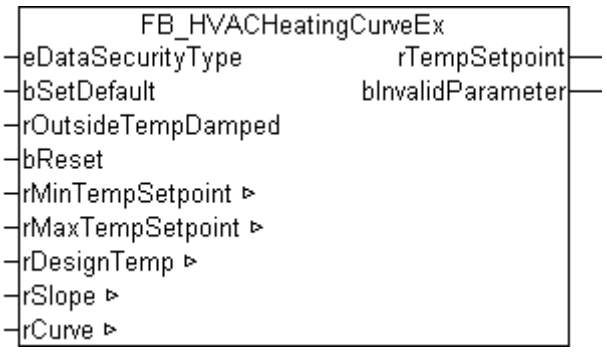
Des Weiteren muss der minimale Wert des Sollwertes für die Vorlauftemperatur <= *rTempSetpoint_Y1* und >= 0 sein. Der maximale Wert des Sollwertes für die Vorlauftemperatur muss >= *rTempSetpoint_Y4* sein.

Wird eine dieser Bedingungen nicht erfüllt; wird die Variable *bInvalidParameter* auf TRUE gesetzt und die Defaultwerte der VAR_IN_OUT Variablen werden übernommen.

Dokumente hierzu

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

3.5.2 FB_HVACHeatingCurveEx



Anwendung

Die Heizkurve dient der Ermittlung des Sollwertes für die Vorlauftemperatur eines Heizkreises in Abhängigkeit der Außentemperatur. Im Vergleich zu dem FB_HVACHeatingCurve ist die Heizkurve als eine Funktion abgebildet.

$$a = rSlope * e^{(b * \ln(z))}$$

$$b = 1/\ln(4) * \ln(1 + rCurve) \qquad z, u < 0 \Longrightarrow z,u = 0.0001$$

$$rTempSetpoint = 20 + a * e^{((1-b) * \ln(u))}$$

z= 20 - rDesignTemp;

u= 20 - rOutsideTemp;

VAR_INPUT

```
eDataSecurityType    : E_HVACDataSecurityType;
bSetDefault          : BOOL;
rOutsideTempDamped   : REAL;
bDayOperation        : BOOL;
bReset               : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

rOutsideTempDamped: Mit dieser Eingangsvariablen wird dem Funktionsbaustein die aktuelle gedämpfte Außentemperatur übergeben.

bReset: Quittierungseingang bei einer Störung.

VAR_OUTPUT

```
rTempSetpoint       : REAL;           ° C
bInvalidParameter   : BOOL;
```

rTempSetpoint: Der errechnete Sollwert für die Vorlauftemperatur.

bInvalidParameter: Zeigt an, dass ein falscher Eingangsparameter anliegt. *bInvalidParameter* muss mit *bReset* quittiert werden.

VAR_IN_OUT

```
rMinTempSetpoint   : REAL;
rMaxTempSetpoint   : REAL;
rDesignTemp        : REAL;
rSlope             : REAL;
rCurve             : REAL;
```

rMinTempSetpoint: Mit dieser Variable wird der minimale Wert des Sollwertes für die Vorlauftemperatur [°C] definiert. Die Variable wird persistent gespeichert. Voreingestellt auf 20.

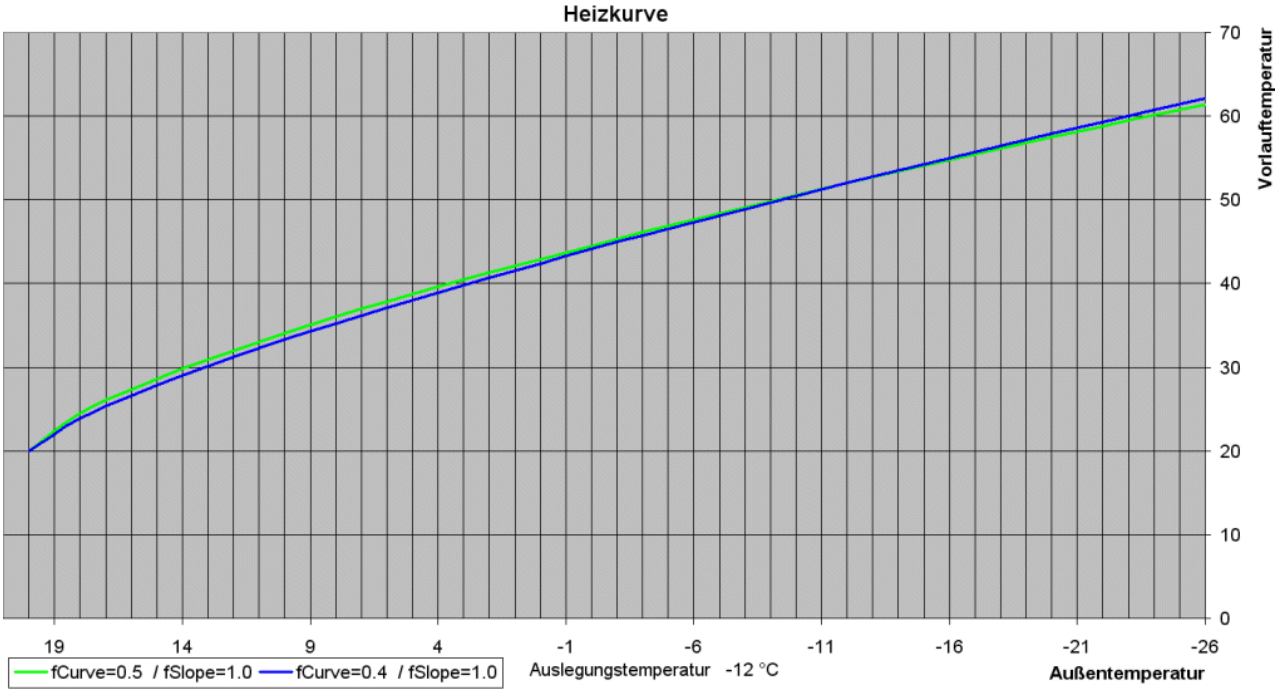
rMaxTempSetpoint: Mit dieser Variable wird der maximale Wert des Sollwertes für die Vorlauftemperatur [°C] definiert. Die Variable wird persistent gespeichert. Voreingestellt auf 90.

rDesignTemp: Auslegungstemperatur [°C] für die Dimensionierung einer Heizungsanlage. Typische Werte für Deutschland sind zwischen -12°C und -16°C. Die Variable wird persistent gespeichert. Voreingestellt auf -16.

rSlope: Faktor für die Steigung. Die Variable wird persistent gespeichert. Voreingestellt auf 1.

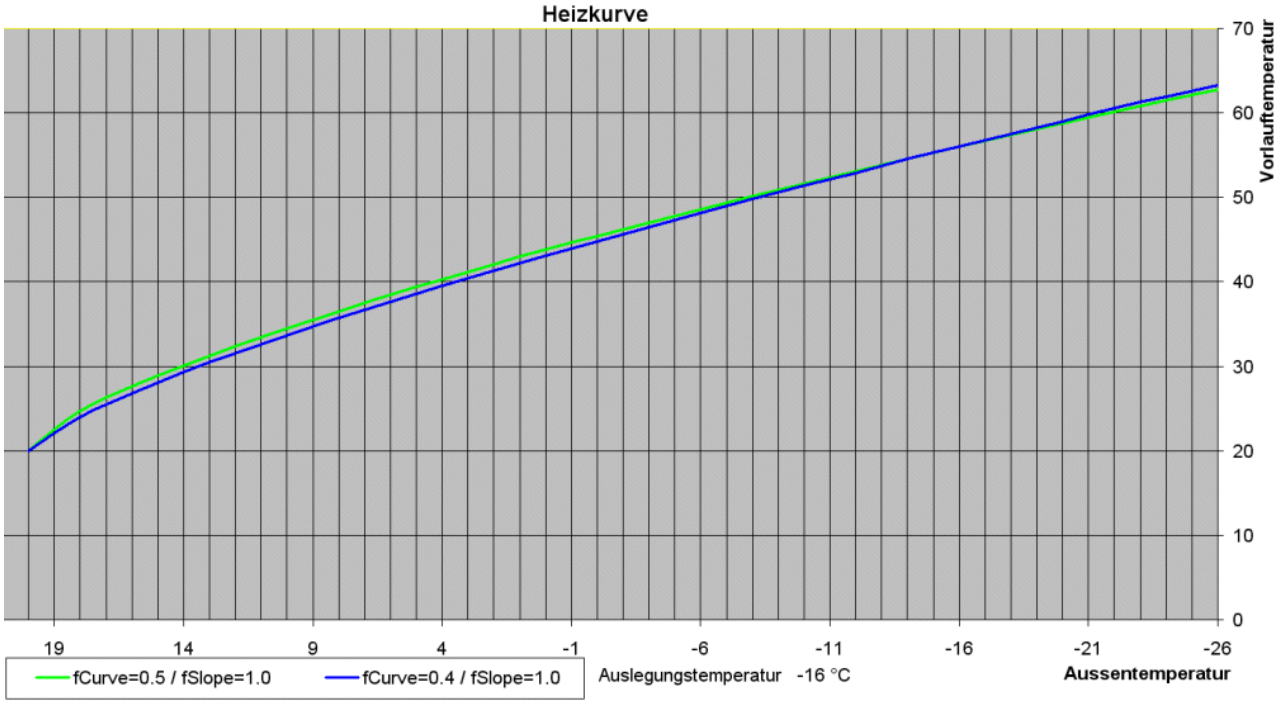
rCurve: Faktor für die Krümmung. Die Variable wird persistent gespeichert. Voreingestellt auf 0,5.

Verlauf von den Heizkurvenkennlinien



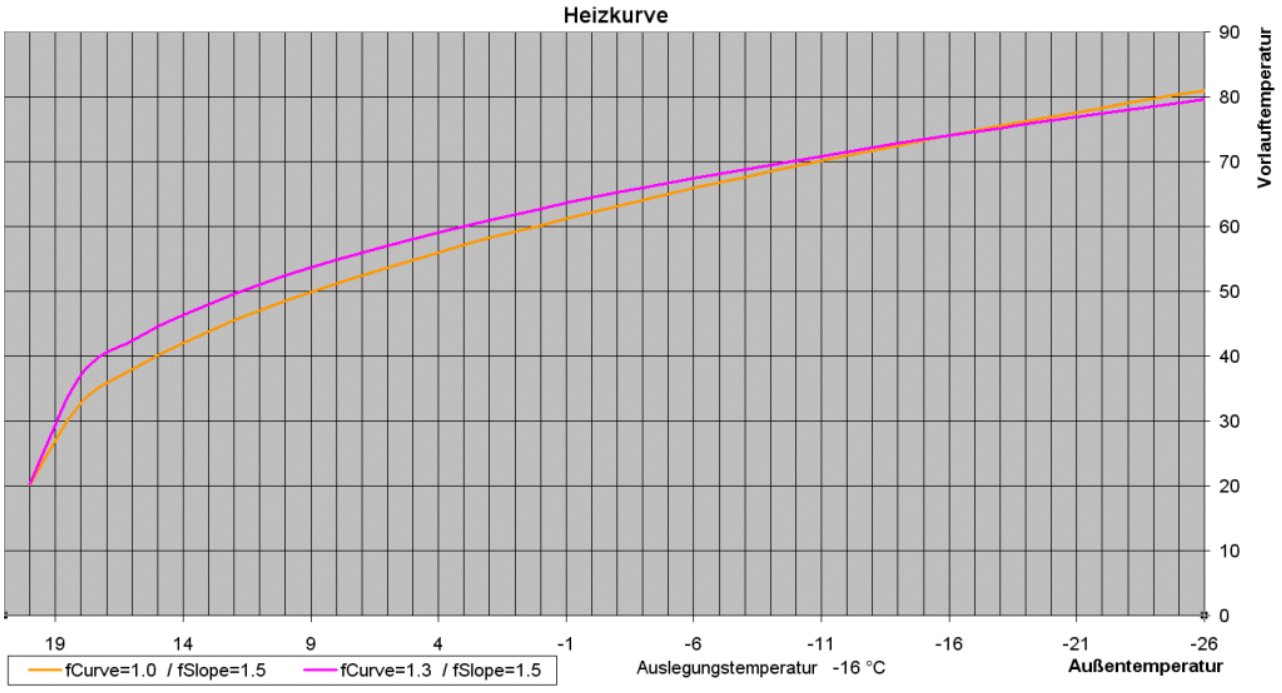
Vorlauftemperatur von 50 °C bei ca. -9.2 °C

Vorlauftemperatur von 50 °C bei ca. -9.4 °C



Vorlauftemperatur von 50 °C bei ca. -7.8 °C

Vorlauftemperatur von 50 °C bei ca. -8.3 °C



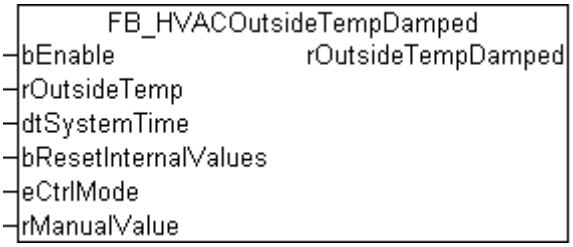
Vorlauftemperatur von 60 °C bei ca. 0.2 °C

Vorlauftemperatur von 60 °C bei ca. 3.0 °C

Dokumente hierzu

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

3.5.3 FB_HVACOutsideTempDamped



Anwendung

Dieser Baustein dient zur Ermittlung der mittleren bzw. gedämpften Außentemperatur. Im Automatikbetrieb wird die mittlere Außentemperatur berechnet aus den Werten der Außentemperatur um 7:00 Uhr, 14:00 Uhr und 21:00 Uhr. Wobei der Wert um 21:00 Uhr doppelt berücksichtigt wird.

VAR_INPUT

```
bEnable             : BOOL;
rOutsideTemp        : REAL;
dtSystemTime        : DT;
bResetInternalValue : BOOL;
eCtrlMode           : E_HVACCtrlMode;
rManualValue        : REAL;
```

bEnable: Mit der Eingangsvariablen *bEnable*wird der Baustein vom SPS-Programm frei gegeben. Bei *bEnable* = FALSE wird der letzte gültige Wert der gedämpften Außentemperatur ausgegeben.

rOutsideTemp: Über diese Eingangsvariable wird dem Funktionsbaustein die aktuelle Außentemperatur übergeben.

dtSystemTime: Über diese Eingangsvariable wird dem Funktionsbaustein Datum und Uhrzeit übergeben.

bResetInternalValues: Über diese Eingangsvariable werden die internen gespeicherten Außentemperaturen zurückgesetzt und die momentan anliegende Außentemperatur wird dann übernommen.

eCtrlMode: Über dieses Enum wird der Betriebsmodus ausgewählt. Hand- oder Automatikbetrieb.

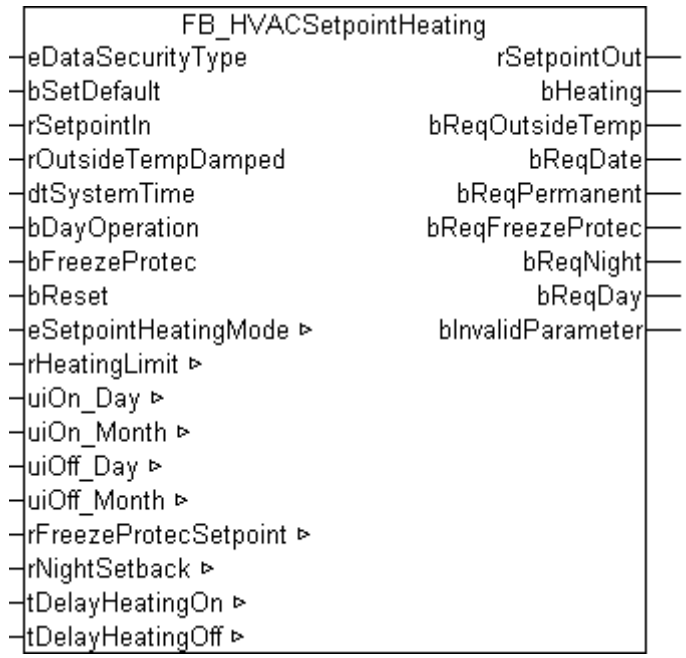
rManualValue: Manueller Wert, der bei *eCtrlMode* = *eHVACCtrlMode_Manual* an den Ausgang *rOutsideTempDamped* gesetzt wird.

VAR_OUTPUT

```
rOutsideTempDamped      : REAL;
```

rOutsideTempDamped: Gedämpfte Außentemperatur.

3.5.4 FB_HVACSetpointHeating



Anwendung

Mit diesem Funktionsbaustein kann ein Heizkreis in unterschiedliche Betriebsarten geschaltet werden. Je nach Betriebsart wird am Ausgang des Funktionsbausteins ein entsprechender Sollwert für die Vorlauftemperaturregelung eines statischen Heizkreises ausgegeben. In Abhängigkeit der Betriebsart und dem Zustand der nachfolgend beschriebenen Parameter, wird der Ausgang *bHeating* gesetzt bzw. zurückgesetzt.

Dieser Funktionsbaustein muss funktionsbedingt mit dem **FB_HVACHeatingCurve** zusammen verwendet werden. Der Grund dafür ist, dass der Betrag für die Nachtabsenkung beim **FB_HVACHeatingCurve** berücksichtigt wird.

VAR_INPUT

```
eDataSecurityType     : E_HVACDataSecurityType;
bSetDefault           : BOOL;
bEnable               : BOOL;
rSetpointIn           : REAL;
rOutsideTempDamped    : REAL;
dtSystemTime          : DT;
bDayOperation         : BOOL;
bFreezeProtec         : BOOL;
bReset                : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Ist die Variable *bEnable* TRUE, dann ist der Funktionsbaustein aktiviert. Bei einem FALSE wird an dem Ausgang *rSetpointOut* null ausgegeben und der Pumpenausgang bleibt FALSE.

rSetpointIn: Bei einem raumtemperaturgeführten Heizkreis wird an *rSetpointIn* der Wert eines Raumsollwertmoduls angelegt. Ist der Heizkreis ein außentemperaturgeführter kommt der Sollwert von der Heizkennlinie. Siehe FB_HVACHeatingCurve [▶ 332].

rOutsideTempDamped: Kurzfristige Temperaturschwankungen der Außentemperatur dürfen sich nicht ungefiltert auf den Sollwert der Vorlauftemperatur auswirken. Deshalb ist die Außentemperatur nicht direkt, sondern über einen Dämpfungsbaustein aufzuschalten. Siehe hierzu auch FB_HVACOutsideTempDamped. [▶ 338]

dtSystemtime: Mit dieser Variablen wird dem Funktionsbaustein die Rechnersystemzeit übergeben.

bDayOperation: An dieser Eingangsvariablen wird die Ausgangsvariable des Zeitschaltprogramms übergeben. Ist *bDayOperation* TRUE**,** dann ist der Heizkreis im Tagbetrieb. Die Nachtabsenkung der Vorlauftemperatur wird damit deaktiviert.

bFreezeProtec: Eingang, an den die Frostschutzmeldung angelegt wird.

bReset: Quittierungseingang bei einer Störung.

VAR_OUTPUT

```
rSetpointOut      : REAL;
bHeating          : BOOL;
bReqOutsideTemp   : BOOL;
bReqDate          : BOOL;
bReqPermanent     : BOOL;
bReqFreezeProtec  : BOOL;
bReqNight         : BOOL;
bReqDay           : BOOL;
bInvalidParameter : BOOL;
```

rSetpointOut: Sollwert für die Vorlauftemperaturregelung.

bHeating: In Abhängigkeit der eingestellten Betriebsart sowie den Ein- und Ausschaltverzögerungen wird der Ausgang *bHeating* sofort oder verzögert gesetzt (TRUE) bzw. zurückgesetzt (FALSE). Der Ausgang kann zur Freigabe des Reglers verwendet werden.

bReqOutsideTemp: Der Heizkreis befindet sich in der Betriebsart, Heizperiode nach der Außentemperatur.

bReqDate: Der Heizkreis befindet sich in der Betriebsart, Heizperiode nach Datum.

bReqPermanent: Der Heizkreis befindet sich in der Betriebsart, Heizkreis ist immer an.

bReqFreezeProtec: Der Heizkreis befindet sich in der Betriebsart - Frostschutzbetrieb.

bReqNight: Der Heizkreis befindet sich in der Betriebsart - Nachtbetrieb.

bReqDay: Der Heizkreis befindet sich in der Betriebsart - Tagbetrieb.

bInvalidParameter: Zeigt an, dass ein falscher Eingangsparameter anliegt. *bInvalidParameter* muss mit *bReset* quittiert werden.

VAR_IN_OUT

```
eSetpointHeatingMode  : E_HVACSetpointHeatingMode;
rHeatingLimit         : REAL;
uiOn_Day              : UINT;
uiOn_Month            : UINT;
uiOff_Day             : UINT;
```

```
uiOff_Month           : UINT;
rFreezeProtecSetpoint : REAL;
rNightSetback         : REAL;
tDelayHeatingOn       : TIME;
tDelayHeatingOff      : TIME;
```

eSetpointHeatingMode: Enum, das die Betriebsart des Heizkreises festlegt. Die Variable wird persistent gespeichert. Voreingestellt auf 1.

rHeatingLimit: Heizgrenze (-60°C..100°C). Notwendig für die Betriebsarten des Heizkreises; Heizperiode nach der Außentemperatur und Heizperiode nach Datum. Die Variable wird persistent gespeichert. Voreingestellt auf 19.

uiOn_Day: Einschalttag des Monats. Notwendig für die Betriebsart des Heizkreises; Heizperiode nach Datum. Die Variable wird persistent gespeichert. Voreingestellt auf 1.

uiOn_Month: Einschaltmonat des Jahres. Notwendig für die Betriebsart des Heizkreises; Heizperiode nach Datum. Die Variable wird persistent gespeichert. Voreingestellt auf 9.

uiOff_Day: Ausschalttag des Monats. Notwendig für die Betriebsart des Heizkreises; Heizperiode nach Datum. Die Variable wird persistent gespeichert. Voreingestellt auf 1.

uiOff_Month: Ausschaltmonat des Jahres. Notwendig für die Betriebsart des Heizkreises; Heizperiode nach Datum. Die Variable wird persistent gespeichert. Voreingestellt auf 5.

rFreezeProtecSetpoint: Sollwert für den Heizkreis bei Frostschutz (0°C..100°C). Die Variable wird persistent gespeichert. Voreingestellt auf 8.

rNightSetback: Mit *rNightSetback* wird der Betrag der Nachtabsenkung in °C angegeben (0°C..100°C). *rNightSetback* wird in der Betriebart *eHVACSetpointHeatingMode_OnNight* berücksichtigt. Die Variable wird persistent gespeichert. Voreingestellt auf 10.

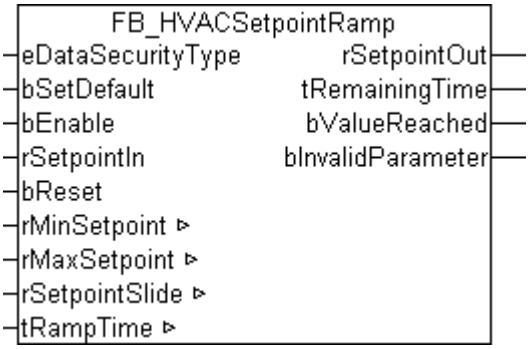
tDelayHeatingOn: Zeitverzögerung für das Setzen des Ausgangs *bHeating* auf TRUE, nachdem die gedämpfte Außentemperatur die Heizgrenze unterschritten hat oder das Datum erreicht wurde (0h..100h). Die Variable wird persistent gespeichert. Voreingestellt auf 0.

tDelayHeatingOff: Zeitverzögerung für das Zurücksetzen des Ausgangs *bHeating* auf FALSE, nachdem die gedämpfte Außentemperatur die Heizgrenze überschritten hat oder das Datum erreicht wurde (0h..100h). Die Variable wird persistent gespeichert. Voreingestellt auf 0.

Dokumente hierzu

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

3.5.5 FB_HVACSetpointRamp



Anwendung

Baustein, der eine gleitende Sollwertrampe erzeugt. Dieser Baustein muss in jedem SPS-Zyklus aufgerufen werden, weil die Berechnung für die Erreichung des Endsollwertes von der Task Time abhängig ist.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
bEnable          : BOOL;
rSetpointIn      : REAL;
bReset           : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Freigabe des Bausteins.

rSetpointIn: Sollwert, auf den der Ausgang *rSetpopintOut* in Abhängigkeit der absoluten Temperaturveränderung *rSetpointSlide* pro Zeitveränderung *tRampTime* geregelt wird. Jede Veränderung des Sollwertes startet eine neue Berechnung des Sollwertes auf den Zielwert der Rampe. Der Zielwert ist der aktuelle Wert von *rSetpointIn*. Der Startwert der Rampe ist letzter Wert vor der Änderung von *rSetpointIn* auf den neuen Zielwert, siehe Bild 1.1.
Mit den Parametern *rMaxSetpoint* und *rMinSetpoint* wird der Wertebereich von *rSetpointIn*festgelegt.
Liegt ein nicht korrekter Variablenwert an *rSetpointIn* an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit *rMinSetpoint* intern weiter gearbeitet. *bInvalidParameter* wird bei falschem Variablenwert gesetzt, der Funktionsbaustein arbeitet normal weiter.

bReset: Quittierungseingang bei einer Störung.

VAR_OUTPUT

```
rSetpointOut         : REAL;
tRemainingTime       : TIME;
bValueReached        : BOOL;
bInvalidParameter    : BOOL;
```

rSetpointOut: Ausgang des Rampengenerators

tRemainingTime: Zeitdauer, in der der Ausgang *rSetpointOut* den Zielwert *rSetpointIn* erreicht hat.

bValueReached: Ein TRUE an diesem Ausgang signalisiert, dass der Ausgang *rSetpointOut* den Zielwert *rSetpointIn* erreicht hat.

bInvalidParameter: Zeigt an, dass ein falscher Parameter an einer der Variablen *rSetpointSlide* oder *tRampTime* anliegt. Eine falsche Parameterangabe führt nicht zum Stillstand des Bausteines, siehe Beschreibung der Variablen. Nach Behebung der falschen Parameterangabe muss die Meldung *bInvalidParameter* mit *bReset* quittiert werden.

VAR_IN_OUT

```
rMinSetpoint     : REAL;
rMaxSetpoint     : REAL;
rSetpointSlide   : REAL;
tRampTime        : TIME;
```

rMinSetpoint/rMaxSetpoint: Mit den Parametern *rMaxSetpoint* und *rMinSetpoint* wird der Wertebereich (-10000..10000) von *rSetpointIn*festgelegt. *rMaxSetpoint* muss größer als *rMinSetpoint* sein.

Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt, der Funktionsbaustein arbeitet normal weiter. Die Variable wird persistent gespeichert. Voreingestellt auf 10 bzw. 40.

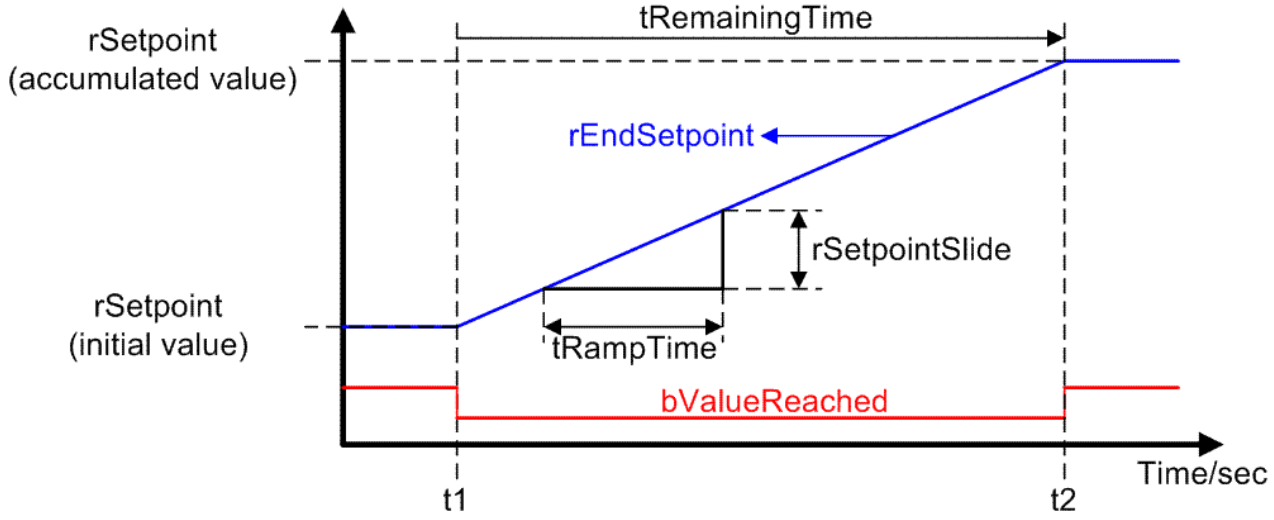
rSetpointSlide: Absolute Temperaturveränderung in [Kelvin/*tRampTime*] mit der der Ausgang von einem niedrigeren bzw. höheren Wert auf einen höheren bzw. niedrigeren Wert schrittweise gleitend überführt wird, siehe Bild 1.1

Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt, der Funktionsbaustein arbeitet normal weiter. Die Variable wird persistent gespeichert. Voreingestellt auf 1.

tRampTime: Rampenzeit (1s..24h), in der der Endsollwert gleitend in Abhängigkeit der absoluten Temperaturveränderung *rSetpointSlide* schrittweise erreicht wird, siehe Bild 1.1

Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden, der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt, der Funktionsbaustein arbeitet normal weiter. Die Variable wird persistent gespeichert. Voreingestellt auf 3600s.

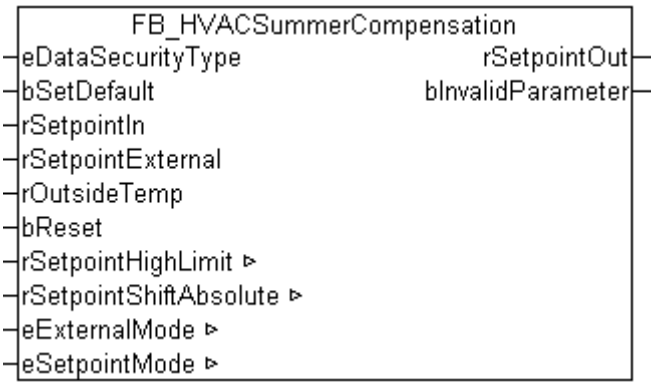
Bild 1.1:



Dokumente hierzu

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

3.5.6 FB_HVACSummerCompensation



Anwendung

Dieser Funktionsbaustein ist ein Sollwertgenerator für die Abluft- bzw. Raumtemperatur einer Raumlufttechnischen Anlage. Mit Hilfe dieser Sommerkompensation werden zu hohe Temperaturunterschiede zwischen der Außentemperatur und der Abluft-/Raumtemperatur im Sommer vermieden. Dieses dient der Energieeinsparung bei gleichzeitiger Komfortsteigerung.

Steigt die Außentemperatur so weit an, dass die Differenz zur Raumtemperatur größer als die eingestellte Differenz von *rSetpointShiftAbsolute* ist, wird der Raumtemperatursollwert (*rSetpointOut*) angehoben. Die zulässige Differenz ist einstellbar von 0 bis 10 Kelvin.

Es wird im Funktionsbaustein überprüft, dass der maximal eingestellte Raumtemperatursollwert nicht überschritten wird, wenn *eSetpointMode:= eHVACSetpointMode_DINLimited* ist.

Beispiel I:

Raumsollwertvorgabe (*rSetpointIn*):= 21 °C

externe Sollwertvorgabe (*rSetpointExternal*):= 2K (z.B. +/- 3K können über ein Poti eingestellt werden, werden aber nicht berücksichtigt, da*eExternalMode:= eHVACExternalMode_Off* ist)

eingestellte Differenzwert zw. Außentemperatur und Raumtemperatur (*rSetpointShiftAbsolute*):= 6 K

Außentemperatur (*rOutsideTemp*):= 28°C

eExternalMode:= e*HVACExternalMode_Off*

eSetpointMode:= *eHVACSetpointMode_DIN*

berechneter Raumtemperatursollwert (*rSetpointOut*):= 22

Beispiel II:

Raumsollwertvorgabe (*rSetpointIn*):= 20°C

externe Sollwertvorgabe (*rSetpointExternal*):= -2K (z.B. +/- 3K können über ein Poti eingestellt werden)

eingestellte Differenzwert zw. Außentemperatur und Raumtemperatur (*rSetpointShiftAbsolute*):= 6 K

Außentemperatur (*rOutsideTemp*):= 28°C

eExternalMode:= *eHVACExternalMode_On*

eSetpointMode:= *eHVACSetpointMode_DIN*

berechneter Raumtemperatursollwert (*rSetpointOut*):= 20

Beispiel III:

Raumsollwertvorgabe (*rSetpointIn*):= 23 °C

externe Sollwertvorgabe (*rSetpointExternal*):= 3K (z.B. +/- 3K können über ein Poti eingestellt werden)

eingestellte Differenzwert zw. Außentemperatur und Raumtemperatur (*rSetpointShiftAbsolute*):= 6 K

Außentemperatur (*rOutsideTemp*):= 34°C

rSetpointHighLimit:= 35°C

eExternalMode:= *eHVACExternalMode_On*

eSetpointMode:= *eHVACSetpointMode_DINLimited*

berechneter Raumtemperatursollwert (*rSetpointOut*):= 31

Beispiel IV:

Raumsollwertvorgabe (*rSetpointIn*):= 24 °C

externe Sollwertvorgabe (*rSetpointExternal*):= 3K (z.B. +/- 3K können über ein Poti eingestellt werden)

eingestellte Differenzwert zw. Außentemperatur und Raumtemperatur (*rSetpointShiftAbsolute*):= 6 K

Außentemperatur (*rOutsideTemp*):= 36°C

rSetpointHighLimit:= 30°C

eExternalMode:= *eHVACExternalMode_On*

eSetpointMode:= *eHVACSetpointMode_DINLimited*

berechneter Raumtemperatursollwert := 31°Cwird aber aufgrund von rSetpointHighLimit der auf 30°Ceingestellt ist auf 30°Cbegrenzt ==> *rSetpointOut*:=30 °C

VAR_INPUT

```
eDataSecurityType    : E_HVACDataSecurityType;
bSetDefault          : BOOL;
rSetpointIn          : REAL;
rSetpointExternal    : REAL;
rOutsideTemp         : REAL;
bReset               : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

rSetpointIn: An diesen Eingang wird der Sollwert für den Raum angelegt.

rSetpointExternal: An diesen Eingang wird die Sollwertverstellung oder -korrektur z.B. von einem Poti angelegt.

rOutsideTemp: Eingang für die Außentemperatur.

bReset: Quittierungseingang bei einer Störung.

VAR_OUTPUT

```
rSetpointOut          : REAL;
bInvalidParameter     : BOOL;
```

rSetpointOut: Berechneter Raumtemperatursollwert.

bInvalidParameter: Zeigt an, dass ein falscher Eingangsparameter anliegt. *bInvalidParameter* muss mit *bReset* quittiert werden.

VAR_IN_OUT

```
rSetpointHighLimit    : REAL;
rSetpointShiftAbsolute : REAL;
eExternalMode         : E_HVACExternalMode;
eSetpointMode         : E_HVACSetpointMode;
```

rSetpointHighLimit: Obere Grenze für den Raumtemperatursollwert (0°C..100°C). Die Variable wird persistent gespeichert. Voreingestellt auf 35.

rSetpointShiftAbsolute: Parameterwert, der die zulässige Abweichung zwischen Außentemperatur und Raumtemperatur definiert. Die zulässige Abweichung ist einstellbar von 0 bis 10 Kelvin. Ist die Abweichung zwischen Außentemperatur und Raumtemperatur größer als der eingestellte Parameterwert wird der Raumtemperatursollwert erhöht. Die Variable wird persistent gespeichert. Voreingestellt auf 6.

eExternalMode: Über das ENUM E_HVACExternalMode erfolgt die Aktivierung/Deaktivierung der externen Sollwertvorgabe.
eHVACExternalMode_Off, entspricht das der externe Sollwertvorgabe über Poti im Tableau deaktiviert ist.
eHVACExternaMode_On, entspricht das der externe Sollwertvorgabe aktiviert ist., z.B +/- 3 °C
eHVACExternalMode_ShiftAbsolut, entspricht externe Sollwertvorgabe absolut in °C

eSetpointMode: Enum, das die Art der Sollwertermittlung festlegt.

Diese Sollwertanhebung bei hohen Außentemperaturen kann unbegrenzt oder begrenzt nach DIN erfolgen. Die Auswahl erfolgt mit dem ENUM E_HVACSetpointMode. Neben diesen beiden Modi für die Sommerkompensation gibt es noch die Möglichkeit einen festen Sollwert vorzugeben.

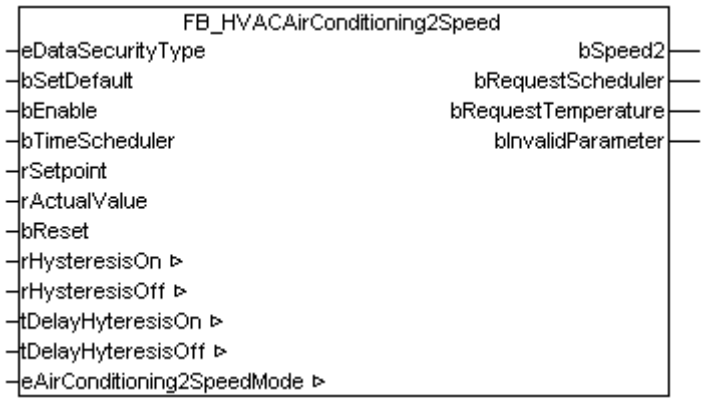
1. Nach DIN (*eHVACSetpointMode_DIN*). Der Raumtemperatursollwert folgt der Außentemperatur, über eine Differenz geführt, nach.
2. Begrenzt (*eHVACSetpointMode_DINLimited*). Der Raumtemperatursollwert folgt der Außentemperatur, über eine Differenz geführt, nach. Er wird jedoch begrenzt. Der Grenzwert ist *rSetpointHighLimit.*
3. Konstant (*eHVACSetpointMode_ConstantValueBase*). Der Raumtemperatursollwert wird als Konstantwert vorgegeben, eine Beeinflussung durch die Außentemperatur findet nicht statt.

Dokumente hierzu

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

3.6 HLK Sonderfunktionen

3.6.1 FB_HVACAirConditioning2Speed



Anwendung

Dieser Funktionsbaustein *FB_HVACAirConditioning2Speed* steuert das Schalten in die zweite Stufe von raumlufttechnischen Anlagen mit zweistufigen Ventilatoren. Das Schalten in die zweite Stufe über die Ausgangsvariable *bSpeed2* kann zum Einen von einem Zeitschaltplan über die Eingangsvariable *bTimeScheduler* erfolgen oder lastabhängig über eine Differenz des Raum- oder Ablufttemperatursollwertes *rSetpoint* und des Raum- oder Ablufttemperaturistwertes *rActualValue* in Abhängigkeit der Zeitverzögerungen *tDelayHysteresisOn*/*tDelayHysteresisOff* , der Ein- und Ausschalthysteresen *rHysteresisOn*/*rHysteresisOff* und der Betriebsarteinstellung über das Enum *eAirConditioning2SpeedMode*.

VAR_INPUT

```
eDataSecurityType    : E_HVACDataSecurityType;
bSetDefault          : BOOL;
bEnable              : BOOL;
bTimeScheduler       : BOOL;
rSetpoint            : REAL;
rActualValue         : REAL;
bReset               : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen. Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Mit der Eingangsvariable *bEnable* = TRUE wird der Baustein vom SPS-Programm frei gegeben. Ohne Freigabe ist die Ausgangsvariable *bSpeed2* = FALSE.

bTimeScheduler: Anforderung der zweiten Ventilatorstufe von einem Zeitschaltprogramm. Wenn *bEnable* und *bTimeScheduler* = TRUE, so sind *bSpeed2* und *bRequestScheduler* = TRUE.

rSetpoint: Raum- oder Ablufttemperatursollwert

rActualValue: Raum- oder Ablufttemperaturistwert

bReset: Eingang zur Quittierung einer falschen Parameterangabe oder einer Störung über eine steigende Flanke.

VAR_OUTPUT

```
bSpeed2            : BOOL;
bRequestScheduler  : BOOL;
bRequestTemperature: BOOL;
bInvalidParameter  : BOOL;
```

bSpeed2: Ausgang zur Ansteuerung der zweiten Ventilatorstufe.

bRequestScheduler: Anforderung der zweiten Ventilatorstufe über die Eingangsvariable *bTimeScheduler*.

bRequestTemperature: Anforderung der zweiten Stufe lastabhängig über eine Differenz des Raum- oder Ablufttemperatursollwertes *rSetpoint* und des Raum- oder Ablufttemperaturistwertes *rActualValue* in Abhängigkeit der Zeitverzögerungen *tDelayHysteresisOn*/*tDelayHysteresisOff* , der Ein- und Ausschalthysteresen *rHysteresisOn*/*rHysteresisOff* und der Moduseinstellung über das Enum *eAirConditioning2SpeedMode*.

bInvalidParameter: Zeigt an, dass ein falscher Parameter an einer der Variablen *rHysteresisOn, rHysteresisOff* und *eAirConditioning2SpeedMode* anliegt. Eine falsche Parameterangabe führt nicht zum Stillstand des Bausteines, siehe Beschreibung der Variablen. Nach Behebung der falschen Parameterangabe muss die Meldung *bInvalidParameter* mit *bReset* quittiert werden.

VAR_IN_OUT

```
rHysteresisOn              : REAL;
rHysteresisOff             : REAL;
tDelayHyteresisOn          : TIME;
tDelayHyteresisOff         : TIME;
eAirConditioning2SpeedMode : E_HVACAirConditioning2SpeedMode;
```

rHysteresisOn: Parameter zur Einschaltung der zweiten Ventilatorstufe über eine Hysterese lastabhängig über die Ausgangsvariable *bSpeed2* in Abhängigkeit des Enums *eAirConditioning2SpeedMode (0..1000)*. *rHysteresisOn* muss größer sein als *rHysteresisOff*, wenn die Betriebsart *eAirConditioning2SpeedMode*=*eHVACAirConditioning2SpeedModeHeatingAndCooling* ist.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden, der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt, der Funktionsbaustein arbeitet normal weiter. Die Variable wird persistent gespeichert. Voreingestellt auf 5.

rHysteresisOff: Parameter zur Ausschaltung der zweiten Ventilatorstufe über eine Hysterese lastabhängig über die Ausgangsvariable *bSpeed2* in Abhängigkeit des Enums *eAirConditioning2SpeedMode (0..1000)*. Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden, der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt, der Funktionsbaustein arbeitet normal weiter. Die Variable wird persistent gespeichert. Voreingestellt auf 1.

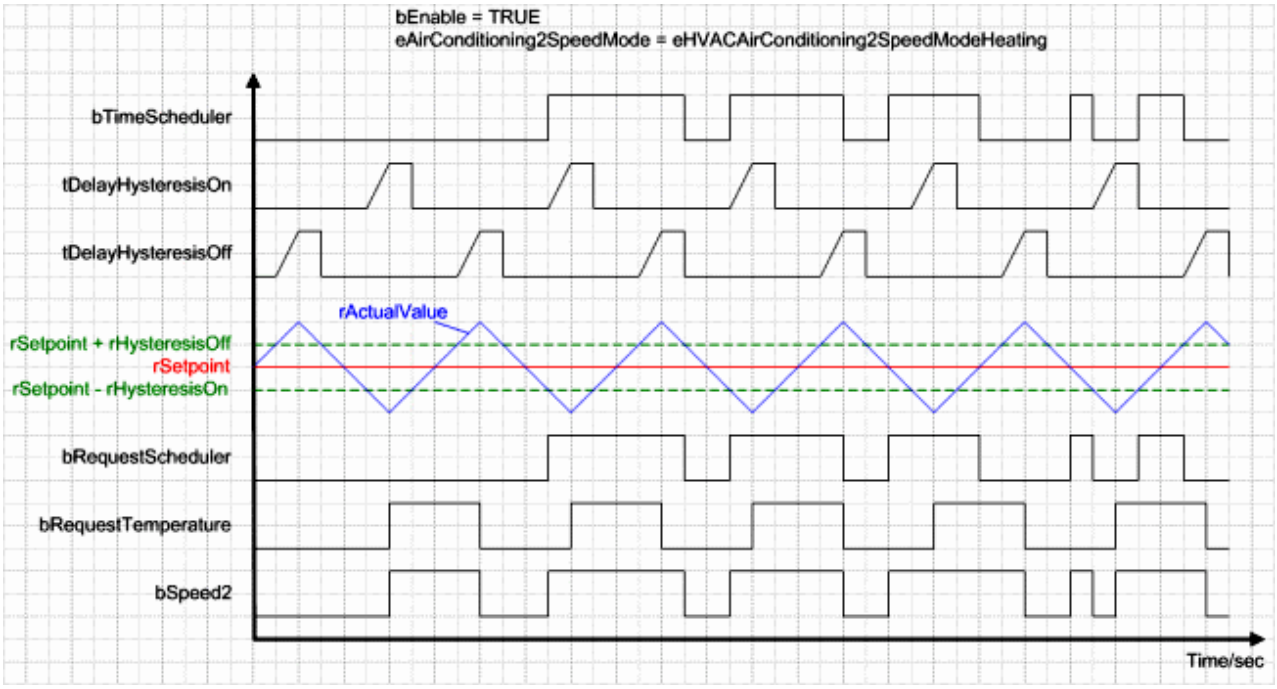
tDelayHysteresisOn: Einschaltverzögerung der Ausgangsvariable *bSpeed2* bei lastabhängiger Anforderung der zweiten Ventilatorstufe. Die Variable wird persistent gespeichert. Voreingestellt auf 300s.

tDelayHysteresisOff: Ausschaltverzögerung der Ausgangsvariable *bSpeed2* bei lastabhängiger Anforderung der zweiten Ventilatorstufe. Die Variable wird persistent gespeichert. Voreingestellt auf 300s.

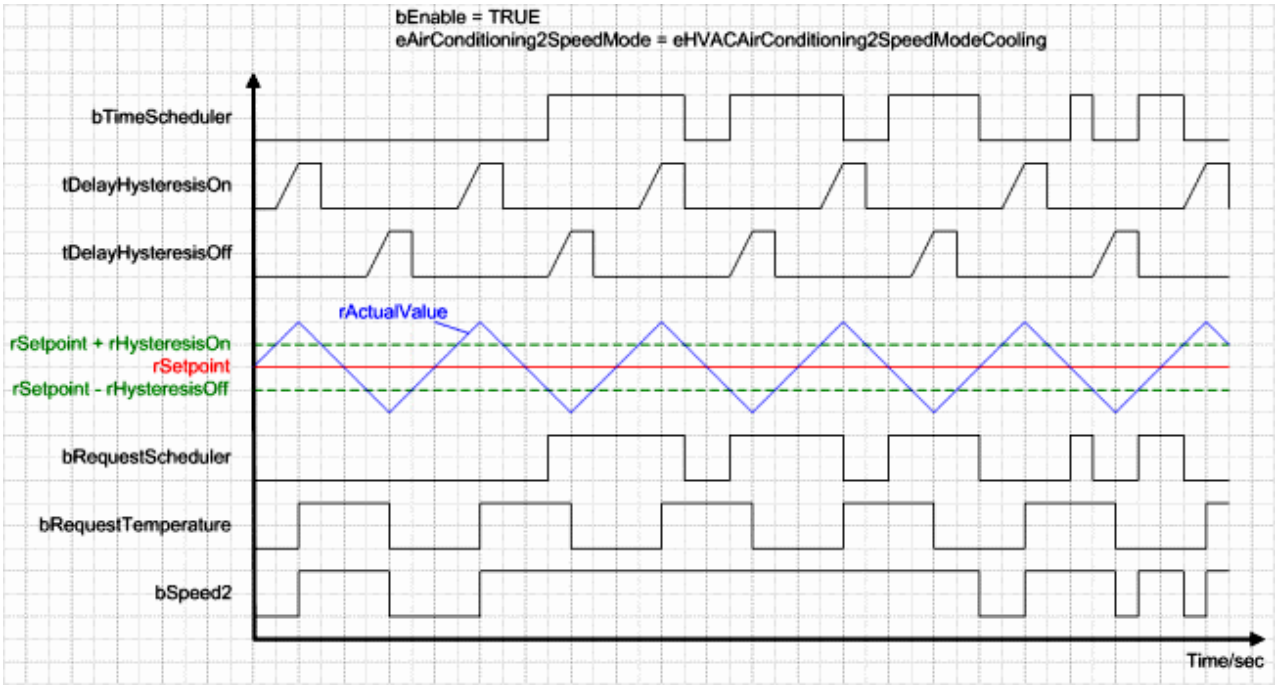
eAirConditioning2SpeedMode: Enum, über welches die lastabhängige Betriebsart für das Einschalten der zweiten Ventilatorstufe über die Ausgangsvariable *bSpeed2* vorgegeben wird.
eAirConditioning2SpeedMode = *eHVACAirConditioning2SpeedMode_Off*: Betriebsart aus
eAirConditioning2SpeedMode = *eHVACAirConditioning2SpeedMode_Heating*: Betriebsart Heizbetrieb. Es

handelt sich hier um eine raumlufttechnische Anlage mit reinem Heizbetrieb.
eAirConditioning2SpeedMode = *eHVACAirConditioning2SpeedMode_Cooling*: Betriebsart Kühlbetrieb. Es handelt sich hier um eine raumlufttechnische Anlage mit reinem Kühlbetrieb.
eAirConditioning2SpeedMode = *eHVACAirConditioning2SpeedMode_HeatingAndCooling*: Betriebsart Heiz- und Kühlbetrieb. Es handelt sich hier um eine raumlufttechnische Anlage mit Heiz- und Kühlbetrieb. Die Variable wird persistent gespeichert. Voreingestellt auf 1.

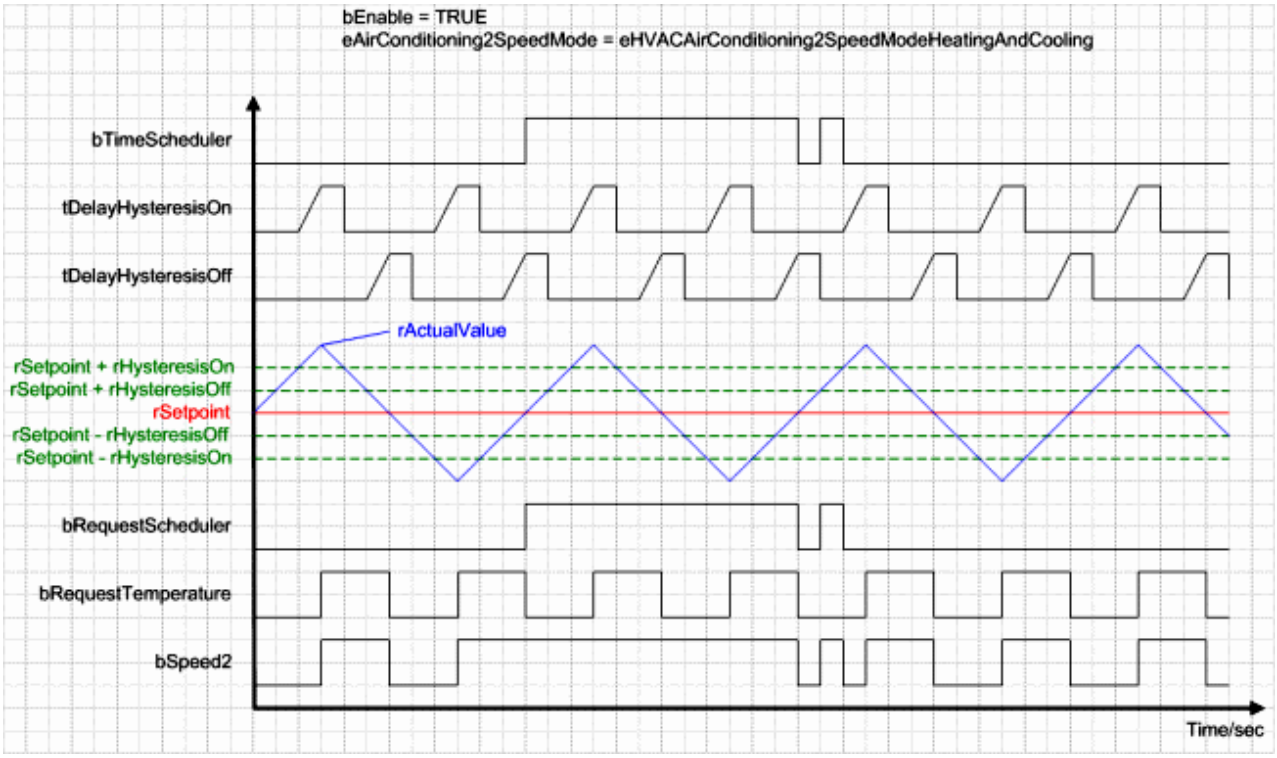
Betriebsart eAirConditioning2SpeedMode = eHVACAirConditioning2SpeedMode_Heating



Betriebsart eAirConditioning2SpeedMode = eHVACAirConditioning2SpeedMode_Cooling



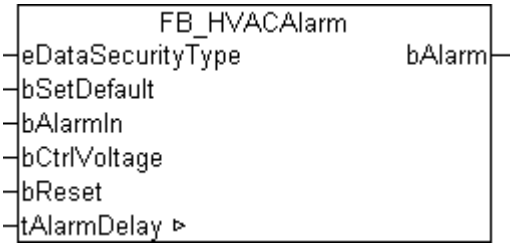
Betriebsart eAirConditioning2SpeedMode = eHVACAirConditioning2SpeedMode_HeatingAndCooling



Dokumente hierzu

- example_persistent_e.zip (Resources/zip/11659714827.zip)
- fb_hvacairconditioning2speeddrawingheating.gif (Resources/gif/11659721995.gif)
- fb_hvacairconditioning2speeddrawingcooling.gif (Resources/gif/11659723403.gif)
- fb_hvacairconditioning2speeddrawingheatingandcooling.gif (Resources/gif/11659724811.gif)

3.6.2 FB_HVACAlarm



Anwendung

Ein TRUE an dem Eingang *bAlarmIn* signalisiert, dass ein Alarm ansteht. Die Weitergabe des Alarmes an den Ausgang *bAlarm* erfolgt erst, wenn *bAlarmIn* länger als die eingestellete Zeit *tAlarmDelay* dauernd ansteht. Desweiteren muss der Eingang *bCtrlVoltage* TRUE sein, damit ein Alarm gemeldet wird. Der Ausgang *bAlarm* bleibt solange auf TRUE gesetzt bis *bAlarmIn* = FALSE ist und der Alarm mit einer positiven Flanke an *bReset* quittiert wurde.

VAR_INPUT

```
eDataSecurityType      : E_HVACDataSecurityType;
bSetDefault        : BOOL;
bAlarmIn           : BOOL;
bCtrlVoltage           : BOOL;
bReset             : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bAlarmIn: Ein TRUE an dem Eingang *bAlarmIn* signalisiert, dass ein Alarm ansteht.

bCtrlVoltage: Über den Eingang *bCtrlVoltage* wird überprüft, ob die Steuerspannung anliegt.

bReset: Quittierungseingang.

VAR_OUTPUT

```
bAlarm            : BOOL;
```

bAlarm: TRUE, wenn ein Alarm gemeldet wird.

VAR_IN_OUT

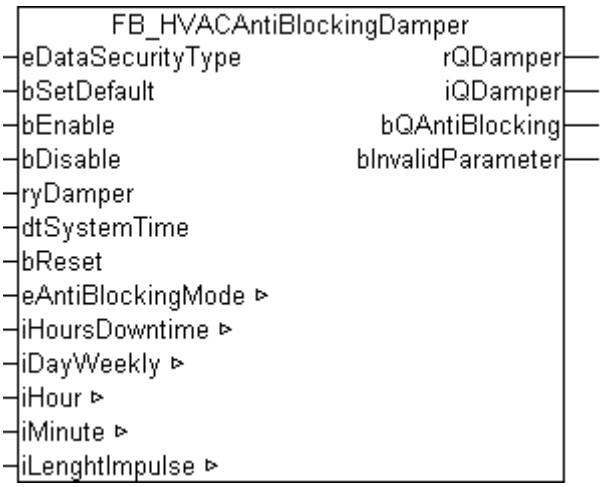
```
tAlarmDelay    : TIME;
```

tAlarmDelay: Zeitverzögerung nach der der Alarm erst gemeldet wird, wenn während der eingestellten Zeit dauernd *bAlarmIn* anstand (0s..500s). Die Variable wird persistent gespeichert. Voreingestellt auf 500s..

Dokumente hierzu

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

3.6.3 FB_HVACAntiBlockingDamper



Anwendung

Dieser Funktionsbaustein verhindert das Blockieren eines Stellantriebs über längere Zeitspannen mit unveränderter Stellgröße. Bei aktivem Blockierschutz wird ein Durchlauf des Stellantriebs von ganz Zu nach ganz Auf erzwungen.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
bDisable          : BOOL;
ryDamper          : REAL;            0 .. 100     %
dtSystemTime      : DT;
bReset            : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Mit der Eingangsvariablen *bEnable*wird die Funktion *AntiBlocking* frei gegeben. Ist *bEnable* = FALSE, so wird nur der Eingangswert an den Ausgang gelegt, die Funktion *AntiBlocking* ist nicht aktiv.

bDisable: Setzt die Ausgänge *rQDamper*, *iQDamper* und *bQAntiBlocking* auf 0 zurück.

ryDamper: Stellgröße vom Regler an das Ventil, die auf den Ausgang *rQDamper* und *iQDamper* geleitet wird. Der Wert muss kleiner 1% sein, damit der Baustein dies als Stillstandszeit erkennt. Für ein Ventil mit einer Stellgröße >= 1% wird keine Stillstandszeit ermittelt.

dtSystemTime: Systemzeit

bReset: Quittierungseingang bei einer Störung. Setzt den Merker *bInvalidParameter* zurück.

VAR_IN_OUT

```
eAntiBlockingMode : E_HVACAntiBlockingMode;
iHoursDowntime    : INT;
iDayWeekly        : INT;
iHour             : INT;
iMinute           : INT;
iLengthImpulse    : INT;
```

eHVACAntiBlockingMode: Enumerationswert, gibt die Art der *Antiblocking*-Methode an.

eHVACAntiBlockingMode_Off:= 0 : Aus

eHVACAntiBlockingMode_Downtime:= 1 : Nach Ablauf einer Stillstandszeit in Stunden, wird der *Antiblocking*-Impuls auf den Ausgang *bOut* geleitet.

eHVACAntiBlockingMode_Weekly:= 2 : Der Antiblocking-Impuls wird nur an einem bestimmtem Wochentag zu einer bestimmten Uhrzeit gebildet, unabhängig wie lange die Stillstandszeit dauerte.

Die Variable wird persistent gespeichert.

iHoursDowntime: Für *eHVACAntiBlockingMode_Downtime*, Zeit in Stunden, die der Eingang *bIn* nicht aktiv sein darf bis der *AntiBlocking*-Impuls gebildet wird (0h..6000h). Die Variable wird persistent gespeichert. Voreingestellt auf 24.

iDayWeekly: Für *eHVACAntiBlockingMode_Weekly,AntiBlocking*-Impuls wird an diesem Tag gebildet. So=0, Mo=1, Di=2, Mi=3, Do=4, Fr=5, Sa=6

Die Variable wird persistent gespeichert. Voreingestellt auf 6.

iHour: Einschaltzeitpunkt in Stunden (0h..23h). Die Variable wird persistent gespeichert. Voreingestellt auf 12.

iMinute: Einschaltzeitpunkt in Minuten (0min..59min). Die Variable wird persistent gespeichert. Voreingestellt auf 0.

iLengthImpulse: Einschaltdauer in Sekunden (0s..600s). Die Variable wird persistent gespeichert. Voreingestellt auf 150.

VAR_OUTPUT

```
rQDamper          : REAL;         0 .. 100        %
iQDamper          : INT;          0 .. 32767
bQAntiBlocking    : BOOL;
bInvalidParameter: BOOL;
```

rQDamper: Dieser Ausgang entspricht dem Eingangswert der Stellgröße. Sobald der Eingang *bEnable* =TRUE ist, wird zusätzlich der *Antiblocking*-Impuls auf den Ausgang gelegt. Wenn der Zustand *Antiblocking* erreicht ist, steht am Ausgang 100% an.

iQDamper: Dieser Ausgang verhält sich wie *rQDamper*, wird jedoch passend für einen Analogausgang als Integerwert von 0 - 32767 ausgegeben, entsprechend 0 - 100%.

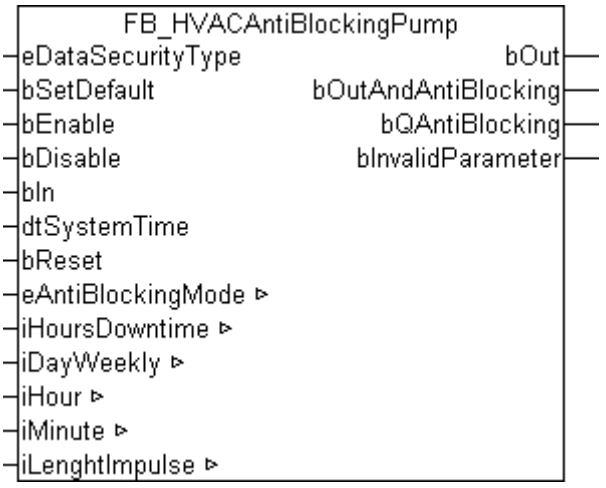
bQAntiBlocking: Dieser Ausgang ist unabhängig vom Einschaltsignal und zeigt nur den *Antiblocking-Impuls*.

bInvalidParameter: Bei der Plausibilitätsüberprüfung ist ein Fehler aufgetreten. Wir durch *bReset* wieder gelöscht.

Dokumente hierzu

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

3.6.4 FB_HVACAntiBlockingPump



Anwendung

Dieser Funktionsbaustein leitet die Einschaltbedingung für einen Pumpenantrieb weiter und kann je nach Modus und Ausschaltzeit einen zusätzlichen Einschaltimpuls erzeugen, um z.B. das Blockieren einer Pumpe zu verhindern.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
bEnable          : BOOL;
bDisable         : BOOL;
bIn              : BOOL;
dtSystemTime     : DT;
bReset           : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Mit der Eingangsvariablen *bEnable*wird die Funktion *AntiBlocking* frei gegeben. Ist *bEnable* = FALSE, so wird nur die Einschaltbedingung an den Ausgang gelegt, die Funktion *AntiBlocking* ist nicht aktiv.

bDisable: Setzt die Ausgänge *bOut* , *bOutAndAntiBlocking* und *bQAntiBlocking* auf 0 zurück. (siehe Beispiel unten)

bIn: Einschaltbedingung, die auf den Ausgang *bOut* geleitet wird.

dtSystemTime: Systemzeit

bReset: Quittierungseingang bei einer Störung. Setzt den Merker *bInvalidParameter* zurück.

VAR_IN_OUT

eAntiBlockingMode: E_HVACAntiBlockingMode;
iHoursDowntime : INT;
iDayWeekly : INT;
iHour : INT;
iMinute : INT;
iLengthImpulse : INT;

eHVACAntiBlockingMode: Enumerationswert, gibt die Art der *Antiblocking*-Methode an.

eHVACAntiBlockingMode_Off:= 0 : Aus

eHVACAntiBlockingMode_Downtime:= 1 : Nach Ablauf einer Stillstandszeit in Stunden, wird der *Antiblocking*-Impuls auf den Ausgang *bOut* geleitet.

eHVACAntiBlockingMode_Weekly:= 2 : Der Antiblocking-Impuls wird nur an einem bestimmtem Wochentag zu einer bestimmten Uhrzeit gebildet, unabhängig wie lange die Stillstandszeit dauerte.

Die Variable wird persistent gespeichert.

iHoursDowntime: Für *eHVACAntiBlockingMode_Downtime*, Zeit in Stunden, die der Eingang *bIn* nicht aktiv sein darf bis der *AntiBlocking*-Impuls gebildet wird(0h..6000h). Die Variable wird persistent gespeichert. Voreingestellt auf 24.

iDayWeekly: Für *eHVACAntiBlockingMode_Weekly,AntiBlocking*-Impuls wird an diesem Tag gebildet. So=0, Mo=1, Di=2, Mi=3, Do=4, Fr=5, Sa=6

Die Variable wird persistent gespeichert. Voreingestellt auf 6.

iHour: Einschaltzeitpunkt in Stunden (0h..23h). Die Variable wird persistent gespeichert. Voreingestellt auf 12.

iMinute: Einschaltzeitpunkt in Minuten (0min..23min). Die Variable wird persistent gespeichert. Voreingestellt auf 0.

iLengthImpulse: Einschaltdauer in Sekunden (0s..600s). Die Variable wird persistent gespeichert. Voreingestellt auf 150.

VAR_OUTPUT

```
bOut               : BOOL;
bOutAndAntiBlocking: BOOL;
bQAntiBlocking     : BOOL;
bInvalidParameter  : BOOL;
```

bOut: Wenn TRUE, dann kommt die Anforderung vom Pumpenbaustein. Dieser Ausgang entspricht der Einschaltbedingung unabhängig vom Zustand des Antiblocking-Impuls und des Eingangs *bEnable*.

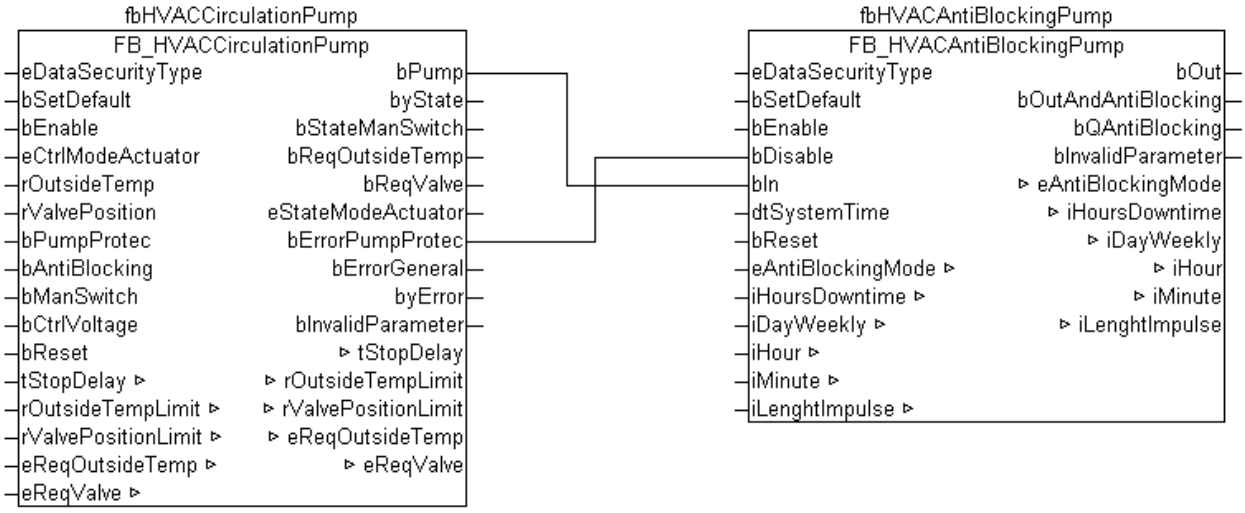
bOutAndAntiBlocking: Der Ausgang kann TRUE werden, wenn die Anforderung vom Pumpenbaustein ODER die Anforderung von der Antiblockierfunktion kommt. An diesen Ausgang wird die Pumpe angeschlossen.
Dieser Ausgang entspricht der Einschaltbedingung und ist abhängig vom Zustand des *Antiblocking-Impuls* und der Eingänge *bEnable* und *bDisable*. Ist*bEnable* = FALSE entspricht der Ausgang dem Ausgang *bOut*. Ist *bEnable* = TRUE, entspricht der Ausgang dem Ausgang *bOut* und ist zusätzlich oder verknüpft mit dem Antiblocking-Impuls.

bQAntiBlocking: Wenn TRUE, dann kommt die Anforderung von der Antiblockierfunktion. Dieser Ausgang ist unabhängig vom Einschaltsignal und zeigt nur den *Antiblocking-Impuls*.

bInvalidParameter: Bei der Plausibilitätsüberprüfung ist ein Fehler aufgetreten. Wir durch *bReset* wieder gelöscht.

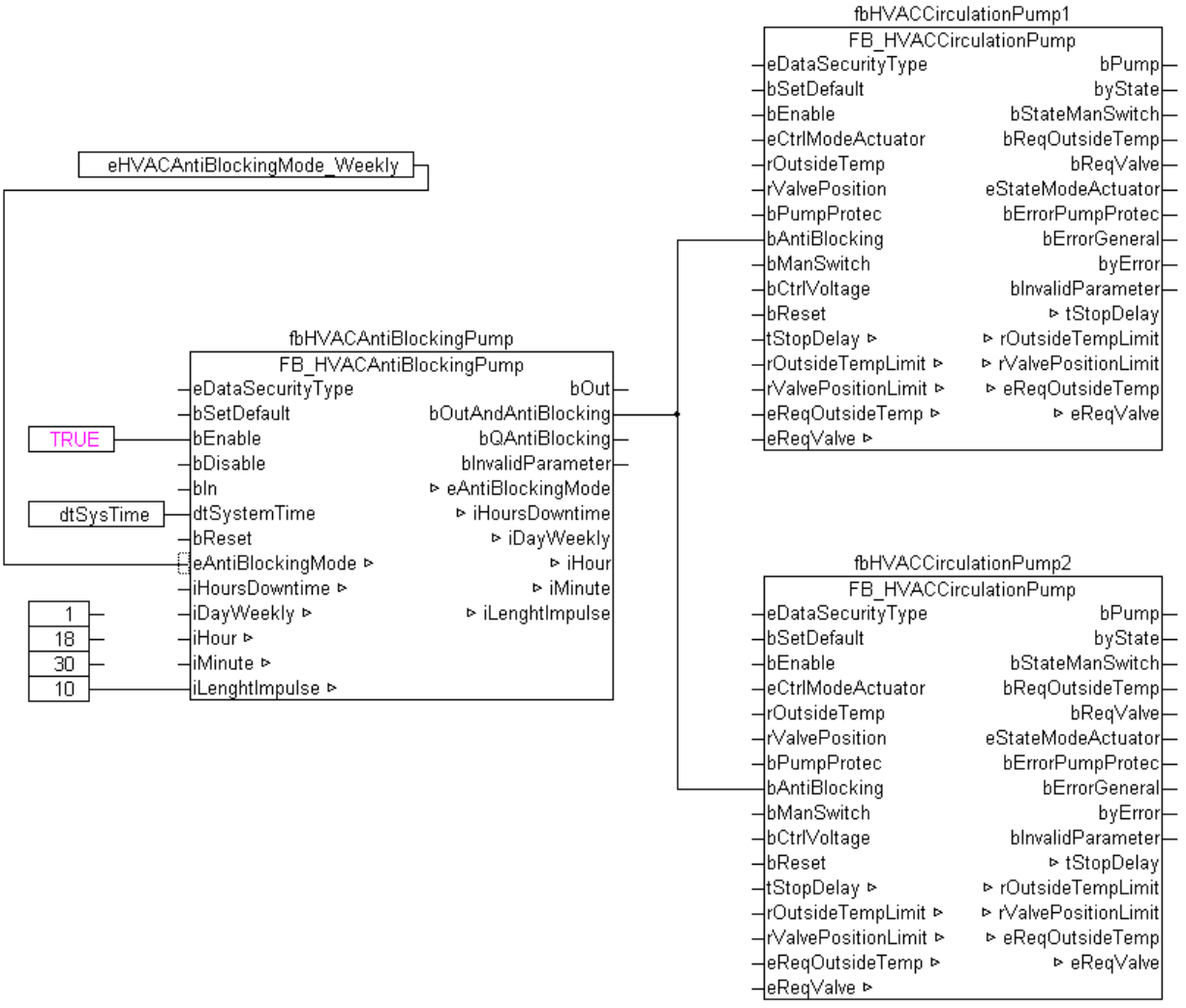
Beispiel einer Kombination von einem Pumpenbaustein und dem FB_HVACAntiBlocking:

- Der Pumpenausgang *bPump* wird auch ohne *bEnable* an den *bIn* des **FB_HVACAntiBlocking** gelegt.
- Geht der Eingang *bPumpProtec* auf FALSE, wird *bPump* abgeschaltet und über die Meldung *bErrorPumpProtec* wird *bDisable* aktiv, so dass *bOut* abgeschaltet wird.



Beispiel für das Steuern mehrerer Pumpen über eine AntiBlocking Instanz:

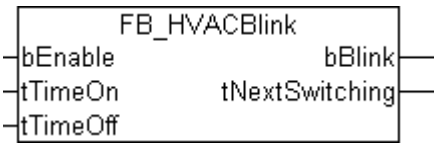
- fbHVACAntiBlocking über *bEnable* freigeben.
- Weekly-Mode auswählen (z.B. Montag um 18:30 für 10sek)



Dokumente hierzu

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

3.6.5 FB_HVACBlink



Anwendung

Der Funktionsbaustein liefert am Ausgang *bBlink* eine Blinksequenz, die abhängig von den beiden einstellbaren Zeiten *tTimeOn* und *tTimeOff* ist.

VAR_INPUT

```
bEnable          :REAL;
tTimeOn          :TIME;
tTimeOff         :TIME;
```

bEnable: Freigabe des Funktionsbausteines.

tTimeOn: Einschaltdauer des Blinkimpuls, *bBlink* = TRUE.

tTimeOff: Ausschaltdauer des Blinkimpuls, *bBlink* = FALSE.

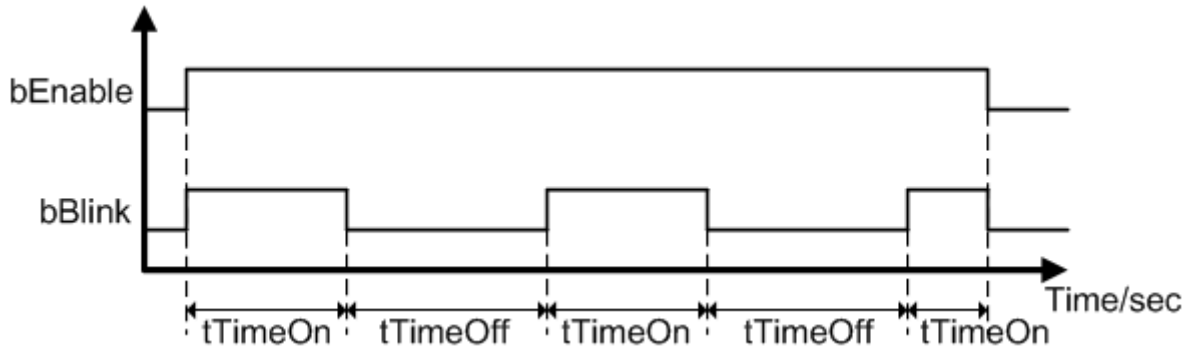
VAR_OUTPUT

```
bBlink           :BOOL;
tNextSwitching   :TIME;
```

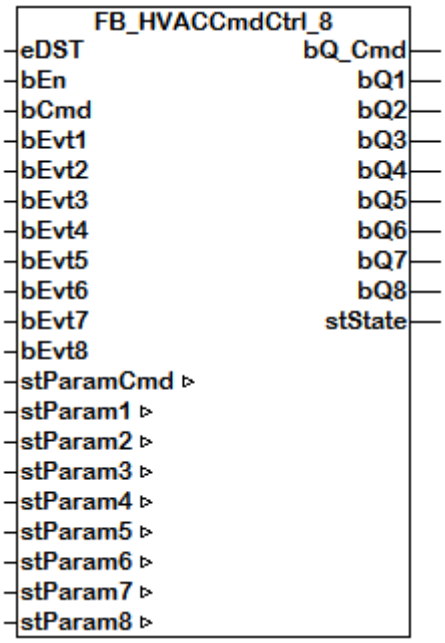
bBlink: Blinkausgang

tNextSwitching: Zeit bis zum nächsten Statuswechsel des Ausganges *bBlink* .

Verhalten der Ausgangsgröße



3.6.6 FB_HVACCmdCtrl_8



Anwendung

Mit dem Funktionsbaustein können einzelne Aggregate einer Anlage in einer bestimmten Reihenfolge sequentiell ein- (*bQ_Cmd* > *bQ1* > *bQ2* ... *bQ8*) bzw. ausgeschaltet (*bQ8* > *bQ7* > *bQ6* ... *bQ_Cmd*) werden. *FB_HVACCmdCtrl_8* kann als Startbaustein einer Lüftungsanlage eingesetzt werden.

Zu jedem Ausgang gehört ein Ereignis und eine Parameterstruktur mit Zeitvariablen. Über diese kann eine Ein- bzw. Ausschaltverzögerung sowie eine Mindestein- bzw. Mindestausschaltdauer des Ausgangs definiert werden.

Einschaltbedingung wenn und TRUE sind: *bEn bCmd*

Q(n) = Q(n - 1) AND Event(n) AND (DelayOn(n) = 0 AND (MinOff(n) = 0 AND*stState.udiStep* = n

Beispiel für den Ausgang bQ_Cmd

bQ_Cmd = *bCmd*AND*stState.udiSecRT_DelayOn* = 0 AND*stState.udiSecRT_MinOff[0]* = 0 AND*stState.udiStep* = 0

Beispiel für den Ausgang bQ2

bQ2 = *bQ1*AND*bEvt2*AND*stState.udiSecRT_DelayOn* = 0 AND*stState[2].udiSecRT_MinOff* = 0 AND*stState.udiStep* = 2

Ausschaltbedingung wenn TRUE und FALSE ist: *bEn bCmd*

Q(n) = NOT Q(n + 1) AND (DelayOff(n) = 0 AND (MinOn(n) = 0 AND*stState.udiStep* = n

Beispiel für den Ausgang bQ8

bQ8 = *stState.udiSecRT_DelayOff* = 0 AND*stState[7].udiSecRT_MinOn* = 0 AND*stState.udiStep* = 8

Beispiel für den Ausgang bQ_Cmd

bQ_Cmd = NOT*bQ1* AND *stState.udiSecRT_DelayOff* = 0 AND*stState[0].udiSecRT_MinOn* = 0 AND*stState.udiStep* = 0

Aktuelle Stufe

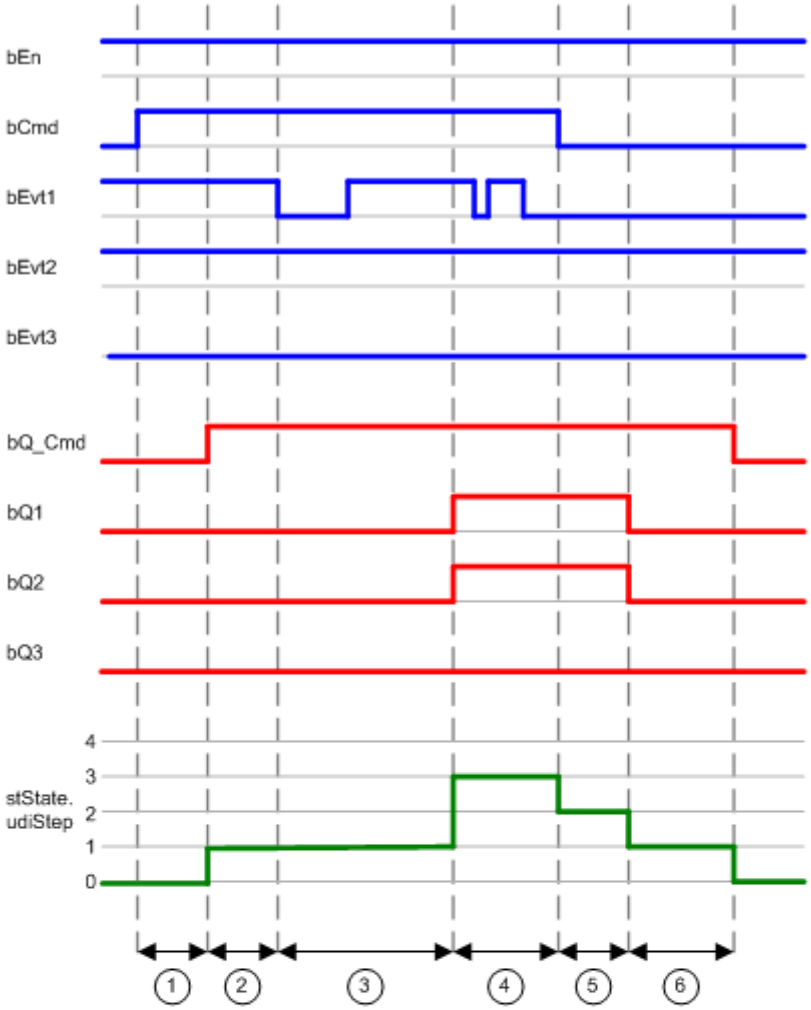
Anhand der Statusvariable *stState.udiStep* ist zu sehen in welcher Stufe sich der Funktionsbaustein befindet.

Ein- und Ausschaltfolge

Es wird von der Einschaltfolge (*bQ_Cmd* > *bQ1* > *bQ2* ... *bQ8*) gesprochen, wenn *bEn* und *bCmd* TRUE sind. Die Ausschaltfolge (*bQ8* > *bQ7* > *bQ6* ... *bQ_Cmd*) ist aktiv, wenn *bEn* = TRUE und *bCmd* = FALSE ist.

Ist ein Ausgang in der Einschaltfolge gesetzt worden, so bleibt dieser in Selbsthaltung und wird erst in der Ausschaltphase zurück gesetzt. In der Ausschaltfolge hat kein Ereignis (*bEvt1-8*) Einfluss auf die Ausschaltbedingungen der Ausgänge.

Befindet sich der Funktionsbaustein in der Ausschaltfolge und über *bCmd* = TRUE wird die Einschaltphase aktiviert, so startet der Funktionsbaustein in Stufe 0 mit den Einschaltbedingungen von *bQ_Cmd*. Für die anderen Ausgänge sind die Mindestein- bzw. Mindestausschaltzeiten aktiv bis zu derem Ablauf.



(1) Start der Einschaltfolge bCmd = TRUE. Nach Ablauf der Einschaltverzögerungszeit stState.udiSecRT_DelayOn wird bQ_Cmd TRUE. Der Status der Stufe (stState.udiStep) ändert seinen Wert von 0 nach 1.

(2) Das Ereignis bEvt1 ändert seinen Status von TRUE nach FALSE. An dem Status von bQ1 ändert sich nichts, weil die eingestellte Einschaltverzögerungszeit der Stufe 1 (stParam1.udiSecDelayOn) noch nicht abgelaufen ist.

(3) Das Ereignis bEvt1 ändert seinen Status von FALSE nach TRUE. Nach Ablauf der Einschaltverzögerungszeit stState.udiSecRT_DelayOn wird bQ1 = TRUE. Der Status der Stufe (stState.udiStep) ändert seinen Wert von 1 nach 2.

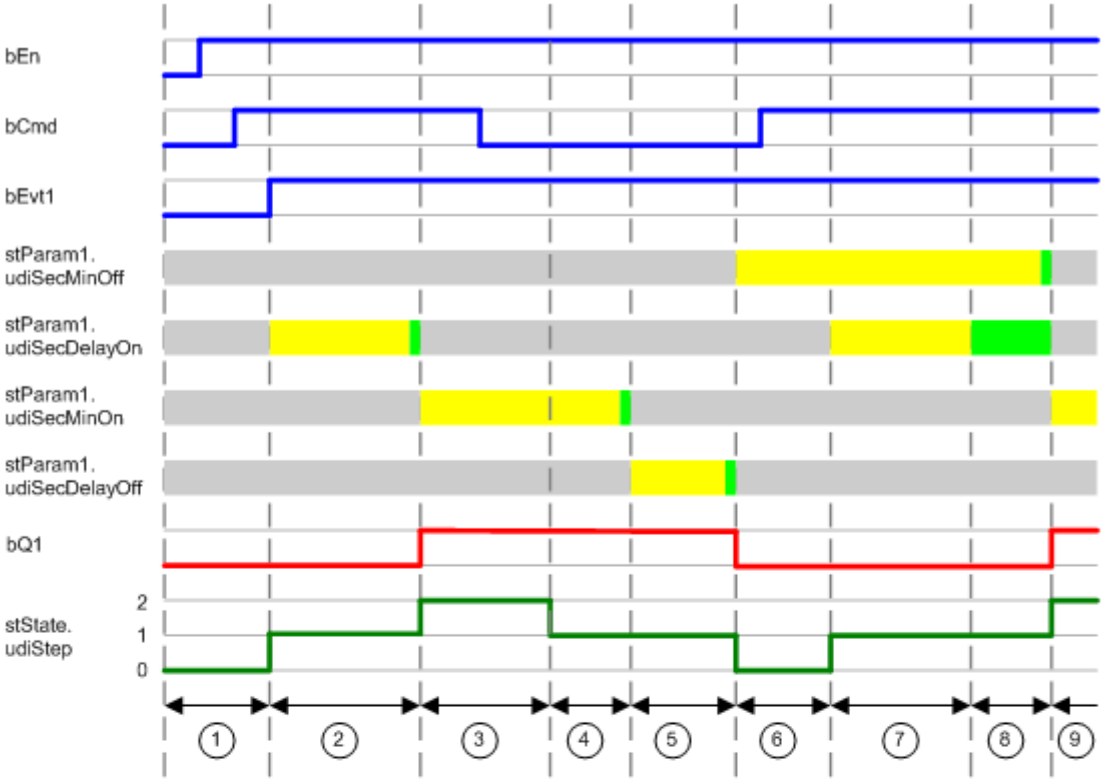
(4) Da das Ereignis bEvt2 vor Freigabe der Stufe 2 TRUE war und für die Stufe 2 keine Einschaltverzögerungszeit eingestellt wurde, ändert sich sofort der Status von bQ2 auf TRUE. Im Zuge dessen ändert sich der Wert der Stufe (stState.udiStep) von 2 nach 3.

(5) Start der Ausschaltfolge bCmd = FALSE. Der Status der Stufe (stState.udiStep) ändert seinen Wert direkt von 3 nach 2, weil in Stufe 3 der Ausgang bQ3 nicht gesetzt wurde.
Nach Ablauf der Auschaltverzögerungszeit stState.udiSecRT_DelayOff ändert sich der Status von bQ2 nach FALSE und der Wert der Stufe von 2 nach 1.

(6) Nach Ablauf der Ausschaltverzögerungszeit stState.udiSecRT_DelayOff ändert sich der Wert der Stufe (stState.udiStep) von 1 nach 0.

Zeitverläufe einer einzelnen Stufe

In dem Diagramm sind die Zeitverläufe der Stufe 1 bzw. des Ausgangs *bQ1* dargestellt.



Zeitstatus	
Steht	
Aktiv	
Abgelaufen	

(1) Freigabe des Funktionsbaustein mit bEn = TRUE. Start der Einschaltfolge bCmd = TRUE.

(2) Der Status der Stufe (stState.udiStep) ändert sich von 0 nach 1. Dadurch wird die Einschaltverzögerungszeit der Stufe 1 (stParam1.udiSecDelayOn) aktiviert.

(3) Die Einschaltverzögerung der Stufe 1 ist abgelaufen. Da die Mindestausschaltzeit nicht aktiv ist und das Ereignis bEvt1 TRUE ist, wird der Ausgang bQ1 TRUE. Die Mindesteinschaltzeit (stParam1.udiSecDelayOn) des Ausgangs bQ1 wird aktiviert. Der Status der Stufe (stState.udiStep) ändert sich von 1 nach 2.

(4) Die Stufe 1 (stState.udiStep) ist wieder aktiv.

(5) Nach Ablauf der Mindesteinschaltzeit (stParam1.udiSecDelayOn) wird die Ausschaltverzögerung (stParam1.udiSecDelayOff) der Stufe 1 aktiviert.

(6) Nach Ablauf der Ausschaltverzögerungszeit (stParam1.udiSecDelayOff) ändert sich die Status von bQ1 auf FALSE und der Stufe (stSate.udiStep) auf 0. Die Mindestausschaltzeit (stParam1.udiSecDelayOff) des Ausgangs bQ1 wird aktiviert.

(7) Der Status der Stufe (stSate.udiStep) ändert sich von 0 nach 1.Die Einschaltverzögerungszeit der Stufe 1 (stParam1.udiSecDelayOn) wird aktiviert.

(8) Die Einschaltverzögerungszeit der Stufe 1 (stParam1.udiSecDelayOn) ist abgelaufen. Der Ausgang bQ1 wird nicht eingeschaltet, weil die Mindestausschaltzeit (stParam1.udiSecMinOff) noch nicht abgelaufen ist.

(9) Der Ausgang bQ1 wird eingeschaltet, weil die Mindestausschaltzeit (stParam1.udiSecMinOff) abgelaufen ist. Der Status der Stufe (stState.udiStep) ändert sich von 1 nach 2.

Anwendungsbeispiel

Das Anwendungsbeispiel zeigt den Funktionsbaustein *FB_HVACCmdCtrl_8* in Verbindung mit dem *FB_HVACPriority_INT_8* als Anlagenstartprogramm einer Lüftungsanlage. Das Beispiel **AC03_StartAC.PRG** ist im Ordner **Language CFC** > **SpecialFunctions > Start** zu finden.

Download	Benötige Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

```
eDST              : E_HVACDataSecurityType;
bEn               : BOOL;
bCmd              : BOOL;
bEvt1             : BOOL;
bEvt2             : BOOL;
bEvt3             : BOOL;
bEvt4             : BOOL;
bEvt5             : BOOL;
bEvt6             : BOOL;
bEvt7             : BOOL;
bEvt8             : BOOL;
```

eDST: Wenn eDST := *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDST:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDST:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bEn: Freigabe des Bausteins. Ist *bEn* FALSE, so ist der Funktionsbaustein gesperrt. Sämtliche Ausgänge werden FALSE und die Zeitglieder für die Ein- bzw. Ausschaltverzögerungen sowie Mindestein- bzw. Mindestausschaltzeiten werden zurück gesetzt. An dem Eingang *bEn* könnte eine Sammelstörmeldung oder ein Notausschalter zum sofortigen Abschalten einer Anlage angelegt werden.

bCmd: Mit *bCmd* wird die Ein- bzw. Ausschaltfolge des Funktionsbausteins festgelegt. Ist *bCmd* TRUE, so befindet sich der Funktionsbaustein in der Einschaltfolge der Ausgänge. Das Einschalten einer Anlage könnte z.B. von einem Zeitschaltprogramm kommen. Ist *bCmd* FALSE, so befindet sich der Funktionsbaustein in der Ausschaltfolge der Ausgänge.

bCmd zählt als eine der Einschaltbedingungen von *bQ_Cmd*.

bEvt1: Das Ereignis *bEvt1* zählt als eine der Einschaltbedingungen von *bQ1*. In der Einschaltphase von *bQ1* ist die Einschaltverzögerung des Ausgangs nur dann aktiv, wenn *bEvt1* TRUE ist.

Einschaltbedingung:

bQ1 = *bQ_Cmd*AND*bEvt1*AND*stState.udiSecRT_DelayOn* = 0 AND*stState.udiSecRT_MinOff[1]* = 0 AND*stState.udiStep* = 1

Auf die Ausschaltbedingung von *bQ1* hat das Ereignis *bEvt1* keinen Einfluss.

bEvt2: Das Ereignis *bEvt2* zählt als eine der Einschaltbedingungen von *bQ2*. In der Einschaltphase von *bQ2* ist die Einschaltverzögerung des Ausgangs nur dann aktiv, wenn *bEvt2* TRUE ist.

Einschaltbedingung:

bQ2 = *bQ1*AND*bEvt2*AND*stState.udiSecRT_DelayOn* = 0 AND*stState.udiSecRT_MinOff[2]* = 0 AND*stState.udiStep* = 2

Auf die Ausschaltbedingung von *bQ2* hat das Ereignis *bEvt2* keinen Einfluss.

bEvt3: Das Ereignis *bEvt3* zählt als eine der Einschaltbedingungen von *bQ3*. In der Einschaltphase von *bQ3* ist die Einschaltverzögerung des Ausgangs nur dann aktiv, wenn *bEvt3* TRUE ist.

Einschaltbedingung:

bQ3 = *bQ2*AND*bEvt3*AND*stState.udiSecRT_DelayOn* = 0 AND*stState.udiSecRT_MinOff[3]* = 0 AND*stState.udiStep* = 3

Auf die Ausschaltbedingung von *bQ3* hat das Ereignis *bEvt3* keinen Einfluss.

bEvt4: Das Ereignis *bEvt4* zählt als eine der Einschaltbedingungen von *bQ4*. In der Einschaltphase von *bQ4* ist die Einschaltverzögerung des Ausgangs nur dann aktiv, wenn *bEvt4* TRUE ist.

Einschaltbedingung:

bQ4 = *bQ3*AND*bEvt4*AND*stState.udiSecRT_DelayOn* = 0 AND*stState.udiSecRT_MinOff[4]* = 0 AND*stState.udiStep* = 4

Auf die Ausschaltbedingung von *bQ4* hat das Ereignis *bEvt4* keinen Einfluss.

bEvt5: Das Ereignis *bEvt5* zählt als eine der Einschaltbedingungen von *bQ5*. In der Einschaltphase von *bQ5* ist die Einschaltverzögerung des Ausgangs nur dann aktiv, wenn *bEvt5* TRUE ist.

Einschaltbedingung:

bQ5 = *bQ4*AND*bEvt5*AND*stState.udiSecRT_DelayOn* = 0 AND*stState.udiSecRT_MinOff[5]* = 0 AND*stState.udiStep* = 5

Auf die Ausschaltbedingung von *bQ5* hat das Ereignis *bEvt5* keinen Einfluss.

bEvt6: Das Ereignis *bEvt6* zählt als eine der Einschaltbedingungen von *bQ6*. In der Einschaltphase von *bQ6* ist die Einschaltverzögerung des Ausgangs nur dann aktiv, wenn *bEvt6* TRUE ist.

Einschaltbedingung:

bQ6 = *bQ5*AND*bEvt6*AND*stState.udiSecRT_DelayOn* = 0 AND*stState.udiSecRT_MinOff[6]* = 0 AND*stState.udiStep* = 6

Auf die Ausschaltbedingung von *bQ6* hat das Ereignis *bEvt6* keinen Einfluss.

bEvt7: Das Ereignis *bEvt7* zählt als eine der Einschaltbedingungen von *bQ7*. In der Einschaltphase von *bQ7* ist die Einschaltverzögerung des Ausgangs nur dann aktiv, wenn *bEvt7* TRUE ist.

Einschaltbedingung:

bQ7 = *bQ6*AND*bEvt7*AND*stState.udiSecRT_DelayOn* = 0 AND*stState.udiSecRT_MinOff[7]* = 0 AND*stState.udiStep* = 7

Auf die Ausschaltbedingung von *bQ7* hat das Ereignis *bEvt7* keinen Einfluss.

bEvt8: Das Ereignis *bEvt8* zählt als eine der Einschaltbedingungen von *bQ8*. In der Einschaltphase von *bQ8* ist die Einschaltverzögerung des Ausgangs nur dann aktiv, wenn *bEvt8* TRUE ist.

Einschaltbedingung:

bQ8 = *bQ7*AND*bEvt8*AND*stState.udiSecRT_DelayOn* = 0 AND*stState.udiSecRT_MinOff[8]* = 0 AND*stState.udiStep* = 8

Auf die Ausschaltbedingung von *bQ8* hat das Ereignis *bEvt8* keinen Einfluss.

VAR_OUTPUT

```
bQ_Cmd        : BOOL;
bQ1           : BOOL;
bQ2           : BOOL;
bQ3           : BOOL;
bQ4           : BOOL;
bQ5           : BOOL;
bQ6           : BOOL;
bQ7           : BOOL;
bQ8           : BOOL;
stState       : ST_HVACCmdCtrl_8State;
```

bQ_Cmd: Ausgang um ein Gerät frei zu geben. Ist der Ausgang *bQ_Cmd* in der Einschaltfolge gesetzt worden, so bleibt dieser in Selbsthaltung und wird erst in der Ausschaltfolge zurück gesetzt.

Zu *bQ_Cmd* gehört die Eingangsvariable *bCmd* und die Parameterstruktur *stParamCmd*. *bCmd* ist eine der Einschaltbedingungen von *bQ_Cmd*. Über die Zeitvariablen der Parameterstruktur kann eine Ein- bzw. Ausschaltverzögerung sowie eine Mindestein- bzw. Mindestausschaltdauer des Ausgangs definiert werden. Der Status der Zeitverläufe wird mit der Ausgangsstruktur *stState* angezeigt.

Einschaltbedingung

bQ_Cmd = *bCmd*AND*stState.udiSecRT_DelayOn* = 0 AND*stState.udiSecRT_MinOff[0]* = 0 AND*stState.udiStep* = 0

Ausschaltbedingung

bQ_Cmd = NOT*bQ1*AND*stState.udiSecRT_DelayOff* = 0 AND*stState.udiSecRT_MinOn[0]* = 0 AND*stState.udiStep* = 0

bQ1: Ausgang um ein Gerät frei zu geben. Ist der Ausgang *bQ1* in der Einschaltfolge gesetzt worden, so bleibt dieser in Selbsthaltung und wird erst in der Ausschaltfolge zurück gesetzt.

Zu *bQ1* gehört das Ereignis 1 *bEvt1* und die Parameterstruktur *stParam1*. *bEvt1* ist eine der Einschaltbedingungen von *bQ1*. Über die Zeitvariablen der Parameterstruktur kann eine Ein- bzw. Ausschaltverzögerung sowie eine Mindestein- bzw. Mindestausschaltdauer des Ausgangs definiert werden. Der Status der Zeitverläufe wird mit der Ausgangsstruktur *stState* angezeigt.

Einschaltbedingung

bQ1 = *bEvt1*AND*stState.udiSecRT_DelayOn* = 0 AND*stState.udiSecRT_MinOff[1]* = 0 AND*stState.udiStep* = 1

Ausschaltbedingung

bQ1 = NOT*bQ1*AND*stState.udiSecRT_DelayOff* = 0 AND*stState.udiSecRT_MinOn[1]* = 0 AND*stState.udiStep* = 1

bQ2: Ausgang um ein Gerät frei zu geben. Ist der Ausgang *bQ2* in der Einschaltfolge gesetzt worden, so bleibt dieser in Selbsthaltung und wird erst in der Ausschaltfolge zurück gesetzt.

Zu *bQ2* gehört das Ereignis 2 *bEvt2* und die Parameterstruktur *stParam2*. *bEvt2* ist eine der Einschaltbedingungen von *bQ2*. Über die Zeitvariablen der Parameterstruktur kann eine Ein- bzw. Ausschaltverzögerung sowie eine Mindestein- bzw. Mindestausschaltdauer des Ausgangs definiert werden. Der Status der Zeitverläufe wird mit der Ausgangsstruktur *stState* angezeigt.

Einschaltbedingung

bQ2 = *bEvt2*AND*stState.udiSecRT_DelayOn* = 0 AND*stState.udiSecRT_MinOff[2]* = 0 AND*stState.udiStep* = 2

Ausschaltbedingung

bQ2 = NOT*bQ2*AND*stState.udiSecRT_DelayOff* = 0 AND*stState.udiSecRT_MinOn[2]* = 0 AND*stState.udiStep* = 2

bQ3: Ausgang um ein Gerät frei zu geben. Ist der Ausgang *bQ3* in der Einschaltfolge gesetzt worden, so bleibt dieser in Selbsthaltung und wird erst in der Ausschaltfolge zurück gesetzt.

Zu *bQ3* gehört das Ereignis 3 *bEvt3* und die Parameterstruktur *stParam3*. *bEvt3* ist eine der Einschaltbedingungen von *bQ3*. Über die Zeitvariablen der Parameterstruktur kann eine Ein- bzw. Ausschaltverzögerung sowie eine Mindestein- bzw. Mindestausschaltdauer des Ausgangs definiert werden. Der Status der Zeitverläufe wird mit der Ausgangsstruktur *stState* angezeigt.

Einschaltbedingung

bQ3 = *bEvt3*AND*stState.udiSecRT_DelayOn* = 0 AND*stState.udiSecRT_MinOff[3]* = 0 AND*stState.udiStep* = 3

Ausschaltbedingung

bQ3 = NOT*bQ3*AND*stState.udiSecRT_DelayOff* = 0 AND*stState.udiSecRT_MinOn[3]* = 0 AND*stState.udiStep* = 3

bQ4: Ausgang um ein Gerät frei zu geben. Ist der Ausgang *bQ4* in der Einschaltfolge gesetzt worden, so bleibt dieser in Selbsthaltung und wird erst in der Ausschaltfolge zurück gesetzt.

Zu *bQ4* gehört das Ereignis 4 *bEvt4* und die Parameterstruktur *stParam4*. *bEvt4* ist eine der Einschaltbedingungen von *bQ4*. Über die Zeitvariablen der Parameterstruktur kann eine Ein- bzw. Ausschaltverzögerung sowie eine Mindestein- bzw. Mindestausschaltdauer des Ausgangs definiert werden. Der Status der Zeitverläufe wird mit der Ausgangsstruktur *stState* angezeigt.

Einschaltbedingung

bQ4 = *bEvt4*AND*stState.udiSecRT_DelayOn* = 0 AND*stState.udiSecRT_MinOff[4]* = 0 AND*stState.udiStep* = 4

Ausschaltbedingung

bQ4 = NOT*bQ4*AND*stState.udiSecRT_DelayOff* = 0 AND*stState.udiSecRT_MinOn[4]* = 0 AND*stState.udiStep* = 4

bQ5: Ausgang um ein Gerät frei zu geben. Ist der Ausgang *bQ5* in der Einschaltfolge gesetzt worden, so bleibt dieser in Selbsthaltung und wird erst in der Ausschaltfolge zurück gesetzt.

Zu *bQ5* gehört das Ereignis 5 *bEvt5* und die Parameterstruktur *stParam5*. *bEvt5* ist eine der Einschaltbedingungen von *bQ5*. Über die Zeitvariablen der Parameterstruktur kann eine Ein- bzw. Ausschaltverzögerung sowie eine Mindestein- bzw. Mindestausschaltdauer des Ausgangs definiert werden. Der Status der Zeitverläufe wird mit der Ausgangsstruktur *stState* angezeigt.

Einschaltbedingung

bQ5 = *bEvt5*AND*stState.udiSecRT_DelayOn* = 0 AND*stState.udiSecRT_MinOff[5]* = 0 AND*stState.udiStep* = 5

Ausschaltbedingung

bQ5 = NOT*bQ5*AND*stState.udiSecRT_DelayOff* = 0 AND*stState.udiSecRT_MinOn[5]* = 0 AND*stState.udiStep* = 5

bQ6: Ausgang um ein Gerät frei zu geben. Ist der Ausgang *bQ6* in der Einschaltfolge gesetzt worden, so bleibt dieser in Selbsthaltung und wird erst in der Ausschaltfolge zurück gesetzt.

Zu *bQ6* gehört das Ereignis 6 *bEvt6* und die Parameterstruktur *stParam6*. *bEvt6* ist eine der Einschaltbedingungen von *bQ6*. Über die Zeitvariablen der Parameterstruktur kann eine Ein- bzw. Ausschaltverzögerung sowie eine Mindestein- bzw. Mindestausschaltdauer des Ausgangs definiert werden. Der Status der Zeitverläufe wird mit der Ausgangsstruktur *stState* angezeigt.

Einschaltbedingung

bQ6 = *bEvt6*AND*stState.udiSecRT_DelayOn* = 0 AND*stState.udiSecRT_MinOff[6]* = 0 AND*stState.udiStep* = 6

Ausschaltbedingung

bQ6 = NOT*bQ6*AND*stState.udiSecRT_DelayOff* = 0 AND*stState.udiSecRT_MinOn[6]* = 0 AND*stState.udiStep* = 6

bQ7: Ausgang um ein Gerät frei zu geben. Ist der Ausgang *bQ7* in der Einschaltfolge gesetzt worden, so bleibt dieser in Selbsthaltung und wird erst in der Ausschaltfolge zurück gesetzt.

Zu *bQ7* gehört das Ereignis 7 *bEvt7* und die Parameterstruktur *stParam7*. *bEvt7* ist eine der Einschaltbedingungen von *bQ7*. Über die Zeitvariablen der Parameterstruktur kann eine Ein- bzw. Ausschaltverzögerung sowie eine Mindestein- bzw. Mindestausschaltdauer des Ausgangs definiert werden. Der Status der Zeitverläufe wird mit der Ausgangsstruktur *stState* angezeigt.

Einschaltbedingung

bQ7 = *bEvt7*AND*stState.udiSecRT_DelayOn* = 0 AND*stState.udiSecRT_MinOff[7]* = 0 AND*stState.udiStep* = 7

Ausschaltbedingung

bQ7 = NOT*bQ7*AND*stState.udiSecRT_DelayOff* = 0 AND*stState.udiSecRT_MinOn[7]* = 0 AND*stState.udiStep* = 7

bQ8: Ausgang um ein Gerät frei zu geben. Ist der Ausgang *bQ8* in der Einschaltfolge gesetzt worden, so bleibt dieser in Selbsthaltung und wird erst in der Ausschaltfolge zurück gesetzt.

Zu *bQ8* gehört das Ereignis 8 *bEvt8* und die Parameterstruktur *stParam8*. *bEvt8* ist eine der Einschaltbedingungen von *bQ8*. Über die Zeitvariablen der Parameterstruktur kann eine Ein- bzw. Ausschaltverzögerung sowie eine Mindestein- bzw. Mindestausschaltdauer des Ausgangs definiert werden. Der Status der Zeitverläufe wird mit der Ausgangsstruktur *stState* angezeigt.

Einschaltbedingung

bQ8 = *bEvt8*AND*stState.udiSecRT_DelayOn* = 0 AND*stState.udiSecRT_MinOff[8]* = 0 AND*stState.udiStep* = 8

Ausschaltbedingung

bQ8 = *stState.udiSecRT_DelayOff* = 0 AND*stState.udiSecRT_MinOn[8]* = 0 AND*stState.udiStep* = 8

stState: Die Struktur zeigt die verbleibenden Zeiten der Ein- bzw. Ausschaltverzögerung der aktiven Stufe und die Mindestein- bzw. Mindestausschaltdauer der Ausgänge an.

stState.udiSecRT_DelayOn: In der Einschaltphase der Ausgänge (*bCmd* = TRUE) wird über *stState.udiSecRT_DelayOn* die verbleibende Zeit der Einschaltverzögerung der aktuellen Stufe *stState.udiStep* angezeigt.

Beispiel: Ist *stState.udiStep* = 5, so ist der Funktionsbaustein in Stufe 5. Für die Einschaltverzögerung des Ausgangs *bQ5* wird die Zeitvariable der Parameterstruktur *stParam5.udiSecDelayOn* verwendet und über *udiSecRT_DelayOn* die verbleibende Zeit angezeigt.

stState.udiSecRT_DelayOff: In der Ausschaltphase der Ausgänge (*bCmd* = FALSE) wird über *udiSecRT_DelayOff* die verbleibende Zeit der Ausschaltverzögerung der aktuellen Stufe *stState.udiStep* angezeigt.

Beispiel: Ist *stState.udiStep* = 0, so ist der Funktionsbaustein in Stufe 0. Für die Ausschaltverzögerung des Ausgangs *bQ_Cmd* wird die Zeitvariable *stParamCmd.udiSecDelayOff* verwendet und über *udiSecRT_DelayOff* die verbleibende Zeit angezeigt.

stState.udiStep: Status in welcher Stufe sich der Funktionsbaustein befindet.

Beispiele:

- Ist *stState.udiStep* = 0, so befindet sich der Funktionsbaustein in der Ein- oder Ausschaltfolge des Ausgangs *bQ_Cmd.*

- Ist *stState.udiStep* = 1, so befindet sich der Funktionsbaustein in der Ein- oder Ausschaltfolge des Ausgangs *bQ1.*

- Ist *stState.udiStep = 8,* so befindet sich der Funktionsbaustein in der Ein- oder Ausschaltfolge des Ausgangs *bQ8*

stState.udiSecRT_MinOn[0]: **..** g_iNumOfCmdCtrl_8 [▶ 541] In dem eindimensionalen Feld (Tabelle) *stState.udiSecRT_MinOn[0..*g_iNumOfCmdCtrl_8 [▶ 541]*]* wird die verbleibende Zeit der Mindesteinschaltdauer der Ausgänge angezeigt. Nach Ablauf der Mindesteinschaltdauer eines Ausgangs kann in der aktuellen Stufe *stState.udiStep* der Ausgang ausgeschaltet werden.

stState.udiSecRT_MinOff[0]: **..** g_iNumOfCmdCtrl_8 [▶ 541] In dem eindimensionalen Feld (Tabelle) *stState.udiSecRT_MinOff[0..*g_iNumOfCmdCtrl_8 [▶ 541]*]* wird die verbleibende Zeit der Mindestausschaltdauer der Ausgänge angezeigt. Nach Ablauf der Mindestausschaltdauer eines Ausgangs kann in der aktuellen Stufe *stState.udiStep* der Ausgang eingeschaltet werden.

VAR_IN_OUT

```
stParamCmd   : ST_HVACCmdCtrl_8Param;
stParam1     : ST_HVACCmdCtrl_8Param;
stParam2     : ST_HVACCmdCtrl_8Param;
stParam3     : ST_HVACCmdCtrl_8Param;
stParam4     : ST_HVACCmdCtrl_8Param;
stParam5     : ST_HVACCmdCtrl_8Param;
stParam6     : ST_HVACCmdCtrl_8Param;
stParam7     : ST_HVACCmdCtrl_8Param;
stParam8     : ST_HVACCmdCtrl_8Param;
```

stParamCmd: Die Struktur enthält Zeitvariablen über die eine Ein- bzw. Ausschaltverzögerung sowie eine Mindestein- bzw. Mindestausschaltdauer des Ausgangs *bQ_CMD* definiert werden kann. Die Variable wird persistent gespeichert.

stParam1: Die Struktur enthält Zeitvariablen über die eine Ein- bzw. Ausschaltverzögerung sowie eine Mindestein- bzw. Mindestausschaltdauer des Ausgangs *bQ1* definiert werden kann. Die Variable wird persistent gespeichert.

stParam2: Die Struktur enthält Zeitvariablen über die eine Ein- bzw. Ausschaltverzögerung sowie eine Mindestein- bzw. Mindestausschaltdauer des Ausgangs *bQ2* definiert werden kann. Die Variable wird persistent gespeichert.

stParam3: Die Struktur enthält Zeitvariablen über die eine Ein- bzw. Ausschaltverzögerung sowie eine Mindestein- bzw. Mindestausschaltdauer des Ausgangs *bQ3* definiert werden kann. Die Variable wird persistent gespeichert.

stParam4: Die Struktur enthält Zeitvariablen über die eine Ein- bzw. Ausschaltverzögerung sowie eine Mindestein- bzw. Mindestausschaltdauer des Ausgangs *bQ4* definiert werden kann. Die Variable wird persistent gespeichert.

stParam5: Die Struktur enthält Zeitvariablen über die eine Ein- bzw. Ausschaltverzögerung sowie eine Mindestein- bzw. Mindestausschaltdauer des Ausgangs *bQ5* definiert werden kann. Die Variable wird persistent gespeichert.

stParam6: Die Struktur enthält Zeitvariablen über die eine Ein- bzw. Ausschaltverzögerung sowie eine Mindestein- bzw. Mindestausschaltdauer des Ausgangs *bQ6* definiert werden kann. Die Variable wird persistent gespeichert.

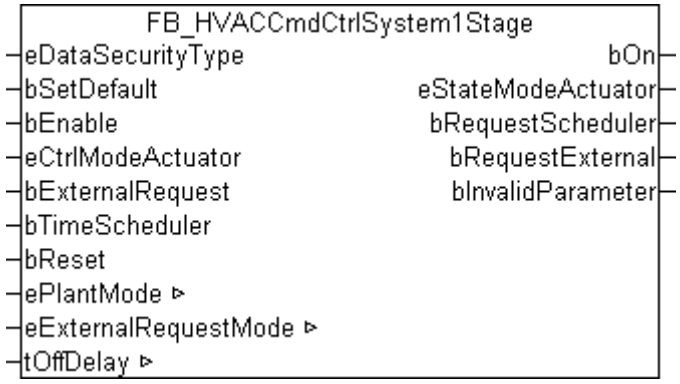
stParam7: Die Struktur enthält Zeitvariablen über die eine Ein- bzw. Ausschaltverzögerung sowie eine Mindestein- bzw. Mindestausschaltdauer des Ausgangs *bQ7* definiert werden kann. Die Variable wird persistent gespeichert.

stParam8: Die Struktur enthält Zeitvariablen über die eine Ein- bzw. Ausschaltverzögerung sowie eine Mindestein- bzw. Mindestausschaltdauer des Ausgangs *bQ8* definiert werden kann. Die Variable wird persistent gespeichert.

Dokumente hierzu

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

3.6.7 FB_HVACCmdCtrlSystem1Stage



Anwendung

Dieser Funktionsbaustein *FB_HVACCmdCtrlSystem1Stage* ist ein Anlagenschalter. Mit Ihm wird beispielsweise eine einstufige Lüftungsanlage in den Automatik- oder Handbetrieb geschaltet. Im Automatikbetrieb könnte die Anlage über ein Zeitschaltprogramm oder über die Anforderung eines Bedientableaus gesteuert werden. Der Funktionsbaustein *FB_HVACCmdCtrlSystem1Stage* ist aktiv, wenn die Eingangsvariablen bEnable = TRUE und *eCtrlModeActuator* = *eHVACActuatorMode_Auto_BMS oder eHVACActuatorMode_Auto_OP* ist.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bEnable           : BOOL;
eCtrlModeActuator : E_HVACActuatorMode;
bExternalRequest  : BOOL;
bTimeScheduler    : BOOL;
bReset            : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Freigabe des Bausteines, wenn *bEnable* = TRUE ist. Ist *bEnable* = FALSE, so ist *bOn* = FALSE.

eCtrlModeActuator: Enum, über welches die Betriebsarten Hand, Auto und Aus der Anlage vorgegeben wird. Bei falscher Angabe wird intern mit der letzten, gültigen Betriebsart weiter gearbeitet. Bei der Erstinbetriebnahme ist diese *eHVACActuatorMode_Auto_BMS*. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

bExternalRequest: Externe Anforderung der Anlage z.B. von einem Bedientableau über einen Taster oder Schalter.

bTimeScheduler: Anforderung der Anlage über ein Zeitschaltprogramm.

bReset: Quittierungseingang bei einem Fehler

VAR_OUTPUT

```
bOn                : BOOL;
eStateModeActuator: E_HVACActuatorMode;
bRequestScheduler : BOOL;
bRequestExternal   : BOOL;
bInvalidParameter : BOOL;
```

bOn: An dieser Ausgangsvariable wird die Freigabe der Anlage ausgegeben.

eStateModeActuator: Enum, über das der Status der Betriebsart des Motors an die Steuerung zurück gegeben wird.

bRequestScheduler: Dieser Ausgang signalisiert, dass die Anlage von der Eingangsvariable *bTimeScheduler* angefordert ist.

bRequestExternal: Dieser Ausgang signalisiert, dass die Anlage von der Eingangsvariable *bExternalRequest* angefordert ist.

bInvalidParameter: Zeigt an, dass ein falscher Parameter an einer der Variablen *eCtrlModeActuator,ePlantMode* oder *eExternalRequestMode* anliegt. Eine falsche Parameterangabe führt nicht zum Stillstand des Bausteines, siehe Beschreibung der Variablen. Nach Behebung der falschen Parameterangabe muss die Meldung *bInvalidParameter* mit *bReset* quittiert werden.

VAR_IN_OUT

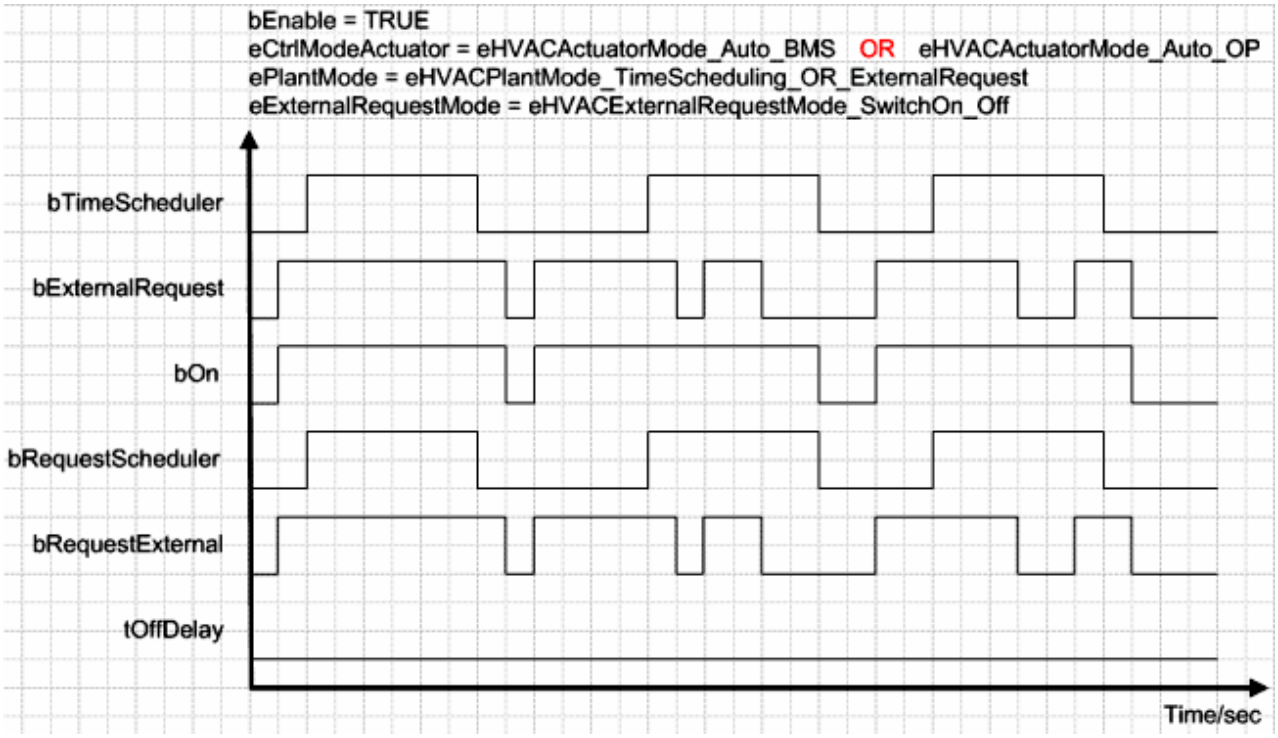
```
ePlantMode           : E_HVACPlantMode;
eExternalRequestMode : E_HVACExternalRequestMode;
tOffDelay            : TIME;
```

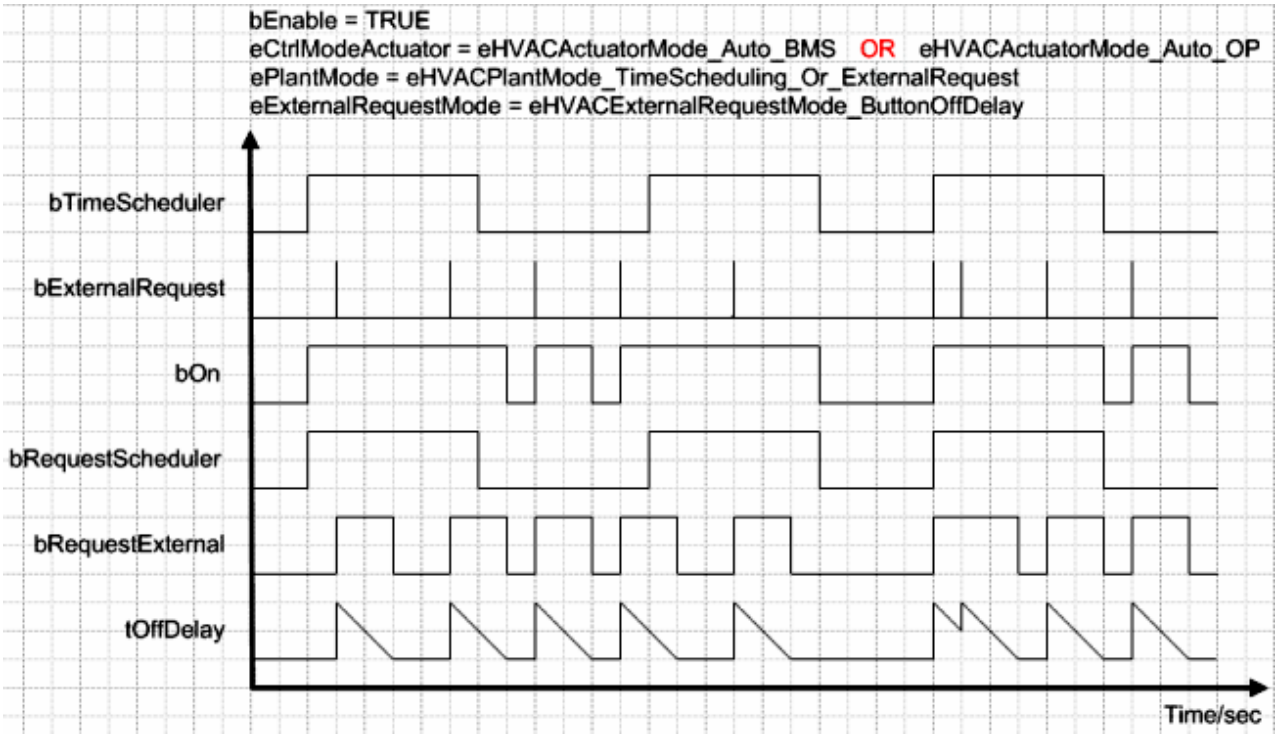
ePlantMode: Mit dieser Aufzählungsvariable können verschiedene Funktionen der Anlage im Automatikbetrieb in Abhängigkeit der Eingangsvariablen *bExternalRequest* und *bTimeScheduler* vorgenommen werden.
ePlantMode = *eHVACPlantMode_TimeSchedulingOnly*: Die Anlage wird ausschließlich über die Eingangsvariable *bTimeScheduler* ein- und ausgeschaltet.
ePlantMode = *eHVACPlantMode_TimeScheduling_And_ExternalRequest*: Die Anlage wird eingeschaltet, wenn die Eingangsvariablen *bTimeScheduler* UND *bExternalRequest* = TRUE sind.
ePlantMode = *eHVACPlantMode_TimeScheduling_Or_ExternalRequest*: Die Anlage wird über die Eingangsvariablen *bTimeScheduler* ODER *bExternalRequest*eingeschaltet.
ePlantMode = *eHVACPlantMode_ExternalRequestOnly*: Die Anlage wird ausschließlich über die Eingangsvariable *bExternalRequest* ein- und ausgeschaltet.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weiter gearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

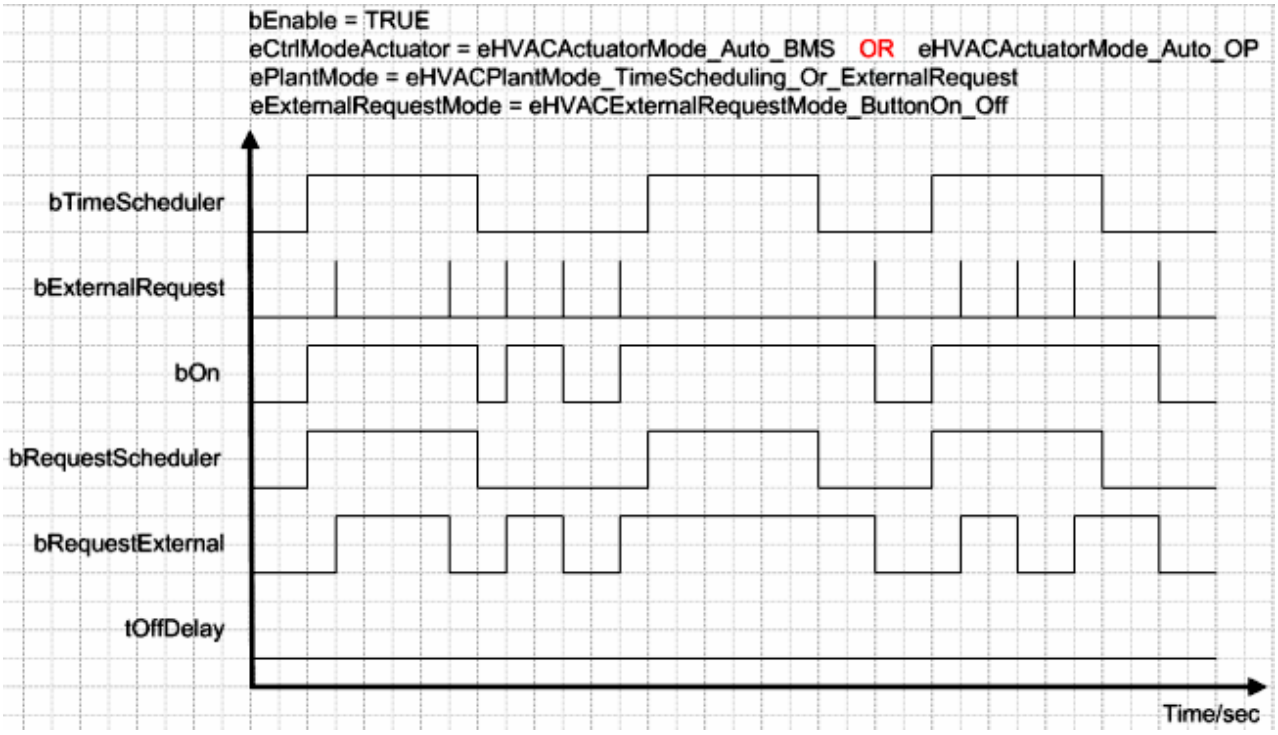
eExternalRequestMode: Mit dem Enum *eExternalRequestMode* wird die Wirkungsweise der Eingangsvariable *bExternalRequest* im Automatikbetrieb in Abhängigkeit des Enums *ePlantMode* vorgegeben.
eExternalRequestMode = *eHVACExternalRequestMode_ButtonOn_Off*: Die externe Anforderung wird nach einer steigenden Flanke von *bExternalRequest* auf TRUE gesetzt. Eine weitere steigende Flanke setzt die Anforderung wieder auf FALSE zurück.
eExternalRequestMode = *eHVACExternalRequestMode_ButtonOffDelay*: Die externe Anforderung verlängert oder setzt die Nutzzeit der Anlage nach einer steigenden Flanke an der Eingangsvariablen *bExternalRequest* um die eingestellte Zeit von *tOffDelay*.
eExternalRequestMode = *eHVACExternalRequestMode_SwitchOn_Off*: Die externe Anforderung ist aktiv, wenn *bExternalRequest* = TRUE ist. Sie wird deaktiv, wenn *bExternalRequest* = FALSE ist.
Liegt ein falscher Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weiter gearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt. Die Variable wird persistent gespeichert. Voreingestellt auf 0.

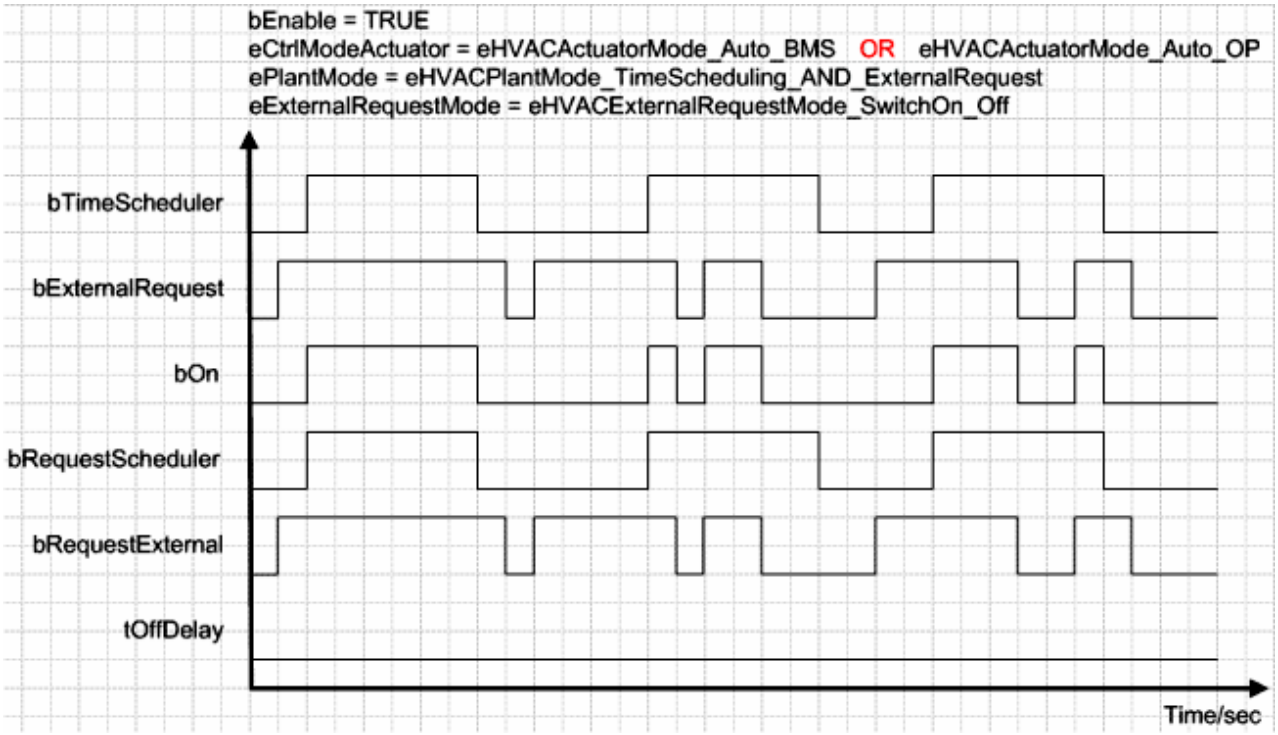
tOffDelay: Zeitwert für die Nutzzeitverlängerung der Anlage. Die Nutzzeitverlängerung kann nur aktiviert werden, wenn eModeExternalRequest = 2 ist. Die Variable wird persistent gespeichert. Voreingestellt auf 30min.

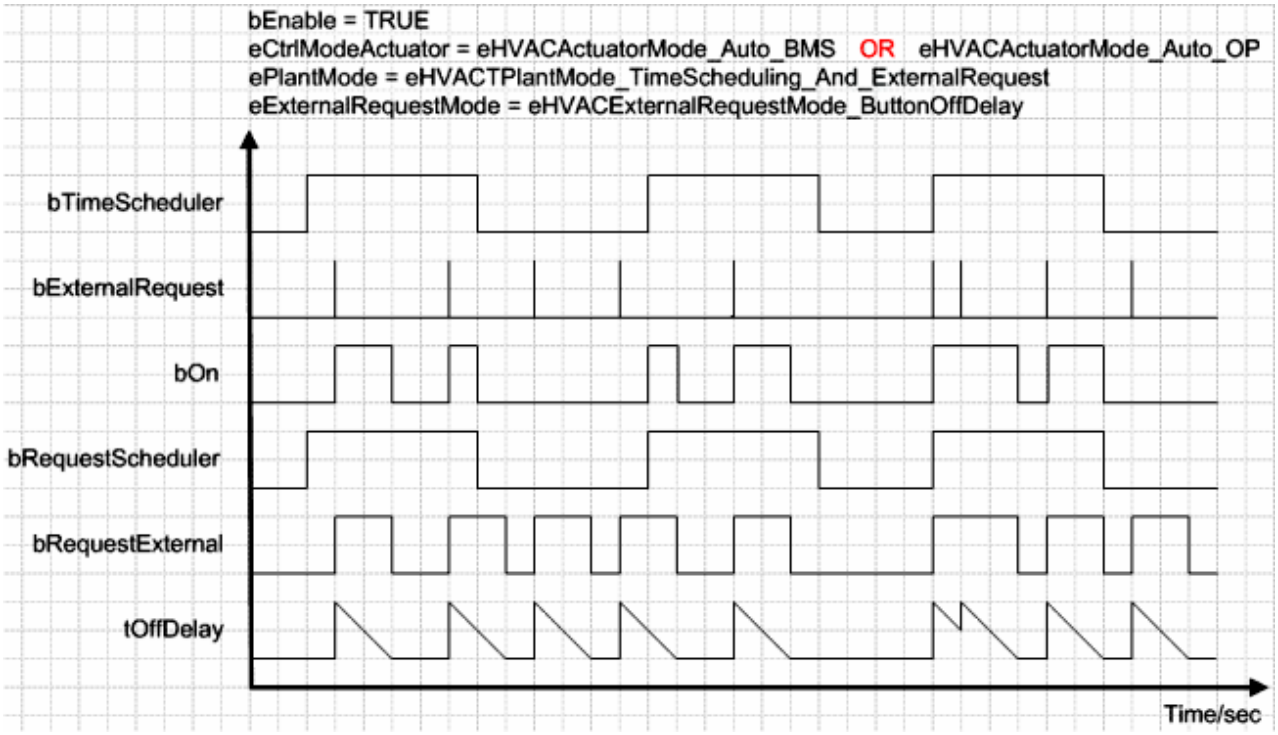
Verhalten der Ausgangsgröße

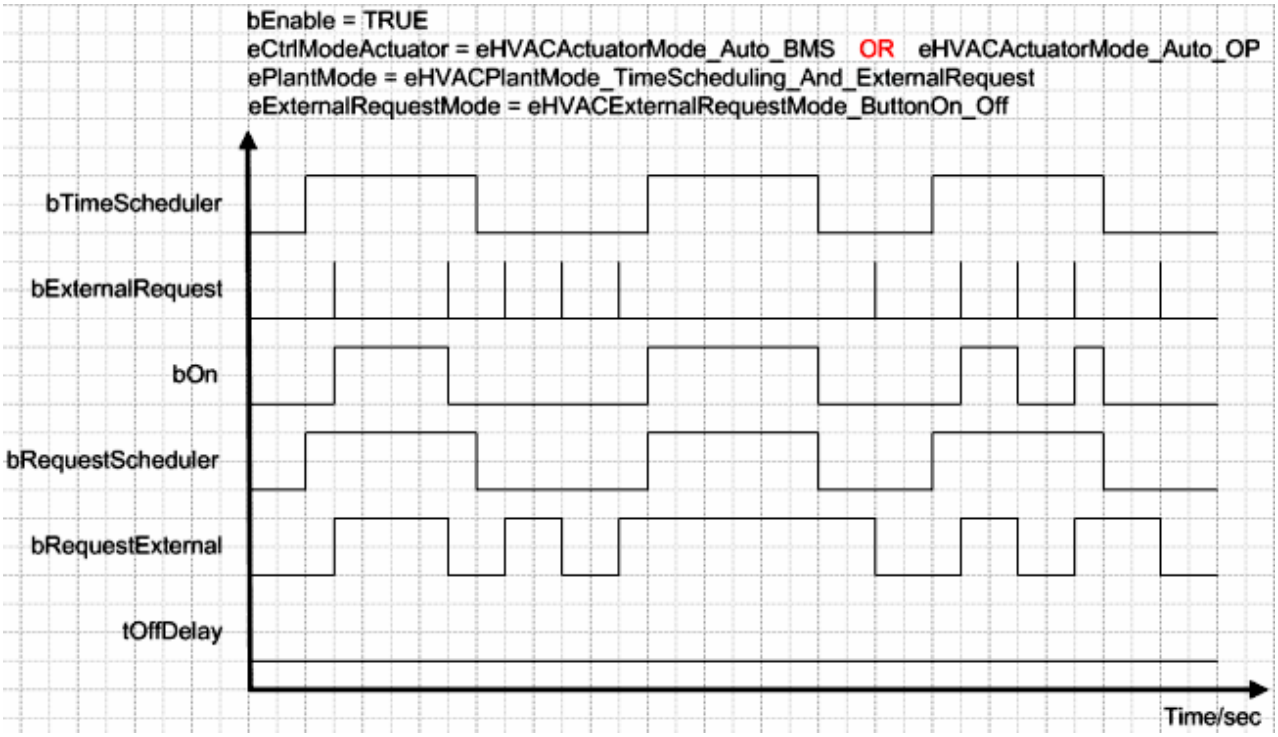


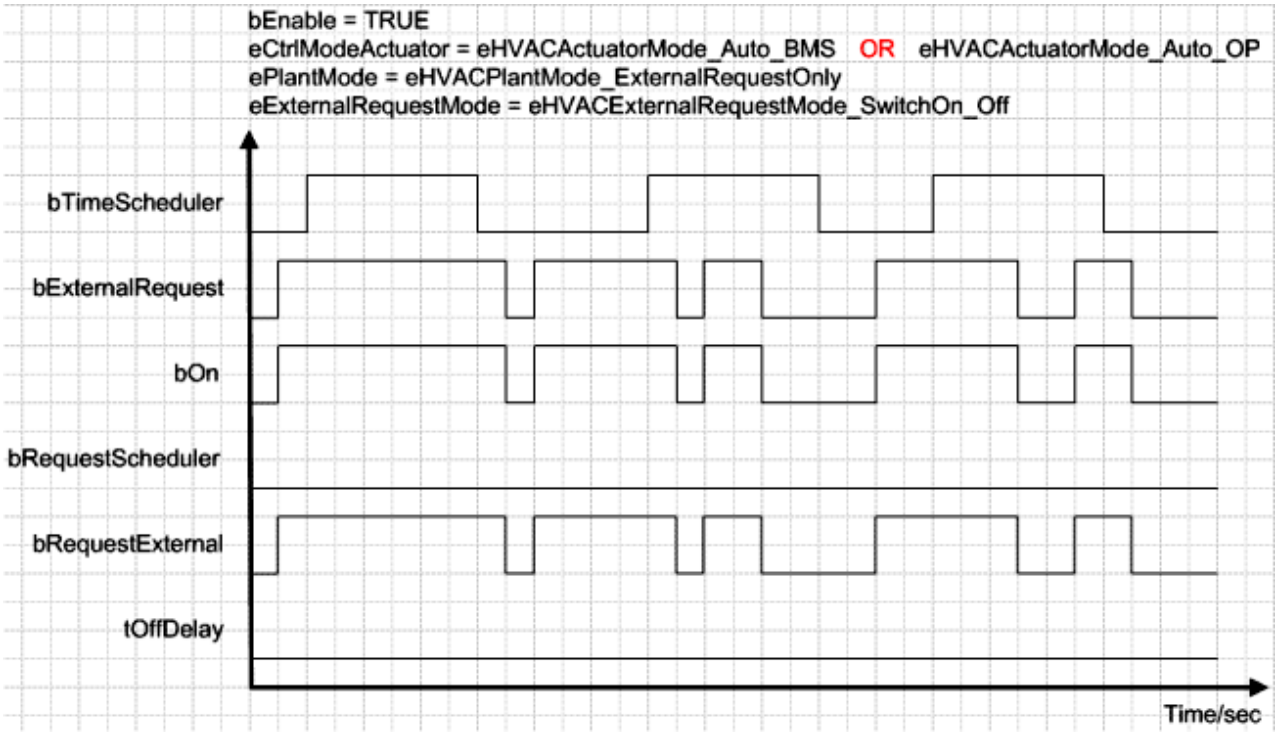


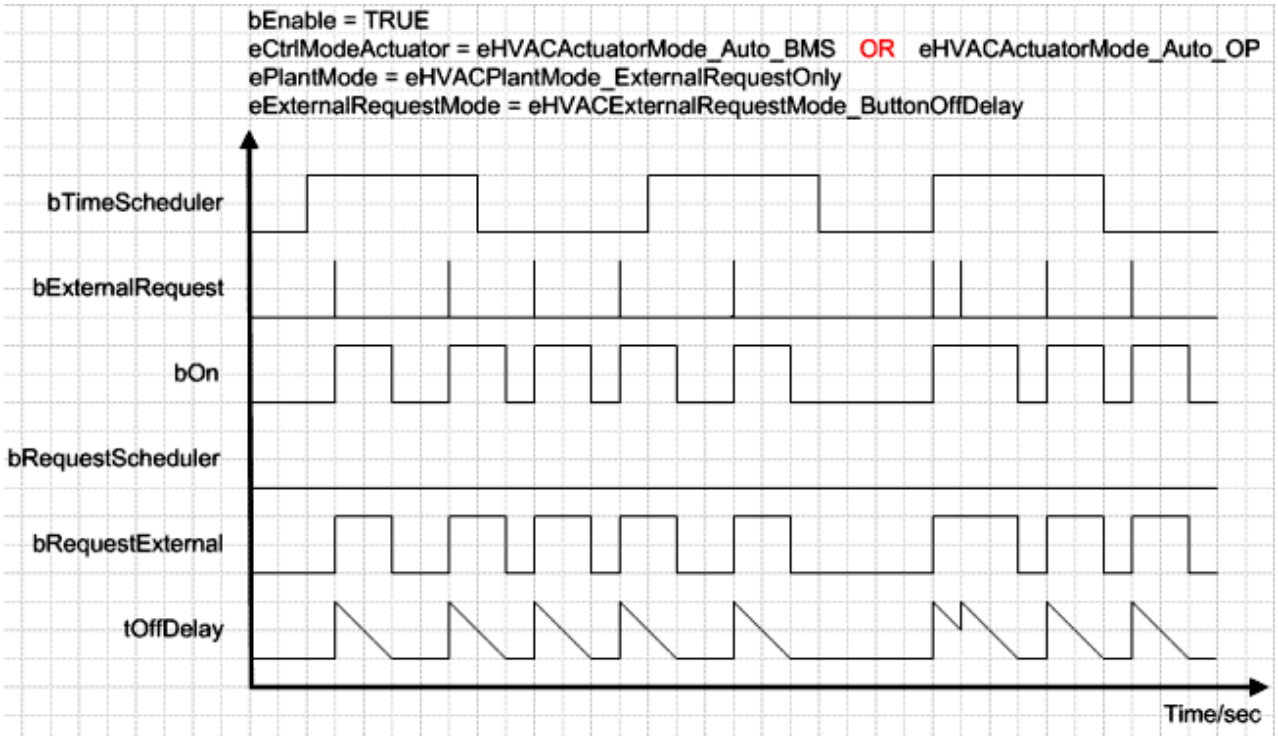

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeScheduling_Or_ExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_ButtonOn_Off
bTimeScheduler
bExternalRequest
bOn
bRequestScheduler
bRequestExternal
tOffDelay
Time/sec

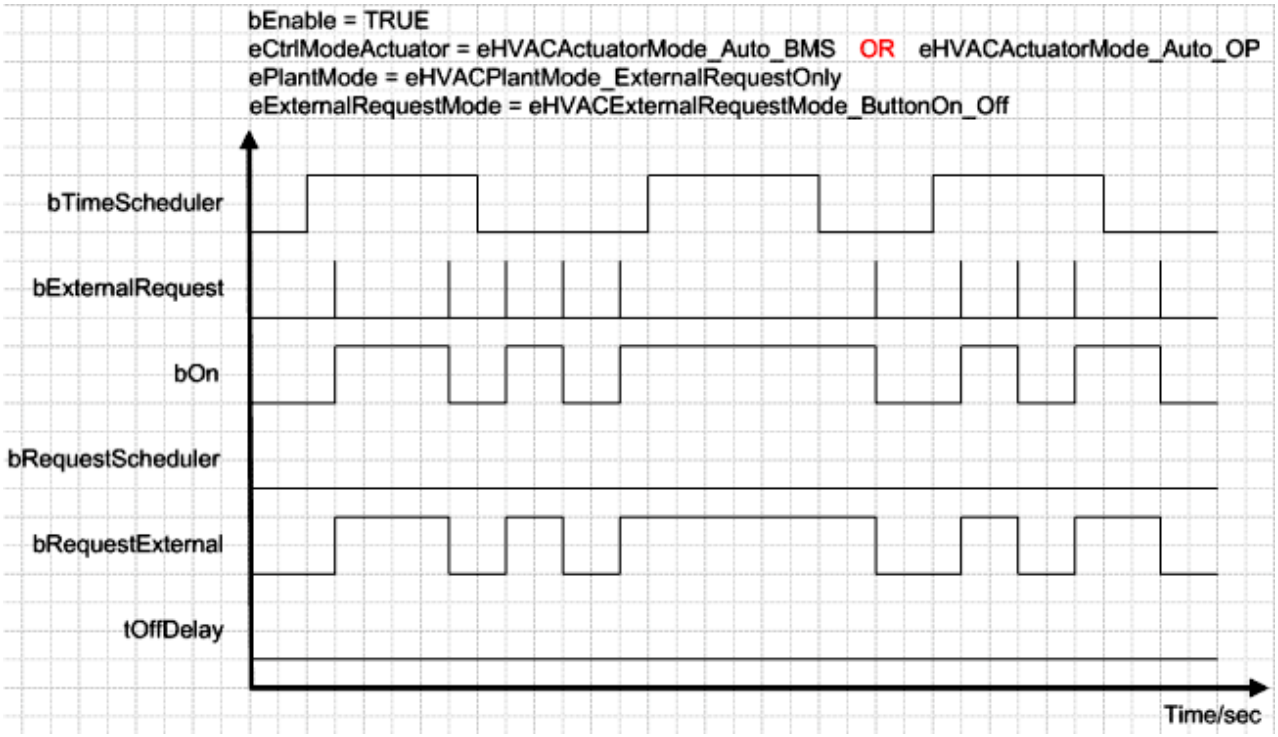

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeScheduling_AND_ExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_SwitchOn_Off
bTimeScheduler
bExternalRequest
bOn
bRequestScheduler
bRequestExternal
tOffDelay
Time/sec

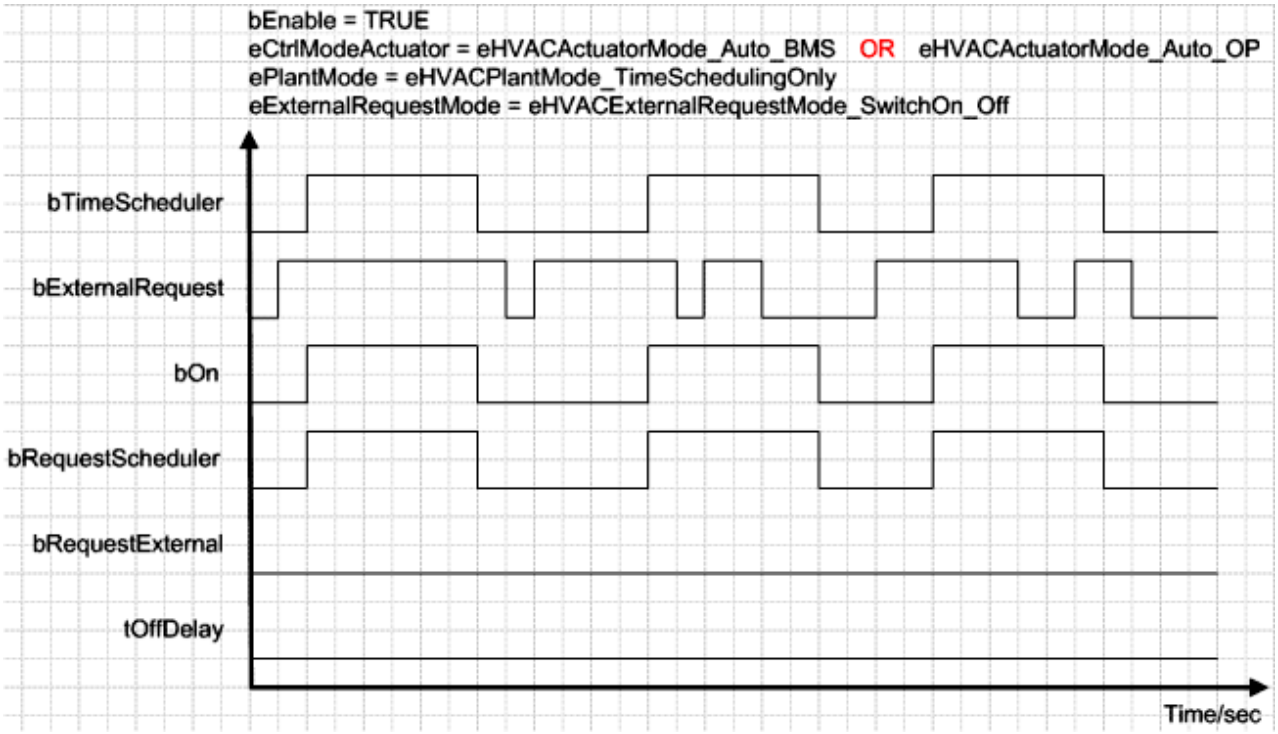

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACTPlantMode_TimeScheduling_And_ExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_ButtonOffDelay
bTimeScheduler
bExternalRequest
bOn
bRequestScheduler
bRequestExternal
tOffDelay
Time/sec

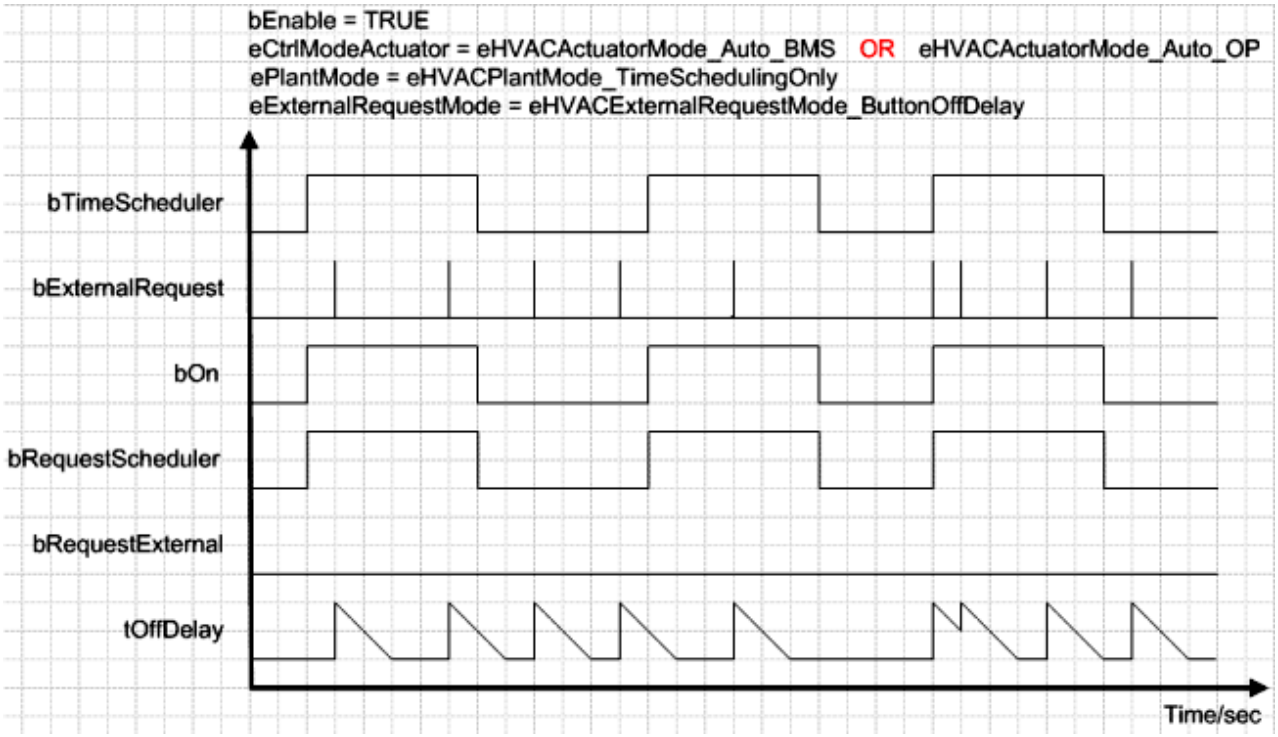

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeScheduling_And_ExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_ButtonOn_Off
bTimeScheduler
bExternalRequest
bOn
bRequestScheduler
bRequestExternal
tOffDelay
Time/sec

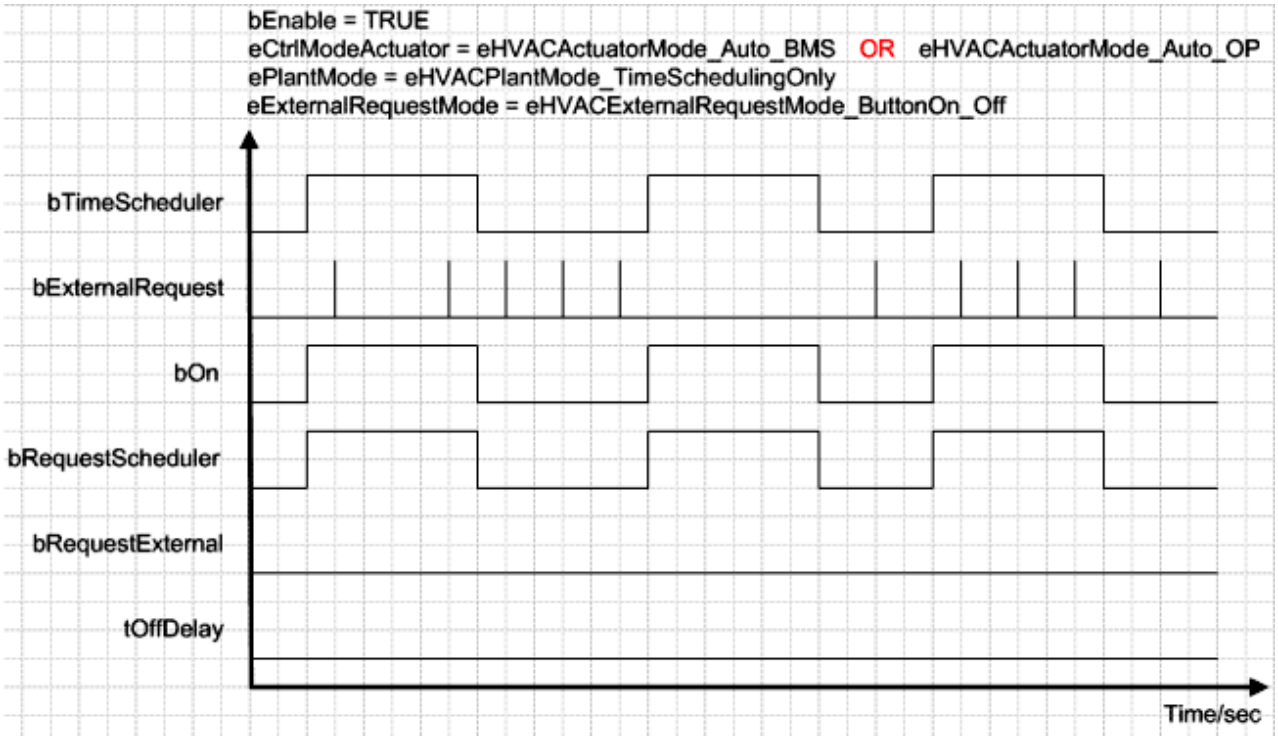

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_ExternalRequestOnly
eExternalRequestMode = eHVACExternalRequestMode_SwitchOn_Off
bTimeScheduler
bExternalRequest
bOn
bRequestScheduler
bRequestExternal
tOffDelay
Time/sec


bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_ExternalRequestOnly
eExternalRequestMode = eHVACExternalRequestMode_ButtonOffDelay
bTimeScheduler
bExternalRequest
bOn
bRequestScheduler
bRequestExternal
tOffDelay
Time/sec


bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_ExternalRequestOnly
eExternalRequestMode = eHVACExternalRequestMode_ButtonOn_Off
bTimeScheduler
bExternalRequest
bOn
bRequestScheduler
bRequestExternal
tOffDelay
Time/sec


bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeSchedulingOnly
eExternalRequestMode = eHVACExternalRequestMode_SwitchOn_Off
bTimeScheduler
bExternalRequest
bOn
bRequestScheduler
bRequestExternal
tOffDelay
Time/sec

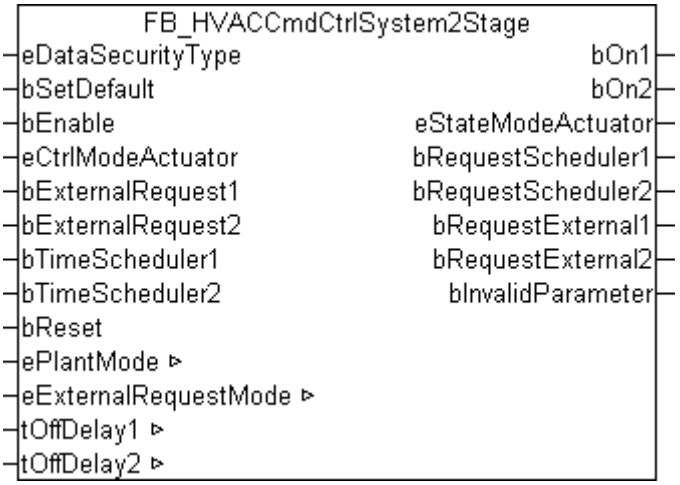




Dokumente hierzu

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

3.6.8 FB_HVACCmdCtrlSystem2Stage



Anwendung

Dieser Funktionsbaustein *FB_HVACCmdCtrlSystem2Stage* ist ein Anlagenschalter. Mit Ihm wird beispielsweise eine zweistufige Lüftungsanlage in den Automatik- oder Handbetrieb geschaltet. Im Automatikbetrieb könnte die Anlage über ein Zeitschaltprogramm oder über die Anforderung eines Bedientableaus gesteuert werden. Der Funktionsbaustein *FB_HVACCmdCtrlSystem2Stage* ist aktiv, wenn die Eingangsvariablen bEnable = TRUE und *eCtrlModeActuator* = *eHVACActuatorMode_Auto_BMS oder eHVACActuatorMode_Auto_OP* ist.

VAR_INPUT

```
eDataSecurityType  : E_HVACDataSecurityType;
bSetDefault        : BOOL;
bEnable            : BOOL;
eCtrlModeActuator  : E_HVACActuatorMode;
bExternalRequest1  : BOOL;
bExternalRequest2  : BOOL;
bTimeScheduler1    : BOOL;
bTimeScheduler2    : BOOL;
bReset             : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Freigabe des Bausteines, wenn *bEnable* = TRUE ist. Ist *bEnable* = FALSE, so sind *bOn1 und bOn2* = FALSE.

eCtrlModeActuator: Enum, über das die Betriebsarten Hand, Auto und Aus der Anlage vorgegeben werden. Bei falscher Angabe wird intern mit der letzten, gültigen Betriebsart weiter gearbeitet. Bei der Erstinbetriebnahme ist diese *eHVACActuatorMode_Auto_BMS*. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

bExternalRequest1: Externe Anforderung der Anlage in Stufe 1 z.B. von einem Bedientableau über einen Taster oder Schalter.

bExternalRequest2: Externe Anforderung der Anlage in Stufe 2 z.B. von einem Bedientableau über einen Taster oder Schalter.

bTimeScheduler1: Anforderung der Anlage in Stufe 1 über ein Zeitschaltprogramm.

bTimeScheduler2: Anforderung der Anlage in Stufe 2 über ein Zeitschaltprogramm.

bReset: Quittierungseingang bei einem Fehler.

VAR_OUTPUT

```
bOn1               : BOOL;
bOn2               : BOOL;
eStateModeActuator : E_HVACActuatorMode;
bRequestScheduler1 : BOOL;
bRequestScheduler2 : BOOL;
bRequestExternal1  : BOOL;
bRequestExternal2  : BOOL;
bInvalidParameter  : BOOL;
```

bOn1: An dieser Ausgangsvariablen wird die Freigabe der Anlage für die Stufe 1 ausgegeben.

bOn2: An dieser Ausgangsvariablen wird die Freigabe der Anlage für die Stufe 2 ausgegeben. Ist die 2. Stufe freigegeben, so ist automatisch die 1. Stufe ebenfalls freigegeben.

eStateModeActuator: Enum, über das der Status der Betriebsart des Motors an die Steuerung zurück gegeben wird.

bRequestScheduler1: Dieser Ausgang signalisiert, dass die Anlage von der Eingangsvariable *bTimeScheduler1* angefordert ist.

bRequestScheduler2: Dieser Ausgang signalisiert, dass die Anlage von der Eingangsvariable *bTimeScheduler2* angefordert ist.

bRequestExternal1: Dieser Ausgang signalisiert, dass die Anlage von der Eingangsvariable *bExternalRequest1* angefordert ist.

bRequestExternal2: Dieser Ausgang signalisiert, dass die Anlage von der Eingangsvariable *bExternalRequest2* angefordert ist.

bInvalidParameter: Zeigt an, dass ein falscher Parameter an einer der Variablen *eCtrlModeActuator,ePlantMode* oder *eExternalRequestMode* anliegt. Eine falsche Parameterangabe führt nicht zum Stillstand des Bausteines, siehe Beschreibung der Variablen. Nach Behebung der falschen Parameterangabe muss die Meldung *bInvalidParameter* mit *bReset* quittiert werden.

VAR_IN_OUT

```
ePlantMode          : E_HVACPlantMode;
eExternalRequestMode: E_HVACExternalRequestMode;
tOffDelay1          : TIME;
tOffDelay2          : TIME;
```

ePlantMode: Mit dieser Aufzählungsvariable können verschiedene Funktionen der Anlage im Automatikbetrieb in Abhängigkeit der Eingangsvariablen *bExternalRequest1, bExternalRequest2, bTimeScheduler1* und *bTimeScheduler2* vorgenommen werden.
ePlantMode = *eHVACPlantMode_TimeSchedulingOnly*: Die Anlage wird ausschließlich über die Eingangsvariablen *bTimeScheduler1* oder *bTimeScheduler2*in die jeweilige Stufe eingeschaltet.
ePlantMode = *eHVACPlantMode_TimeScheduling_And_ExternalRequest*: Die Anlage wird in Stufe 1 eingeschaltet, wenn die Eingangsvariablen *bTimeScheduler1* UND *bExternalRequest1* = TRUE sind. Sind

die Eingangsvariablen *bTimeScheduler2* UND *bExternalRequest2* = TRUE, so wird die Anlage in Stufe 2 eingeschaltet.
ePlantMode = *eHVACPlantMode_TimeScheduling_Or_ExternalRequest*: Die Anlage wird über die Eingangsvariablen *bExternalRequest1, bExternalRequest2, ODER bTimeScheduler1, bTimeScheduler2* in die jeweilige Stufe eingeschaltet.
ePlantMode = *eHVACPlantMode_ExternalRequestOnly*: Die Anlage wird ausschließlich über die Eingangsvariable *bExternalRequest1*oder *bExternalRequest2* in die jeweilige Stufe eingeschaltet.
Liegt ein falscher Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weiter gearbeitet.
bInvalidParameter wird bei falscher Parameterangabe gesetzt.

Die Variable wird persistent gespeichert. Voreingestellt auf 0.

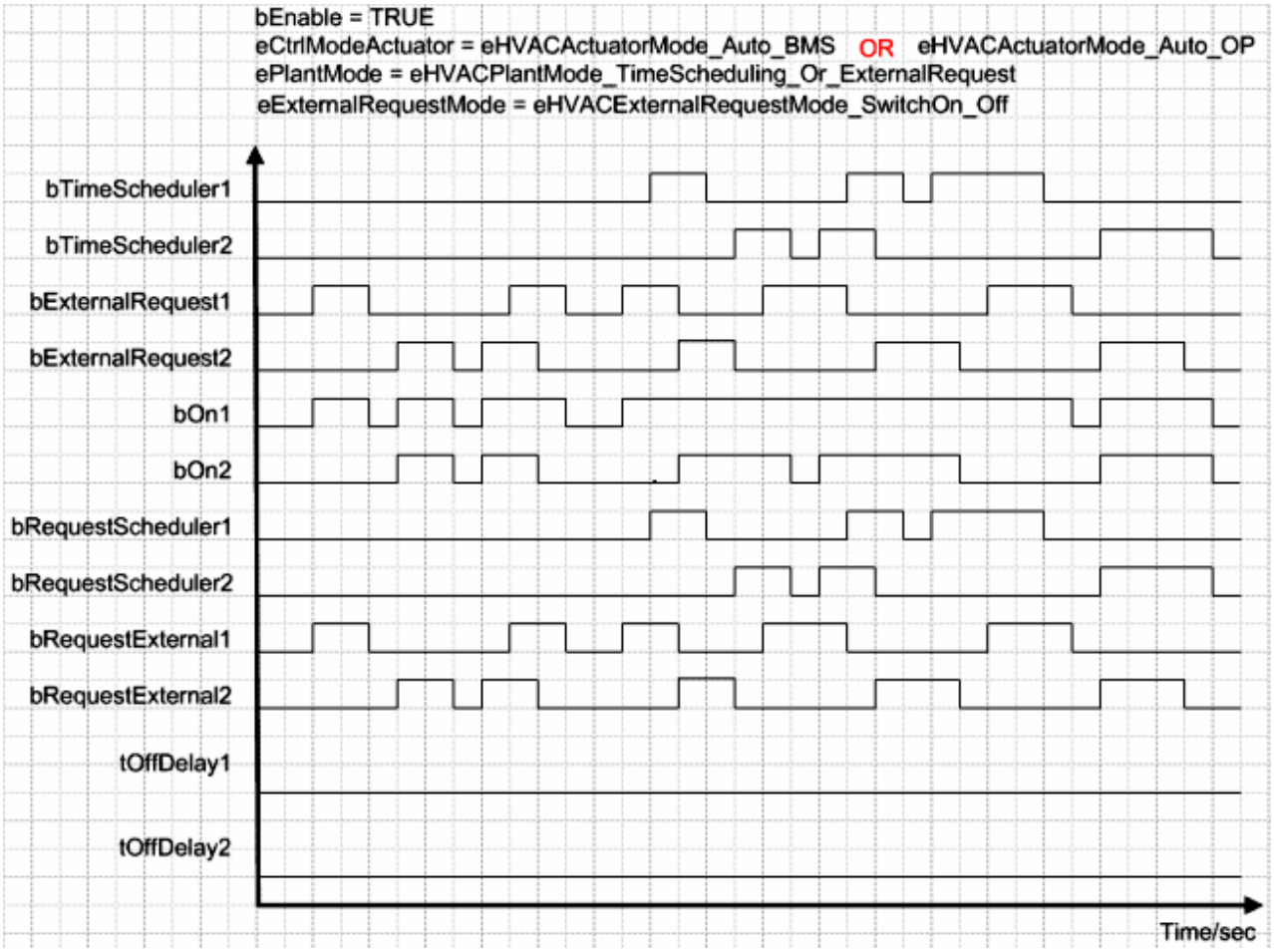
eExternalRequestMode: Mit dem Enum *eExternalRequestMode* wird die Wirkungsweise der Eingangsvariable *bExternalRequest1* und *bExternalRequest2* im Automatikbetrieb in Abhängigkeit des Enums *ePlantMode* vorgegeben.
eExternalRequestMode = *eHVACExternalRequestMode_ButtonOn_Off*: Die externe Anforderung wird nach einer steigenden Flanke von *bExternalRequest1* oder *bExternalRequest2* auf TRUE gesetzt. Eine weitere steigende Flanke setzt die Anforderung wieder auf FALSE zurück.
eExternalRequestMode = *eHVACExternalRequestMode_ButtonOffDelay*. Die externe Anforderung verlängert oder setzt die Nutzzeit der Anlage in der jeweiligen Stufe nach einer steigenden Flanke an der Eingangsvariablen *bExternalRequest1* oder *bExternalRequest2* um die eingestellte Zeit von *tOffDelay1* oder *tOffDelay2*.
eExternalRequestMode = *eHVACExternalRequestMode_SwitchOn_Off*: Die externe Anforderung ist aktiv, wenn *bExternalRequest1* oder *bExternalRequest2* = TRUE ist. Sie wird deaktiviert, wenn *bExternalRequest1* oder *bExternalRequest2* = FALSE ist.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weiter gearbeitet.
bInvalidParameter wird bei falscher Parameterangabe gesetzt.

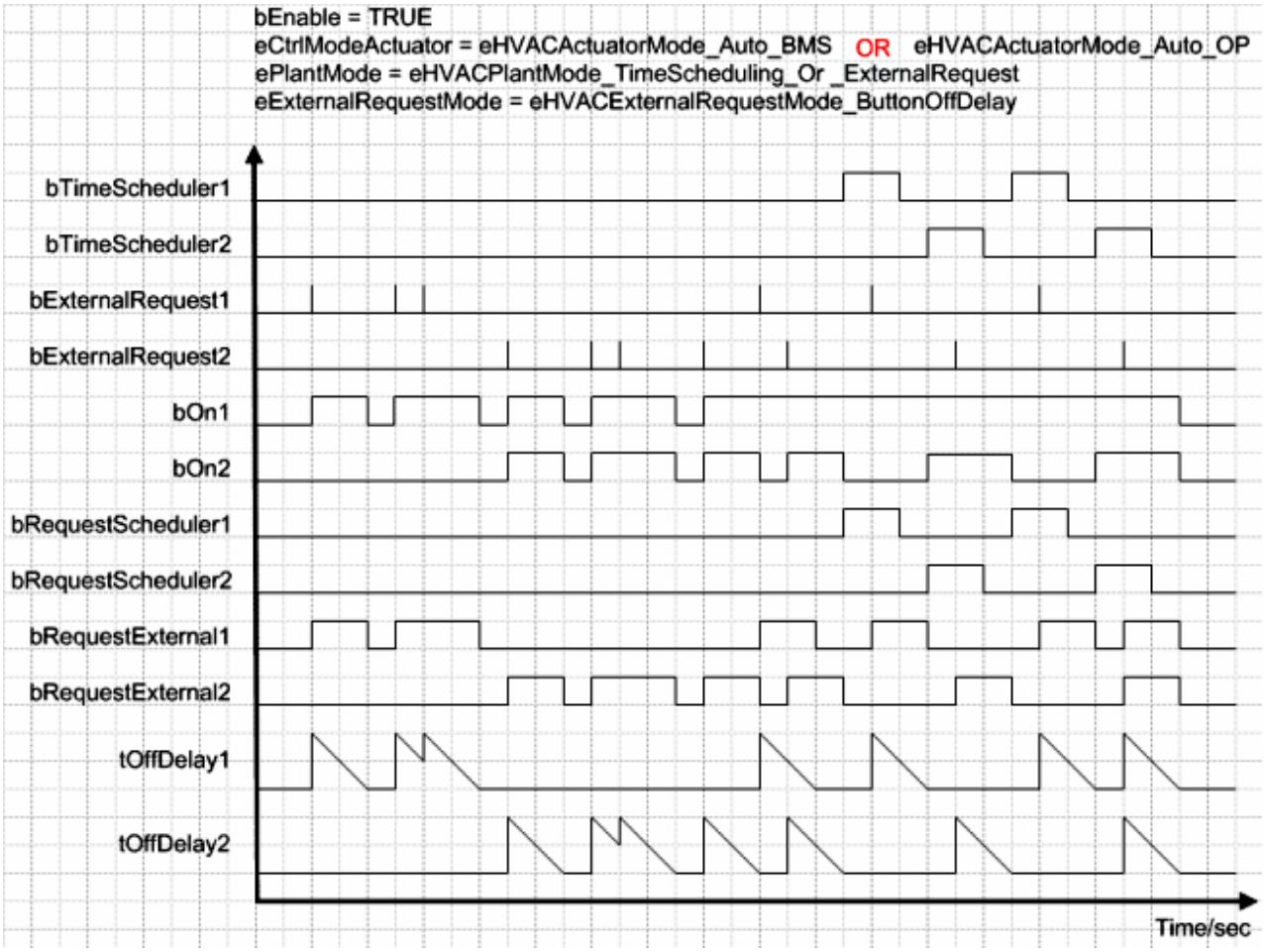
Die Variable wird persistent gespeichert. Voreingestellt auf 0.

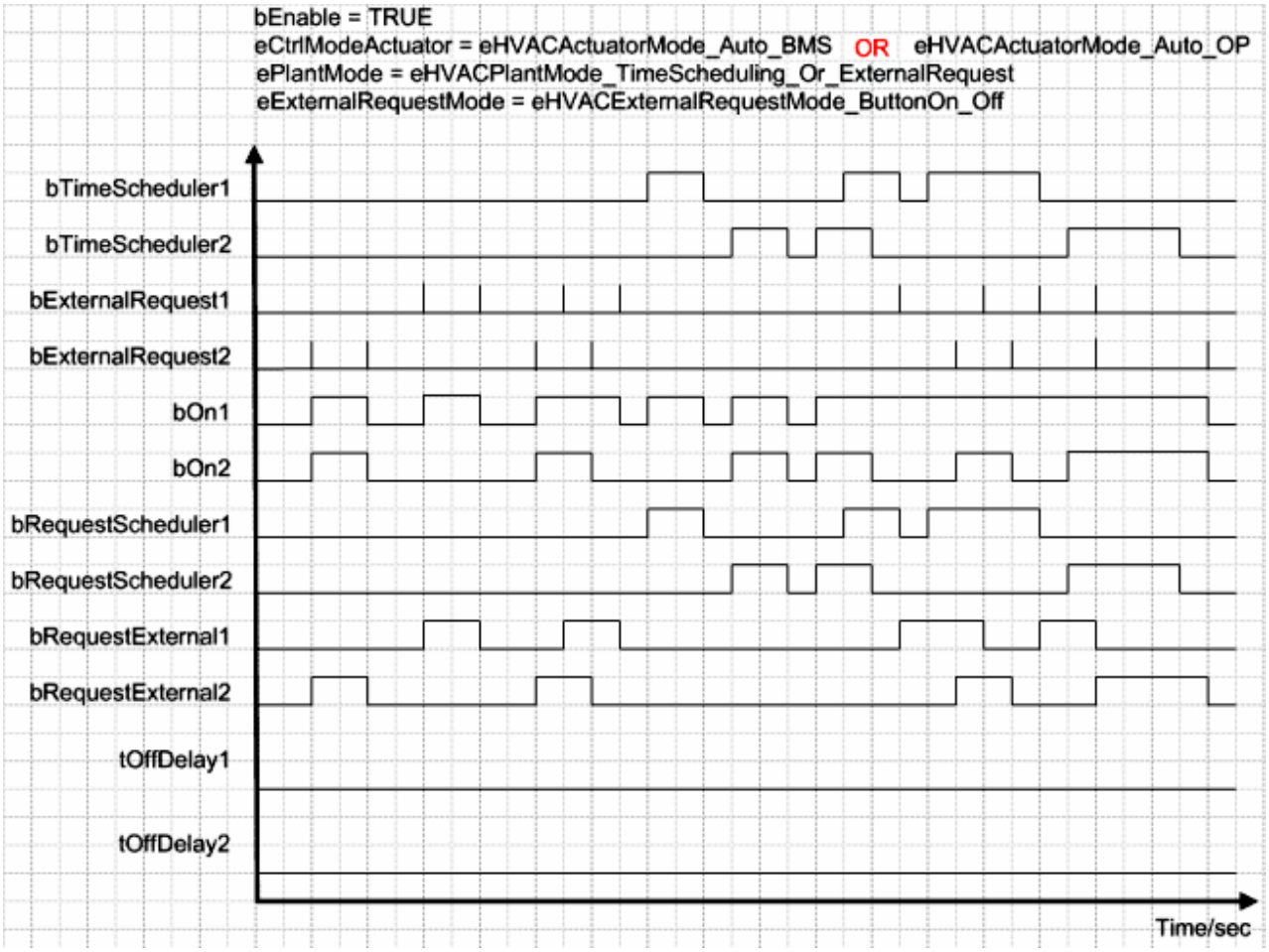
tOffDelay1: Zeitwert für die Nutzzeitverlängerung der Anlage in Stufe 1. Die Nutzzeitverlängerung kann nur aktiviert werden, wenn *eModeExternalRequest* = 2 ist. Die Variable wird persistent gespeichert. Voreingestellt auf 30min.

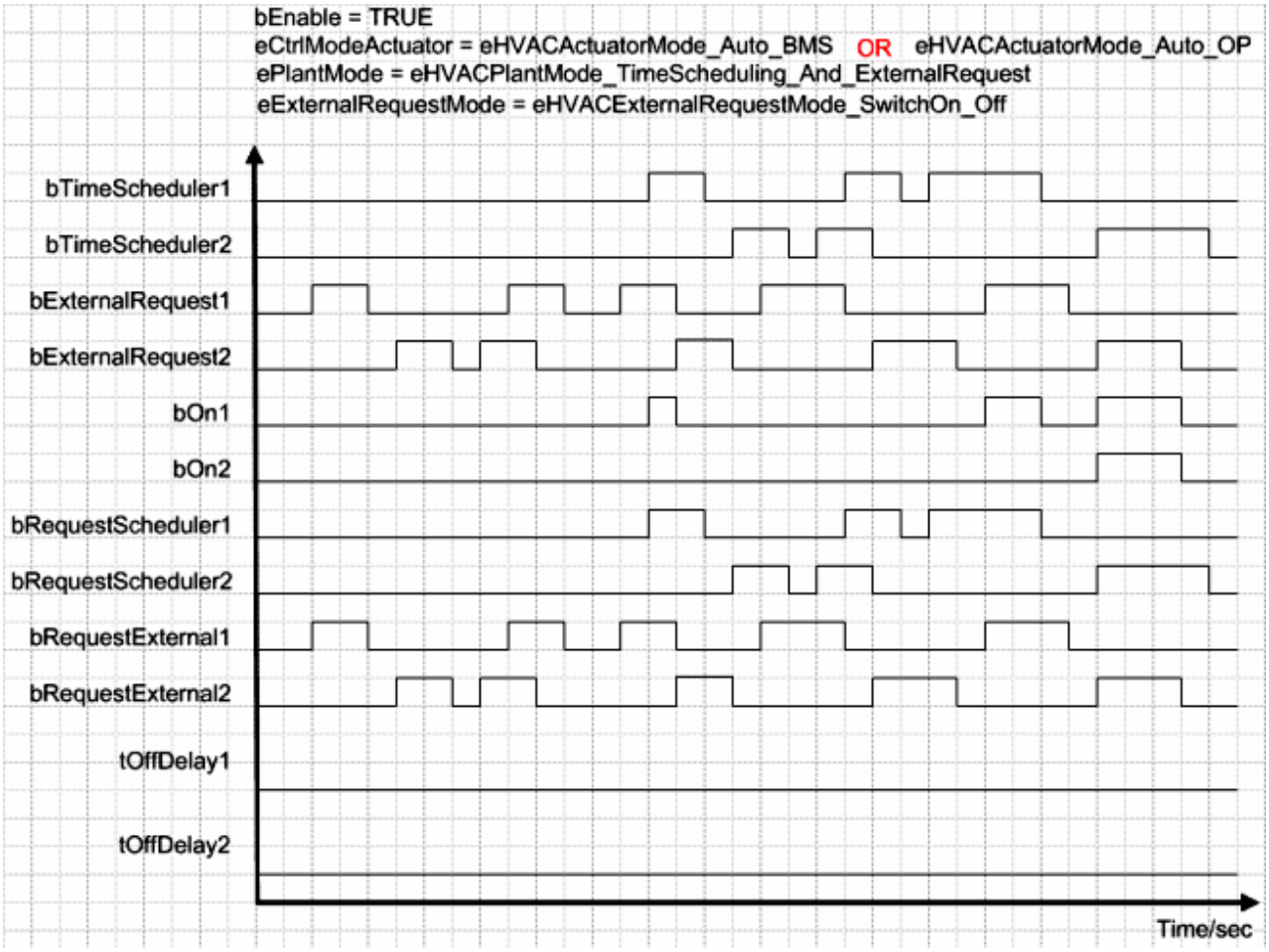
tOffDelay2: Zeitwert für die Nutzzeitverlängerung der Anlage in Stufe 2. Die Nutzzeitverlängerung kann nur aktiviert werden, wenn *eModeExternalRequest* = 2 ist. Die Variable wird persistent gespeichert. Voreingestellt auf 30min.

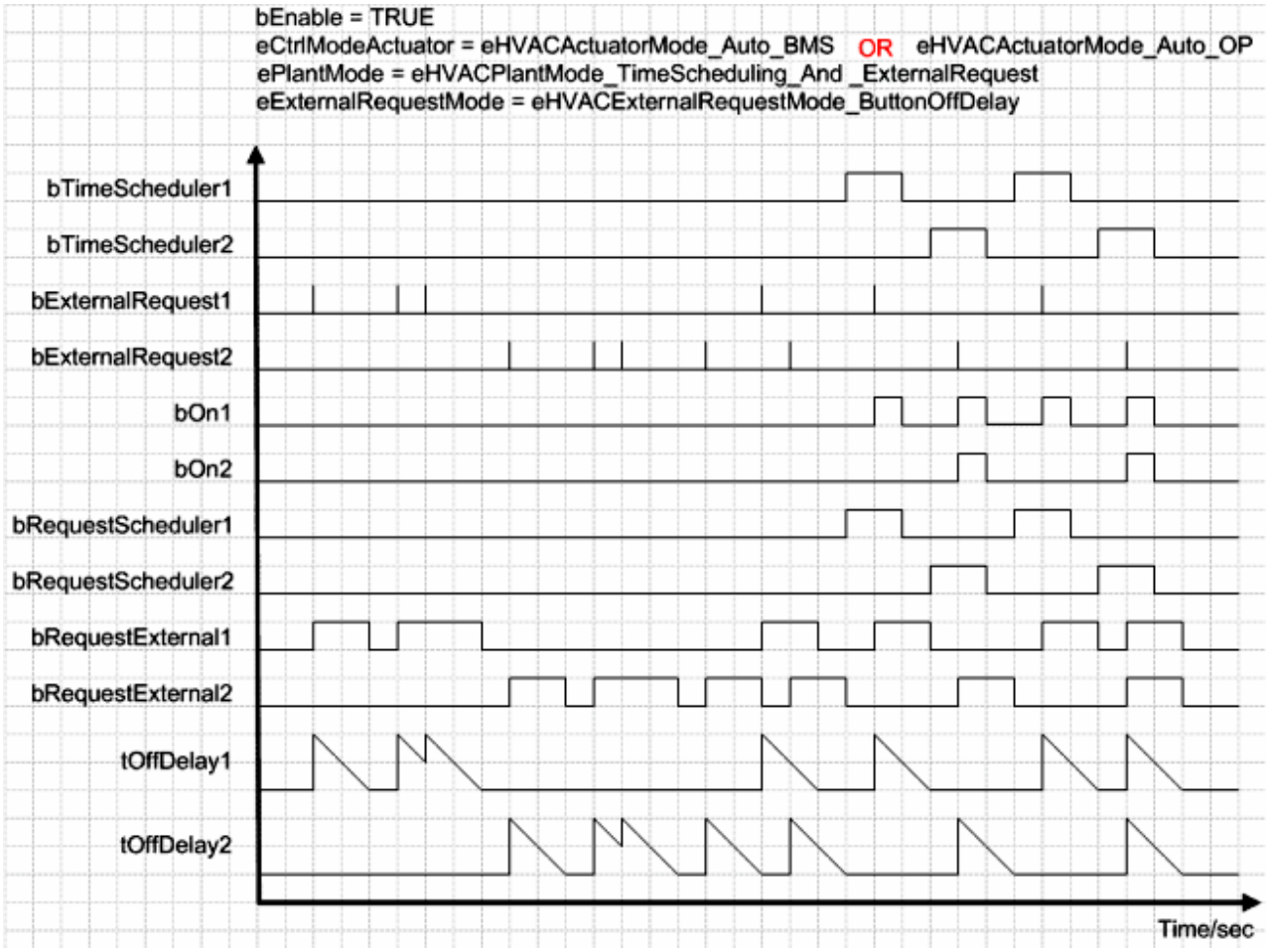
Verhalten der Ausgangsgröße

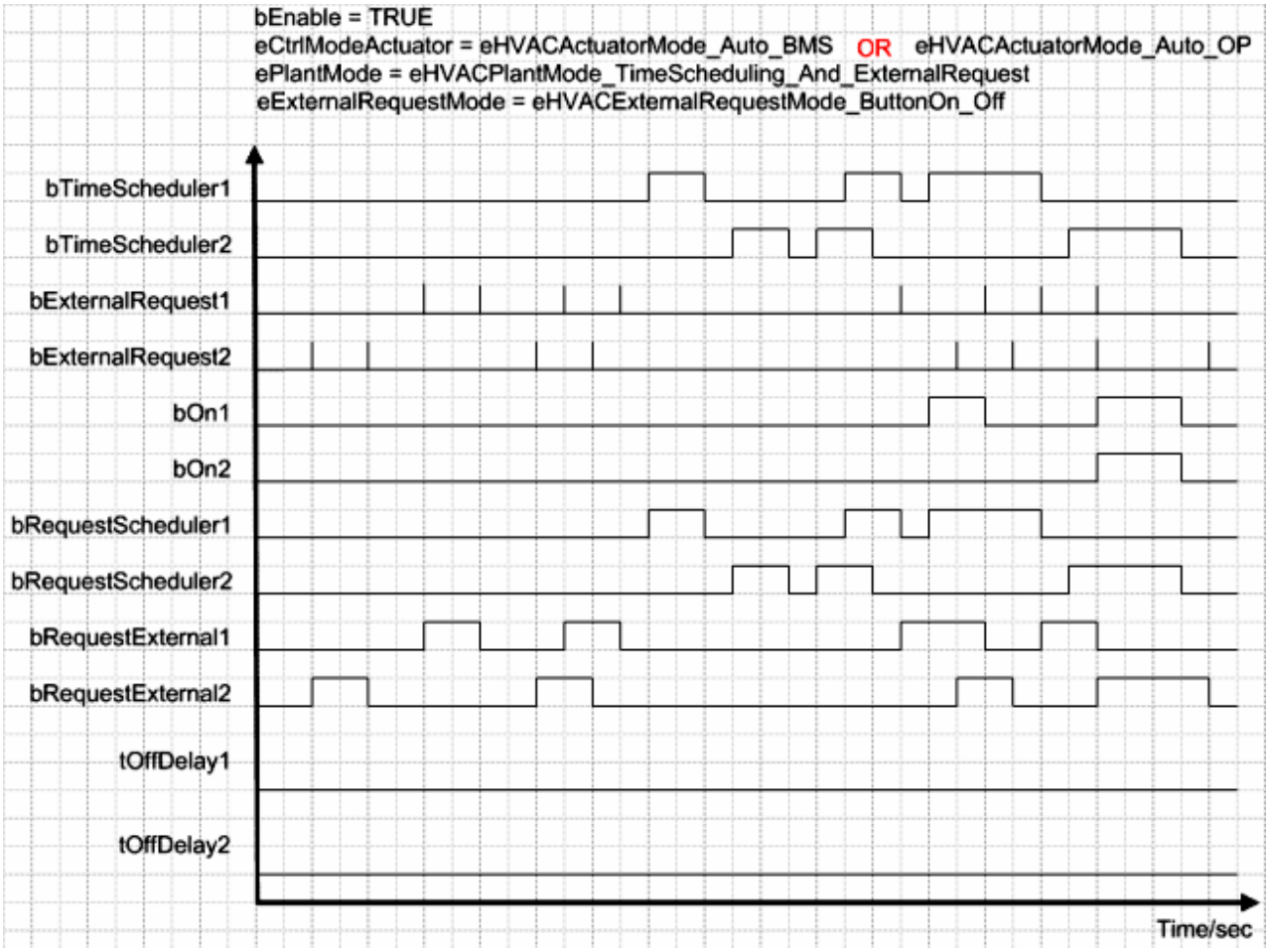


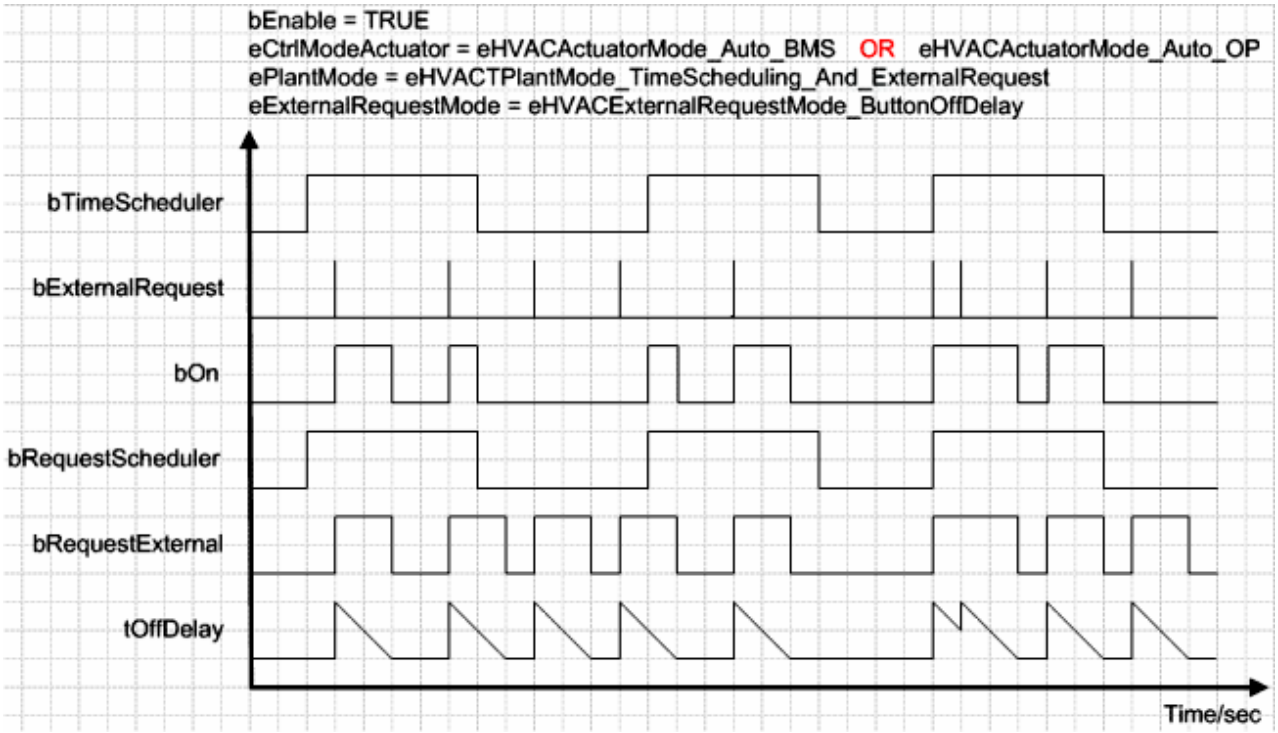

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeScheduling_Or _ExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_ButtonOffDelay
bTimeScheduler1
bTimeScheduler2
bExternalRequest1
bExternalRequest2
bOn1
bOn2
bRequestScheduler1
bRequestScheduler2
bRequestExternal1
bRequestExternal2
tOffDelay1
tOffDelay2
Time/sec

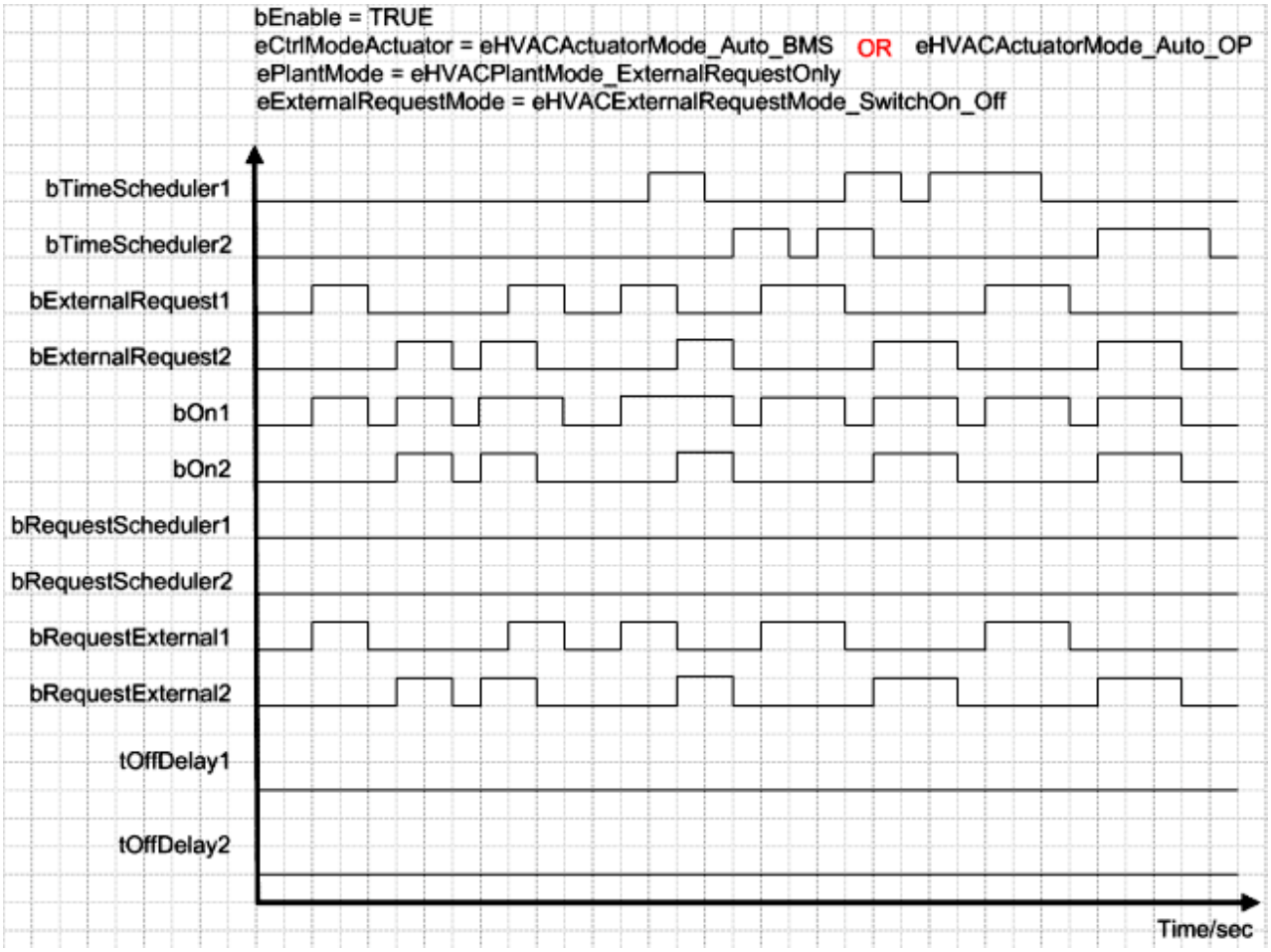

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeScheduling_Or_ExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_ButtonOn_Off
bTimeScheduler1
bTimeScheduler2
bExternalRequest1
bExternalRequest2
bOn1
bOn2
bRequestScheduler1
bRequestScheduler2
bRequestExternal1
bRequestExternal2
tOffDelay1
tOffDelay2
Time/sec

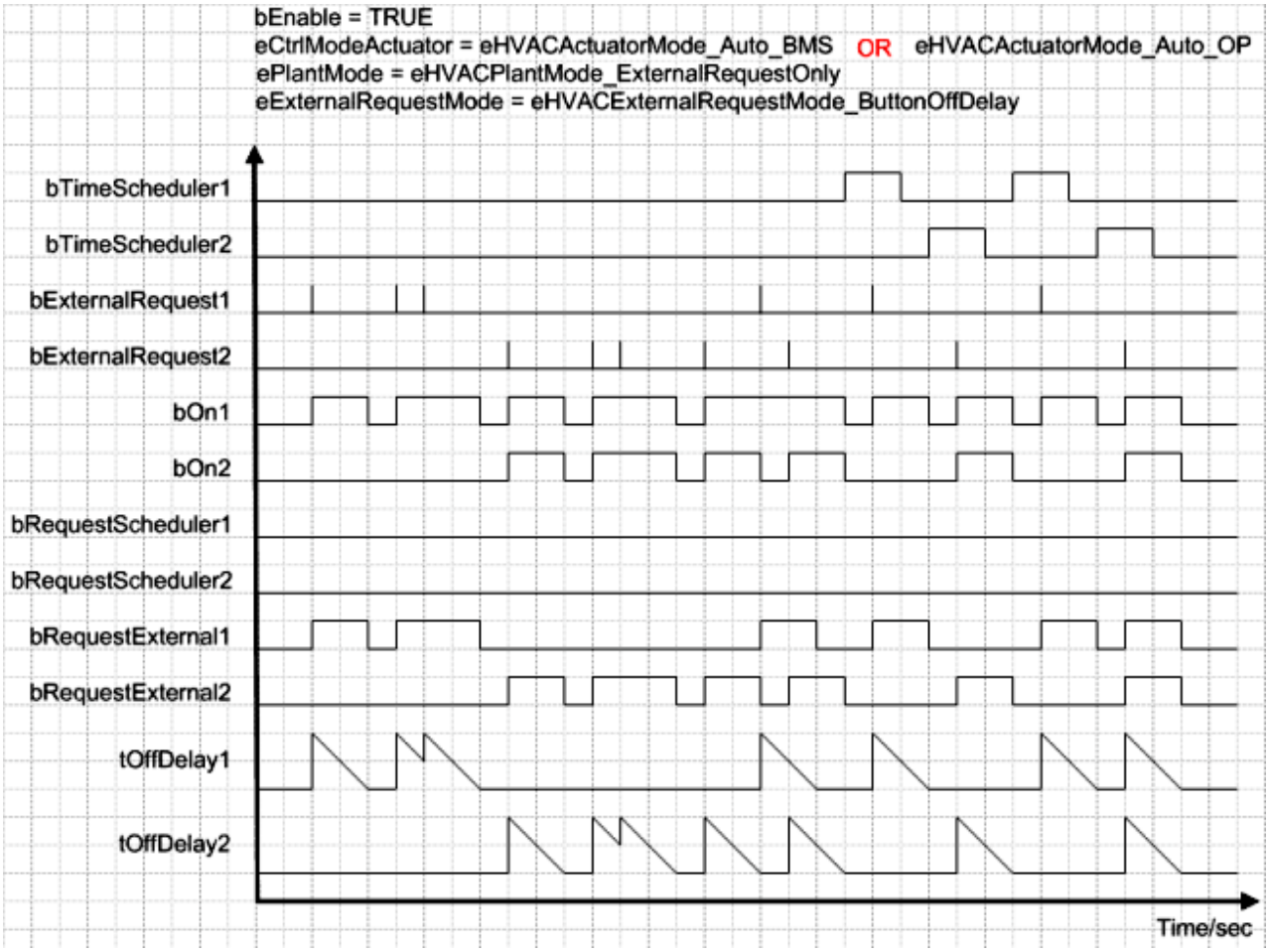

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeScheduling_And_ExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_SwitchOn_Off
bTimeScheduler1
bTimeScheduler2
bExternalRequest1
bExternalRequest2
bOn1
bOn2
bRequestScheduler1
bRequestScheduler2
bRequestExternal1
bRequestExternal2
tOffDelay1
tOffDelay2
Time/sec

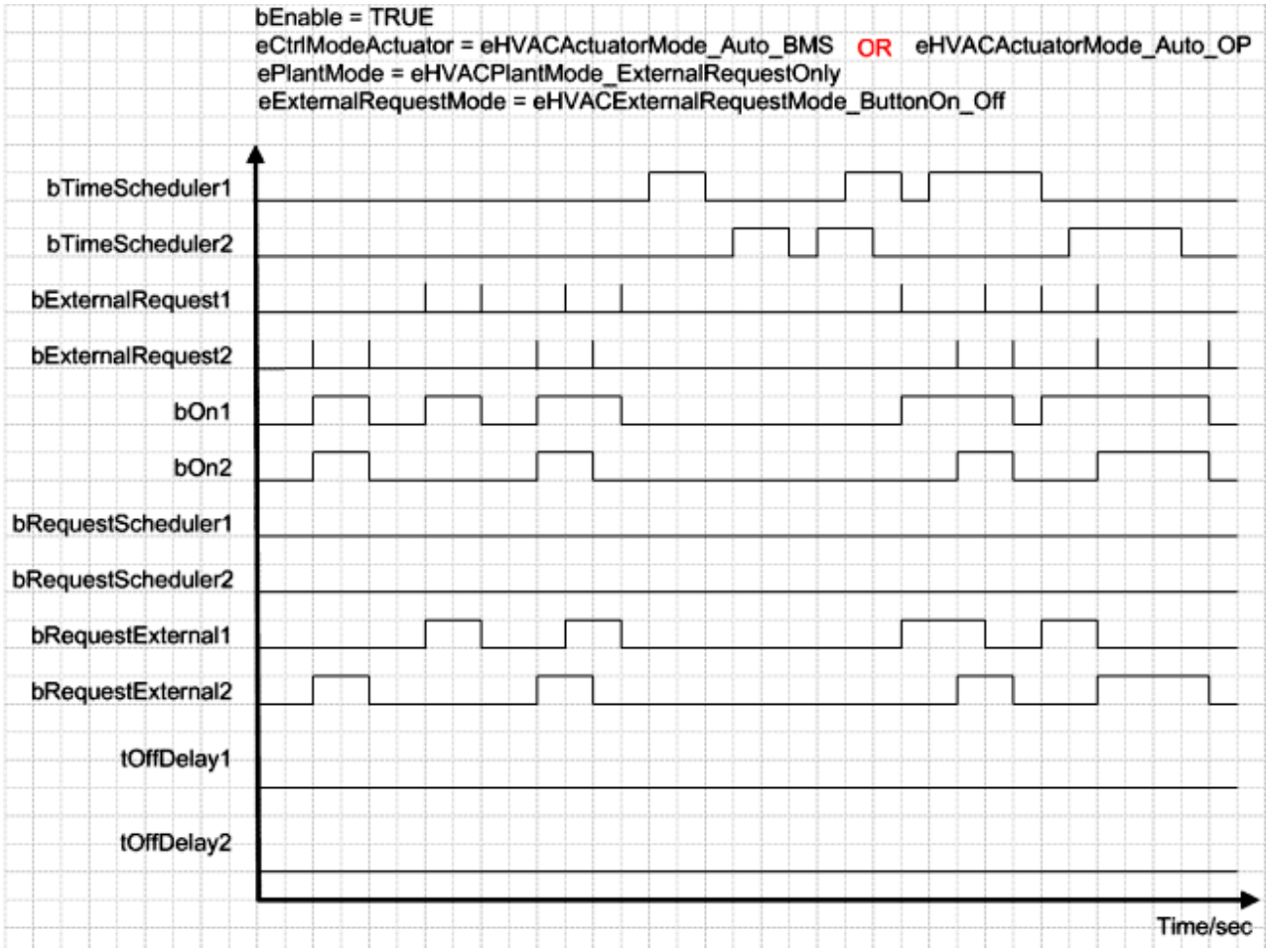

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeScheduling_And _ExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_ButtonOffDelay
bTimeScheduler1
bTimeScheduler2
bExternalRequest1
bExternalRequest2
bOn1
bOn2
bRequestScheduler1
bRequestScheduler2
bRequestExternal1
bRequestExternal2
tOffDelay1
tOffDelay2
Time/sec

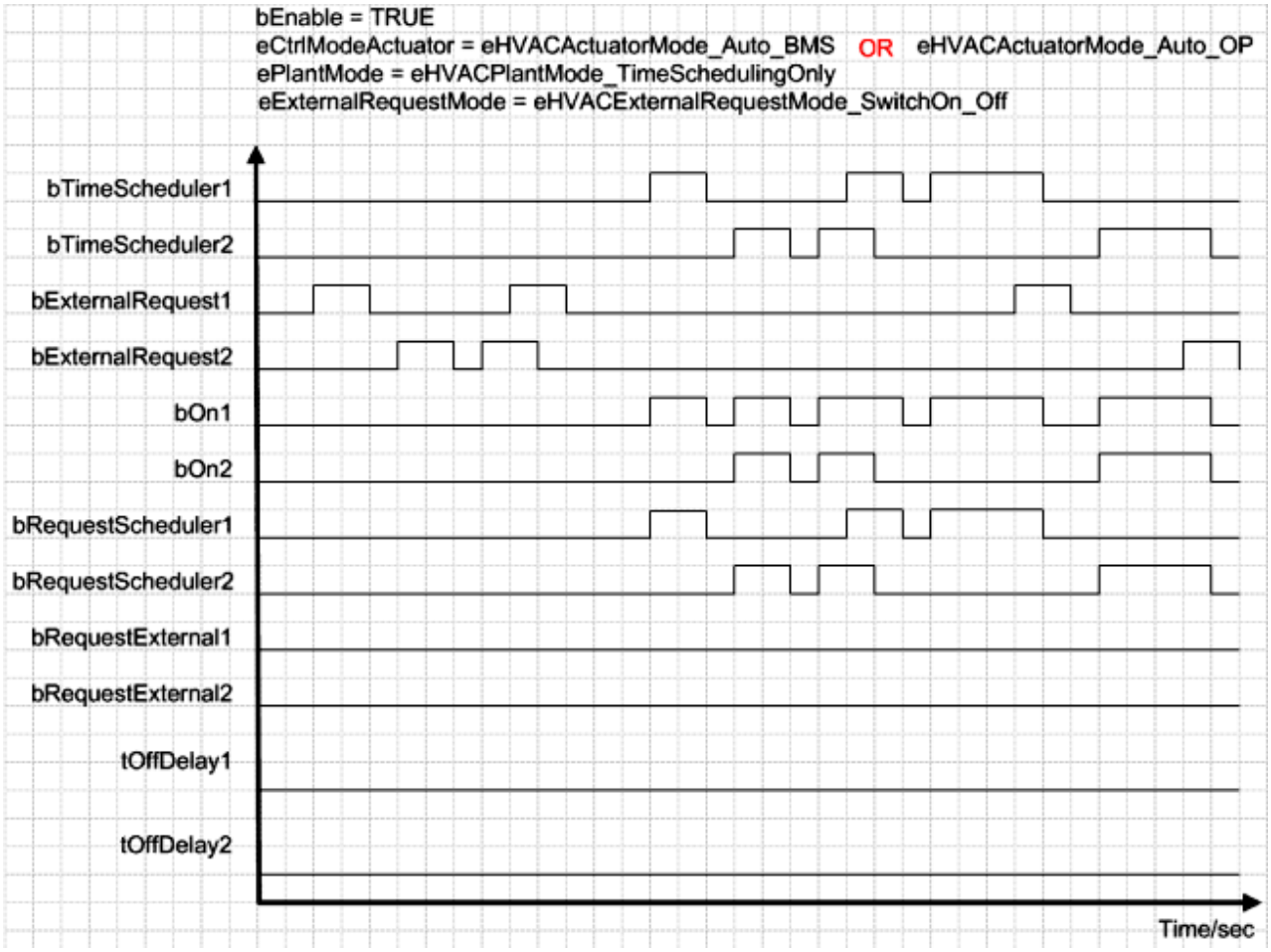

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeScheduling_And_ExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_ButtonOn_Off
bTimeScheduler1
bTimeScheduler2
bExternalRequest1
bExternalRequest2
bOn1
bOn2
bRequestScheduler1
bRequestScheduler2
bRequestExternal1
bRequestExternal2
tOffDelay1
tOffDelay2
Time/sec

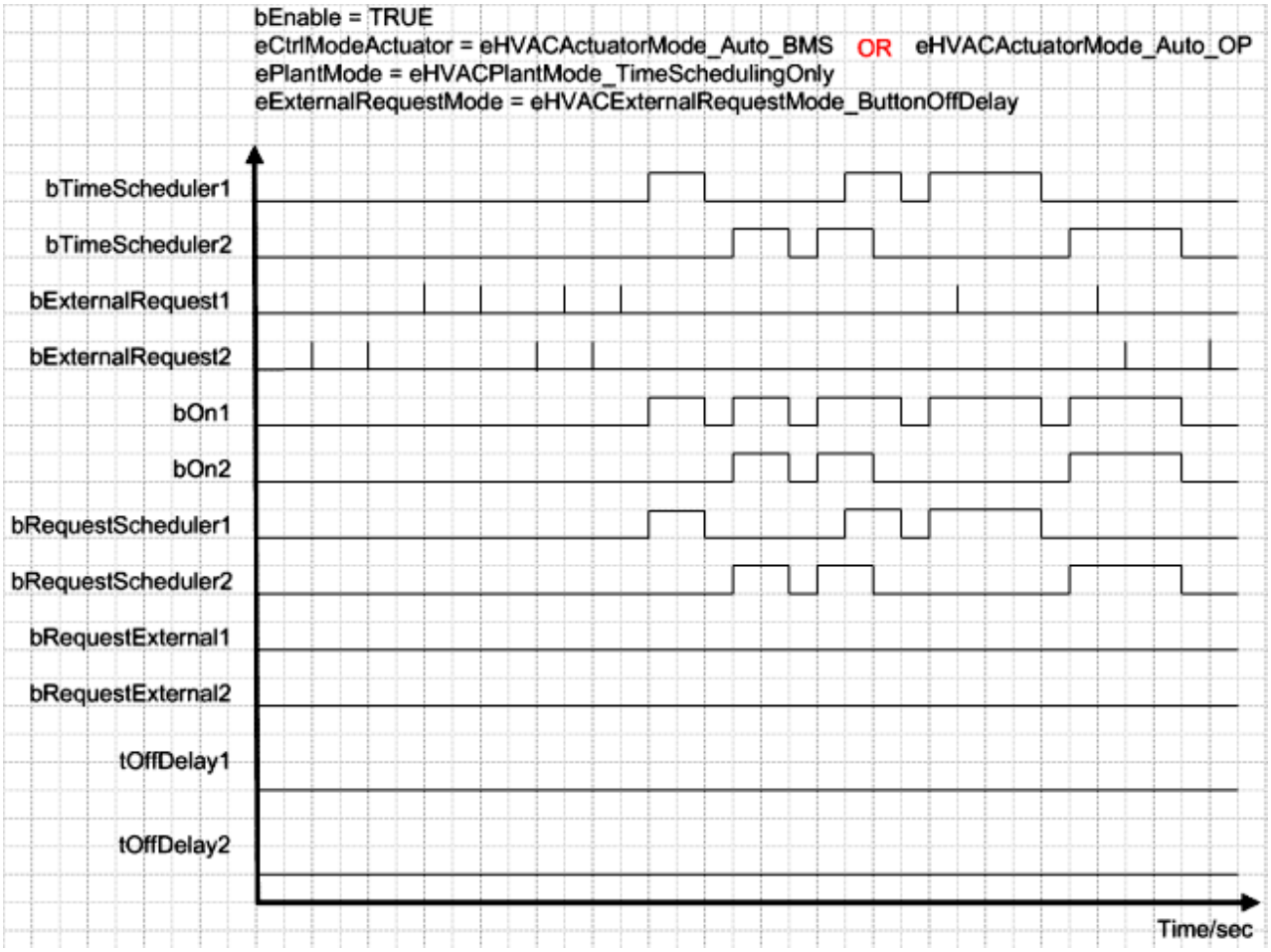

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACTPlantMode_TimeScheduling_And_ExternalRequest
eExternalRequestMode = eHVACExternalRequestMode_ButtonOffDelay
bTimeScheduler
bExternalRequest
bOn
bRequestScheduler
bRequestExternal
tOffDelay
Time/sec

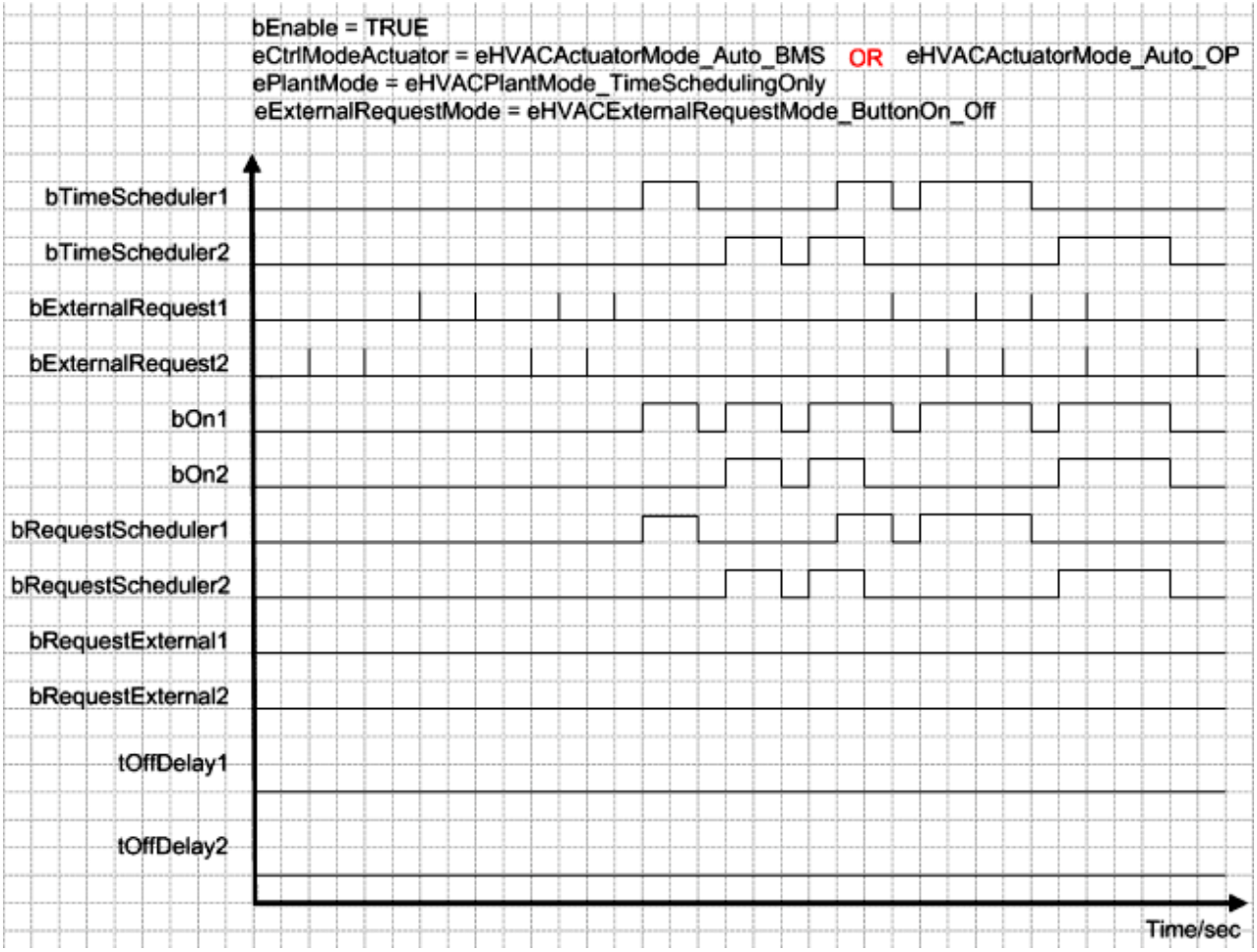

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_ExternalRequestOnly
eExternalRequestMode = eHVACExternalRequestMode_SwitchOn_Off
bTimeScheduler1
bTimeScheduler2
bExternalRequest1
bExternalRequest2
bOn1
bOn2
bRequestScheduler1
bRequestScheduler2
bRequestExternal1
bRequestExternal2
tOffDelay1
tOffDelay2
Time/sec


bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_ExternalRequestOnly
eExternalRequestMode = eHVACExternalRequestMode_ButtonOffDelay
bTimeScheduler1
bTimeScheduler2
bExternalRequest1
bExternalRequest2
bOn1
bOn2
bRequestScheduler1
bRequestScheduler2
bRequestExternal1
bRequestExternal2
tOffDelay1
tOffDelay2
Time/sec


bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_ExternalRequestOnly
eExternalRequestMode = eHVACExternalRequestMode_ButtonOn_Off
bTimeScheduler1
bTimeScheduler2
bExternalRequest1
bExternalRequest2
bOn1
bOn2
bRequestScheduler1
bRequestScheduler2
bRequestExternal1
bRequestExternal2
tOffDelay1
tOffDelay2
Time/sec


bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeSchedulingOnly
eExternalRequestMode = eHVACExternalRequestMode_SwitchOn_Off
bTimeScheduler1
bTimeScheduler2
bExternalRequest1
bExternalRequest2
bOn1
bOn2
bRequestScheduler1
bRequestScheduler2
bRequestExternal1
bRequestExternal2
tOffDelay1
tOffDelay2
Time/sec

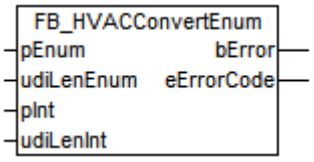

bEnable = TRUE
eCtrlModeActuator = eHVACActuatorMode_Auto_BMS OR eHVACActuatorMode_Auto_OP
ePlantMode = eHVACPlantMode_TimeSchedulingOnly
eExternalRequestMode = eHVACExternalRequestMode_ButtonOffDelay
bTimeScheduler1
bTimeScheduler2
bExternalRequest1
bExternalRequest2
bOn1
bOn2
bRequestScheduler1
bRequestScheduler2
bRequestExternal1
bRequestExternal2
tOffDelay1
tOffDelay2
Time/sec



Dokumente hierzu

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

3.6.9 FB_HVACConvertEnum



Anwendung

Der Funktionsbaustein konvertiert ein Enum in einen Integer-Wert und umgekehrt. Diese Konvertierung ist speziell für Enums geeignet, die als VAR_IN_OUT-Variablen an Funktionsbausteinen verwendet werden.

VAR_INPUT

```
pEnum           : UDINT;
udiLenEnum      : UDINT;
pInt            : UDINT;
udiLenInt       : UDINT;
```

pEnum: Adresse des Enums, welches konvertiert werden soll. Die Adresse wird mit dem ADR-Operator ermittelt.

udiLenEnum: Anzahl der Bytes aus dem der Datentyp Enum besteht. Die Anzahl wird mit dem SIZEOF-Operator ermittelt.

pInt: Adresse der Integer-Variable, welche konvertiert werden soll. Die Adresse wird mit dem ADR-Operator ermittelt.

udiLenInt: Anzahl der Bytes aus dem der Datentyp Integer besteht. Die Anzahl wird mit dem SIZEOF-Operator ermittelt.

VAR_OUTPUT

```
bError          : BOOL;
eErrorCode      : E_HVACErrorCodes;
```

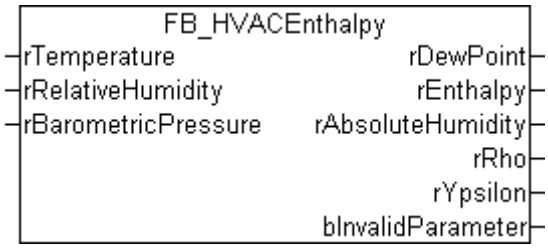
bError: Der Ausgang signalisiert mit einem TRUE, dass ein Fehler anliegt. (Die Anzahl der Bytes aus denen die Datentypen Integer oder Enum bestehen ist falsch.)

eErrorCode: Liefert bei einem gesetzten *bError*-Ausgang die Fehlernummer [▶ 529]. Folgende Fehler können in diesem Funktionsbaustein vorkommen: *eHVACErrorCodes_Error_LEN_Int* (43), *eHVACErrorCodes_Error_LEN_Enum* (44)

Anwendungsbeispiel

Download	Benötige Bibliothek
https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659726219.zip	**TcHVAC.lib**

3.6.10 FB_HVACEnthalpy



Anwendung

Mit diesem Funktionsbaustein wird der Taupunkt, die spezifische Enthalpie und die absolute Feuchte berechnet. Für die Berechnung der Größen werden die Temperatur, die relative Feuchte und der barometrische Luftdruck benötigt.
Die Enthalpie ist ein Maß für die Energie eines thermodynamischen Systems.

VAR_INPUT

```
rTemperature           : REAL;          [°C ]
rRelativeHumidity      : REAL;          [ % ]       1..100
rBarometricPressure    : REAL;          [hPa]
```

rTemperature: Eingang für den Temperaturwert, in Grad Celsius .

rRelativeHumidity: Eingang für die relative Feuchte, in Prozent. Der Wert muss größer oder gleich 1 sein.

rBarometricPressure: Eingang für den Luftdruck, in Hekto-Pascal.

VAR_OUTPUT

```
rDewPoint          : REAL;        [°C ]
rEnthalpy          : REAL;        [kJ/kg]
rAbsoluteHumidity  : REAL;        [kg/kg]
rRho               : REAL;        [kg/m³]
rYpsilon           : REAL;        [m³/kg]
bInvalidParameter  : BOOL;
```

rDewPoint: Taupunkt [°C].

rEnthalpy: Enthalpie [kJ / kg].

rAbsoluteHumidity: Absolute Feuchte [kg/kg]. Um die absolute Feuchte in [g/kg] zu erhalten muss das Ergebnis mit 1000 multipliziert werden.

rRho: Dichte feuchter Luft, welche die Gemischmasse bezogen auf das Volumen angibt. Die Einheit ist [kg Gemisch / m³].

rYpsion: Spezifisches Volumen, das auf 1 kg trockene Luft bezogen ist. Die Einheit ist [m³ / kg trockene Luft].

bInvalidParameter: Wird auf TRUE gesetzt, wenn die Eingangsvariable *rRelativeHumidity* oder *rBarometricPressure* kleiner gleich Null ist .

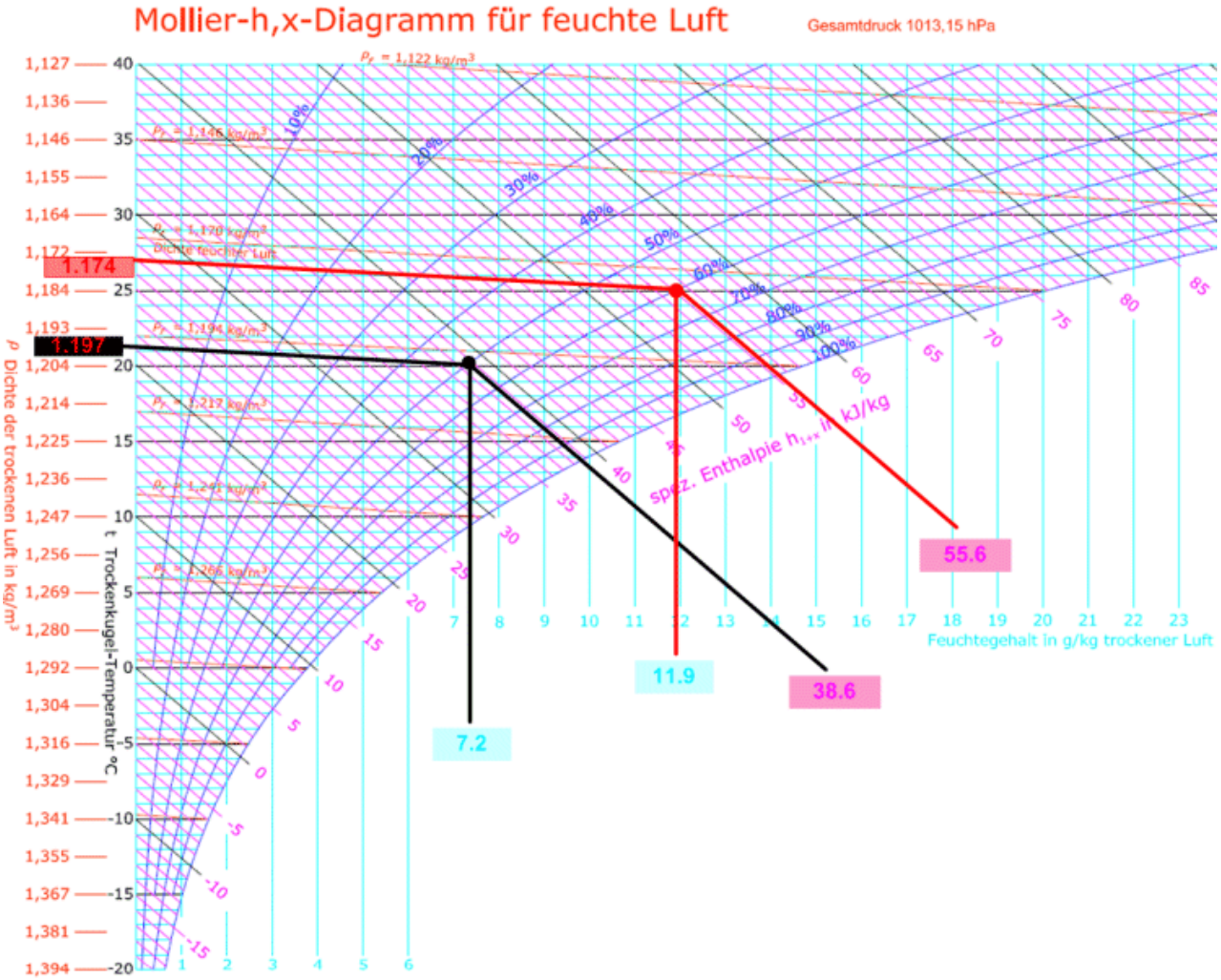


Abb. 1 h,x Diagramm [www.air2000.de]

Beispiel 1:

rTemperature:= 20 °C
rRelativeHumidity:= 50 %
rBarometricPressure:= 1013.15 hPa

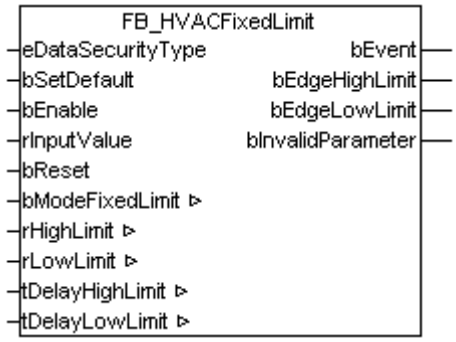
Ergebnisse vom Funktionsbaustein: rDewPoint:= 9.4 °C
rEnthalpy:= 38.6 kJ/kg
rAbsoluteHumidity:= 0.0072 kg/kg , umgerechnet in g/kg ==> 7.2 g/kg

Beispiel 2:

rTemperature:= 25 °C
rRelativeHumidity:= 60 %
rBarometricPressure:= 1013.15 hPa

Ergebnisse vom Funktionsbaustein: rDewPoint:= 17 °C
rEnthalpy:= 55.6 kJ/kg
rAbsoluteHumidity:= 0.0119 kg/kg , umgerechnet in g/kg ==> 11.9 g/kg

3.6.11 FB_HVACFixedLimit



Anwendung

Dieser Funktionsbaustein stellt einen Grenzwertschalter dar, dessen Ausgangssignal in Abhängigkeit der Zeitverzögerungen *tDealyHighLimit* / *tDelayLowLimit,* des Modus *bModeFixedLimit,* der Grenzwerte *rHighLimit* / *rLowLimit* und des Eingangssignals *rInputValue* geschaltet wird.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
bEnable          : BOOL;
rInputValue      : REAL;
bReset           : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Mit einer steigenden Flanke an dieser Eingangsvariable werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Freigabe des Bausteins, wenn *bEnable* = TRUE ist.

rInputValue: Eingangssignal

bReset: Eingang zur Quittierung der falschen Parameterangabe über eine steigende Flanke.

VAR_OUTPUT

```
bEvent           : BOOL;
bEdgeHighLimit   : BOOL;
bEdgeLowLimit    : BOOL;
bInvalidParameter: BOOL;
```

bEvent: Ausgangssignal, welches in Abhängigkeit der Zeitverzögerungen *tDelayHighLimit* / *tDelayLowLimit,* des Modus *bModeFixedLimit*, des Eingangssignals *rInputValue* und der Grenzwerte *rHighLimit* / *rLowLimit* geschaltet wird.

bEdgeHighLimit: Ist für einen SPS-Zyklus TRUE nach dem das Eingangssignal *rInputValue* die Variable *rHighLimit* für die Zeitdauer von *tDelayHighLimit* überschritten hat.

bEdgeLowLimit: Ist für einen SPS-Zyklus TRUE nach dem das Eingangssignal *rInputValue* die Variable *rLowLimit* für die Zeitdauer von *tDelayLowLimit* unterschritten hat.

bInvalidParameter: Zeigt an, dass ein falscher Parameter an einer der Variablen *rHighLimit, rLowLimit, tDelayHighLimit* oder *tDelayHighLimit* anliegt. Eine falsche Parameterangabe führt nicht zum Stillstand des Bausteines, siehe Beschreibung der Variablen. Nach Behebung der falschen Parameterangabe muss die Meldung *bInvalidParameter* mit *bReset* quittiert werden.

VAR_IN_OUT

```
bModeFixedLimit  : BOOL;
rHighLimit       : REAL;
rLowLimit        : REAL;
tDelayHighLimit  : TIME;
tDelayLowLimit   : TIME;
```

bModeFixedLimit: Ist *bModeFixedLimit* = TRUE, dann wird die Ausgangsvariable *bEvent* TRUE, wenn das Eingangssignal *rInputValue* die Variable *rHighLimit* für die Zeitdauer von *tDelayHighLimit* überschritten hat. Ist *bEvent* = TRUE, dann ist *bEdgeHighLimit* für einen SPS-Zyklus gesetzt. Unterschreitet das Eingangssignal *rInputValue* die Variable *rLowLimit* für die Zeitdauer von *tDelayLowLimit*, dann wird *bEvent* FALSE und *bEdgeLowLimit* für einen SPS-Zyklus gesetzt.
Ist *bModeFixedLimit* = FALSE, dann wird die Ausgangsvariable *bEvent* TRUE, wenn das Eingangssignal *rInputValue* die Variable *rLowLimit* für die Zeitdauer von *tDelayLowLimit* unterschritten hat. Ist *bEvent* = TRUE, dann ist *bEdgeLowLimit* für einen SPS-Zyklus gesetzt. Überschreitet das Eingangssignal *rInputValue* die Variable *rHighLimit* für die Zeitdauer von *tDelayHighLimit*, dann wird *bEvent* FALSE und *bEdgeHighLimit* für einen SPS-Zyklus gesetzt.

Die Variable wird persistent gespeichert. Voreingestellt auf 1.

rHighLimit: Oberer Grenzwert.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden, der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt, der Funktionsbaustein arbeitet normal weiter.

Die Variable wird persistent gespeichert. Voreingestellt auf 22.

rLowLimit: Unterer Grenzwert.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden, der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt, der Funktionsbaustein arbeitet normal weiter.

Die Variable wird persistent gespeichert. Voreingestellt auf 18.

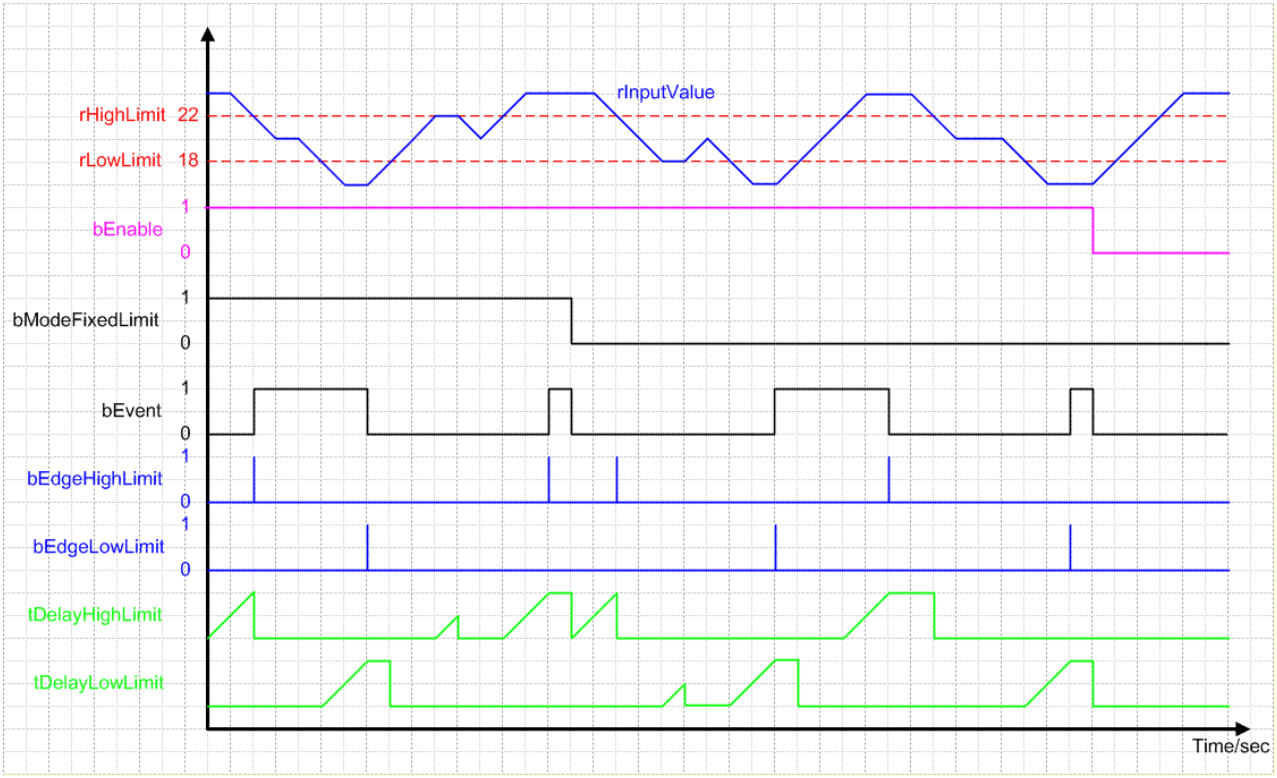
tDelayHighLimit: Schaltverzögerung [s] beim Überschreiten des oberen Grenzwertes.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden, der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt, der Funktionsbaustein arbeitet normal weiter.

Die Variable wird persistent gespeichert. Voreingestellt auf 1s.

tDelayLowLimit: Schaltverzögerung [s] beim Unterschreiten des unteren Grenzwertes.
Liegt ein nicht korrekter Variablenwert an, dann wird, wenn vorhanden, der letzte, gültige Variablenwert genommen. Wenn kein gültiger, letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt, der Funktionsbaustein arbeitet normal weiter.

Die Variable wird persistent gespeichert. Voreingestellt auf 1s.

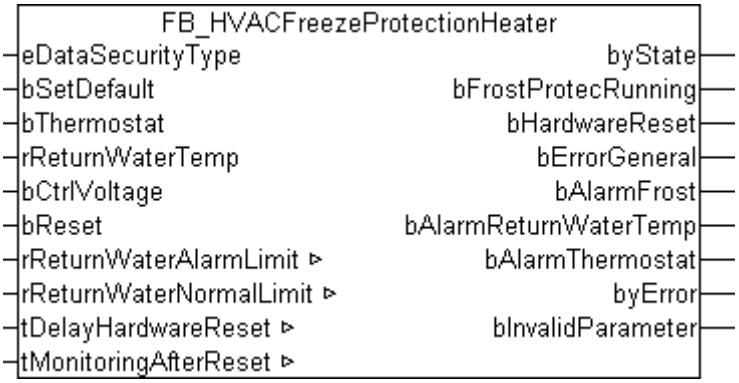
Verhalten der Ausgangsgröße



Dokumente hierzu

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

3.6.12 FB_HVACFreezeProtectionHeater



Anwendung

Dieser Funktionsbaustein ist für die Frostschutzüberwachung eines Lufterhitzers in einer Klimaanlage. Die Frostschutzüberwachung erfolgt sowohl luftseitig über einen Frostschutzthermostaten als auch wasserseitig über einen Temperaturfühler im Rücklauf des Erhitzers.

Das Frostschutzprogramm wird aktiv, wenn die Rücklauftemperatur des Lufterhitzers *rReturnWaterTemp* kleiner als der Grenzwert *rReturnWaterAlarmLimit* oder der Eingang *bThermostat* FALSE = Störung ist. Bei jedem dieser Störungen wird die jeweilige Meldung *bAlarmReturnWaterTemp* und/oder *bAlarmThermostat* am Ausgang des Funktionsbausteins TRUE. Ebenfalls wird der Ausgang *bAlarmFrost* TRUE und die Aktivierung des Frostschutzprogramms mit einem TRUE an der Ausgangsvariablen *bFrostProtectionRunning* signalisiert.

Um beim Ansprechen des Frostschutzes ein Einfrieren des Lufterhitzers zu vermeiden muss die Erhitzerpumpe eingeschaltet und das Erhitzerventil zwangsweise aufgefahren werden. Diese Funktion ist in vielen Anlagen nicht nur softwareseitig, sondern aus Sicherheitsgründen im Schaltschrank zusätzlich hardwaremäßig durch eine Relaisschaltung realisiert. Das Frostschutzrelais ist im Normalfall in Selbsthaltung, da die Frostschutzschaltung nach dem Ruhestromprinzip aufgebaut ist. Nach einer Störung wird das Frostschutzrelais meistens durch einen Reset-Taster am Schaltschrank wieder in seine Selbsthaltung zurück gebracht. Automatisch wird diese Funktion durch einen kurzen Impuls am Ausgang *bHardwareReset* realisiert. Die Bedingung dafür ist, dass der Frostschutzthermostat wieder in den Normalzustand zurückgegangen ist, die Temperatur im Rücklauf des Erhitzers den Wert von *rReturnWaterNormalLimit* erreicht hat und die Zeit von tDelayFirstWarning abgelaufen ist.

Nachdem die Anlage nach einer automatischen Quittierung wieder angelaufen ist, wird der Timer *tMonitoringAfterReset* gestartet.
Erfolgt innerhalb dieser Zeit eine erneute Frostwarnung wird der Ausgang *bAlarmFrost* wieder TRUE. Das Rücksetzen des Frostalarms ist nur mit einer steigenden Flanke am Eingang *bReset* möglich.

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bSetDefault       : BOOL;
bThermostat       : BOOL;
rReturnWaterTemp  : REAL;
bCtrlVoltage      : BOOL;
bReset            : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bThermostat: Signal vom Frostschutzthermostaten. Im Normalfall TRUE.

rReturnWaterTemp: Wassertemperatur im Rücklauf des Lufterhitzers.

bCtrlVoltage: Steuerspannungsüberwachung. Steuerspannung vorhanden *bCtrlVoltage* = TRUE.

bReset: Eingang zur Quittierung der Störung.

VAR_OUTPUT

```
byState              : BYTE;
bFrostProtecRunning  : BOOL;
bHardwareReset       : BOOL;
bErrorGeneral        : BOOL;
bAlarmFrost          : BOOL;
bAlarmReturnWaterTemp: BOOL;
bAlarmThermostat     : BOOL;
byError              : BYTE;
bInvalidParameter    : BOOL;
```

byState: Ausgabe des Status als Byte.

byState.0 := bFrostProtecRunning;
byState.1 := bHardwareReset;
byState.2 := bThermostat;
byState.7 := bCtrlVoltage;

bFrostProtecRunning: TRUE, wenn das Frostschutzprogramm aktiv ist.

bHardwareReset: Ausgang, der mit einem Impuls von 1s signalisiert, dass das Frostschutzrelais nach einer Störung sich in der Selbsthaltung befindet.

bErrorGeneral: Es liegt allgemein ein Fehler an.

bAlarmFrost: TRUE, wenn Frostwarnung anliegt.

bAlarmReturnWaterTemp: Die Grenze der Wassertemperatur im Rücklauf des Lufterhitzers wurde unterschritten.

bAlarmThermostat: TRUE, wenn ein Fehler am Frostschutzthermostaten ist.

byError: Ausgabe der Fehler als Byte.

byError.1 := bInvalidParameter;
byError.2 := bErrorGeneral;
byError.3 := bAlarmFrost;
byError.4 := bAlarmReturnWaterTemp;
byError.5 := bAlarmThermostat;

bInvalidParameter: Zeigt an, dass ein falscher Eingangsparameter anliegt. *bInvalidParameter* muss mit *bReset* quittiert werden.

VAR_IN_OUT

```
rReturnWaterAlarmLimit  : REAL;
rReturnWaterNormalLimit : REAL;
tDelayHardwareReset     : TIME;
tMonitoringAfterReset   : TIME;
```

rReturnWaterAlarmLimit: Unterhalb eines Temperaturwertes im Rücklauf des Lufterhitzers von *rReturnWaterAlarmLimit* wird der Frostalarm aktiv (0°C..22°C). Die Variable wird persistent gespeichert.

rReturnWaterNormalLimit: Der Lufterhitzer geht wieder in den Normalzustand, wenn nach einer Temperaturunterschreitung die Temperatur wieder über *bReturnWaterNormalLimit* angestiegen ist (0°C..70°C). Die Variable wird persistent gespeichert.

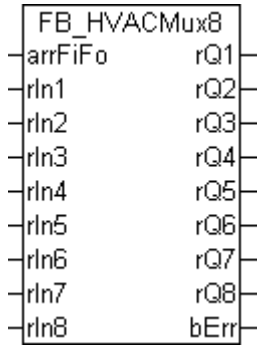
tDelayHardwareReset: Zeit, die nach Eingang zwischen Eingang der Froststörung und Ausgabe eines Quittierimpulses vergeht, um die Anlage wieder in Normalbetrieb zu versetzen. Die Variable wird persistent gespeichert.

tMonitoringAfterReset: Zeitintervall der Überwachung nach der ersten Frostwarnung. Die Variable wird persistent gespeichert.

Dokumente hierzu

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

3.6.13 FB_HVACMux8

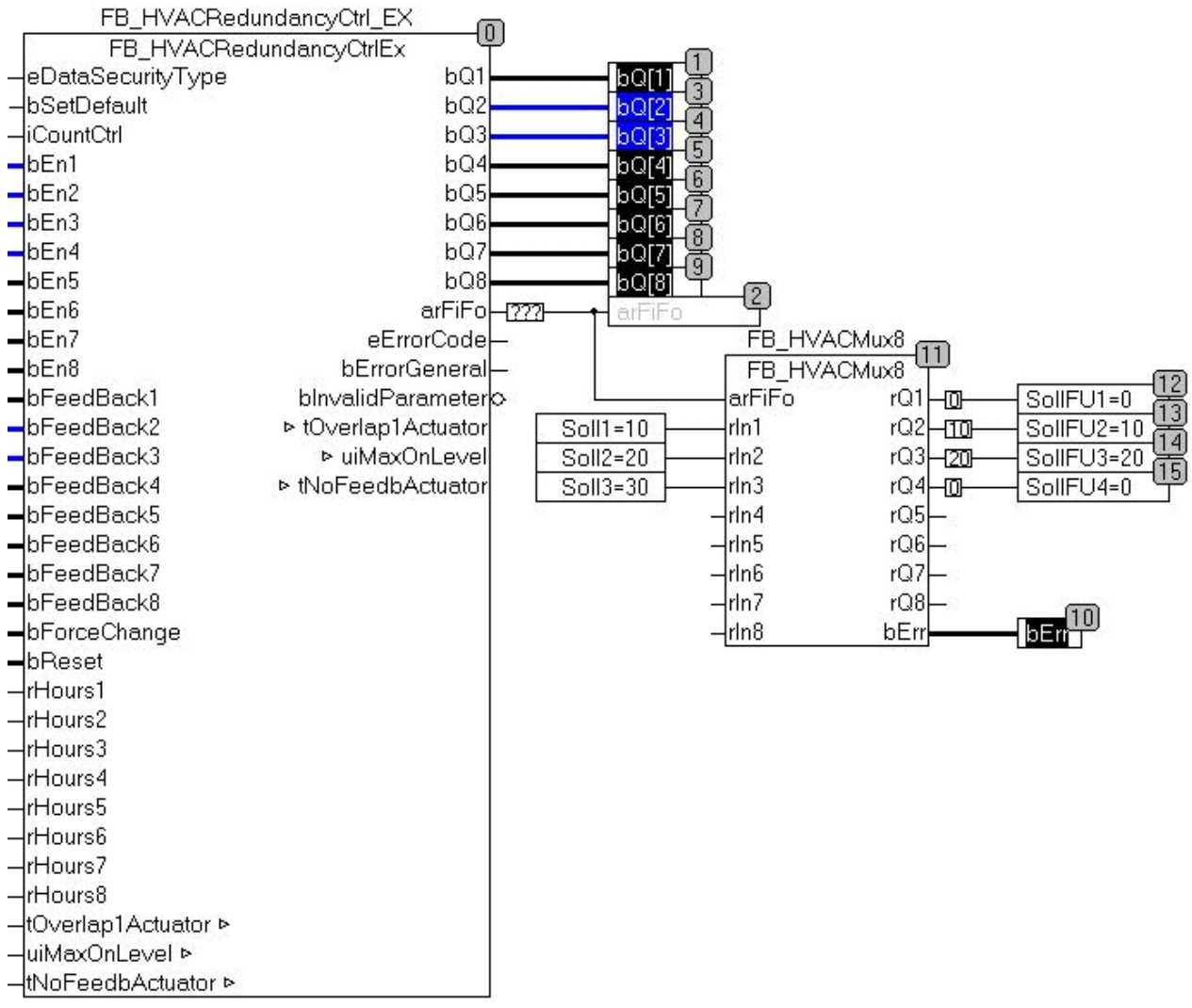


Anwendung

Dieser Funktionsbaustein dient zum Auswerten des FIFO-Speichers aus dem FB_HVACRedundancyCtrlEx. Die Eingänge werden nach FiFo-Tabelle auf die entsprechenden Ausgänge gemappt.

Wird dieser Baustein z.B. genutzt um drehzahlgeregelte Lüfter anzusteuern, so können die Sollwerte aus einer Reglerkaskade den entsprechenden Ausgängen (Antrieben) zugeordnet werden. Dazu werden die Sollwerte auf die Eingänge des FB_HVACMux8 gelegt. Die Verarbeitung erfolgt mit Hilfe der Informationen aus dem FiFo-Speicher (welcher Antrieb läuft und in welcher Reihenfolge wurde eingeschaltet).

Beispiel:



Es sind 4 Antriebe mit Drehzahlstellern (SollFU1 - SollFU4) vorhanden, von denen max. 3 laufen sollen. Der Regelbereich von 0-300% wird in 3 Sollwerte zerlegt (Soll1 bis Soll3). Der Ausgang *arrFiFo* des *FB_HVACRedundancyCtrlEx* liefert dem *FB_HVACMux8* die Zuordnungsbeschreibung. Im Beispiel steht in dem Array : 2,3,0,0,0,0,0,0. Es wurde also erst Q2 und dann Q3 eingeschaltet. Als Ergebnis wird im *FB_HVACMux8* nun *rIn1* auf den Ausgang *rQ2*, und *rIn2* auf den Ausgang *rQ3* ausgegeben.

VAR_INPUT

```
arrFiFo             : Array[1..8] of INT;       0 - 8
rIn1 - rIn8         : REAL;
```

arrFiFo: Enthält die Zuordnungstabelle. Erster Wert gibt an wohin der erste Eingang kopiert wird, zweiter Wert gibt an wohin der zweite Werte kopiert wird, etc.. Gleiche Wert sind zulässig. Bei "0" wird keine Zuordnung vorgenommen.

rIn1 - 8: Sollwerte die gemappt werden sollen.

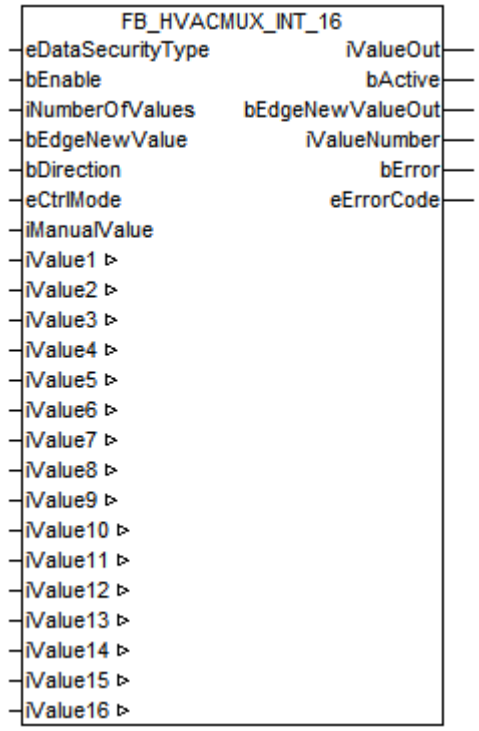
VAR_OUTPUT

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

rQ1 -rQ8: Aktorsollwert, laut FiFo-Tabelle gemappter Eingangswert.

bErr: Wertebereich in der FiFo-Tabelle wurde verletzt, zulässig 0-8.

3.6.14 FB_HVACMUX_INT_16



Anwendung

Anwendung

Der Funktionsbaustein beinhaltet zwei verschiedene Typen von Multiplexern jeweils für die Betriebsart *eHVACCtrlMode_Auto*bzw.*eHVACCtrlMode_Manual.* Diese Auswahl wird mit Hilfe des Enums *eCtrlMode* vorgenommen.
Es müssen vorab die folgenden Bedingungen erfüllt sein:

bEnable = TRUEAND*bError* = FALSE

Bei *eCtrlMode* = *eHVACCtrlMode_Auto* wird durch die Variable *bDirection* bestimmt, ob bei der Verwendung des Eingangs *bEdgeNewValue* die Ausgangsvariable *iValueNumber* in- bzw. dekrementiert wird. Wenn *bDirection* = FALSE ist, wird inkrementiert, wenn *bDirection* = TRUE ist, wird dekrementiert.

Bei *eCtrlMode* = *eHVACCtrlMode_Manual* wird durch den Wert der Variable *iManualValue* der Ausgang *iValueNumber* bestimmt. Die Variablen *iNumberOfValues, bEdgeNewValue* und *bDirection* werden in dieser Betriebsart nicht berücksichtigt.

Der Ausgang *iValueNumber* zeigt immer die Nummer der Variablen *iValue***X** an, deren Inhalt in der Ausgangsvariablen *iValueOut* ausgegeben wird.

Beipiele zu *eCtrlMode* = *eHVACCtrlMode_Auto*

- Ist *bDirection* = FALSEAND*iNumberOfValues* = **12** und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann inkrementiert *iValueNumber*. Ist *iValueNumber* = 6, so ist *iValueOut* = *iValue6*. Ist *iValueNumber*= *iNumberOfValues***(12)** und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = 1 und dementsprechend *iValueOut* = *iValue1.*

- Ist *bDirection* = TRUEAND*iNumberOfValues* = **8** und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann dekrementiert *iValueNumber*. Ist *iValueNumber* = 5, so ist *iValueOut* = *iValue5*. Ist *iValueNumber*= 1 und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = *iNumberOfValues***(8)** und dementsprechend *iValueOut* = *iValue8.*

Das Startverhalten der Ausgänge *iValueNumber* und *iValueOut* sieht folgendermaßen aus:

- Ist *bDirection* = FALSEAND*iNumberOfValues* > 0, so ist *iValueNumber* = 1 und damit *iValueOut* = *iValue1.*

- Ist *bDirection* = TRUEAND*iNumberOfValues* = 12, so ist *iNumberOfValues* = 12 und damit *iValueOut* = *iValue12.*

Beipiele zu *eCtrlMode* = *eHVACCtrlMode_Manual*

- Der Wert von *iValueNumber* wird von *iManualValue* bestimmt. Ist *iManualValue* = 7, so ist *iValueNumber* = 7 und dieses bedeutet dann, dass *iValueOut* = *iValue7* ist.

Das Startverhalten der Ausgänge *iValueNumber* und *iValueOut* sieht folgendermaßen aus:

- Ist *iManualValue* = 13, so ist *iValueNumber* = 13 und somit *iValueOut* = *iValue13.*

- Ist *iManualValue* = 0, so ist *iValueNumber* = 0 und somit *iValueOut* = *0.*

HINWEIS

Eine sich häufig ändernde Variable darf nicht an die VAR_IN_OUT-Variablen iValue1-16 angelegt werden, wenn eDataSecurityType = eHVACDataSecurityType_Persistentist. Dieses würde zu einem frühzeitigen Verschleiß des Speichermediums der Steuerung führen. Wenn sich die VAR_IN_OUT-Variablen iValue1-16 häufig ändern und nicht persistent abgelegt werden sollen, muss eDataSecurityType = eDataSecurityType_Idle sein.

Anwendungsbeispiel

Download	Benötige Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
iNumberOfValues   : INT;              1..16
bEdgeNewValue     : BOOL;
bDirection        : BOOL;
eCtrlMode         : E_HVACCtrlMode;
iManualValue      : INT;              0..16
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bEnable: Über ein TRUE wird der Funktionsbaustein freigegeben. Ist *bEnable* = FALSE, so wird *iValueOut* und *iValueNumber* konstant auf 0 gesetzt.

iNumberOfValues: Gibt die Anzahl der Variablen *iValue1-16* an, die über *iValueOut* ausgegeben werden können. Ist *iNumberOfValues* = 8, so bewegt sich der Ausgang *iValueNumber* zwischen 1 und 8. Es werden dann für die Ausgabe *iValueOut* die Variablen *iValue1* bis *iValue8* berücksichtigt, siehe Anwendung [▶ 399]
iNumberOfValues bestimmt das Startverhalten der Ausgänge *iValueOut* und *iValueNumber*, siehe Anwendung [▶ 399]
iNumberOfValues wird in der Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual* nicht berücksichtigt.
iNumberOfValues darf 1 nicht unterschreiten und 16 nicht überschreiten. Ansonsten wird mit *bError* = TRUE ein Fehler angezeigt und die Abarbeitung des Funktionsbausteins wird gestoppt.

bEdgeNewValue: Liegt eine steigende Flanke an *bEdgeNewValue*, dann in- oder dekrementiert *iValueNumber*.
Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*iNumberOfValues* > 0 und es liegt eine steigende Flanke an *bEdgeNewValue*, dann inkrementiert *iValueNumber*. Ist *iValueNumber* = 6, so ist *iValueOut* = *iValue6*. Ist *iValueNumber*= *iNumberOfValues* und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = 1 und somit*iValueOut* = *iValue1.*
Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*iNumberOfValues* = **12** und es liegt eine steigende Flanke an *bEdgeNewValue*, dann dekrementiert *iValueNumber*. Ist *iValueNumber* = 5, so ist *iValueOut* = *iValue5*. Ist *iValueNumber*= 1 und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = *iNumberOfValues***(12)** und somit *iValueOut* = *iValue12.*
bEdgeNewValue wird in der Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual* nicht berücksichtigt.

bDirection: *bDirection* bestimmt den Wirksinn des Funktionsbausteins.
bDirection = FALSE bedeutet, dass *iValueNumber* in aufsteigender Form von 1 nach *iNumberOfValues* inkrementiert und dadurch den Ausgabewert *iValueOut* bestimmt, siehe Anwendung [▶ 399]
bDirection = TRUE bedeutet, dass *iValueNumber* in absteigender Form von *iNumberOfValues* nach 1 dekrementiert und dadurch den Ausgabewert *iValueOut* bestimmt, siehe Anwendung [▶ 399]
bDirection wird in der Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual* nicht berücksichtigt.

eCtrlMode: Mittels des Enums wird die Betriebsart des Funktionsbausteins vorgegeben, siehe Anwendung [▶ 399]
Ist die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Auto,* so wird die Variable *iManualValue* nicht berücksichtigt.
Ist die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual,* so werden die Variablen *iNumberOfValues, bEdgeNewValue* und *bDirection* nicht berücksichtigt.

iManualValue: Ist die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual*AND*bEnable* = TRUEAND*bError* = FALSEAND*iManualValue* = 15, so ist *iValueNumber* = 15. Der Inhalt der Variable *iValue15* wird dann über *iValueOut* ausgegeben, siehe Anwendung [▶ 399].
iManualValue wird nur in der Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual* berücksichtigt. Der Wert von *iValueNumber* wird von *iManualValue* bestimmt. *iManualValue* sollte im Bereich von 0 und 16 liegen. Unterschreitet *iManualValue* den Wert 0, so wird über *iValueNumber* eine 0 ausgegeben. Überschreitet *iManualValue* den Wert 16, so wird über *iValueNumber* eine 16 ausgegeben. Ist *iManualValue* = 0, so ist *iValueNumber* = 0 und dementsprechend *iValueOut* = *0.*

VAR_OUTPUT

```
iValueOut           : INT;
bActive             : BOOL;
bEdgeNewValueOut    : BOOL;
iValueNumber        : INT;
bError              : BOOL;
eErrorCode          : E_HVACErrorCodes;
```

iValueOut: Über *iValueOut* wird der Wert der Variablen *iValue1-16* ausgegeben.
Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*iNumberOfValues* > 0 und es liegt eine steigende Flanke an *bEdgeNewValue*, dann inkrementiert *iValueNumber*. Ist *iValueNumber* = 6, so ist *iValueOut* = *iValue6*. Ist *iValueNumber*= *iNumberOfValues* und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = 1 und somit*iValueOut* = *iValue1.*
Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*iNumberOfValues* = **12** und es liegt eine steigende Flanke an *bEdgeNewValue*, dann dekrementiert *iValueNumber*. Ist *iValueNumber* = 5, so ist *iValueOut* = *iValue5*. Ist *iValueNumber*= 1 und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = *iNumberOfValues***(12)** und somit *iValueOut* = *iValue12.*
Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Manual,* so wird der Wert von *iValueNumber* von *iManualValue* bestimmt. Ist *iManualValue* = 7, so ist *iValueNumber* = 7 und dieses bedeutet dann, dass *iValueOut* = *iValue7* ist.

Das Startverhalten des Ausgangs *iValueOut* sieht folgendermaßen aus, wenn *bEnable* = TRUEAND*bError* = FALSEAND

- *eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*iNumberOfValues* > 0, so ist *iValueNumber* = 1 und *iValueOut* = *iValue1.*

- *eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*iNumberOfValues* = 12, so ist *iNumberOfValues* = 12 und *iValueOut* = *iValue12.*

- *eCtrlMode* = *eHVACCtrlMode_Manual*AND*iManualValue* = 13, so ist *iValueNumber* = 13 und *iValueOut* = *iValue13.*

- *eCtrlMode* = *eHVACCtrlMode_Manual*AND*iManualValue* = 0, so ist *iValueNumber* = 0 und *iValueOut* = *0.*

bActive: *bActive* wird TRUE , wenn

1. *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto* ist.

2. *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND*iManualValue* > 0 ist.

bEdgeNewValueOut: Ist für einen SPS-Zyklus TRUE, wenn *bEnable* = TRUEAND*bError* = FALSEAND*iValueNumber* seinen Wert ändert.

iValueNumber: Der Ausgang *iValueNumber* zeigt immer die Nummer der Variablen *iValue***X** an, deren Inhalt in der Ausgangsvariablen *iValueOut* ausgegeben wird.
Bei *eCtrlMode* = *eHVACCtrlMode_Auto* wird durch die Variable *bDirection* bestimmt, ob bei der Verwendung des Eingangs *bEdgeNewValue* die Ausgangsvariable *iValueNumber* in- bzw. dekrementiert wird. Wenn *bDirection* = FALSE ist, wird inkrementiert, wenn *bDirection* = TRUE ist, wird dekrementiert.
Bei *eCtrlMode* = *eHVACCtrlMode_Manual* wird der Wert von *iValueNumber* von *iManualValue* bestimmt.

bError: Der Ausgang signalisiert mit einem TRUE, dass ein Fehler anliegt und ein falscher Parameter an der Variable *iNumberOfValues* anliegt. *iValueOut* und *iValueNumber* werden konstant auf 0 gesetzt und das Enum *eErrorCode* zeigt die Fehlernummer an. Nach Behebung des Fehlers muss die Meldung *bError* nicht quittiert werden.

eErrorCode: Liefert bei einem gesetzten *bError*-Ausgang die Fehlernummer [▶ 529]. Folgender Fehler kann in diesem Funktionsbaustein vorkommen:

eHVACErrorCodes_InvalidParam_iNumberOfValues: Es liegt ein fehlerhafter Wert an *iNumberOfValues* an. *iNumberOfValues* darf 1 nicht unterschreiten und 16 nicht überschreiten.

Um in der SPS an die Fehlernummern des Enums zu gelangen, kann eErrorCode einer Variablen vom Datentyp WORD zugewiesen werden.eHVACErrorCodes_InvalidParam_iNumberOfValues = 42

VAR_IN_OUT

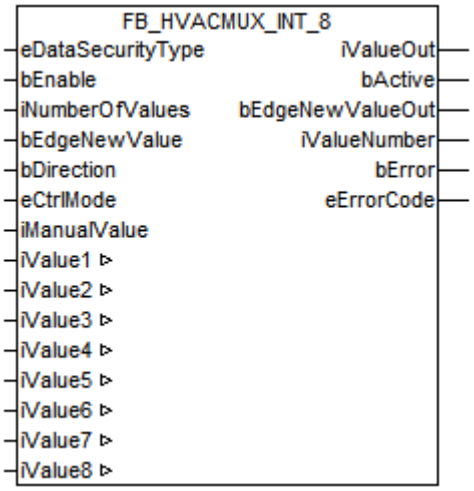
```
iValue1-16        : INT;
```

iValue1-16: Mittels der Variablen *iValue1* bis *iValue16* wird der Wert der Ausgangsvariable *iValueOut* bestimmt. Ist *iValueNumber* = 6, so ist *iValueOut* = *iValue6*.
Ist *iNumberOfValues* = 8, so bewegt sich der Ausgang *iValueNumber* zwischen 1 und 8. Es werden dann für die Ausgabe *iValueOut* die Variablen *iValue1* bis *iValue8* berücksichtigt, wenn die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Auto* ist, siehe Anwendung [▶ 399]

Dokumente hierzu

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

3.6.15 FB_HVACMUX_INT_8



Anwendung

Anwendung

Der Funktionsbaustein beinhaltet zwei verschiedene Typen von Multiplexern, jeweils für die Betriebsart *eHVACCtrlMode_Auto*bzw.*eHVACCtrlMode_Manual.* Diese Auswahl wird mit Hilfe des Enums *eCtrlMode* vorgenommen.
Es müssen vorab die folgenden Bedingungen erfüllt sein:

bEnable = TRUEAND*bError* = FALSE

Bei *eCtrlMode* = *eHVACCtrlMode_Auto* wird durch die Variable *bDirection* bestimmt, ob bei der Verwendung des Eingangs *bEdgeNewValue* die Ausgangsvariable *iValueNumber* in- bzw. dekrementiert wird. Wenn *bDirection* = FALSE ist, wird inkrementiert, wenn *bDirection* = TRUE ist, wird dekrementiert.

Bei *eCtrlMode* = *eHVACCtrlMode_Manual* wird durch den Wert der Variable *iManualValue* der Ausgang *iValueNumber* bestimmt. Die Variablen *iNumberOfValues, bEdgeNewValue* und *bDirection* werden in dieser Betriebsart nicht berücksichtigt.

Der Ausgang *iValueNumber* zeigt immer die Nummer der Variablen *iValue***X** an, deren Inhalt in der Ausgangsvariablen *iValueOut* ausgegeben wird.

Beipiele zu *eCtrlMode* = *eHVACCtrlMode_Auto*

- Ist *bDirection* = FALSEAND*iNumberOfValues* = **7** und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann inkrementiert *iValueNumber*. Ist *iValueNumber* = 4, so ist *iValueOut* = *iValue4*. Ist *iValueNumber*= *iNumberOfValues***(7)** und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = 1 und somit *iValueOut* = *iValue1.*

- Ist *bDirection* = TRUEAND*iNumberOfValues* = **8** und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann dekrementiert *iValueNumber*. Ist *iValueNumber* = 3, so ist *iValueOut* = *iValue3*. Ist *iValueNumber*= 1 und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = *iNumberOfValues***(8)** und somit *iValueOut* = *iValue8.*

Das Startverhalten der Ausgänge *iValueNumber* und *iValueOut* sieht folgendermaßen aus:

- Ist *bDirection* = FALSEAND*iNumberOfValues* > 0, so ist *iValueNumber* = 1 und *iValueOut* = *iValue1.*

- Ist *bDirection* = TRUEAND*iNumberOfValues* = 6, so ist *iNumberOfValues* = 6 und *iValueOut* = *iValue6.*

Beipiele zu *eCtrlMode* = *eHVACCtrlMode_Manual*

- Der Wert von *iValueNumber* wird von *iManualValue* bestimmt. Ist *iManualValue* = 7, so ist *iValueNumber* = 7 und dieses bedeutet dann, dass *iValueOut* = *iValue7* ist.

Das Startverhalten der Ausgänge *iValueNumber* und *iValueOut* sieht folgendermaßen aus:

- Ist *iManualValue* = 13, so ist *iValueNumber* = 13 und *iValueOut* = *iValue13.*

- Ist *iManualValue* = 0, so ist *iValueNumber* = 0 und *iValueOut* = *0.*

HINWEIS

Eine sich häufig ändernde Variable darf nicht an die VAR_IN_OUT-Variablen iValue1-8 angelegt werden, wenn eDataSecurityType = eHVACDataSecurityType_Persistentist. Dieses würde zu einem frühzeitigen Verschleiß des Speichermediums der Steuerung führen. Wenn sich die VAR_IN_OUT-Variablen iValue1-8 häufig ändern und nicht persistent abgelegt werden sollen, muss eDataSecurityType = eDataSecurityType_Idle sein.

Anwendungsbeispiel

Download	Benötige Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
iNumberOfValues   : INT;              1..8
bEdgeNewValue     : BOOL;
bDirection        : BOOL;
eCtrlMode         : E_HVACCtrlMode;
iManualValue      : INT;              0..8
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bEnable: Über ein TRUE wird der Funktionsbaustein freigegeben. Ist *bEnable* = FALSE, so wird *iValueOut* und *iValueNumber* konstant auf 0 gesetzt.

iNumberOfValues: Gibt die Anzahl der Variablen *iValue1-8* an, die über *iValueOut* ausgegeben werden können. Ist *iNumberOfValues* = 8, so bewegt sich der Ausgang *iValueNumber* zwischen 1 und 8. Es werden dann für die Ausgabe *iValueOut* die Variablen *iValue1* bis *iValue8* berücksichtigt, siehe Anwendung [▶ 403]
iNumberOfValues bestimmt das Startverhalten der Ausgänge *iValueOut* und *iValueNumber*, siehe Anwendung [▶ 403]
iNumberOfValues wird in der Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual* nicht berücksichtigt.
iNumberOfValues darf 1 nicht unterschreiten und 8 nicht überschreiten. Ansonsten wird mit *bError* = TRUE ein Fehler angezeigt und die Abarbeitung des Funktionsbausteins wird gestoppt.

bEdgeNewValue: Liegt eine steigende Flanke an *bEdgeNewValue*, dann in- oder dekrementiert *iValueNumber*.
Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*iNumberOfValues* > 0 und es liegt eine steigende Flanke an *bEdgeNewValue*, dann inkrementiert *iValueNumber*. Ist *iValueNumber* = 6, so ist *iValueOut* = *iValue6*. Ist *iValueNumber*= *iNumberOfValues* und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = 1 und somit *iValueOut* = *iValue1.*
Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*iNumberOfValues* = **6** und es liegt eine steigende Flanke an *bEdgeNewValue*, dann dekrementiert *iValueNumber*. Ist *iValueNumber* = 5, so ist *iValueOut* = *iValue5*. Ist *iValueNumber*= 1 und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = *iNumberOfValues***(6)** und somit *iValueOut* = *iValue6.*
bEdgeNewValue wird in der Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual* nicht berücksichtigt.

bDirection: *bDirection* bestimmt den Wirksinn des Funktionsbausteins.
bDirection = FALSE bedeutet, dass *iValueNumber* in aufsteigender Form von 1 nach *iNumberOfValues* inkrementiert und dadurch den Ausgabewert *iValueOut* bestimmt, siehe Anwendung [▶ 403]
bDirection = TRUE bedeutet, dass *iValueNumber* in absteigender Form von *iNumberOfValues* nach 1 dekrementiert und dadurch den Ausgabewert *iValueOut* bestimmt, siehe Anwendung [▶ 403]
bDirection wird in der Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual* nicht berücksichtigt.

eCtrlMode: Mittels des Enums wird die Betriebsart des Funktionsbausteins vorgegeben, siehe Anwendung [▶ 403]
Ist die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Auto,* so wird die Variable *iManualValue* nicht berücksichtigt.
Ist die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual,* so werden die Variablen *iNumberOfValues, bEdgeNewValue* und *bDirection* nicht berücksichtigt.

iManualValue: Ist die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual*AND*bEnable* = TRUEAND*bError* = FALSEAND*iManualValue* = 15, so ist iValueNumber = 15. Der Inhalt der Variable *iValue15* wird dann über *iValueOut* ausgegeben, siehe Anwendung [▶ 403].
iManualValue wird nur in der Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual* berücksichtigt. Der Wert von *iValueNumber* wird von *iManualValue* bestimmt. *iManualValue* sollte im Bereich von 0 und 16 liegen. Unterschreitet *iManualValue* den Wert 0, so wird über *iValueNumber* eine 0 ausgegeben. Überschreitet *iManualValue* den Wert 16, so wird über *iValueNumber* eine 16 ausgegeben. Ist *iManualValue* = 0, so ist *iValueNumber* = 0 und *iValueOut* = *0.*

VAR_OUTPUT

```
iValueOut           : INT;
bActive             : BOOL;
bEdgeNewValueOut    : BOOL;
iValueNumber        : INT;
bError              : BOOL;
eErrorCode          : E_HVACErrorCodes;
```

iValueOut: Über *iValueOut* wird der Wert der Variablen *iValue1-8* ausgegeben.
Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*iNumberOfValues* > 0 und es liegt eine steigende Flanke an *bEdgeNewValue*, dann inkrementiert *iValueNumber*. Ist *iValueNumber* = 6, so ist *iValueOut* = *iValue6*. Ist *iValueNumber*= *iNumberOfValues* und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = 1 und somit *iValueOut* = *iValue1.*
Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*iNumberOfValues* = **7** und es liegt eine steigende Flanke an *bEdgeNewValue*, dann dekrementiert *iValueNumber*. Ist *iValueNumber* = 5, so ist *iValueOut* = *iValue5*. Ist *iValueNumber*= 1 und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = *iNumberOfValues***(7)** und somit *iValueOut* = *iValue7.*
Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Manual,* so wird der Wert von *iValueNumber* von *iManualValue* bestimmt. Ist *iManualValue* = 7, so ist *iValueNumber* = 7 und dieses bedeutet dann, dass *iValueOut* = *iValue7* ist.

Das Startverhalten des Ausgangs *iValueOut* sieht folgendermaßen aus, wenn *bEnable* = TRUEAND*bError* = FALSEAND

- *eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*iNumberOfValues* > 0, so ist *iValueNumber* = 1 und *iValueOut* = *iValue1.*

- *eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*iNumberOfValues* = 7, so ist *iNumberOfValues* = 7 und *iValueOut* = *iValue7.*

- *eCtrlMode* = *eHVACCtrlMode_Manual*AND*iManualValue* = 13, so ist *iValueNumber* = 13 und *iValueOut* = *iValue13.*

- *eCtrlMode* = *eHVACCtrlMode_Manual*AND*iManualValue* = 0, so ist *iValueNumber* = 0 und *iValueOut* = *0.*

bActive: *bActive* wird TRUE , wenn

1. *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto* ist.

2. *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND*iManualValue* > 0 ist.

bEdgeNewValueOut: Ist für einen SPS-Zyklus TRUE, wenn *bEnable* = TRUEAND*bError* = FALSEAND*iValueNumber* seinen Wert ändert.

iValueNumber: Der Ausgang *iValueNumber* zeigt immer die Nummer der Variablen *iValue***X** an, deren Inhalt in der Ausgangsvariablen *iValueOut* ausgegeben wird.
Bei *eCtrlMode* = *eHVACCtrlMode_Auto* wird durch die Variable *bDirection* bestimmt, ob bei der Verwendung des Eingangs *bEdgeNewValue* die Ausgangsvariable *iValueNumber* in- bzw. dekrementiert wird. Wenn *bDirection* = FALSE ist, wird inkrementiert, wenn *bDirection* = TRUE ist, wird dekrementiert.
Bei *eCtrlMode* = *eHVACCtrlMode_Manual* wird der Wert von *iValueNumber* von *iManualValue* bestimmt.

bError: Der Ausgang signalisiert mit einem TRUE, dass ein Fehler anliegt und ein falscher Parameter an der Variable *iNumberOfValues* anliegt. *iValueOut* und *iValueNumber* werden konstant auf 0 gesetzt und das Enum *eErrorCode* zeigt die Fehlernummer an. Nach Behebung des Fehlers muss die Meldung *bError* nicht quittiert werden.

eErrorCode: Liefert bei einem gesetzten *bError*-Ausgang die Fehlernummer [▶ 529]. Folgender Fehler kann in diesem Funktionsbaustein vorkommen:

eHVACErrorCodes_InvalidParam_iNumberOfValues: Es liegt ein fehlerhafter Wert an *iNumberOfValues* an. *iNumberOfValues* darf 1 nicht unterschreiten und 8 nicht überschreiten.

Um in der SPS an die Fehlernummern des Enums zu gelangen, kann eErrorCode einer Variablen vom Datentyp WORD zugewiesen werden.eHVACErrorCodes_InvalidParam_iNumberOfValues = 42

VAR_IN_OUT

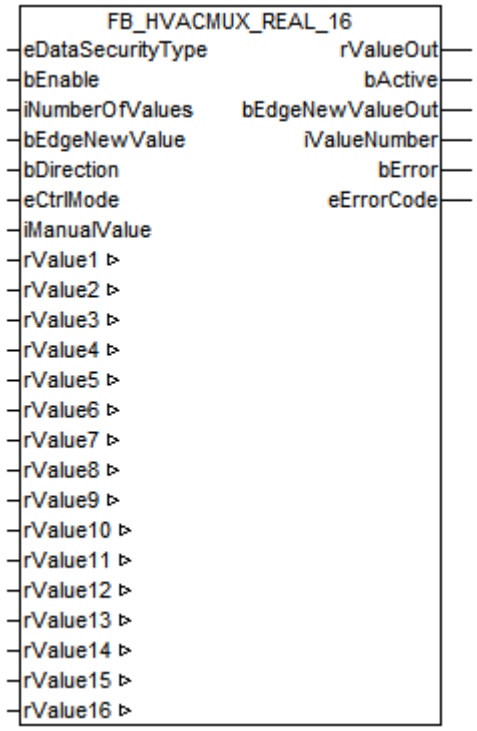
```
iValue1-8        : INT;
```

iValue1-8: Mittels der Variablen *iValue1* bis *iValue8* wird der Wert der Ausgangsvariable *iValueOut* bestimmt. Ist *iValueNumber* = 6, so ist *iValueOut* = *iValue6*.
Ist *iNumberOfValues* = 8, so bewegt sich der Ausgang *iValueNumber* zwischen 1 und 8. Es werden dann für die Ausgabe *iValueOut* die Variablen *iValue1* bis *iValue8* berücksichtigt, wenn die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Auto* ist, siehe Anwendung [▶ 403]

Dokumente hierzu

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

3.6.16 FB_HVACMUX_REAL_16



Anwendung

Anwendung

Der Funktionsbaustein beinhaltet zwei verschiedene Typen von Multiplexern, jeweils für die Betriebsart *eHVACCtrlMode_Auto*bzw.*eHVACCtrlMode_Manual.* Diese Auswahl wird mit Hilfe des Enums *eCtrlMode* vorgenommen.
Es müssen vorab die folgenden Bedingungen erfüllt sein:

bEnable = TRUEAND*bError* = FALSE

Bei *eCtrlMode* = *eHVACCtrlMode_Auto* wird durch die Variable *bDirection* bestimmt, ob bei der Verwendung des Eingangs *bEdgeNewValue* die Ausgangsvariable *iValueNumber* in- bzw. dekrementiert wird. Wenn *bDirection* = FALSE ist, wird inkrementiert, wenn *bDirection* = TRUE ist, wird dekrementiert.

Bei *eCtrlMode* = *eHVACCtrlMode_Manual* wird durch den Wert der Variable *iManualValue* der Ausgang *iValueNumber* bestimmt. Die Variablen *iNumberOfValues, bEdgeNewValue* und *bDirection* werden in dieser Betriebsart nicht berücksichtigt.

Der Ausgang *iValueNumber* zeigt immer die Nummer der Variablen *rValue***X** an, deren Inhalt in der Ausgangsvariablen *rValueOut* ausgegeben wird.

Beipiele zu *eCtrlMode* = *eHVACCtrlMode_Auto*

- Ist *bDirection* = FALSEAND*iNumberOfValues* = **12** und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann inkrementiert *iValueNumber*. Ist *iValueNumber* = 6, so ist *rValueOut* = *rValue6*. Ist *iValueNumber*= *iNumberOfValues***(12)** und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = 1 und somit *rValueOut* = *rValue1.*

- Ist *bDirection* = TRUEAND*iNumberOfValues* = **8** und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann dekrementiert *iValueNumber*. Ist *rValueNumber* = 5, so ist *rValueOut* = *rValue5*. Ist *iValueNumber*= 1 und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = *iNumberOfValues***(8)** und somit *rValueOut* = *rValue8.*

Das Startverhalten der Ausgänge *iValueNumber* und *rValueOut* sieht folgendermaßen aus:

- Ist *bDirection* = FALSEAND*iNumberOfValues* > 0, so ist *iValueNumber* = 1 und damit *rValueOut* = *rValue1.*

- Ist *bDirection* = TRUEAND*iNumberOfValues* = 12, so ist *iNumberOfValues* = 12 und damit *rValueOut* = *rValue12.*

Beipiele zu *eCtrlMode* = *eHVACCtrlMode_Manual*

- Der Wert von *iValueNumber* wird von *iManualValue* bestimmt. Ist *iManualValue* = 7, so ist *iValueNumber* = 7 und dieses bedeutet dann, dass *rValueOut* = *rValue7* ist.

Das Startverhalten der Ausgänge *iValueNumber* und *rValueOut* sieht folgendermaßen aus:

- Ist *iManualValue* = 13, so ist *iValueNumber* = 13 und *rValueOut* = *rValue13.*

- Ist *iManualValue* = 0, so ist *iValueNumber* = 0 und *rValueOut* = *0.*

HINWEIS

Eine sich häufig ändernde Variable darf nicht an die VAR_IN_OUT-Variablen rValue1-16 angelegt werden, wenn eDataSecurityType = eHVACDataSecurityType_Persistentist. Dieses würde zu einem frühzeitigen Verschleiß des Speichermediums der Steuerung führen. Wenn sich die VAR_IN_OUT-Variablen rValue1-16 häufig ändern und nicht persistent abgelegt werden sollen, muss eDataSecurityType = eDataSecurityType_Idle sein.

Anwendungsbeispiel

Download	Benötige Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
iNumberOfValues   : INT;              1..16
bEdgeNewValue     : BOOL;
bDirection        : BOOL;
eCtrlMode         : E_HVACCtrlMode;
iManualValue      : INT;              0..16
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bEnable: Über ein TRUE wird der Funktionsbaustein freigegeben. Ist *bEnable* = FALSE, so wird *rValueOut* und *iValueNumber* konstant auf 0 gesetzt.

iNumberOfValues: Gibt die Anzahl der Variablen *rValue1-16* an, die über *rValueOut* ausgegeben werden können. Ist *iNumberOfValues* = 8, so bewegt sich der Ausgang *iValueNumber* zwischen 1 und 8. Es werden dann für die Ausgabe *rValueOut* die Variablen *rValue1* bis *rValue8* berücksichtigt, siehe Anwendung [▶ 407]
iNumberOfValues bestimmt das Startverhalten der Ausgänge *rValueOut* und *iValueNumber*, siehe Anwendung [▶ 407]
iNumberOfValues wird in der Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual* nicht berücksichtigt.
iNumberOfValues darf 1 nicht unterschreiten und 16 nicht überschreiten. Ansonsten wird mit *bError* = TRUE ein Fehler angezeigt und die Abarbeitung des Funktionsbausteins wird gestoppt.

bEdgeNewValue: Liegt eine steigende Flanke an *bEdgeNewValue*, dann in- oder dekrementiert *iValueNumber*.
Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*iNumberOfValues* > 0 und es liegt eine steigende Flanke an *bEdgeNewValue*, dann inkrementiert *iValueNumber*. Ist *iValueNumber* = 6, so ist *rValueOut* = *rValue6*. Ist *iValueNumber*= *iNumberOfValues* und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = 1 und somit *rValueOut* = *rValue1.*
Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*iNumberOfValues* = **12** und es liegt eine steigende Flanke an *bEdgeNewValue*, dann dekrementiert *iValueNumber*. Ist *iValueNumber* = 5, so ist *rValueOut* = *rValue5*. Ist *iValueNumber*= 1 und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = *iNumberOfValues***(12)** und somit *rValueOut* = *rValue12.*
bEdgeNewValue wird in der Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual* nicht berücksichtigt.

bDirection: *bDirection* bestimmt den Wirksinn des Funktionsbausteins.
bDirection = FALSE bedeutet, dass *iValueNumber* in aufsteigender Form von 1 nach *iNumberOfValues* inkrementiert und dadurch den Ausgabewert *rValueOut* bestimmt, siehe Anwendung [▶ 407]
bDirection = TRUE bedeutet, dass *iValueNumber* in absteigender Form von *iNumberOfValues* nach 1 dekrementiert und dadurch den Ausgabewert *rValueOut* bestimmt, siehe Anwendung [▶ 407]
bDirection wird in der Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual* nicht berücksichtigt.

eCtrlMode: Mittels des Enums wird die Betriebsart des Funktionsbausteins vorgegeben, siehe Anwendung [▶ 407]
Ist die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Auto,* so wird die Variable *iManualValue* nicht berücksichtigt.
Ist die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual,* so werden die Variablen *iNumberOfValues, bEdgeNewValue* und *bDirection* nicht berücksichtigt.

iManualValue: Ist die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual*AND*bEnable* = TRUEAND*bError* = FALSEAND*iManualValue* = 15, so ist *iValueNumber* = 15. Der Inhalt der Variable *rValue15* wird dann über *rValueOut* ausgegeben, siehe Anwendung [▶ 407].
iManualValue wird nur in der Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual* berücksichtigt. Der Wert von *iValueNumber* wird von *iManualValue* bestimmt. *iManualValue* sollte im Bereich von 0 und 16 liegen. Unterschreitet *iManualValue* den Wert 0, so wird über *iValueNumber* eine 0 ausgegeben. Überschreitet *iManualValue* den Wert 16, so wird über *iValueNumber* eine 16 ausgegeben. Ist *iManualValue* = 0, so ist *iValueNumber* = 0 und *rValueOut* = *0.*

VAR_OUTPUT

```
rValueOut           : REAL;
bActive             : BOOL;
bEdgeNewValueOut    : BOOL;
iValueNumber        : INT;
bError              : BOOL;
eErrorCode          : E_HVACErrorCodes;
```

rValueOut: Über *rValueOut* wird der Wert der Variablen *rValue1-16* ausgegeben.
Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*iNumberOfValues* > 0 und es liegt eine steigende Flanke an *bEdgeNewValue*, dann inkrementiert *iValueNumber*. Ist *iValueNumber* = 6, so ist *rValueOut* = *rValue6*. Ist *iValueNumber*= *iNumberOfValues* und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = 1 und somit *rValueOut* = *rValue1.*
Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*iNumberOfValues* = **12** und es liegt eine steigende Flanke an *bEdgeNewValue*, dann dekrementiert *iValueNumber*. Ist *iValueNumber* = 5, so ist *rValueOut* = *rValue5.* Ist *iValueNumber*= 1 und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = *iNumberOfValues***(12)** und somit *rValueOut* = *rValue12.*
Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Manual,* so wird der Wert von *iValueNumber* von *iManualValue* bestimmt. Ist *iManualValue* = 7, so ist *iValueNumber* = 7 und dieses bedeutet dann, dass *rValueOut* = *rValue7* ist.

Das Startverhalten des Ausgangs *rValueOut* sieht folgendermaßen aus, wenn *bEnable* = TRUEAND*bError* = FALSEAND

- *eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*iNumberOfValues* > 0, so ist *iValueNumber* = 1 und *rValueOut* = *rValue1*.

- *eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*iNumberOfValues* = 12, so ist *iNumberOfValues* = 12 und *rValueOut* = *rValue12.*

- *eCtrlMode* = *eHVACCtrlMode_Manual*AND*iManualValue* = 13, so ist *iValueNumber* = 13 und *rValueOut* = *rValue13.*

- *eCtrlMode* = *eHVACCtrlMode_Manual*AND*iManualValue* = 0, so ist *iValueNumber* = 0 und *rValueOut* = *0.*

bActive: *bActive* wird TRUE , wenn

1. *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto* ist.

2. *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND*iManualValue* > 0 ist.

bEdgeNewValueOut: Ist für einen SPS-Zyklus TRUE, wenn *bEnable* = TRUEAND*bError* = FALSEAND*iValueNumber* seinen Wert ändert.

iValueNumber: Der Ausgang *iValueNumber* zeigt immer die Nummer der Variablen *rValue***X** an, deren Inhalt in der Ausgangsvariablen *rValueOut* ausgegeben wird.
Bei *eCtrlMode* = *eHVACCtrlMode_Auto* wird durch die Variable *bDirection* bestimmt, ob bei der Verwendung des Eingangs *bEdgeNewValue* die Ausgangsvariable *iValueNumber* in- bzw. dekrementiert wird. Wenn *bDirection* = FALSE ist, wird inkrementiert, wenn *bDirection* = TRUE ist, wird dekrementiert.
Bei *eCtrlMode* = *eHVACCtrlMode_Manual* wird der Wert von *iValueNumber* von *iManualValue* bestimmt.

bError: Der Ausgang signalisiert mit einem TRUE, dass ein Fehler anliegt und ein falscher Parameter an der Variable *iNumberOfValues* anliegt. *rValueOut* und *iValueNumber* werden konstant auf 0 gesetzt und das Enum *eErrorCode* zeigt die Fehlernummer an. Nach Behebung des Fehlers muss die Meldung *bError* nicht quittiert werden.

eErrorCode: Liefert bei einem gesetzten *bError*-Ausgang die Fehlernummer [▶ 529]. Folgender Fehler kann in diesem Funktionsbaustein vorkommen:

eHVACErrorCodes_InvalidParam_iNumberOfValues: Es liegt ein fehlerhafter Wert an *iNumberOfValues* an. *iNumberOfValues* darf 1 nicht unterschreiten und 16 nicht überschreiten.

Um in der SPS an die Fehlernummern des Enums zu gelangen, kann eErrorCode einer Variablen vom Datentyp WORD zugewiesen werden.eHVACErrorCodes_InvalidParam_iNumberOfValues = 42

VAR_IN_OUT

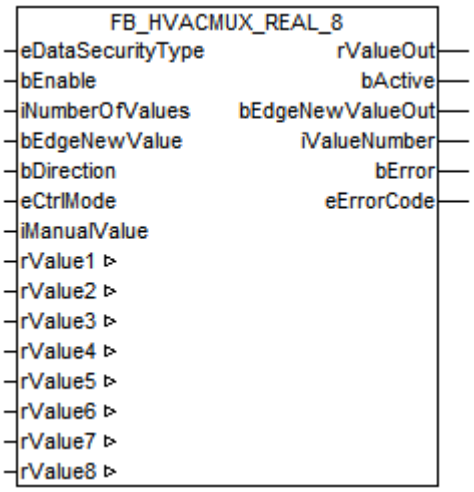
```
rValue1-16        : REAL;
```

rValue1-16: Mittels der Variablen *rValue1* bis *rValue16* wird der Wert der Ausgangsvariable *rValueOut* bestimmt. Ist *iValueNumber* = 6, so ist *rValueOut* = *rValue6*.
Ist *iNumberOfValues* = 8, so bewegt sich der Ausgang *iValueNumber* zwischen 1 und 8. Es werden dann für die Ausgabe *rValueOut* die Variablen *rValue1* bis *rValue8* berücksichtigt, wenn die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Auto* ist, siehe Anwendung [▶ 407]: Die Variable wird persistent gespeichert.

Dokumente hierzu

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

3.6.17 FB_HVACMUX_REAL_8



Anwendung

Anwendung

Der Funktionsbaustein beinhaltet zwei verschiedene Typen von Multiplexern, jeweils für die Betriebsart *eHVACCtrlMode_Auto*bzw.*eHVACCtrlMode_Manual.* Diese Auswahl wird mit Hilfe des Enums *eCtrlMode* vorgenommen.
Es müssen vorab die folgenden Bedingungen erfüllt sein:

bEnable = TRUEAND*bError* = FALSE

Bei *eCtrlMode* = *eHVACCtrlMode_Auto* wird durch die Variable *bDirection* bestimmt, ob bei der Verwendung des Eingangs *bEdgeNewValue* die Ausgangsvariable *iValueNumber* in- bzw. dekrementiert wird. Wenn *bDirection* = FALSE ist, wird inkrementiert, wenn *bDirection* = TRUE ist, wird dekrementiert.

Bei *eCtrlMode* = *eHVACCtrlMode_Manual* wird durch den Wert der Variable *iManualValue* der Ausgang *iValueNumber* bestimmt. Die Variablen *iNumberOfValues, bEdgeNewValue* und *bDirection* werden in dieser Betriebsart nicht berücksichtigt.

Der Ausgang *iValueNumber* zeigt immer die Nummer der Variablen *rValue***X** an, deren Inhalt in der Ausgangsvariablen *rValueOut* ausgegeben wird.

Beipiele zu *eCtrlMode* = *eHVACCtrlMode_Auto*

- Ist *bDirection* = FALSEAND*iNumberOfValues* = **7** und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann inkrementiert *iValueNumber*. Ist *iValueNumber* = 4, so ist *rValueOut* = *rValue4*. Ist *iValueNumber*= *iNumberOfValues***(7)** und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = 1 und somit *rValueOut* = *rValue1.*

- Ist *bDirection* = TRUEAND*iNumberOfValues* = **8** und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann dekrementiert *iValueNumber*. Ist *iValueNumber* = 3, so ist *rValueOut* = *rValue3*. Ist *iValueNumber*= 1 und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = *iNumberOfValues***(8)** und somit *rValueOut* = *rValue8.*

Das Startverhalten der Ausgänge *iValueNumber* und *rValueOut* sieht folgendermaßen aus:

- Ist *bDirection* = FALSEAND*iNumberOfValues* > 0, so ist *iValueNumber* = 1 und somit *rValueOut* = *rValue1.*

- Ist *bDirection* = TRUEAND*iNumberOfValues* = 6, so ist *iNumberOfValues* = 6 und somit *rValueOut* = *rValue6.*

Beipiele zu *eCtrlMode* = *eHVACCtrlMode_Manual*

- Der Wert von *iValueNumber* wird von *iManualValue* bestimmt. Ist *iManualValue* = 7, so ist *iValueNumber* = 7 und dieses bedeutet dann, dass *rValueOut* = *rValue7* ist.

Das Startverhalten der Ausgänge *iValueNumber* und *rValueOut* sieht folgendermaßen aus:

- Ist *iManualValue* = 13, so ist *iValueNumber* = 13 und *rValueOut* = *rValue13.*

- Ist *iManualValue* = 0, so ist *iValueNumber* = 0 und *rValueOut* = *0.*

HINWEIS
Eine sich häufig ändernde Variable darf nicht an die VAR_IN_OUT-Variablen rValue1-8 angelegt werden, wenn eDataSecurityType = eHVACDataSecurityType_Persistentist. Dieses würde zu einem frühzeitigen Verschleiß des Speichermediums der Steuerung führen. Wenn sich die VAR_IN_OUT-Variablen rValue1-8 häufig ändern und nicht persistent abgelegt werden sollen, muss eDataSecurityType = eDataSecurityType_Idle sein.

Anwendungsbeispiel

Download	Benötige Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
iNumberOfValues   : INT;              1..8
bEdgeNewValue     : BOOL;
bDirection        : BOOL;
eCtrlMode         : E_HVACCtrlMode;
iManualValue      : INT;              0..8
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bEnable: Über ein TRUE wird der Funktionsbaustein freigegeben. Ist *bEnable* = FALSE, so wird *rValueOut* und *iValueNumber* konstant auf 0 gesetzt.

iNumberOfValues: Gibt die Anzahl der Variablen *rValue1-8* an, die über *rValueOut* ausgegeben werden können. Ist *iNumberOfValues* = 8, so bewegt sich der Ausgang *iValueNumber* zwischen 1 und 8. Es werden dann für die Ausgabe *rValueOut* die Variablen *rValue1* bis *rValue8* berücksichtigt, siehe Anwendung [▶ 411]
iNumberOfValues bestimmt das Startverhalten der Ausgänge *rValueOut* und *iValueNumber*, siehe Anwendung [▶ 411]
iNumberOfValues wird in der Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual* nicht berücksichtigt.
iNumberOfValues darf 1 nicht unterschreiten und 8 nicht überschreiten. Ansonsten wird mit *bError* = TRUE ein Fehler angezeigt und die Abarbeitung des Funktionsbausteins wird gestoppt.

bEdgeNewValue: Liegt eine steigende Flanke an *bEdgeNewValue*, dann in- oder dekrementiert *iValueNumber*.
Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*iNumberOfValues* > 0 und es liegt eine steigende Flanke an *bEdgeNewValue*, dann inkrementiert *iValueNumber*. Ist *iValueNumber* = 6, so ist *rValueOut* = *rValue6*. Ist *iValueNumber*= *iNumberOfValues* und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = 1 und *rValueOut* = *rValue1.*
Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*iNumberOfValues* = **6** und es liegt eine steigende Flanke an *bEdgeNewValue*, dann dekrementiert *iValueNumber*. Ist *iValueNumber* = 5, so ist *rValueOut* = *rValue5*. Ist *iValueNumber*= 1 und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = *iNumberOfValues***(6)** und *rValueOut* = *rValue6.*
bEdgeNewValue wird in der Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual* nicht berücksichtigt.

bDirection: *bDirection* bestimmt den Wirksinn des Funktionsbausteins.
bDirection = FALSE bedeutet, dass *iValueNumber* in aufsteigender Form von 1 nach *iNumberOfValues* inkrementiert und dadurch den Ausgabewert *rValueOut* bestimmt, siehe Anwendung [▶ 411]
bDirection = TRUE bedeutet, dass *iValueNumber* in absteigender Form von *iNumberOfValues* nach 1 dekrementiert und dadurch den Ausgabewert *rValueOut* bestimmt, siehe Anwendung [▶ 411]
bDirection wird in der Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual* nicht berücksichtigt.

eCtrlMode: Mittels des Enums wird die Betriebsart des Funktionsbausteins vorgegeben, siehe Anwendung [▶ 411]
Ist die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Auto,* so wird die Variable *iManualValue* nicht berücksichtigt.
Ist die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual,* so werden die Variablen *iNumberOfValues, bEdgeNewValue* und *bDirection* nicht berücksichtigt.

iManualValue: Ist die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual*AND*bEnable* = TRUEAND*bError* = FALSEAND*iManualValue* = 15, so ist iValueNumber = 15. Der Inhalt der Variable *rValue15* wird dann über *rValueOut* ausgegeben, siehe Anwendung [▶ 411].
iManualValue wird nur in der Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual* berücksichtigt. Der Wert von *iValueNumber* wird von *iManualValue* bestimmt. *iManualValue* sollte im Bereich von 0 und 16 liegen. Unterschreitet *iManualValue* den Wert 0, so wird über *iValueNumber* eine 0 ausgegeben. Überschreitet *iManualValue* den Wert 16, so wird über *iValueNumber* eine 16 ausgegeben. Ist *iManualValue* = 0, so ist *iValueNumber* = 0 und *rValueOut* = *0.*

VAR_OUTPUT

```
rValueOut           : INT;
bActive             : BOOL;
bEdgeNewValueOut    : BOOL;
iValueNumber        : INT;
bError              : BOOL;
eErrorCode          : E_HVACErrorCodes;
```

rValueOut: Über *rValueOut* wird der Wert der Variablen *rValue1-8* ausgegeben.
Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*iNumberOfValues* > 0 und es liegt eine steigende Flanke an *bEdgeNewValue*, dann inkrementiert *iValueNumber*. Ist *iValueNumber* = 6, so ist *rValueOut* = *rValue6*. Ist *iValueNumber*= *iNumberOfValues* und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = 1 und *rValueOut* = *rValue1.*
Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*iNumberOfValues* = **7** und es liegt eine steigende Flanke an *bEdgeNewValue*, dann dekrementiert *iValueNumber*. Ist *iValueNumber* = 5, so ist *rValueOut* = *rValue5*. Ist *iValueNumber*= 1 und es liegt eine steigende Flanke an *bEdgeNewValue* an, dann ist *iValueNumber* = *iNumberOfValues***(7)** und *rValueOut* = *rValue7.*
Ist *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Manual,* so wird der Wert von *iValueNumber* von *iManualValue* bestimmt. Ist *iManualValue* = 7, so ist *iValueNumber* = 7 und dieses bedeutet dann, dass *rValueOut* = *rValue7* ist.

Das Startverhalten des Ausgangs *rValueOut* sieht folgendermaßen aus, wenn *bEnable* = TRUEAND*bError* = FALSEAND

- *eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*iNumberOfValues* > 0, so ist *iValueNumber* = 1 und *rValueOut* = *rValue1.*

- *eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*iNumberOfValues* = 7, so ist *iNumberOfValues* = 7 und *rValueOut* = *rValue7.*

- *eCtrlMode* = *eHVACCtrlMode_Manual*AND*iManualValue* = 13, so ist *iValueNumber* = 13 und *rValueOut* = *rValue13.*

- *eCtrlMode* = *eHVACCtrlMode_Manual*AND*iManualValue* = 0, so ist *iValueNumber* = 0 und *rValueOut* = *0.*

bActive: *bActive* wird TRUE , wenn

1. *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Auto* ist.

2. *bEnable* = TRUEAND*bError* = FALSEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND*iManualValue* > 0 ist.

bEdgeNewValueOut: Ist für einen SPS-Zyklus TRUE, wenn *bEnable* = TRUEAND*bError* = FALSEAND*iValueNumber* seinen Wert ändert.

iValueNumber: Der Ausgang *iValueNumber* zeigt immer die Nummer der Variablen *rValue***X** an, deren Inhalt in der Ausgangsvariablen *rValueOut* ausgegeben wird.
Bei *eCtrlMode* = *eHVACCtrlMode_Auto* wird durch die Variable *bDirection* bestimmt, ob bei der Verwendung des Eingangs *bEdgeNewValue* die Ausgangsvariable *iValueNumber* in- bzw. dekrementiert wird. Wenn *bDirection* = FALSE ist, wird inkrementiert, wenn *bDirection* = TRUE ist, wird dekrementiert.
Bei *eCtrlMode* = *eHVACCtrlMode_Manual* wird der Wert von *iValueNumber* von *iManualValue* bestimmt.

bError: Der Ausgang signalisiert mit einem TRUE, dass ein Fehler anliegt und ein falscher Parameter an der Variable *iNumberOfValues* anliegt. *rValueOut* und *iValueNumber* werden konstant auf 0 gesetzt und das Enum *eErrorCode* zeigt die Fehlernummer an. Nach Behebung des Fehlers muss die Meldung *bError* nicht quittiert werden.

eErrorCode: Liefert bei einem gesetzten *bError*-Ausgang die Fehlernummer [▶ 529]. Folgender Fehler kann in diesem Funktionsbaustein vorkommen:

eHVACErrorCodes_InvalidParam_iNumberOfValues: Es liegt ein fehlerhafter Wert an *iNumberOfValues* an. *iNumberOfValues* darf 1 nicht unterschreiten und 8 nicht überschreiten.

Um in der SPS an die Fehlernummern des Enums zu gelangen, kann eErrorCode einer Variablen vom Datentyp WORD zugewiesen werden.eHVACErrorCodes_InvalidParam_iNumberOfValues = 42

VAR_IN_OUT

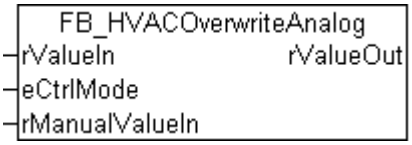
```
rValue1-8        : INT;
```

rValue1-8: Mittels der Variablen *rValue1* bis *rValue8* wird der Wert der Ausgangsvariable *rValueOut* bestimmt. Ist *iValueNumber* = 6, so ist *rValueOut* = *rValue6*.
Ist *iNumberOfValues* = 8, so bewegt sich der Ausgang *iValueNumber* zwischen 1 und 8. Es werden dann für die Ausgabe *rValueOut* die Variablen *rValue1* bis *rValue8* berücksichtigt, wenn die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Auto* ist, siehe Anwendung [▶ 411]: Die Variable wird persistent gespeichert.

Dokumente hierzu

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

3.6.18 FB_HVACOverwriteAnalog



Anwendung

Dieser Funktionsbaustein übersteuert den Eingang *rValueIn*, wenn *eCtrlMode* = *eHVACCtrlMode_Manual*, und leitet den *rManualValueIn* an den Ausgang *rValueOut* weiter.

VAR_INPUT

```
rValueIn       :REAL;
eCtrlMode      :E_HVACCtrlMode;
rManualValueIn :REAL;
```

rValueIn: Analoger Eingangswert, der an den Ausgang *rValueOut* weitergeleitet wird, wenn *eCtlrMode* = *eHVACCtrlMode_Auto* ist.

eCtrlMode: Über dieses Enum wird der Betriebsmodus ausgewählt. Hand- oder Automatikbetrieb.

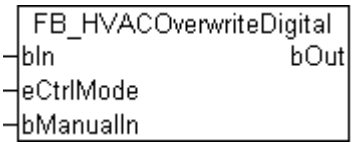
rManualValueIn: Analoger manueller Eingangswert, der an den Ausgang *rValueOut* weitergeleitet wird, wenn *eCtrlMode* = *eHVACCtrlMode_Manual* ist.

VAR_OUTPUT

```
rValueOut      :REAL;
```

rValueOut: Analoger Ausgangswert.

3.6.19 FB_HVACOverwriteDigital



Anwendung

Dieser Funktionsbaustein übersteuert den Zustand des Einganges *bIn*, wenn *eCtrlMode* = *eHVACCtrlMode_Manual*, und leitet ihn an den Ausgang *bOut* weiter.

VAR_INPUT

```
bIn            :BOOL;
eCtrlMode      :E_HVACCtrlMode;
bManualIn      :BOOL;
```

bIn: Digitale Eingangsvariable, deren Zustand an den Ausgang *bOut* weitergeleitet wird, wenn *eCtlrMode* = *eHVACCtrlMode_Auto* ist.

eCtrlMode: Über dieses Enum wird der Betriebsmodus ausgewählt. Hand- oder Automatikbetrieb.

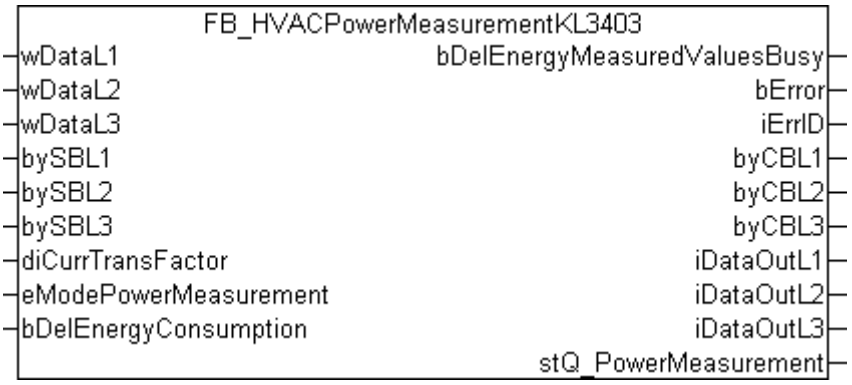
bManualIn: Digitale Eingangsvariable, deren Zustand an den Ausgang *bOut* weitergeleitet wird, wenn *eCtrlMode = eHVACCtrlMode_Manual* ist.

VAR_OUTPUT

```
bOut            :BOOL;
```

bOut: Digitale Ausgangsvariable.

3.6.20 FB_HVACPowerMeasurementKL3403



Anwendung

Dieser Funktionsbaustein dient dazu eine 3-Phasen-Leistungsmessklemme (KL/KS3403) zu steuern. Die Daten der Klemme werden ausgelesen und alle davon abhängigen Größen abgeleitet. Die Busklemme KL3403 ermöglicht die Messung aller relevanten elektrischen Daten des Versorgungsnetzes. Die Spannung wird über den direkten Anschluss von L1, L2, L3 und N gemessen. Der Strom der drei Phasen L1, L2 und L3 wird über einfache Stromwandler eingespeist. Die Messwerte aller Ströme und Spannungen stehen als Effektivwert zur Verfügung. In der KL3403 wird für jede Phase die Wirkleistung und der Energieverbrauch berechnet. Durch den Bezug der Effektivwerte von Spannung U * Strom I zur Wirkleistung P können alle weiteren Informationen wie Scheinleistung S oder der Phasenverschiebungswinkel cos φ abgeleitet werden. Die KL3403 stellt jedem Feldbus eine umfangreiche Netzanalyse und die Möglichkeit zu einem Energiemanagement zur Verfügung. Die Daten werden in 8 Gruppen nacheinander gelesen. Abhängige Werte werden zyklisch berechnet. Die Energiemessung wird mit dem Überlauf aus Register 1 als 32 Bit-Wert aus der Klemme gelesen. Zum Löschen dieses Zählwertes kann der Eingang **bDelEnergyConsumption** genutzt werden.

Die Ergebnisse der Leistungsmessung stehen in der Ausgangsstruktur *ST_HVACPowerMeasurement*zur Verfügung. Die Struktur ist 120 Bytes lang.

Die Ein- und Ausgangsvariablen wDataL1, wDataL2, wDataL3, bySBL1, bySBL2, bySBL3, byCBL1, byCBL2, byCBL3, iDataOutL1, iDataOutL2 und iDataOutL3 müssen mit der Busklemme KL3403 verknüpft sein. Diese werden benötigt, um sämtliche Daten aus der Klemme zu bekommen.

TYPE ST_HVACPowerMeasurement

```
Name                    : Type       Unit    Description

STRUCT
  diIL1, diIL2, diIL3          : DINT;       A    Ieff Strom (Effektivwert) Auflösung: 0,1 A
  diUL1, diUL2, diUL3          : DINT;       V    Ueff Spannung (Effektivwert) Auflösung: 0,1 V
  diPL1, diPL2, diPL3          : DINT;      kW    P Wirkleistung pro Phase Auflösung: 0,1 kW
  diPg                  : DINT;      kW    Pges Wirkleistung Auflösung: 0,1 kW
  diCosPhiL1, diCosPhiL2, diCosPhiL3   : DINT;        cosPhi Leistungsfaktor pro Phase Auflösung: 0,
01
  diCosPhi                 : DINT;        cosPhiges Leistungsfaktor 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 Spitzenwert des Stroms Auflösung: 0,1 A
  diUmaxL1, diUmaxL2, diUmaxL3     : DINT;       V    Umax Spitzenwert der Spannung Resolution: 0,1
V
  diPmaxL1, diPmaxL2, diPmaxL3     : DINT;      kW    Pmax Spitzenwert der Wirkleistung Resolution:
0,1 kW
```

```
  diSg                 : DINT;      kVA       Sges Scheinleistung Resolution: 0,1 kVA
  diQg                 : DINT;      kvar      Qges Blindleistung Resolution: 0,1 kvar
  dummy                : DINT;        Reserve, füllt die Struktur auf 120Bytes auf
END_STRUCT
```

VAR_INPUT

```
wDataL1, wDataL2, wDataL3   : WORD;
bySBL1, bySBL2, bySBL3      : BYTE;
diCurrTransFactor           : DINT;
eModePowerMeasurement       : E_HVACPowerMeasurementMode;
bDelEnergyConsumption       : BOOL;
```

wDataLx: Eingangsdaten aus den drei Kanälen der KL3403.

bySBLx: Statusbyte der drei Kanäle der KL3403. Meldet zurück welcher Wert über den Eingang (*wDataLx*) gelesen werden kann.

diCurrTransFactor: Transformationsfaktor des Stromwandlers, dient zur Umrechnung auf den tatsächlichen Strangstrom und der damit verbundenen Werte.
Da unabhängig vom Klemmentyp der Messendwert immer dez. 1000 ergibt (100,0%), muss für den Faktor der primäre Endwert des Wandler angegeben werden.

Beispiel KL3403-0000: Ein 400/1A Wandler ergibt einen *diCurrTransFactor* von *400.* Bei einem Messwert von max. 1A, ergibt sich bei einer internen Auflösung von 0,001A ein Endwert von dez. 1000 (100,0%) * *diCurrTransFactor = 400A.*

Beispiel KL3403-0010: Ein 400/5A Wandler ergibt einen *diCurrTransFactor* von *400.* Bei einem Messwert von max. 5A, ergibt sich bei einer internen Auflösung von 0,005A ein Endwert von dez. 1000 (100,0%) * *diCurrTransFactor = 400A.*

eModePowerMeasurement: Wenn dieser Parameter im Wertbereich zwischen 1 und 8 liegt, wird das automatische Auslesen aller Daten unterbrochen. Es wird nur die entsprechende ausgewählte Messgröße pro Zyklus gelesen.
TYPE E_HVACPowerMeasurementMode :
(
eHVACPowerMeasurementMode_AutoAllValues := 0,
eHVACPowerMeasurementMode_Current := 1,
eHVACPowerMeasurementMode_Voltage := 2,
eHVACPowerMeasurementMode_EffectivePower := 3,
eHVACPowerMeasurementMode_PowerFactor := 4,
eHVACPowerMeasurementMode_EnergyConsumption := 5,
eHVACPowerMeasurementMode_PeakCurrentValu := 6, [1])
eHVACPowerMeasurementMode_PeakVoltageValue := 7, [1])
eHVACPowerMeasurementMode_PeakPowerValue := 8[1])
);
END_TYPE

bDelEnergyConsumption : Positive Flanke an diesem Eingang löscht den Energieverbrauch im EEPROM. Der Energieverbrauch wird im RAM gezählt und zyklisch alle 15 Minuten in das EEPROM gespeichert. Dort bleibt er auch beim Abschalten der KL3403 erhalten.

[1]) Die Minimal- und Spitzenwerte werden beim Abschalten der KL3403 gelöscht.

VAR_OUTPUT

```
BDelEnergyMeasuredValuesBusy      : BOOL;
bError                            : BOOL;
iErrID                            : UDINT;
byCBL1, byCBL2, byCBL3            : BYTE;
iDataOutL1, iDataOutL2, iDataOutL3: INT;
stQ_PowerMeasurement              : ST_HVACPowerMeasurement;
```

bDelEnergyMeasuredValuesBusy: Da die Energiemesswerte aus dem internen EEPROM gelöscht werden müssen, wird während dieser Zeit kein Wert aktualisiert und das Flag *bDelEnergyMeasuredValuesBusy* zeigt TRUE.

bError: Wenn TRUE, dann ist ein Fehler in der Registerkommunikation aufgetreten.

iErrID: Fehler ID der Registerkommunikation

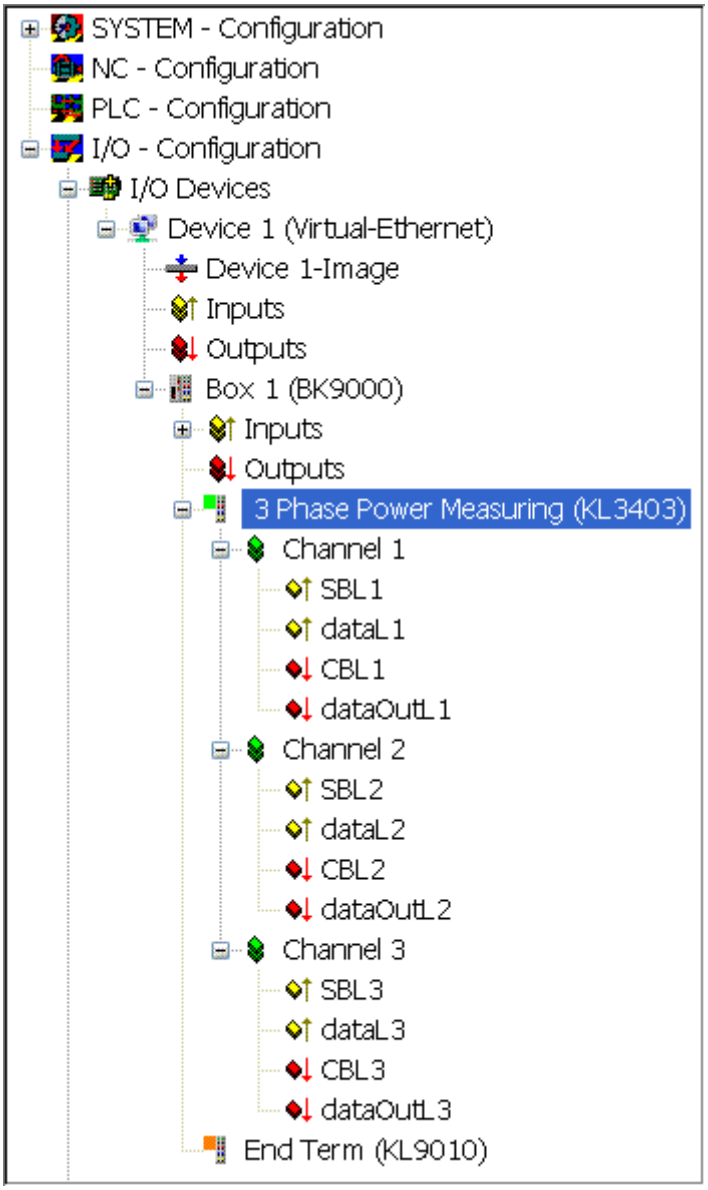
0x100 Timeout-Fehler. Die zulässige Ausführungszeit wurde überschritten.
0x200 Parameter Fehler (z.B. bei einer unzulässigen Registernummer).
0x300 Der gelesene Wert unterscheidet sich von dem geschriebenen Wert (Schreibzugriff auf diesen Register möglicherweise nicht erlaubt oder fehlgeschlagen)

byCBLx: Dieser Ausgang dient zur Auswahl des gewünschten Eingangswertes und zur Registerkommunikation um den Energieverbrauch zu löschen.

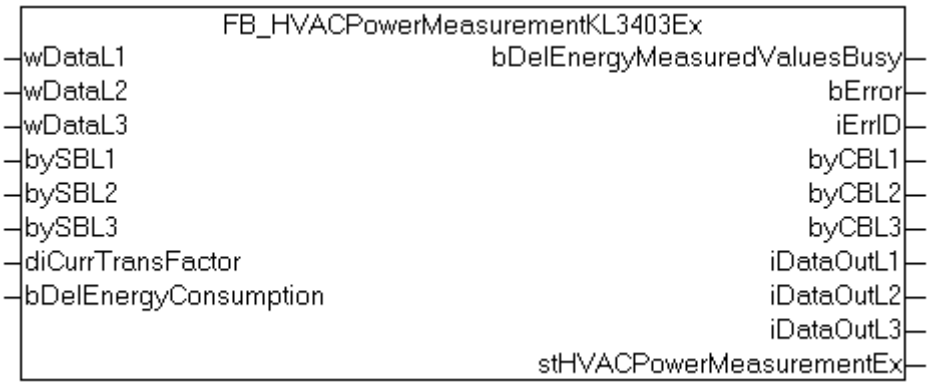
iDataOutLx: Dieser Ausgang dient zur Registerkommunikation um den Energieverbrauch zu löschen.

stQ_PowerMeasurement: Die Ergebnisse der Leistungsmessung als Ausgangsdatenstruktur.

Das Beispiel zeigt, wie eine KL3403-Klemme mit den Ein- und Ausgangsvariablen der SPS verknüpft werden muss



3.6.21 FB_HVACPowerMeasurementKL3403EX



Anwendung

Dieser Funktionsbaustein dient dazu eine 3-Phasen-Leistungsmessklemme (KL/KS3403) zu steuern. Die Daten der Klemme werden ausgelesen und alle davon abhängigen Größen abgeleitet. Die Busklemme KL3403 ermöglicht die Messung aller relevanten elektrischen Daten des Versorgungsnetzes. Die Spannung wird über den direkten Anschluss von L1, L2, L3 und N gemessen. Der Strom der drei Phasen L1, L2 und L3 wird über einfache Stromwandler eingespeist. Die Messwerte aller Ströme und Spannungen stehen als Effektivwert zur Verfügung. In der KL3403 wird für jede Phase die Wirkleistung und der Energieverbrauch berechnet. Durch den Bezug der Effektivwerte von Spannung U * Strom I zur Wirkleistung P können alle weiteren Informationen wie Scheinleistung S oder der Phasenverschiebungswinkel cos φ abgeleitet werden. Die KL3403 stellt jedem Feldbus eine umfangreiche Netzanalyse und die Möglichkeit zu einem Energiemanagement zur Verfügung. Die Daten werden in 8 Gruppen nacheinander gelesen. Abhängige Werte werden zyklisch berechnet. Die Energiemessung wird mit dem Überlauf aus Register 1 als 32 Bit-Wert aus der Klemme gelesen. Zum Löschen dieses Zählwertes kann der Eingang **bDelEnergyConsumption** genutzt werden.

Die Ergebnisse der Leistungsmessung stehen in der Ausgangsstruktur *ST_HVACPowerMeasurementEx*zur Verfügung.

Der Unterschied zum bisherigen Baustein FB_HVACPowerMeasurementKL3403 besteht in dem Ausgabeformat der Ergebnisse. Die Ergebnisse stehen in der Struktur *ST_HVACPowerMeasurementEx* im LREAL Format zur Verfügung. Die Ausgabe wurde um die Frequenzen der drei Phasen erweitert.

Die Ein- und Ausgangsvariablen wDataL1, wDataL2, wDataL3, bySBL1, bySBL2, bySBL3, byCBL1, byCBL2, byCBL3, iDataOutL1, iDataOutL2 und iDataOutL3 müssen mit der Busklemme KL3403 verknüpft sein. Diese werden benötigt, um sämtliche Daten aus der Klemme zu bekommen.

TYPE ST_HVACPowerMeasurementEx

```
Name                    : Type       Unit       Description
STRUCT
  fIL1, fIL2, fIL3                  : LREAL;      A     Ieff Strom (Effektivwert)
  fIg                   : LREAL;         A    Gesamt Strom (Effektivwert)
  fUL1, fUL2, fUL3                  : LREAL;      V     Ueff Spannung (Effektivwert)
  fPL1, fPL2, fPL3                  : LREAL;      kW    P Wirkleistung pro Phase
  fPg                   : LREAL;      kW    Pges Wirkleistung
  fCosPhiL1, fCosPhiL2, fCosPhiL3          : LREAL;        cosPhi Leistungsfaktor pro Phase
  fCosPhi                   : LREAL;        cosPhiges Leistungsfaktor
  fWL1, fWL2, fWL3                  : LREAL;      kWh       Energieverbrauch
  fWg                   : LREAL;      kWh       Energieverbrauch
  fImaxL1, fImaxL2, fImaxL3             : LREAL;       A    Imax Spitzenwert des Stroms
  fUmaxL1, fUmaxL2, fUmaxL3             : LREAL;       V    Umax Spitzenwert der Spannung
  fPmaxL1, fPmaxL2, fPmaxL3             : LREAL;      kW    Pmax Spitzenwert der Wirkleistung
  fSg                   : LREAL;      kVA       Sges Scheinleistung
  fQg                   : LREAL;      kvar      Qges Blindleistung
  fFrequencyL1, fFrequencyL2, fFrequencyL3      : LREAL;       Hz       Frequenz
END_STRUCT
```

VAR_INPUT

```
wDataL1, wDataL2, wDataL3   : WORD;
bySBL1, bySBL2, bySBL3      : BYTE;
diCurrTransFactor           : LREAL;
bDelEnergyConsumption       : BOOL;
```

wDataLx: Eingangsdaten aus den drei Kanälen der KL3403.

bySBLx: Statusbyte der drei Kanäle der KL3403. Meldet zurück welcher Wert über den Eingang (*wDataLx*) gelesen werden kann.

diCurrTransFactor: Transformationsfaktor des Stromwandlers, dient zur Umrechnung auf den tatsächlichen Strangstrom und der damit verbundenen Werte.

Da unabhängig vom Klemmentyp der Messendwert immer dez. 1000 ergibt (100,0%), muss für den Faktor der primäre Endwert des Wandler angegeben werden.

Beispiel KL3403-0000: Ein 400/1A Wandler ergibt einen *diCurrTransFactor* von *400.* Bei einem Messwert von max. 1A, ergibt sich bei einer internen Auflösung von 0,001A ein Endwert von dez. 1000 (100,0%) * *diCurrTransFactor = 400A.*

Beispiel KL3403-0010: Ein 400/5A Wandler ergibt einen *diCurrTransFactor* von *400.* Bei einem Messwert von max. 5A, ergibt sich bei einer internen Auflösung von 0,005A ein Endwert von dez. 1000 (100,0%) * *diCurrTransFactor = 400A.*

bDelEnergyConsumption : Positive Flanke an diesem Eingang löscht den Energieverbrauch im EEPROM. Der Energieverbrauch wird im RAM gezählt und zyklisch alle 15 Minuten in das EEPROM gespeichert. Dort bleibt er auch beim Abschalten der KL3403 erhalten.

[1]) Die Minimal- und Spitzenwerte werden beim Abschalten der KL3403 gelöscht.

VAR_OUTPUT

```
BDelEnergyMeasuredValuesBusy      : BOOL;
bError                            : BOOL;
iErrID                            : UDINT;
byCBL1, byCBL2, byCBL3            : BYTE;
iDataOutL1, iDataOutL2, iDataOutL3: INT;
stQ_PowerMeasurementEx            : ST_HVACPowerMeasurementEx;
```

bDelEnergyMeasuredValuesBusy: Da die Energiemesswerte aus dem internen EEPROM gelöscht werden müssen, wird während dieser Zeit kein Wert aktualisiert und das Flag *bDelEnergyMeasuredValuesBusy* zeigt TRUE.

bError: Wenn TRUE, dann ist ein Fehler in der Registerkommunikation aufgetreten.

iErrID: Fehler ID der Registerkommunikation

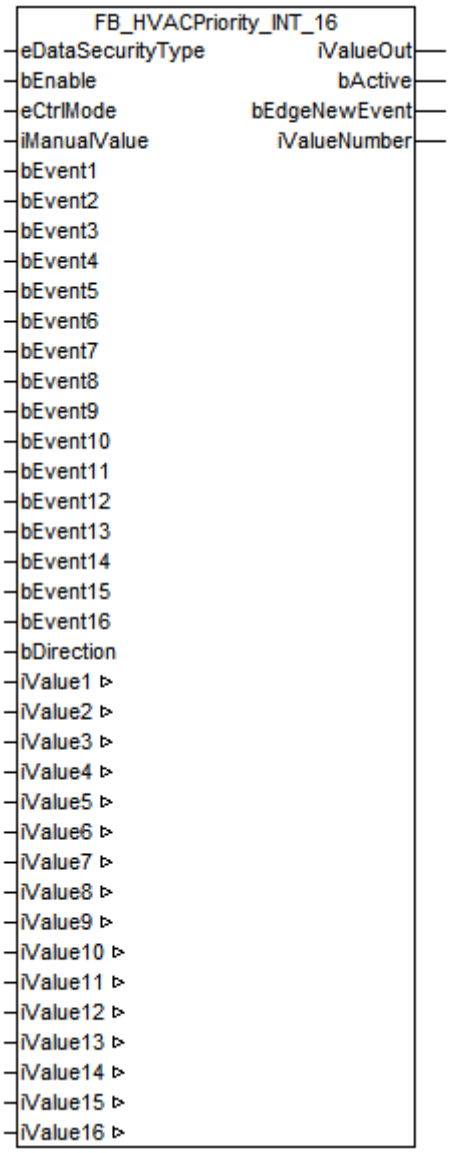
0x100 Timeout-Fehler. Die zulässige Ausführungszeit wurde überschritten.
0x200 Parameter Fehler (z.B. bei einer unzulässigen Registernummer).
0x300 Der gelesene Wert unterscheidet sich von dem geschriebenen Wert (Schreibzugriff auf diesen Register möglicherweise nicht erlaubt oder fehlgeschlagen)

byCBLx: Dieser Ausgang dient zur Auswahl des gewünschten Eingangswertes und zur Registerkommunikation, um den Energieverbrauch zu löschen.

iDataOutLx: Dieser Ausgang dient zur Registerkommunikation, um den Energieverbrauch zu löschen.

stQ_PowerMeasurementEx: Die Ergebnisse der Leistungsmessung als Ausgangsdatenstruktur.

3.6.22 FB_HVACPriority_INT_16



Anwendung

Der Funktionsbaustein kann zur Priorisierung von Ereignissen oder als Muliplexer eingesetzt werden. Diese Auswahl wird mittels des Enums *eCtrlMode* vorgenommen.

Der Funktionsbaustein kann zur Priorisierung von Ereignissen verwendet werden, wenn *eCtrlMode* = *eHVACCtrlMode_Auto* ist. Der Ausgang *iValueOut wird* über die eintretenden Ereignisse der Eingänge *bEvent1-16*, *iValue1-16* und dem Wirksinn der Priorisierung *bDirection* gesteuert.
bDirection = FALSE bedeutet, dass die Ereignisse in aufsteigender Form von 1 nach 16 eine höhere Priorität besitzen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*bEvent15* = TRUE (höchstes eingetretenes Ereignis in Tabelle 1 [▶ 422] Spalte 1), dann hat *iValueOut* den Wert von *iValue15*.
bDirection = TRUE bedeutet, dass die Ereignisse in absteigender Form von 16 nach 1 eine höhere Priorität besitzen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*bEvent1* = TRUE (niedrigstes eingetretenes Ereignis in Tabelle 2 [▶ 422] Spalte 1), dann hat *iValueOut* den Wert von *iValue1*.

Der Funktionsbaustein kann als Multiplexer verwendet werden, wenn *eCtrlMode* = *eHVACCtrlMode_Manual* ist. Der Wert von *iManualValue* bezieht sich auf eine der VAR_IN_OUT-Variablen *iValue1-16*, deren Wert über *iValueOut* ausgegeben wird. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND *iManualValue* = 2, so ist *iValueOut* = *iValue2,* siehe Tabelle 3 [▶ 423].

HINWEIS

Eine sich häufig ändernde Variable darf nicht an die VAR_IN_OUT-Variablen iValue1-16 angelegt werden, wenn eDataSecurityType = eHVACDataSecurityType_Persistentist. Dieses würde zu einem frühzeitigen Verschleiß des Speichermediums der Steuerung führen. Wenn sich die VAR_IN_OUT-Variablen iValue1-16 häufig ändern und nicht persistent abgelegt werden sollen, muss eDataSecurityType = eDataSecurityType_Idle sein.

Tabelle 1: Priorisierung von Ereignissen in aufsteigender Form von 1 nach 16

Tabelle 1

In der Tabelle 1 ist zu sehen, dass *bDirection* = FALSE ist. Das bedeutet, dass die Ereignisse in aufsteigender Form von 1 nach 16 den Ausgabewert *iValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*bEvent15* = TRUE (höchstes eingetretenes Ereignis in Spalte 1), dann ist *iValueOut* = *iValue15*.

	bEnable = TRUE eCtrlMode = eHVACCtrlMode_Auto bDircetion = FALSE					
bEvent1	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent2	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent3	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent4	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent5	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent6	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent7	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent8	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent9	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent10	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent11	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
bEvent12	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent13	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent14	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
bEvent15	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent16	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
iValueOut	iValue15	iValue8	iValue16	iValue14	0	iValue16
iValueNumber	15	8	16	14	0	16

Tabelle 2: Priorisierung von Ereignissen in absteigender Form von 16 nach 1

Tabelle 2

In der Tabelle 2 ist zu sehen, dass *bDirection* = TRUE ist. Das bedeutet, dass die Ereignisse in absteigender Form von 16 nach 1 den Ausgabewert *iValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*bEvent1* = TRUE (niedrigstes eingetretenes Ereignis in Tabelle 2 [▶ 422] Spalte 1), dann ist *iValueOut* = *iValue1*.

	bEnable = TRUE eCtrlMode = eHVACCtrlMode_Auto bDircetion = TRUE					
bEvent1	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent2	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent3	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent4	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent5	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent6	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent7	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent8	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent9	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent10	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent11	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
bEvent12	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent13	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent14	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
bEvent15	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent16	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
iValueOut	iValue1	iValue1	iValue9	iValue3	0	iValue1
iValueNumber	1	1	9	3	0	1

Tabelle 3: Multiplexer

Tabelle 3

Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND*iManualValue* = *2,* dann ist *iValueOut* = 2.

	bEnable = TRUE eCtrlMode = eHVACCtrlMode_Manual					
iManualValue	2	3	5	15	0	16
iValueOut	iValue2	iValue3	iValue5	iValue15	0	iValue16
iValueNumber	2	3	5	15	0	16

Anwendungsbeispiel

Download	Benötige Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
bEvent1 - bEvent8 : BOOL;
eCtrlMode         : E_HVACCtrlMode;
iManualValue      : INT;             0..16
bDirection        : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bEnable: Über ein TRUE wird der Funktionsbaustein freigegeben. Ist *bEnable* = FALSE, so wird *iValueOut* konstant auf 0 gesetzt.

eCtrlMode: Mittels des Enums wird die Betriebsart des Funktionsbausteins vorgegeben.
Ist *eCtrlMode* = *eHVACCtrlMode_Auto*, so stellt der Funktionsbaustein eine Priorisierung von Ereignissen dar. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto* so wird der Ausgang *iValueOut* über die eintretenden Ereignisse der Eingänge *bEvent1-16, iValue1-16* und des Wirksinns der Priorisierung *bDirection* gesteuert, siehe Tabelle 1 [▶ 422] und Tabelle 2 [▶ 422]
Ist *eCtrlMode* = *eHVACCtrlMode_Manual*, so stellt der Funktionsbaustein einen Multiplexer dar. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual, so* wird der Wert des Ausgangs *iValueOut* über *iManualValue* gesteuert. Ist *iManualProfile* = 5, so ist *iValueOut* = *iValue5*, siehe Tabelle 3 [▶ 423]

iManualProfile: Ist die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual* (Multiplexer) AND*bEnable* = TRUEAND*bError* = FALSE*, so* wird der Wert des Ausgangs *iValueOut* über *iManualValue* gesteuert. Ist *iManualProfile* = 5, so ist *iValueOut* = *iValue5*, siehe Tabelle 3 [▶ 423]

bEvent1-16: Der Ausgang *iValueOut* wird über die eintretenden Ereignisse der Eingänge *bEvent1-16*, *iValue1-16* und dem Wirksinn der Priorisierung *bDirection* gesteuert.
bDirection = FALSE bedeutet, dass die Ereignisse in aufsteigender Form von 1 nach 16 den Ausgabewert *iValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*bEvent15* = TRUE (höchstes eingetretenes Ereignis in Tabelle 1 [▶ 422] Spalte 1), dann ist *iValueOut* = *iValue15*.
bDirection = TRUE bedeutet, dass die Ereignisse in absteigender Form von 16 nach 1 den Ausgabewert *iValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*bEvent1* = TRUE (niedrigstes eingetretenes Ereignis in Tabelle 2 [▶ 422] Spalte 1), dann ist *iValueOut* = *iValue1*.
Die Variablen *bEvent1-16* und *bDirection* werden nur dann berücksichtigt, wenn *eCtrlMode* = *eHVACCtrlMode_Auto* ist.

bDirection: Der Funktionsbaustein kann als Priorisierung von Ereignissen verwendet werden, wenn *eCtrlMode* = *eHVACCtrlMode_Auto* ist. Der Ausgang *iValueOut wird* über die eintretenden Ereignisse der Eingänge *bEvent1-16*, *iValue1-16* und dem Wirksinn der Priorisierung *bDirection* gesteuert.
bDirection = FALSE bedeutet, dass die Ereignisse in aufsteigender Form von 1 nach 16 den Ausgabewert *iValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*bEvent15* = TRUE (höchstes eingetretenes Ereignis in Tabelle 1 [▶ 422] Spalte 1), dann ist *iValueOut* = *iValue15*.
bDirection = TRUE bedeutet, dass die Ereignisse in absteigender Form von 16 nach 1 den Ausgabewert *iValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*bEvent1* = TRUE (niedrigstes eingetretenes Ereignis in Tabelle 2 [▶ 422] Spalte 1), dann ist *iValueOut* = *iValue1*.
Die Variablen *bEvent1-16* und *bDirection* werden nur dann berücksichtigt, wenn *eCtrlMode* = *eHVACCtrlMode_Auto* ist.

VAR_OUTPUT

```
iValueOut           : INT;
bActive             : BOOL;
bEdgeNewEvent       : BOOL;
iValueNumber        : INT;
```

iValueOut: Der Funktionsbaustein kann als Priorisierung von Ereignissen oder als Muliplexer eingesetzt werden. Diese Auswahl wird mittels des Enums *eCtrlMode* vorgenommen.

Der Funktionsbaustein kann als Priorisierung von Ereignissen verwendet werden, wenn *eCtrlMode* = *eHVACCtrlMode_Auto* ist. Der Ausgang *iValueOut wird* über die eintretenden Ereignisse der Eingänge *bEvent1-16*, *iValue1-16* und dem Wirksinn der Priorisierung *bDirection* gesteuert.
bDirection = FALSE bedeutet, dass die Ereignisse in aufsteigender Form von 1 nach 16 den Ausgabewert *iValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*bEvent15* = TRUE (höchstes eingetretenes Ereignis in Tabelle 1 [▶ 422] Spalte 1), dann ist *iValueOut* = *iValue15*.
bDirection = TRUE bedeutet, dass die Ereignisse in absteigender Form von 16 nach 1 den Ausgabewert *iValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*bEvent1* = TRUE (niedrigstes eingetretenes Ereignis in Tabelle 2 [▶ 422] Spalte 1), dann ist *iValueOut* = *iValue1*.

Der Funktionsbaustein kann als Multiplexer verwendet werden, wenn *eCtrlMode* = *eHVACCtrlMode_Manual* ist. Der Wert von *iManualValue* bezieht sich auf eine von den VAR_IN_OUT-Variablen *iValue1-16*, deren Wert über *iValueOut* ausgegeben wird. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND *iManualValue* = 2, so ist *iValueOut* = *iValue2,* siehe Tabelle 3 [▶ 423].

bActive: *bActive* wird TRUE , wenn

1. *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto* und eine der Eingangsvariablen *bEvent1-16* = TRUE ist.

2. *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND*iManualValue* > 0 ist

bEdgeNewEvent: Ist für einen SPS-Zyklus TRUE, wenn

1. *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto* und sich das entscheidende Ereignis *(bEvent1-16)* zur Steuerung des Ausgangs *iValueOut* mittels *iValue1-16* geändert hat.

2. *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual* und mit jeder Änderung *von iManualValue*

iValueNumber: Mittels der Variable *iValueNumber* wird angezeigt, von welcher Variable der Wert an *iValueOut* ausgegeben wird. Ist *iValueOut* = *iValue7*, so ist *iValueNumber* = 7, siehe Tabelle 1 [▶ 422], Tabelle 2 [▶ 422] und Tabelle 3 [▶ 423]

VAR_IN_OUT

```
Name              : Type        Persistent
iValue1-16        : INT;        X
```

iValue1-16: Mittels der Variablen *iValue1* bis *iValue16* wird der Wert der Ausgangsvariable *iValueOut* bestimmt. *iValueOut* = *iValueX*

Wird der Funktionsbaustein als Priorisierung von Ereignissen eingesetzt, so wird jede der Variablen *iValue1-16* einem Ereignis zugeordnet. *iValue1* wird dem Ereignis *bEvent1* zugeordnet, *iValue2* dem Ereignis *bEvent2*, *iValue3* dem Ereignis *bEvent3*,..., *iValue16* dem Ereignis *bEvent16*

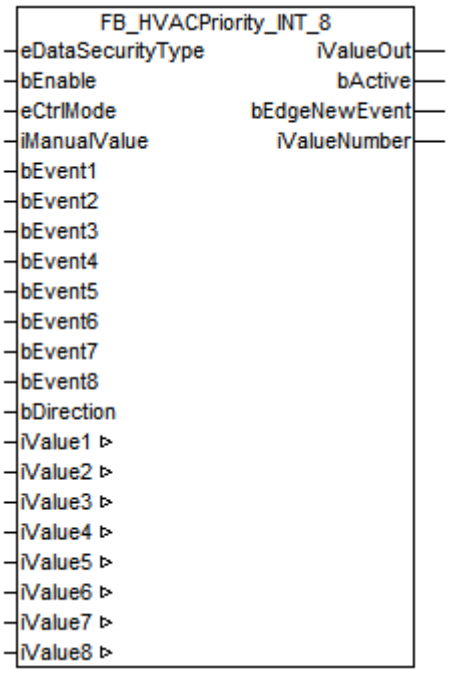
Wird der Funktionsbaustein als Muliplexer eingesetzt werden, so wird jede der Variablen *iValue1-16* dem Wert von *iManualValue* zugeordnet. Ist *iManualValue* = 1, dann ist *iValueOut* = *iValue1*. Ist *iManualValue* = 2, dann ist *iValueOut* = *iValue2*.... Ist *iManualValue* = 16, dann ist *iValueOut* = *iValue16*.

Die Variable wird persistent gespeichert.

Dokumente hierzu

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

3.6.23 FB_HVACPriority_INT_8



Anwendung

Der Funktionsbaustein kann als Priorisierung von Ereignissen oder als Muliplexer eingesetzt werden. Diese Auswahl wird mittels des Enums *eCtrlMode* vorgenommen.

Der Funktionsbaustein kann als Priorisierung von Ereignissen verwendet werden, wenn *eCtrlMode* = *eHVACCtrlMode_Auto* ist. Der Ausgang *iValueOut wird* über die eintretenden Ereignisse der Eingänge *bEvent1-8*, *iValue1-8* und dem Wirksinn der Priorisierung *bDirection* gesteuert.
bDirection = FALSE bedeutet, dass die Ereignisse in aufsteigender Form von 1 nach 8 eine höhere Priorität besitzen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*bEvent7* = TRUE (höchstes eingetretenes Ereignis in Tabelle 1 [▶ 426] Spalte 1), dann hat *iValueOut* den Wert von *iValue7*.
bDirection = TRUE bedeutet, dass die Ereignisse in absteigender Form von 8 nach 1 eine höhere Priorität besitzen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*bEvent1* = TRUE (niedrigstes eingetretenes Ereignis in Tabelle 2 [▶ 426] Spalte 1), dann hat *iValueOut* den Wert von *iValue1*.

Der Funktionsbaustein kann als Multiplexer verwendet werden, wenn *eCtrlMode* = *eHVACCtrlMode_Manual* ist. Der Wert von *iManualValue* bezieht sich auf eine der VAR_IN_OUT-Variablen *iValue1-8*, deren Wert über *iValueOut* ausgegeben wird. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND *iManualValue* = 2, so ist *iValueOut* = *iValue2,* siehe Tabelle 3 [▶ 427].

HINWEIS

Eine sich häufig ändernde Variable darf nicht an die VAR_IN_OUT-Variablen iValue1-8 angelegt werden, wenn eDataSecurityType = eHVACDataSecurityType_Persistentist. Dieses würde zu einem frühzeitigen Verschleiß des Speichermediums der Steuerung führen. Wenn sich die VAR_IN_OUT-Variablen iValue1-8 häufig ändern und nicht persistent abgelegt werden sollen, muss eDataSecurityType = eDataSecurityType_Idle sein.

Tabelle 1: Priorisierung von Ereignissen in aufsteigender Form von 1 nach 8

Tabelle 1

In der Tabelle 1 ist zu sehen, dass *bDirection* = FALSE ist. Das bedeutet, dass die Ereignisse in aufsteigender Form von 1 nach 8 den Ausgabewert *iValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*bEvent7* = TRUE (höchstes eingetretenes Ereignis in Spalte 1), dann ist *iValueOut* = *iValue7*.

	bEnable = TRUE eCtrlMode = eHVACCtrlMode_Auto bDircetion = FALSE					
bEvent1	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE
bEvent2	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE
bEvent3	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE
bEvent4	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent5	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent6	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
bEvent7	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
bEvent8	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
iValueOut	iValue7	iValue8	0	iValue6	iValue3	iValue5
iValueNumber	7	8	0	6	3	5

Tabelle 2: Priorisierung von Ereignissen in absteigender Form von 8 nach 1

Tabelle 2

In der Tabelle 2 ist zu sehen, dass *bDirection* = TRUE ist. Das bedeutet, dass die Ereignisse in absteigender Form von 8 nach 1 den Ausgabewert *iValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*bEvent1* = TRUE (niedrigstes eingetretenes Ereignis in Tabelle 2 [▶ 427] Spalte 1), dann ist *iValueOut* = *iValue1*.

	bEnable = TRUE eCtrlMode = eHVACCtrlMode_Auto bDircetion = TRUE					
bEvent1	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE
bEvent2	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE
bEvent3	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE
bEvent4	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent5	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent6	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
bEvent7	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
bEvent8	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
iValueOut	iValue1	iValue1	0	iValue3	iValue1	iValue1
iValueNumber	1	1	0	3	1	1

Tabelle 3: Multiplexer

Tabelle 3

Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND*iManualValue* = *2,* dann ist *iValueOut* = 2.

	bEnable = TRUE eCtrlMode = eHVACCtrlMode_Manual					
iManualValue	2	3	5	7	0	8
iValueOut	iValue2	iValue3	iValue5	iValue7	0	iValue8
iValueNumber	2	3	5	7	0	8

Anwendungsbeispiel

Download	Benötige Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
bEvent1 - bEvent8 : BOOL;
eCtrlMode         : E_HVACCtrlMode;
iManualValue      : INT;             0..8
bDirection        : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bEnable: Über ein TRUE wird der Funktionsbaustein freigegeben. Ist *bEnable* = FALSE, so wird *iValueOut* konstant auf 0 gesetzt.

eCtrlMode: Mittels des Enums wird die Betriebsart des Funktionsbausteins vorgegeben.
Ist *eCtrlMode* = *eHVACCtrlMode_Auto*, so stellt der Funktionsbaustein eine Priorisierung von Ereignissen dar. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto* so wird der Ausgang *iValueOut* über die eintretenden Ereignisse der Eingänge *bEvent1-8, iValue1-8* und des Wirksinns der Priorisierung *bDirection* gesteuert, siehe Tabelle 1 [▶ 426] und Tabelle 2 [▶ 427]
Ist *eCtrlMode* = *eHVACCtrlMode_Manual*, so stellt der Funktionsbaustein einen Multiplexer dar. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual, so* wird der Wert des Ausgangs *iValueOut* über *iManualValue* gesteuert. Ist *iManualProfile* = 5, so ist *iValueOut* = *iValue5*, siehe Tabelle 3 [▶ 427]

iManualProfile: Ist die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual* (Multiplexer) AND*bEnable* = TRUEAND*bError* = FALSE*, so* wird der Wert des Ausgangs *iValueOut* über *iManualValue* gesteuert. Ist *iManualProfile* = 5, so ist *iValueOut* = *iValue5*, siehe Tabelle 3 [▶ 427]

bEvent1-8: Der Ausgang *iValueOut* wird über die eintretenden Ereignisse der Eingänge *bEvent1-8*, *iValue1-8* und dem Wirksinn der Priorisierung *bDirection* gesteuert.
bDirection = FALSE bedeutet, dass die Ereignisse in aufsteigender Form von 1 nach 8 den Ausgabewert *iValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*bEvent7* = TRUE (höchstes eingetretenes Ereignis in Tabelle 1 [▶ 426] Spalte 1), dann ist *iValueOut* = *iValue7*.
bDirection = TRUE bedeutet, dass die Ereignisse in absteigender Form von 8 nach 1 den Ausgabewert *iValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*bEvent1* = TRUE (niedrigstes eingetretenes Ereignis in Tabelle 2 [▶ 426] Spalte 1), dann ist *iValueOut* = *iValue1*.
Die Variablen *bEvent1-8* und *bDirection* werden nur dann berücksichtigt, wenn *eCtrlMode* = *eHVACCtrlMode_Auto* ist.

bDirection: Der Funktionsbaustein kann als Priorisierung von Ereignissen verwendet werden, wenn *eCtrlMode* = *eHVACCtrlMode_Auto* ist. Der Ausgang *iValueOut wird* über die eintretenden Ereignisse der Eingänge *bEvent1-8*, *iValue1-8* und dem Wirksinn der Priorisierung *bDirection* gesteuert.
bDirection = FALSE bedeutet, dass die Ereignisse in aufsteigender Form von 1 nach 8 den Ausgabewert *iValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*bEvent7* = TRUE (höchstes eingetretenes Ereignis in Tabelle 1 [▶ 426] Spalte 1), dann ist

iValueOut = *iValue7*.
bDirection = TRUE bedeutet, dass die Ereignisse in absteigender Form von 8 nach 1 den Ausgabewert *iValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*bEvent1* = TRUE (niedrigstes eingetretenes Ereignis in Tabelle 2 [▶ 426] Spalte 1), dann ist *iValueOut* = *iValue1*.
Die Variablen *bEvent1-8* und *bDirection* werden nur dann berücksichtigt, wenn *eCtrlMode* = *eHVACCtrlMode_Auto* ist.

VAR_OUTPUT

```
iValueOut           : INT;
bActive             : BOOL;
bEdgeNewEvent       : BOOL;
iValueNumber        : INT;
```

iValueOut: Der Funktionsbaustein kann als Priorisierung von Ereignissen oder als Muliplexer eingesetzt werden. Diese Auswahl wird mittels des Enums *eCtrlMode* vorgenommen.

Der Funktionsbaustein kann als Priorisierung von Ereignissen verwendet werden, wenn *eCtrlMode* = *eHVACCtrlMode_Auto* ist. Der Ausgang *iValueOut wird* über die eintretenden Ereignisse der Eingänge *bEvent1-8*, *iValue1-8* und dem Wirksinn der Priorisierung *bDirection* gesteuert.
bDirection = FALSE bedeutet, dass die Ereignisse in aufsteigender Form von 1 nach 8 den Ausgabewert *iValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*bEvent7* = TRUE (höchstes eingetretenes Ereignis in Tabelle 1 [▶ 426] Spalte 1), dann ist *iValueOut* = *iValue7*.
bDirection = TRUE bedeutet, dass die Ereignisse in absteigender Form von 8 nach 1 den Ausgabewert *iValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*bEvent1* = TRUE (niedrigstes eingetretenes Ereignis in Tabelle 2 [▶ 426] Spalte 1), dann ist *iValueOut* = *iValue1*.

Der Funktionsbaustein kann als Multiplexer verwendet werden, wenn *eCtrlMode* = *eHVACCtrlMode_Manual* ist. Der Wert von *iManualValue* bezieht sich auf eine von den VAR_IN_OUT-Variablen *iValue1-8*, deren Wert über *iValueOut* ausgegeben wird. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND *iManualValue* = 2, so ist *iValueOut* = *iValue2,* siehe Tabelle 3 [▶ 427].

bActive: *bActive* wird TRUE , wenn

1. *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto* und eine der Eingangsvariablen *bEvent1-8* = TRUE ist.

2. *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND*iManualValue* > 0 ist

bEdgeNewEvent: Ist für einen SPS-Zyklus TRUE, wenn

1. *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto* und sich das entscheidende Ereignis *(bEvent1-8)* zur Steuerung des Ausgangs *iValueOut* mittels *iValue1-8* geändert hat.

2. *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual* und mit jeder Änderung *von iManualValue*

iValueNumber: Mittels der Variable *iValueNumber* wird angezeigt, von welcher Variable der Wert an *iValueOut* ausgegeben wird. Ist *iValueOut* = *iValue7*, so ist *iValueNumber* = 7, siehe Tabelle 1 [▶ 426], Tabelle 2 [▶ 427] und Tabelle 3 [▶ 427]

VAR_IN_OUT

```
iValue1-8           : INT;
```

iValue1-8: Mittels der Variablen *iValue1* bis *iValue8* wird der Wert der Ausgangsvariable *iValueOut* bestimmt. *iValueOut* = *iValueX*

Wird der Funktionsbaustein als Priorisierung von Ereignissen eingesetzt, so wird jede der Variablen *iValue1-8* einem Ereignis zugeordnet. *iValue1* wird dem Ereignis *bEvent1* zugeordnet, *iValue2* dem Ereignis *bEvent2*, *iValue3* dem Ereignis *bEvent3*,..., *iValue8* dem Ereignis *bEvent8*

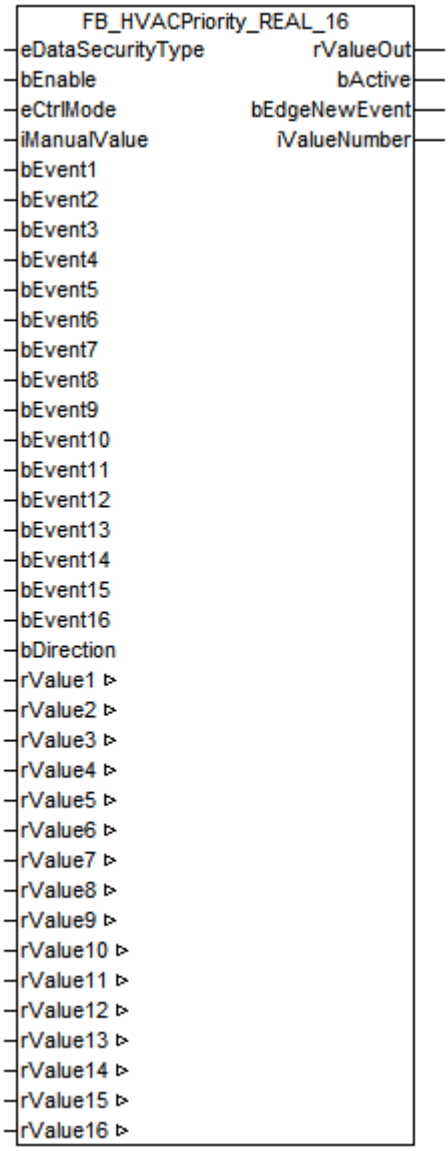
Wird der Funktionsbaustein als Muliplexer eingesetzt werden, so wird jede der Variablen *iValue1-8* dem Wert von *iManualValue* zugeordnet. Ist *iManualValue* = 1, dann ist *iValueOut* = *iValue1*. Ist *iManualValue* = 2, dann ist *iValueOut* = *iValue2*.... Ist *iManualValue* = 8, dann ist *iValueOut* = *iValue8*.

Die Variable wird persistent gespeichert.

Dokumente hierzu

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

3.6.24 FB_HVACPriority_REAL_16



Anwendung

Der Funktionsbaustein kann als Priorisierung von Ereignissen oder als Muliplexer eingesetzt werden. Diese Auswahl wird mittels des Enums *eCtrlMode* vorgenommen.

Der Funktionsbaustein kann als Priorisierung von Ereignissen verwendet werden, wenn *eCtrlMode* = *eHVACCtrlMode_Auto* ist. Der Ausgang *rValueOut wird* über die eintretenden Ereignisse der Eingänge *bEvent1-16*, *rValue1-16* und dem Wirksinn der Priorisierung *bDirection* gesteuert.
bDirection = FALSE bedeutet, dass die Ereignisse in aufsteigender Form von 1 nach 16 eine höhere Priorität besitzen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*bEvent15* = TRUE (höchstes eingetretenes Ereignis in Tabelle 1 [▶ 431] Spalte 1), dann hat *rValueOut* den Wert von *rValue15*.
bDirection = TRUE bedeutet, dass die Ereignisse in absteigender Form von 16 nach 1 eine höhere Priorität

besitzen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*bEvent1* = TRUE (niedrigstes eingetretenes Ereignis in Tabelle 2 [▶ 431] Spalte 1), dann hat *rValueOut* den Wert von *rValue1*.

Der Funktionsbaustein kann als Multiplexer verwendet werden, wenn *eCtrlMode* = *eHVACCtrlMode_Manual* ist. Der Wert von *iManualValue* bezieht sich auf eine der VAR_IN_OUT-Variablen *rValue1-16*, deren Wert über *rValueOut* ausgegeben wird. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND *iManualValue* = 2, so ist *rValueOut* = *rValue2,* siehe Tabelle 3 [▶ 432].

HINWEIS

Eine sich häufig ändernde Variable darf nicht an die VAR_IN_OUT-Variablen rValue1-16 angelegt werden, wenn eDataSecurityType = eHVACDataSecurityType_Persistentist. Dieses würde zu einem frühzeitigen Verschleiß des Speichermediums der Steuerung führen. Wenn sich die VAR_IN_OUT-Variablen rValue1-16 häufig ändern und nicht persistent abgelegt werden sollen, muss eDataSecurityType = eDataSecurityType_Idle sein.

Tabelle 1: Priorisierung von Ereignissen in aufsteigender Form von 1 nach 16

Tabelle 1

In der Tabelle 1 ist zu sehen, dass *bDirection* = FALSE ist. Das bedeutet, dass die Ereignisse in aufsteigender Form von 1 nach 16 den Ausgabewert *rValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*bEvent15* = TRUE (höchstes eingetretenes Ereignis in Spalte 1), dann ist *rValueOut* = *rValue15*.

	bEnable = TRUE eCtrlMode = eHVACCtrlMode_Auto bDircetion = FALSE					
bEvent1	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent2	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent3	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent4	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent5	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent6	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent7	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent8	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent9	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent10	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent11	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
bEvent12	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent13	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent14	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
bEvent15	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent16	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
rValueOut	rValue15	rValue8	rValue16	rValue14	0	rValue16
iValueNumber	15	8	16	14	0	16

Tabelle 2: Priorisierung von Ereignissen in absteigender Form von 16 nach 1

Tabelle 2

In der Tabelle 2 ist zu sehen, dass *bDirection* = TRUE ist. Das bedeutet, dass die Ereignisse in absteigender Form von 16 nach 1 den Ausgabewert *rValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*bEvent1* = TRUE (niedrigstes eingetretenes Ereignis in Tabelle 2 [▶ 431] Spalte 1), dann ist *rValueOut* = *rValue1*.

	bEnable = TRUE eCtrlMode = eHVACCtrlMode_Auto bDircetion = TRUE					
bEvent1	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent2	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent3	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent4	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent5	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent6	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent7	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent8	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent9	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent10	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent11	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
bEvent12	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent13	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent14	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
bEvent15	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
bEvent16	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
rValueOut	rValue1	rValue1	rValue9	rValue3	0	rValue1
iValueNumber	1	1	9	3	0	1

Tabelle 3: Multiplexer

Tabelle 3

Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND*iManualValue* = *2,* dann ist *rValueOut* = 2.

	bEnable = TRUE eCtrlMode = eHVACCtrlMode_Manual					
iManualValue	2	3	5	15	0	16
rValueOut	rValue2	rValue3	rValue5	rValue15	0	rValue16
iValueNumber	2	3	5	15	0	16

Anwendungsbeispiel

Download	Benötige Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
bEvent1 - bEvent8 : BOOL;
eCtrlMode         : E_HVACCtrlMode;
iManualValue      : INT;              0..16
bDirection        : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bEnable: Über ein TRUE wird der Funktionsbaustein freigegeben. Ist *bEnable* = FALSE, so wird *rValueOut* konstant auf 0 gesetzt.

eCtrlMode: Mittels des Enums wird die Betriebsart des Funktionsbausteins vorgegeben.
Ist *eCtrlMode* = *eHVACCtrlMode_Auto*, so stellt der Funktionsbaustein eine Priorisierung von Ereignissen dar. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto* so wird der Ausgang *rValueOut* über die eintretenden Ereignisse der Eingänge *bEvent1-16, rValue1-16* und des Wirksinns der Priorisierung *bDirection* gesteuert, siehe Tabelle 1 [▶ 431] und Tabelle 2 [▶ 431]
Ist *eCtrlMode* = *eHVACCtrlMode_Manual*, so stellt der Funktionsbaustein einen Multiplexer dar. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual, so* wird der Wert des Ausgangs *rValueOut* über *iManualValue* gesteuert. Ist *iManualProfile* = 5, so ist *rValueOut* = *rValue5*, siehe Tabelle 3 [▶ 432]

iManualProfile: Ist die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual* (Multiplexer) AND*bEnable* = TRUEAND*bError* = FALSE*, so* wird der Wert des Ausgangs *rValueOut* über *iManualValue* gesteuert. Ist *iManualProfile* = 5, so ist *rValueOut* = *rValue5*, siehe Tabelle 3 [▶ 432]

bEvent1-16: Der Ausgang *rValueOut* wird über die eintretenden Ereignisse der Eingänge *bEvent1-16*, *rValue1-16* und dem Wirksinn der Priorisierung *bDirection* gesteuert.
bDirection = FALSE bedeutet, dass die Ereignisse in aufsteigender Form von 1 nach 16 den Ausgabewert *rValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*bEvent15* = TRUE (höchstes eingetretenes Ereignis in Tabelle 1 [▶ 431] Spalte 1), dann ist *rValueOut* = *rValue15*.
bDirection = TRUE bedeutet, dass die Ereignisse in absteigender Form von 16 nach 1 den Ausgabewert *rValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*bEvent1* = TRUE (niedrigstes eingetretenes Ereignis in Tabelle 2 [▶ 431] Spalte 1), dann ist *rValueOut* = *rValue1*.
Die Variablen *bEvent1-16* und *bDirection* werden nur dann berücksichtigt, wenn *eCtrlMode* = *eHVACCtrlMode_Auto* ist.

bDirection: Der Funktionsbaustein kann als Priorisierung von Ereignissen verwendet werden, wenn *eCtrlMode* = *eHVACCtrlMode_Auto* ist. Der Ausgang *rValueOut wird* über die eintretenden Ereignisse der Eingänge *bEvent1-16*, *rValue1-16* und dem Wirksinn der Priorisierung *bDirection* gesteuert.
bDirection = FALSE bedeutet, dass die Ereignisse in aufsteigender Form von 1 nach 16 den Ausgabewert *rValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*bEvent15* = TRUE (höchstes eingetretenes Ereignis in Tabelle 1 [▶ 431] Spalte 1), dann ist *rValueOut* = *rValue15*.
bDirection = TRUE bedeutet, dass die Ereignisse in absteigender Form von 16 nach 1 den Ausgabewert *rValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*bEvent1* = TRUE (niedrigstes eingetretenes Ereignis in Tabelle 2 [▶ 431] Spalte 1), dann ist *rValueOut* = *rValue1*.
Die Variablen *bEvent1-16* und *bDirection* werden nur dann berücksichtigt, wenn *eCtrlMode* = *eHVACCtrlMode_Auto* ist.

VAR_OUTPUT

```
rValueOut           : REAL;
bActive             : BOOL;
bEdgeNewEvent       : BOOL;
iValueOut           : INT;
```

rValueOut: Der Funktionsbaustein kann als Priorisierung von Ereignissen oder als Muliplexer eingesetzt werden. Diese Auswahl wird mittels des Enums *eCtrlMode* vorgenommen.

Der Funktionsbaustein kann als Priorisierung von Ereignissen verwendet werden, wenn *eCtrlMode* = *eHVACCtrlMode_Auto* ist. Der Ausgang *rValueOut wird* über die eintretenden Ereignisse der Eingänge *bEvent1-16*, *rValue1-16* und dem Wirksinn der Priorisierung *bDirection* gesteuert.
bDirection = FALSE bedeutet, dass die Ereignisse in aufsteigender Form von 1 nach 16 den Ausgabewert *rValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*bEvent15* = TRUE (höchstes eingetretenes Ereignis in Tabelle 1 [▶ 431] Spalte 1), dann ist *rValueOut* = *rValue15*.
bDirection = TRUE bedeutet, dass die Ereignisse in absteigender Form von 16 nach 1 den Ausgabewert *rValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*bEvent1* = TRUE (niedrigstes eingetretenes Ereignis in Tabelle 2 [▶ 431] Spalte 1), dann ist *rValueOut* = *rValue1*.

Der Funktionsbaustein kann als Multiplexer verwendet werden, wenn *eCtrlMode* = *eHVACCtrlMode_Manual* ist. Der Wert von *iManualValue* bezieht sich auf eine von den VAR_IN_OUT-Variablen *rValue1-16*, deren Wert über *rValueOut* ausgegeben wird. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND *iManualValue* = 2, so ist *rValueOut* = *rValue2,* siehe Tabelle 3 [▶ 432].

bActive: *bActive* wird TRUE , wenn

1. *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto* und eine der Eingangsvariablen *bEvent1-16* = TRUE ist.

2. *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND*iManualValue* > 0 ist

bEdgeNewEvent: Ist für einen SPS-Zyklus TRUE, wenn

1. *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto* und sich das entscheidende Ereignis *(bEvent1-16)* zur Steuerung des Ausgangs *rValueOut* mittels *rValue1-16* geändert hat.

2. *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual* und mit jeder Änderung *von iManualValue*

iValueNumber: Mittels der Variable *iValueNumber* wird angezeigt, von welcher Variable der Wert an *rValueOut* ausgegeben wird. Ist *rValueOut* = *rValue7*, so ist *iValueNumber* = 7, siehe Tabelle 1 [▶ 431], Tabelle 2 [▶ 431] und Tabelle 3 [▶ 432]

VAR_IN_OUT

```
rValue1-16        : REAL;
```

rValue1-16: Mittels der Variablen *rValue1* bis *rValue16* wird der Wert der Ausgangsvariable *rValueOut* bestimmt. *rValueOut* = *rValueX*

Wird der Funktionsbaustein als Priorisierung von Ereignissen eingesetzt, so wird jede der Variablen *rValue1-16* einem Ereignis zugeordnet. *rValue1* wird dem Ereignis *bEvent1* zugeordnet, *rValue2* dem Ereignis *bEvent2*, *rValue3* dem Ereignis *bEvent3*,..., *rValue16* dem Ereignis *bEvent16*

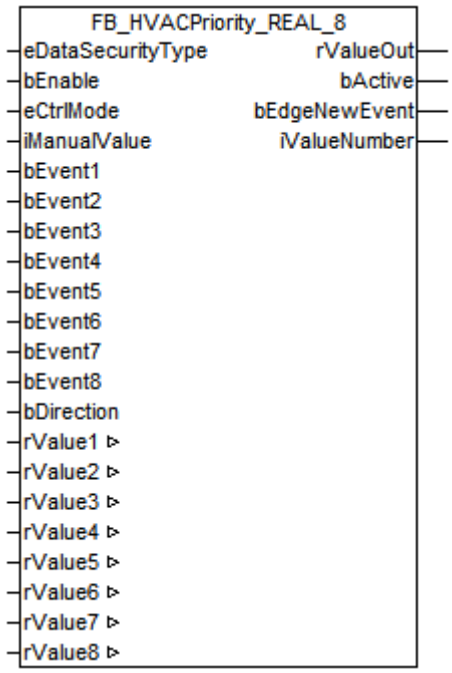
Wird der Funktionsbaustein als Muliplexer eingesetzt , so wird jede der Variablen *rValue1-16* dem Wert von *iManualValue* zugeordnet. Ist *iManualValue* = 1, dann ist *rValueOut* = *rValue1*. Ist *iManualValue* = 2, dann ist *rValueOut* = *rValue2*.... Ist *iManualValue* = 16, dann ist *rValueOut* = *rValue16*.

Die Variable wird persistent gespeichert.

Dokumente hierzu

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

3.6.25 FB_HVACPriority_REAL_8



Anwendung

Der Funktionsbaustein kann als Priorisierung von Ereignissen oder als Muliplexer eingesetzt werden. Diese Auswahl wird mittels des Enums *eCtrlMode* vorgenommen.

Der Funktionsbaustein kann als Priorisierung von Ereignissen verwendet werden, wenn *eCtrlMode* = *eHVACCtrlMode_Auto* ist. Der Ausgang *rValueOut wird* über die eintretenden Ereignisse der Eingänge *bEvent1-8*, *rValue1-8* und dem Wirksinn der Priorisierung *bDirection* gesteuert.
bDirection = FALSE bedeutet, dass die Ereignisse in aufsteigender Form von 1 nach 8 eine höhere Priorität besitzen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*bEvent7* = TRUE (höchstes eingetretenes Ereignis in Tabelle 1 [▶ 435] Spalte 1), dann hat *rValueOut* den Wert von *rValue7*.
bDirection = TRUE bedeutet, dass die Ereignisse in absteigender Form von 8 nach 1 eine höhere Priorität besitzen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*bEvent1* = TRUE (niedrigstes eingetretenes Ereignis in Tabelle 2 [▶ 435] Spalte 1), dann hat *rValueOut* den Wert von *rValue1*.

Der Funktionsbaustein kann als Multiplexer verwendet werden, wenn *eCtrlMode* = *eHVACCtrlMode_Manual* ist. Der Wert von *iManualValue* bezieht sich auf eine der VAR_IN_OUT-Variablen *rValue1-8*, deren Wert über *rValueOut* ausgegeben wird. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND *iManualValue* = 2, so ist *rValueOut* = *rValue2,* siehe Tabelle 3 [▶ 436].

HINWEIS

Eine sich häufig ändernde Variable darf nicht an die VAR_IN_OUT-Variablen rValue1-8 angelegt werden, wenn eDataSecurityType = eHVACDataSecurityType_Persistentist. Dieses würde zu einem frühzeitigen Verschleiß des Speichermediums der Steuerung führen. Wenn sich die VAR_IN_OUT-Variablen rValue1-8 häufig ändern und nicht persistent abgelegt werden sollen, muss eDataSecurityType = eDataSecurityType_Idle sein.

Tabelle 1: Priorisierung von Ereignissen in aufsteigender Form von 1 nach 8

Tabelle 1

In der Tabelle 1 ist zu sehen, dass *bDirection* = FALSE ist. Das bedeutet, dass die Ereignisse in aufsteigender Form von 1 nach 8 den Ausgabewert *rValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*bEvent7* = TRUE (höchstes eingetretenes Ereignis in Spalte 1), dann ist *rValueOut* = *rValue7*.

	bEnable = TRUE eCtrlMode = eHVACCtrlMode_Auto bDircetion = FALSE					
bEvent1	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE
bEvent2	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE
bEvent3	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE
bEvent4	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent5	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent6	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
bEvent7	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
bEvent8	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
rValueOut	rValue7	rValue8	0	rValue6	rValue3	rValue5
iValueNumber	7	8	0	6	3	5

Tabelle 2: Priorisierung von Ereignissen in absteigender Form von 8 nach 1

Tabelle 2

In der Tabelle 2 ist zu sehen, dass *bDirection* = TRUE ist. Das bedeutet, dass die Ereignisse in absteigender Form von 8 nach 1 den Ausgabewert *rValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*bEvent1* = TRUE (niedrigstes eingetretenes Ereignis in Tabelle 2 [▶ 436] Spalte 1), dann ist *rValueOut* = *rValue1*.

	bEnable = TRUE eCtrlMode = eHVACCtrlMode_Auto bDircetion = TRUE					
bEvent1	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE
bEvent2	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE
bEvent3	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE
bEvent4	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
bEvent5	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
bEvent6	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
bEvent7	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
bEvent8	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
rValueOut	rValue1	rValue1	0	rValue3	rValue1	rValue1
iValueNumber	1	1	0	3	1	1

Tabelle 3: Multiplexer

Tabelle 3

Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND*iManualValue* = *2,* dann ist *rValueOut* = 2.

	bEnable = TRUE eCtrlMode = eHVACCtrlMode_Manual					
iManualValue	2	3	5	1	0	8
rValueOut	rValue2	rValue3	rValue5	rValue1	0	rValue8
iValueNumber	2	3	5	1	0	8

Anwendungsbeispiel

Download	Benötige Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

```
eDataSecurityType : E_HVACDataSecurityType;
bEnable           : BOOL;
bEvent1 - bEvent8 : BOOL;
eCtrlMode         : E_HVACCtrlMode;
iManualValue      : INT;             0..8
bDirection        : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bEnable: Über ein TRUE wird der Funktionsbaustein freigegeben. Ist *bEnable* = FALSE, so wird *rValueOut* konstant auf 0 gesetzt.

eCtrlMode: Mittels des Enums wird die Betriebsart des Funktionsbausteins vorgegeben.
Ist *eCtrlMode* = *eHVACCtrlMode_Auto*, so stellt der Funktionsbaustein eine Priorisierung von Ereignissen dar. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto* so wird der Ausgang *rValueOut* über die eintretenden Ereignisse der Eingänge *bEvent1-8, rValue1-8* und des Wirksinns der Priorisierung *bDirection* gesteuert, siehe Tabelle 1 [▶ 435] und Tabelle 2 [▶ 436]
Ist *eCtrlMode* = *eHVACCtrlMode_Manual*, so stellt der Funktionsbaustein einen Multiplexer dar. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual, so* wird der Wert des Ausgangs *rValueOut* über *iManualValue* gesteuert. Ist *iManualProfile* = 5, so ist *rValueOut* = *rValue5*, siehe Tabelle 3 [▶ 436]

iManualProfile: Ist die Betriebsart *eCtrlMode* = *eHVACCtrlMode_Manual* (Multiplexer) AND*bEnable* = TRUEAND*bError* = FALSE*, so* wird der Wert des Ausgangs *rValueOut* über *iManualValue* gesteuert. Ist *iManualProfile* = 5, so ist *rValueOut* = *rValue5*, siehe Tabelle 3 [▶ 436]

bEvent1-8: Der Ausgang *rValueOut* wird über die eintretenden Ereignisse der Eingänge *bEvent1-8*, *rValue1-8* und dem Wirksinn der Priorisierung *bDirection* gesteuert.
bDirection = FALSE bedeutet, dass die Ereignisse in aufsteigender Form von 1 nach 8 den Ausgabewert *rValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*bEvent7* = TRUE (höchstes eingetretenes Ereignis in Tabelle 1 [▶ 435] Spalte 1), dann ist *rValueOut* = *rValue7*.
bDirection = TRUE bedeutet, dass die Ereignisse in absteigender Form von 8 nach 1 den Ausgabewert *rValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*bEvent1* = TRUE (niedrigstes eingetretenes Ereignis in Tabelle 2 [▶ 435] Spalte 1), dann ist *rValueOut* = *rValue1*.
Die Variablen *bEvent1-8* und *bDirection* werden nur dann berücksichtigt, wenn *eCtrlMode* = *eHVACCtrlMode_Auto* ist.

bDirection: Der Funktionsbaustein kann als Priorisierung von Ereignissen verwendet werden, wenn *eCtrlMode* = *eHVACCtrlMode_Auto* ist. Der Ausgang *rValueOut wird* über die eintretenden Ereignisse der Eingänge *bEvent1-8*, *rValue1-8* und dem Wirksinn der Priorisierung *bDirection* gesteuert.
bDirection = FALSE bedeutet, dass die Ereignisse in aufsteigender Form von 1 nach 8 den Ausgabewert *rValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*bEvent7* = TRUE (höchstes eingetretenes Ereignis in Tabelle 1 [▶ 435] Spalte 1), dann ist

rValueOut = *rValue7*.
bDirection = TRUE bedeutet, dass die Ereignisse in absteigender Form von 8 nach 1 den Ausgabewert *rValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*bEvent1* = TRUE (niedrigstes eingetretenes Ereignis in Tabelle 2 [▶ 435] Spalte 1), dann ist *rValueOut* = *rValue1*.
Die Variablen *bEvent1-8* und *bDirection* werden nur dann berücksichtigt, wenn *eCtrlMode* = *eHVACCtrlMode_Auto* ist.

VAR_OUTPUT

```
rValueOut           : REAL;
bActive             : BOOL;
bEdgeNewEvent       : BOOL;
iValueNumber        : INT;
```

rValueOut: Der Funktionsbaustein kann als Priorisierung von Ereignissen oder als Muliplexer eingesetzt werden. Diese Auswahl wird mittels des Enums *eCtrlMode* vorgenommen.

Der Funktionsbaustein kann als Priorisierung von Ereignissen verwendet werden, wenn *eCtrlMode* = *eHVACCtrlMode_Auto* ist. Der Ausgang *rValueOut wird* über die eintretenden Ereignisse der Eingänge *bEvent1-8*, *rValue1-8* und dem Wirksinn der Priorisierung *bDirection* gesteuert.
bDirection = FALSE bedeutet, dass die Ereignisse in aufsteigender Form von 1 nach 8 den Ausgabewert *rValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = FALSEAND*bEvent7* = TRUE (höchstes eingetretenes Ereignis in Tabelle 1 [▶ 435] Spalte 1), dann ist *rValueOut* = *rValue7*.
bDirection = TRUE bedeutet, dass die Ereignisse in absteigender Form von 8 nach 1 den Ausgabewert *rValueOut* bestimmen. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto*AND*bDirection* = TRUEAND*bEvent1* = TRUE (niedrigstes eingetretenes Ereignis in Tabelle 2 [▶ 435] Spalte 1), dann ist *rValueOut* = *rValue1*.

Der Funktionsbaustein kann als Multiplexer verwendet werden, wenn *eCtrlMode* = *eHVACCtrlMode_Manual* ist. Der Wert von *iManualValue* bezieht sich auf eine von den VAR_IN_OUT-Variablen *rValue1-8*, deren Wert über *rValueOut* ausgegeben wird. Ist *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND *iManualValue* = 2, so ist *rValueOut* = *rValue2,* siehe Tabelle 3 [▶ 436].

bActive: *bActive* wird TRUE , wenn

1. *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto* und eine der Eingangsvariablen *bEvent1-8* = TRUE ist.

2. *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual*AND*iManualValue* > 0 ist

bEdgeNewEvent: Ist für einen SPS-Zyklus TRUE, wenn

1. *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Auto* und sich das entscheidende Ereignis *(bEvent1-8)* zur Steuerung des Ausgangs *rValueOut* mittels *rValue1-8* geändert hat.

2. *bEnable* = TRUEAND*eCtrlMode* = *eHVACCtrlMode_Manual* und mit jeder Änderung *von iManualValue*

iValueNumber: Mittels der Variable *iValueNumber* wird angezeigt, von welcher Variable der Wert an *rValueOut* ausgegeben wird. Ist *rValueOut* = *rValue7*, so ist *iValueNumber* = 7, siehe Tabelle 1 [▶ 435], Tabelle 2 [▶ 436] und Tabelle 3 [▶ 436]

VAR_IN_OUT

```
rValue1-8         : REAL;
```

rValue1-8: Mittels der Variablen *rValue1* bis *rValue8* wird der Wert der Ausgangsvariable *rValueOut* bestimmt. *rValueOut* = *rValueX*

Wird der Funktionsbaustein als Priorisierung von Ereignissen eingesetzt, so wird jede der Variablen *rValue1-8* einem Ereignis zugeordnet. *rValue1* wird dem Ereignis *bEvent1* zugeordnet, *rValue2* dem Ereignis *bEvent2*, *rValue3* dem Ereignis *bEvent3*,..., *rValue8* dem Ereignis *bEvent8*

Wird der Funktionsbaustein als Muliplexer eingesetzt werden, so wird jede der Variablen *rValue1-8* dem Wert von *iManualValue* zugeordnet. Ist *iManualValue* = 1, dann ist *rValueOut* = *rValue1*. Ist *iManualValue* = 2, dann ist *rValueOut* = *rValue2*.... Ist *iManualValue* = 8, dann ist *rValueOut* = *rValue8*.

Die Variable wird persistent gespeichert.

Dokumente hierzu

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

3.6.26 FB_HVACOptimizedOn

Funktionsbaustein zum optimierten Einschalten von Heizkesseln und Klimaanlagen in Verbindung mit Schaltzeitbausteinen.

Die Ausführungen in dieser Dokumentation beziehen sich auf das Heizverhalten. Der Einsatz des Bausteines in Verbindung mit Kühlgeräten erfolgt analog.

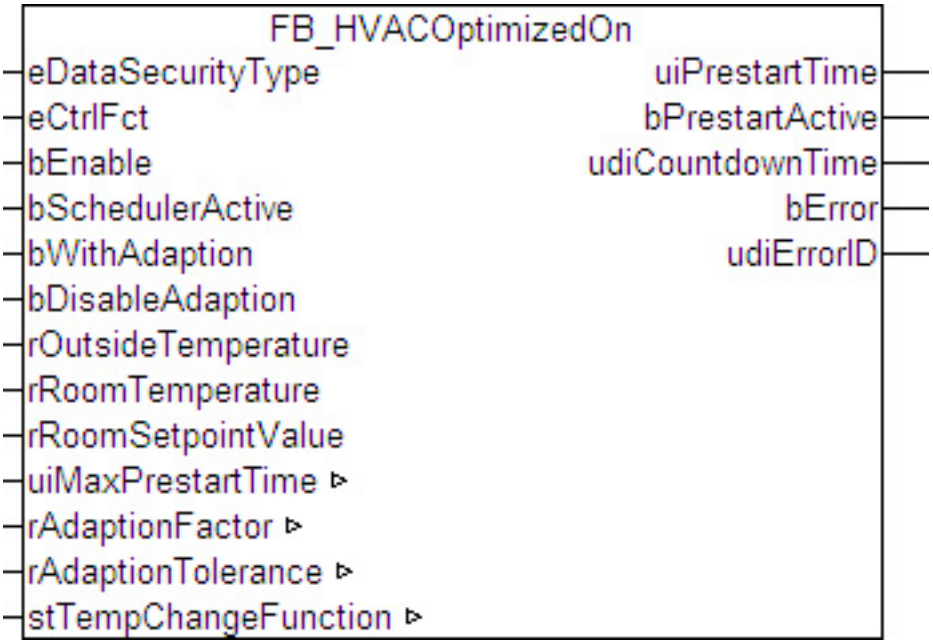
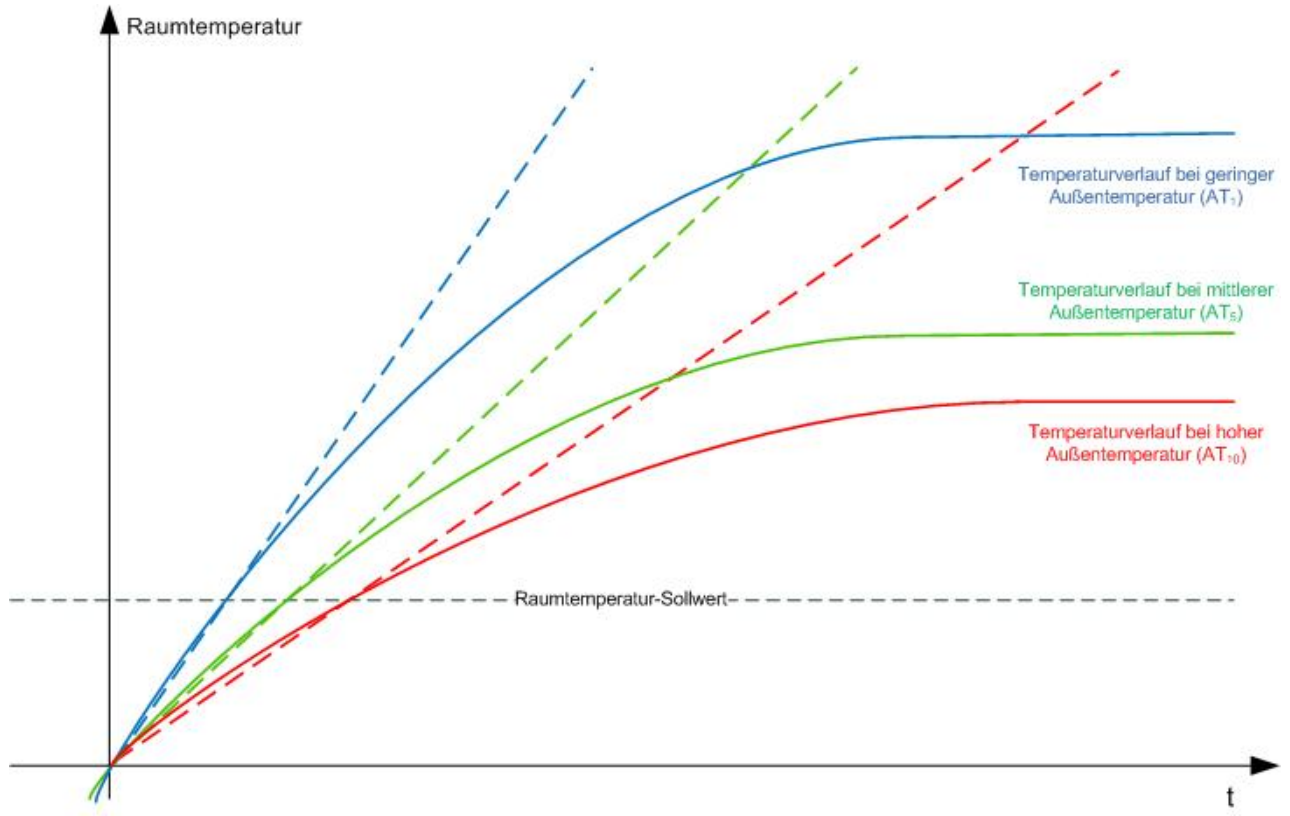


Abb. 17: FB_HVACOptimizedOn

Berechnung der Vorstartzeit

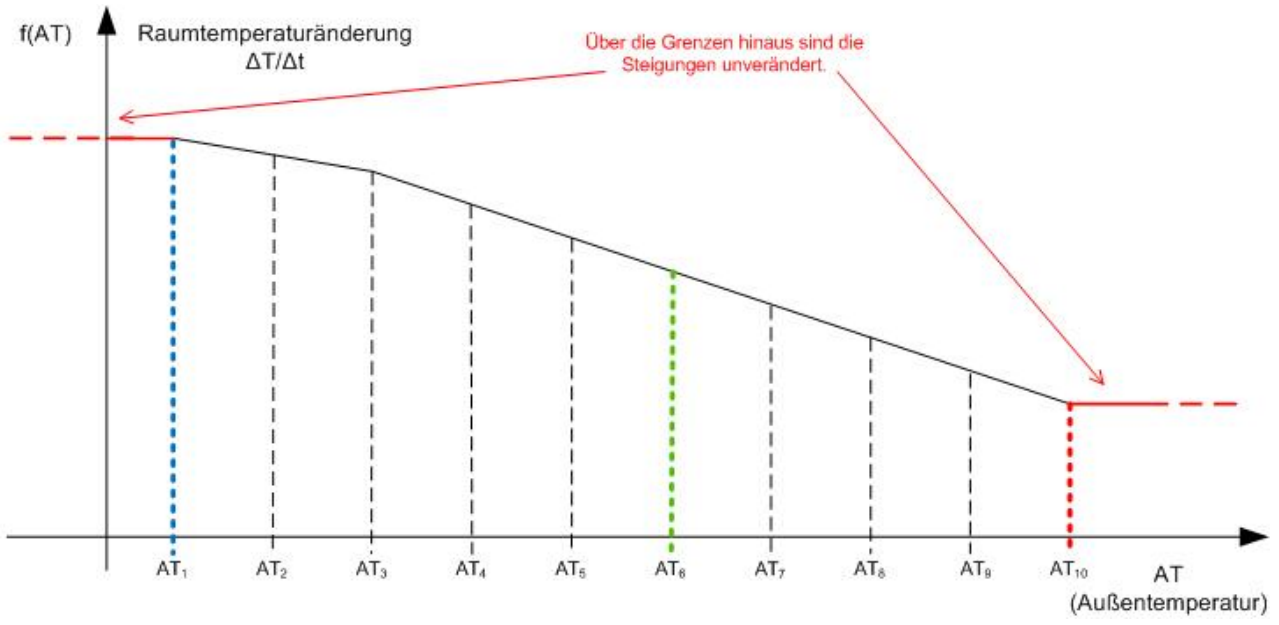
Firmengebäude und öffentliche Einrichtungen sind über Nacht und an den Wochenenden meist unbelegt, so dass die Heizkessel dort oft nur in einem Standby-Modus arbeiten. Ziel eines optimierten Einschaltens ist es, den Heizkessel so früh zu starten, dass das Gebäude zu einem bestimmten Zeitpunkt genügend aufgeheizt ist. Umgekehrt gilt dies in den Sommermonaten auch für eine Kühlung des Gebäudes. Diese Vorstartzeit ist freilich nicht konstant, sondern immer abhängig von verschiedenen Faktoren, wie Außentemperatur, Raumtemperaturdifferenz (Ist- zu Sollwert) und baulichen Beschaffenheiten wie beispielsweise die Gebäudemasse und die Wärmedämmung. Eine universelle Formel zu entwickeln, welche für alle Gebäude gültig ist, wäre eine sehr komplexe Aufgabe, die alleine schon an der Eingabe unzähliger Parameter scheitern würde. Einfacher und letztendlich absolut ausreichend ist ein selbst lernendes (adaptives) Näherungsverfahren.

Der Vorlauftemperatur des Heizkessels richtet sich nach der Außentemperatur. Damit ergeben sich für unterschiedliche Außentemperaturen verschiedene Aufheizverläufe, die im Wesentlichen einer Exponentialfunktion entsprechen:



Die Vorlauftemperatur ist dabei immer deutlich höher als die zu erreichende Raumtemperatur. Zur Ermittlung der Vorstartzeit wird angenommen, dass der Bereich der Funktionen zwischen Raumtemperatur-Startwert und Raumtemperatur-Sollwert linear ist. Damit ergibt sich für jede Außentemperatur eine charakteristische Temperaturänderung ΔT/Δt, welche hier durch die gestrichelte Linie dargestellt wird.

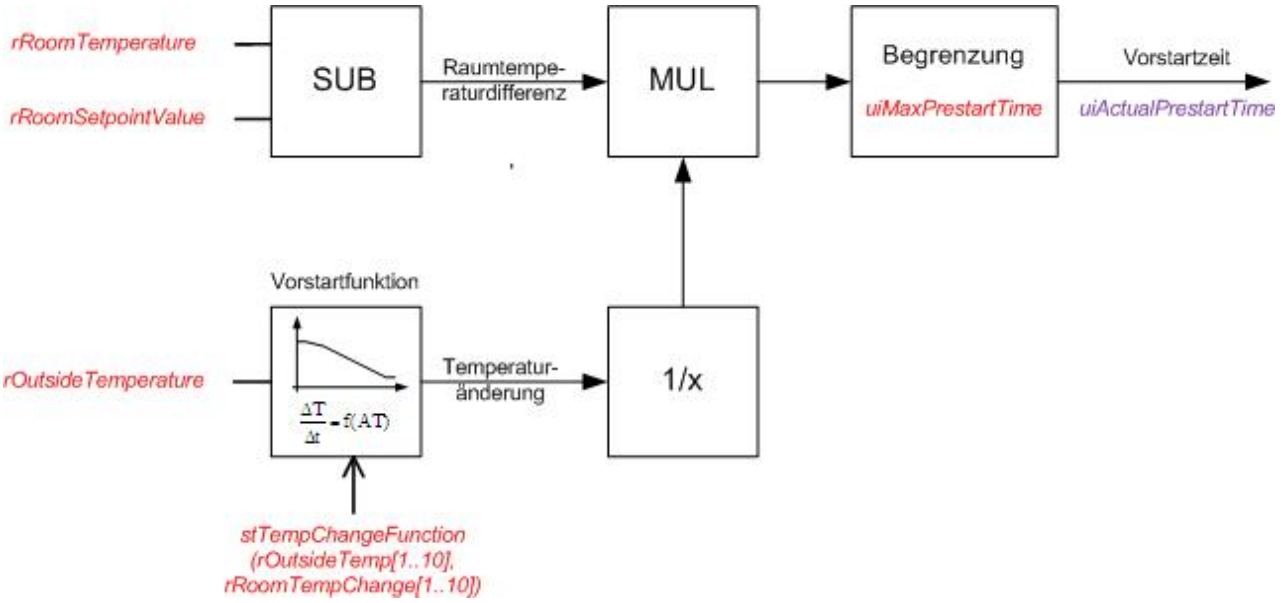
Dem Funktionsbaustein FB_HVACOptimizedOn liegt mit der Strukturvariable stTempChangeFunction [▶ 538] eine Tabelle zugrunde, in der für 10 diskrete Außentemperaturen die jeweilig zu erwartende Raumtemperaturänderung zugeordnet ist. Bei Inbetriebnahme bzw. Erststart der Anlage muß die Vorstartfunktion grob vordefiniert sein, zum einen, um den Außentemeraturbereich festzulegen, zum anderen um den Adaptionsvorgang zu beschleunigen. Diese Eingabe geschieht mit dem Baustein FB_HVACTempChangeFunctionEntry [▶ 459]. Mit diesen Werten lässt sich näherungsweise die Vorstartzeit ermitteln. Diese Temperaturänderungs-Funktion f(AT) nimmt typischerweise folgenden Verlauf an:



Funktionswerte innerhalb dieser 10 Punkte werden über Geradengleichungen definiert, Werte außerhalb entsprechen dem Funktionswert f(AT_1) bzw. f(AT_{10}):

- $AT < AT_1$: $f(AT) = f(AT_1)$
- $AT > AT_{10}$: $f(AT) = f(AT_{10})$

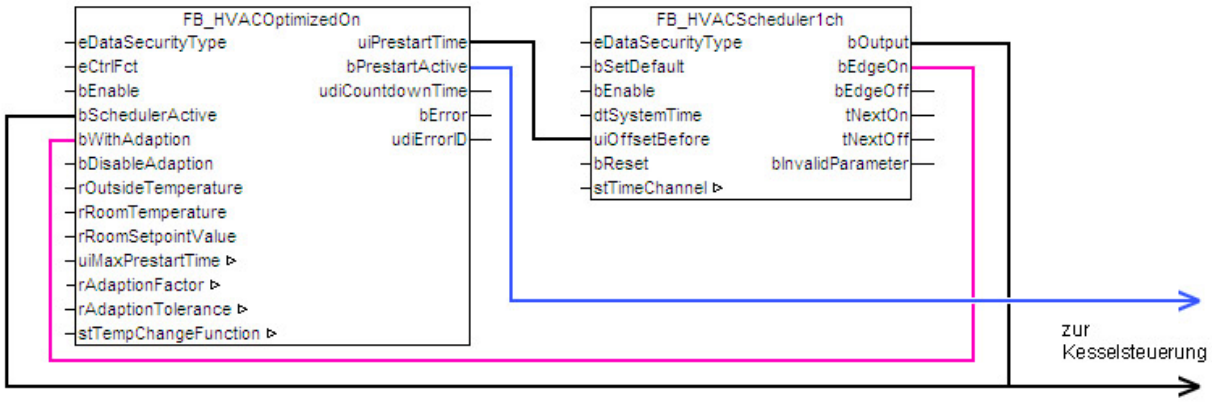
Die Ermittlung der Vorstartzeit geschieht dann nach folgendem Ablauf:



Rot dargestellt sind hier die Ein- und Ausgabevariablen des Bausteines. Violett dargestellt ist hier der Interne Merker *uiActualPrestartTime* der zu jedem Zeitpunkt die aktuelle Vorstartzeit enthält. Der Ausgang *uiPrestartTime* entspricht diesem Merker, wird jedoch während der Vorstartphase "eingefroren". Dieses wird im Folgenden weiter erläutert.

Bausteinverknüpfung und Vorstartphase

Aus diesen Vorgaben wird der Baustein dann kontinuierlich eine Vorstartzeit berechnen, die er dem Schaltuhrbaustein vorgibt. Soll ein Heizkessel dann beispielsweise Montagmorgens um 6 Uhr das Gebäude aufgeheizt haben und der Vorstartbaustein ermittelt um 5 Uhr morgens 60 Minuten Vorstartzeit, so wird die Schaltzeituhr den Kessel umgehend in den entsprechenden Heizmodus versetzen. Das Aufheizverhalten wird dann in einem Referenzraum über die Vorstartphase hinweg beobachtet und die Vorstartkurve entsprechend korrigiert.



Der Vorstart-Baustein gibt für die Schaltuhr eine Vorstartzeit an. Schaltet die Schaltuhr den Ausgang *bOutput*, so wird dem Vorstart-Baustein dies über seinen Eingang *bSchedulerActive* angezeigt. Intern startet der Vorstart-Baustein dann einen Countdown mit der zuvor ausgegebenen Vorstartzeit *uiPrestartTime* [min].

Der Countdown läuft entweder bis zum Ende durch oder wird mit Erreichen der Raum-Solltemperatur vorzeitig abgeschlossen und stellt die Vorstartphase dar. Während des Countdowns wird der Ausgang *bPrestartActive* gesetzt , der dann beispielsweise für einen Schnellstart des Kessels genutzt werden kann. Der Ausgang *uiPrestartTime* folgt kontinuierlich der oben aufgeführten Berechnung - während eines gestarteten Countdowns jedoch wir er konstant gehalten, damit eine Schwankung in der Außentemperatur nicht eine geringere Vorstartzeit angibt und den Kessel plötzlich abschaltet.

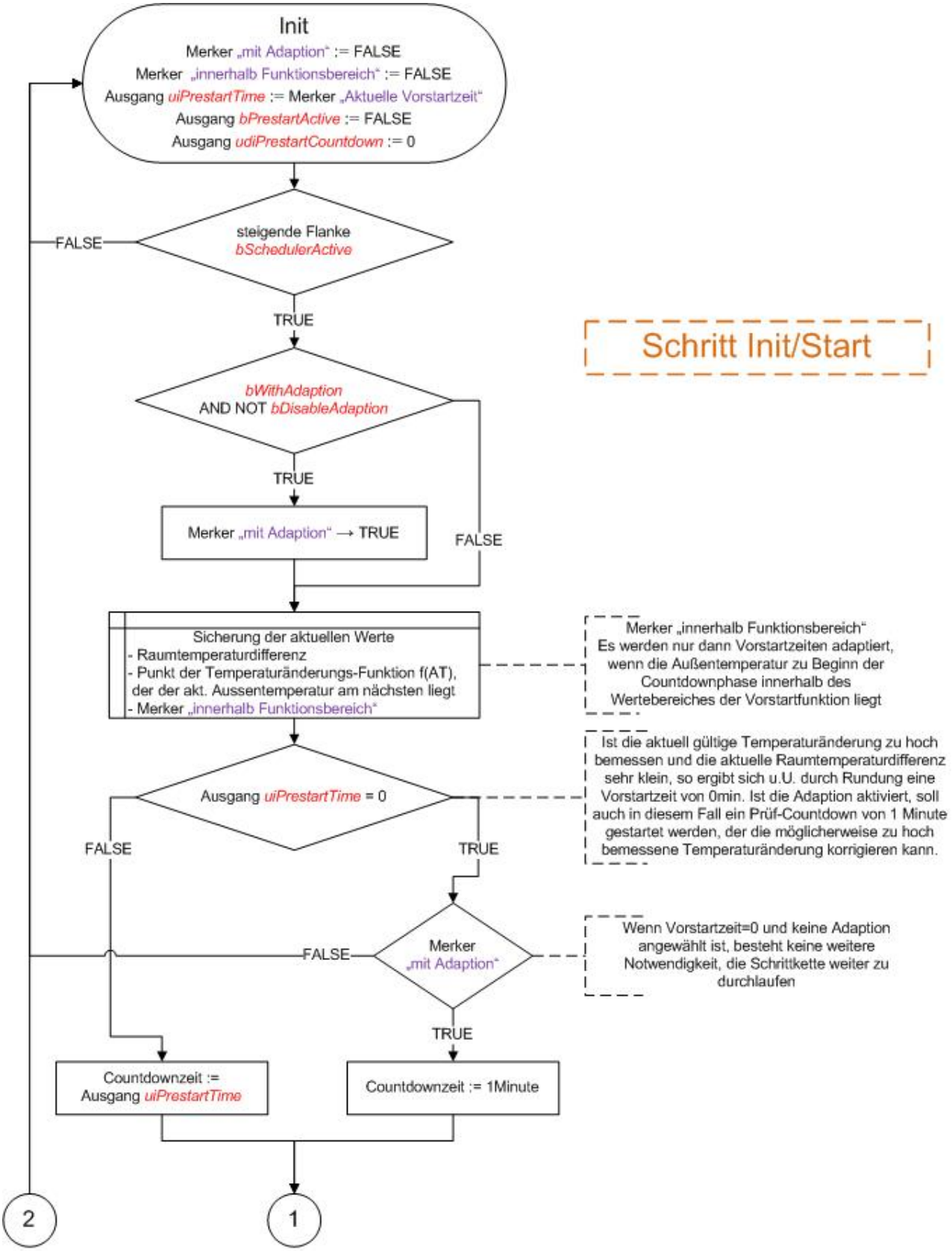
Teilt der Schaltuhrbaustein dem Optimierungsbaustein bei einem Vorstart noch den Adaptionsauftrag als Flanke mit (rote Linie), so wird dieser nach der Vorstartphase entscheiden, ob die zuvor ermittelte Vorstartzeit innerhalb einer Toleranz genau oder zu klein bzw. zu groß bemessen war und die Temperaturänderungs-Funktion entsprechend korrigieren. Dabei wird die Vorstartzeit des Punktes, dessen Außentemperatur der tatsächlichen zu Beginn des Countdowns am nächsten lag, nach oben bzw. nach unten hin korrigiert. Dieser Vorgang wird "Adaption [▶ 445]" genannt.

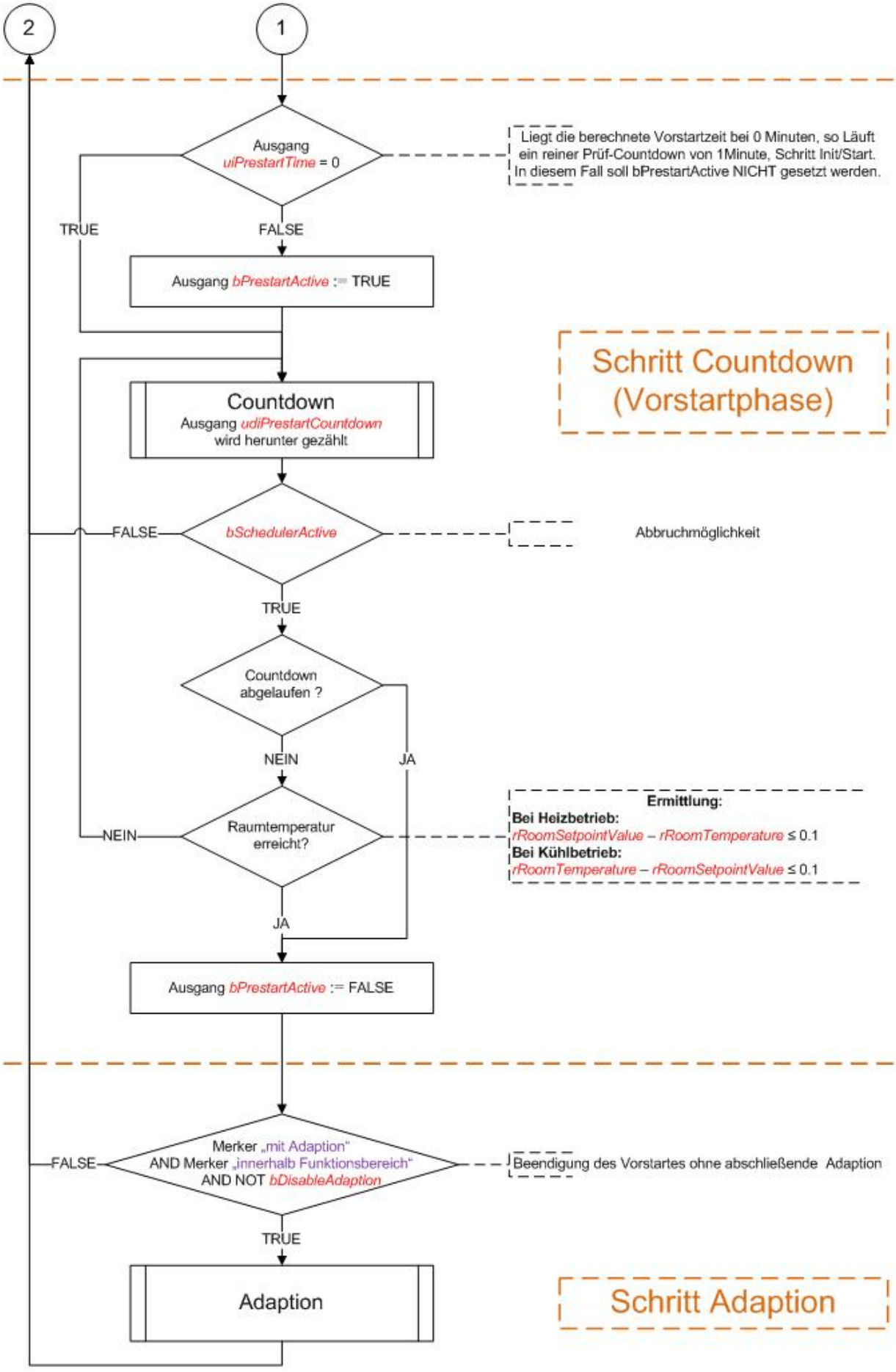
Es werden nur dann Temperaturänderungen korrigiert, wenn die Außentemperatur zu Beginn des Vorstarts innerhalb der Stützstellen, d.h. des Funktionsbereiches gelegen hat, siehe "Adaption [▶ 445]".

Bei gewünschter Adaption muss der Eingang *bWithAdaption* zeitgleich mit dem Eingang *bSchedulerActive* gesetzt sein. Für *bWithApation* reicht dabei ein reiner Triggerimpuls. Ein Abfallen des Einganges *bSchedulerActive* unterbricht einen zuvor gestarteten Countdown sofort. Die Adaption kann durch den Eingang *bDisableAdaption* unterdrückt werden. Diese Option ist dann zu wählen, wenn nach einer gewissen Anzahl von Adaptionsvorgängen die Temperaturänderungs-Funktion nicht mehr verändert werden soll.

Der folgende Programmablaufplan soll das Verhalten besser veranschaulichen, rot dargestellt sind hier die Ein- und Ausgabevariablen des Bausteines:

Der interne Merker "Aktuelle Vorstartzeit" ist das ständig aktualisierte Ergebnis der Berechnung nach dem oben aufgeführten Diagramm.


Init
Merker „mit Adaption" := FALSE
Merker „innerhalb Funktionsbereich" := FALSE
Ausgang uiPrestartTime := Merker „Aktuelle Vorstartzeit"
Ausgang bPrestartActive := FALSE
Ausgang udiPrestartCountdown := 0
steigende Flanke
bSchedulerActive
FALSE
TRUE
Schritt Init/Start
bWithAdaption
AND NOT bDisableAdaption
TRUE
FALSE
Merker „mit Adaption" → TRUE
Sicherung der aktuellen Werte
- Raumtemperaturdifferenz
- Punkt der Temperaturänderungs-Funktion f(AT), der der akt. Aussentemperatur am nächsten liegt
- Merker „innerhalb Funktionsbereich"
Merker „innerhalb Funktionsbereich"
Es werden nur dann Vorstartzeiten adaptiert, wenn die Außentemperatur zu Beginn der Countdownphase innerhalb des Wertebereiches der Vorstartfunktion liegt
Ausgang uiPrestartTime = 0
Ist die aktuell gültige Temperaturänderung zu hoch bemessen und die aktuelle Raumtemperaturdifferenz sehr klein, so ergibt sich u.U. durch Rundung eine Vorstartzeit von 0min. Ist die Adaption aktiviert, soll auch in diesem Fall ein Prüf-Countdown von 1 Minute gestartet werden, der die möglicherweise zu hoch bemessene Temperaturänderung korrigieren kann.
FALSE
TRUE
Merker „mit Adaption"
FALSE
Wenn Vorstartzeit=0 und keine Adaption angewählt ist, besteht keine weitere Notwendigkeit, die Schrittkette weiter zu durchlaufen
TRUE
Countdownzeit := Ausgang uiPrestartTime
Countdownzeit := 1Minute
2
1


2
1
Ausgang uiPrestartTime = 0
Liegt die berechnete Vorstartzeit bei 0 Minuten, so Läuft ein reiner Prüf-Countdown von 1Minute, Schritt Init/Start. In diesem Fall soll bPrestartActive NICHT gesetzt werden.
TRUE
FALSE
Ausgang bPrestartActive := TRUE
Schritt Countdown (Vorstartphase)
Countdown
Ausgang udiPrestartCountdown wird herunter gezählt
FALSE
bSchedulerActive
Abbruchmöglichkeit
TRUE
Countdown abgelaufen ?
NEIN
JA
NEIN
Raumtemperatur erreicht?
Ermittlung:
Bei Heizbetrieb:
rRoomSetpointValue – rRoomTemperature ≤ 0.1
Bei Kühlbetrieb:
rRoomTemperature – rRoomSetpointValue ≤ 0.1
JA
Ausgang bPrestartActive := FALSE
FALSE
Merker „mit Adaption“ AND Merker „innerhalb Funktionsbereich“ AND NOT bDisableAdaption
Beendigung des Vorstartes ohne abschließende Adaption
TRUE
Adaption
Schritt Adaption

Der Eingang *bEnable*, dessen Funktionsweise hier nicht dargestellt ist, setzt, wenn er auf FALSE steht, den Baustein faktisch außer Funktion: Als Vorstartzeit *uiPrestartTime* wird "0" ausgegeben und der oben dargestellte Ablauf wird nicht gestartet bzw. unmittelbar zurück gesetzt.

Adaption

Adaption

Ist eine Adaption angewählt und die Vorstartphase beendet (Countdown = 0 oder Raumtemperatur erreicht) so können 3 Fälle eintreten:

1. Der Countdown ist abgelaufen **und** die Raumtemperatur-Differenz ist unterhalb des Toleranz-Grenzwertes *rAdaptionTolerance* -> es erfolgt keine Adaption.
2. Der Countdown ist noch nicht abgelaufen, die Raum-Solltemperatur jedoch ist erreicht -> die Vorstartzeit *uiPrestartTime* war zu hoch.
3. Der Countdown ist abgelaufen, es besteht jedoch noch eine positive Temperaturdifferenz -> die Vorstartzeit *uiPrestartTime* war zu gering.

Anhand der abgelaufenen Countdown-Zeit und der erfolgten Änderung der Raumtemperaturdifferenz kann nun der Wert ΔT/Δt, neu bestimmt werden

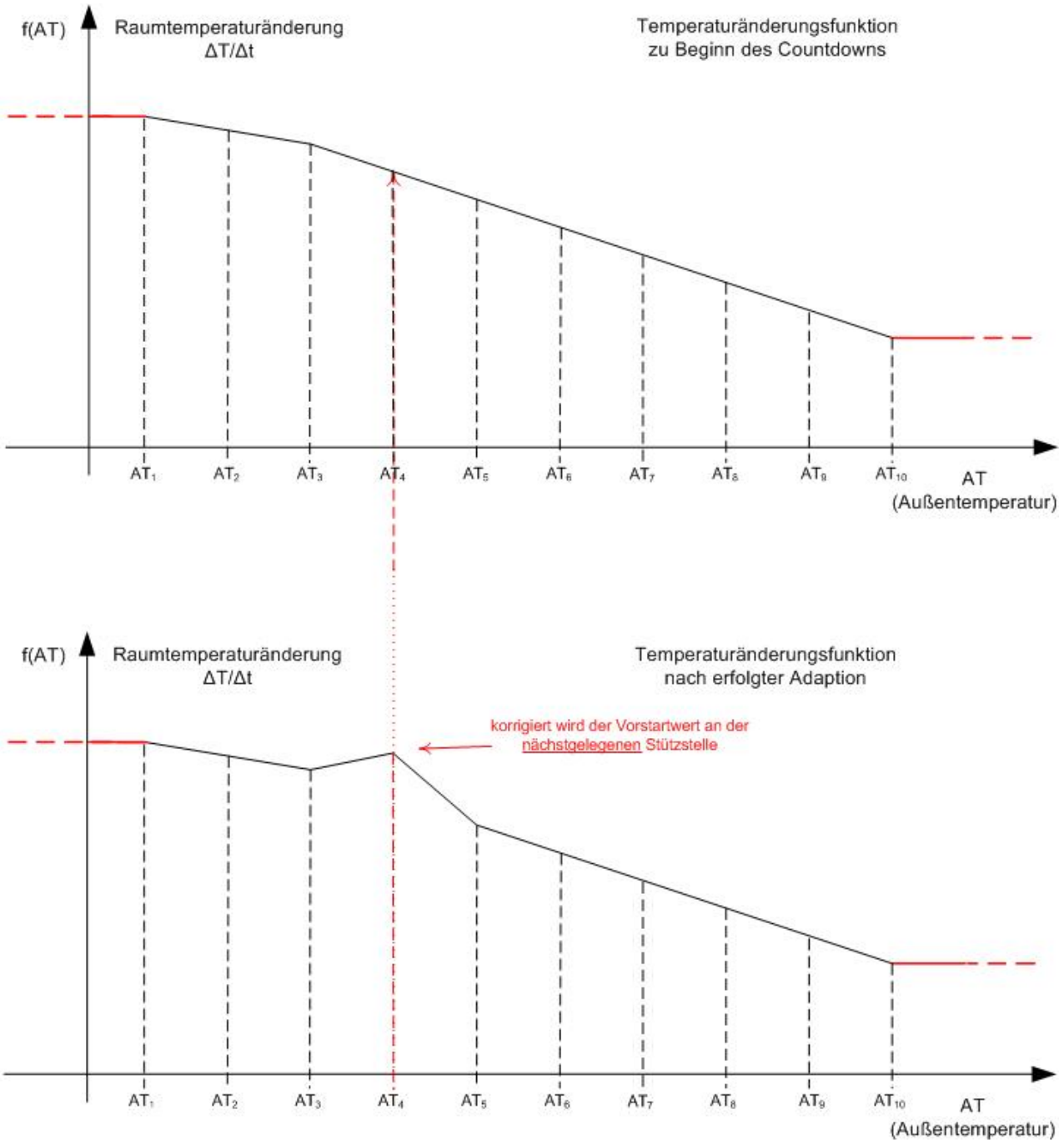
1. keine Änderung.
2. zu viel Zeit war berechnet - neuer Wert: "Raumtemperaturdifferenz am Anfang" / "bislang abgelaufene Zeit"
3. zu wenig Zeit war berechnet - neuer Wert: ("Raumtemperaturdifferenz am Anfang" -"Raumtemperaturdifferenz am Ende") / "Countdown-Zeit"

Die Temperaturänderungs-Funktion wird nun an der Stützstelle korrigiert, dessen Außentemperaturwert dem zu Beginn der Vorstopphase am nächsten lag. Der betreffende Punkt hierzu wurde vor Beginn des Countdowns abgespeichert, siehe Programmablaufplan.

Der zuvor berechnete Wert ΔT/Δt wird jedoch nicht unbedingt zu 100 Prozent als neuer Wert an der Stützstelle übernommen. Es besteht vielmehr die Möglichkeit den neuen Wert aus einer Gewichtung von altem und berechnetem Wert zu mischen. Diese Gewichtung geschieht mit Hilfe des so genannten Adaptionsfaktors *rAdaptionFactor*:

$$f(AT)_{NEU} := \frac{f(AT)_{ALT} \cdot (100 - rAdaptionFactor) + (\Delta T/\Delta t_{berechnet} \cdot rAdaptionFactor)}{100}$$

Bei einem Adaptionsfaktor von 100% wird damit der neu berechnete Wert voll übernommen, bei 0% bleibt der alte Wert erhalten.



Dadurch, dass immer die Vorstartzeit des nächstgelegenen Punktes verändert wird, kann eine Adaption nur dann stattfinden, wenn die Außentemperatur zu Beginn der Vorstartphase innerhalb von AT1 bis AT10 liegt.

Ein- und Ausgabevariablen

VAR_INPUT

```
eDataSecurityType   : E_HVACDataSecurityType;
eCtrlFct            : E_BARCtrlFct;
bEnable             : BOOL;
bSchedulerActive    : BOOL;
bWithAdaption       : BOOL;
bDisableAdaption    : BOOL;
rOutsideTemperature: REAL;
rRoomTemperature    : REAL;
rRoomSetpointValue : REAL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eDataSecurityType_Persistentist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

eCtrlFct : Mit *eCtrlFct* = *eBARCtrlFct_Heating* wird dem Baustein angezeigt, dass es sich um eine Nutzung im Heizbetrieb handelt. Bei *eCtrlFct* = *eBARCtrlFct_Cooling* wird der Kühlbetrieb angezeigt. Jede andere Eingabe an diesem Eingang ist nicht zulässig und führt zu einer Fehlermeldung. Die beiden möglichen Eingaben dienen der unterschiedlichen Berechnung der Temperaturdifferenz von Raumist- zu Raumsollwert.

bEnable : Ein FALSE-Signal an diesem Eingang unterdrückt das optimierte Einschalten der angeschlossenen Schaltuhren. Der Ausgabewert der Vorstartzeit wird dazu direkt auf "0" gesetzt. Weder wird ein Countdown gestartet, noch eine Adaption ausgeführt - der Baustein wird zurück gesetzt.

bSchedulerActive : Eine steigende Flanke an diesem Eingang startet den internen Countdown der Vorstartzeit. Während der Countdown einer Vorstartzeit abläuft, wird der Ausgang *bPrestartMode* auf TRUE gesetzt. Wird im Verlauf des Countdowns der Ausgang *bSchedulerActive* wieder auf FALSE geschaltet, so wird der Countdown unmittelbar unterbrochen und der Ausgang *bPrestartMode* auf FALSE gesetzt. Der Countdown wird ebenfalls gestoppt und *bPrestartMode* auf FALSE gesetzt, wenn der Raumsollwert erreicht ist.

bWithAdaption : Wird dieser Eingang *zeitgleich* mit dem Eingang *bSchedulerActive* gesetzt, so findet nach Ablauf des Countdowns eine Adaption statt. Hierbei reicht ein Triggerimpuls zusammen mit der steigenden Flanke von *bSchedulerActive* . Dieser Eingang kann nur in Verbindung mit *bSchedulerActive* genutzt werden, ein alleiniges Setzen hat keinerlei Auswirkung.

bDisableAdaption : Ein TRUE-Signal an diesem Eingang unterdrückt lediglich die dem Countdown folgende Adaption.

rOutsideTemperature : Außentemperatur in Grad Celsius.

rRoomTemperature : Raumtemperatur in Grad Celsius.

rRoomSetpointValue : Raumtemperatur-Sollwert in Grad Celsius.

VAR_OUTPUT

```
uiPrestartTime   : UINT;
bPrestartActive  : BOOL;
udiCountdownTime : UDINT;
bError           : BOOL;
udiErrorID       : UDINT;
```

uiPrestartTime : Ausgabewert der optimierten Vorstartzeit an die betreffenden Schaltuhren in Minuten. Dieser Wert wird kontinuierlich aus der Außentemperatur-abhängigen Vorstartfunktion gebildet. Steht der Eingang *bDisableOptimization* hingegen auf TRUE ist, so wird dieser Ausgang auf "0" gesetzt.

bPrestartActive : Solange der interne Countdown läuft und nicht durch Ablauf der Vorstartzeit oder Erreichen der Raumsolltemperatur oder Unterbrechung (*bSchedulerActive* = FALSE) beendet wird, befindet sich der Baustein im Vorstart-Modus. Dies wird durch ein TRUE-Signal an diesem Ausgang angezeigt.

udiCountdownTime : Dieser Ausgang zeigt das Ablaufen des internen Countdowns in Sekunden an. Ist der Baustein nicht mehr im Vorstart-Modus (siehe *bPrestartMode*), so wird dieser Ausgang auf "0" gesetzt.

bError : Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId : Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

VAR_IN_OUT

Die Notwendigkeit, dass eingetragene Parameter über einen Steuerungsausfall hinweg erhalten bleiben, macht es erforderlich, dass sie als In-Out-Variablen deklariert werden. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variable wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

```
uiMaxPrestartTime   : UINT;
rAdaptionFactor     : REAL;
rAdaptionTolerance  : REAL;
stTempChangeFunction: ST_HVACTempChangeFunction;
```

uiMaxPrestartTime : Durch die Adaption werden die Vorstartzeiten innerhalb der Temperaturänderungs-Funktion nach unten und nach oben hin verändert. Während sie nach unten sinngemäß automatisch auf 0 Minuten begrenzt sind, lässt sich durch diesen Eingang die Begrenzung nach oben in Minuten festlegen.

rAdaptionFactor : Im Adaptionsschritt wird für die nächst gelegene Außentemperatur-Stützstelle die erfolgte Temperaturänderung ΔT/Δt berechnet. Diese wird jedoch nicht unbedingt zu 100 Prozent als neuer Wert übernommen. Es besteht vielmehr die Möglichkeit den neuen Wert aus einer Gewichtung von altem und berechnetem Wert zu mischen. Der Adaptionsfaktor (in Prozent) stellt dabei die Gewichtung dar.

$$f(AT)_{NEU} := \frac{f(AT)_{ALT} \cdot (100 - rAdaptionFactor) + (\Delta T/\Delta t_{berechnet} \cdot rAdaptionFactor)}{100}$$

Bei einem Adaptionsfaktor von 100% wird damit der neu berechnete Wert voll übernommen, bei 0% bleibt der alte Wert erhalten.

rAdaptionTolerance : Ist bei eingeschalteter Adaption der Countdown abgelaufen und der Raumsollwert erreicht, so wird keine Adaption durchgeführt, da die Vorstartzeit exakt ausreicht. Der Wert *rAdaptionTolerance* definiert einen Toleranzbereich: liegt der Istwert im Bereich von *rRoomSetpointValue* .. *rRoomSetpointValue* + *rAdaptionTolerance*, so wird auch dieses als Erreichen des Sollwertes angesehen.

stTempChangeFunction : Strukturvariable vom Typ ST_HVACTempChangeFunction [▶ 538], welche die 10 Wertepaare (Außentemperatur, Innentemperatur-Änderung) enthält. Diese Wertepaare, welche nach aufsteigender Außentemperatur sortiert in der Feldvariable eingetragen sein müssen, definieren die 9 Teilgeraden der Temperaturänderungs-Funktion. Initialeingabe durch FB_HVACTempChangeFunctionEntry [▶ 459].

3.6.27 FB_HVACOptimizedOff

Funktionsbaustein zum optimierten Ausschalten (Vorstopp) von Heizkesseln und Klimaanlagen in Verbindung mit Schaltzeitbausteinen.

Die Ausführungen in dieser Dokumentation beziehen sich auf das Heizverhalten. Der Einsatz des Bausteines in Verbindung mit Kühlgeräten erfolgt analog.

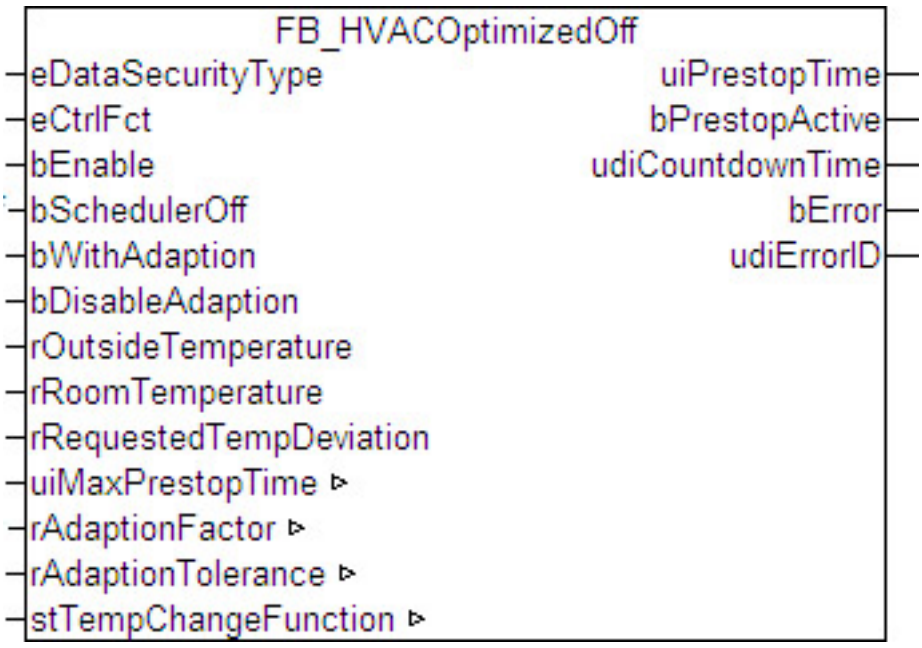
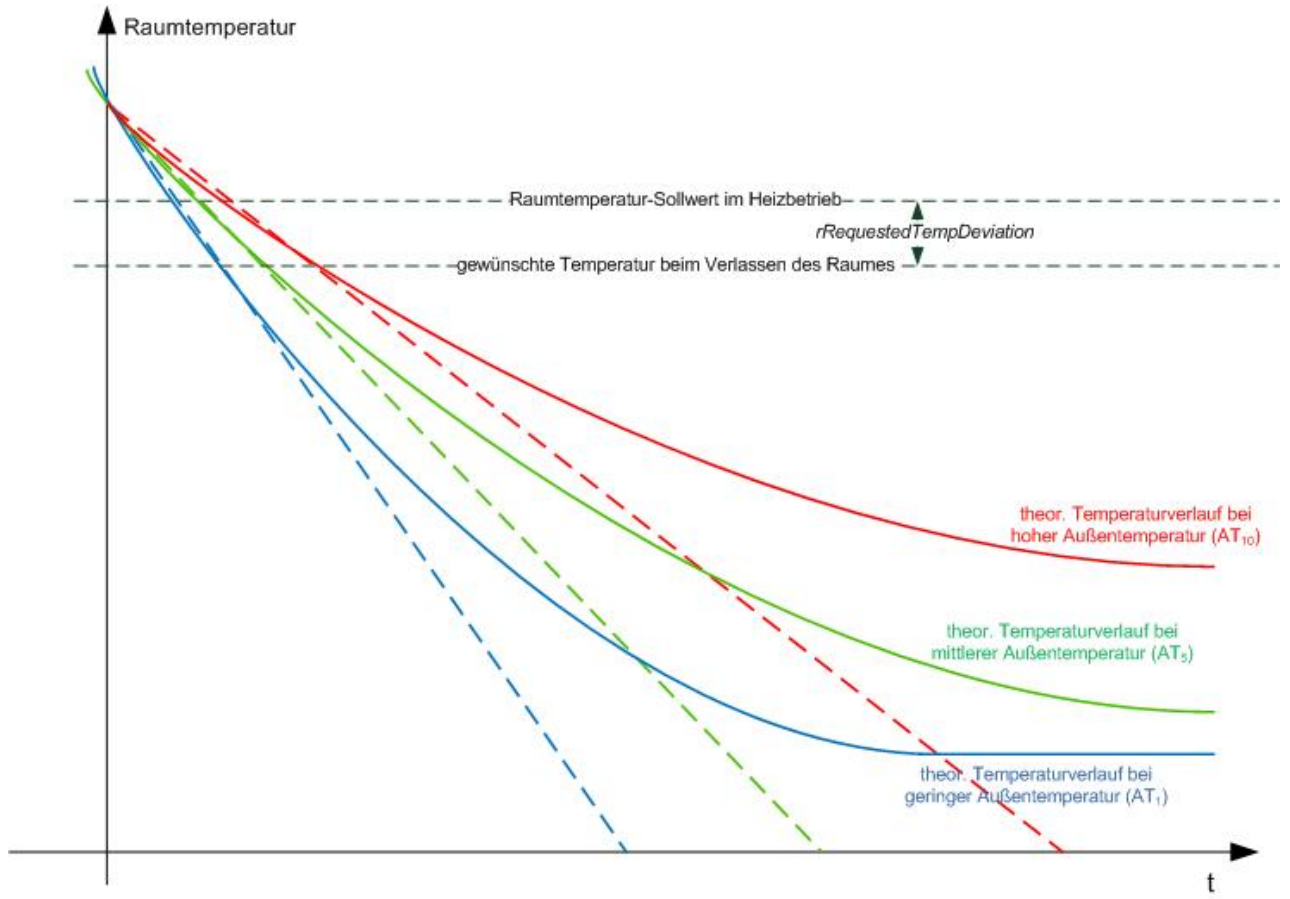


Abb. 18: FB_HVACOptimizedOff

Berechnung der Vorstoppzeit

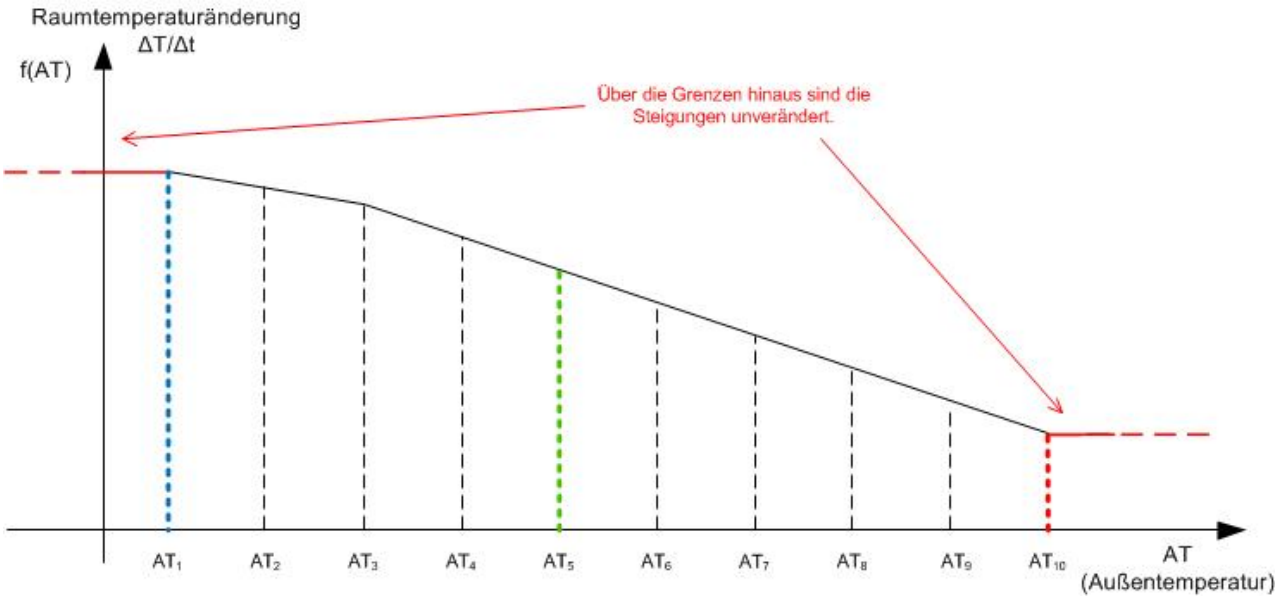
Gebäude und Gebäudeteile, welche zu einem bestimmten und planbarem Zeitpunkt belegt sind, wie beispielsweise Schulen oder Konferenzräume, müssen nicht über die gesamten Belegt-Zeit beheizt werden. Dadurch, dass die Gebäude die Wärme speichern, ist es möglich, die Heizung etwas früher abzuschalten. Die Raumtemperatur fällt dann bis zum Belegungs-Ende auf ein kaum spürbares niedrigeres Temperaturniveau. Es handelt sich hierbei um ein gewolltes Abfallen auf eine Zieltemperatur - die Abfalldifferenz ist ein Eingangsparameter des Bausteines und wird *rRequestedTempDeviation* genannt.

Der Vorlauftemperatur des Heizkessels richtet sich nach der Außentemperatur. Damit ergeben sich für unterschiedliche Außentemperaturen verschiedene Abkühlverläufe, die im Wesentlichen einer Exponentialfunktion entsprechen:



Zur Ermittlung der Vorstoppzeit wird angenommen, dass der Bereich der Funktionen zwischen Raumtemperatur-Startwert und Raumtemperatur-Sollwert linear ist. Damit ergibt sich für jede Außentemperatur eine charakteristische Temperaturänderung ΔT/Δt, welche hier durch die gestrichelte Linie dargestellt wird.

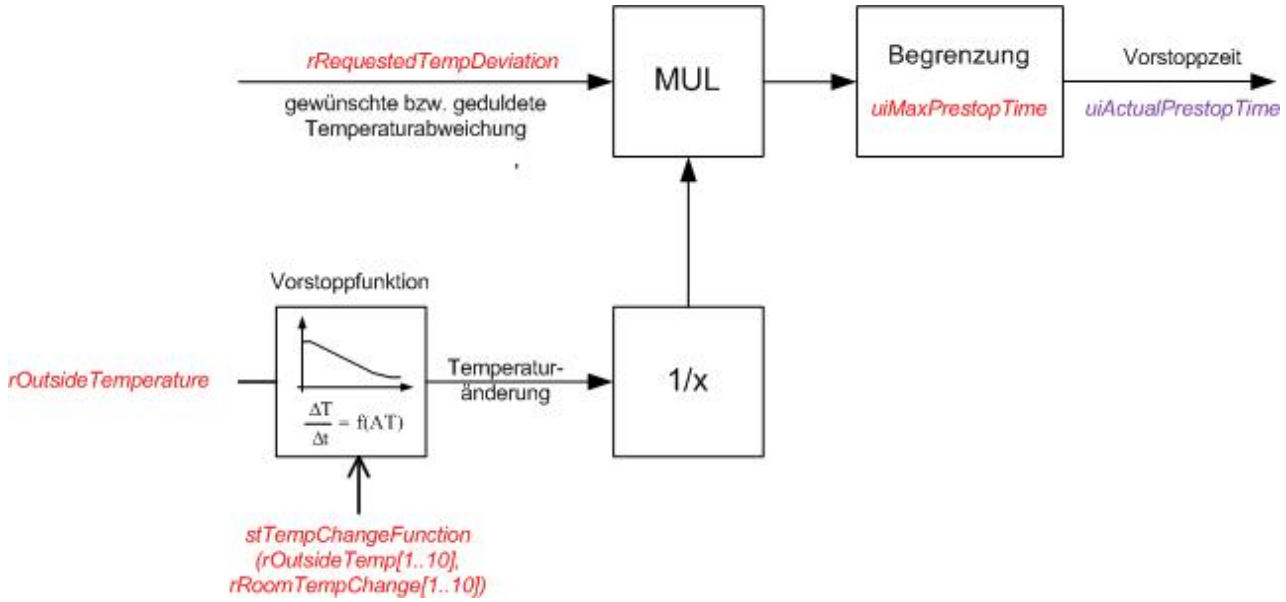
Dem Funktionsbaustein FB_HVACOptimizedOff liegt mit der Strukturvariable stTempChangeFunction [▶ 538] eine Tabelle zugrunde, in der für 10 diskrete Außentemperaturen die jeweilig zu erwartende Raumtemperaturänderung zugeordnet ist. Bei Inbetriebnahme bzw. Erststart der Anlage muß die Vorstartfunktion grob vordefiniert sein, zum einen, um den Außentemeraturbereich festzulegen, zum anderen um den Adaptionsvorgang zu beschleunigen. Diese Eingabe geschieht mit dem Baustein FB_HVACTempChangeFunctionEntry [▶ 459]. Mit diesen Werten lässt sich näherungsweise die Vorstoppzeit ermitteln. Diese Temperaturänderungs-Funktion f(AT) nimmt typischerweise folgenden Verlauf an, wobei die Änderung als Betrag, also positiv dargestellt ist:



Funktionswerte innerhalb dieser 10 Punkte werden über Geradengleichungen definiert, Werte außerhalb entsprechen dem Funktionswert f(AT_1) bzw. f(AT_{10}):

- $AT < AT_1$: $f(AT) = f(AT_1)$
- $AT > AT_{10}$: $f(AT) = f(AT_{10})$

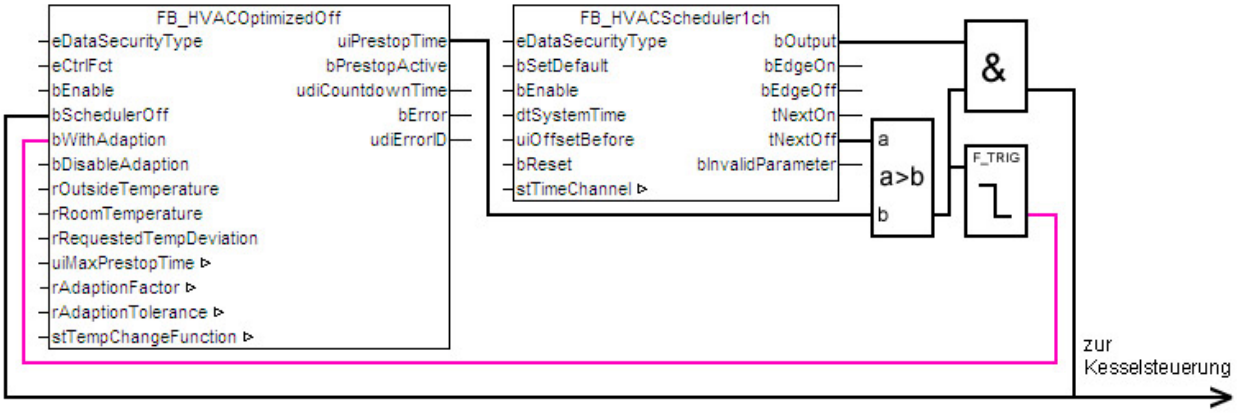
Die Ermittlung der Vorstoppzeit geschieht dann nach folgendem Ablauf:



Rot dargestellt sind hier die Ein- und Ausgabevariablen des Bausteines. Violett dargestellt ist hier der Interne Merker *uiActualPrestopTime* der zu jedem Zeitpunkt die aktuelle Vorstoppzeit enthält. Der Ausgang *uiPrestopTime* entspricht diesem Merker, wird jedoch während der Vorstoppphase "eingefroren". Dieses wird im Folgenden weiter erläutert.

Bausteinverknüpfung und Vorstoppphase

Aus diesen Vorgaben wird der Baustein dann kontinuierlich eine Vorstoppzeit berechnen, die er dem Schaltuhrbaustein vorgibt. Soll ein Heizkessel dann beispielsweise um 20 Uhr abends abgeschaltet werden und der Vorstoppbaustein ermittelt um 19 Uhr dass bei Abschalten des Kessels in 60 Minuten das Temperaturniveau um den gewünschten Wert *rRequestedTempDeviation* abfallen würde, so wird der Heizkessel unmittelbar abgeschaltet. Das Abkühlverhalten wird dann in einem Referenzraum über die Vorstopphase hinweg beobachtet und die Temperaturänderungs-Kurve entsprechend korrigiert.



Der Vorstopp-Baustein gibt für die Schaltuhr eineVorstoppzeit an. Schaltet die Schaltuhr den Ausgang *bOutput* auf FALSE zurück, so wird dem Vorstopp-Baustein dies über seinen Eingang *bSchedulerOff* angezeigt. Intern startet der Vorstopp-Baustein dann einen Countdown mit der zuvor ausgegebenen Vorstoppzeit *uiPrestopTime* [min]. Der Countdown läuft entweder bis zum Ende durch oder wird mit Erreichen der gewünschten bzw. tolerierten Raumtemperatur vorzeitig abgeschlossen. Während des Countdowns wird der Ausgang *bPrestopActive* gesetzt.
Der Ausgang *uiPrestopTime* folgt kontinuierlich der oben aufgeführten Berechnung - während eines gestarteten Countdowns jedoch wir er konstant gehalten, damit eine Schwankung in der Außentemperatur nicht eine geringere Vorstoppzeit angibt und den Kessel plötzlich wieder eingeschaltet wird.

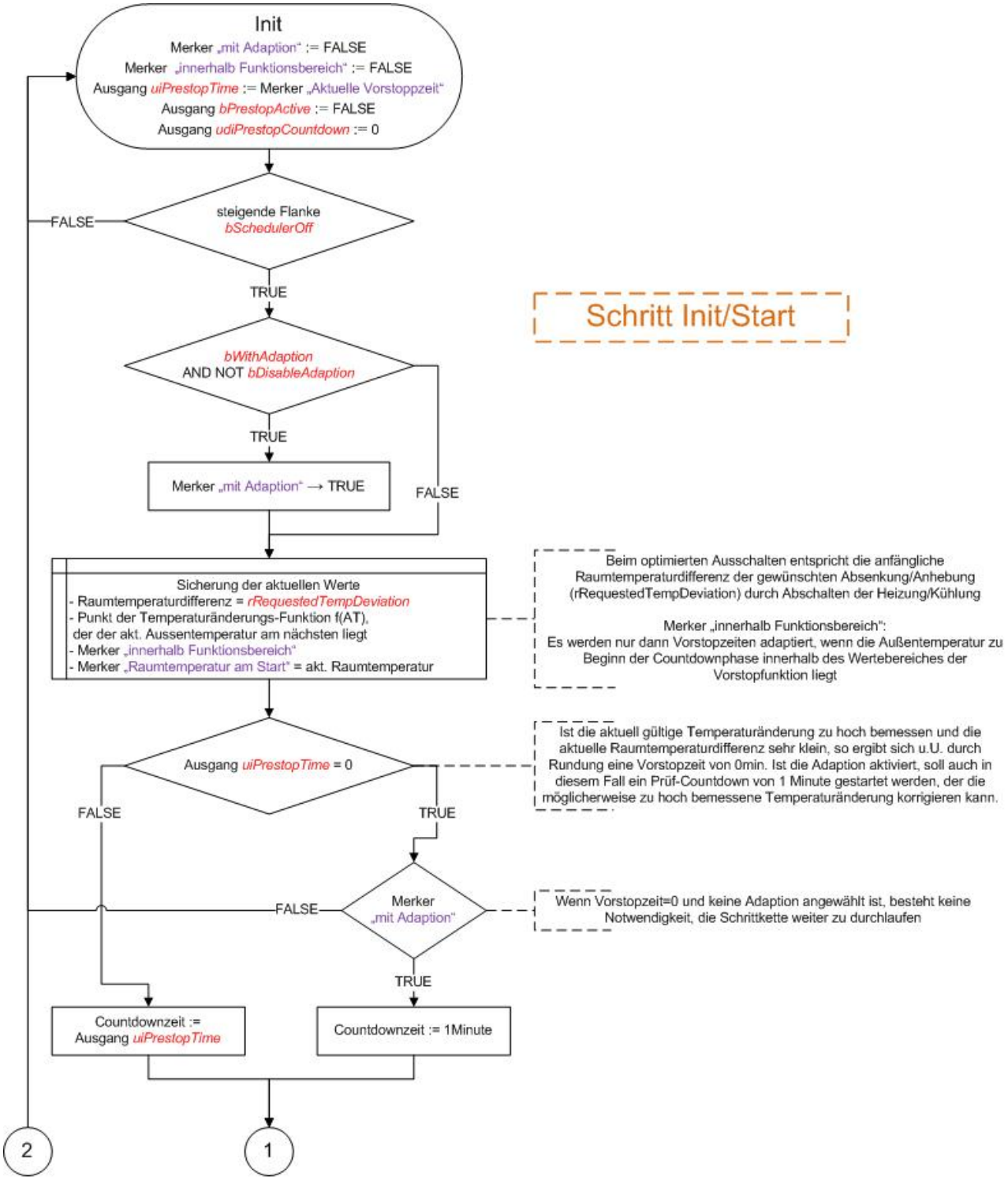
Teilt der Schaltuhrbaustein dem Optimierungsbaustein bei einem Vorstopp noch den Adaptionsauftrag als Flanke mit (rote Linie), so wird dieser nach der Vorstoppphase entscheiden, ob die zuvor ermittelte Vorstoppzeit innerhalb einer Toleranz genau oder zu klein bzw. zu groß bemessen war und die Temperaturänderungs-Funktion entsprechend korrigieren. Dabei wird die Vorstoppzeit des Punktes, dessen Außentemperatur der tatsächlichen zu Beginn des Countdowns am nächsten lag, nach oben bzw. nach unten hin korrigiert. Dieser Vorgang wird "Adaption [▶ 455]" genannt.

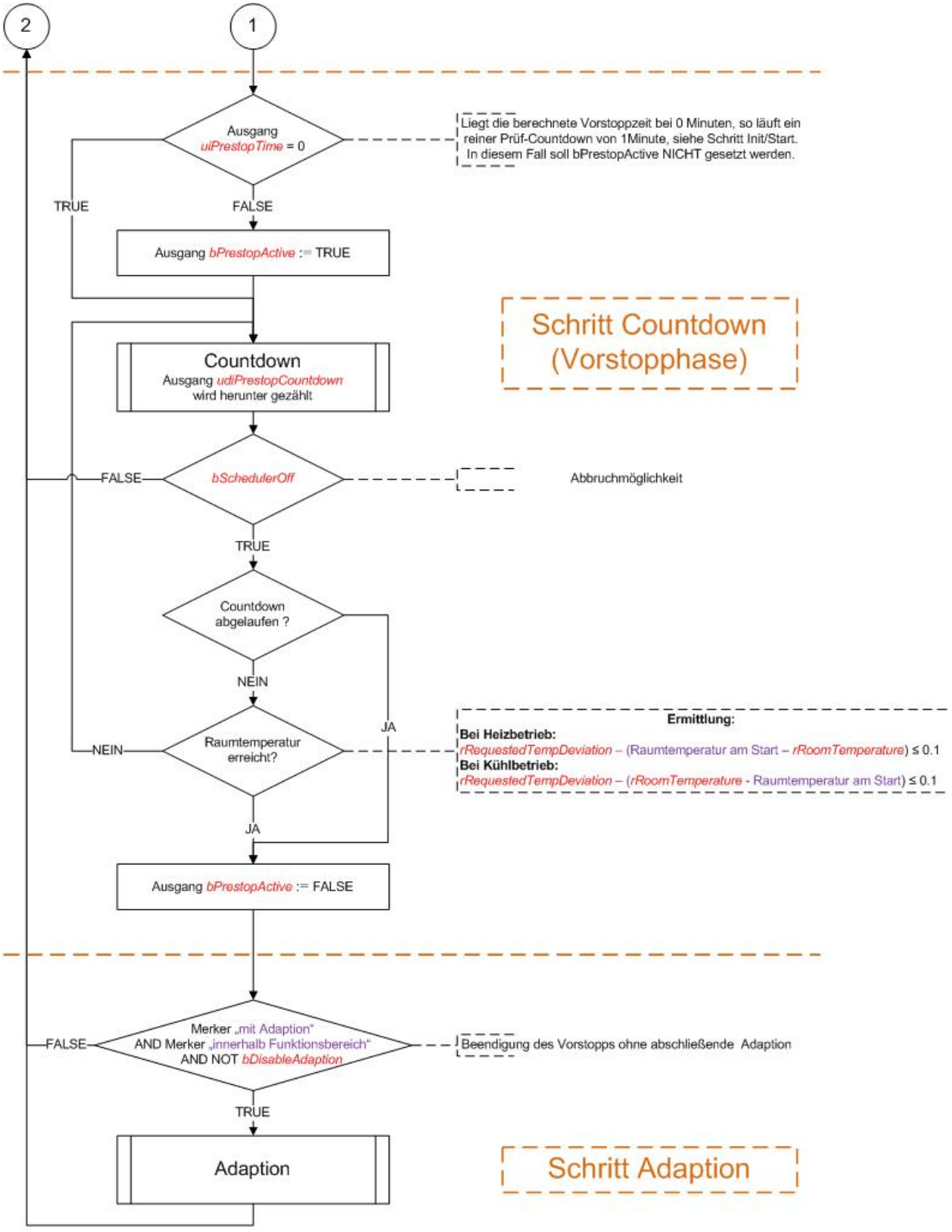
Es werden nur dann Temperaturänderungen korrigiert, wenn die Außentemperatur zu Beginn des Vorstopps innerhalb der Stützstellen, d.h. des Funktionsbereiches gelegen hat, siehe "Adaption [▶ 455]".

Bei gewünschter Adaption muss der Eingang *bWithAdaption* zeitgleich mit dem Eingang *bSchedulerOff* gesetzt sein. Für *bWithApation* reicht dabei ein reiner Triggerimpuls. Ein Abfallen des Einganges *bSchedulerOff* unterbricht einen zuvor gestarteten Countdown sofort. Die Adaption kann durch den Eingang *bDisableAdaption* unterdrückt werden. Diese Option ist dann zu wählen, wenn nach einer gewissen Anzahl von Adaptionsvorgängen die Temperaturänderungs-Funktion nicht mehr verändert werden soll.

Der folgende Programmablaufplan soll das Verhalten besser veranschaulichen, rot dargestellt sind hier die Ein- und Ausgabevariablen des Bausteines:

Der interne Merker "Aktuelle Vorstoppzeit" ist das ständig aktualisierte Ergebnis der Berechnung nach dem oben aufgeführten Diagramm.


Init
Merker „mit Adaption“ := FALSE
Merker „innerhalb Funktionsbereich“ := FALSE
Ausgang uiPrestopTime := Merker „Aktuelle Vorstoppzeit“
Ausgang bPrestopActive := FALSE
Ausgang udiPrestopCountdown := 0
steigende Flanke
bSchedulerOff
FALSE
TRUE
bWithAdaption
AND NOT bDisableAdaption
TRUE
FALSE
Merker „mit Adaption“ → TRUE
Schritt Init/Start
Sicherung der aktuellen Werte
- Raumtemperaturdifferenz = rRequestedTempDeviation
- Punkt der Temperaturänderungs-Funktion f(AT),
der der akt. Aussentemperatur am nächsten liegt
- Merker „innerhalb Funktionsbereich“
- Merker „Raumtemperatur am Start“ = akt. Raumtemperatur
Beim optimierten Ausschalten entspricht die anfängliche Raumtemperaturdifferenz der gewünschten Absenkung/Anhebung (rRequestedTempDeviation) durch Abschalten der Heizung/Kühlung
Merker „innerhalb Funktionsbereich“:
Es werden nur dann Vorstopzeiten adaptiert, wenn die Außentemperatur zu Beginn der Countdownphase innerhalb des Wertebereiches der Vorstopfunktion liegt
Ausgang uiPrestopTime = 0
FALSE
TRUE
Ist die aktuell gültige Temperaturänderung zu hoch bemessen und die aktuelle Raumtemperaturdifferenz sehr klein, so ergibt sich u.U. durch Rundung eine Vorstopzeit von 0min. Ist die Adaption aktiviert, soll auch in diesem Fall ein Prüf-Countdown von 1 Minute gestartet werden, der die möglicherweise zu hoch bemessene Temperaturänderung korrigieren kann.
Merker „mit Adaption“
FALSE
TRUE
Wenn Vorstopzeit=0 und keine Adaption angewählt ist, besteht keine Notwendigkeit, die Schrittkette weiter zu durchlaufen
Countdownzeit := Ausgang uiPrestopTime
Countdownzeit := 1Minute
2
1



Der Eingang *bEnable*, dessen Funktionsweise hier nicht dargestellt ist, setzt, wenn er auf FALSE steht, den Baustein faktisch außer Funktion: Als Vorstoppzeit *uiPrestopTime* wird "0" ausgegeben und der oben dargestellte Ablauf wird nicht gestartet bzw. unmittelbar zurückgesetzt.

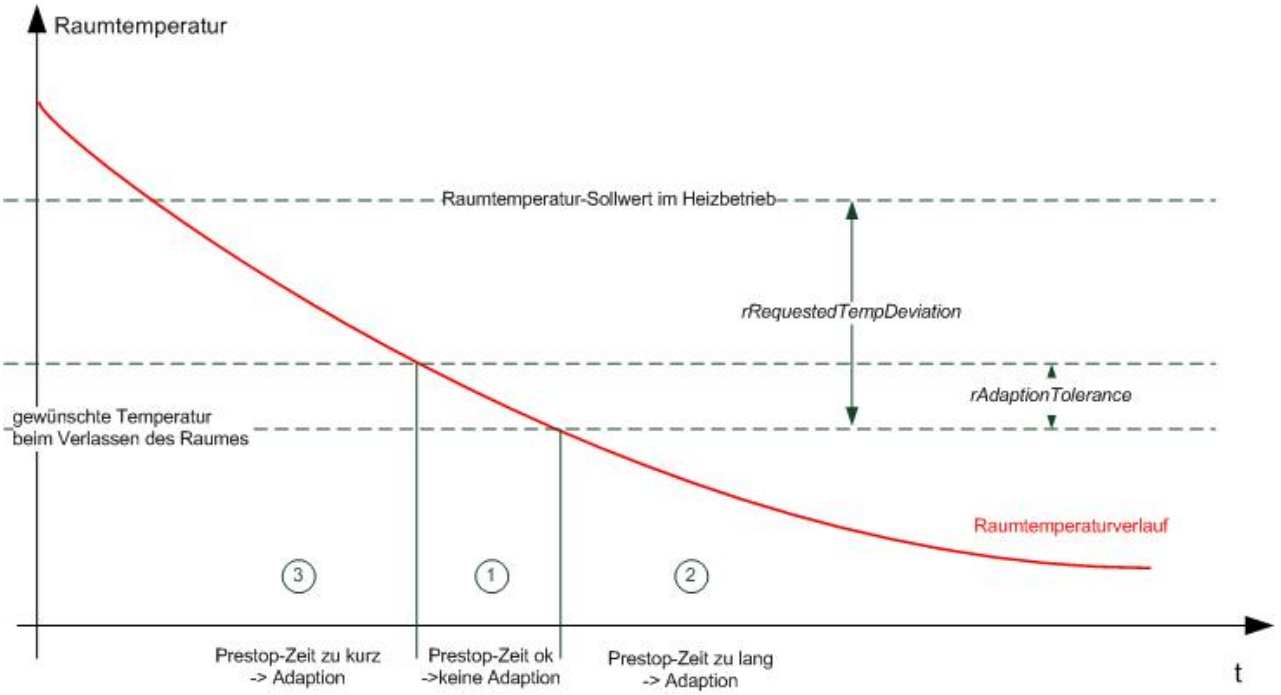
Adaption

Adaption

Ziel der Adaption ist es, die Temperaturänderungsfunktion derart genau anzupassen, dass in der ermittelten Vorstoppzeit die gewünschte bzw. tolerierte Temperaturabsenkung rRequestedTempDeviation erreicht wird (im Kühlbetrieb wäre es die Temperaturanhebung nach Abschalten der Kühlgeräte).

Ist eine Adaption angewählt und die Vorstopphase beendet (Countdown = 0 oder Raumtemperatur erreicht) so können 3 Fälle eintreten:

1. Der Countdown ist abgelaufen **und** die gewünschte Abweichung *rRequestedDeviation* ist bis auf eine Toleranz *rAdaptionTolerance* ausgeschöpft -> es erfolgt keine Adaption.
2. Der Countdown ist noch nicht abgelaufen, die gewünschte Abweichung *rRequestedDeviation* jedoch ist überschritten -> die Vorstoppzeit *uiPrestopTime* war zu hoch.
3. Der Countdown ist abgelaufen, die gewünschte Abweichung *rRequestedDeviation* jedoch ist noch nicht ganz ausgeschöpft -> die Vorstoppzeit *uiPrestopTime* war zu gering, es kann also noch früher abgeschaltet werden.



Anhand der abgelaufenen Countdown-Zeit und der erfolgten Änderung der Raumtemperaturdifferenz kann nun der Wert ΔT/Δt, neu bestimmt werden

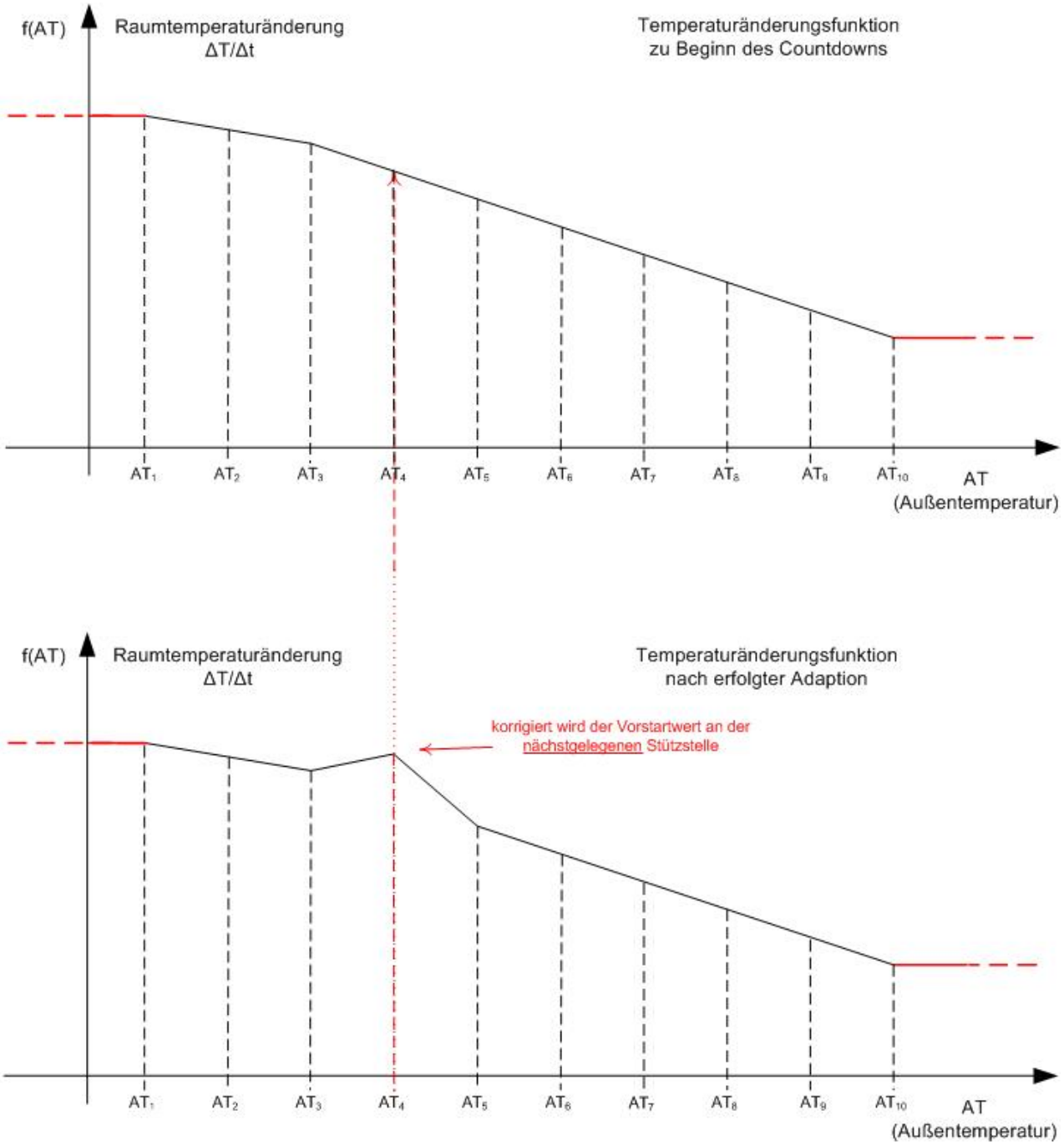
1. keine Änderung.
2. zu viel Zeit war berechnet - neuer Wert: "gewünschte Abweichung *rRequestedDeviation*" / "bislang abgelaufene Zeit"
3. zu wenig Zeit war berechnet - neuer Wert: ("gewünschte Abweichung *rRequestedDeviation*" -"Abweichung am Ende") / "Countdown-Zeit"

Die Temperaturänderungs-Funktion wird nun an der Stützstelle korrigiert, dessen Außentemperaturwert dem zu Beginn der Vorstopphase am nächsten lag. Der betreffende Punkt hierzu wurde vor Beginn des Countdowns abgespeichert, siehe Programmablaufplan.

Der zuvor berechnete Wert ΔT/Δt wird jedoch nicht unbedingt zu 100 Prozent als neuer Wert an der Stützstelle übernommen. Es besteht vielmehr die Möglichkeit den neuen Wert aus einer Gewichtung von altem und berechnetem Wert zu mischen. Diese Gewichtung geschieht mit Hilfe des so genannten Adaptionsfaktors *rAdaptionFactor*:

$$f(AT)_{NEU} := \frac{f(AT)_{ALT} \cdot (100 - rAdaptionFactor) + (\Delta T/\Delta t_{berechnet} \cdot rAdaptionFactor)}{100}$$

Bei einem Adaptionsfaktor von 100% wird damit der neu berechnete Wert voll übernommen, bei 0% bleibt der alte Wert erhalten.



Hinweis: Dadurch, dass immer die Vorstoppzeit des nächst gelegenen Punktes verändert wird, kann eine Adaption nur dann stattfinden, wenn die Außentemperatur zu Beginn der Vorstopphase *innerhalb* von AT_1 bis AT_{10} liegt.

Ein- und Ausgabevariablen

VAR_INPUT

```
eDataSecurityType       : E_HVACDataSecurityType;
eCtrlFct                : E_BARCtrlFct;
bEnable                 : BOOL;
bSchedulerOff           : BOOL;
bWithAdaption           : BOOL;
bDisableAdaption        : BOOL;
rOutsideTemperature     : REAL;
rRoomTemperature        : REAL;
rRequestedTempDeviation: REAL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eDataSecurityType_Persistentist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

eCtrlFct : Mit *eCtrlFct* = *eBARCtrlFct_Heating* wird dem Baustein angezeigt, dass es sich um eine Nutzung im Heizbetrieb handelt. Bei *eCtrlFct* = *eBARCtrlFct_Cooling* wird der Kühlbetrieb angezeigt. Jede andere Eingabe an diesem Eingang ist nicht zulässig und führt zu einer Fehlermeldung. Die beiden möglichen Eingaben dienen der unterschiedlichen Bewertung der Abweichung *rRequestedTempDeviation*.

bEnable : Ein FALSE-Signal an diesem Eingang unterdrückt das optimierte Ausschalten der angeschlossenen Schaltuhren. Der Ausgabewert der Vorstoppzeit wird dazu direkt auf "0" gesetzt. Weder wird ein Countdown gestartet, noch eine Adaption ausgeführt - der Baustein wird zurück gesetzt.

bSchedulerOff : Eine steigende Flanke an diesem Eingang startet den internen Countdown der Vorstoppzeit. Während der Countdown einer Vorstoppzeit abläuft, wird der Ausgang *bPrestopMode* auf TRUE gesetzt. Wird im Verlauf des Countdowns der Ausgang *bSchedulerOff* wieder auf FALSE geschaltet, so wird der Countdown unmittelbar unterbrochen und der Ausgang *bPrestopMode* auf FALSE gesetzt. Der Countdown wird ebenfalls gestoppt und *bPrestopMode* auf FALSE gesetzt, wenn der Raumsollwert erreicht ist.

bWithAdaption : Wird dieser Eingang *zeitgleich* mit dem Eingang *bSchedulerOff* gesetzt, so findet nach Ablauf des Countdowns eine Adaption statt. Hierbei reicht ein Triggerimpuls zusammen mit der steigenden Flanke von *bSchedulerOff* . Dieser Eingang kann nur in Verbindung mit *bSchedulerOff* genutzt werden, ein alleiniges Setzen hat keinerlei Auswirkung.

bDisableAdaption : Ein TRUE-Signal an diesem Eingang unterdrückt lediglich die dem Countdown folgende Adaption.

rOutsideTemperature : Außentemperatur in Grad Celsius.

rRoomTemperature : Raumtemperatur in Grad Celsius.

rRequestedTempDeviation : Tolerierte Temperaturabweichung nach unten (Heizbetrieb) oder nach oben (Kühlbetrieb) nach dem Abschalten des Kessels bzw. der Klimaanlage bis zum Verlassen des Raumes.

VAR_OUTPUT

```
uiPrestopTime   : UINT;
bPrestopActive  : BOOL;
udiCountdownTime: UDINT;
bError          : BOOL;
udiErrorID      : UDINT;
```

uiPrestopTime : Ausgabewert der optimierten Vorstoppzeit an die betreffenden Schaltuhren in Minuten. Dieser Wert wird kontinuierlich aus der Außentemperatur-abhängigen Vorstoppfunktion gebildet. Steht der Eingang *bDisableOptimization* hingegen auf TRUE ist, so wird dieser Ausgang auf "0" gesetzt.

bPrestopActive : Solange der interne Countdown läuft und nicht durch Ablauf der Vorstoppzeit oder Erreichen der Raumsolltemperatur oder Unterbrechung (*bSchedulerOff* = FALSE) beendet wird, befindet sich der Baustein im Vorstopp-Modus. Dies wird durch ein TRUE-Signal an diesem Ausgang angezeigt.

udiCountdownTime : Dieser Ausgang zeigt das Ablaufen des internen Countdowns in Sekunden an. Ist der Baustein nicht mehr im Vorstopp-Modus (siehe *bPrestopMode*), so wird dieser Ausgang auf "0" gesetzt.

bError : Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

udiErrorId : Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

VAR_IN_OUT

Die Notwendigkeit, dass eingetragene Parameter über einen Steuerungsausfall hinweg erhalten bleiben, macht es erforderlich, dass sie als In-Out-Variablen deklariert werden. Im Programm wird ihnen dann eine Referenz-Variable zugewiesen. Jede Änderung des Wertes dieser Referenz-Variable wird im Funktionsbaustein persistent gespeichert und nach einem Steuerungsausfall und -wiederanlauf zurück in die Referenz-Variable geschrieben. Wären die Parameter nur als Eingangsvariablen deklariert, so könnten sie eine Referenzvariable **nicht** beschreiben.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip.

```
uiMaxPrestopTime      : UINT;
rAdaptionFactor       : REAL;
rAdaptionTolerance    : REAL;
stTempChangeFunction  : ST_HVACTempChangeFunction;
```

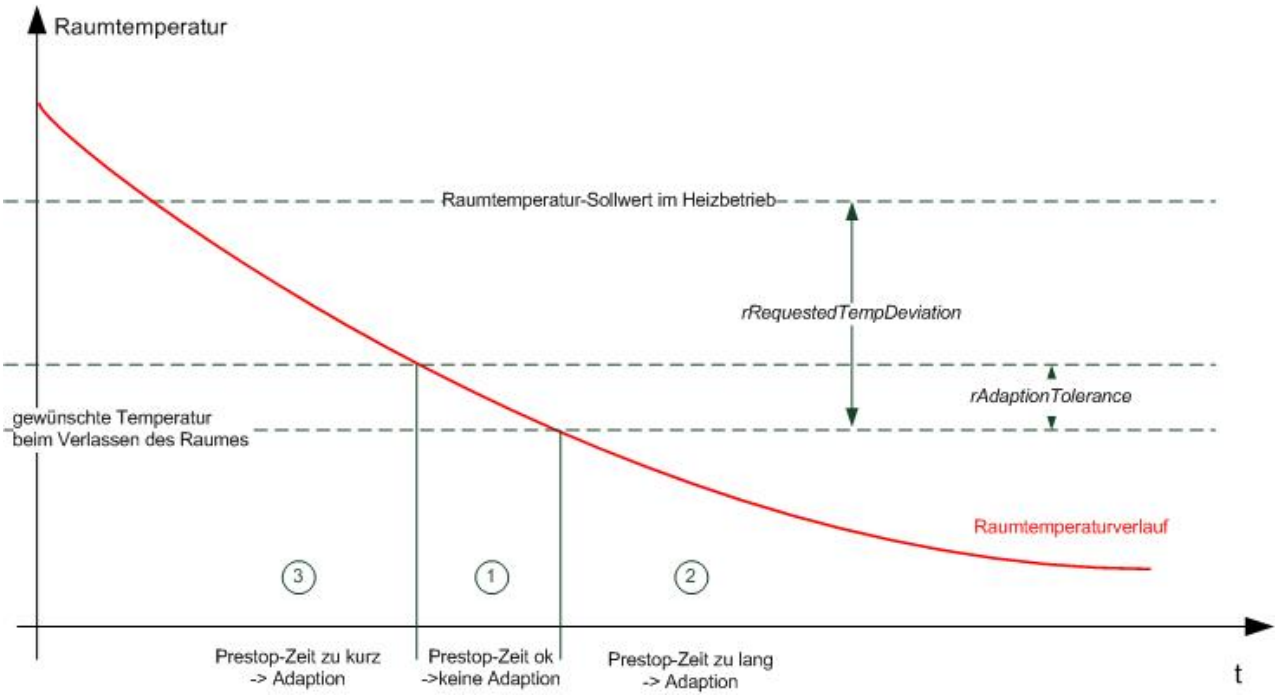
uiMaxPrestopTime : Durch die Adaption werden die Vorstoppzeiten innerhalb der Temperaturänderungs-Funktion nach unten und nach oben hin verändert. Während sie nach unten sinngemäß automatisch auf 0 Minuten begrenzt sind, lässt sich durch diesen Eingang die Begrenzung nach oben in Minuten festlegen.

rAdaptionFactor : Im Adaptionsschritt wird für die nächst gelegene Außentemperatur-Stützstelle die erfolgte Temperaturänderung ΔT/Δt berechnet. Diese wird jedoch nicht unbedingt zu 100 Prozent als neuer Wert übernommen. Es besteht vielmehr die Möglichkeit den neuen Wert aus einer Gewichtung von altem und berechnetem Wert zu mischen. Der Adaptionsfaktor (in Prozent) stellt dabei die Gewichtung dar.

$$f(AT)_{NEU} := \frac{f(AT)_{ALT} \cdot (100 - rAdaptionFactor) + (\Delta T/\Delta t_{berechnet} \cdot rAdaptionFactor)}{100}$$

Bei einem Adaptionsfaktor von 100% wird damit der neu berechnete Wert voll übernommen, bei 0% bleibt der alte Wert erhalten.

rAdaptionTolerance : Ist bei eingeschalteter Adaption der Countdown abgelaufen und die gewünschte Abweichung *rRequestedDeviation* ausgeschöpft, so wird keine Adaption durchgeführt, da die Vorstoppzeit exakt ausreicht. Der Wert *rAdaptionTolerance* definiert einen zusätzlichen Toleranzbereich:



stTempChangeFunction : Strukturvariable vom Typ ST_HVACTempChangeFunction [▶ 538], welche die 10 Wertepaare (Außentemperatur, Innentemperatur-Änderung) enthält. Diese Wertepaare, welche nach aufsteigender Außentemperatur sortiert in der Feldvariable eingetragen sein müssen, definieren die 9 Teilgeraden der Temperaturänderungs-Funktion. Initialeingabe durch FB_HVACTempChangeFunctionEntry [▶ 459].

3.6.28 FB_HVACTempChangeFunctionEntry

Funktionsbaustein zur Eingabe der Stützstellen der Vorstartfunktion.

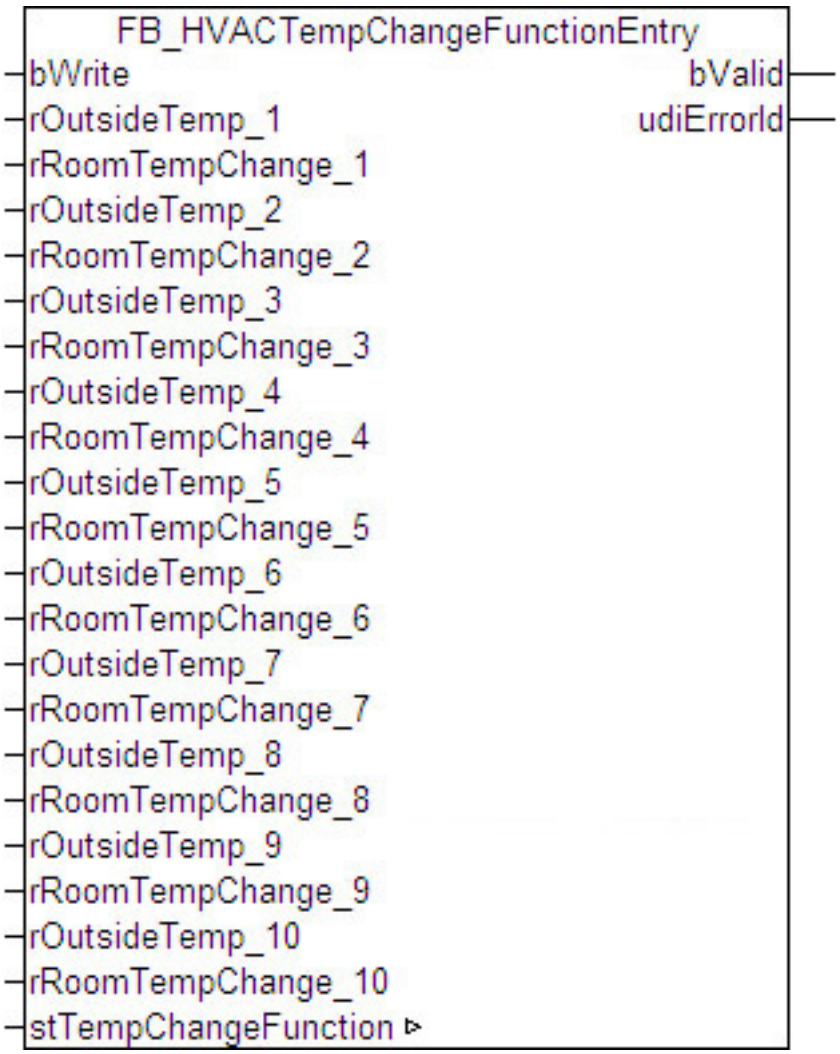


Abb. 19: FB_HVACTempChangeFunctionEntry

Um die Bausteine FB_HVACOptimizedOn [▶ 439] und FB_HVACOptimizedOff [▶ 448] übersichtlich zu halten, enthalten diese keine Eingabe der einzelnen Wertepaare für die Temperaturänderungs-Funktion - die Bausteine greifen über eine IN-OUT-Variable auf die Strukturvariable der Temperaturändeungs-Funktion (ST_HVACTempChangeFunction [▶ 538]) zu. Der Funktionsbaustein FB_HVACTempChangeFunction ermöglicht das Beschreiben der Strukturvariable in übersichtlicher Form und achtet zusätzlich darauf, ob die Wertepaare, wie gefordert, nach der Außentemperatur in aufsteigender Reihenfolge sortiert eingegeben wurden und dass es keine zwei Punkte derselben Außentemperatur gibt. In diesem Falle gäbe es mathematisch keinen eindeutigen funktionalen Zusammenhang. Die Wertepaare sind an den entsprechenden Eingängen *rOutsideTemperature_1.. rOutsideTemperature_10* (Außentemperatur) und *rRoomTempChange_1..rRoomTempChange_10* (Raumtemperatur-Änderung) einzugeben. Der Baustein überprüft kontinuierlich, ob die beschriebene Forderung nach aufsteigender Reihenfolge bezüglich der Außentemperatur erfüllt ist und ob keine zwei Wertepaare derselben Außentemperatur existieren.

Das Beschreiben der Temperaturänderungs-Funktionen sollte **einmalig** geschehen, um den Vorstart- Vorstopp-Bausteinen Grundwerte zu geben, die dann im Laufe der Zeit von diesen Bausteinen kontinuierlich verbessert werden.

VAR_INPUT

```
bWrite             : BOOL;
rOutsideTemp_1     : REAL;
rRoomTempChange_1  : REAL;
rOutsideTemp_2     : REAL;
rRoomTempChange_2  : REAL;
rOutsideTemp_3     : REAL;
rRoomTempChange_3  : REAL;
rOutsideTemp_4     : REAL;
rRoomTempChange_4  : REAL;
```

```
rOutsideTemp_5     : REAL;
rRoomTempChange_5  : REAL;
rOutsideTemp_6     : REAL;
rRoomTempChange_6  : REAL;
rOutsideTemp_7     : REAL;
rRoomTempChange_7  : REAL;
rOutsideTemp_8     : REAL;
rRoomTempChange_8  : REAL;
rOutsideTemp_9     : REAL;
rRoomTempChange_9  : REAL;
rOutsideTemp_10    : REAL;
rRoomTempChange_10 : REAL;
```

bWrite : Eine steigende Flanke an diesem Eingang kopiert die an den Eingängen eingetragenen Werte in die Vorstartfunktion.

(rOutsideTemp_1 - rRoomTempChange1) ... (rOutsideTemp_10 - rRoomTempChange10): Wertepaare der Vorstart-Funktion: Raumtemperaturänderung (*rRoomTempChange*) in Grad Kelvin pro Minute bei Außentemperatur (*rOutsideTemp*) in Grad Celsius.

VAR_OUTPUT

```
bValid          : BOOL;
udiErrorID      : UDINT;
```

bValid : Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter **nicht** fehlerhaft sind.

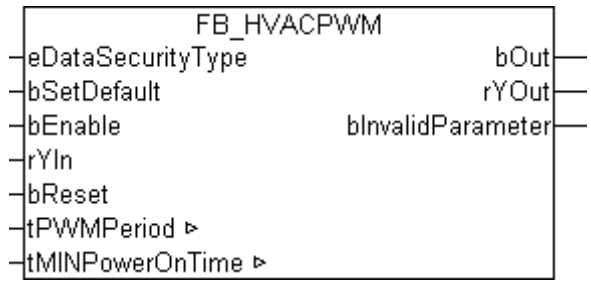
udiErrorId : Enthält den Fehlercode, sollten die eingetragenen Werte fehlerhaft sein. Siehe Fehlercodes [▶ 243].

VAR_IN_OUT

```
stTempChangeFunction  : ST_HVACTempChangeFunction;
```

stTempChangeFunction : Strukturvariable vom Typ ST_HVACTempChangeFunction [▶ 538], welche die 10 Wertepaare (Außentemperatur, Raumtemperaturänderung) enthält. Diese Wertepaare, welche nach aufsteigender Außentemperatur sortiert in der Feldvariable eingetragen sein müssen, definieren die 9 Teilgeraden der Vorstartfunktion.

3.6.29 FB_HVACPWM



Anwendung

Dieser Funktionsbaustein erzeugt aus dem analogen Eingangssignal *rYIn* ein PWM-moduliertes Signal. Des Weiteren kann eine Mindesteinschaltzeit parametriert werden.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
bEnable          : BOOL;
rYIn             : REAL;                0 .. 100       %
bReset           : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Freigabe des Bausteins, wenn *bEnable* = TRUE ist.

rYIn: Analoge Eingangsgröße des Funktionsbausteins.

bReset: Quittierungseingang bei einer Störung.

VAR_OUTPUT

```
bOut             : BOOL;
rYOut            : REAL;     0 .. 100       %
bInvalidParameter: BOOL;
```

bOut: PWM-Signal.

rYOut: Ausgabe der Eingangsgröße des Funktionsbausteines.

bInvalidParameter: Zeigt an, dass ein falscher Eingangsparameter anliegt. *bInvalidParameter* muss mit *bReset* quittiert werden.

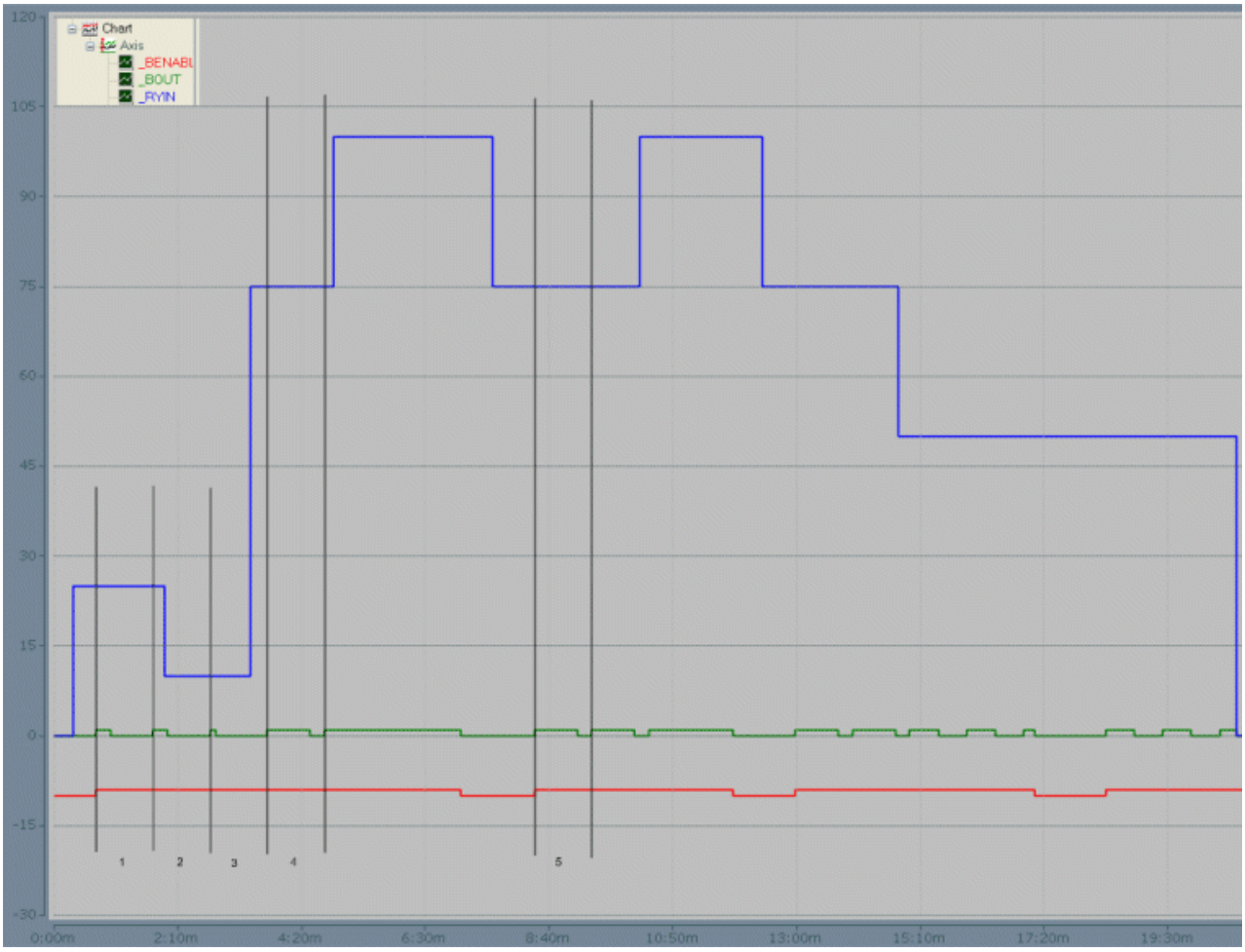
VAR_IN_OUT

```
tPWMPeriod      : TIME;
tMINPowerOnTime : TIME;
```

tPWMPeriod: Periodendauer des PWM-Signals. Die Variable wird persistent gespeichert. Voreingestellt auf 30min.

tMINPowerOnTime: Mindesteinschaltzeit des gepulsten Ausgangs *bOut*. Die Variable wird persistent gespeichert. Voreingestellt auf 0s.

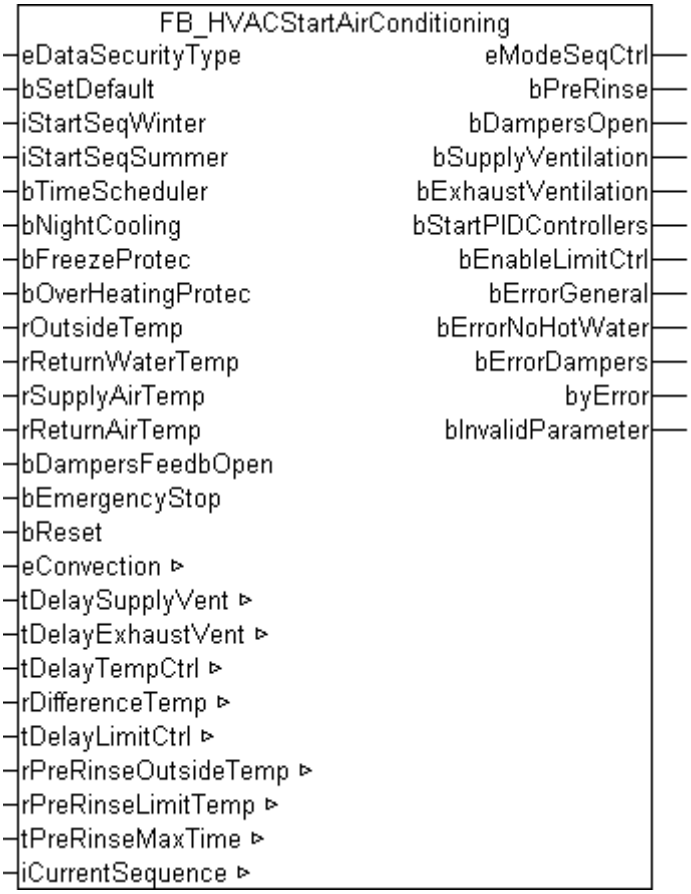
Bild1: Scope2 Aufzeichnung, die die funktionsweise des Bausteines zusätzlich erklären soll. Mit der Freigabe des Bausteines (*bEnable* = TRUE) wird der Ausgang (*bOut* = TRUE) gesetzt. Siehe auch Abschnitt 1 und 5. Eine Veränderung der Eingangsgröße während einer laufenden Periode wird erst in der nächsten Periode berücksichtigt. Siehe auch Abschnitt 2, 3 und 4.



Dokumente hierzu

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

3.6.30 FB_HVACStartAirConditioning



Anwendung

Mit dem Startprogramm *FB_HVACStartAirConditioning* wird die raumlufttechnische Anlage stufenweise angefahren. Dabei werden nacheinander die Absperrklappen, die Ventilatoren, die Regelung und die Grenzwertüberwachung der Analogeingänge freigegeben.

Bei niedrigen Außentemperaturen unterhalb des Wertes von *rPreRinseOutsideTemp* wird vor dem Öffnen der Außen- und Fortluftklappe zunächst das Heizregister mit Warmwasser durchspült. Bei RLT-Anlagen mit einem Rücklauftemperaturfühler an dem Heizregister dauert der Spülvorgang so lange bis die Temperatur *rReturnWaterTemp*den Wert der Eingangsvariablen *rPreRinseLimitTemp*überschritten hat. Beim Erreichen der angegebenen Temperatur im Rücklauf des Lufterhitzers werden die Außen- und die Fortluftklappe geöffnet. Wird die Temperatur nach dem Ablauf des Timers *tPreRinseMaxTime*nicht überschritten, wird die Störmeldung mit einem TRUE am Ausgang *bErrorNoHotWater* gemeldet**.**

Falls in dem Rücklauf des Erhitzers kein Temperaturfühler vorhanden ist, wird an dem Eingang *rReturnWaterTemp*eine Konstante mit dem Wert **-100** angelegt. Der Funktionsbaustein **FB_HVACStartAirConditioning** erkennt so, dass der Vorspülvorgang ohne Rücklaufüberwachung ablaufen muss. In diesem Fall wird bis zum Ablauf des Timers *tPreRinseMaxTime*gespült. **Eine Überwachung des Spülvorgangs erfolgt so nicht!**

Nach der Beendigung des Vorspülvorgangs werden die Außen- und die Fortluftklappe geöffnet. Damit wird der Ausgang *bDampersOpen* TRUE.
Um Schäden an den Luftkanälen und den Klappen zu vermeiden, wird der Anlauf der Ventilatoren so lange verzögert bis die Rückmeldung *"Auf* "der Klappen am Eingang *bDampersFeedbOpen* ansteht. Zusätzlich muss der Timer *tDelaySupplyVent* abgelaufen sein. Die eingestellte Zeit muss größer als die Verfahrzeit des Klappenantriebs eingestellt werden.
Da dieser Timer auch beim Abschalten der Anlage wirksam ist, sind auch Herunterfahrrampen von Frequenzumrichtern zu berücksichtigen!
Erfolgt innerhalb der eingestellten Zeit keine Rückmeldung der Klappenantriebe wird der Ausgang *bErrorDampers*TRUE. Zur Vermeidung von hohen Anlaufströmen erfolgt das Zuschalten des Abluftventilators mit dem Timer *tDelayExhaustVent.*

Bei Anlagen mit einer Mischluftkammer ist es sinnvoll die Regelung erst frei zu geben, wenn die Zulufttemperatur *rSupplyAirTemp*annähernd den Wert der Ablufttemperatur erreicht hat. Die maximale Abweichung wird mit dem Parameter *rDifferenceTemp* angegeben. Bei Anlagen ohne Mischluftsystem wird hier einfach ein sehr großer Wert als Konstante z.B. 100 angegeben. Die Regelung wird dann unverzögert freigegeben.

Nach dem Ablauf des Timers *tDelayLimitCtrl*wird der Ausgang *bEnableLimitCtrl*frei gegeben und damit ist die Grenzwertüberwachung der Analogeingänge aktiviert.

Bei Außentemperaturen unterhalb des Wertes von *rPreRinseOutsideTemp* wird an dem Ausgang der Wert von *iCurrentSequence* gleich dem von *iStartSeqWinter* gesetzt. Es muss hier die Nummer des Vorerhitzerreglers eingetragen werden. (Siehe Kapitel Sequenzregler) Bei Außentemperaturen größer als *rPreRinseOutsideTemp*kann mit der Wärmerückgewinnung oder der Mischluftkammer angefahren werden. Am Eingang *iStartSeqSummer* wird also die Nummer des Sequenzreglers von der Mischluftkammer oder von der Wärmerückgewinnung eingegeben.

Die Anfahrschaltung wird bei Betriebsanforderungen der Automatikprogramme, der Zeitschaltplänen, Sommernachtkühlung, Stützbetrieb und Überhitzungsschutz aktiv.
Zur Ansteuerung der Sequenzregler wird die Aufzählungsvariable *eModeSeqCtrl*am Ausgang des Funktionsbausteines wie folgt gesetzt:

```
TYPE E_HVACSequenceCtrlMode :
(
eHVACSequenceCtrlMode_Stop := 0,
eHVACSequenceCtrlMode_On := 1,
eHVACSequenceCtrlMode_NightCooling := 2,
eHVACSequenceCtrlMode_FreezeProtection := 3,
eHVACSequenceCtrlMode_OverheatingProtection := 4,
eHVACSequenceCtrlMode_NightCoolingAndOverheatingProtection := 5
);
END_TYPE
```

Die Anforderungen aller Automatikprogramme werden an das Anlagenstartprogramm mittels der Eingangsvariablen *bTimeScheduler*, *bNightCooling*, *bFreezeProtec* und *bOverHeatingProtec* übergeben. Die Anforderungen der Erhaltungsfunktionen kommen nur zur Geltung, wenn vom Zeitschaltprogramm an der Eingangsvariable *bTimeScheduler* keine Anforderung vorliegt.

Die Programme für die Sommernachtkühlung (bNightCooling), den Auskühlschutz (bFreezeProtec) und den Überhitzungsschutz (bOverHeatingProtec) sind nur einsetzbar, wenn ein Raumtemperaturfühler statt eines Ablufttemperaturfühlers eingesetzt wird. Gegenüber den anderen Automatikprogrammen hat das Programm Auskühlschutz immer Vorrang, es sei den die Betriebsanforderung vom Zeitschaltprogramm steht an.

Da die Sommernachtkühlung als Zwangsbelüftung mit Ventilatoren oder als Konvektionsbelüftung ohne Ventilatoren erfolgen kann, wird der Befehl vom Programm FB_HVACSummerNightCooling mit der Aufzählungsvariablen E_HVACConvectionMode übertragen. Dieser Ausgang muss an die Eingangsvariable *eConvection* angelegt werden.

An den Eingang *bEmergencyStop* wird eine Sammlung aller Störmeldungen dieser RLT-Anlage angelegt, welche zur Abschaltung der Anlage führen.

Beim Abschalten der raumlufttechnischen Anlage erfolgen die Einschaltschritte in umgekehrter Reihenfolge.

- Deaktivieren der Limitüberwachung an den Analogeingängen
- Deaktivieren der Regler (Rücksetzen von *iCurrentSequence:= 0*)
- Abschalten der Ventilatoren
- Schließen der Klappen nach Ablauf des Timers *tDelaySupplyVent* .

VAR_INPUT

```
eDataSecurityType     : E_HVACDataSecurityType;
bSetDefault           : BOOL;
iStartSeqWinter       : INT;
iStartSeqSummer       : INT;
bTimeScheduler        : BOOL;
bNightCooling         : BOOL;
bFreezeProtec         : BOOL;
bOverHeatingProtec    : BOOL;
rOutsideTemp          : REAL;
```

```
rReturnWaterTemp      : REAL;
rSupplyAirTemp        : REAL;
rReturnAirTemp        : REAL;
bDampersFeedbOpen     : BOOL;
bEmergencyStop        : BOOL;
bReset                : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen. Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

iStartSeqWinter: Hier muss die Nummer des Sequenzreglers von dem Vorerhitzer eingegeben werden.

iStartSeqSummer: Hier muss die Nummer des Sequenzreglers von der Mischluftkammer oder von der Wärmerückgewinnung eingegeben werden.

bTimeScheduler: Eingang für den Zeitschaltuhrbefehl.

bNightCooling: Eingang für die Funktion Nachtkühlung.

bFreezeProtec: Eingang für die Funktion Frostschutz.

bOverHeatingProtec: Eingang für die Funktion Überhitzungsschutz.

rOutsideTemp: Eingang für die Außentemperatur.

rReturnWaterTemp: Eingang für die Wassertemperatur aus dem Rücklauf des Lufterhitzers.

rSupplyAirTemp: Eingang für die Zulufttemperatur.

rReturnAirTemp: Eingang für die Raumtemperatur oder Ablufttemperatur.

bDampersFeedbOpen: Eingang für die Rückmeldung von den Klappen.

bEmergencyStop: Über diesen Eingang kann eine Sammlung aller Störmeldungen der RLT-Anlage angelegt werden, welche dann zur Notabschaltung der RLT-Anlage führen.

bReset: Eingang zur Quittierung der Störung.

VAR_OUTPUT

```
eModeSeqCtrl         : E_HVACSequenceCtrlMode;
bPreRinse            : BOOL;
bDampersOpen         : BOOL;
bSupplyVentilation   : BOOL;
bExhaustVentilation  : BOOL;
bStartPIDControllers: BOOL;
bEnableLimitCtrl     : BOOL;
bErrorGeneral        : BOOL;
bErrorNoHotWater     : BOOL;
```

```
bErrorDampers       : BOOL;
byError             : BYTE;
bInvalidParameter   : BOOL;
```

eModeSeqCtrl: Enum, das die Betriebsanforderungen mitteilt.

bPreRinse: TRUE, wenn Vorspülvorgang aktiv.

bDampersOpen: Bei TRUE werden die Außen- und die Fortluftklappen geöffnet.

bSupplyVentilation: Bei TRUE Zuluftventilator einschalten.

bExhaustVentilation: Bei TRUE Abluftventilator einschalten.

bStartPIDControllers: Freigabe der Regelung.

bEnableLimitCtrl: Nach dem Ablauf des Timers *tDelayLimitCtrl*wird der Ausgang *bEnableLimitCtrl*frei gegeben und damit ist die Grenzwertüberwachung der Analogeingänge aktiviert.

bErrorGeneral: Es liegt allgemein ein Fehler an.

bErrorNoHotWater: Wird auf TRUE gesetzt, wenn der Spülvorgang beendet wurde und die eingestellte Temperatur im Rücklauf nicht erreicht wurde.

bErrorDampers: Es liegt ein Fehler bei den Klappen an.

byError: Ausgabe der Fehler als Byte.

byError.1:= bInvalidParameter;
byError.2:= bErrorGeneral;
byError.3:= bErrorNoHotWater;
byError.4:= bErrorDampers;

bInvalidParameter: Zeigt an, dass ein falscher Eingangsparameter anliegt. *bInvalidParameter* muss mit *bReset* quittiert werden.

VAR_IN_OUT

```
eConvection           : E_HVACConvectionMode;
tDelaySupplyVent      : TIME;
tDelayExhaustVent     : TIME;
tDelayTempCtrl        : TIME;
rDifferenceTemp       : REAL;
tDelayLimitCtrl       : TIME;
rPreRinseOutsideTemp  : REAL;
rPreRinseLimitTemp    : REAL;
tPreRinseMaxTime      : TIME;
iCurrentSequence      : INT;
```

eConvection: ENUM, über das man die Sommernachtkühlung als Zwangsbelüftung mit Ventilatoren oder als Konvektionsbelüftung ohne Ventilatoren erfolgen kann.

TYPE E_HVACConvectionMode :
(
eHVACConvectionMode_WithFan := 0,
eHVACConvectionMode_WithoutFan := 1
);
END_TYPE

Die Variable wird persistent gespeichert. Voreingestellt auf 1.

tDelaySupplyVent: Zeitverzögerung, die den Anlauf der Zuluftventilatoren verzögert. Die Variable wird persistent gespeichert. Voreingestellt auf 120s.

tDelayExhaustVent: Zeitverzögerung, die den Anlauf der Abluftventilatoren verzögert. Die Variable wird persistent gespeichert. Voreingestellt auf 5s.

tDelayTempCtrl: Zeitverzögerung für die Regelung. Die Variable wird persistent gespeichert. Voreingestellt auf 10s.

rDifferenceTemp: Ist die Differenz zw. der Raumtemperatur und der Zulufttemperatur kleiner als *rDifferenceTemp* wird die Regelung freigegeben. Die Variable wird persistent gespeichert. Voreingestellt auf 5K.

tDelayLimitCtrl: Zeitverzögerung, die die Grenzüberwachung der Temperaturfühler frei gibt. Siehe hierzu im FB_HVACTemperature die Variable bEnableLimitCtrl. Die Variable wird persistent gespeichert. Voreingestellt auf 360s.

rPreRinseOutsideTemp: Außentemperatur unterhalb dessen die RLT-Anlage mit dem Vorspülen des Heizregisters starten soll. Die Variable wird persistent gespeichert. Voreingestellt auf 10K.

rPreRinseLimitTemp: Rücklauftemperatur am Heizregister die erreicht sein muss, um den Vorspülgang abzuschließen um mit dem Öffnen der Klappen fortzufahren. Die Variable wird persistent gespeichert. Voreingestellt auf 30K.

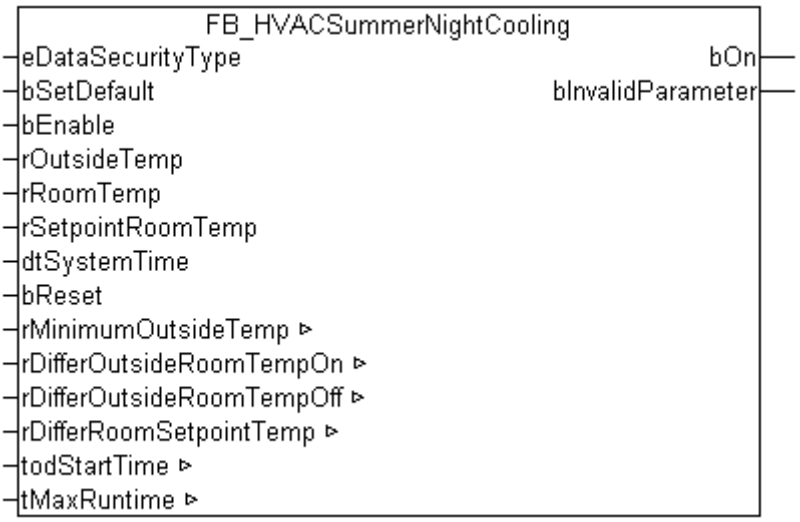
tPreRinseMaxTime: Maximale Zeit des Vorspülvorgangs. Die Variable wird persistent gespeichert. Voreingestellt auf 300s.

iCurrentSequence: Hier wird der Startregler bei der Reglerfreigabe festgelegt. Der Wert wird von dem Startprogramm nur in einem Zyklus beschrieben, da dieser Parameter danach im Anlagenbetrieb für das Ein- bzw. Ausschalten der Sequenzregler wieder frei beschreibbar sein muss. Die Variable wird persistent gespeichert.

Dokumente hierzu

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

3.6.31 FB_HVACSummerNightCooling



Anwendung

Dieser Funktionsbaustein wird verwendet um nachts mit kühler Außenluft die Tags zuvor aufgeheizten Räume herunter zu kühlen. Die Funktion der Sommernachtkühlung dient zur Verbesserung der Luftqualität und Einsparung von elektrischer Energie. In den ersten Stunden des nächsten Sommertages wird elektrische Energie zur Kälteerzeugung gespart.

Durch Parametrierung des Funktionsbausteins **FB_HVACSummerNightCooling** werden die Startbedingungen für die Sommernachtkühlung definiert. Der Baustein kann verwendet werden, um motorisch betätigte Fenster zu öffnen oder Klimaanlagen außerhalb ihrer normalen Betriebszeiten in den Sommernachtkühlbetrieb zu schalten. Der Sommernachtbetrieb ist aktiv, wenn die Ausgangsvariable *bOn* TRUE ist.

Folgende Bedingungen müssen für die Aktivierung der Sommernachtkühlung erfüllt sein:
- *bEnable* = TRUE
- *rOutsideTemp* > (*rMinimumOutsideTemp* + 0.2K)
- ((rRoomTemp - rOutsideTemp) > rDifferOutsideRoomTempOn)
- ((rRoomTemp - rSetpointRoomTemp) > rDifferRoomSetpointTemp)
- die Systemzeit *dtSystemtime* muß in dem Zeitfenster *todStartTime* bis 12:00 Uhr mittags liegen

Es reicht für die Deaktivierung der Sommernachtkühlung *bOn* = FALSE aus, wenn eine der folgenden Bedingung erfüllt ist:
- *bEnable* = FALSE
- (*rOutsideTemp* < (*rMinimumOutsideTemp* - 0.2K))
- ((*rRoomTemp* - *rOutsideTemp*) < *rDifferOutsideRoomTempOff*)
- die Systemzeit *dtSystemtime* liegt ausserhalb *des* Zeitfensters von *todStartTime* bis 12:00 Uhr mittags
- die Sommernachtkühlung war für die maximale Zeitangabe *tMaxRuntime* innerhalb des Zeitfensters von *todStartTime* bis 12:00 Uhr mittags aktiviert gewesen. Die Sommernachtkühlung kann innerhalb dieses Zeitfensters mehrmals eingeschaltet werden.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
bEnable          : BOOL;
rOutsideTemp     : REAL;
rRoomTemp        : REAL;
rSetpointRoomTemp: REAL;
dtSystemTime     : DT;
bReset           : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Mit der Eingangsvariablen *bEnable*wird der Baustein vom SPS-Programm frei gegeben.

rOutsideTemp: Eingang für die Außentemperatur.

rRoomTemp: Eingang für die Raumtemperatur.

rSetpointRoomTemp: Sollwert für die Raumtemperatur

dtSystemTime: Mit dieser Variablen wird dem Funktionsbaustein die Rechnersystemzeit übergeben.

bReset: Quittierungseingang bei einer Störung.

VAR_OUTPUT

```
bOn              : BOOL;
bInvalidParameter: BOOL;
```

bOn: Ist *bOn* = TRUE, so ist die Sommernachtkühlung aktiviert.
Folgende Bedingungen müssen für die Aktivierung der Sommernachtkühlung erfüllt sein:
bEnable = TRUEAND*rOutsideTemp* > (*rMinimumOutsideTemp* + 0.2K) AND ((rRoomTemp - rOutsideTemp) > rDifferOutsideRoomTempOn) AND
((rRoomTemp - rSetpointRoomTemp) > rDifferRoomSetpointTemp) AND die Systemzeit *dtSystemtime* muss in dem Zeitfenster *todStartTime* bis 12:00 Uhr mittags liegen.

Es reicht für die Deaktivierung der Sommernachtkühlung *bOn* = FALSE aus, wenn eine der folgenden Bedingung erfüllt ist:
bEnable = FALSEOR (*rOutsideTemp* < (*rMinimumOutsideTemp* - 0.2K)) OR ((*rRoomTemp* - *rOutsideTemp*) < *rDifferOutsideRoomTempOff*) OR die Systemzeit *dtSystemtime* liegt außerhalb *des* Zeitfensters von *todStartTime* bis 12:00 Uhr mittags OR die Sommernachtkühlung war für die maximale Zeitangabe *tMaxRuntime* innerhalb des Zeitfensters von *todStartTime* bis 12:00 Uhr mittags aktiviert gewesen. Die Sommernachtkühlung kann innerhalb dieses Zeitfensters mehrmals eingeschaltet werden.

bInvalidParameter: Zeigt an, dass ein falscher Eingangsparameter anliegt. *bInvalidParameter* muss mit *bReset* quittiert werden.

VAR_IN_OUT

```
rMinimumOutsideTemp        : REAL;
rDifferOutsideRoomTempOn   : REAL;
rDifferOutsideRoomTempOff  : REAL;
rDifferRoomSetpointTemp    : REAL;
todStartTime               : TOD;
tMaxRuntime                : TIME;
```

rMinimumOutsideTemp: Unterschreitet die Außentemperatur *rOutsideTemp* den Grenzwert *(rMinimumOutsideTemp* - 0.2K), so ist die Funktion Sommernachtkühlung *bOn* = FALSE deaktiviert. Überschreitet die Außentemperatur *rOutsideTemp* den Grenzwert (*rMinimumOutsideTemp* + 0.2K), so ist eine der Bedingungen zur Aktivierung der Sommernachtkühlung *bOn* = TRUE erfüllt (4..100). Die Variable wird persistent gespeichert. Voreingestellt auf 10°C.

rMinimumOutsideTemp muss innerhalb seines Bereiches liegen.
Liegt ein nicht korrekter Variablenwert an dann wird, wenn vorhanden, der letzte gültige Variablenwert genommen. Wenn kein gültiger letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

rDifferOutsideRoomTempOn: Die Differenz *rRoomTemp* - *rOutsideTemp* muss größer sein als der Betrag von *rDifferOutsideRoomTempOn* damit eine der Bedingungen zur Aktivierung der Sommernachtkühlung *bOn* = TRUE erfüllt ist (0..100).
((rRoomTemp - rOutsideTemp) > rDifferOutsideRoomTempOn)

rDifferOutsideRoomTempOn muss um 0.4K größer sein als *rDifferOutsideRoomTempOff.* Außerdem muss *rDifferOutsideRoomTempOn* innerhalb seines Bereiches liegen.
Liegt ein nicht korrekter Variablenwert an dann wird, wenn vorhanden, der letzte gültige Variablenwert genommen. Wenn kein gültiger letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

Die Variable wird persistent gespeichert. Voreingestellt auf 5°C.

rDifferOutsideRoomTempOff: Die Differenz *rRoomTemp* - *rOutsideTemp* muss kleiner sein als der Betrag von *rDifferOutsideRoomTempOff* um die Sommernachtkühlung *bOn* = FALSE zu deaktivieren.
((rRoomTemp - rOutsideTemp) < rDifferOutsideRoomTempOff)

rDifferOutsideRoomTempOff muss um 0.4K kleiner sein als *rDifferOutsideRoomTempOn.* Außerdem muss *rDifferOutsideRoomTempOff* innerhalb seines Bereiches liegen (0..100).
Liegt ein nicht korrekter Variablenwert an dann wird, wenn vorhanden, der letzte gültige Variablenwert genommen. Wenn kein gültiger letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt. Die Variable wird persistent gespeichert. Voreingestellt auf 2°C.

rDifferRoomSetpointTemp: Um diesen Betrag muss die Raumtemperatur größer sein als der Raumtemperatursollwert damit eine der Bedingungen zur Aktivierung Sommernachtkühlung *bOn* = TRUE erfüllt ist.
((*rRoomTemp* - *rSetpointRoomTemp*) > *rDifferRoomSetpointTemp*)

rDifferRoomSetpointTemp muss innerhalb seines Bereiches liegen (0..100).
Liegt ein nicht korrekter Variablenwert an dann wird, wenn vorhanden, der letzte gültige Variablenwert genommen. Wenn kein gültiger letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt.

Die Variable wird persistent gespeichert. Voreingestellt auf 2°C.

todStartTime: Startzeit für das Zeitfenster innerhalb dessen die Sommernachtkühlung aktiviert werden kann (0..24). Das Zeitfenster für die Freigabe der Sommernachtkühlung fängt mit der Startzeit *todStartTime* an und endet um 12:00 Uhr mittags.

todStartTime muss innerhalb seines Bereiches liegen.
Liegt ein nicht korrekter Variablenwert an dann wird, wenn vorhanden, der letzte gültige Variablenwert genommen. Wenn kein gültiger letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt. Die Variable wird persistent gespeichert. Voreingestellt auf 2Uhr.

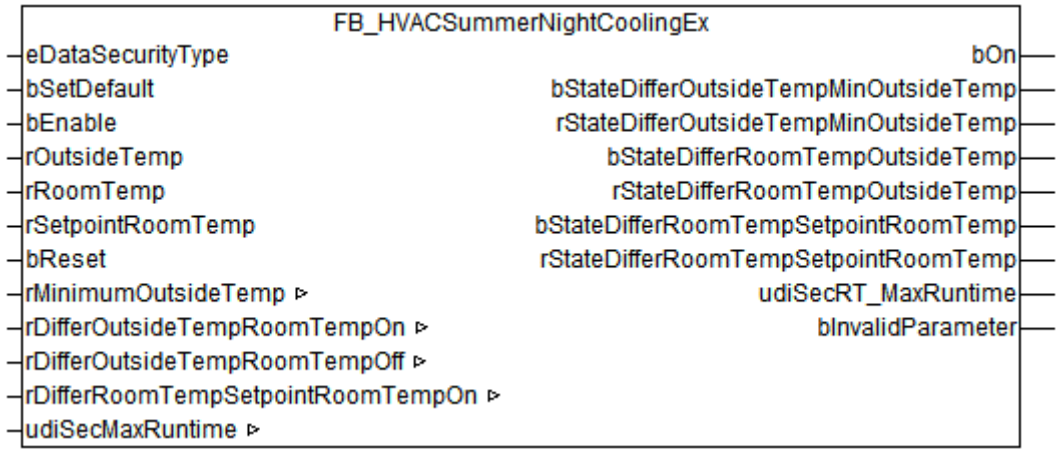
tMaxRuntime: Maximale Laufzeit der Funktion Sommernachtkühlung innerhalb des Zeitfensters *todStartTime* bis 12:00 Uhr mittags (>0s). Die Sommernachtkühlung kann mehrmals in diesem Zeitfenster eingeschaltet werden aber insgesamt nur für die Zeitangabe von *tMaxRuntime*.

tMaxRuntime muss größer als T#0s sein.
Liegt ein nicht korrekter Variablenwert an dann wird, wenn vorhanden, der letzte gültige Variablenwert genommen. Wenn kein gültiger letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt. Die Variable wird persistent gespeichert. Voreingestellt auf 20min.

Dokumente hierzu

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

3.6.32 FB_HVACSummerNightCoolingEx



Anwendung

Dieser Funktionsbaustein wird verwendet um nachts mit kühler Außenluft die Tags zuvor aufgeheizten Räume herunter zu kühlen. Die Funktion der Sommernachtkühlung dient zur Verbesserung der Luftqualität und Einsparung von elektrischer Energie. In den ersten Stunden des nächsten Sommertages wird elektrische Energie zur Kälteerzeugung gespart.

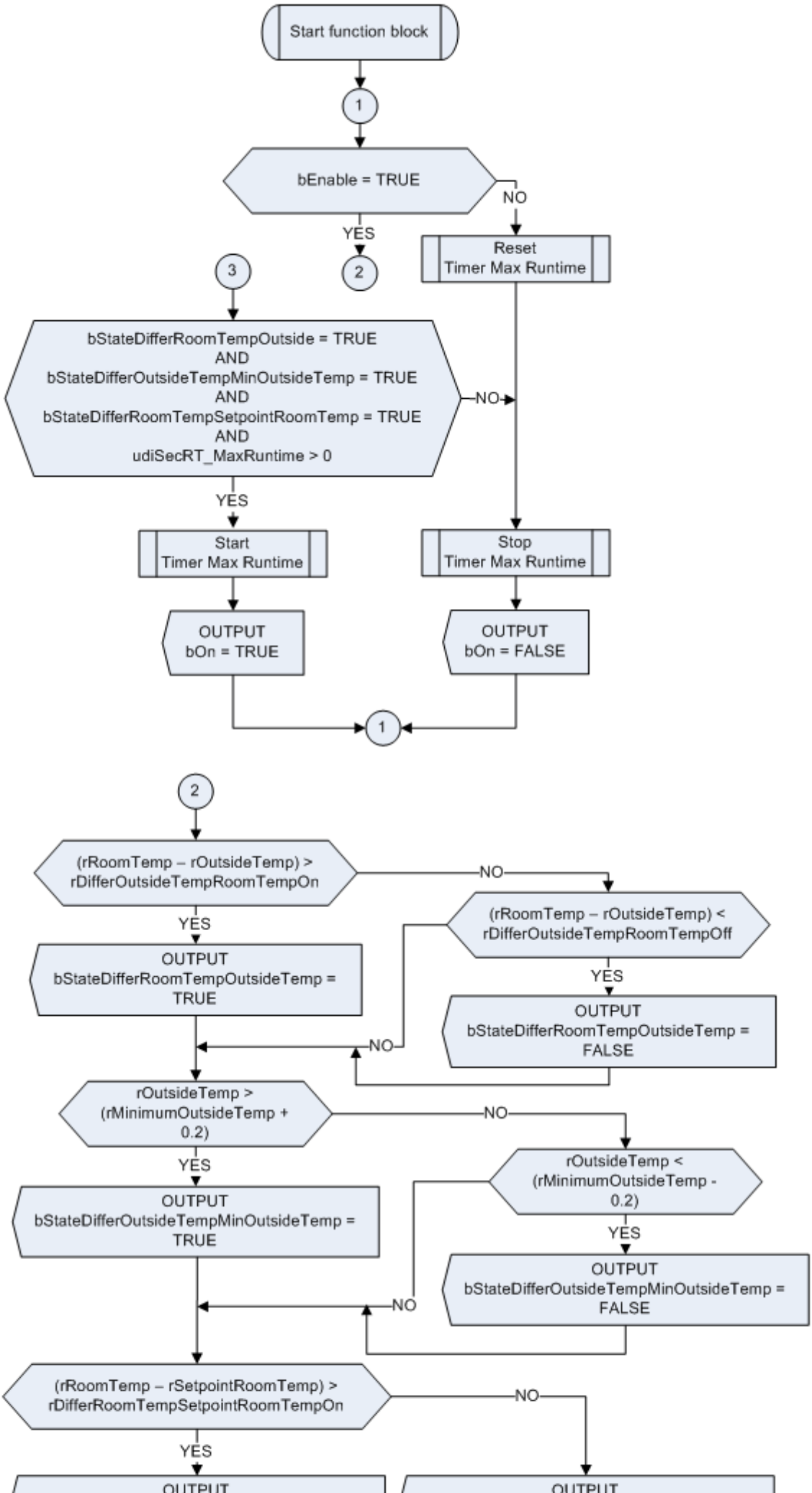
Durch Parametrierung des Funktionsbausteins **FB_HVACSummerNightCooling** werden die Startbedingungen für die Sommernachtkühlung definiert. Der Baustein kann verwendet werden, um motorisch betätigte Fenster zu öffnen oder Klimaanlagen außerhalb ihrer normalen Betriebszeiten in den Sommernachtkühlbetrieb zu schalten. Der Sommernachtbetrieb ist aktiv, wenn die Ausgangsvariable *bOn* TRUE ist.

Folgende Bedingungen müssen für die Aktivierung der Sommernachtkühlung erfüllt sein:
- *bEnable* = TRUE
- *rOutsideTemp* > (*rMinimumOutsideTemp* + 0.2K)*;* der Wert von 0.2K ist eine interne Konstante des Funktionsbaustein.
- (rRoomTemp - rOutsideTemp) > rDifferOutsideTempRoomTempOn
-(rRoomTemp - rSetpointRoomTemp) > rDifferRoomTempSetpointRoomTempOn
- *udiSecRT_MaxRuntime* > 0

Es reicht für die Deaktivierung der Sommernachtkühlung *bOn* = FALSE aus, wenn eine der folgenden Bedingung erfüllt ist:
- *bEnable* = FALSE
- *rOutsideTemp* < (*rMinimumOutsideTemp* - 0.2K); der Wert von 0.2K ist eine interne Konstante des Funktionsbaustein.
- (*rRoomTemp* - *rOutsideTemp*) < *rDifferOutsideTempRoomTempOff*
-*udiSecRT_MaxRuntime* = 0

AchtungDie Sommernachtkühlung kann innerhalb der Zeitangabe udiSecMaxRuntime mehrmals ein- und ausgeschaltet werden. Nach Ablauf von udiSecRT_MaxRuntime kann die Funktion Sommernachtkühlung nur dann wieder aktiviert werden, wenn bEnable mindestens für einen SPS-Zyklus FALSE ist. Die Freigabe des Funktionsbausteins bzw. die Funktion der Sommernachtkühlung kann mittels einer Zeitschaltuhr realisiert werden.

Programmablaufplan


Start function block
1
bEnable = TRUE
NO
YES
Reset
Timer Max Runtime
3
2
bStateDifferRoomTempOutside = TRUE
AND
bStateDifferOutsideTempMinOutsideTemp = TRUE
AND
bStateDifferRoomTempSetpointRoomTemp = TRUE
AND
udiSecRT_MaxRuntime > 0
NO
YES
Start
Timer Max Runtime
Stop
Timer Max Runtime
OUTPUT
bOn = TRUE
OUTPUT
bOn = FALSE
1
2
(rRoomTemp – rOutsideTemp) >
rDifferOutsideTempRoomTempOn
NO
YES
(rRoomTemp – rOutsideTemp) <
rDifferOutsideTempRoomTempOff
YES
OUTPUT
bStateDifferRoomTempOutsideTemp =
TRUE
OUTPUT
bStateDifferRoomTempOutsideTemp =
FALSE
NO
rOutsideTemp >
(rMinimumOutsideTemp +
0.2)
NO
YES
rOutsideTemp <
(rMinimumOutsideTemp -
0.2)
YES
OUTPUT
bStateDifferOutsideTempMinOutsideTemp =
TRUE
OUTPUT
bStateDifferOutsideTempMinOutsideTemp =
FALSE
NO
(rRoomTemp – rSetpointRoomTemp) >
rDifferRoomTempSetpointRoomTempOn
NO
YES

Anwendungsbeispiel

Das Anwendungsbeispiel zeigt den Funktionsbaustein FB_HVACSummerNightCoolingEx in Verbindung mit der Wochenzeitschaltuhr FB_HVACScheduler1ch. Das Beispiel liegt in den Programmiersprachen ST und CFC vor. Das Programmbeispiel **P_CFC_SummernightCoolingEx.PRG** in der Programmiersprache CFC befindet sich im Ordner **Language CFC** > **SpecialFunctions**, das Programmbeispiel **P_ST_SummernightCoolingEx.PRG** in der Programmiersprache ST liegt im Ordner **Language Structure Text** > **SpecialFunctions**.

Download	Benötigte Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
bEnable          : BOOL;
rOutsideTemp     : REAL;
rRoomTemp        : REAL;
rSetpointRoomTemp: REAL;
bReset           : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Ansonsten wird der instanziierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

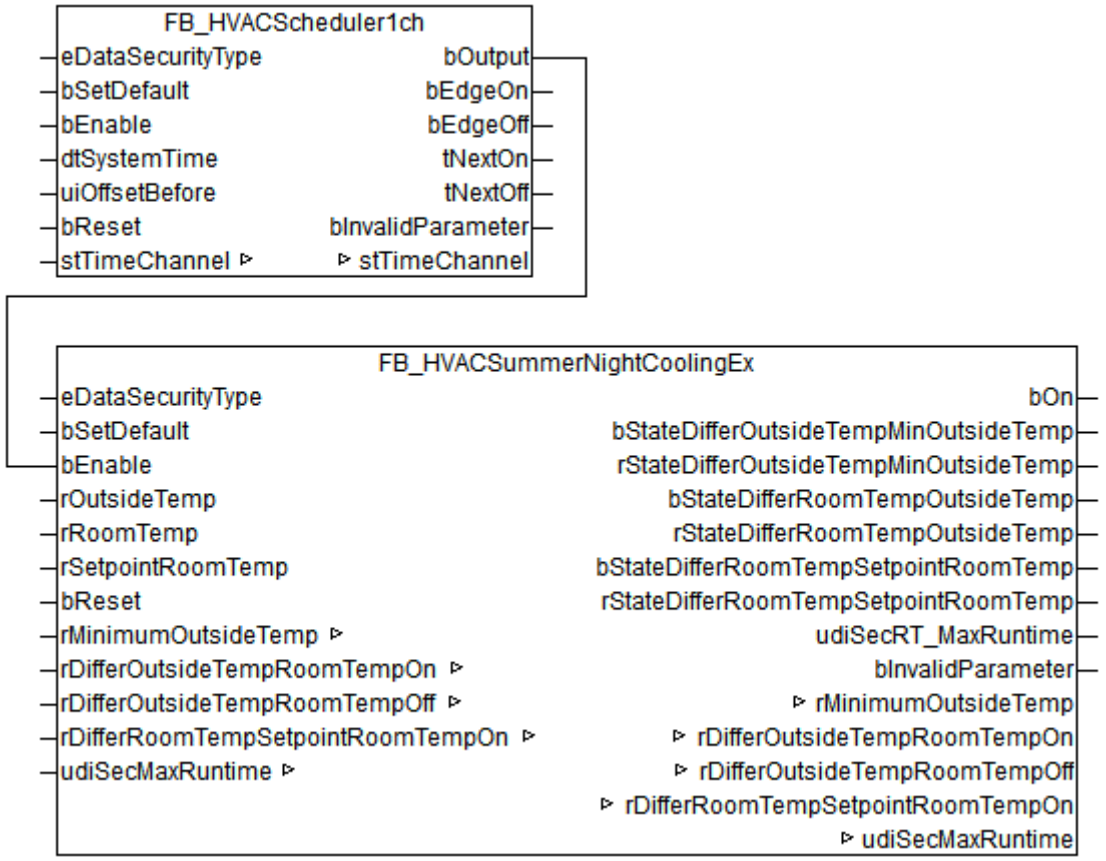
HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Mit der Eingangsvariablen *bEnable*wird der Baustein vom SPS-Programm frei gegeben. Nach Ablauf der Zeit *udiSecMaxRuntime* kann die Funktion Sommernachtauskühlung nur dann wieder aktiviert werden, wenn *bEnable* mindestens für einen SPS-Zyklus FALSE ist. Die Freigabe des Funktionsbausteins

könnte über eine Zeitschaltuhr realisiert werden.



rOutsideTemp: Eingang für die Außentemperatur.

rRoomTemp: Eingang für die Raumtemperatur

rSetpointRoomTemp: Sollwert für die Raumtemperatur

bReset: Quittierungseingang bei einer Störung.

VAR_OUTPUT

```
bOn                                 : BOOL;
bStateDifferOutsideTempMinOutsideTemp: BOOL;
rStateDifferOutsideTempMinOutsideTemp: REAL;
bStateDifferRoomTempOutsideTemp      : BOOL;
rStateDifferRoomTempOutsideTemp      : REAL;
bStateDifferRoomTempSetpointRoomTemp : BOOL;
rStateDifferRoomTempSetpointRoomTemp : REAL;
udiSecRT_MaxRuntime                  : UDINT;
bInvalidParameter                    : BOOL;
```

bOn: Ist *bOn* = TRUE, so ist die Sommernachtkühlung aktiviert.

Folgende Bedingungen müssen für die Aktivierung der Sommernachtkühlung erfüllt sein:
- *bEnable* = TRUE
- *rOutsideTemp* > (*rMinimumOutsideTemp* + 0.2K)*;* der Wert von 0.2K ist eine interne Konstante des Funktionsbaustein.
- (rRoomTemp - rOutsideTemp) > rDifferOutsideTempRoomTempOn
-(rRoomTemp - rSetpointRoomTemp) > rDifferRoomTempSetpointRoomTempOn
- *udiSecRT_MaxRuntime* > 0

Es reicht für die Deaktivierung der Sommernachtkühlung *bOn* = FALSE aus, wenn eine der folgenden Bedingung erfüllt ist:
- *bEnable* = FALSE
- *rOutsideTemp* < (*rMinimumOutsideTemp* - 0.2K); der Wert von 0.2K ist eine interne Konstante des

Funktionsbaustein.
- (*rRoomTemp* - *rOutsideTemp*) < *rDifferOutsideTempRoomTempOff*
-*udiSecRT_MaxRuntime* = 0

bStateDifferOutsideTempMinOutsideTemp: Statusanzeige. *bStateDifferOutsideTempMinOutsideTemp* ist TRUE, wenn *rOutsideTemp* > (*rMinimumOutsideTemp* + 0.2) ist. *bStateDifferOutsideTempMinOutsideTemp* ist FALSE, wenn *rOutsideTemp* < (*rMinimumOutsideTemp* - 0.2) ist.

rStateDifferOutsideTempMinOutsideTemp: Wert der Differenz *rOutsideTemp* - *rMinimumOutsideTemp*

bStateDifferRoomTempOutsideTemp: Statusanzeige. *bStateDifferRoomTempOutsideTemp*ist TRUE, wenn (*rRoomTemp* - *rOutsideTemp*) > *rDifferOutsideTempRoomTempOn* ist. *bStateDifferRoomTempOutsideTemp*ist FALSE, wenn (*rRoomTemp* - *rOutsideTemp*) < *rDifferOutsideTempRoomTempOff* ist.

rStateDifferRoomTempOutsideTemp: Wert der Differenz *rRoomTemp* - *rOutsideTemp*

bStateDifferRoomTempSetpointRoomTemp: Statusanzeige. *bStateDifferRoomTempSetpointRoomTemp* ist TRUE, wenn (*rRoomTemp* - *rSetpointRoomTemp*) > *rDifferRoomTempSetpointRoomTempOn* ist. *bStateDifferRoomTempSetpointRoomTemp* ist FALSE, wenn (*rRoomTemp* - *rSetpointRoomTemp*) <= *rDifferRoomTempSetpointRoomTempOn* ist.

rStateDifferRoomTempSetpointRoomTemp: Wert der Differenz *rRoomTemp* - *rSetpointRoomTemp*

udiSecRT_MaxRuntime: Verbleibende Zeit der Sommernachtkühlung. Ist *udiSecRT_MaxRuntime* = 0, so kann die Funktion Sommernachtkühlung nur dann wieder aktiviert werden, wenn *bEnable* mindestens für einen SPS-Zyklus FALSE ist

bInvalidParameter: Zeigt an, dass ein falscher Eingangsparameter anliegt. *bInvalidParameter* muss mit *bReset* quittiert werden.

VAR_IN_OUT

```
rMinimumOutsideTemp               : REAL;
rDifferOutsideTempRoomTempOn      : REAL;
rDifferOutsideTempRoomTempOff     : REAL;
rDifferRoomTempSetpointRoomTempOn: REAL;
udiSecMaxRuntime                  : UDINT;
```

rMinimumOutsideTemp: Unterschreitet die Außentemperatur *rOutsideTemp* den Grenzwert *(rMinimumOutsideTemp* - 0.2K), so ist die Funktion Sommernachtkühlung *bOn* = FALSE deaktiviert. Überschreitet die Außentemperatur *rOutsideTemp* den Grenzwert (*rMinimumOutsideTemp* + 0.2K), so ist eine der Bedingungen zur Aktivierung der Sommernachtkühlung *bOn* = TRUE erfüllt (4..100).

rMinimumOutsideTemp muss innerhalb seines Bereiches liegen.
Liegt ein nicht korrekter Variablenwert an dann wird, wenn vorhanden, der letzte gültige Variablenwert genommen. Wenn kein gültiger letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt. Die Variable wird persistent gespeichert. Voreingestellt auf 10°C.

rDifferOutsideTempRoomTempOn: Die Differenz *rRoomTemp* - *rOutsideTemp* muss größer sein als der Betrag von *rDifferOutsideTempRoomTempOn* damit eine der Bedingungen zur Aktivierung der Sommernachtkühlung *bOn* = TRUE erfüllt ist (0..10).
((rRoomTemp - rOutsideTemp) > rDifferOutsideRoomTempOn)

rDifferOutsideRoomTempOn muss um 0.4K größer sein als *rDifferOutsideRoomTempOff.* Außerdem muss *rDifferOutsideRoomTempOn* innerhalb seines Bereiches liegen.
Liegt ein nicht korrekter Variablenwert an dann wird, wenn vorhanden, der letzte gültige Variablenwert genommen. Wenn kein gültiger letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt. Die Variable wird persistent gespeichert. Voreingestellt auf 5°C.

rDifferOutsideTempRoomTempOff: Die Differenz *rRoomTemp* - *rOutsideTemp* muss kleiner sein als der Betrag von *rDifferOutsideTempRoomTempOff* um die Sommernachtkühlung *bOn* = FALSE zu deaktivieren.
((rRoomTemp - rOutsideTemp) < rDifferOutsideRoomTempOff)

rDifferOutsideRoomTempOff muss um 0.4K kleiner sein als *rDifferOutsideRoomTempOn.* Außerdem muss *rDifferOutsideRoomTempOff* innerhalb seines Bereiches liegen (0..10).
Liegt ein nicht korrekter Variablenwert an dann wird, wenn vorhanden, der letzte gültige Variablenwert genommen. Wenn kein gültiger letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt. Die Variable wird persistent gespeichert. Voreingestellt auf 2°C.

rDifferRoomTempSetpointRoomTempOn: Um diesen Betrag muss die Raumtemperatur größer sein als der Raumtemperatursollwert damit eine der Bedingungen zur Aktivierung Sommernachtkühlung *bOn* = TRUE erfüllt ist (0..10).
((*rRoomTemp* - *rSetpointRoomTemp*) > *rDifferRoomSetpointTemp*)

rDifferRoomSetpointTemp muss innerhalb seines Bereiches liegen.
Liegt ein nicht korrekter Variablenwert an dann wird, wenn vorhanden, der letzte gültige Variablenwert genommen. Wenn kein gültiger letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt. Die Variable wird persistent gespeichert. Voreingestellt auf 2°C.

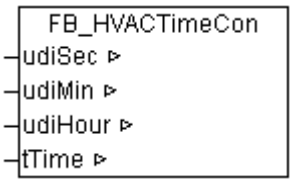
udiSecMaxRuntime: Maximale Laufzeit der Funktion Sommernachtkühlung. Die Sommernachtkühlung kann innerhalb der Zeitangabe mehrmals ein- und ausgeschaltet werden. Die verbleibende Zeit der Sommernachtkühlung wird mit der Variable *udiSecRT_MaxRuntime* angezeigt.

udiSecMaxRuntime muss größer als 0 sein.
Liegt ein nicht korrekter Variablenwert an dann wird, wenn vorhanden, der letzte gültige Variablenwert genommen. Wenn kein gültiger letzter Wert vorliegt, dann wird mit dem Default-Wert weitergearbeitet. *bInvalidParameter* wird bei falscher Parameterangabe gesetzt. Die Variable wird persistent gespeichert. Voreingestellt auf 180.

Dokumente hierzu

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

3.6.33 FB_HVACTimeCon



Anwendung

Dieser Funktionsbaustein konvertiert eine TIME-Variable *tTime* in die drei UDINT-Variablen *udiSec, udiMin* und *udiHour*, welche die Sekunden, Minuten und Stunden angeben. Diese Konvertierung ist umgekehrt auch möglich, so dass aus Sekunden, Minuten und Stunden dann eine TIME-Variable gebildet wird.

VAR_IN_OUT

```
udiSec      : UDINT;
udiMin      : UDINT;
udiHour     : UDINT;
tTime       : TIME;
```

udiSec: Variable, die die Sekunden anzeigt (0..59).

udiMin: Variable, die die Minuten anzeigt (0..59).

udiHour: Variable, die die Stunden anzeigt (0..1191).

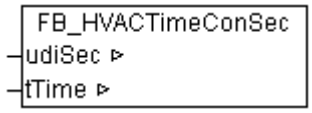
tTime: Variable vom Typ Time.

Anwendungsbeispiel

Download	Benötige Bibliothek

TcHVAC.pro [▶ 540]	**TcHVAC.lib**

3.6.34 FB_HVACTimeConSec



Anwendung

Dieser Funktionsbaustein konvertiert eine TIME-Variable *tTime* in eine UDINT-Variable *udiSec*, welche die Sekunden angibt. Diese Konvertierung ist umgekehrt auch möglich, so dass aus Sekunden dann eine TIME-Variable gebildet wird.

VAR_IN_OUT

```
udiSec      : UDINT;
tTime       : TIME;
```

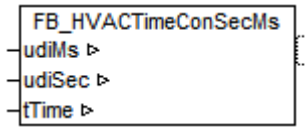
udiSec: Variable, die die Sekunden anzeigt (0..4294967)

tTime: Variable vom Typ Time

Anwendungsbeispiel

Download	**Benötige Bibliothek**
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

3.6.35 FB_HVACTimeConSecMs



Anwendung

Dieser Funktionsbaustein konvertiert eine TIME-Variable *tTime* in zwei UDINT-Variable *udiSec, udiMs*, welche die Sekunden und Millisekunden angeben. Diese Konvertierung ist umgekehrt auch möglich, so dass aus den Sekunden und Millisekunden dann eine TIME-Variable gebildet wird.

VAR_IN_OUT

```
udiMs       : UDINT;
udiSec      : UDINT;
tTime       : TIME;
```

udiMs: Variable, die die Millisekunden anzeigt (0..999).

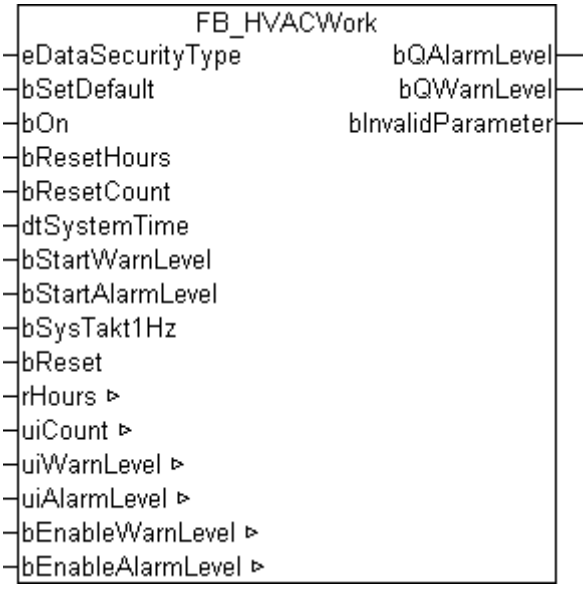
udiSec: Variable, die die Sekunden anzeigt (0..4294966).

tTime: Variable vom Typ Time.

Anwendungsbeispiel

Download	**Benötige Bibliothek**
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

3.6.36 FB_HVACWork



Anwendung

Dieser Funktionsbaustein dient zur Betriebsstundenerfassung und zur Erfassung der Einschaltzyklen. Weiterhin ist es möglich, das Erreichen von vorgegebenen Betriebszeiten zu melden.

Betriebsstundenerfassung

Sobald der Eingang *bOn* den Betrieb meldet, wird die Betriebszeit mit einer Auflösung von intern 1s erfasst. Die Ausgabe erfolgt als Realwert in Stunden mit 2 Nachkommastellen (1/100 h). Sollte keine Systemzeit mit Sekundenauflösung zur Verfügung stehen, oder diese gestört sein, so wird nach 2s auf den Takteingang *bSysTakt1Hz* umgeschaltet. Es wird auf die steigende Flanke an diesem Eingang getriggert. Die Variante mit der Systemzeit ist aufgrund der Möglichkeit der Synchronisierung über das Netzwerk mit einer Masteruhr genauer, als die Variante mit dem Taktgeber.

Jede positive Flanke an *bOn* wird als Einschaltzyklus gezählt, und steht als *iCount* am Ausgang zur Verfügung.

VAR_INPUT

```
eDataSecurityType     : E_HVACDataSecurityType;
bSetDefault           : BOOL;
bOn                   : BOOL;
bResetHours           : BOOL;
bResetCount           : BOOL;
dtSystemtime          : DT;
bStartWarnLevel       : BOOL;
bStartAlarmLevel      : BOOL;
bSysTakt1Hz           : BOOL;
bReset                : BOOL;
```

HINWEIS

Wenn E_HVACDataSecurityType := eHVACDataSecurityType_Persistent gewählt wurde, wird vom Standardverfahren abgewichen. Die Daten werden nach Änderung nicht mit einer Verzögerung entsprechend g_tHVACWriteBackupDataTime (default t#5s) in den Flashspeicher geschrieben, sondern konstant nur alle Stunde. Dies dient dem Schutz des Flash-Speichers, da sonst bei laufendem Betriebszähler alle 5sek alle persistenten Daten auf diesen geschrieben würden.

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Ansonsten wird der instanziierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen. Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bOn: Betriebsmeldung, die Betriebszeit wird aufsummiert solange die Meldung ansteht. Die steigende Flanke wird als Einschaltzyklus gezählt.

bResetHours: Steigende Flanke setzt den Betriebsstundenzähler auf 0.

bResetCount: Steigende Flanke setzt den Einschaltzyklenzähler auf 0.

dtSystemtime: Systemzeit mit Sekundenanzeige.

bStartWarnLevel: Steigende Flanke am Eingang setzt den Ausgang *bQWarnLevel* zurück und startet die Zeitmessung für diesen neu.

bStartAlarmLevel: Steigende Flanke am Eingang setzt den Ausgang *bQAlarmLevel* zurück und startet die Zeitmessung für diesen neu.

bSysTakt1Hz: Taktsignal mit 1Hz als Ersatz für *dtSystemTime*, sollte *dtSystemTime* nicht vorhanden sein, oder länger als 2sek keine Wertänderung aufweisen, wird als Ersatz das Taktsignal genutzt.

bReset: Quittierungseingang bei einer Störung. Setzt den Merker *bInvalidParameter* zurück.

VAR_IN_OUT

```
rHours            : REAL;
uiCount           : UINT;
uiWarnLevel       : UINT;
uiAlarmLevel      : UINT;
bEnableWarnLevel  : BOOL;
bEnableAlarmLevel : BOOL;
```

rHours: Betriebsstunden mit einer Auflösung von 1/100 Stunden (intern mit 1s). Die Variable wird persistent gespeichert.

uiCount: Einschaltzyklenzähler. Die Variable wird persistent gespeichert.

uiWarnLevel: Betriebsstunden nach deren Erreichen eine Warnung ausgegeben wird (0..50000). Bei steigender Flanke am Eingang *bStartWarnLevel* wird der aktuelle Betriebsstundenzählerstand mit dem Wert *uiWarnLevel* addiert und als absolute Betriebszeit gespeichert. Sobald der Betriebsstundenzählerstand diesen Wert erreicht hat und die Meldung über *bEnableWarnLevel* freigegeben ist, wird der Ausgang *bQWarnLevel* auf TRUE gesetzt. Die Variable wird persistent gespeichert. Voreingestellt auf 0h.

uiAlarmLevel: Betriebsstunden nach deren Erreichen ein Alarm ausgegeben wird (0..50000). Bei steigender Flanke am Eingang *bStartAlarmLevel* wird der aktuelle Betriebsstundenzählerstand mit dem Wert *uiAlarmLevel* addiert und als absolute Betriebszeit gespeichert. Sobald der Betriebsstundenzählerstand diesen Wert erreicht hat und die Meldung über *bEnableAlarmLevel* freigegeben ist, wird der Ausgang *bQAlarmLevel* auf TRUE gesetzt. Die Variable wird persistent gespeichert. Voreingestellt auf 0h.

bEnableWarnLevel: Gibt den Ausgang *bQWarnLevel* frei. Die Variable wird persistent gespeichert. Voreingestellt auf FALSE.

bEnableAlarmLevel: Gibt den Ausgang *bQAlarmLevel* frei. Die Variable wird persistent gespeichert. Voreingestellt auf FALSE.

VAR_OUTPUT

```
bQAlarmLevel     : BOOL;
bQWarnLevel      : BOOL;
bInvalidParameter: BOOL;
```

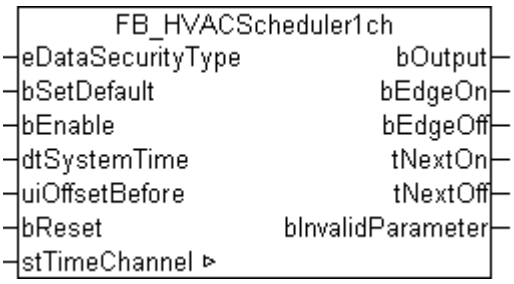
bQAlarmLevel: Betriebsstundenzähler hat den Warnwert erreicht.

bQWarnLevel: Betriebsstundenzähler hat den Alarmwert erreicht.

bInvalidParameter: Bei der Plausibilitätsüberprüfung ist ein Fehler aufgetreten. Wir durch *bReset* wieder gelöscht.

3.7 HLK Uhren

3.7.1 FB_HVACScheduler1ch



Anwendung

Dieser Funktionsbaustein ist eine Wochenzeitschaltuhr mit einem Zeitschaltkanal. Dem Zeitschaltkanal kann eine beliebige Kombination von Wochentagen zugewiesen werden. Der Zeitschaltkanal kann für einmalige oder wiederkehrende Ereignisse definiert werden. Als Ausgänge stehen neben dem Schaltsignal, die Ein- und Ausschaltflanken, sowie der nächste Ein- bzw. Ausschaltpunkt als Countdown zur Verfügung. Die Daten des Zeitschaltkanals wird als Referenz übergeben. Der Baustein ist daher in der Lage, die Daten auf Veränderung zu überprüfen und als persistente Daten zu sichern. Nach dem Booten stehen die Daten damit auch der GLT wieder zur Verfügung. Der Ein- und Ausschaltpunkt kann sekundengenau vorgegeben werden, dazu dienen die Variablen **tiOn_ss** und *tiOff_ss* des jeweiligen Zeitkanals.

Ist die Eingangsvariable bEnable = FALSE, so ist der Ausgang bOutput = FALSE. Der Zeitschaltkanal stTimeChannel ist weiterhin aktiv.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
bEnable          : BOOL;
dtSystemTime     : DT;
uiOffsetBefore   : UINT;            0 .. 240     min
bReset           : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Ansonsten wird der instanziierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen. Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Mit der Variablen **bEnable** wird der Funktionsbaustein frei gegeben. Ist *bEnable* = FALSE erfolgt keine Schaltung am Ausgang **bOutput.**

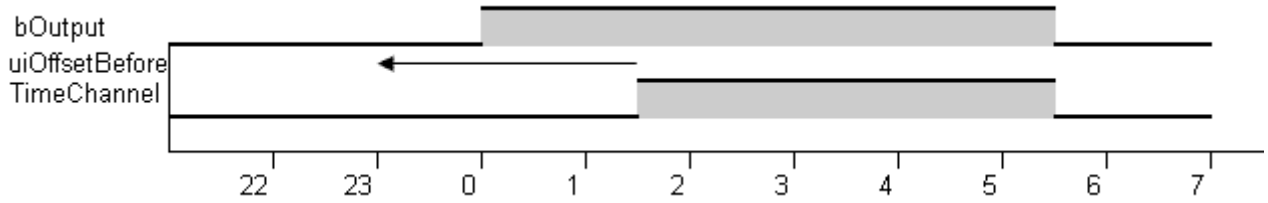
dtSystemTime: Mit der Variablen **dtSystemTime** wird dem Zeitschaltprogramm die Rechnersystemzeit übergeben.

uiOffsetBefore: Der Einschaltzeitpunkt wird um die Anzahl Minuten vorverlegt. Diese Funktionalität ist nutzbar für das zeitoptimierte Einschalten von Heizkreisen. Es können maximal 240 min, also 4h eingegeben werden. Sollte der Einschaltzeitpunkt über Mitternacht hinweg auf den Vortag fallen, so wird als Einschaltzeitpunkt Mitternacht gewählt. Der Ausschaltpunkt wird nicht beeinflusst.

bReset: Quittierungseingang bei einer Störung. Setzt den Merker *bInvalidParameter* zurück

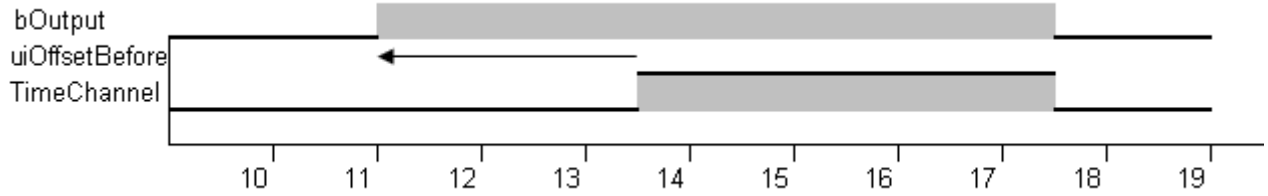
Beispiel 1:

Einschaltzeitpunkt von 1:30 Uhr bis 5:30 Uhr, *uiOffsetBefor* hat den Wert 150 min. Der Einschaltzeitpunkt wird nur um 90min auf 0:00 Uhr vorverlegt.



Beispiel 2:

Einschaltzeitpunkt von 13:30 Uhr bis 17:30 Uhr, uiOffsetBfor hat den Wert 150min. Der Einschaltzeitpunkt wird um 150min auf 11:00 Uhr vorverlegt.



VAR_IN_OUT

```
stTimeChannel     : ST_HVACTimeChannel;
```

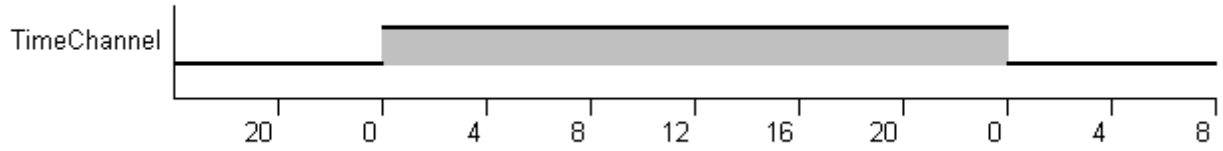
stTimeChannel: Mit der Variable *stTimeChannel* ist es möglich eine Einschaltzeit pro Wochentag zu generieren.

```
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
```

Besonderheiten bei der Uhrzeiteingabe

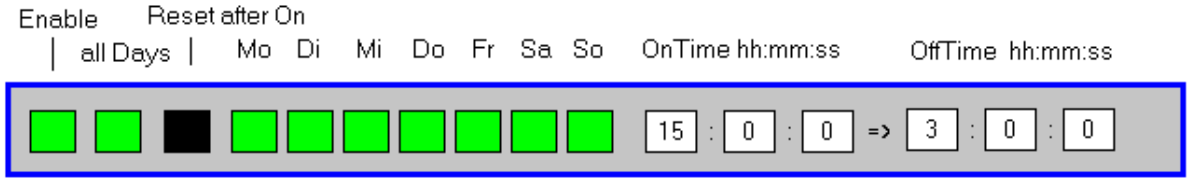
Es gilt:

- Einschaltzeit 00:00:00: Einschalten beginnt um 0 Uhr des gewählten Tages.
- Ausschaltzeit 00:00:00: Es wird um 23:59:59 Uhr des gewählten Tages ausgeschaltet.

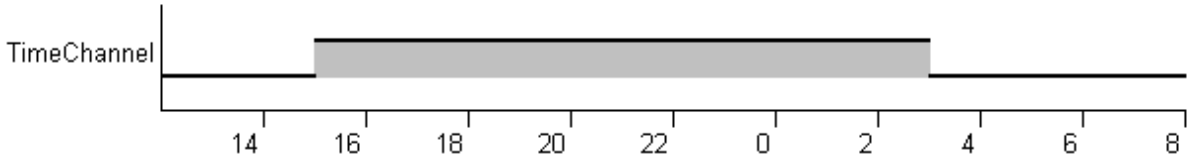


Es gilt:

Ist die Einschaltzeit im Wert größer als die Ausschaltzeit, so wird über Mitternacht bis zum nächsten Tag hinein eingeschaltet, unabhängig davon ob der nächste Tag angewählt ist.



Diese Vorgabe ergibt:



Für jede Instanz des Funktionsbausteins **FB_HVACScheduler1ch** muss eine Strukturvariable vom Typ *ST_HVACTimeChannel* erzeugt werden, die als Referenz (VAR_IN_OUT) übergeben wird.

Beispiel:

```
VAR_GLOBAL
    stTimeChannel : ST_HVACTimeChannel;
END_VAR
```

Innerhalb des Wochenplaners wird jeder Zeitschaltkanal mit der Variablen **bEnable** (Ein/Aus) frei geschaltet. Ist *bEnable* = FALSE ist der Zeitschaltkanal deaktiviert, eingetragene Zeiten sind dann ohne Wirkung. Ist *bAllDays* (Mo-So) = TRUE, dann gelten die eingestellten Zeiten von *u***iOn_hh, uiOn_mm, uiOff_hh und uiOff_mm** für alle Wochentage von Montag bis Sonntag. Wenn *bAllDays* FALSE ist, können die Wochentage mit den Variablen **bMonday** bis **bSunday** selektiv einem Zeitschaltkanal zugewiesen werden. Für einmalige Ereignisse wird die Variable *bResetAfterOn* (einmalig) aktiviert. Nach dem Ablauf der in diesem Kanal eingestellten Zeit bleiben die unter *u***iOn_hh, uiOn_mm, uiOff_hh, uiOff_mm** und den Wochentagen erstellten Einträge erhalten. Die Variable **bEnable** (Ein/Aus) wird jedoch auf FALSE zurückgesetzt. So kann ein einmaliges Ereignis einmal definiert und bei Bedarf wieder aktiviert werden.

Visualisierungsvorlage für die Target VISU mit einem Zeitschaltkanal: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659727627.zip

VAR_OUTPUT

```
bOutput          : BOOL;
bEdgeOn          : BOOL;
bEdgeOff         : BOOL;
tNextOn          : TIME;    0 .. 10080   min       (* Nächster Einschaltpunkt, max 7 Tage * 24 h * 6
0 min = 10080 *)
tNextOff         : TIME;    0 .. 1440    min       (* Nächster Ausschaltpunkt, max gleicher Tage 24
h * 60 min = 1440 *)
bInvalidParameter: BOOL;
```

bOutput: Ist gleich TRUE wenn einer der Zeitschaltkanäle eingeschaltet hat.

bEdgeOn: Ist für einen SPS-Zyklus TRUE nach dem Einschalten von bOutput.

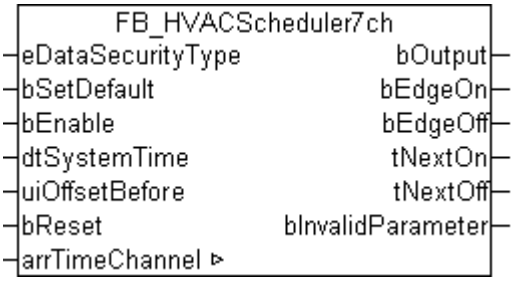
bEdgeOff: Ist für einen SPS-Zyklus TRUE nach dem Ausschalten von bOutput.

tNextOn: Zeit bis zum nächsten Einschalten des Zeitschaltprogramms. Maximalwert 10080 Minuten (1 Woche).

tNextOff: Zeit bis zum nächsten Abschalten des Zeitschaltprogramms. Maximale Voraussage bis Mitternacht .

bInvalidParameter: Bei der Plausibilitätsüberprüfung ist ein Fehler aufgetreten. Wir durch *bReset* wieder gelöscht.

3.7.2 FB_HVACScheduler7ch



Anwendung

Dieser Funktionsbaustein ist eine Wochenzeitschaltuhr mit 7 Zeitschaltkanälen. Jedem Zeitschaltkanal kann eine beliebige Kombination von Wochentagen zugewiesen werden. Die Zeitschaltkanäle können für einmalige oder wiederkehrende Ereignisse definiert werden. Als Ausgänge stehen neben dem Schaltsignal, die Ein- und Ausschaltflanken, sowie der nächste Ein- bzw. Ausschaltpunkt als Countdown zur Verfügung. Die Daten der Zeitschaltkanäle werden als Referenz übergeben. Der Baustein ist daher in der Lage, die Daten auf Veränderung zu überprüfen und als persistente Daten zu sichern. Nach dem Booten stehen die Daten damit auch der GLT wieder zur Verfügung. Der Ein- und Ausschaltpunkt kann sekundengenau vorgegeben werden, dazu dienen die Variablen *tiOn_ss* und *tiOff_ss* des jeweiligen Zeitkanals.

Ist die Eingangsvariable bEnable = FALSE, so ist der Ausgang bOutput = FALSE. Die Zeitschaltkanäle arrTimeChannel sind weiterhin aktiv.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
bEnable          : BOOL;
dtSystemTime     : DT;
uiOffsetBefore   : UINT;           0 .. 240    min
bReset           : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Mit der Variablen *bEnable*wird der Funktionsbaustein frei gegeben. Ist *bEnable* = FALSE erfolgt keine Schaltung am Ausgang *bOutput.*

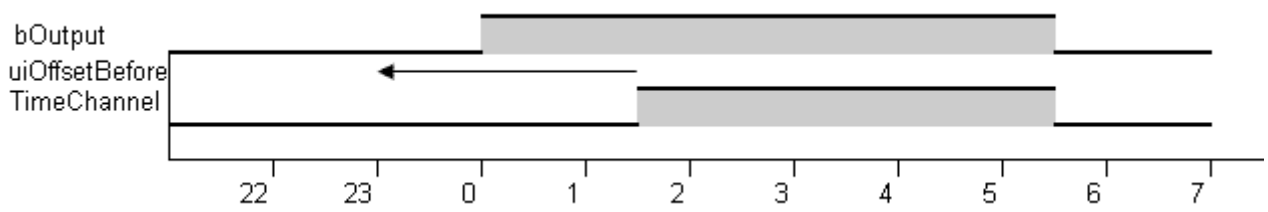
dtSystemTime: Mit der Variablen *dtSystemTime* wird dem Zeitschaltprogramm die Rechnersystemzeit übergeben.

uiOffsetBefore: Der Einschaltzeitpunkt wird um die Anzahl Minuten vorverlegt. Diese Funktionalität ist nutzbar für das zeitoptimierte Einschalten von Heizkreisen. Es können maximal 240 min, also 4h eingegeben werden. Sollte der Einschaltzeitpunkt über Mitternacht hinweg auf den Vortag fallen, so wird als Einschaltzeitpunkt Mitternacht gewählt. Der Ausschaltpunkt wird nicht beeinflusst.

bReset: Quittierungseingang bei einer Störung. Setzt den Merker *bInvalidParameter* zurück.

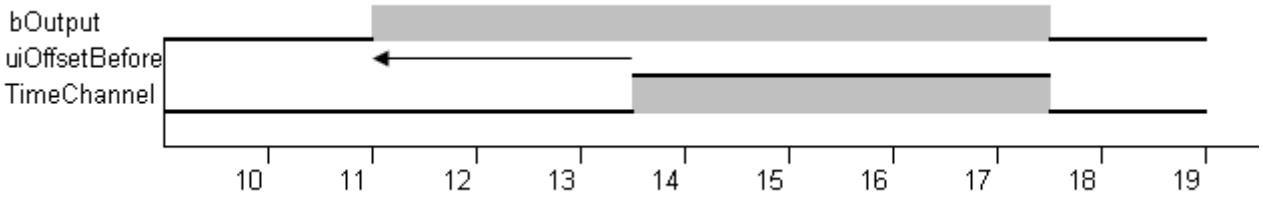
Beispiel 1:

Einschaltzeitpunkt von 1:30 Uhr bis 5:30 Uhr, uiOffsetBefore hat den Wert 150 min. Der Einschaltzeitpunkt wird nur um 90min auf 0:00 Uhr vorverlegt.



Beispiel 2:

Einschaltzeitpunkt von 13:30 Uhr bis 17:30 Uhr, uiOffsetBefore hat den Wert 150min. Der Einschaltzeitpunkt wird um 150min auf 11:00 Uhr vorverlegt.



VAR_IN_OUT

```
arrTimeChannel    : ARRAY[1..7] OF ST_HVACTimeChannel;
```

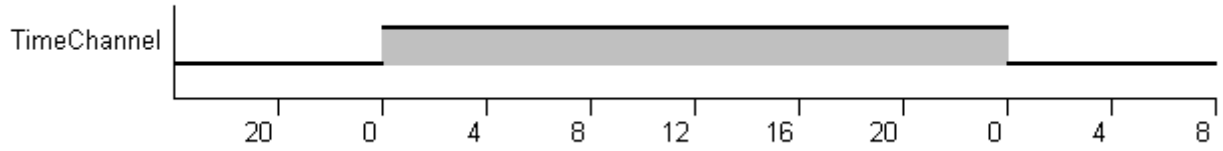
arrScheduleWeekly: Die Variable *arrScheduleWeekly* ist ein Array mit 7 Deklarationen der Strukturvariablen **ST_HVACTimeChannel**. Mit den 7 Kanälen ist es möglich, eine Einschaltzeit pro Wochentag zu generieren.

```
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
```

Besonderheiten bei der Uhrzeiteingabe

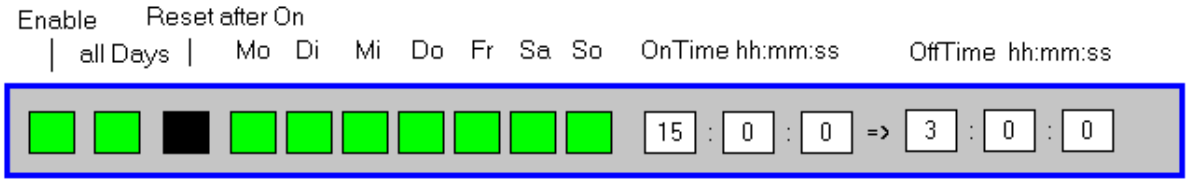
Es gilt:

- Einschaltzeit 00:00:00: Einschalten beginnt um 0 Uhr des gewählten Tags.
- Ausschaltzeit 00:00:00: Es wird um 23:59:59 Uhr des gewählten Tags ausgeschaltet.

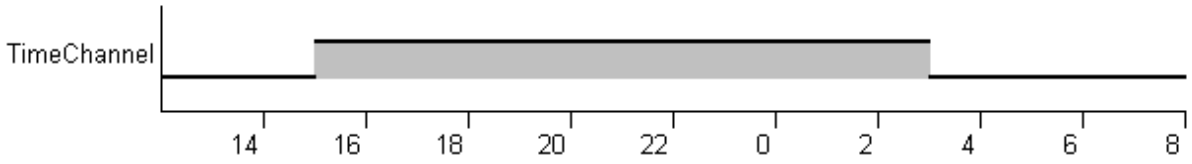


Es gilt:

Ist die Einschaltzeit im Wert größer als die Ausschaltzeit, so wird über Mitternacht bis zum nächsten Tag hinein eingeschaltet, unabhängig davon ob der nächste Tag angewählt ist.



Diese Vorgabe ergibt:



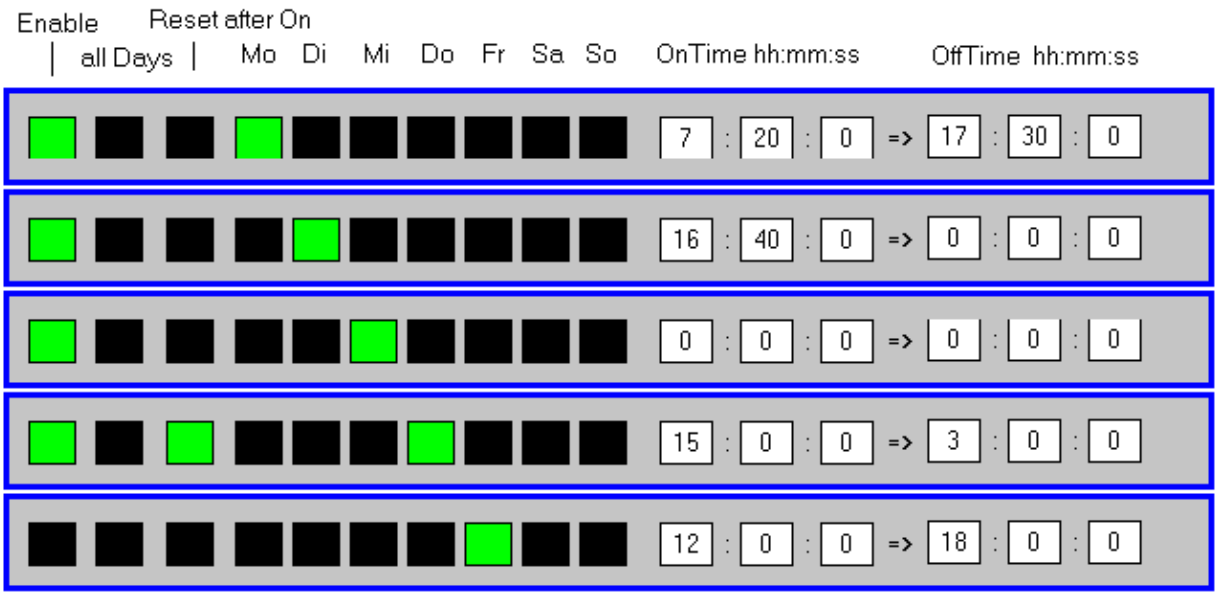
Für jede Instanz des Funktionsbausteins **FB_HVACSchedulerWeekly** muss ein Array vom Typ **ST_HVACTimeChannel** erzeugt werden. Die Größe des Arrays muss 7 sein, da der Funktionsbaustein insgesamt 7 Wochenzeitschaltzyklen verarbeitet und diese als Referenz (VAR_IN_OUT) erwartet.

Beispiel

```
VAR_GLOBAL
    arrTimeChannel : ARRAY[1..7] OF ST_HVACTimeChannel;
END_VAR
```

Innerhalb des Wochenplaners wird jeder Zeitschaltkanal mit der Variablen *bEnable* (Ein/Aus) frei geschaltet. Ist *bEnable* = FALSE ist der Zeitschaltkanal deaktiviert, eingetragene Zeiten sind dann ohne Wirkung. Ist *bAllDays* (Mo-So) = TRUE, dann gelten die eingestellten Zeiten von *uiOn_hh, uiOn_mm, uiOff_hh* und *uiOff_mm* für alle Wochentage von Montag bis Sonntag. Wenn *bAllDays* FALSE ist, können die Wochentage mit den Variablen *bMonday* bis *bSunday* selektiv einem Zeitschaltkanal zugewiesen werden. Für einmalige Ereignisse wird die Variable *bResetAfterOn* (einmalig) aktiviert. Nach dem Ablauf der in diesem Kanal eingestellten Zeit bleiben die unter *uiOn_hh, uiOn_mm, uiOff_hh, uiOff_mm*und den Wochentagen erstellten Einträge erhalten. Die Variable *bEnable* (Ein/Aus) wird jedoch auf FALSE zurückgesetzt. So kann ein einmaliges Ereignis einmal definiert und bei Bedarf wieder aktiviert werden.

Beispiel



Montag von 07:20 Uhr bis 17:30 Uhr eingeschaltet

Dienstag von 16:40 Uhr bis 00:00 Uhr eingeschaltet

Mittwoch von 00:00 Uhr bis 23:59:59 Uhr eingeschaltet (rund um die Uhr 24 Stunden!)

Donnerstag von 15:00 Uhr bis 03:00 Uhr Freitagmorgens eingeschaltet, einmalig!

Freitag Zeitschaltkanal nicht aktiviert, immer ausgeschaltet!

VAR_OUTPUT

```
bOutput           : BOOL;
bEdgeOn           : BOOL;
bEdgeOff          : BOOL;
tNextOn           : TIME;     0 .. 10080    min     (* Nächster Einschaltpunkt, max 7 Tage * 24 h * 6
0 min = 10080 *)
tNextOff          : TIME;     0 .. 1440     min     (* Nächster Ausschaltpunkt, max gleicher Tage 24
h * 60 min = 1440 *)
bInvalidParameter: BOOL;
```

bOutput: Ist gleich TRUE wenn einer der Zeitschaltkanäle eingeschaltet hat.

bEdgeOn: Ist für einen SPS-Zyklus TRUE nach dem Einschalten von bOutput.

bEdgeOff: Ist für einen SPS-Zyklus TRUE nach dem Ausschalten von bOutput.

tNextOn: Zeit bis zum nächsten Einschalten des Zeitschaltprogramms. Maximalwert 10080 Minuten (1 Woche).

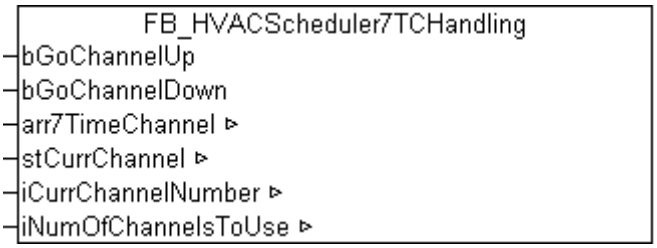
tNextOff: Zeit bis zum nächsten Abschalten des Zeitschaltprogramms. Maximale Voraussage bis Mitternacht .

bInvalidParameter: Bei der Plausibilitätsüberprüfung ist ein Fehler aufgetreten. Wir durch *bReset* wieder gelöscht.

Dokumente hierzu

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

3.7.3 FB_HVACScheduler7TCHandling



Anwendung

Mit diesem Funktionsbaustein kann eine einzelne Zeile aus dem Datenarray einer Wochenzeitschaltuhr ausgewählt und verändert werden. Somit ist es möglich die Datensätze sehr einfach auch mit einer Visualisierung, die nur eine geringe grafische Auflösung bietet, zu bearbeiten, ohne dass alle Daten des Arrays zur Anzeige gebracht werden müssen.
Der Datensatz aus **arr7TimeChannel** mit dem Index **iCurrChannelNumber** wird in Datenstruktur **stCurrChannel** kopiert. Änderungen der Daten in der Struktur **stCurrChannel** werden sofort übernommen und in das Array **arr7TimeChannel** übertragen.

VAR_INPUT

```
bGoChannelUp    : BOOL;
bGoChannelDown  : BOOL;
```

bGoChannelUp: Mit einer positiven Flanke wird der Fokus auf den nächst höheren Index im Datenarray *arr7TimeChannel* gelegt .

bGoChannelDown: Mit einer positiven Flanke wird der Fokus auf den nächst kleineren Index im Datenarray *arr7TimeChannel* gelegt .

VAR_IN_OUT

```
arr7TimeChannel    : ARRAY[1..7] OF ST_HVACTimeChannel;
stCurrChannel      : ST_HVACTimeChannel;
iCurrChannelNumber : INT;
iNumOfChannelsToUse: INT;
```

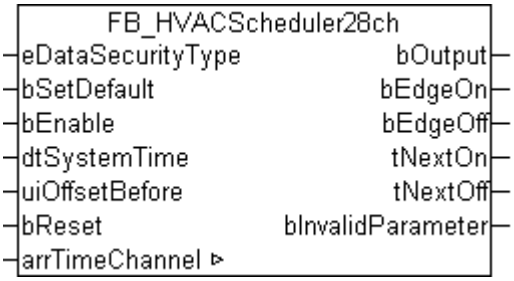
arr7TimeChannel : Array der Datensätze der Zeitschaltuhr *FB_HVACScheduler7CH*.

stCurrChannel: Enthält den Inhalt des Datensatzes aus dem Array **arr7TimeChannel** , der über die Variable *iCurrChannelNumber* selektiert wird.

iCurrChannelNumber : Index für das Array **arr7TimeChannel** .

iNumOfChannelsToUse : Maximal zulässiger Index für **iCurrChannelNumber,** für den Fall das nicht alle Kanäle benötigt werden.

3.7.4 FB_HVACScheduler28ch



Anwendung

Dieser Funktionsbaustein ist eine Wochenzeitschaltuhr mit 28 Zeitschaltkanälen. Jedem Zeitschaltkanal kann eine beliebige Kombination von Wochentagen zugewiesen werden. Die Zeitschaltkanäle können für einmalige oder wiederkehrende Ereignisse definiert werden. Als Ausgänge stehen neben dem Schaltsignal, die Ein- und Ausschaltflanken, sowie der nächste Ein- bzw. Ausschaltpunkt als Countdown zur Verfügung. Die Daten der Zeitschaltkanäle werden als Referenz übergeben. Der Baustein ist daher in der Lage, die Daten auf Veränderung zu überprüfen und als persistente Daten zu sichern. Nach dem Booten stehen die Daten damit auch der GLT wieder zur Verfügung. Der Ein- und Ausschaltpunkt kann sekundengenau vorgegeben werden, dazu dienen die Variablen *tiOn_ss* und *tiOff_ss* des jeweiligen Zeitkanals.

Ist die Eingangsvariable bEnable = FALSE, so ist der Ausgang bOutput = FALSE. Die Zeitschaltkanäle arrTimeChannel sind weiterhin aktiv.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
bEnable          : BOOL;
dtSystemTime     : DT;
uiOffsetBefore   : UINT;            0 .. 240     min
bReset           : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanziieren. Ansonsten wird der instanziierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen. Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Mit der Variablen *bEnable*wird der Funktionsbaustein frei gegeben. Ist *bEnable* = FALSE erfolgt keine Schaltung am Ausgang *bOutput.*

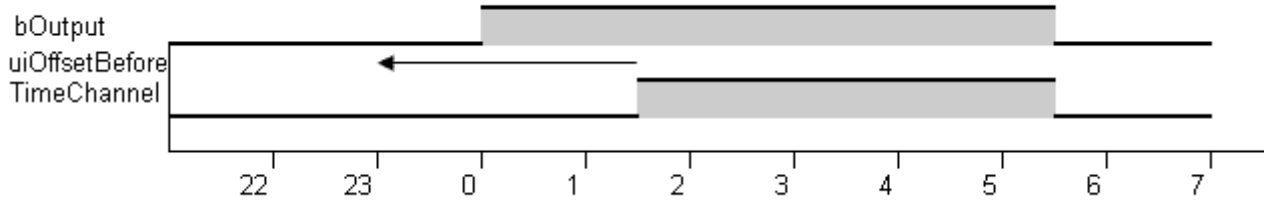
dtSystemTime: Mit der Variablen *dtSystemTime* wird dem Zeitschaltprogramm die Rechnersystemzeit übergeben.

uiOffsetBefore: Der Einschaltzeitpunkt wird um die Anzahl Minuten vorverlegt. Diese Funktionalität ist nutzbar für das zeitoptimierte Einschalten von Heizkreisen. Es können maximal 240 min, also 4h eingegeben werden. Sollte der Einschaltzeitpunkt über Mitternacht hinweg auf den Vortag fallen, so wird als Einschaltzeitpunkt Mitternacht gewählt. Der Ausschaltpunkt wird nicht beeinflusst.

bReset: Quittierungseingang bei einer Störung. Setzt den Merker *bInvalidParameter* zurück.

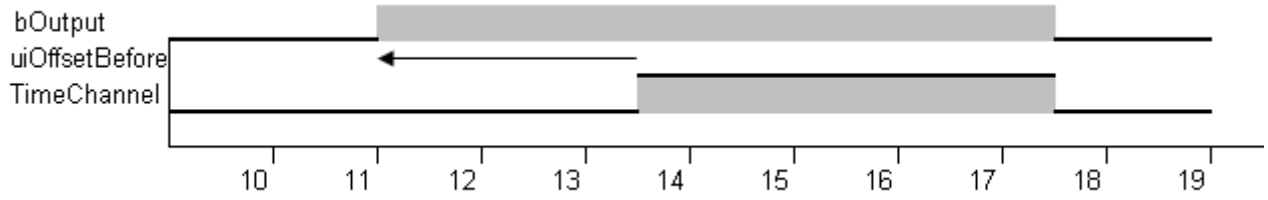
Beispiel 1

Einschaltzeitpunkt von 1:30 Uhr bis 5:30 Uhr, uiOffsetBefore hat den Wert 150 min. Der Einschaltzeitpunkt wird nur um 90min auf 0:00 Uhr vorverlegt.



Beispiel 2

Einschaltzeitpunkt von 13:30 Uhr bis 17:30 Uhr, uiOffsetBefore hat den Wert 150min. Der Einschaltzeitpunkt wird um 150min auf 11:00 Uhr vorverlegt.



VAR_IN_OUT

```
arrTimeChannel   : ARRAY[1..28] OF ST_HVACTimeChannel;
```

arrTimeChannel: Die Variable **arrTimeChannel** ist ein Array mit 28 Deklarationen der Strukturvariablen **ST_HVACTimeChannel.** Mit den 28 Kanälen ist es möglich, bis zu 3 Einschaltzeiten pro Wochentag zu generieren.

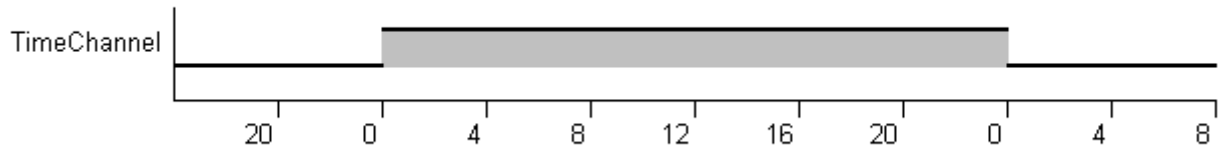
```
TYPE ST_HVACTimeChannel :
STRUCT
  uiOn_hh      : UINT;
  uiOn_mm      : UINT;
  uiOn_ss      : UINT;
```

```
  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
```

Besonderheiten bei der Uhrzeiteingabe

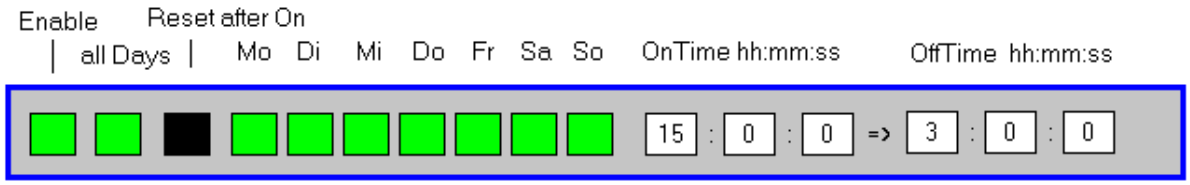
Es gilt:

- Einschaltzeit 00:00:00: Einschalten beginnt um 0 Uhr des gewählten Tags.
- Ausschaltzeit 00:00:00: Es wird um 23:59:59 Uhr des gewählten Tags ausgeschaltet.

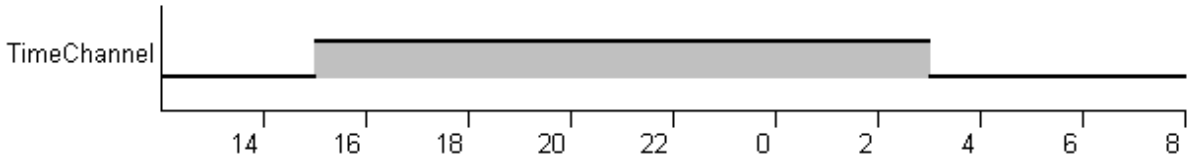


Es gilt:

Ist die Einschaltzeit im Wert größer als die Ausschaltzeit, so wird über Mitternacht bis zum nächsten Tag hinein eingeschaltet, unabhängig davon ob der nächste Tag angewählt ist.



Diese Vorgabe ergibt:



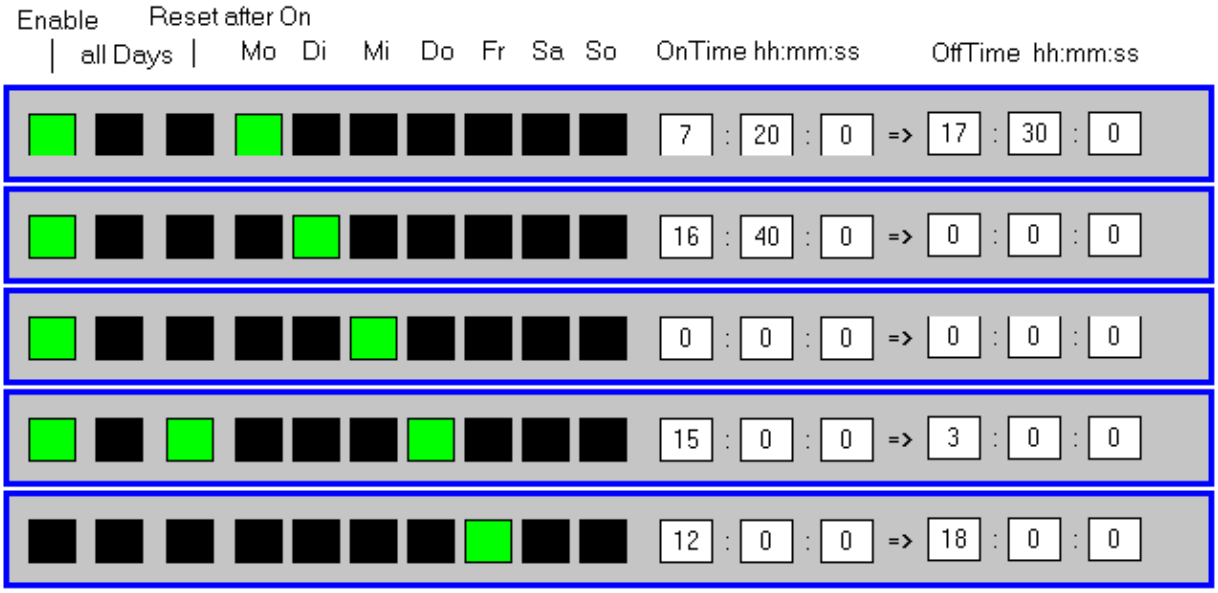
Für jede Instanz des Funktionsbausteins **FB_HVACScheduler28ch** muss ein Array vom Typ **ST_HVACTimeChannel** erzeugt werden. Die Größe des Arrays muss 28 sein, da der Funktionsbaustein insgesamt 28 Wochenzeitschaltzyklen verarbeitet und diese als Referenz (VAR_IN_OUT) erwartet.

Beispiel

```
VAR_GLOBAL
    arrTimeChannel : ARRAY[1..28] OF ST_HVACTimeChannel;
END_VAR
```

Innerhalb des Wochenplaners wird jeder Zeitschaltkanal mit der Variablen *bEnable* (Ein/Aus) frei geschaltet. Ist *bEnable* = FALSE ist der Zeitschaltkanal deaktiviert, eingetragene Zeiten sind dann ohne Wirkung. Ist *bAllDays* (Mo-So) = TRUE, dann gelten die eingestellten Zeiten von *uiOn_hh, uiOn_mm, uiOff_hh und uiOff_mm* für alle Wochentage von Montag bis Sonntag. Wenn *bAllDays* FALSE ist, können die Wochentage mit den Variablen *bMonday* bis *bSunday* selektiv einem Zeitschaltkanal zugewiesen werden. Für einmalige Ereignisse wird die Variable *bResetAfterOn* (einmalig) aktiviert. Nach dem Ablauf der in diesem Kanal eingestellten Zeit bleiben die unter *uiOn_hh, uiOn_mm, uiOff_hh, uiOff_mm*und den Wochentagen erstellten Einträge erhalten. Die Variable *bEnable* (Ein/Aus) wird jedoch auf FALSE zurückgesetzt. So kann ein einmaliges Ereignis einmal definiert und bei Bedarf wieder aktiviert werden.

Beispiel



Montag von 07:20 Uhr bis 17:30 Uhr eingeschaltet

Dienstag von 16:40 Uhr bis 00:00 Uhr eingeschaltet

Mittwoch von 00:00 Uhr bis 23:59:59 Uhr eingeschaltet (rund um die Uhr 24 Stunden!)

Donnerstag von 15:00 Uhr bis 03:00 Uhr Freitagmorgens eingeschaltet, einmalig!

Freitag Zeitschaltkanal nicht aktiviert, immer ausgeschaltet!

VAR_OUTPUT

```
bOutput          : BOOL;
bEdgeOn          : BOOL;
bEdgeOff         : BOOL;
tNextOn          : TIME;    0 .. 10080   min       (* Nächster Einschaltpunkt, max 7 Tage * 24 h * 6
0 min = 10080 *)
tNextOff         : TIME;    0 .. 1440    min       (* Nächster Ausschaltpunkt, max gleicher Tage 24
h * 60 min = 1440 *)
bInvalidParameter: BOOL;
```

bOutput: Ist gleich TRUE, wenn einer der Zeitschaltkanäle eingeschaltet hat.

bEdgeOn: Ist für einen SPS-Zyklus TRUE nach dem Einschalten von *bOutput.*

bEdgeOff: Ist für einen SPS-Zyklus TRUE nach dem Ausschalten von *bOutput*.

tNextOn: Zeit bis zum nächsten Einschalten des Zeitschaltprogramms. Maximalwert 10080 Minuten (1 Woche).

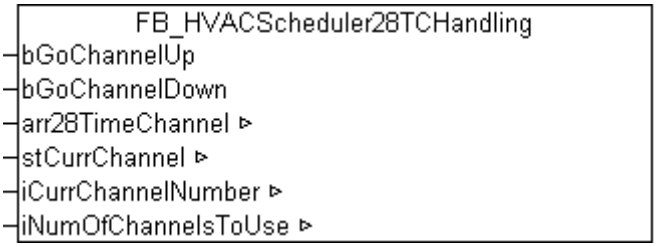
tNextOff: Zeit bis zum nächsten Abschalten des Zeitschaltprogramms. Maximale Voraussage bis Mitternacht.

bInvalidParameter: Bei der Plausibilitätsüberprüfung ist ein Fehler aufgetreten. Wir durch *bReset* wieder gelöscht.

Dokumente hierzu

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

3.7.5 FB_HVACScheduler28TCHandling



Anwendung

Mit diesem Funktionsbaustein kann eine einzelne Zeile aus dem Datenarray einer Wochenzeitschaltuhr ausgewählt und verändert werden. Somit ist es möglich die Datensätze sehr einfach auch mit einer Visualisierung, die nur eine geringe grafische Auflösung bietet, zu bearbeiten, ohne dass alle Daten des Arrays zur Anzeige gebracht werden müssen.
Der Datensatz aus **arr7TimeChannel** mit dem Index **iCurrChannelNumber** wird in Datenstruktur **stCurrChannel** kopiert. Änderungen der Daten in der Struktur **stCurrChannel** werden sofort übernommen und in das Array **arr28TimeChannel** übertragen.

VAR_INPUT

```
bGoChannelUp    : BOOL;
bGoChannelDown  : BOOL;
```

bGoChannelUp: Mit einer positiven Flanke wird der Fokus auf den nächst höheren Index im Datenarray *arr28TimeChannel* gelegt .

bGoChannelDown: Mit einer positiven Flanke wird der Fokus auf den nächst kleineren Index im Datenarray *arr28TimeChannel* gelegt .

VAR_IN_OUT

arr28TimeChannel : ARRAY[1..28]OF ST_HVACTimeChannel;
stCurrChannel : ST_HVACTimeChannel;
iCurrChannelNumber : INT;
iNumOfChannelsToUse : INT;

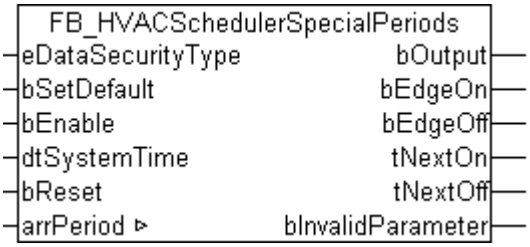
arr28TimeChannel : Array der Datensätze der Zeitschaltuhr *FB_HVACScheduler28CH*.

stCurrChannel: Enthält den Inhalt des Datensatzes aus dem Array **arr28TimeChannel** , der über die Variable *iCurrChannelNumber* selektiert wird.

iCurrChannelNumber : Index für das Array **arr28TimeChannel** .

iNumOfChannelsToUse : Maximal zulässiger Index für **iCurrChannelNumber,** falls nicht alle Kanäle genutzt werden.

3.7.6 FB_HVACSchedulerSpecialPeriods



Anwendung

Dieser Funktionsbaustein ist ein Jahreszeitschaltplaner in dem z.B. Schulferien oder Betriebsferien eingegeben werden können. Mit der Eingangsvariable *bEnable*wird der Baustein aktiviert. Der Eingang *dtSystemTime* wird mit der aktuellen Systemzeit verknüpft. Der Ausgang *bOutput* wird gesetzt, wenn die Zeitschaltbedingung erfüllt ist. Als Ausgänge stehen neben dem Schaltsignal, die Ein- und Ausschaltflanken, sowie der nächste Ein- bzw. Ausschaltpunkt als Countdown zur Verfügung. Die Daten der Zeitschaltkanäle werden als Referenz übergeben. Der Baustein ist daher in der Lage, die Daten auf Veränderung zu überprüfen und als persistente Daten zu sichern. Nach dem Booten stehen die Daten damit auch der GLT wieder zur Verfügung. Der Ein- und Ausschaltpunkt am jeweiligen Tag kann minutengenau vorgegeben werden.

Ist die Eingangsvariable bEnable = FALSE, so ist der Ausgang bOutput = FALSE. Die Zeitschaltkanäle arrPeriod sind weiterhin aktiv.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
bEnable          : BOOL;
dtSystemTime     : DT;
bReset           : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Mit der Variablen *bEnable*wird der Funktionsbaustein frei gegeben. Ist *bEnable* = FALSE erfolgt keine Schaltung am Ausgang *bOutput.*

dtSystemTime: Mit der Variablen *dtSystemTime* wird dem Zeitschaltprogramm die Rechnersystemzeit übergeben.

bReset: Quittierungseingang bei einer Störung. Setzt den Merker *bInvalidParameter* zurück.

VAR_IN_OUT

```
arrPeriod          : ARRAY[1..20] OF ST_HVACPeriod;
```

arrPeriod: Die Variable **arrPeriod** ist ein Array mit 20 Deklarationen der Strukturvariablen **ST_HVACPeriod.**

Für jede Instanz des Funktionsbausteins **FB_HVACSchedulerSpecialPeriods** muss ein Array vom Typ **ST_HVACPeriod** erzeugt werden. Die Größe des Arrays muss 20 sein, da der Funktionsbaustein insgesamt 20 Zeitkanäle verarbeitet und diese als Referenz (VAR_IN_OUT) erwartet.

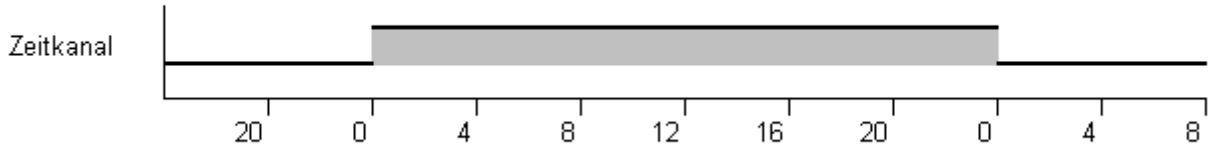
```
TYPE ST_HVACTimeChannel :
STRUCT
  uiOn_hh      : UINT;
  uiOn_mm      : UINT;
  uiOn_ss      : UINT;
  uiOff_hh     : UINT;
  uiOff_mm     : UINT;
  uiOff_ss     : UINT;
  bEnable      : BOOL;
  bAllDays     : BOOL;
  bMonday      : BOOL;
  bTuesday     : BOOL;
  bWednesday   : BOOL;
  bThursday    : BOOL;
  bFriday      : BOOL;
  bSaturday    : BOOL;
  bSunday      : BOOL;
  bResetAfterOn: BOOL;
  bQ           : BOOL;
END_STRUCT
END_TYPE
```

Besonderheiten bei der Uhrzeiteingabe

Es gilt:

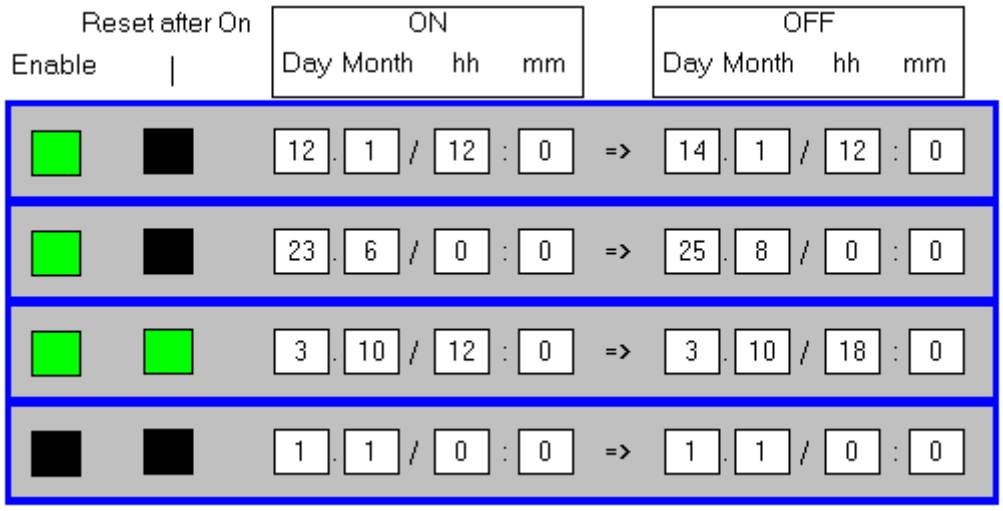
Einschaltzeit 00:00:00: Einschalten beginnt um 0 Uhr des gewählten Tags.

Ausschaltzeit 00:00:00: Es wird um 23:59:59 Uhr des gewählten Tags ausgeschaltet.



Jeder Zeitschaltkanal wird mit der Variablen *bEnable* (Ein/Aus) frei geschaltet. Ist *bEnable* = FALSE ist der Zeitschaltkanal deaktiviert, eingetragene Zeiten sind dann ohne Wirkung. Für einmalige Ereignisse wird die Variable *bResetAfterOn* (einmalig) aktiviert. Nach dem Ablauf der in diesem Kanal eingestellten Zeit bleiben die unter *uiOn_hh, uiOn_mm, uiOff_hh, uiOff_mm , uiOn_Day, uiOff_Day, uiOn_Month* und *uiOff_Month*erstellten Einträge erhalten. Die Variable *bEnable* (Ein/Aus) wird jedoch auf FALSE zurückgesetzt. So kann ein einmaliges Ereignis einmal definiert und bei Bedarf wieder aktiviert werden.

Beispiel:



12.01. von 12:20 Uhr bis 14.01. um 12:00 Uhr eingeschaltet

23.06. von 00:00 Uhr bis 25.08. um 23:59:59 Uhr eingeschaltet

03.10. von 12:00 Uhr bis 03.10. um 18:00 Uhr eingeschaltet, einmalig!

1.1. bis 1.1. Zeitschaltkanal nicht aktiviert, immer ausgeschaltet!

Visualisierungsvorlage für die Target VISU mit einem Zeitschaltkanal: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659729035.zip

VAR_OUTPUT

```
bOutput           : BOOL;
bEdgeOn           : BOOL;
bEdgeOff          : BOOL;
tNextOn           : TIME;      0 .. t#71582m47s295msmin    (* Nächster Einschaltpunkt, Range max.  =
Obergrenze von Time ca.50 Tage *)
tNextOff          : TIME;      0 .. t#71582m47s295msmin    (* Nächster Ausschaltpunkt, Range max.  =
Obergrenze von Time ca.50 Tage *)
bInvalidParameter: BOOL;
```

bOutput: Ist gleich TRUE wenn einer der Zeitschaltkanäle eingeschaltet hat.

bEdgeOn: Ist für einen SPS-Zyklus TRUE nach dem Einschalten von bOutput.

bEdgeOff: Ist für einen SPS-Zyklus TRUE nach dem Ausschalten von bOutput.

tNextOn: Zeit bis zum nächsten Einschalten des Zeitschaltprogramms.

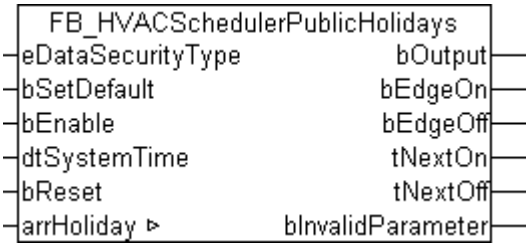
tNextOff: Zeit bis zum nächsten Abschalten des Zeitschaltprogramms.

bInvalidParameter: Bei der Plausibilitätsüberprüfung ist ein Fehler aufgetreten. Wir durch *bReset* wieder gelöscht.

Dokumente hierzu

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

3.7.7 FB_HVACSchedulerPublicHolidays



Anwendung

Dieser Funktionsbaustein ist ein Jahreszeitschaltplaner in dem z.B. Schulferien oder Betriebsferien eingegeben werden können. Mit der Eingangsvariable *bEnable*wird der Baustein aktiviert. Der Eingang *dtSystemTime* wird mit der aktuellen Systemzeit verknüpft. Der Ausgang *bOutput* wird gesetzt, wenn die Zeitschaltbedingung erfüllt ist. Als Ausgänge stehen neben dem Schaltsignal, die Ein- und Ausschaltflanken, sowie der nächste Ein- bzw. Ausschaltpunkt als Countdown zur Verfügung. Die Daten der Zeitschaltkanäle werden als Referenz übergeben. Der Baustein ist daher in der Lage die Daten auf Veränderung zu überprüfen und als persistente Daten zu sichern. Nach dem Booten stehen die Daten damit auch der GLT wieder zur Verfügung. Im Gegensatz zum **FB_HVACSchedulerSpecialPeriods** ist hier nur eine tageweise Schaltung möglich.

Ist die Eingangsvariable bEnable = FALSE, so ist der Ausgang bOutput = FALSE. Die Zeitschaltkanäle arrHoliday sind weiterhin aktiv.

VAR_INPUT

```
eDataSecurityType: E_HVACDataSecurityType;
bSetDefault      : BOOL;
bEnable          : BOOL;
dtSystemTime     : DT;
bReset           : BOOL;
```

eDataSecurityType: Wenn eDataSecurityType:= *eHVACDataSecurityType_Persistent* ist, werden die persistenten VAR_IN_OUT-Variablen des Funktionsbausteins bei einer Wertänderung im Flash des Rechners abgelegt. Dafür ist es zwingend erforderlich den Funktionsbaustein FB_HVACPersistentDataHandling einmalig im Hauptprogramm, das zyklisch aufgerufen wird, zu instanzieren. Ansonsten wird der instanzierte FB intern nicht freigegeben.

Eine Wertänderung kann vom Gebäudeleitsystem, einem lokalen Bediengerät oder von einem Schreibzugriff von TwinCAT aus erfolgen.Beim Neustart des Rechners werden die gesicherten Daten automatisch vom Flash in den RAM zurück gelesen.

Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

Bei eDataSecurityType:= *eHVACDataSecurityType_Idle* werden die persistent deklarierten Variablen nicht spannungsausfallsicher gespeichert.

HINWEIS

Eine sich zyklisch ändernde Variable darf niemals mit der IN_OUT-Variablen eines Funktionsbausteins verbunden werden, wenn eDataSecurityType:= eHVACDataSecurityType_Persistent ist. Es würde zu einem frühzeitigen Verschleiß des Flashspeichers führen.

bSetDefault: Wenn die Variable TRUE ist, werden die Default-Werte der VAR_IN_OUT Variablen übernommen.

bEnable: Mit der Variablen *bEnable*wird der Funktionsbaustein frei gegeben. Ist *bEnable* = FALSE erfolgt keine Schaltung am Ausgang *bOutput.*

dtSystemTime: Mit der Variablen *dtSystemTime* wird dem Zeitschaltprogramm die Rechnersystemzeit übergeben.

bReset: Quittierungseingang bei einer Störung. Setzt den Merker *bInvalidParameter* zurück.

VAR_IN_OUT

```
arrHoliday   : ARRAY[1..20] OF ST_HVACHoliday;
```

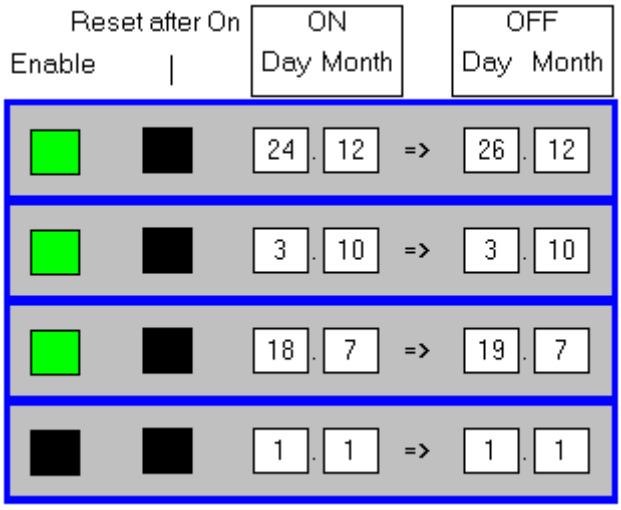
arrHoliday: Die Variable *arrHoliday* ist ein Array mit 20 Deklarationen der Strukturvariablen **ST_HVACHoliday**.

Für jede Instanz des Funktionsbausteins **FB_HVACSchedulerPublicHolidays** muss ein Array vom Typ **ST_HVACHoliday** erzeugt werden. Die Größe des Arrays muss 20 sein, da der Funktionsbaustein insgesamt 20 Zeitkanäle verarbeitet und diese als Referenz (VAR_IN_OUT) erwartet.

```
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
```

Jeder Zeitschaltkanal wird mit der Variablen *bEnable* (Ein/Aus) frei geschaltet. Ist *bEnable* = FALSE ist der Zeitschaltkanal deaktiviert, eingetragene Zeiten sind dann ohne Wirkung. Für einmalige Ereignisse wird die Variable *bResetAfterOn* (einmalig) aktiviert. Nach dem Ablauf der in diesem Kanal eingestellten Zeit bleiben die unter *uiOn_Day, uiOff_Day, uiOn_Month* und *uiOff_Month*erstellten Einträge erhalten. Die Variable *bEnable* (Ein/Aus) wird jedoch auf FALSE zurückgesetzt. So kann ein einmaliges Ereignis einmal definiert und bei Bedarf wieder aktiviert werden.

Beispiel



24.12. von 00:00 Uhr bis 26.12. um 23:59:59 Uhr eingeschaltet

03.10. von 00:00 Uhr bis 03.10. um 23:59:59 Uhr eingeschaltet

18.07. von 00:00 Uhr bis 19.07. um 23:59:59 Uhr eingeschaltet

1.1. bis 1.1. Zeitschaltkanal nicht aktiviert, immer ausgeschaltet!

Visualisierungsvorlage für die Target VISU mit einem Zeitschaltkanal: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659730443.zip

VAR_OUTPUT

```
bOutput          : BOOL;
bEdgeOn          : BOOL;
bEdgeOff         : BOOL;
tNextOn          : TIME;      0 .. t#71582m47s295msmin      (* Nächster Einschaltpunkt, Range max.
= Obergrenze von Time ca.50 Tage *)
tNextOff         : TIME;      0 .. t#71582m47s295msmin      (* Nächster Ausschaltpunkt, Range max.
= Obergrenze von Time ca.50 Tage *)
bInvalidParameter: BOOL;
```

bOutput: Ist gleich TRUE wenn einer der Zeitschaltkanäle eingeschaltet hat.

bEdgeOn: Ist für einen SPS-Zyklus TRUE nach dem Einschalten von *bOutput*.

bEdgeOff: Ist für einen SPS-Zyklus TRUE nach dem Ausschalten von *bOutput*.

tNextOn: Zeit bis zum nächsten Einschalten des Zeitschaltprogramms.

tNextOff: Zeit bis zum nächsten Abschalten des Zeitschaltprogramms.

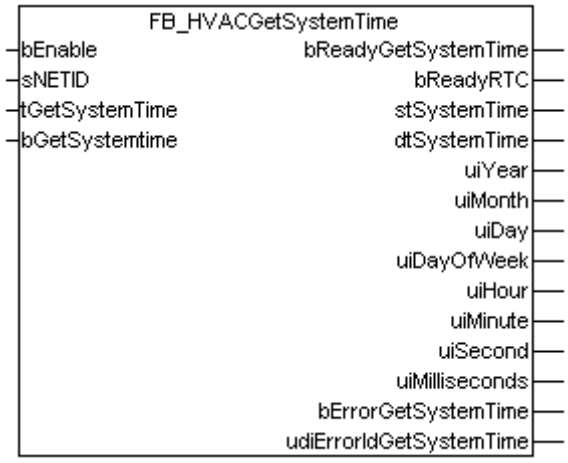
bInvalidParameter: Bei der Plausibilitätsüberprüfung ist ein Fehler aufgetreten. Wir durch *bReset* wieder gelöscht.

Dokumente hierzu

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

3.8 HLK System

3.8.1 FB_HVACGetSystemTime



Anwendung

Mit diesem Funktionsbaustein kann eine interne Uhr (Real Time Clock RTC) in der TwinCAT SPS realisiert werden. Die RTC-Uhr wird mit der Freigabe des Funktionsbausteines über *bEnable* mit der aktuellen NT-Systemzeit initialisiert. Es wird ein Systemtakt der CPU benutzt um die aktuelle RTC-Zeit zu berechnen. Der

Funktionsbausteins muss in jedem Zyklus der SPS ein Mal aufgerufen werden, damit die aktuelle Zeit berechnet werden kann. Intern wird in dem Funktionsbaustein eine Instanz der Funktionsbausteine **NT_GetTime** und **RTC_EX2** aus der Utilities-Bibliothek aufgerufen.

HinweisDie RTC-Zeit weicht bedingt durch die Systemeigenschaften von einer Referenzzeit ab. Die Abweichung hängt von der Zykluszeit der SPS, dem Wert des System-Basis-Ticks und der verwendeten Hardware ab. Die RTC wird deshalb intern im Funktionsbaustein mit der NT-Systemzeit in Abhängigkeit der Eingangsvariablen sNETID und tGetSystemTime synchronisiert. Die lokale NT-Systemzeit können Sie wiederum mit der Hilfe des SNTP-Protokolls mit einer Referenzzeit synchronisieren. Mehr Informationen dazu finden Sie im Beckhoff Information System unter:Beckhoff Information System > Embedded-PC > Betriebssysteme > CE > SNTP: Simple Network Time Protocol

Anwendungsbeispiel

Download	Benötige Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

```
bEnable       : BOOL;
sNETID        : T_AmsNetId;
tGetSystemTime: TIME;
bGetSystemTime: BOOL;
```

bEnable: Freigabe des Bausteins. Ist *bEnable* = TRUE, so wird die RTC-Uhr mit der NT-Systemzeit initialisiert.

sNETID: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden dessen NT-Systemzeit ermittelt werden soll. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

tGetSystemTime: Zeitangabe, mit der die RTC-Uhr mit der NT-Systemzeit regelmäßig synchronisiert wird. Diese Zeitangabe muss größer gleich 5 Sekunden sein, ansonsten wird die RTC-Uhr nicht synchronisiert. Parallel zu der Zeitangabe *tGetSystemTime* kann die RTC-Uhr über die Eingangsvariable *bGetSystemTime* synchronisiert werden.

bGetSystemTime: Mit einer steigenden Flanke an diesem Eingang wird die RTC-Uhr mit der NT-Systemzeit synchronisiert wird. Dieses geschieht parallel zu der Zeitangabe *tGetSystemTime*.

VAR_OUTPUT

```
bReadyGetSystemTime    : BOOL;
bReadyRTC              : BOOL;
stSystemTime           : TIMESTRUCT;
dtSystemTime           : DT;
uiYear                 : UINT;
uiMonth                : UINT;
uiDay                  : UINT;
uiDayOfWeek            : UINT;
uiHour                 : UINT;
uiMinute               : UINT;
uiSecond               : UINT;
uiMilliseconds         : UINT;
bErrorGetSystemTime    : BOOL;
udiErrorIdGetSystemTime: UDINT;
```

bReadyGetSystemTime: Der Funktionsbaustein wurde erfolgreich mit der NT-Systemzeit initialisiert oder synchronisiert.

bReadyRTC: Wurde der Funktionsbaustein mindestens einmal initialisiert, so wird dieser Ausgang gesetzt. Ist dieser Ausgang gesetzt, dann sind die Werte für das Datum, Uhrzeit und Millisekunden an den Ausgängen gültig.

stSystemTime: Struktur mit der aktuellen RTC-Zeit.

dtSystemTime: Datum und Tageszeit der RTC-Zeit.

uiYear: Das Jahr: 1970 ~ 2106;

uiMonth: Der Monat: 1 ~ 12 (Januar = 1, Februar = 2 usw.);

uiDay: Tag des Monats: 1 ~ 31;

uiDayOfWeek: Der Wochentag: 0 ~ 6 (Sonntag = 0, Montag = 1 usw.);

uiHour: Stunde: 0 ~ 23;

uiMinute: Minute: 0 ~ 59;

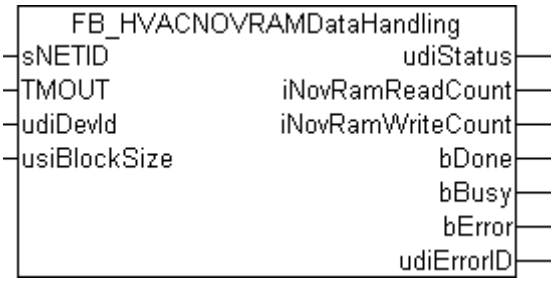
uiSecond: Sekunde: 0 ~ 59;

uiMilliseconds: Millisekunde: 0 ~ 999;

bErrorGetSystemTime: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt. Der Ausgang signalisiert mit einem TRUE, dass ein Fehler beim Initialisieren oder Synchronisieren mit der NT-Systemzeit anliegt. Es wird im Funktionsbaustein so lange versucht die RTC-Uhr zu initialisieren oder synchronisieren bis der Fehler behoben ist. Die RTC-Uhr startet mit einer falschen Datums- und Zeitangabe und muss deshalb mit der NT-Systemzeit synchronisiert werden.

udiErrorIdGetSystemTime: Liefert bei einem gesetzten *bErrorGetSystemTime*-Ausgang die ADS-Fehlernummer.

3.8.2 FB_HVACNOVRAMDataHandling



Anwendung

Mit diesem Funktionsbaustein werden SPS-Variablen spannungsausfallsicher ins NOVRAM geschrieben. Nach einem Spannungsausfall oder nach einem Neustart der Steuerung wird das NOVRAM komplett ausgelesen.

Das Verhalten vom Funktionsbaustein sieht wie folgt aus, wenn beim Aufstarten alles korrekt war:
udiStatus:= 1;
iNovRamReadCount:= 1;
iNovRamWriteCount:= 0;
bDone:= TRUE;

HinweisBei ARM basierten Laufzeitsystemen bitte darauf achten, dass die Speicheradressen der SPS-Variablen durch 4 teilbar sind!

Bemerkungen

Die global deklarierten Variablen *g_dwHVACVarConfigStart* und *g_dwHVACVarConfigEnd* werden mit der Anfangsadresse und Endadresse des Speicherbereiches lokiert. Der Anwender muss darauf achten, dass es nicht zu Speicherüberlappungen kommt. Aus den beiden Adressen wird dann intern die Blockgröße ermittelt. Der gesamte Merkerbereich wird automatisch bei Änderung der Merkervariablen in das NOVRAM weggeschrieben.

siehe auch Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659731851.zip

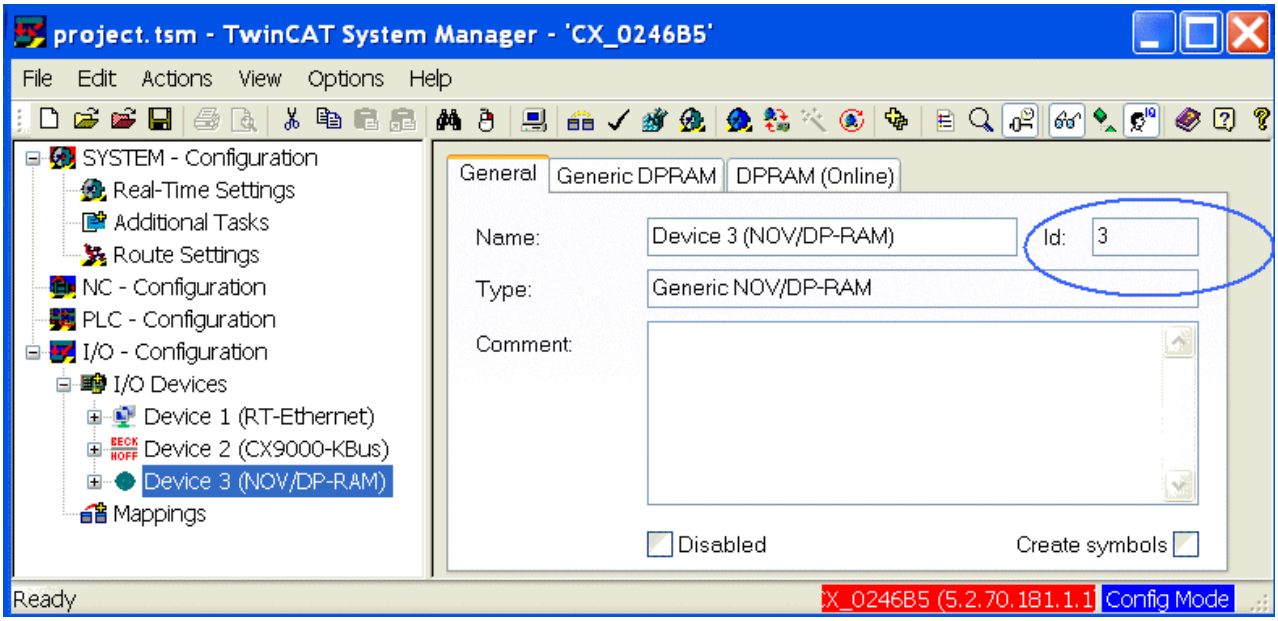
VAR_INPUT

```
sNETID          : T_AmsNetId;
TMOUT           : TIME;
udiDevID        : UDINT;
usiBlockSize    : USINT;
```

sNETID : AmsNetId des TwinCAT-Rechners auf dem die Funktion ausgeführt werden soll. Für den lokalen Rechner, kann auch ein Leerstring angegeben werden.

TMOUT : Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

udiDevID : Über die Geräte-ID wird das NOVRAM des CX90xx oder CX10xx spezifiziert, auf das mit dem Funktionsbaustein schreibend oder lesend zugegriffen werden soll. Die Geräte-IDs werden während der Hardware-Konfiguration vom TwinCAT System Manager festgelegt.



usiBlockSize : Die Blockgröße auf die pro Lese-/Schreibzyklus zugegriffen wird, wird in Prozent angegeben. z.B 20, d.h. es werden 5 Lese-/Schreibzyklen benötigt, um auf den gesamten Merkerbereich zuzugreifen.

VAR_OUTPUT

```
udiStatus          : UDINT;
iNovramReadCount   : INT;
iNovramWriteCount  : INT;
bDone              : BOOL;
bBusy              : BOOL;
bError             : BOOL;
udiErrorID         : UDINT;
```

udiStatus : = 0 , kein Status

= 1 , gültige Daten beim letzten Lesen des NOVRAM

= 2 , ungültige Daten beim letzten Lesen des NOVRAM. Daten wurden verworfen.

iNovramReadCount : Zähler, der beim Auslesen des NOVRAM um 1 inkrementiert.

iNovramWriteCount: Zähler, der beim Beschreiben des NOVRAM um 1 inkrementiert.

bDone: Wird auf TRUE gesetzt, wenn der Funktionsblock ausgeführt wurde.

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

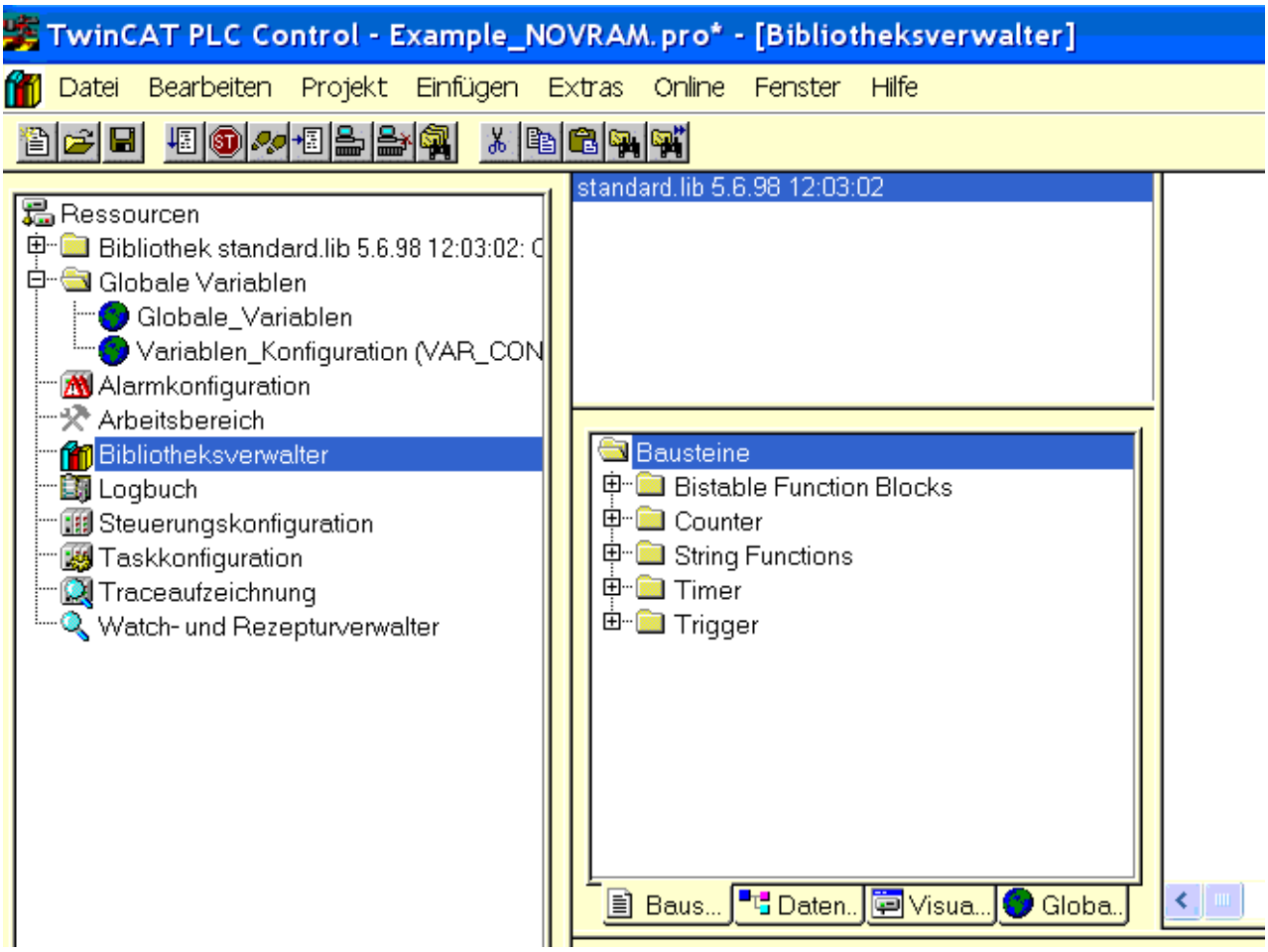
bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorID* enthalten.

udiErrorID: Enthält den befehlsspezifischen Fehlercode. Siehe bitte ADS Return Codes.

Beispiel zur Handhabung innerhalb TwinCAT PLC

1.Schritt

- Neues Projekt anlegen
- Einfügen der TcHVAC.lib, dies geschieht über die Registerkarte Ressourcen im Object Organizer



2. Schritt

- Projekt übersetzen, dabei erhält man zwei Warnungen, da in der TcHVAC.lib zwei nicht vollständig definierte Adressen vorhanden sind. Über diese beiden Adressen muss der Anwender die Start- und Endadresse des Speicherbereiches festlegen.

3. Schritt

- Über den Karteireiter Ressourcen im Object Organizer unter Variablen_Konfiguration kann der Menübefehl **Alle Instanzpfade** im Menü **Einfügen** benutzt werden. Nach der Durchführung dieses Schrittes kann der Anwender die Startadresse und Endadresse definieren.

4. Schritt

- Definieren der Adressen in der Variablen_Konfiguration unter den Globalen Variablen.

Auf ARM basierenden Plattformen (z.B. CX90xx) muss darauf geachtet werden, dass die Adressen der lokierten Variablen durch 4 teilbar sind.

5. Schritt

- In diesem Beispiel wurde als Anfangadresse 0 und als Endadresse 56 definiert.

6. Schritt

- Nachdem die Schritte 1 bis 5 durchgeführt worden sind , muss das Projekt neu übersetzt werden.

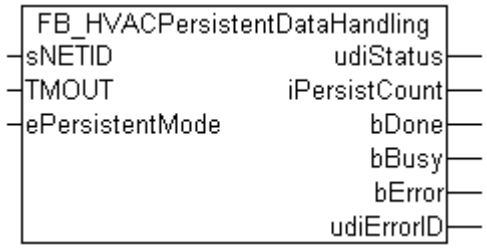
- Aus Performance-Gründen ist es sinnvoll, die lokierten Variablen (VAR_CONFIG Objekt) lückenlos zu adressieren.
- Wenn die Variablen-Konfiguration (VAR_CONFIG) nachträglich geändert wird, ist es zwingend erforderlich die Änderungen über ein "Online Change" und nicht über "Load all" auf das Zielsystem zu laden. Nur bei einem "Online Change" wird sichergestellt, dass die NovRAM Daten korrekt erhalten bleiben. Bei "Load all" wurden die NovRAM Daten als ungültig erkannt.

Beispiel für die Deklaration von lokierten Variablen:

```
Heizkurve.fbRealMinTemp_NOVRAM.rVar_N        AT %MB12 : REAL;
Heizkurve.fbRealMaxTemp_NOVRAM.rVar_N        AT %MB16 : REAL;
Heizkurve.fbRealNightSetback_NOVRAM.rVar_N       AT %MB20 : REAL;
Heizkurve.fbRealSetpoint_Y1_NOVRAM.rVar_N        AT %MB24 : REAL;
Heizkurve.fbRealSetpoint_Y2_NOVRAM.rVar_N        AT %MB28 : REAL;
Heizkurve.fbRealSetpoint_Y3_NOVRAM.rVar_N        AT %MB32 : REAL;
Heizkurve.fbRealSetpoint_Y4_NOVRAM.rVar_N        AT %MB36 : REAL;
Heizkurve.fbRealOutsideTemp_X1_NOVRAM.rVar_N     AT %MB40 : REAL;
Heizkurve.fbRealOutsideTemp_X2_NOVRAM.rVar_N     AT %MB44 : REAL;
Heizkurve.fbRealOutsideTemp_X3_NOVRAM.rVar_N     AT %MB48 : REAL;
Heizkurve.fbRealOutsideTemp_X4_NOVRAM.rVar_N     AT %MB52 : REAL;

.g_dwHVACVarConfigStart                  AT %MB0  : DWORD;
.g_dwHVACVarConfigEnd                AT %MB56 : DWORD;
```

3.8.3 FB_HVACPersistentDataHandling



Anwendung

Mit diesem Funktionsbaustein werden alle SPS Variablen die persistent deklariert sind (VAR PERSISTENT) spannungsausfallsicher in eine Datei geschrieben. Die als persistent deklarierten SPS Variablen innerhalb der Funktionsbausteine aus der TcHVAC.lib werden auf Änderung in diese Datei geschrieben. Diese Datei wird gespeichert unter ..\TwinCAT\Boot Verzeichnis und heißt wie folgt: TCPLC_T_x.wbp (x = Nummer des Laufzeitsystems). Damit nicht sofort nach einer Änderung der persistenten Variablen der neue Zustand gesichert wird, ist in der Standardeinstellung ein Zeitfenster von 5s eingebaut. Dieses Zeitfenster kann vom Anwender geändert werden. Er muss der global deklarierten konstanten Variable **g_tHVACWriteBackupDataTime : TIME := t#5s** die neue Zeit zuweisen.

Erst nach Ablauf des Zeitfensters, wird der neue Zustand gesichert. Im ungünstigsten Fall kann das bedeuten, dass Änderungen, die innerhalb des letzten Zeitfenster getätigt wurden und ein Spannungsausfall hat stattgefunden, nicht gesichert sind.

Bei jedem Neustart der Steuerung wird der Zustand der persistenten Daten aus der Datei ausgelesen. Über die Ausgangsvariable *udiStatus* wird angezeigt, ob die Daten gültig sind.

Um die persistenten Variablen innerhalb der Funktionsbausteine spannungsausfallsicher zu machen, ist es notwendig im MAIN-Programm eine Instanz von dem FB_HVACPersistentDataHandling Baustein anzulegen. Anhand des folgenden Applikationsbeispiels wird die Handhabung dieser Funktionalität nähergebracht.

siehe auch Anwendungsbeispiel: https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659716235.zip

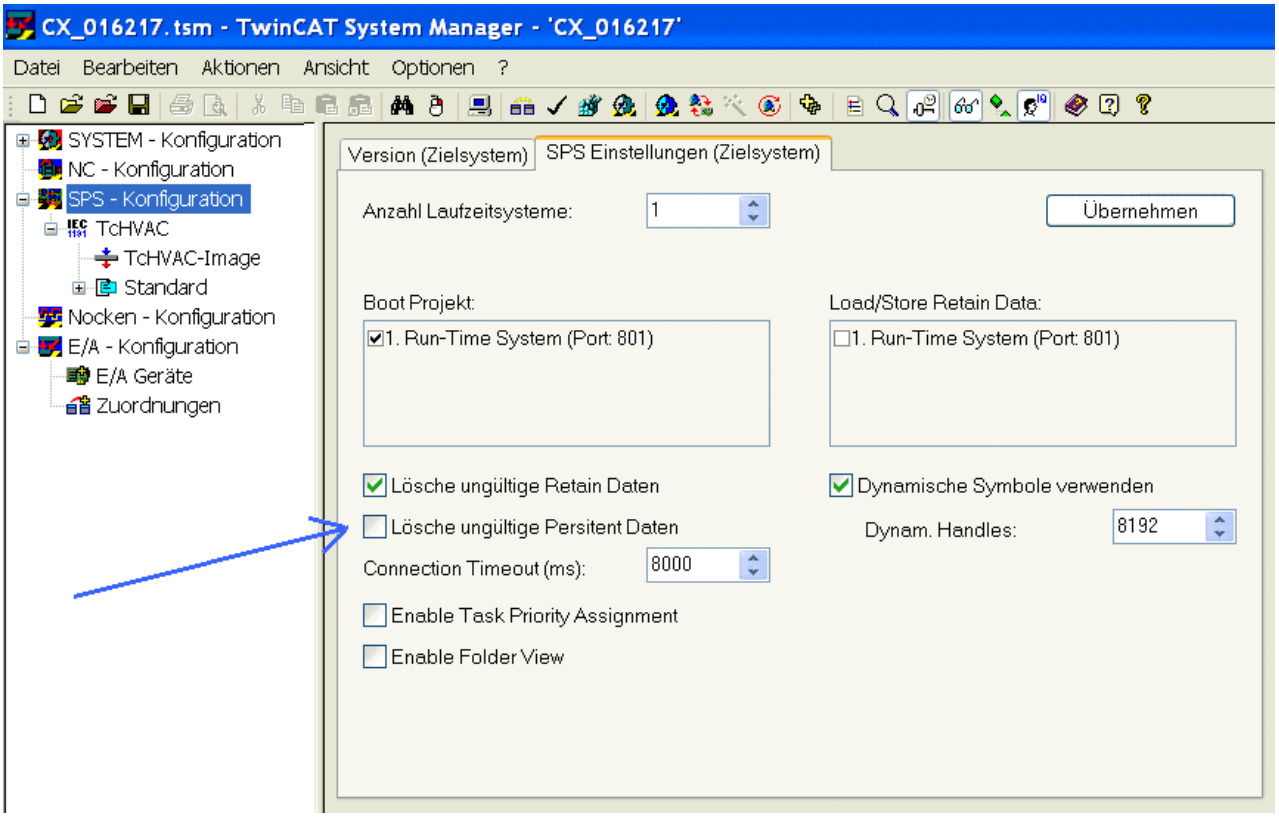
Das Verhalten vom Funktionsbaustein sieht wie folgt aus, wenn beim Aufstarten alles korrekt war:
udiStatus:= 1;
iPersistCount:= 1;
bDone:= FALSE;

Ist das Ergebnis der Abfrage von usiStatus=1 bedeutet dies, dass die persistenten Variablen aus dem Speicher gelesen wurden. bDone wird erst TRUE, wenn die persistenten Variablen erfolgreich geschrieben wurden.

Folgende Einstellung muss im TwinCAT System Manager vorgenommen werden:

Unter SPS-Konfiguration im Karteireiter SPS Einstellungen (Zielsystem) muss die Option

Lösche ungültige Persistent Daten **deaktiviert** sein!! Das deaktivieren muss mit dem Button **Übernehmen** durchgeführt werden.



VAR_INPUT

```
sNETID          : T_AmsNetId;
TMOUT           : TIME;
ePersistentMode : E_PersistentMode;
```

sNETID : AmsNetId des TwinCAT-Rechners auf dem die Funktion ausgeführt werden soll. Für den lokalen Rechner, kann auch ein Leerstring angegeben werden.

TMOUT : Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

ePersistentMode: Modus in dem die persistenten Daten geschrieben werden sollen. Siehe auch E_PersistentMode.

VAR_OUTPUT

```
udiStatus        : UDINT;
iPersistCount    : INT;
bDone            : BOOL;
bBusy            : BOOL;
bError           : BOOL;
udiErrorID       : UDINT;
```

udiStatus : 0 = No status
1 = Persistent data read successfully
2 = Backup from persistent data read
3 = Neither persistent nor backup data read

iPersistCount: Zähler, der nach erfolgreichem Schreiben um 1 inkrementiert.

bDone: Wird auf TRUE gesetzt, wenn der Funktionsblock ausgeführt wurde.

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

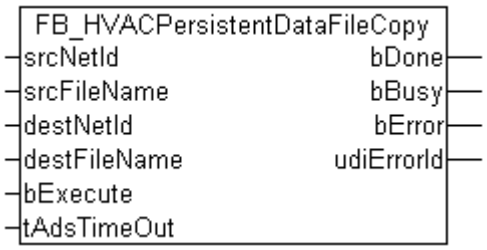
bError: Dieser Ausgang wird auf TRUE gesetzt, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist.

udiErrorID: Enthält den befehlsspezifischen Fehlercode. Siehe bitte ADS Return Codes.

Dokumente hierzu

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

3.8.4 FB_HVACPersistentDataFileCopy



Anwendung

Mit diesem Baustein können Binärdateien z.B. auf dem lokalen TwinCAT PC oder von einem Remote-TwinCAT PC auf den lokalen TwinCAT PC kopiert werden. Mit dem Funktionsbaustein kann nicht auf Netzlaufwerke zugegriffen werden. Bei einer steigenden Flanke am *bExecute* Eingang werden folgende Schritte ausgeführt.

a) Öffnen der Quell- und Ziel-Datei;

b) Lesen der Quell-Datei in einen Puffer;

c) Schreiben der gelesenen Bytes aus dem Puffer in die Ziel-Datei;

d) Überprüfen ob das Ende der Quell-Datei erreicht wurde. Wenn nicht dann b) und c) wiederholen. Wenn ja, dann zu e) springen;

e) Schließen der Quell- und Ziel-Datei;

Die Datei wird stückweise kopiert. Die Größe des Puffers wurde im Baustein auf 100 Byte festgelegt, kann aber geändert werden.

Anwendungsbeispiel

Download	Benötige Bibliothek

TcHVAC.pro [▶ 540]	TcHVAC.lib

VAR_INPUT

```
srcNETID    : T_AmsNetId;
srcFileName : T_MaxString;
destNETID   : T_AmsNetId;
destFileName: T_MaxString;
bExecute    : BOOL;
tAdsTimeOut : TIME;
```

srcNETID : AmsNetId des TwinCAT-Rechners auf dem die Funktion ausgeführt werden soll. Für den lokalen Rechner, kann auch ein Leerstring angegeben werden.

srcFileName: Enthält den Pfad- und Dateinamen der zu öffnenden Datei.

Der Pfad kann nur auf das lokale File System des Rechners zeigen! Das bedeutet, Netzwerkpfade können hier nicht angegeben werden!

destNETID : AmsNetId des TwinCAT-Rechners auf dem die Datei kopiert werden soll.

destFileName : Enthält den Pfad- und Dateinamen der zur Zieldatei. Hinweis: Der Pfad kann nur auf das lokale File System des Rechners zeigen! Das bedeutet, Netzwerkpfade können hier nicht angegeben werden!

bExecute : Mit einer steigenden Flanke an *bExecute* Eingang werden die o.a. Schritte ausgeführt.

tAdsTimeOut : Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

```
bDone           : BOOL;
bBusy           : BOOL;
bError          : BOOL;
udiErrorID      : UDINT;
```

bDone: Wird auf TRUE geschaltet, wenn der Funktionsblock ausgeführt wurde.

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

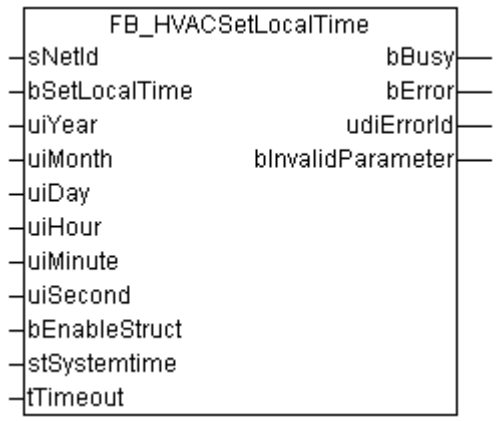
bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist.

udiErrorID: Enthält den befehlsspezifischen Fehlercode. Siehe bitte ADS Return Codes.

Dokumente hierzu

tchvac.zip (Resources/zip/11659726219.zip)

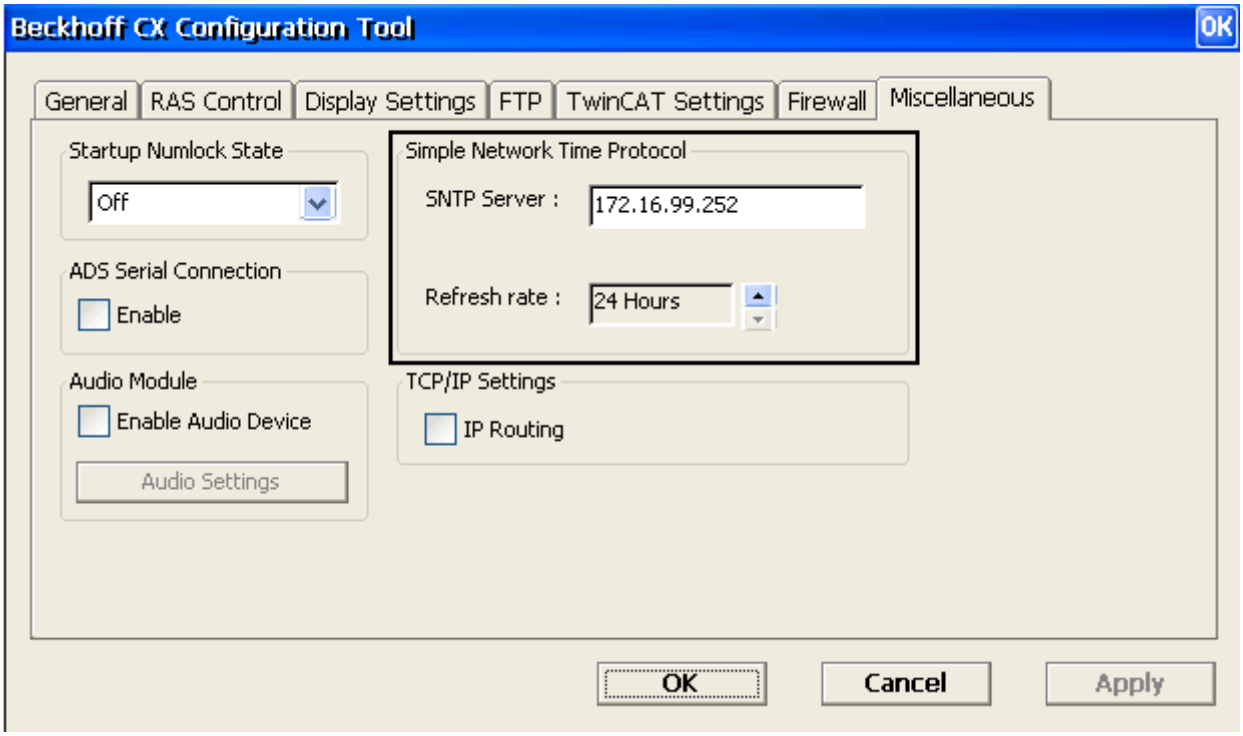
3.8.5 FB_HVACSetLocalTime



Anwendung

Mit dem Funktionsbaustein *FB_HVACSetLocalTime* kann die lokale NT-Systemzeit und das Datum eines TwinCAT-Systems gesetzt werden (die lokale NT-Systemzeit wird in der Taskleiste eingeblendet). Die Systemzeit kann entweder über die einzelnen Variablen *uiYear, uiMonth, uiDay, uiHour, uiMinute* und *uiSecond* oder der Struktur *stSystemtime* vorgegeben werden, siehe *bEnableStruct.*
Intern wird in dem Funktionsbaustein eine Instanz des Funktionsbausteins **NT_SetLocalTime** aus der TcUtilities-Bibliothek aufgerufen.

> **Hinweis** Die lokale NT-Systemzeit können Sie auch mit der Hilfe des SNTP-Protokolls mit einer Referenzzeit synchronisieren. Mehr Informationen dazu finden Sie im Beckhoff Information System unter: Beckhoff Information System > Embedded-PC > Betriebssysteme > CE > SNTP: Simple Network Time Protocol



Anwendungsbeispiel

Download	Benötige Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

VAR_INPUT

```
sNetId          : T_AmsNetId;
bSetLocalTime   : BOOL;
uiYear          : UINT;         1970 - 2106
uiMonth         : UINT;         1 - 12
uiDay           : UINT;         1 - 31
uiHour          : UINT;         0 - 23
uiMinute        : UINT;         0 - 59
uiSecond        : UINT;         0 - 59
bEnableStruct   : BOOL;
stSystemtime    : TIMESTRUCT;
tTimeout        : TIME;
```

sNETID: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden dessen lokale NT-Systemzeit gesetzt werden soll. Für den lokalen Rechner kann auch ein Leerstring *sNetId* := ''; angegeben werden.

bSetLocalTime: Aktivierung des Funktionsbausteins mit einer steigenden Flanke.

uiYear: Das Jahr: 1970 ~ 2106; die Variable ist aktiv, wenn *bEnableStruct* = FALSE ist. Bei falscher Angabe wird *bInvalidParameter* = TRUE und die neue lokale NT-Systemzeit wurde nicht gesetzt.

uiMonth: Der Monat: 1 ~ 12 (Januar = 1, Februar = 2 usw.); die Variable ist aktiv, wenn *bEnableStruct* = FALSE ist. Bei falscher Angabe wird *bInvalidParameter* = TRUE und die neue lokale NT-Systemzeit wurde nicht gesetzt.

uiDay: Tag des Monats: 1 ~ 31; Februar mit 28 oder 29 Tagen sowie die Monate mit 30 oder 31 Tagen werden überprüft. Die Variable ist aktiv, wenn *bEnableStruct* = FALSE ist. Bei falscher Angabe wird *bInvalidParameter* = TRUE und die neue lokale NT-Systemzeit wurde nicht gesetzt.

uiHour: Stunde: 0 ~ 23; die Variable ist aktiv, wenn *bEnableStruct* = FALSE ist. Bei falscher Angabe wird *bInvalidParameter* = TRUE und die neue lokale NT-Systemzeit wurde nicht gesetzt.

uiMinute: Minute: 0 ~ 59; die Variable ist aktiv, wenn *bEnableStruct* = FALSE ist. Bei falscher Angabe wird *bInvalidParameter* = TRUE und die neue lokale NT-Systemzeit wurde nicht gesetzt.

uiSecond: Sekunde: 0 ~ 59; die Variable ist aktiv, wenn *bEnableStruct* = FALSE ist. Bei falscher Angabe wird *bInvalidParameter* = TRUE und die neue lokale NT-Systemzeit wurde nicht gesetzt.

bEnableStruct: Wenn *bEnableStruct* = TRUE ist, so wird die neue lokale NT-Systemzeit über die Eingangsvariable *stSystemtime* gesetzt. Ist *bEnableStruct* = FALSE, so wird die neue lokale NT-Systemzeit über die Eingangsvariablen *uiYear, uiMonth, uiDay, uiHour, uiMinute* und *uiSecond* gesetzt.

stSystemtime: Struktur mit der neuen lokalen NT-Systemzeit. Die Struktur ist aktiv, wenn *bEnableStruct* = TRUE ist. Für die Struktur gelten die selben Bereiche (Range) wie für die Eingangsvariablen *uiYear, uiMonth, uiDay, uiHour, uiMinute* und *uiSecond.* Bei falscher Angabe wird *bInvalidParameter* = TRUE und die neue lokale NT-Systemzeit wurde nicht gesetzt.

tTimeout: Gibt die Timeout-Zeit an, die bei der Ausführung nicht überschritten werden darf.

VAR_OUTPUT

```
bBusy            : BOOL;
bError           : BOOL;
udiErrorId       : UDINT;
bInvalidParameter: BOOL;
```

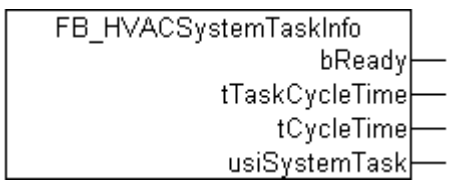
bBusy: Bei der Aktivierung des Funktionsbausteins über eine steigende Flanke an *bSetLocalTime* wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Sollte ein Fehler bei der Übertragung der NT-Systemzeit erfolgen, dann wird dieser Ausgang gesetzt. *bError* wird mit der Aktivierung des Funktionsbaustein über die Eingangsvariable *bSetLocalTime* zurück gesetzt.

udiErrorId: Liefert bei einem gesetzten *bError* die ADS-Errorcode.

bInvalidParameter: *bInvalidParameter* ist TRUE, wenn die Bereiche (Range) der Variablen für die Uhrzeit und des Datums nicht eingehalten wurden: *uiYear, uiMonth, uiDay, uiHour, uiMinute* und *uiSecond.* Für die Zeitstruktur *stSystemtime* gelten dieselben Bereiche (Range) wie für die Eingangsvariablen. Ist *bInvalidParameter* = TRUE, so wurde die neue lokale NT-Systemzeit nicht gesetzt. *bInvalidParameter* wird mit der Aktivierung des Funktionsbaustein über die Eingangsvariable *bSetLocalTime* zurückgesetzt.

3.8.6 FB_HVACSystemTaskInfo



Der Funktionsbaustein ermittelt Systemvariablen der Task mit einer Auflösung von 1ms , in der er aktuell aufgerufen wird. Liegt die aktuelle Zykluszeit unter 1ms, so wird an der Ausgangsvariable *tCycleTime*die eingestellte Taskzykluszeit *tTaskCycleTime* ausgegeben.

HINWEIS

Die *tTaskCycleTime* des SPS-Programms darf nicht höher als 100 ms sein, weil ansonsten die digitalen Ausgänge abfallen.
Das liegt daran, dass der interne K-Bus der Busklemmen synchron zum SPS Programm läuft, dieser nicht mehr früh genug
angetriggert wird und der Watchdog der Busklemmen aktiv wird.

VAR_OUTPUT

```
bReady              : BOOL;
tTaskCycleTime      : TIME;
tCycleTime          : TIME;
usiSystemTask       : USINT;
```

bReady: Die Variable ist TRUE, wenn die System Informationen ausgelesen sind.

tTaskCycleTime: Eingestellte Taskzykluszeit.

tCycleTime: Benötigte Zykluszeit für den letzten Zyklus.

usiSystemTask: Taskindex der Task.

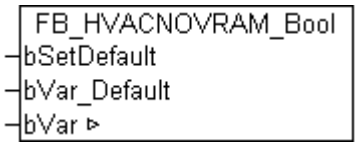
4 Backup Bausteine

4.1 BackupVar NOVRAM

Der Benutzer kann Standard Datentypen (siehe Tabelle) verwenden und diese in seiner Applikation einbinden um definierte SPS-Variablen spannungsausfallsicher zu speichern.

Datentyp	BackupVar NOVRAM Baustein
BOOL	FB_HVACNOVRAM_Bool [▶ 512]
BYTE	FB_HVACNOVRAM_Byte [▶ 512]
DINT	FB_HVACNOVRAM_Dint [▶ 513]
DWORD	FB_HVACNOVRAM_Dword [▶ 513]
INT	FB_HVACNOVRAM_Int [▶ 514]
LREAL	FB_HVACNOVRAM_Lreal [▶ 514]
REAL	FB_HVACNOVRAM_Real [▶ 514]
TIME	FB_HVACNOVRAM_Time [▶ 515]
SINT	FB_HVACNOVRAM_Sint [▶ 515]
UDINT	FB_HVACNOVRAM_Udint [▶ 515]
UINT	FB_HVACNOVRAM_Uint [▶ 516]
USINT	FB_HVACNOVRAM_Usint [▶ 516]
WORD	FB_HVACNOVRAM_Word [▶ 516]

4.1.1 FB_HVACNOVRAM_Bool



VAR_INPUT

```
bSetDefault         : BOOL;
bVar_Default        : BOOL;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *bVar_Default* übernommen.

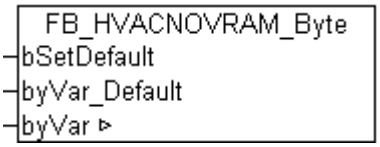
bVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp BOOL.

VAR_IN_OUT

```
Name                : Type
bVar                : BOOL;
```

bVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.1.2 FB_HVACNOVRAM_Byte



VAR_INPUT

```
bSetDefault         : BOOL;
byVar_Default       : BYTE;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *byVar_Default* übernommen.

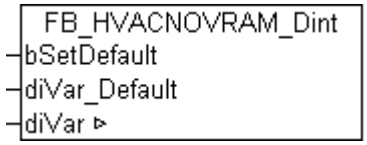
byVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp BYTE.

VAR_IN_OUT

```
byVar           : BYTE;
```

byVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.1.3 FB_HVACNOVRAM_Dint



VAR_INPUT

```
bSetDefault           : BOOL;
diVar_Default         : DINT;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *diVar_Default* übernommen.

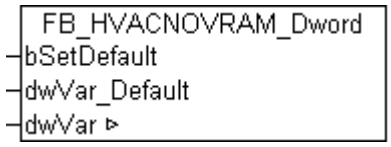
diVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp DINT.

VAR_IN_OUT

```
diVar             : DINT;
```

diVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.1.4 FB_HVACNOVRAM_Dword



VAR_INPUT

```
bSetDefault           : BOOL;
dwVar_Default         : DWORD;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *dwVar_Default* übernommen.

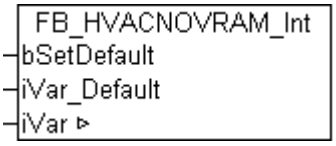
dwVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp DWORD.

VAR_IN_OUT

```
dwVar             : DWORD;
```

dwVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.1.5 FB_HVACNOVRAM_Int



VAR_INPUT

```
bSetDefault           : BOOL;
iVar_Default          : INT;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *iVar_Default* übernommen.

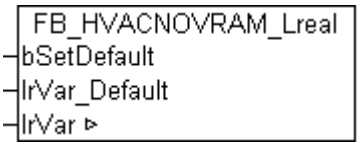
iVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp INT.

VAR_IN_OUT

```
iVar              : INT;
```

iVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.1.6 FB_HVACNOVRAM_Lreal



VAR_INPUT

```
bSetDefault           : BOOL;
lrVar_Default         : LREAL;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *lrVar_Default* übernommen.

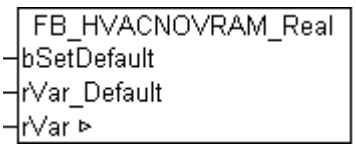
lrVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp LREAL.

VAR_IN_OUT

```
lrVar             : LREAL;
```

lrVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.1.7 FB_HVACNOVRAM_Real



VAR_INPUT

```
bSetDefault           : BOOL;
rVar_Default          : REAL;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *rVar_Default* übernommen.

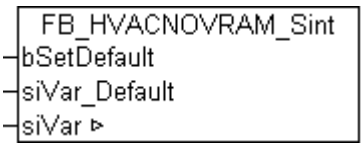
rVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp REAL.

VAR_IN_OUT

```
rVar              : REAL;
```

rVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.1.8 FB_HVACNOVRAM_Sint



VAR_INPUT

```
bSetDefault           : BOOL;
siVar_Default         : SINT;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *siVar_Default* übernommen.

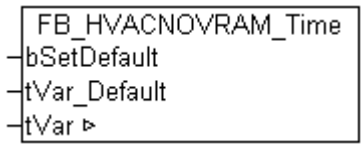
siVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp SINT.

VAR_IN_OUT

```
siVar             : SINT;
```

siVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.1.9 FB_HVACNOVRAM_Time



VAR_INPUT

```
bSetDefault         : BOOL;
tVar_Default         : TIME;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *tVar_Default* übernommen.

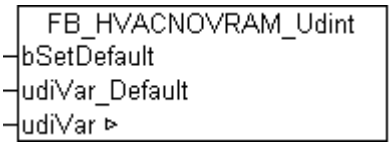
tVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp TIME.

VAR_IN_OUT

```
tVar              : TIME;
```

tVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.1.10 FB_HVACNOVRAM_Udint



VAR_INPUT

```
bSetDefault           : BOOL;
udiVar_Default        : UDINT;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *udiVar_Default* übernommen.

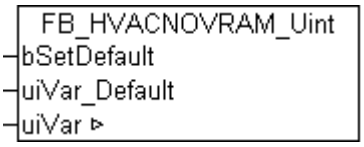
udiVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp UDINT.

VAR_IN_OUT

```
udiVar          : UDINT;
```

udiVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.1.11 FB_HVACNOVRAM_Uint



VAR_INPUT

```
bSetDefault          : BOOL;
uiVar_Default        : UINT;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *uiVar_Default* übernommen.

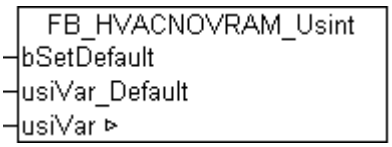
uiVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp UINT.

VAR_IN_OUT

```
uiVar            : UINT;
```

uiVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.1.12 FB_HVACNOVRAM_Usint



VAR_INPUT

```
bSetDefault          : BOOL;
usiVar_Default       : USINT;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *usiVar_Default* übernommen.

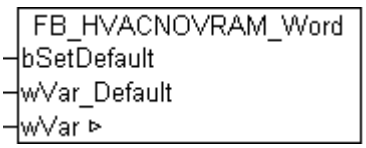
usiVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp USINT.

VAR_IN_OUT

```
usiVar           : USINT;
```

usiVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.1.13 FB_HVACNOVRAM_Word



VAR_INPUT

```
bSetDefault       : BOOL;
wVar_Default      : WORD;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *wVar_Default* übernommen.

wVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp WORD.

VAR_IN_OUT

```
wVar          : WORD;
```

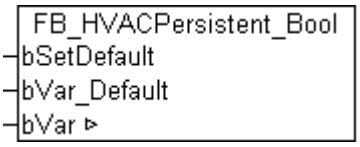
wVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.2 BackupVar Persistent

Der Benutzer kann Standard Datentypen (siehe Tabelle) verwenden und diese in seiner Applikation einbinden um definierte SPS-Variablen spannungsausfallsicher zu speichern.

Datentyp	BackupVar Persistent Baustein
BOOL	FB_HVACPersistent_Bool [▶ 517]
BYTE	FB_HVACPersistent_Byte [▶ 518]
DINT	FB_HVACPersistent_Dint [▶ 518]
DWORD	FB_HVACPersistent_Dword [▶ 518]
INT	FB_HVACPersistent_Int [▶ 519]
LREAL	FB_HVACPersistent_Lreal [▶ 519]
REAL	FB_HVACPersistent_Real [▶ 519]
SINT	FB_HVACPersistent_Sint [▶ 520]
STRING	FB_HVACPersistent_String [▶ 520]
STRUCT	FB_HVACPersistent_Struct [▶ 521]
TIME	FB_HVACPersistent_Time [▶ 522]
UDINT	FB_HVACPersistent_Udint [▶ 522]
UINT	FB_HVACPersistent_Uint [▶ 523]
USINT	FB_HVACPersistent_Usint [▶ 523]
WORD	FB_HVACPersistent_Word [▶ 523]

4.2.1 FB_HVACPersistent_Bool



VAR_INPUT

```
bSetDefault          : BOOL;
bVar_Default         : BOOL;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *bVar_Default* übernommen.

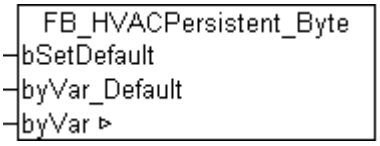
bVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp BOOL.

VAR_IN_OUT

```
bVar             : BOOL;
```

bVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.2.2 FB_HVACPersistent_Byte



VAR_INPUT

```
bSetDefault          : BOOL;
byVar_Default        : BYTE;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *byVar_Default* übernommen.

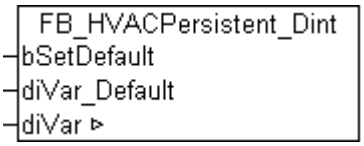
byVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp BYTE.

VAR_IN_OUT

```
byVar           : BYTE;
```

byVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.2.3 FB_HVACPersistent_Dint



VAR_INPUT

```
bSetDefault          : BOOL;
diVar_Default        : DINT;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *diVar_Default* übernommen.

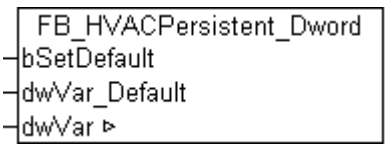
diVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp DINT.

VAR_IN_OUT

```
diVar           : DINT;
```

diVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.2.4 FB_HVACPersistent_Dword



VAR_INPUT

```
bSetDefault          : BOOL;
dwVar_Default        : DWORD;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *dwVar_Default* übernommen.

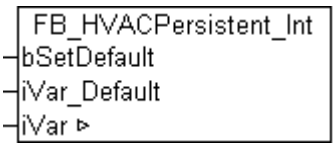
dwVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp DWORD.

VAR_IN_OUT

```
dwVar           : DWORD;
```

dwVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.2.5 FB_HVACPersistent_Int



VAR_INPUT

```
bSetDefault          : BOOL;
iVar_Default         : INT;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *iVar_Default* übernommen.

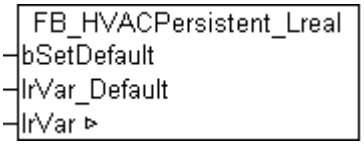
iVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp INT.

VAR_IN_OUT

```
iVar             : INT;
```

iVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.2.6 FB_HVACPersistent_Lreal



VAR_INPUT

```
bSetDefault          : BOOL;
lrVar_Default        : LREAL;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *lrVar_Default* übernommen.

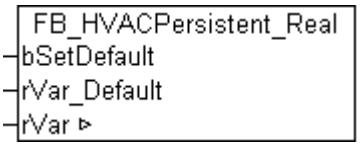
lrVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp LREAL.

VAR_IN_OUT

```
lrVar            : LREAL;
```

lrVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.2.7 FB_HVACPersistent_Real



VAR_INPUT

```
bSetDefault          : BOOL;
rVar_Default         : REAL;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *rVar_Default* übernommen.

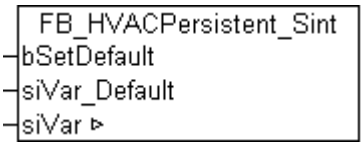
rVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp REAL.

VAR_IN_OUT

```
rVar            : REAL;
```

rVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.2.8 FB_HVACPersistent_Sint



VAR_INPUT

```
bSetDefault         : BOOL;
siVar_Default       : SINT;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *siVar_Default* übernommen.

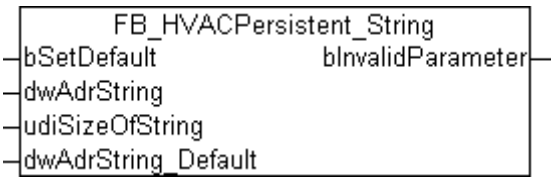
siVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp SINT.

VAR_IN_OUT

```
siVar           : SINT;
```

siVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.2.9 FB_HVACPersistent_String



VAR_INPUT

```
bSetDefault         : BOOL;
dwAdrString         : DWORD;
udiSizeOfString     : UDINT;
dwAdrString_Default: DWORD;
```

bSetDefault: Wenn die Variable TRUE ist, werden die Voreinstellwerte der deklarierten Variable *dwAdrString_Default* übernommen und auf die *dwAdrString* Adresse kopiert.

dwAdrString: Adresse der deklarierten Variable, die persistent gespeichert werden soll.

udiSizeOfString: Größe der deklarierten Variable in Bytes. (1 Byte pro Zeichen (Größe bei der Deklaration) + 1 Byte für die Nullterminierung)

dwAdrString_Default: Adresse vom String mit dem Voreinstellwert.

VAR_OUTPUT

```
bInvalidParameter     : BOOL;
```

bInvalidParameter: TRUE, wenn *udiSizeOfString* > *g_udiMaxSizeOfString* ODER *dwAdrString* = *0* ODER *dwAdrString_Default* = *0*

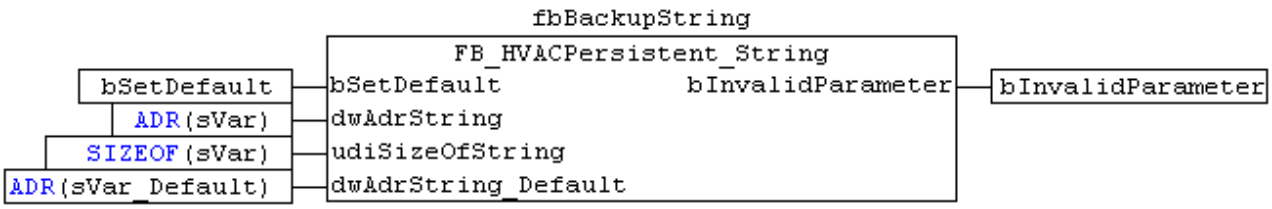
VAR_GLOBAL CONSTANT

```
Name                : Type
g_udiMaxSizeOfString: UDINT := 255; (* size in number of characters *)
```

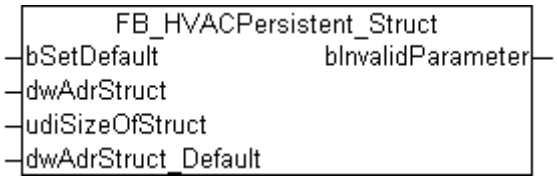
g_udiMaxSizeOfString: Über diese global deklarierte Konstante kann eine beliebige Größe der Zeichenkette angegeben werden. Voreinstellwert ist 255.

Beispiel:

```
VAR
    fbBackupString     : FB_HVACPersistent_STRING;
    bSetDefault        : BOOL;                          (* default flag *)
    sVar               : STRING(55);                    (* normal value *)
    sVar_Default       : STRING(55) := 'Store the default String Value';    (* default value *)
    bInvalidParameter  : BOOL;
END_VAR
```



4.2.10 FB_HVACPersistent_Struct



VAR_INPUT

```
bSetDefault         : BOOL;
dwAdrStruct         : DWORD;
udiSizeOfStruct     : UDINT;
dwAdrStruct_Default: DWORD;
```

bSetDefault: Wenn die Variable TRUE ist, werden die Voreinstellwerte der lokierten Strukturinstanz *dwAdrStruct_Default* übernommen und auf die *dwAdrStruct* Adresse kopiert.

dwAdrStruct: Adresse der deklarierten Struktur Variable, die persistent gespeichert werden soll.

udiSizeOfStruct: Größe der instanzierten Struktur in Bytes.

dwAdrStruct_Default: Adresse der lokierten Strukturinstanz mit den Voreinstellwerten.

VAR_OUTPUT

```
bInvalidParameter     : BOOL;
```

bInvalidParameter: TRUE, wenn *udiSizeOfStruct* > *g_udiMaxNoOfBytesInStruct* ODER *dwAdrStruct = 0* ODER *dwAdrStruct_Default = 0*

VAR_GLOBAL CONSTANT

```
g_udiMaxNoOfBytesInStruct  : UDINT := 16;  (* size in number of bytes *)
```

g_udiMaxNoOfBytesInStruct: Über diese global deklarierte Konstante wird die Größe der Struktur in Bytes definiert. Standardmäßig sind 16 Bytes definiert. Diese Konstante kann den Erfordernissen angepasst werden.

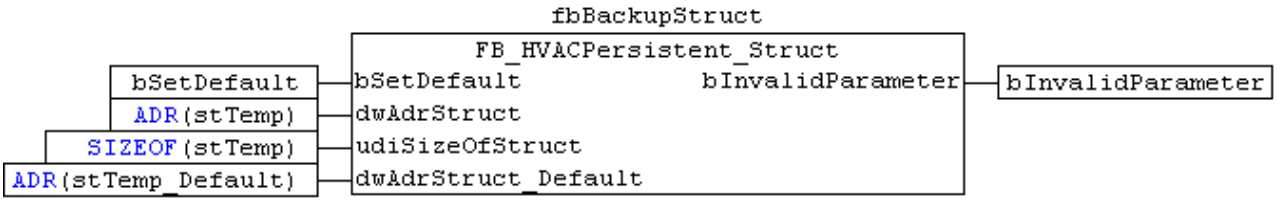
Beispiel:

```
VAR
    fbBackupStruct     : FB_HVACPersistent_STRUCT;
    bSetDefault        : BOOL;             (* default flag *)
    stTemp             : ST_Temp;              (* structure values *)
    stTemp_Default     : ST_Temp;              (* default Values *)
    bInvalidParameter  : BOOL;
END_VAR
```

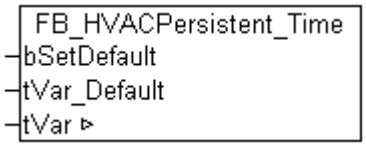
ST_Temp:

```
TYPE ST_Temp :
STRUCT
       (* total of 8 bytes *)
       byVar1      : BYTE;
       byVar2      : BYTE;
       byVar3      : BYTE;
       byVar4      : BYTE;
       iVar1       : INT;
       iVar2       : INT;
END_STRUCT
END_TYPE
```

```
stTemp_Default.byVar1:= 11;     (* one byte *)
stTemp_Default.byVar2:= 22;     (* one byte *)
stTemp_Default.byVar3:= 33;     (* one byte *)
stTemp_Default.byVar4:= 44;     (* one byte *)
stTemp_Default.iVar1:= 55;      (* two bytes *)
stTemp_Default.iVar2:= 66;      (* two bytes *)
```



4.2.11 FB_HVACPersistent_Time



VAR_INPUT

```
bSetDefault           : BOOL;
tVar_Default          : TIME;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *tVar_Default* übernommen.

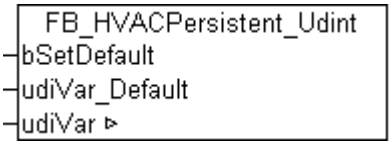
tVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp TIME.

VAR_IN_OUT

```
tVar              : TIME;
```

tVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.2.12 FB_HVACPersistent_Udint



VAR_INPUT

```
bSetDefault          : BOOL;
udiVar_Default       : UDINT;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *udiVar_Default* übernommen.

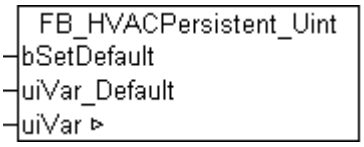
udiVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp UDINT.

VAR_IN_OUT

```
udiVar          : UDINT;
```

udiVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.2.13 FB_HVACPersistent_Uint



VAR_INPUT

```
bSetDefault         : BOOL;
uiVar_Default       : UINT;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *uiVar_Default* übernommen.

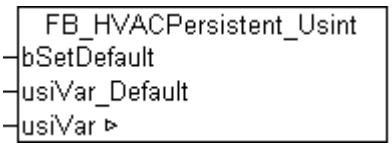
uiVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp UINT.

VAR_IN_OUT

```
uiVar           : UINT;
```

uiVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.2.14 FB_HVACPersistent_Usint



VAR_INPUT

```
bSetDefault         : BOOL;
usiVar_Default      : USINT;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *usiVar_Default* übernommen.

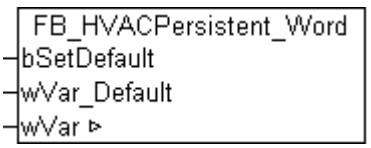
usiVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp USINT.

VAR_IN_OUT

```
usiVar          : USINT;
```

usiVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

4.2.15 FB_HVACPersistent_Word



VAR_INPUT

```
bSetDefault         : BOOL;
wVar_Default        : WORD;
```

bSetDefault: Wenn die Variable TRUE ist, wird der Wert der Eingangsvariable *wVar_Default* übernommen.

wVar_Default: Defaultwert, der vom Anwender definiert wird. Variable ist vom Datentyp WORD.

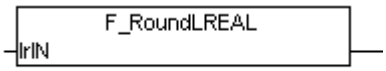
VAR_IN_OUT

```
wVar                : WORD;
```

wVar: Variable, die vom Anwender spannungausfallsicher programmiert wird.

5 Funktionen

5.1 F_RoundLREAL



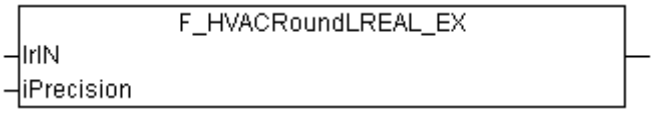
Anwendung

Die Funktion *F_RoundLREAL* rundet die Eingangsvariable *lrIN* vom Datentyp LREALauf 1 Nachkommastelle. Diese Funktion kann auch für den Datentypen REAL angewendet werden.

VAR_INPUT

```
lrIN                : LREAL;
```

5.2 F_RoundLREAL_EX



Anwendung

Die Funktion *F_RoundLREAL_EX* rundet die Eingangsvariable *lrIN* vom Datentyp LREALauf die durch *iPrecision* angegebenen Nachkommastellen. Diese Funktion kann auch für den Datentypen REAL angewendet werden. Intern wird in der Funktion eine Instanz der Funktion **LTRUNC** aus der Bibliothek TcMath.lib verwendet.

VAR_INPUT

```
lrIN                : LREAL;
iPrecision          : INT;      0..5
```

lrIN: Fließkommazahl, die formatiert werden soll.

- Ist lrIN < 0.1 AND lrIN >= 0.05, dann ist der Rückgabewert der Funktion 0.1

iPrecision: Präzision. Der Wert bestimmt die Anzahl der Zahlen hinter dem Dezimalpunkt. Der Bereich der Nachkommastellen beginnt bei 0 und endet bei 5.

Anwendungsbeispiel

Download	Benötigte Bibliothek
TcHVAC.pro [▶ 540]	**TcHVAC.lib**

6 Aufzählungen und Strukturen

6.1 E_HVAC2PointActuatorMode

```
TYPE E_HVAC2PointActuatorMode:
(
    eHVAC2PointActuatorMode_Auto_BMS    := 0,
    eHVAC2PointActuatorMode_Open_BMS    := 1,
    eHVAC2PointActuatorMode_Close_BMS       := 2,
    eHVAC2PointActuatorMode_Auto_OP     := 3,
    eHVAC2PointActuatorMode_Open_OP     := 4,
    eHVAC2PointActuatorMode_Close_OP    := 5
);
END_TYPE
```

..**_BMS**: **B**uilding **M**anagement **S**ystem; fernbedienen des Aktors von der Management-Ebene.

..**OP**: **O**perator **P**anel; örtliche Bedienung des Aktors von einem lokalen Bedienpanel.

6.2 E_HVAC2PointCtrlMode

```
TYPE E_HVAC2PointCtrlMode:
(
     eHVAC2PointCtrlMode_Auto_BMS   := 0,
     eHVAC2PointCtrlMode_On_BMS     := 1,
     eHVAC2PointCtrlMode_Off_BMS    := 2,
     eHVAC2PointCtrlMode_Auto_OP    := 3,
     eHVAC2PointCtrlMode_On_OP      := 4,
     eHVAC2PointCtrlMode_Off_OP     := 5
);
END_TYPE
```

..**_BMS**: **B**uilding **M**anagement **S**ystem; fernbedienen des Aktors von der Management-Ebene.

..**OP**: **O**perator **P**anel; örtliche Bedienung des Aktors von einem lokalen Bedienpanel.

6.3 E_HVAC3PointActuatorMode

```
TYPE E_HVAC3PointActuatorMode:
(
    eHVAC3PointActuatorMode_Auto_BMS       := 0,
    eHVAC3PointActuatorMode_Open_BMS       := 1,
    eHVAC3PointActuatorMode_Close_BMS      := 2,
    eHVAC3PointActuatorMode_Stop_BMS       := 3,
    eHVAC3PointActuatorMode_Auto_OP    := 4,
    eHVAC3PointActuatorMode_Open_OP    := 5,
    eHVAC3PointActuatorMode_Close_OP       := 6,
    eHVAC3PointActuatorMode_Stop_OP    := 7
);
END_TYPE
```

..**_BMS**: **B**uilding **M**anagement **S**ystem; fernbedienen des Aktors von der Management-Ebene.

..**OP**: **O**perator **P**anel; örtliche Bedienung des Aktors von einem lokalen Bedienpanel.

6.4 E_HVACActuatorMode

```
TYPE E_HVACActuatorMode :
(
  eHVACActuatorMode_Auto_BMS       := 0,
  eHVACActuatorMode_Off_BMS    := 1,
  eHVACActuatorMode_Speed1_BMS     := 2,
  eHVACActuatorMode_Speed2_BMS     := 3,
  eHVACActuatorMode_Speed3_BMS     := 4,
  eHVACActuatorMode_Speed4_BMS     := 5,
```

```
  eHVACActuatorMode_Auto_OP    := 6,
  eHVACActuatorMode_Off_OP     := 7,
  eHVACActuatorMode_Speed1_OP      := 8,
  eHVACActuatorMode_Speed2_OP      := 9,
  eHVACActuatorMode_Speed3_OP      := 10,
  eHVACActuatorMode_Speed4_OP      := 11
);
END_TYPE
```

..**_BMS**: **B**uilding **M**anagement **S**ystem; fernbedienen des Aktors von der Management-Ebene.

..**OP**: **O**perator **P**anel; örtliche Bedienung des Aktors von einem lokalen Bedienpanel.

6.5 E_HVACAirConditioning2SpeedMode

```
TYPE E_HVACAirConditioning2SpeedMode:
(
    eHVACAirConditioning2SpeedMode_Off            := 0,
    eHVACAirConditioning2SpeedMode_Heating        := 1,
    eHVACAirConditioning2SpeedMode_Cooling        := 2,
    eHVACAirConditioning2SpeedMode_HeatingAndCooling      := 3
);
END_TYPE
```

6.6 E_HVACAnalogOutputMode

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

..**_BMS**: **B**uilding **M**anagement **S**ystem; fernbedienen des Aktors von der Management-Ebene.

..**OP**: **O**perator **P**anel; örtliche Bedienung des Aktors von einem lokalen Bedienpanel.

6.7 E_HVACAntiBlockingMode

```
TYPE E_HVACAntiBlockingMode:
(
    eHVACAntiBlockingMode_Off       := 0,
    eHVACAntiBlockingMode_Downtime  := 1,
    eHVACAntiBlockingMode_Weekly    := 2
);
END_TYPE
```

6.8 E_HVACBusTerminal_KL32xx

```
TYPE E_HVACBusTerminal_KL32xx :
(
    eHVACBusTerminal_None := 0,
(*=============================================================================*)
    eHVACBusTerminal_KL3201_0000 := 1,   (*Standard PT100*)
    eHVACBusTerminal_KL3201_0010 := 2,   (*PT200*)
    eHVACBusTerminal_KL3201_0011 := 3,   (*not supported, PT200 Siemens format*)
    eHVACBusTerminal_KL3201_0012 := 4,   (*PT500*)
    eHVACBusTerminal_KL3201_0013 := 5,   (*not supported, PT500 Siemens format*)
    eHVACBusTerminal_KL3201_0014 := 6,   (*PT1000*)
    eHVACBusTerminal_KL3201_0015 := 7,   (*not supported, PT1000 Siemens format*)
    eHVACBusTerminal_KL3201_0016 := 8,   (*Ni100*)
    eHVACBusTerminal_KL3201_0017 := 9,   (*not supported, Ni100 Siemens format*)
    eHVACBusTerminal_KL3201_0018 := 10,  (*not supported*)
    eHVACBusTerminal_KL3201_0020 := 11,  (*Ohm10_1200*)
    eHVACBusTerminal_KL3201_0021 := 12,  (*not supported, PT100 Siemens format*)
    eHVACBusTerminal_KL3201_0023 := 13,  (*Ni120*)
    eHVACBusTerminal_KL3201_0024 := 14,  (*not supported, Ni120 Siemens format*)
```

```
    eHVACBusTerminal_KL3201_0025 := 15,  (*Ni1000*)
    eHVACBusTerminal_KL3201_0026 := 16,  (*not supported, Ni1000 Siemens format*)
    eHVACBusTerminal_KL3201_0027 := 17,  (*not supported, Ohm10_10000*)
    eHVACBusTerminal_KL3201_0028 := 18,  (*not supported, high resolution 0,01°C*)
    eHVACBusTerminal_KL3201_0029 := 19,  (*Ni1000_LS, Landis&Stefa, TK5000*)
    eHVACBusTerminal_KL3201_0030 := 20,  (*not supported*)
    eHVACBusTerminal_KL3201_0031 := 21,  (*PT1000,2 wire connection*)
(*=============================================================================*)
    eHVACBusTerminal_KL3202_0000 := 22,  (*Standard PT100*)
    eHVACBusTerminal_KL3202_0010 := 23,  (*PT200*)
    eHVACBusTerminal_KL3202_0011 := 24,  (*not supported, PT200 Siemens format*)
    eHVACBusTerminal_KL3202_0012 := 25,  (*PT500*)
    eHVACBusTerminal_KL3202_0013 := 26,  (*not supported, PT500 Siemens format*)
    eHVACBusTerminal_KL3202_0014 := 27,  (*PT1000*)
    eHVACBusTerminal_KL3202_0015 := 28,  (*not supported, PT1000 Siemens format*)
    eHVACBusTerminal_KL3202_0016 := 29,  (*Ni100*)
    eHVACBusTerminal_KL3202_0017 := 30,  (*not supported, Ni100 Siemens format*)
    eHVACBusTerminal_KL3202_0018 := 31,  (*not supported*)
    eHVACBusTerminal_KL3202_0020 := 32,  (*Ohm10_1200*)
    eHVACBusTerminal_KL3202_0021 := 33,  (*not supported, PT100 Siemens format*)
    eHVACBusTerminal_KL3202_0023 := 34,  (*Ni120*)
    eHVACBusTerminal_KL3202_0024 := 35,  (*not supported, Ni120 Siemens format*)
    eHVACBusTerminal_KL3202_0025 := 36,  (*Ni1000*)
    eHVACBusTerminal_KL3202_0026 := 37,  (*not supported, Ni1000 Siemens format*)
    eHVACBusTerminal_KL3202_0027 := 38,  (*not supported, Ohm10_10000*)
    eHVACBusTerminal_KL3202_0028 := 39,  (*not supported, high resolution 0,01°C*)
    eHVACBusTerminal_KL3202_0029 := 40,  (*Ni1000_LS, Landis&Stefa, TK5000*)
    eHVACBusTerminal_KL3202_0030 := 41,  (*not supported*)
    eHVACBusTerminal_KL3202_0031 := 42,  (*PT1000,2 wire connection*)
(*=============================================================================*)
    eHVACBusTerminal_KL3204_0000 := 43,  (*Standard PT100*)
    eHVACBusTerminal_KL3204_0010 := 44,  (*PT200*)
    eHVACBusTerminal_KL3204_0011 := 45,  (*not supported, PT200 Siemens format*)
    eHVACBusTerminal_KL3204_0012 := 46,  (*PT500*)
    eHVACBusTerminal_KL3204_0013 := 47,  (*not supported, PT500 Siemens format*)
    eHVACBusTerminal_KL3204_0014 := 48,  (*PT1000*)
    eHVACBusTerminal_KL3204_0015 := 49,  (*not supported, PT1000 Siemens format*)
    eHVACBusTerminal_KL3204_0016 := 50,  (*Ni100*)
    eHVACBusTerminal_KL3204_0017 := 51,  (*not supported, Ni100 Siemens format*)
    eHVACBusTerminal_KL3204_0018 := 52,  (*not supported*)
    eHVACBusTerminal_KL3204_0020 := 53,  (*Ohm10_1200*)
    eHVACBusTerminal_KL3204_0021 := 54,  (*not supported, PT100 Siemens format*)
    eHVACBusTerminal_KL3204_0023 := 55,  (*Ni120*)
    eHVACBusTerminal_KL3204_0024 := 56,  (*not supported, Ni120 Siemens format*)
    eHVACBusTerminal_KL3204_0025 := 57,  (*Ni1000*)
    eHVACBusTerminal_KL3204_0026 := 58,  (*not supported, Ni1000 Siemens format*)
    eHVACBusTerminal_KL3204_0027 := 59,  (*not supported, Ohm10_10000*)
    eHVACBusTerminal_KL3204_0028 := 60,  (*not supported, high resolution 0,01°C*)
    eHVACBusTerminal_KL3204_0029 := 61,  (*Ni1000_LS, Landis&Stefa, TK5000*)
    eHVACBusTerminal_KL3204_0030 := 62,  (*not supported*)
(*=============================================================================*)
    eHVACBusTerminal_KL3228_0000 := 63,  (*Standard PT1000*)
(*=============================================================================*)
    eHVACBusTerminal_NotSupported    := 64,  (*error, bus terminal not supported*)
    eHVACBusTerminal_KL3201_NotSupported := 65,  (*error, bus terminal not supported*)
    eHVACBusTerminal_KL3202_NotSupported := 66,  (*error, bus terminal not supported*)
    eHVACBusTerminal_KL3204_NotSupported := 67,  (*error, bus terminal not supported*)
    eHVACBusTerminal_KL3228_NotSupported := 68   (*error, bus terminal not supported*)
(*=============================================================================*)
    eHVACBusTerminal_KL3208_0010     := 69,(*Standard PT1000*)
    eHVACBusTerminal_KL3208_NotSupported := 70(*error, bus terminal not supported*)
);
END_TYPE
```

6.9 E_HVACCtrlMode

```
TYPE E_HVACCtrlMode :
(
    eHVACCtrlMode_Auto     := 0,
    eHVACCtrlMode_Manual   := 1
);
END_TYPE
```

6.10 E_HVACConvectionMode

```
TYPE E_HVACConvectionMode :
(
    eHVACConvectionMode_WithFan       := 0,
    eHVACConvectionMode_WithoutFan    := 1
);
END_TYPE
```

6.11 E_HVACDataSecurityType

```
TYPE E_HVACDataSecurityType:
(
    eHVACDataSecurityType_Persistent     := 0,
    eHVACDataSecurityType_Idle       := 1
);
END_TYPE
```

6.12 E_HVACErrorCodes

```
TYPE E_HVACErrorCodes :
(
eHVACErrorCodes_NoError                     := 0,
eHVACErrorCodes_Error_tTaskCycleTime            := 1,
eHVACErrorCodes_Error_tCtrlCycleTime            := 2,
eHVACErrorCodes_InvalidParam_rDeadRange             := 3,
eHVACErrorCodes_InvalidParam_rDeadBand              := 4,
eHVACErrorCodes_InvalidParam_rKpIsLessThanZero          := 5,
eHVACErrorCodes_InvalidParam_tTi                := 6,
eHVACErrorCodes_InvalidParam_tTv                := 7,
eHVACErrorCodes_InvalidParam_tTd                := 8,
eHVACErrorCodes_InvalidParam_fBaseTime              := 9,
eHVACErrorCodes_InvalidParam_rYMaxIsLessThanrYMin       := 10,
eHVACErrorCodes_Error_MEMCPY                := 11,
eHVACErrorCodes_Error_ModeNotSupported              := 12,
eHVACErrorCodes_Error_rErrHighLimitOverrun          := 13,
eHVACErrorCodes_Error_rErrLowLimitUnderrun          := 14,
eHVACErrorCodes_Error_NoFeedbackActuator1           := 15,
eHVACErrorCodes_Error_NoFeedbackActuator2           := 16,
eHVACErrorCodes_Error_NoFeedbackActuator3           := 17,
eHVACErrorCodes_Error_NoFeedbackActuator4           := 18,
eHVACErrorCodes_Error_NoFeedbackActuator5           := 19,
eHVACErrorCodes_Error_NoFeedbackActuator6           := 20,
eHVACErrorCodes_Error_NoFeedbackActuator7           := 21,
eHVACErrorCodes_Error_NoFeedbackActuator8           := 22,
eHVACErrorCodes_InvalidParam_tOverlap1Actuator          := 23,
eHVACErrorCodes_InvalidParam_iCountCtrl             := 24,
eHVACErrorCodes_iMaxOnLevel                 := 25,
eHVACErrorCodes_tNoFeedbActuator                := 26,
eHVACErrorCodes_Error_iNumberOfSequences            := 27,
eHVACErrorCodes_Error_iMyNumberOfSequences          := 28,
eHVACErrorCodes_Error_iCurrentSequences             := 29,
eHVACErrorCodes_Error_MEMSET                := 30,
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
```

6.13 E_HVACExternalMode

```
TYPE E_HVACExternalMode :
(
    eHVACExternalMode_Off        := 0,
```

```
    eHVACExternalMode_On         := 1,
    eHVACExternalMode_ShiftAbsolute      := 2
);
END_TYPE
```

6.14 E_HVACExternalRequestMode

```
TYPE E_HVACExternalRequestMode:
(
    eHVACExternalRequestMode_ButtonOn_Off          := 0,
    eHVACExternalRequestMode_ButtonOffDelay        := 1,
    eHVACExternalRequestMode_SwitchOn_Off          := 2
);
END_TYPE
```

6.15 E_HVACPlantMode

```
TYPE E_HVACPlantMode:
(
    eHVACPlantMode_TimeSchedulingOnly             := 0,
    eHVACPlantMode_ExternalRequestOnly            := 1,
    eHVACPlantMode_TimeScheduling_And_ExternalRequest     := 2,
    eHVACPlantMode_TimeScheduling_Or_ExternalRequest      := 3
);
END_TYPE
```

6.16 E_HVACPowerMeasurementMode

```
TYPE E_HVACPowerMeasurementMode:
(
    eHVACPowerMeasurementMode_AutoAllValues     := 0,
    eHVACPowerMeasurementMode_Current           := 1,
    eHVACPowerMeasurementMode_Voltage           := 2,
    eHVACPowerMeasurementMode_EffectivePower    := 3,
    eHVACPowerMeasurementMode_EnergyConsumption     := 4,
    eHVACPowerMeasurementMode_PeakCurrentValue      := 5,
    eHVACPowerMeasurementMode_PeakVoltageValue      := 6,
    eHVACPowerMeasurementMode_PeakPowerValue    := 7
);
END_TYPE
```

6.17 E_HVACReferencingMode

```
TYPE E_HVACReferencingMode :
(
    eHVACReferencingMode_Emulation      := 0,
    eHVACReferencingMode_AnalogFeedback     := 1
);
END_TYPE
```

6.18 E_HVACRegOutsideTemp

```
TYPE E_HVACRegOutsideTemp:
(
    eHVACReqOutsideTemp_NoRequest      := 0,
    eHVACReqOutsideTemp_OTLowerLimit       := 1,
    eHVACReqOutsideTemp_OTHigherLimit      := 2
);
END_TYPE
```

6.19 E_HVACReqPump

```
TYPE E_HVACReqPump:
(
    eHVACReqPump_No          := 0,
    eHVACReqPump_OT_LL       := 1,
    eHVACReqPump_OT_HL       := 2,
    eHVACReqPump_VP          := 3,
    eHVACReqPump_OT_LL_OR_VP     := 4,
    eHVACReqPump_OT_HL_OR_VP     := 5,
```

```
    eHVACReqPump_OT_LL_AND_VP   := 6,
    eHVACReqPump_OT_HL_AND_VP   := 7
END_TYPE
```

eHVACReqPump_No: Es besteht keine Anforderung zur Steuerung der Pumpe seitens des Enums

eHVACReqPump_OT_LL: Die Außentemperatur (OT = *rOutsideTemp*) muss kleiner sein als *rOutsideTempLowLimit* (LL = Lower Limit)

eHVACReqPump_OT_HL: Die Außentemperatur (OT = *rOutsideTemp*) muss größer sein als *rOutsideTempHighLimit* (HL = Higher Limit)

eHVACReqPump_VP: Die Ventilstellung (VP = *rValvePosition*) muss größer sein als *rValvePositionLimitOn*

eHVACReqPump_OT_LL_OR_VP: Die Außentemperatur (OT = *rOutsideTemp*) muss kleiner sein als *rOutsideTempLowLimit* (LL = Lower Limit) ODER die Ventilstellung (VP = *rValvePosition*) muss größer sein als *rValvePositionLimitOn.*

eHVACReqPump_OT_HL_OR_VP: Die Außentemperatur (OT = *rOutsideTemp*) muss größer sein als *rOutsideTempHighLimit* (HL = Higher Limit) ODER die Ventilstellung (VP = *rValvePosition*) muss größer sein als *rValvePositionLimitOn.*

eHVACReqPump_OT_LL_AND_VP: Die Außentemperatur (OT = *rOutsideTemp*) muss kleiner sein als *rOutsideTempLowLimit* (LL = Lower Limit) UND die Ventilstellung (VP = *rValvePosition*) muss größer sein als *rValvePositionLimitOn.*

eHVACReqPump_OT_HL_AND_VP: Die Außentemperatur (OT = *rOutsideTemp*) muss größer sein als *rOutsideTempHighLimit* (HL = Higher Limit) UND die Ventilstellung (VP = *rValvePosition*) muss größer sein als *rValvePositionLimitOn.*

6.20 E_HVACRegValve

```
TYPE E_HVACRegValve:
(
    eHVACReqValve_NoRequest         := 0,
    eHVACReqValve_OrValvePosHigherLimit    := 1,
    eHVACReqValve_AndValvePosHigherLimit   := 2
);
END_TYPE
```

6.21 E_HVACSensorType

```
TYPE E_HVACSensorType :
(
    eHVACSensorType_None        := 0,
    eHVACSensorType_PT100       := 1,
    eHVACSensorType_PT200       := 2,
    eHVACSensorType_PT500       := 3,
    eHVACSensorType_PT1000      := 4,
    eHVACSensorType_NI100       := 5,
    eHVACSensorType_NI120       := 6,
    eHVACSensorType_NI1000      := 7,(*Ni1000,DIN*)
    eHVACSensorType_NI1000_LS       := 8,(*Ni1000_LS, Landis&Staefa, TK5000*)
    eHVACSensorType_Ohm10_1200      := 9,
    eHVACSensorType_Ohm10_5000      := 10,
    eHVACSensorType_Ohm10_10000     := 11,
    eHVACSensorType_NotSupported    := 12
);
END_TYPE
```

6.22 E_HVACSequenceCtrlMode

Über dieses ENUM werden die Regler innerhalb der Sequenz in Abhängigkeit der Anlagenbetriebsarten freigegeben und gesteuert.

```
TYPE E_HVACSequenceCtrlMode :
(
    eHVACSequenceCtrlMode_Stop                      := 0,
    eHVACSequenceCtrlMode_On                    := 1,
    eHVACSequenceCtrlMode_NightCooling                  := 2,
```

```
    eHVACSequenceCtrlMode_FreezeProtection              := 3,
    eHVACSequenceCtrlMode_OverheatingProtection             := 4,
    eHVACSequenceCtrlMode_NightCoolingAndOverheatingProtection      := 5
);
END_TYPE
```

6.23 E_HVACSetpointHeatingMode

```
TYPE E_HVACSetpointHeatingMode :
(
    eHVACSetpointHeatingMode_Off            := 0,
    eHVACSetpointHeatingMode_OnOutsideTemp  := 1,
    eHVACSetpointHeatingMode_OnDate         := 2,
    eHVACSetpointHeatingMode_OnPermanent    := 3,
    eHVACSetpointHeatingMode_OnFreezeProtec := 4,
    eHVACSetpointHeatingMode_OnNight        := 5,
    eHVACSetpointHeatingMode_OnDay          := 6
);
END_TYPE
```

eHVACSetpointHeatingMode_Off: Der Heizkreis ist komplett ausgeschaltet.

In dieser Betriebsart ist auch die Frostschutzfunktion deaktiviert!

eHVACSetpointHeatingMode_OnOutsideTemp: Heizperiode nach Außentemperatur. Wenn die mittlere Außentemperatur *rOutsideTempDamped* kleiner
als der Wert von *rHeatingLimit* ist, wird der Heizkreis eingeschaltet. Der von der Heizkennlinie oder dem Raumsollwertmodul errechnete Vorlauftemperatursollwert wird zum Ausgang *rSetpoint* durchgeschaltet. Der Ausgang *bHeatingPump* wird TRUE und damit wird die Heizkreispumpe eingeschaltet. Das Ein- bzw. Abschalten nach dem Unter- bzw. Überschreiten der Heizgrenze kann mit dem Parameter *tDelayHeatingOn* und *tDelayHeatingOff* verzögert werden.

eHVACSetpointHeatingMode_OnDate: Heizperiode nach Datum. Der Heizkreis ist vom Datum *iOn_Day / iOn_Month* bis *iOff_Day / iOff_Month*
frei geschaltet. Das Ein- bzw. Abschalten der Heizkreispumpe kann nach dem Erreichen der Ein- bzw. Ausschaltgrenzen
verzögert werden.

eHVACSetpointHeatingMode_OnPermanent: Der Heizkreis ist immer eingeschaltet.

eHVACSetpointHeatingMode_OnFreezeProtec: Der Heizkreis ist im Frostschutzbetrieb. Wenn die Eingangsvariable bFreezProtec TRUE ist ,
wird der Wert der an rFreezeProtecSetpoint anliegt an den Ausgang rSetpointOut durchgeleitet.

eHVACSetpointHeatingMode_OnNight: Der Sollwert für die Vorlauftemperatur wird in Abhängigkeit der Außentemperatur berechnet. Der Heizkreis ist dauerhaft eingeschaltet und immer im Nachtabsenkbetrieb. In dieser Betriebsart wird der Wert von der Eingangsvariable rNightSetback von dem rSetpointIn abgezogen.

eHVACSetpointHeatingMode_OnDay: Der Heizkreis ist im Tagbetrieb.

6.24 E_HVACSetpointMode

```
TYPE E_HVACSetpointMode :
(
    eHVACSetpointMode_DIN         := 0,
    eHVACSetpointMode_DINLimited      := 1,
    eHVACSetpointMode_ConstantValueBase   := 2
);
END_TYPE
```

6.25 E_HVACState

```
TYPE E_HVACState :
(
  eHVACState_Idle    := 0,
  eHVACState_Active      := 1,
```

```
  eHVACState_Error       := 2
);
END_TYPE
```

Dieser Ausgang zeigt den aktuellen internen Zustand an, in dem sich der Baustein befindet.

eHVACState_Idle: Ein Reset des Bausteins ist erfolgreich ausgeführt worden, der Baustein wartet auf eine Betriebsartenumschaltung.
eHVACState_Active: Der Baustein befindet sich in Active-State, dieses ist der normale Betriebszustand.
eHVACState_Error: Es ist ein Fehler aufgetreten und der Baustein wird in diesem State nicht ausgeführt.

6.26 E_HVACTemperatureCurve

```
TYPE E_HVACTemperatureCurve :
(
    eHVACTemperatureCurve_None         := 0,
    eHVACTemperatureCurve_NTC1k8       := 1,         (*NTC 1.8k, negativ, Thermokon*)
    eHVACTemperatureCurve_Ni1000Tk5000     := 2,         (*Ni1000 Tk5000, positiv, Thermokon*)
    eHVACTemperatureCurve_NTC1k_3_A1       := 3,         (*NTC1k, 1K3 A1, negativ, S+S*)
    eHVACTemperatureCurve_NTC1k8_3_A1      := 4,         (*NTC1.8k, 1.8K3 A1, negativ, S+S*)
    eHVACTemperatureCurve_NTC2k2_3_A1      := 5,         (*NTC2.2k, 2.2K3 A1, negativ, S+S*)
    eHVACTemperatureCurve_NTC3k3_3_A1      := 6,         (*NTC3.3k, 3.3K3 A1, negativ, S+S*)
    eHVACTemperatureCurve_Ni1000Tk5000_TCR := 7,         (*Ni1000 Tk5000 TCR, positiv, S+S*)
    eHVACTemperatureCurve_Ni1000_DIN       := 8,         (*Ni1000 DIN, positiv, S+S*)
    eHVACTemperatureCurve_Pt1000_DIN       := 9          (*Ni1000 DIN, positiv, S+S*)
);
END_TYPE
```

6.27 E_HVACTemperatureSensorMode

E_HVACTemperatureSensorMode

```
TYPE E_HVACTemperatureSensorMode :
(
  eHVACTemperatureSensorMode_ReplacementValue        := 0,
  eHVACTemperatureSensorMode_LastValue               := 1,
  eHVACTemperatureSensorMode_AutoResetReplacementValue       := 2,
  eHVACTemperatureSensorMode_AutoResetLastValue          := 3
);
END_TYPE
```

6.28 ST_HVAC2PointCtrlSequence

```
TYPE ST_HVAC2PointCtrlSequence :
STRUCT
     tRemainingTimeIncreaseSequence    : TIME;
     tRemainingTimeDecreaseSequence    : TIME;
     rX                : REAL;
     rW_Max                : REAL;
     rW_Min                : REAL;
     rE                : REAL;
     rCtrl_I_HighLimit         : REAL;
     rCtrl_I_LowLimit          : REAL;
     rCtrl_I_Out               : REAL;
     e2PointCtrlState          : E_HVAC2PointCtrlMode;
     iNumberOfSequences        : INT;
     iMyNumberInSequence           : INT;
     iCurrentSequence          : INT;
     bEnable               : BOOL;
     bError                : BOOL;
     bOut                  : BOOL;
     bActiveCtrl               : BOOL;
     b_rW_Max              : BOOL;
     b_rW_Min              : BOOL;
END_STRUCT
END_TYPE
```

6.29 ST_HVACAggregate

```
TYPE ST_HVACAggregate :
STRUCT
    rY_Max          : REAL;
    rY_Min          : REAL;
    iAggregateStep      : INT;
    bBlock          : BOOL;
END_STRUCT
END_TYPE
```

rY_Max: Parameterangabe für stetige Aggregate.

rY_Min: Parameterangabe für stetige Aggregate.

iAggregateStep: Parameterangabe in welcher Stufe das angesprochene Aggregat fixiert oder regeln soll, siehe *bBlock*.

bBlock: Ist *bBlock* = FALSE, so wird das angesprochene Aggregat in der vorgegebenen Stufe über *iAggregateStep* fixiert. Ist *bBlock* = TRUE, so wird die Regelung des angesprochenen Aggregats frei gegeben von der Ausstufe (0) bis zur vorgegebenen Stufe über iAggregateStep.

6.30 ST_HVACCmdCtrl_8Param

```
TYPE ST_HVACCmdCtrl_8Param :
STRUCT
     udiSecDelayOn              : UDINT;
     udiSecDelayOff             : UDINT;
     udiSecMinOn                : UDINT;
     udiSecMinOff               : UDINT;
END_STRUCT
END_TYPE
```

udiSecDelayOn: Einschaltverzögerung in Sekunden

udiSecDelayOff: Ausschaltverzögerung in Sekunden

udiSecMinOn: Mindesteinschaltzeit in Sekunden

udiSecMinOff: Mindestausschaltzeit in Sekunden

6.31 ST_HVACCmdCtrl_8State

```
TYPE ST_HVACCmdCtrl_8State :
STRUCT
     udiSecRT_DelayOn           : UDINT;
     udiSecRT_DelayOff          : UDINT;
     udiStep                : UDINT;
     udiSecRT_MinOn             : ARRAY[0..g_iNumOfCmdCtrl_8] OF UDINT;
     udiSecRT_MinOff            : ARRAY[0..g_iNumOfCmdCtrl_8] OF UDINT;
END_STRUCT
END_TYPE
```

udiSecRT_DelayOn: Es wird die die verbleibende Zeit der Einschaltverzögerung der aktuellen Stufe *udiStep* angezeigt.

udiSecRT_DelayOff: Es wird die verbleibende Zeit der Ausschaltverzögerung der aktuellen Stufe *udiStep* angezeigt.

udiStep: Status in welcher Stufe sich der Funktionsbaustein befindet.

udiSecRT_MinOn[0.. g_iNumOfCmdCtrl_8 [▶ 541] **]:** In dem eindimmensionalen Feld (Tabelle) *udiSecRT_MinOn[0..*g_iNumOfCmdCtrl_8 [▶ 541]*]* wird die verbleibende Zeit der Mindesteinschaltdauer der Ausgänge angezeigt.

udiSecRT_MinOff[0.. g_iNumOfCmdCtrl_8 [▶ 541] **]:** In dem eindimmensionalen Feld (Tabelle) *udiSecRT_MinOff[0..*g_iNumOfCmdCtrl_8 [▶ 541]*]* wird die verbleibende Zeit der Mindestausschaltdauer der Ausgänge angezeigt.

6.32 ST_HVACHoliday

```
TYPE ST_HVACHoliday :
STRUCT
  uiOn_Day    : UINT;
  uiOn_Month      : UINT;
  uiOff_Day       : UINT;
  uiOff_Month     : UINT;
  bEnable     : BOOL;
  bResetAfterOn       : BOOL;
  bQ          : BOOL;
END_STRUCT
END_TYPE
```

6.33 ST_HVACI_Ctrl

```
TYPE ST_HVACI_Ctrl :
STRUCT
    rIntegralHigh    : REAL;
    rIntegralLow     : REAL;
    udiSecDelayHigh      : UDINT;
    udiSecDelayLow       : UDINT;
END_STRUCT
END_TYPE
```

rIntegralHigh: Positiver Wert für das obere Limit an dem die Integration des I-Übertragungsgliedes angehalten wird.

rIntegralLow: Positiver Wert für das untere Limit an dem die Integration des I-Übertragungsgliedes angehalten wird.

udiSecDelayHigh: Verzögerungszeit nach deren Ablauf das I-Übertragungsglied aktiviert wird.

udiSecDelayLow:

Verzögerungszeit nach deren Ablauf das I-Übertragungsglied aktiviert wird.

6.34 ST_HVACPeriod

```
TYPE ST_HVACPeriod :
STRUCT
  uiOn_hh      : UINT;
  uiOn_mm      : UINT;
  uiOn_Day     : UINT;
  uiOn_Month       : UINT;
  uiOff_hh     : UINT;
  uiOff_mm     : UINT;
  uiOff_Day    : UINT;
  uiOff_Month      : UINT;
  bEnable      : BOOL;
  bResetAfterOn    : BOOL;
  bQ           : BOOL;
END_STRUCT
END_TYPE
```

6.35 ST_HVACParameterScale_nPoint

```
TYPE ST_HVACParameterScale_nPoint :
STRUCT
    rX           : ARRAY [1..g_iMaxNoOfScale_nPoint] OF REAL;
```

```
    rY          : ARRAY [1..g_iMaxNoOfScale_nPoint] OF REAL;
    iNumberOfPoint   : INT;
END_STRUCT
END_TYPE
```

Struktur über der die einzelnen Punkte der X-Y-Koordinaten ihre Wertigkeit bekommen.

rX[1..g_iMaxNoOfScale_nPoint]: Array welches die Wertigkeit der einzelnen Punkte der X-Achse beinhaltet. Wie viele Punkte angegeben werden, ist abhängig von *iNumberOfPoint*.

Folgendes muß beachtet werden: rX[1] muß größer als rX[2], rX[2] muß größer rX[n] sein ODER rX[1] muß kleiner als rX[2], rX[2] muß kleiner rX[n] sein.

rY[1..g_iMaxNoOfScale_nPoint]: Array welches die Wertigkeit der einzelnen Punkte der Y-Achse beinhaltet.Wie viele Punkte angegeben werden, ist abhängig von *iNumberOfPoint*.

iNumberOfPoint: Anzahl der einzelnen Punkte der X-Y-Koordinaten.

Folgendes muß beachtet werden: iNumberOfPoint darf nicht kleiner als 2 ODER größer g_iMaxNoOfScale_nPoint(60) sein.

6.36 ST_HVACPowerMeasurement

```
TYPE ST_HVACPowerMeasurement :
STRUCT
  diIL1, diIL2, diIL3              : DINT;
  diUL1, diUL2, diUL3              : DINT;
  diPL1, diPL2, diPL3              : DINT;
  diPg                      : DINT;
  diCosPhiL1, diCosPhiL2, diCosPhiL3       : DINT;
  diCosPhi                  : DINT;
  diWL1, diWL2, diWL3              : DINT;
  diWg                      : DINT;
  diImaxL1, diImaxL2, diImaxL3         : DINT;
  diUmaxL1, diUmaxL2, diUmaxL3         : DINT;
  diPmaxL1, diPmaxL2, diPmaxL3         : DINT;
  diSg                      : DINT;
  diQg                      : DINT;
  dummy                     : DINT;
END_STRUCT
END_TYPE
```

6.37 ST_HVACPowerMeasurementEx

```
TYPE ST_HVACPowerMeasurementEx :
STRUCT
  fIL1, fIL2, fIL3              : LREAL;
  fIg                    : LREAL;
  fUL1, fUL2, fUL3                  : LREAL;
  fPL1, fPL2, fPL3                  : LREAL;
  fPg                       : LREAL;
  fCosPhiL1, fCosPhiL2, fCosPhiL3           : LREAL;
  fCosPhi                       : LREAL;
  fWL1, fWL2, fWL3                  : LREAL;
  fWg                       : LREAL;
  fImaxL1, fImaxL2, fImaxL3             : LREAL;
  fUmaxL1, fUmaxL2, fUmaxL3             : LREAL;
  fPmaxL1, fPmaxL2, fPmaxL3             : LREAL;
  fSg                       : LREAL;
  fQg                       : LREAL;
```

```
  fFrequencyL1, fFrequencyL2, fFrequencyL3      : LREAL;
END_STRUCT
END_TYPE
```

6.38 ST_HVACPowerRangeTable

```
TYPE ST_HVACPowerRangeTable :
STRUCT
iAggregate        : INT;
iAggregateStep    : INT;
rY_Max        : REAL;
rY_Min        : REAL;
rIntegralHigh     : REAL;
rIntegralLow      : REAL;
udiSecDelayHigh       : UDINT;
udiSecDelayLow    : UDINT;
bBlock        : BOOL;
END_STRUCT
END_TYPE
```

iAggregate: Parameterangabe in welche Ausgangsstruktur *stAggregate1-6* des Funktionsbausteins FB_HVACPowerRangeTable [▶ 281] die Variablen *rY_Min*, *rY_Max*, *iAggregateStep* und *bBlock* geschrieben werden.

iAggregateStep: Parameterangabe in welcher Stufe das angesprochene Aggregat fixiert oder regeln soll in Abhängigkeit von *bBlock*. Diese Variable wird über die Struktur *stAggregateX* ausgegeben.

rY_Max: Parameterangabe für stetige Aggregate. Die Variablenwert darf g_rY_Min [▶ 541] nicht unterschreiten und g_rY_Max [▶ 541] nicht überschreiten und *rY_Max* darf nicht kleiner sein als *rY_Min*. Ansonsten wird mit *bError* = TRUE ein Fehler angezeigt und die Abarbeitung des Funktionsbausteins FB_HVACPowerRangeTable [▶ 281] wird gestoppt. Diese Variable wird über die Struktur *stAggregateX* ausgegeben.

rY_Min: Parameterangabe für stetige Aggregate. Die Variablenwert darf g_rY_Min [▶ 541] nicht unterschreiten und g_rY_Max [▶ 541] nicht überschreiten und *rY_Max* darf nicht kleiner sein als *rY_Min*. Ansonsten wird mit *bError* = TRUE ein Fehler angezeigt und die Abarbeitung des Funktionsbausteins FB_HVACPowerRangeTable [▶ 281] wird gestoppt. Diese Variable wird über die Struktur *stAggregateX* ausgegeben.

rIntegralHigh: Positiver Wert für das obere Limit an dem die Integration des I-Übertragungsgliedes angehalten wird, siehe der VAR_IN_OUT-Variable *rIntegralHigh* im FB_HVACI_CtrlStep [▶ 255]. Diese Variable wird über die Struktur *stI_Ctrl* ausgegeben.

rIntegralLow: Positiver Wert für das untere Limit an dem die Integration des I-Übertragungsgliedes angehalten wird, siehe der VAR_IN_OUT-Variable *rIntegralHigh* im FB_HVACI_CtrlStep [▶ 255]. Diese Variable wird über die Struktur *stI_Ctrl* ausgegeben.

udiSecDelayHigh: Verzögerungszeit nach deren Ablauf das I-Übertragungsglied aktiviert wird, siehe der VAR_IN_OUT-Variable *udiSecDelayHigh* im FB_HVACI_CtrlStep [▶ 255]. Diese Variable wird über die Struktur *stI_Ctrl* ausgegeben.

udiSecDelayLow: Verzögerungszeit nach deren Ablauf das I-Übertragungsglied aktiviert wird, siehe der VAR_IN_OUT-Variable *udiSecDelayHigh* im FB_HVACI_CtrlStep [▶ 255]. Diese Variable wird über die Struktur *stI_Ctrl* ausgegeben.

bBlock: Ist *bBlock* = FALSE, so wird das angesprochene Aggregat in der vorgegebenen Stufe über *iAggregateStep* fixiert. Ist *bBlock* = TRUE, so wird die Regelung des angesprochenen Aggregats frei gegeben von der Ausstufe (0) bis zur vorgegebenen Stufe über iAggregateStep. Diese Variable wird über die Struktur *stAggregateX* ausgegeben.

6.39 ST_HVACTimeChannel

```
TYPE ST_HVACTimeChannel :
STRUCT
  uiOn_hh      : UINT;
  uiOn_mm      : UINT;
  uiOn_ss      : UINT;
  uiOff_hh     : UINT;
  uiOff_mm     : UINT;
  uiOff_ss     : UINT;
  bEnable      : BOOL;
  bAllDays     : BOOL;
  bMonday      : BOOL;
  bTuesday     : BOOL;
  bWednesday       : BOOL;
  bThursday    : BOOL;
  bFriday      : BOOL;
  bSaturday    : BOOL;
  bSunday      : BOOL;
  bResetAfterOn    : BOOL;
  bQ           : BOOL;
END_STRUCT
END_TYPE
```

6.40 ST_HVACTempChangeFunction

Struktur der Stützpunkteinträge für die Temperaturänderungs-Funktion in den Bausteinen FB_HVACOptimizedOn [▶ 439] und FB_HVACOptimizedOff [▶ 448].

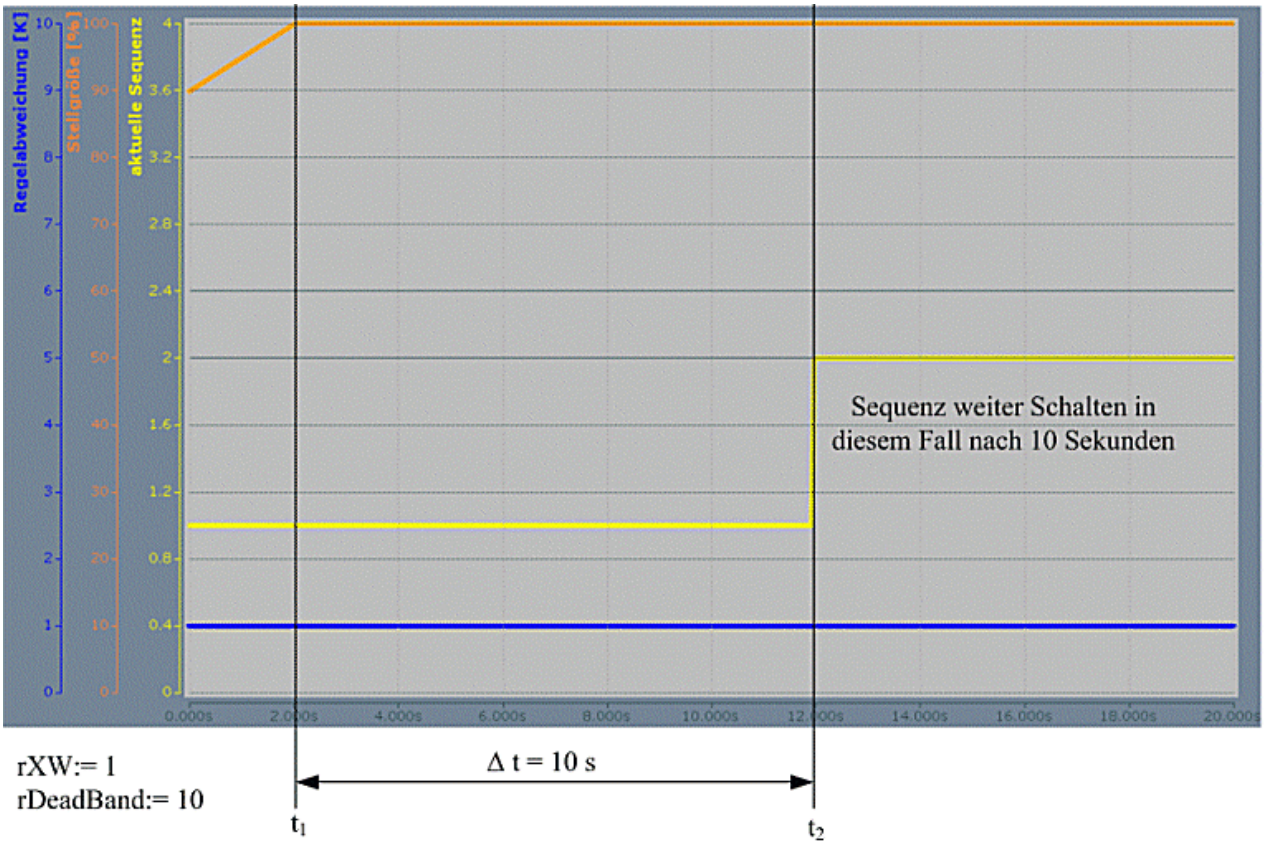
```
TYPE ST_HVACPrestartFunction :
STRUCT
   rOutsideTemp    : ARRAY[1..10] OF REAL; (*[degC]*)
   rRoomTempChange : ARRAY[1..10] OF REAL; (*[degK/min]*)
END_STRUCT
END_TYPE
```

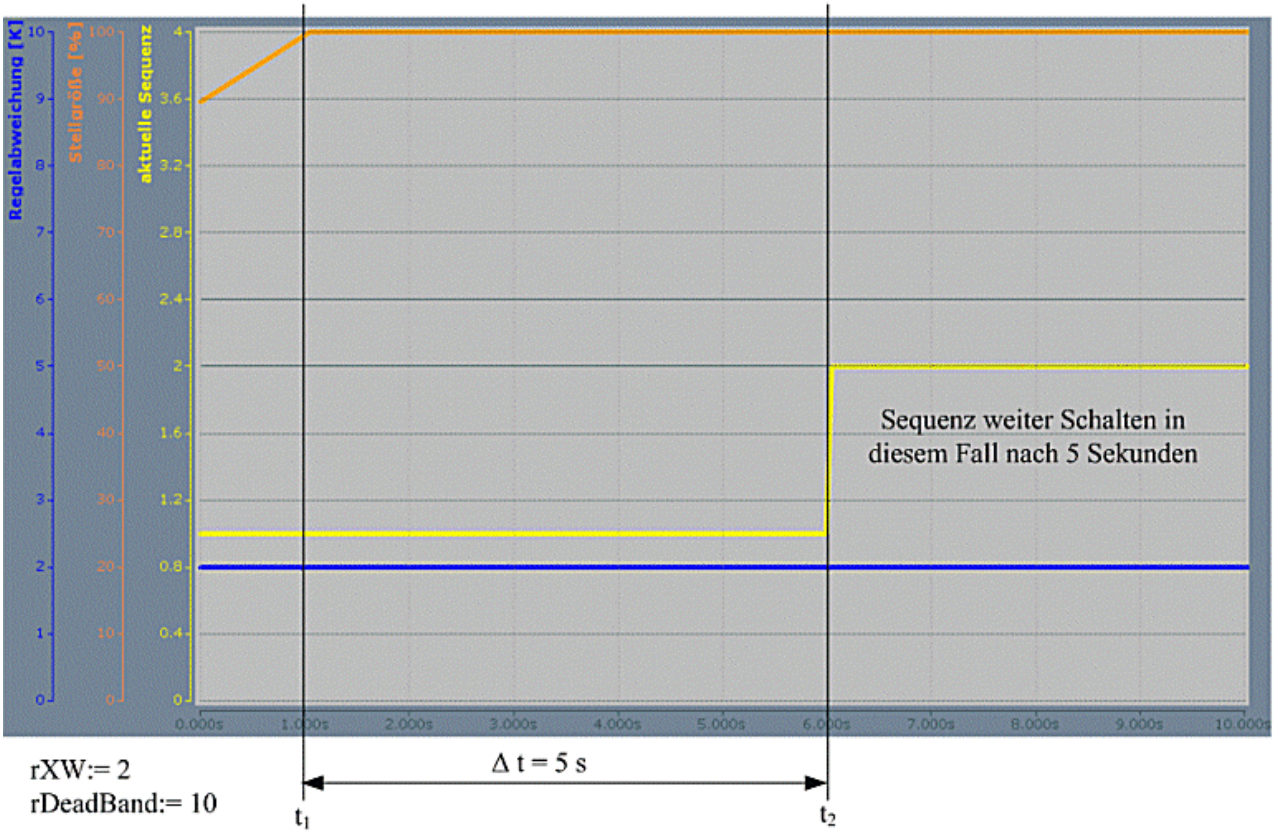
rOutsideTemp / rRoomTempChange : Die 10 Stützstellen (Außentemperatur in Grad Celsius zu Innentemperaturänderung in Grad Kelvin/Minute), welche übergeben werden.

7 Anhang

7.1 Berechnung Umschaltzeit bei Sequenzwechsel

Beispiele für die Ermittlung der Umschaltzeit von einer Sequenz auf die nächste Sequenz. Die Umschaltzeit ergibt sich aus dem Integral und der Regelabweichung.





7.2 Beispielprojekt

Download	Benötige Bibliothek

https://infosys.beckhoff.com/content/1031/tcplclibhvac/Resources/11659726219.zip 	**TcHVAC.lib**

7.3 VAR_GLOBAL CONSTANT

```
VAR_GLOBAL CONSTANT
    g_tHVACWriteBackupDataTime    : TIME := T#5s;
    g_udiMaxNoOfBytesInStruct     : UDINT := 16;     (* Size in number of bytes *)
    g_udiMaxSizeOfString          : UDINT := 256;    (* Size in number of characters(as in string de
claration) + 1 byte for termination*)
    g_iMaxNoOfScale_nPoint        : INT := 60;       (* FB_HVACScale_nPoint*)
    g_iMaxNumberOfSteps           : INT := 32;       (*FB_HVACI_CtrlStep/FB_HVACPowerRangeTable*)
    g_iMinNumberOfSteps           : INT := 0;        (*FB_HVACI_CtrlStep/FB_HVACPowerRangeTable*)
    g_iMaxNumberOfProfiles        : INT := 16;       (*FB_HVACPowerRangeTable*)
    g_iMinNumberOfProfiles        : INT := 1;        (*FB_HVACPowerRangeTable*)
    g_iMaxNumberOfAggregates      : INT := 6;        (*FB_HVACPowerRangeTable*)
    g_iMinNumberOfAggregates      : INT := 1;        (*FB_HVACPowerRangeTable*)
    g_rYMin               : REAL := -1000;       (*FB_HVACPowerRangeTable*)
    g_rYMax               : REAL := 1000;    (*FB_HVACPowerRangeTable*)
    g_iAggregateMinNumberOfSteps      : INT := 0;        (*FB_HVACPowerRangeTable*)
    g_iAggregateMaxNumberOfSteps      : INT := 6;        (*FB_HVACPowerRangeTable*)
    g_udiMaxSec               : UDINT := 4294967;    (*FB_HVACI_CtrlStep / FB_HVACPowerRangeTable*)
    g_iMinNumberOfSequences       : INT := 1;        (*FB_HVAC2PointCtrlSequence*)
    g_iMaxNumberOfSequences       : INT := 16;       (*FB_HVAC2PointCtrlSequence*)
    g_iNumOfCmdCtrl_8         : INT := 8;        (*FB_HVACCmdCtrl_8*)
END_VAR
```

7.4 Tabelle Betriebsarten Sequenzregler

Betriebsarten

Eine weitere Besonderheit der Sequenzregler ist Ihre Steuerung mit dem Enum E_HVACSequenceCtrlMode [▶ 531]
Mittels des Enums E_HVACSequenceCtrlMode [▶ 531]erfolgt nicht nur die Freigabe der Regelung, sondern auch die Übertragung der Betriebsart der Raumlufttechnischen Anlage an die Regelungsbausteine in der Sequenz. Damit reagiert jeder Sequenzregler in Abhängigkeit der Betriebsarten speziell auf die Werte des Enums E_HVACSequenceCtrlMode [▶ 531] wie in der Tabelle dargestellt.

Wert von: E_HVACSequenceCtrlMode	0 (Stop)	1 (On)	2 (Nachtkühlung)	3 (Stützbetrieb)	4 (Überhitzungsschutz)	5 (Nachtkühlung und Überhitzungsschutz)
FB_HVACMasterSequenceCtrl Führungsregler	gesperrt 0%	Freigabe	gesperrt 0%	Freigabe rY=ZuluftMax.Temp	Freigabe rY=ZuluftMin.Temp	Freigabe rY=ZuluftMin.Temp
FB_HVACPIDPreHeating Vorerhitzer	gesperrt 0%	Freigabe	gesperrt 0%	Freigabe 0%-100%	gesperrt 0%	gesperrt 0%
FB_HVACPIDReHeating Nacherhitzer	gesperrt 0%	Freigabe	gesperrt 0%	gesperrt 0%	gesperrt 0%	gesperrt 0%

Wert von: E_HVACSequenceCtrlMode	0 (Stop)	1 (On)	2 (Nachtkühlung)	3 (Stützbetrieb)	4 (Überhitzungsschutz)	5 (Nachtkühlung und Überhitzungsschutz)
FB_HVACPIDCooling Kühler	gesperrt 0%	Freigabe	gesperrt 0%	gesperrt 0%	Freigabe 0%-100%	Freigabe 0%-100%
FB_HVACPIDEnergyRecovery Wärmerückgewinnung	gesperrt 0%	Freigabe	gesperrt 0%	gesperrt 0%	gesperrt 0%	gesperrt 0%
FB_HVACPIDMixedAir Mischluftklappen	gesperrt 0%	Freigabe	max. Außenluftrate 100%	0% reine Umluft	0% reine Umluft	max. Außenluftrate 100%

Mehr Informationen:
www.beckhoff.de/ts8000

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