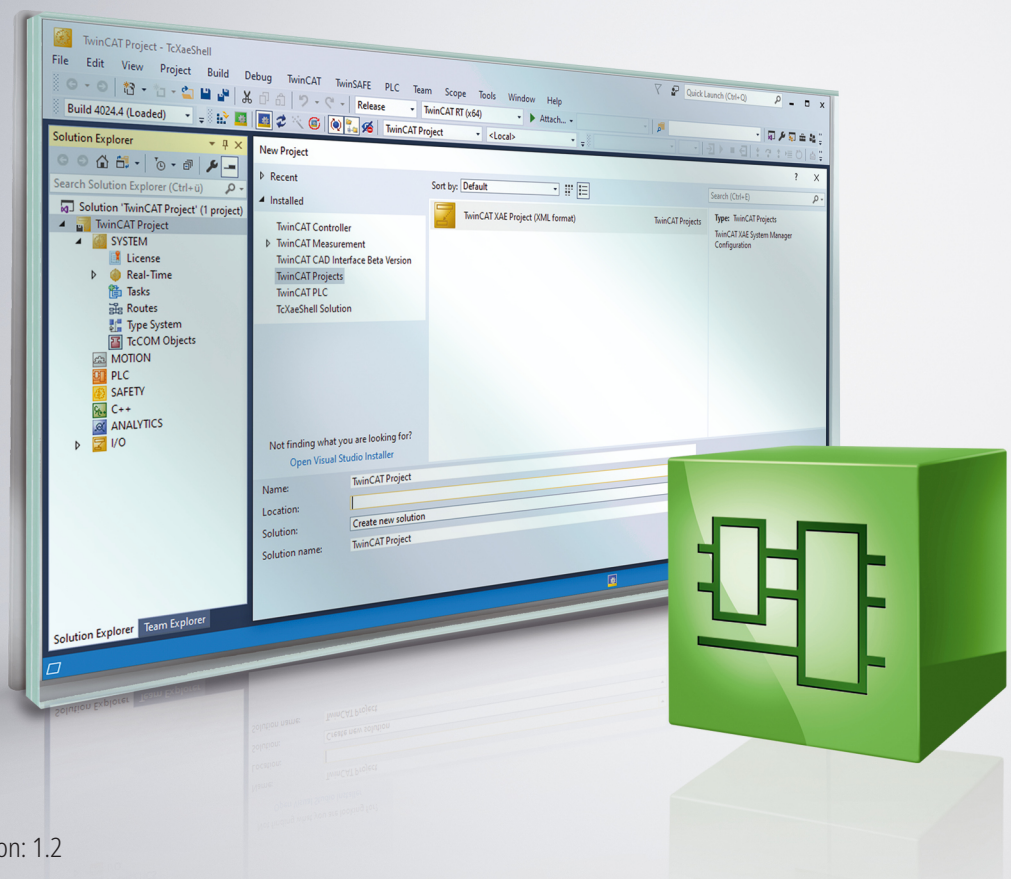
BECKHOFF New Automation Technology

Handbuch | DE

TE1000

TwinCAT 3 | PLC-Bibliothek: Tc3_BA_Common



15.12.2021 | 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®

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

Copyright

1.2 Sicherheitshinweise

Sicherheitsbestimmungen

Beachten Sie die folgenden Sicherheitshinweise und Erklärungen!
Produktspezifische Sicherheitshinweise finden Sie auf den folgenden Seiten oder in den Bereichen Montage, Verdrahtung, Inbetriebnahme usw.

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.

Erklärung der Symbole

In der vorliegenden Dokumentation werden die folgenden Symbole mit einem nebenstehenden Sicherheitshinweis oder Hinweistext verwendet. Die Sicherheitshinweise sind aufmerksam zu lesen und unbedingt zu befolgen!

⚠ GEFAHR

Akute Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht unmittelbare Gefahr für Leben und Gesundheit von Personen!

⚠ WARNUNG

Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht Gefahr für Leben und Gesundheit von Personen!

⚠ VORSICHT

Schädigung von Personen!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, können Personen geschädigt werden!

HINWEIS

Schädigung von Umwelt oder Geräten

Wenn der Hinweis neben diesem Symbol nicht beachtet wird, können Umwelt oder Geräte geschädigt werden.

Tipp oder Fingerzeig

Dieses Symbol kennzeichnet Informationen, die zum besseren Verständnis beitragen.

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

Die TwinCAT3 Building Automation Bibliothek (TC3 BA Common) stellt Reglerbausteine und einen Sequenz-Linker-Baustein zur Verfügung.
Diese Bausteine werden sowohl von der Bibliothek TC3_BA, als auch von der Bibliothek Tc2_BACnetRev12 verwendet.

3 Allgemeine Informationen

Weitere erforderliche Bibliotheken

Für PC-Systeme und Embedded-PCs (CXxxxx):

- Tc2_IoFunctions
- Tc2_Standard
- Tc2_System
- Tc2_Utilities

4 Programmierung

4.1 POUs

4.1.1 Regler

Funktionsbausteine

Name	Beschreibung
FB_BA_SeqCtrl [▶ 16]	Sequenzregler (siehe Einleitung Sequenzregler [▶ 10]).
FB_BA_SeqLink [▶ 20]	Sequenzregler-Steuerbaustein.
FB_BA_PIDCtrl [▶ 13]	Universeller PID-Regler.

4.1.1.1 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 drei Stellorgane beteiligt. Im Projekt wird für jedes dieser Stellorgane ein eigener Sequenzregler instanziiert.

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 *lrY*.

Wenn die Wirkung des aktiven Stellgliedes (Reglers) bei dem Erreichen einer Endlage nicht ausreicht, wird von dem aktiven Regler auf den links oder rechts benachbarten Regler umgeschaltet.
Dieser übernimmt damit die Regelung. Der zuvor aktive Regler verharrt je nach Wirksinn in der Endlage von *lrYMax* oder *lrYMin*. 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 Erhöhung der Zuluft Temperatur ermöglicht, ganz rechts das Stellglied, das die größtmögliche Verringerung der Zuluft Temperatur bewirkt.

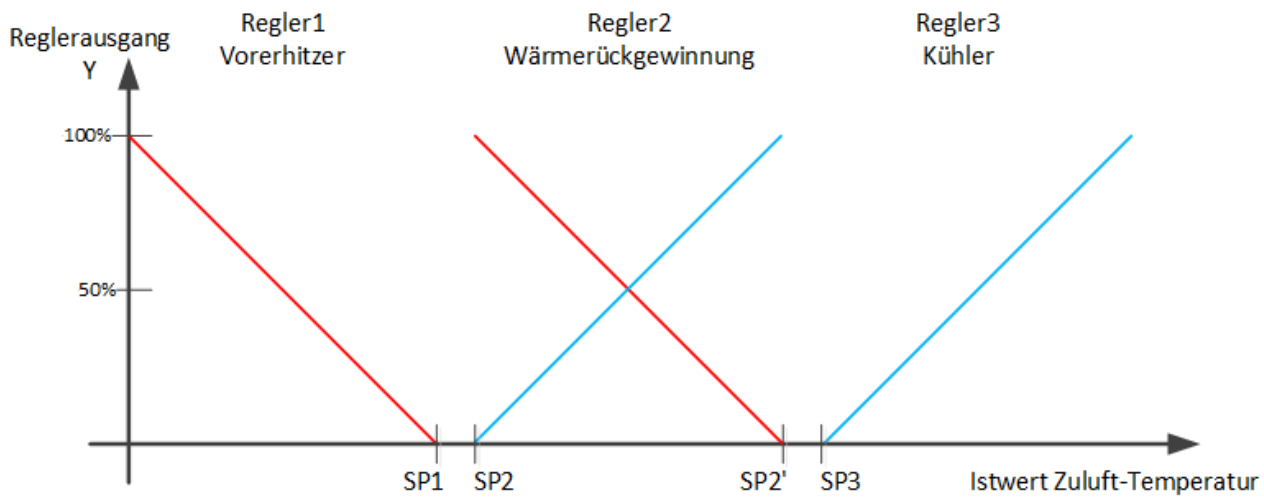
Manche Stellglieder, wie z.B. eine Umluftklappe oder eine Wärmerückgewinnung wechseln während des Betriebs ihre Wirkrichtung. (indirekt = heizen, direkt = kühlen)

Stellglieder mit wechselnden Wirksinn, wie z.B. Außenluft- und Umluftklappe oder Wärmerückgewinnung, werden nur einmal aufgeführt.

- 1: Vorerhitzer Regler
- 2: Mischluft Regler
- 3: Kühler

Schematische Darstellung

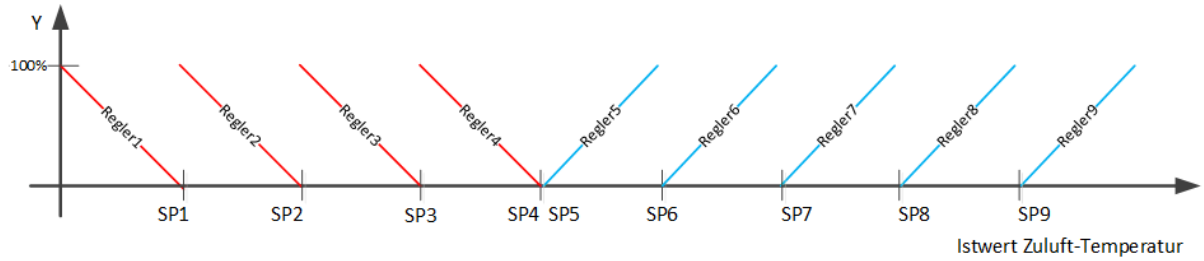
Diese Anlage wird schematisch wie folgt dargestellt:



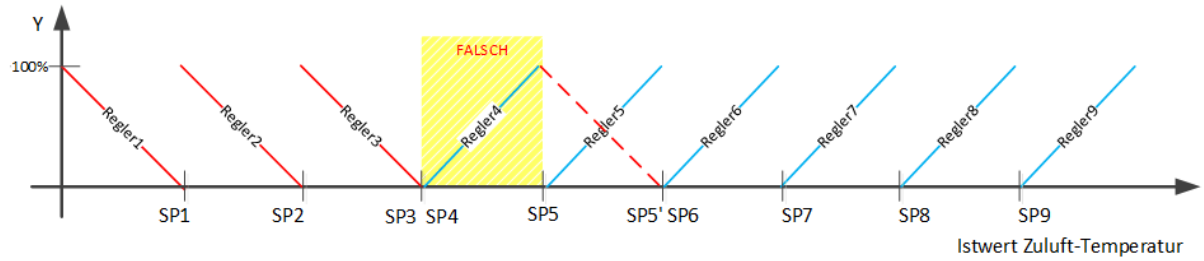
Regeln für die Aufstellung einer Sequenz

Für die Aufstellung der Sequenzen sind folgende Regeln zu beachten, wobei thematisch auf eine Zuluftregelung Bezug genommen wird:

- Die Nummerierung der Sequenzregler erfolgt von den Heizsequenzen mit niedrigen Ordnungszahlen hin zu den Kühlsequenzen mit hohen Ordnungszahlen kontinuierlich steigend.



- In eine Folge von Heizsequenzen sollte keine Kühlsequenz eingebunden werden. Ebenfalls sollte in eine Folge von Kühlsequenzen keine Heizsequenz eingebunden werden. Sequenzen mit einer Wirksinnumkehr für ein Mischluftsystem oder eine Wärmerückgewinnung sind zwischen den Heiz- und den Kühlsequenzen zu positionieren.



  In diesem Bild wäre Regler 4 falsch platziert, wenn Regler 5 in den Heizbetrieb wechseln würde. Oder: Regler 4 ist richtig, Regler 5 müsste jedoch ein reiner Kühlregler sein.
  In beiden Fällen wären 2 Wechsel von Heizen nach Kühlen vorhanden.

- Die Sollwerte innerhalb der Sequenz müssen monoton steigend sein. Diese Forderung ergibt sich aus dem oben erläuterten Umschaltverhalten: Ist der Sollwert eines niedrigeren Reglers höher als der des nächst höheren, so kann es zu einem ständigen Hin- und Herschalten zwischen zwei Reglern führen. Wie oben bereits erwähnt, haben jedoch Regler mit gleichem Wirksinn üblicherweise den gleichen Sollwert.
  $SP1 \leq SP2 \leq SP3 \leq SP4 \leq SP5 \leq SP6 \leq SP7 \leq SP8 \leq SP9$

Sequenzregler in der PLC

Für die Realisierung einer Sequenzregelung im SPS-Programm stellt die *TC3_BA_Common Bibliothek* zwei Bausteine zur Verfügung:

Der Baustein FB_BA_SeqCtrl [▶ 16]: Er stellt einen einzelnen Regler als Bestandteil einer Sequenz von maximal 16 Reglern dar.

Der Baustein FB_BA_SeqLink [▶ 20]: Dieser Baustein ist der Kontrollbaustein der Sequenz und existiert demnach pro Sequenz nur einmal. Er entscheidet, welcher Regler der Sequenz gerade aktiv ist und überprüft die Sequenz auf bestimmte Fehlerzustände, wie beispielsweise die doppelte Vergabe von Ordnungszahlen an den Reglern.

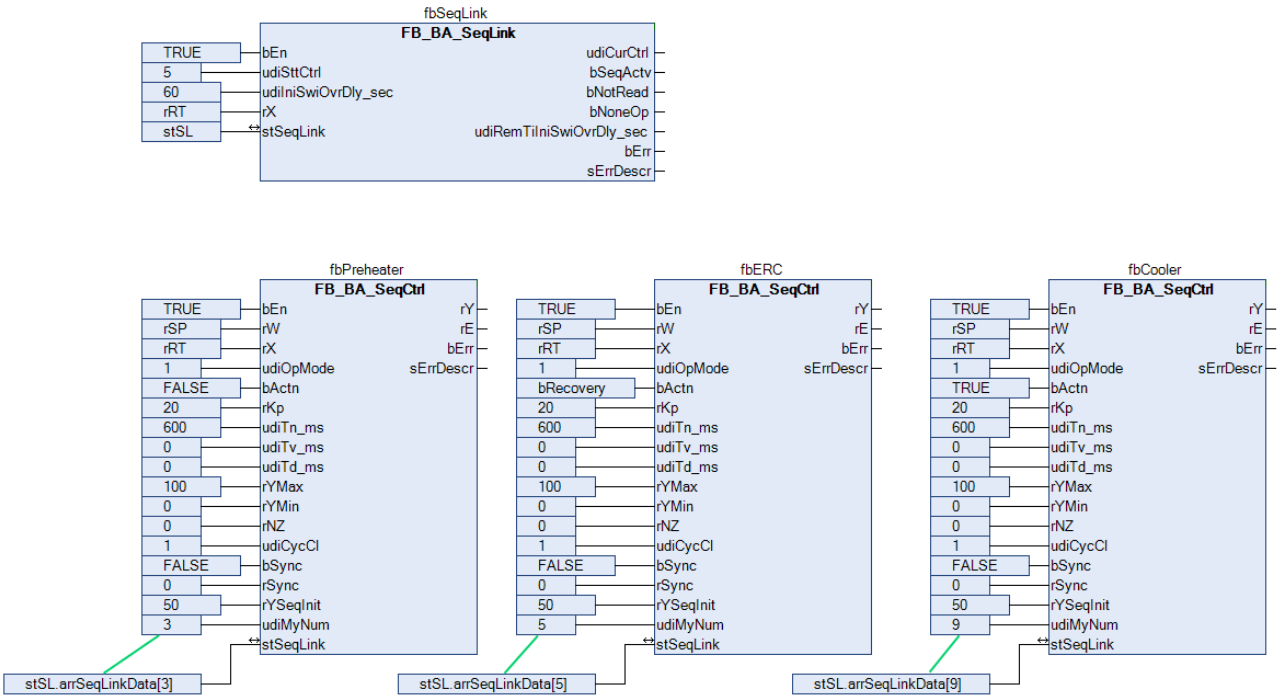
Die Strukturvariable ST_BA_SeqLink [▶ 40] dient zur Verbindung der Sequenzregler mit dem Sequenzlinker FB_BA_SeqLink [▶ 20].

Diese Strukturvariable ist pro Sequenzregelung einmal zu deklarieren.

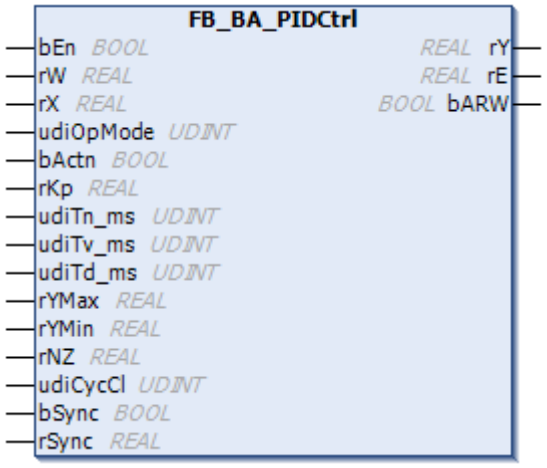
Die Freigabe der Sequenzregelung erfolgt am Eingang *bEn* des Funktionsbausteins FB_BA_SeqLink [▶ 20]. Mit der Variablen *udiSttCtrl* wird bestimmt, mit welchem Regler nach dem Start des Regelbetriebs begonnen wird zu regeln. Im Beispiel wird mit dem Sequenzregler mit der Nr. 5 gestartet. Das Umschalten von dem Regler 5 auf einen anderen Regler in der Sequenz ist nach dem Neustart der Regelung um den Wert der Eingangsvariable *udiIniSwiOvrDly_sec* gesperrt.

```
PROGRAM Sequence
VAR
    fbSeqLink    :   FB_BA_SeqLink;
    fbPreheater  :   FB_BA_SeqCtrl;
    fbERC        :   FB_BA_SeqCtrl;
    fbCooler     :   FB_BA_SeqCtrl;

    stSL         :   ST_BA_SeqLink;
END_VAR
```



4.1.1.2 FB_BA_PIDCtrl



Universeller PID-Regler, wahlweise in Parallelstruktur oder mit vorgelagertem P-Glied.

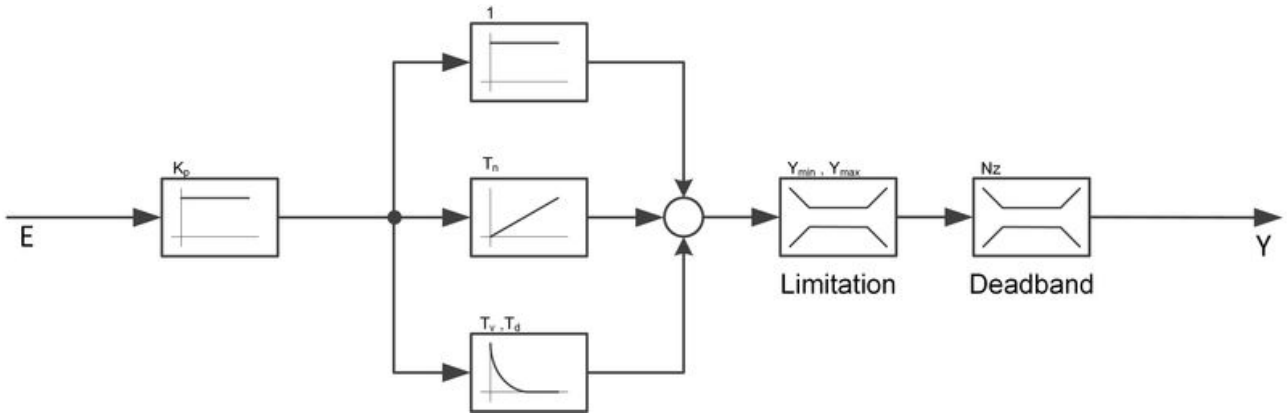
Funktionsbeschreibung

Dieser Regler ist intern in zwei aufeinander folgende Teile gegliedert:

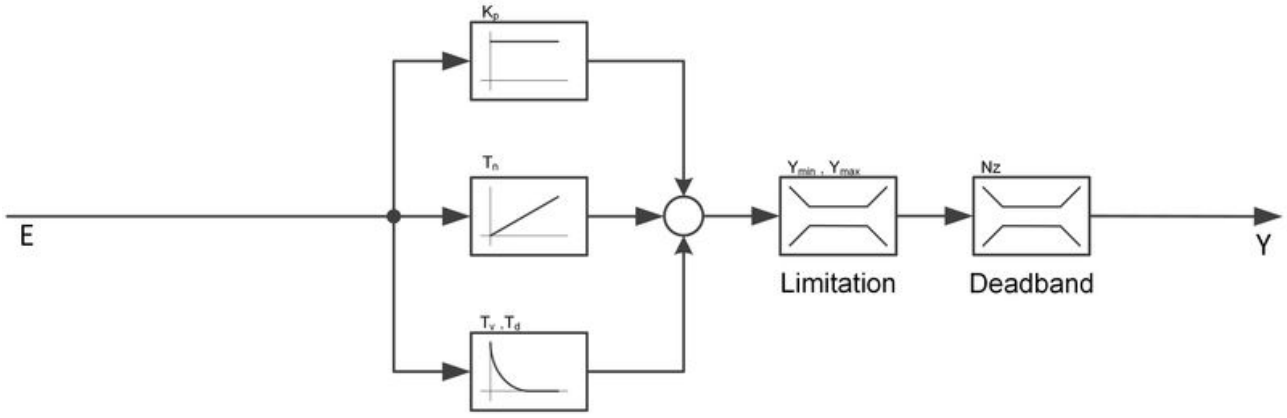
- der Regler selbst, in den unten aufgeführten Wirkungsplänen als P-, I- und D-Anteil dargestellt mit einer Ausgangsbegrenzung (Limitation).
- einem Totbandglied (Deadband), welches die Ausgangsänderungen des Reglers mit einer Hysterese beaufschlagt.

Wirkunsplan

udiMode=0 (P-Anteil vorgelagert):



udiMode=1 (Parallelstruktur):



Passiv-Verhalten (bEn = FALSE)

Die Ausgänge werden wie folgt gesetzt:

rY	0,0
rE	0,0
bARW	FALSE

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 ohne Vergangenheitswerte berechnet.

Aktiv-Verhalten (bEn = TRUE)

Im ersten Zyklus werden I- und D-Anteil wie bereits erwähnt ohne Vergangenheitswerte berechnet und somit "sauber" aufgestartet.

Synchronisation

Ein positives Signal an *bSync* setzt den I-Anteil so, dass die Stellgröße den Wert *rSync* annimmt. Diese Methode kann, wenn *bEn* und *bSync* 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 *bSync* ausgewertet wird, da es sich um eine Setz-Aktion handelt. Für ein erneutes Synchronisieren, etwa auf einen Übergabewert, muss am Eingang *bSync* ein erneutes TRUE-Signal angelegt werden.

Anti-Reset-Windup

Ist der I-Anteil aktiv, so sorgt der Regler dafür, dass dieser festgehalten wird, sollte der Reglerausgang *rY* über die Grenzen *rYMin* oder *rYMax* hinausgehen wollen. Innerhalb des Reglers wird in jedem Zyklus eine Vorberechnung des Regler Ausgangs gemacht. Ist diese kleiner als die untere Ausgangsgrenze *rYMin* oder größer als die obere Grenze *rYMax*, so wird der I-Anteil derart angepasst, dass die Summe aus P-, I- und D-Anteil *rYMin* bzw. *rYMax* ergibt. Damit ist gewährleistet, 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 Integralanteil zunächst noch abgebaut werden muss.

Wirksinn

Mit *bActn* = 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:

bActn	rXW (Regelabweichung)	Wirksinn
TRUE	*rX-rW* (Istwert-Sollwert)	direkt (Kü*hlen)*
FALSE	*rW-rX* (Sollwert-Istwert)	indirekt (Heizen)

Neutrale Zone

Ein Wert von *rNZ* > 0.0 gibt die Funktion der neutralen Zone (Deadband) frei. Ein Wert gleich Null deaktiviert das Totbandglied und die Werte am Eingang werden direkt durchgereicht.

Ist im aktiven Fall die Änderung am Eingang des Gliedes *rYin* in einem SPS-Zyklus im Vergleich zum vorhergehenden SPS-Zyklus kleiner als *rNZ*/2, so wird der Ausgang auf dem Wert des vorhergehenden Zyklus gehalten, bis die Änderung größer oder gleich *rNZ*/2 ist.

Beispiel: *rNZ*=1, *rYin*=55.0, *rY*=55.0

PLC-Zyklus+1	*rYin*=55.2	*rY*=55.0
PLC-Zyklus+2	*rYin*=55.3	*rY*=55.0
PLC-Zyklus+3	*rYin*=55.1	*rY*=55.0
PLC-Zyklus+4	*rYin*=55.6	*rY*=55.6
PLC-Zyklus+5	*rYin*=55.4	*rY*=55.6
PLC-Zyklus+6	*rYin*=55.3	*rY*=55.6
PLC-Zyklus+7	*rYin*=55.1	*rY*=55.1

Durch diese Funktion sollen unnötig viele Stellimpulse vermieden werden.

VAR_INPUT

```
bEn          : BOOL;
rW           : REAL;
rX           : REAL;
udiOpMode    : UDINT;
bActn        : BOOL;
rKp          : REAL;
udiTn_ms     : UDINT;
udiTv_ms     : UDINT;
udiTd_ms     : UDINT;
rYMax        : REAL;
rYMin        : REAL;
rNZ          : REAL;
udiCycCl     : UDINT;
bSync        : BOOL;
rSync        : REAL;
```

bEn: Regleraktivierung.

rW: Sollwert.

rX: Istwert.

udiOpMode: *udiMode*=0: Regler mit vorgelagertem P-Anteil, *udiMode*=1: Regler in Parallelstruktur. Intern begrenzt auf die Werte 0 und 1.

bActn: Wirksinn [▶ 14] des Reglers.

rKp: Reglerverstärkung. Wirkt nur auf den P-Anteil. Intern begrenzt auf einen Minimalwert von 0.

udiTn_ms: Nachstellzeit des I-Anteiles [ms]. Ein Nullwert an diesem Parameter schaltet den I-Anteil ab. Intern begrenzt auf einen Minimalwert von 0.

udiTv_ms: Vorhaltezeit des D-Anteiles [ms]. Ein Nullwert an diesem Parameter schaltet den D-Anteil ab. Intern begrenzt auf einen Minimalwert von 0.

udiTd_ms: Dämpfungszeit des D-Anteiles [ms]. Intern begrenzt auf einen Minimalwert von 0.

rYMax: Obere Ausgabebegrenzung des Reglers. Wählbarer Bereich: 0..100%.

rYMin: Untere Ausgabebegrenzung des Reglers [%]. Wählbarer Bereich: 0..100%. Der Wert *rYMin* wird nach oben hin durch *rYMax* begrenzt.

rNZ: neutrale Zone.

udiCycCl: Aufrufzyklus des Bausteines als Vielfaches der Zykluszeit. Intern begrenzt auf einen Minimalwert von 1.

Beispiel: *tTaskCycleTime* = 20ms, *udiCtrlCycleCall* =10 -> Der Regelalgorithmus wird alle 200ms aufgerufen. Damit werden aber auch nur alle 200ms die Ausgänge aktualisiert.

bSync / rSync: Synchronisationsbefehl: Ausgangswert *rY* auf *rSync* setzen. Der Wert *rSync* wird intern begrenzt auf Werte von *rYMin* bis *rYMax*.

VAR_OUTPUT

```
rY          : REAL;
rE          : REAL;
bARW        : BOOL;
```

rY: Stellgröße. Bereich durch *rYMin* und *rYMax* eingeschränkt.

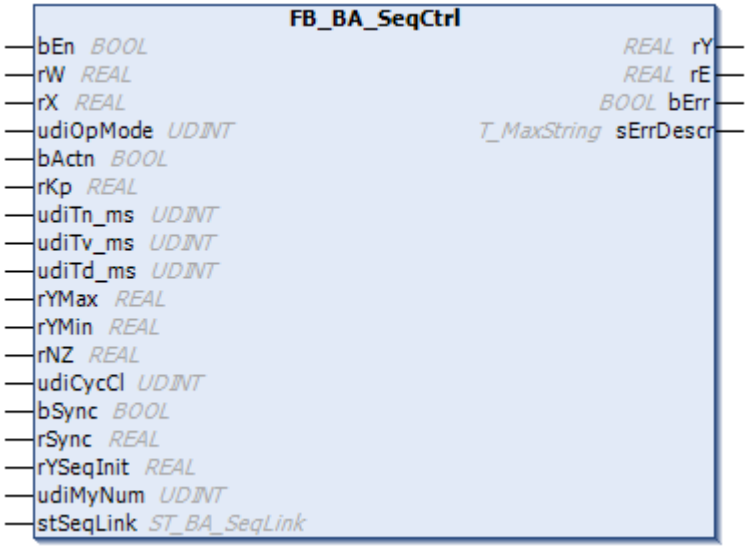
rE: Regelabweichung (Die Berechnung ist abhängig vom Wirksinn [▶ 14]).

bARW: Anti-Reset-Windup-Funktion ist aktiv.

Voraussetzungen

Entwicklungsumgebung	Erforderliche Bibliothek	Erforderliche Function
TwinCAT3.1 4022.16	Tc3Building Automation Common ab V1.0.4.3	TF8040 | TwinCAT Building Automation ab V1.0.5.0

4.1.1.3 FB_BA_SeqCtrl

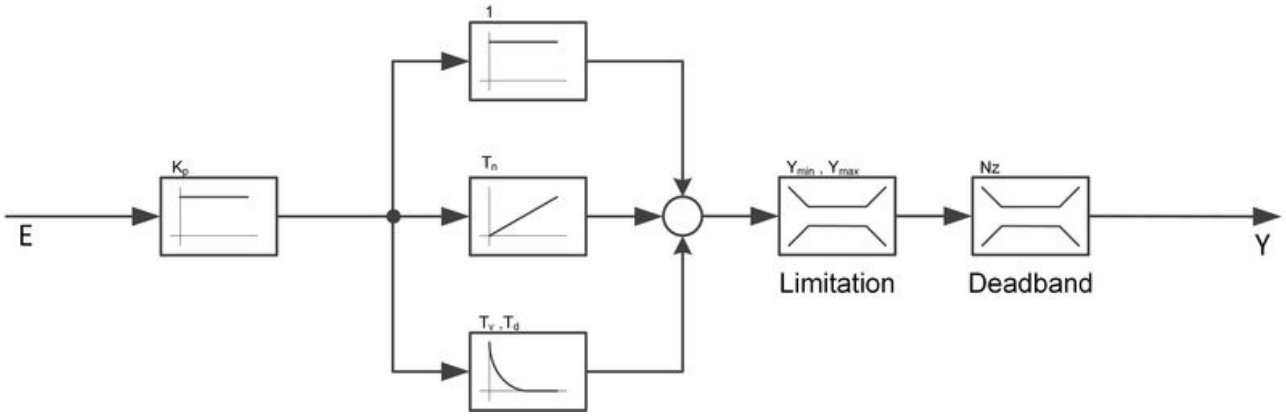


PID-Regler als Bestandteil einer Sequenz.

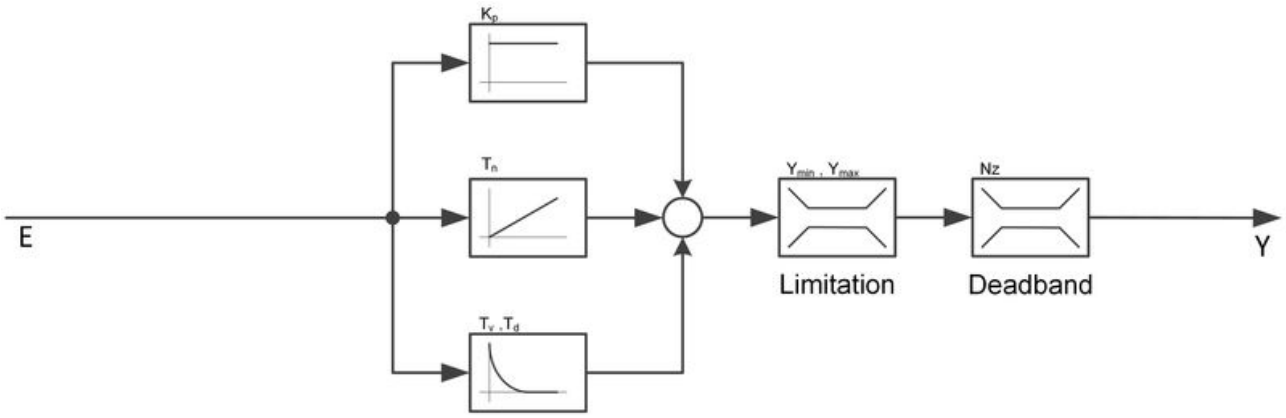
Funktionsbeschreibung

Dieser Regler ist in seinen Funktionalitäten identisch mit dem FB_BA_PIDCtrl [▶ 13].

udiOpMode=0 (P-Anteil vorgelagert)



udiOpMode=1 (Parallelstruktur)



Zudem wird der Regler, wenn er durch *bEn*=TRUE freigegeben ist, über eine übergeordneten Kontrollbaustein FB_BA_SeqLink [▶ 20] gesteuert.

Der Datenaustausch zwischen dem Steuerbaustein FB_BA_SeqLink [▶ 20] und den Sequenzreglern *FB_BA_SeqCtrl* erfolgt über die Strukturvariable stSeqLink [▶ 40].

Heiz - Kühlfolge

Die Reglersequenz ist so zu konzipieren, dass die Sequenzregler mit niedriger Ordnungszahl zum Heizen und die mit höherer zum Kühlen zu nutzen sind. Dabei ist nur ein Wechsel zulässig:

- Sequenzregler n (*udiMyNum*=n, *bActn*=TRUE)
- Sequenzregler n+1 (*udiMyNum* =n+1, *bActn*=FALSE)

Möglich ist aber auch die ausschließliche Programmierung von Kühl- bzw. Heizreglern.

Eine Parametrierung, die dieser Konvention widerspricht wird erkannt und als Fehler am Steuerbaustein FB_BA_SeqLink [▶ 20] angezeigt.

Reglerausgabe

Der Steuerbaustein FB_BA_SeqLink [▶ 20] gibt vor, welcher Sequenzregler aktiv ist. Innerhalb der einzelnen Sequenzregler wird ermittelt was am jeweiligen Steuerausgang *rY* ausgegeben wird. Über die In-Out-Variable *stSeqLink* erhält jeder Regler die Information über den Zustand der anderen Regler und wertet intern 4 Fälle aus.

1. Keiner der Sequenzregler ist freigegeben, sei es durch fehlende Freigaben (*bEn*) am Eingang oder durch einen erkannten Fehler am Steuerbaustein FB_BA_SeqLink [▶ 20]
   -> Die internen PID-Regler sind inaktiv und geben am Steuerausgang *rY*=0.0 aus.
2. Der Sequenzregler hat eine Freigabe und ist vom Steuerbaustein FB_BA_SeqLink [▶ 20] aktiv gesetzt.
   -> Der interne PID-Regler ist aktiv. Dessen Ausgangssignal wird am Steuerausgang *rY* ausgegeben.
3. Der Sequenzregler hat eine Freigabe, jedoch ist ein Sequenzregler **höherer** Ordnungszahl vom Steuerbaustein FB_BA_SeqLink [▶ 20] aktiv gesetzt
   -> Der interne PID-Regler ist inaktiv. Befindet sich der Sequenzregler im Heizbetrieb (*bActn*=FALSE), so wird er seinen Minimalwert *rYMin* am Steuerausgang *rY* ausgeben. Befindet er sich hingegen im Kühlbetrieb (*bActn*=TRUE), so gibt er den Maximalwert *rYMax* am Steuerausgang *rY* aus.
4. Der Sequenzregler hat eine Freigabe, jedoch ist ein Sequenzregler **niedrigerer** Ordnungszahl vom Steuerbaustein FB_BA_SeqLink [▶ 20] als aktiv gesetzt
   -> Der interne PID-Regler ist inaktiv. Befindet sich der Sequenzregler im Heizbetrieb (*bActn*=FALSE), so wird er seinen Maximalwert *rYMax* am Steuerausgang *rY* ausgeben. Befindet er sich hingegen im Kühlbetrieb (*bActn*=TRUE), so gibt er seinen Minimalwert *rYMin* am Steuerausgang *rY* aus.

Synchronisation

Wird ein Sequenzregler von der übergeordneten Steuerung aktiv geschaltet, so erfolgt immer eine Synchonisation, d.h. der Regler beginnt mit einem festen Startwert am Ausgang *rY*. Dabei werden 3 Fälle unterschieden:

1. Die gesamte Sequenzregleung wurde gerade über den Eingang *bEn* der übergeordneten Steuerung FB_BA_SeqLink [▶ 20] eingeschaltet. Der Regler, mit der Ordnungszahl *udiSttCtrl* am Eingang von FB_BA_SeqLink [▶ 20] ist der Startregler.
   -> Der Sequenzregler wird mit dem Wert, der an seinem Eingang *rYSeqInit* eingetragen ist, synchronisiert.
2. Der Sequenzregler, der gerade aktiviert wurde, hatte einen Regler höherer Ordnungszahl als "Vorgänger"
   -> Befindet sich der Sequenzregler im Heizbetrieb (*bActn*=FALSE), so wird er mit seinem Minimalwert *rYMin* synchronisiert. Befindet er sich hingegen im Kühlbetrieb (*bActn*=TRUE), so ist der Synchronisationswert sein Maximalwert *rYMax*.
3. Der Sequenzregler, der gerade aktiviert wurde, hatte einen Regler niedrigerer Ordnungszahl als "Vorgänger"
   -> Befindet sich der Sequenzregler im Heizbetrieb (*bActn*=FALSE), so wird er mit seinem Maximalwert *rYMax* synchronisiert. Befindet er sich hingegen im Kühlbetrieb (*bActn*=TRUE), so ist der Synchronisationswert sein Minimalwert *rYMin*.

Jeder Sequenzregler kann auch über Wertvorgabe *rSync* und Aktivierung *bSync* extra synchronisiert werden, wenn er gerade von der übergeordneten Steuerung aktiv gesetzt ist. Ein ständiges TRUE-Signal am Eingang *bSync* (etwa durch Versehen) wird intern durch Flankenbildung abgefangen und behindert dadurch nicht die oben beschriebene Synchronisation bei Aktiv-Setzung.

Startverhalten

Um ein "vernünftiges" Einregeln der gesamten Regelsequenz zu ermöglichen, wird der Startregler mindestens für die Zeit *udiIniSwiOvrDly_sec* [s], welche am Baustein FB_BA_SeqLink [▶ 20] eingetragen ist, aktiv gehalten, d.h. in dieser Zeit findet kein Umschalten auf eine anderen Regler dieser Sequenz statt. Der Ausgang *rY* des Startreglers wird **einmalig** auf seinen Wert *rYSeqInit* synchronisiert.

VAR_INPUT

```
bEn             : BOOL;
rW              : REAL;
rX              : REAL;
udiOpMode       : UDINT;
bActn           : BOOL;
rKp             : REAL;
udiTn_ms        : UDINT;
udiTv_ms        : UDINT;
udiTd_ms        : UDINT;
rYMax           : REAL;
rYMin           : REAL;
rNZ             : REAL;
udiCycCl        : UDINT;
bSync           : BOOL;
rSync           : REAL;
rYSeqInit       : REAL;
udiMyNum        : UDINT;
```

bEn: Aktivierung des Sequenzreglers

rW: Sollwert

rX: Istwert

udiOpMode: *udiOpMode*=0: Regler mit vorgelagertem P-Anteil, *udiOpMode* =1: Regler in Parallelstruktur. Intern sind die Werte auf 0 und 1 begrenzt.

bActn: Wirksinn-Umkehrung des Reglers. Bei Heiz-Kühleinsatz: *bActn*=FALSE entspricht Heizbetrieb, *bActn*=TRUE entspricht Kühlbetrieb.

rKp: Reglerverstärkung. Wirkt nur auf den P-Anteil. Intern begrenzt auf einen Minimalwert von 0.

udiTn_ms: Nachstellzeit des I-Anteiles [ms]. Ein Nullwert an diesem Parameter schaltet den I-Anteil ab. Intern begrenzt auf einen Minimalwert von 0.

udiTv_ms: Vorhaltezeit des D-Anteiles [ms]. Ein Nullwert an diesem Parameter schaltet den D-Anteil ab. Intern begrenzt auf einen Minimalwert von 0.

udiTd_ms: Dämpfungszeit des D-Anteiles [ms]. Intern begrenzt auf einen Minimalwert von 0.

rYMax: Obere Ausgabebegrenzung des Reglers [%]. Wählbarer Bereich: 0..100%.

rYMin: Untere Ausgabebegrenzung des Reglers [%]. Wählbarer Bereich: 0..100%. Der Wert *lrMin* wird nach oben hin durch *lrYMax* begrenzt.

rNZ: neutrale Zone (siehe Diagramm Deadband). Intern begrenzt auf einen Minimalwert von 0. Wirkungsweise wie beim FB_BA_PID_Ctrl [▶ 13].

udiCycCl: Aufrufzyklus des Bausteines als Vielfaches der Zykluszeit. Intern begrenzt auf einen Minimalwert von 1.
Beispiel: *tTaskCycleTime* = 20ms, *udiCycCl* =10 -> Der Regelalgorithmus wird alle 200ms aufgerufen. Damit werden aber auch nur alle 200ms die Ausgänge aktualisiert.

bSync / rSync: Synchronisationsbefehl: Ausgangswert *rY* auf *rSync* setzen. Der Wert *rSync* wird intern begrenzt auf Werte von *rYMin* bis *rYMax*.

rYSeqInit: Startwert des Reglers nach dem Neustart der gesamten Regelsequenz.

udiMyNum: Ordnungszahl des Sequenzreglers. Intern begrenzt auf Werte von 0 bis *gBA_cMaxSeqCtrl*.

VAR_OUTPUT

```
rY         : REAL;
rE         : REAL;
bErr       : BOOL;
sErrDescr  : T_MAXSTRING;
```

rY: Stellgröße. Bereich: 0..100%, soweit nicht weiter durch *rYMin* und *rYMax* eingeschränkt.

rE: Regelabweichung (Die Berechnung ist abhängig vom Wirksinn [▶ 14]).

bErr: Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind.

sErrDescr: Enthält die Fehlerbeschreibung.

Fehlerbeschreibung
01: Fehler: Die Reglerordnungszahl *udiMyNum* ist doppelt vergeben
02: Fehler: Die Reglerordnungszahl *udiMyNum* des freigegebenen Reglers ist 0. Das ist nur für nicht verwendete und damit nicht freigegebene Regler erlaubt.

VAR_IN_OUT

```
stSeqLink : ST_BA_SeqLink;
```

stSeqLink: Daten- und Befehlsstruktur (siehe ST_BA_SeqLink / ST_BA_SeqLinkData [▶ 40]) zwischen den einzelnen Sequenzreglern und dem Steuerbaustein FB_BA_SeqLink [▶ 20].

Tragen mehrere Sequenzregler dieselbe Nummer (*diMyNum*), so wird dies erkannt und als Fehler sowohl am Sequenzregler als auch am Steuerbaustein FB_BA_SeqLink [▶ 20] ausgegeben.

Voraussetzungen

Entwicklungsumgebung	Erforderliche Bibliothek	Erforderliche Function
TwinCAT3.1 4022.16	Tc3Building Automation Common ab V1.0.4.3	TF8040 | TwinCAT Building Automation ab V1.0.5.0

4.1.1.4 FB_BA_SeqLink

Dieser Baustein stellt die übergeordnete Steuereinheit dar, die vorgibt, welcher Sequenzregler gerade aktiv ist.

Der Datenaustausch zwischen dem Steuerbaustein FB_BA_SeqLink und den Sequenzreglern FB_BA_SeqCtrl [▶ 16] erfolgt über die Strukturvariable stSeqLink [▶ 40].

Funktionsbeschreibung

Startverhalten

Ein TRUE-Signal am Eingang *bEn* aktiviert die komplette Sequenzregelung. Der Baustein wird zunächst denjenigen Sequenzregler aktiv schalten, welcher an *udiSttCtrl* benannt ist. Alle anderen Sequenzregler richten ihren Ausgabewert nach der Rangordnung des aktiven Reglers, siehe FB_BA_SeqCtrl [▶ 16]. Der Startregler wird beim Sequenz-Start einmalig auf seinen Wert *rSync* gesetzt.
Um ein "vernünftiges" Einregeln der gesamten Regelsequenz zu ermöglichen, wird der Startregler mindestens für die Zeit *udiIniSwiOvrDly_sec* [s] lang aktiv gehalten, d.h. in dieser Zeit findet kein Umschalten auf einen anderen Regler dieser Sequenz statt.

Umschaltverhalten

Erreicht ein Sequenzregler seinem Maximal- oder Minimalwert, so wird je nach Regler-Wirksinn auf den nächsten Regler in der Rangfolge umgeschaltet, sofern der Istwert den Sollwert des nächsten Reglers unterschreitet bzw. überschreitet.
Dabei werden 4 Fälle unterschieden:

- Der noch aktive Regler hat direkten Wirksinn (Kühlen) und befindet sich auf seinem Maximalwert: Es wird in der Rangfolge der nächsthöhere Regler gewählt, wenn der Istwert den Sollwert dieses Reglers überschreitet.
- Der noch aktive Regler hat direkten Wirksinn (Kühlen) und befindet sich auf seinem Minimalwert: Es wird in der Rangfolge der nächst niedrigere Regler gewählt, wenn der Istwert den Sollwert dieses Reglers unterschreitet.
- Der noch aktive Regler hat indirekten Wirksinn (Heizen) und befindet sich auf seinem Maximalwert: Es wird in der Rangfolge der nächst niedrigere Regler gewählt, wenn der Istwert den Sollwert dieses Reglers unterschreitet.
- Der noch aktive Regler hat indirekten Wirksinn (Heizen) und befindet sich auf seinem Minimalwert: Es wird in der Rangfolge der nächsthöhere Regler gewählt, wenn der Istwert den Sollwert dieses Reglers überschreitet.

Abschaltverhalten

Wird einem Regler der Sequenz seine Freigabe weggenommen oder geht er in Störung, so ist er für die gesamte Sequenz nicht mehr verfügbar.
Handelt es sich dabei nicht um den vorher aktiven Regler, so kann es, je nachdem welche Stellgröße dieser Regler ausgegeben hat, zu einer Temperaturänderung kommen, die durch die Reglersequenz nach Möglichkeit wieder ausgeglichen wird.
Handelt es sich jedoch um den aktiven Regler, dessen Freigabe weggenommen wird, so muss auf den nächsten "sinnvollen" Regler umgeschaltet werden. Der Sequenz-Link-Baustein geht dabei nach folgenden Regeln vor:

- Der gerade deaktivierte Regler hatte direkten Wirksinn (Kühlen)

Es gibt einen betriebsbereiten Regler mit höherer Ordnungszahl → auf den nächsthöheren betriebsbereiten Regler schalten.
Es gibt nur einen betriebsbereiten Regler mit niedrigerer Ordnungszahl → auf den nächst niedrigeren betriebsbereiten Regler schalten.
Es gibt keinen betriebsbereiten Regler mehr → Störmeldung

- Der gerade deaktivierte Regler hatte indirekten Wirksinn (Heizen)
- Es gibt einen betriebsbereiten Regler mit niedrigerer Ordnungszahl → auf den nächst niedrigeren betriebsbereiten Regler schalten.
  Es gibt nur einen betriebsbereiten Regler mit höherer Ordnungszahl → auf den nächsthöheren betriebsbereiten Regler schalten.
  Es gibt keinen betriebsbereiten Regler mehr → Störmeldung

Zuschaltverhalten

Wird ein Regler der Sequenz hinzugefügt, so ist er in jedem Fall zunächst inaktiv und wird, je nach Wirksinn und Positionierung, im Sinne der Sequenzreihenfolge, seinen Minimal- bzw. Maximalwert ausgeben. Die daraus resultierende Temperaturänderung wird durch die Reglersequenz nach Möglichkeit wieder ausgeglichen.

VAR_INPUT

```
bEn                 : BOOL;
udiSttCtrl          : UDINT;
udiIniSwiOvrDly_sec : UDINT;
rX                  : REAL
```

bEn: Aktivierung des Sequenzreglers.

udiSttCtrl: Ordnungszahl des Sequenzreglers, welcher bei der Gesamtaktivierung der Startregler sein soll. Intern begrenzt auf Werte von 0 bis *gBA_cMaxSeqCtrl*.

udiIniSwiOvrDly_sec: Der erste Regler bleibt für mindestens diese Zeit [s] in der Sequenz aktiv, bevor andere Kriterien (siehe FB_BA_SeqLink [▶ 20]) auf einen anderen Regler schalten lassen.

rX: Istwert der Regelung.

VAR_OUTPUT

```
udiCurCtrl              : UDINT;
bSeqActv                : BOOL;
bNotRead                : BOOL;
bNoneOp                 : BOOL;
udiRemTiIniSwiOvrDly_sec : UDINT;
bErr                    : BOOL;
sErrDescr               : T_MAXSTRING;
```

udiCurCtrl: Ordnungszahl des aktuell aktiven Sequenzreglers. Ist keiner aktiv, so wird hier 0 ausgegeben.

bSeqActv: Der Sequenzbaustein ist freigegeben (*bEn*) und hat keinen abschaltenden Fehler, siehe Fehlererkennung.

bNotRead: Jeder Sequenzregler übermittelt dem Steuerbaustein Daten über die Struktur *stSeqLink*. Solange noch keine Daten übermittelt wurden - dies ist beim Einschalten der SPS der Fall - steht dieser Ausgang auf TRUE.

bNoneOp: Dieser Ausgang wird auf TRUE geschaltet, wenn keiner der Sequenzregler seine eigene Freigabe (*bEn*=TRUE) hat.

udiRemTiIniSwiOvrDly_sec: Verbleibende Initialisierungszeit [s] vor dem ersten Umschalten (siehe FB_BA_SeqLink [▶ 20]).

bErr: Dieser Ausgang wird auf TRUE geschaltet, wenn die eingetragenen Parameter fehlerhaft sind. Dieser Baustein stellt im Fehlerfall seine Abarbeitung u.U. nicht ein, siehe Fehlererkennung.

sErrDescr: Enthält die Fehlerbeschreibung.

Fehlerbeschreibung
01: Fehler: Dem Sequenz Link wurde mitgeteilt, dass eine Reglerordnungszahl *udiMyNum* doppelt vergeben worden ist.
02: Warnung: Doppelter Wirksinn-Wechsel in der Reglersequenz.
03: Warnung: In der Reglersequenz hat ein Regler höherer Ordnungszahl einen geringeren Sollwert als sein "Vorgänger". Es erfolgt keine Korrektur - die Reglersequenz läuft mit den eingegebenen Parametern.
04: Warnung: Der Sequenzregler, der als Startregler definiert ist (*udiSttCtrl*) ist gar nicht parametriert, d.h. vorhanden. Es wird der Regler mit der geringsten Ordnungszahl als Startregler verwendet.
05: Warnung: Die Ordnungszahl des Start-Reglers ist höher als die maximal erlaubte Anzahl an Reglern oder Null. Es wird der Regler mit der geringsten Ordnungszahl als Startregler verwendet.
06: Warnung: Der Sequenzregler, der als Startregler definiert ist (*udiSttCtrl*) ist nicht frei gegeben, d.h. vorhanden. Es wird der Regler mit der geringsten Ordnungszahl als Startregler verwendet.

Nur der erste Fehler lässt den Sequenz-Link-Baustein in Störung gehen bzw. sperrt seine Abarbeitung (bSeqActv = FALSE). Keiner der zugehörigen Regler ist dann mehr aktiv und alle Regler geben die Stellgröße "0" aus. Der Baustein ist nicht aktiv:

VAR_IN_OUT

```
stSeqLink : ST_BA_SeqLink;
```

stSeqLink: Daten- und Befehlsstruktur (siehe ST_BA_SeqLink / ST_BA_SeqLinkData [▶ 40]) zwischen den einzelnen Sequenzreglern und dem Steuerbaustein FB_BA_SeqLink . Über diese Struktur empfängt der Sequenz-Link-Baustein alle relevanten Daten der Sequenzregler und teilt diesen gleichzeitig mit, welcher der aktive ist.

Tragen mehrere Sequenzregler dieselbe Nummer (*udiMyNum*), so wird dies erkannt und als Fehler sowohl am Sequenzregler als auch am Steuerbaustein ausgegeben.

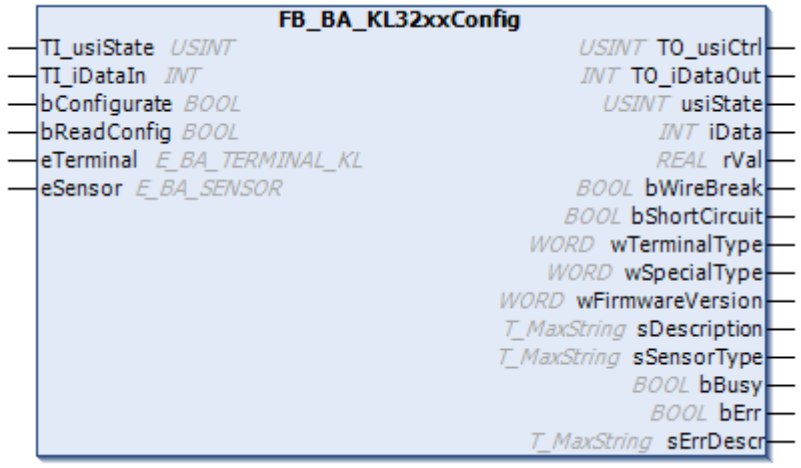
Voraussetzungen

Entwicklungsumgebung	Erforderliche Bibliothek	Erforderliche Function
TwinCAT3.1 4022.16	Tc3Building Automation Common ab V1.0.4.3	TF8040 | TwinCAT Building Automation ab V1.0.5.0

4.1.2 Universal

4.1.2.1 Analoge Ein-/Ausgänge

4.1.2.1.1 FB_BA_KL32xxConfig



Konfiguration der Busklemmen zur Temperaturmessung.

Funktionsbeschreibung

Der Funktionsbaustein dient der Konfiguration von Busklemmen der Typen KL3208_0010, KL3201, KL3202 und KL3204.

VAR_INPUT

```
TI_usiState    : USINT;
TI_iDataIn     : INT;
bConfigurate   : BOOL;
bReadConfig    : BOOL;
eTerminal      : E_BA_TERMINAL_KL;
eSensor        : E_BA_SENSOR;
```

TI_usiState: Verknüpfung mit dem entsprechenden Status Byte der Busklemme im E/A Bereich des Programms.

TI_iDataIn: Verknüpfung mit den entsprechenden Rohdaten (Data In) der Busklemme im E/A Bereich des Programms (0..32767).

bConfigurate: Eine steigende Flanke startet die Konfiguration der Busklemme.

bReadConfig: Eine steigende Flanke startet das Auslesen der Busklemme.

eTerminal: Auswahl der entsprechenden Busklemme (siehe E_BA_Terminal_KL [▶ 41]).

eSensor: Auswahl des Sensortyps (siehe E_BA_Sensor [▶ 43]).

VAR_OUTPUT

```
TO_usiCtrl         : USINT;
TO_iDataOut        : INT;
usiState           : USINT;
iData              : INT;
rVal               : REAL;
bWireBreak         : BOOL;
bShortCircuit      : BOOL;
wTerminalType      : WORD;
wSpecialType       : WORD;
wFirmwareVersion   : WORD;
sDescription       : STRING;
```

```
sSensorType       : STRING;
bErr              : BOOL;
sErrDescr         : T_MAXSTRING;
```

TO_usiCtrl: Verknüpfung mit dem entsprechenden Control Byte der Busklemme im E/A Bereich des Programms.

TO_iDataOut: Verknüpfung mit den entsprechenden Rohdaten (Data Out) der Busklemme im E/A Bereich des Programms.

usiState: Ausgabe des aktuellen Klemmenstatus.

iData: Ausgabe der aktuellen Prozessdaten.

rVal: Skalierter Ausgangswert.

bWireBreak: Anzeige des Kanalstatus, Drahtbruch am Sensor.

bShortCircuit: Anzeige des Kanalstatus, Kurzschluss am Sensor.

wTerminalType: Anzeige des Klemmentyps.

wSpecialType: Anzeige der speziellen Version der Klemme.

wFirmwareVersion: Anzeige der Firmware der Klemme.

sDescription: Anzeige des Klemmentyps und der Firmware.

sSensorType: Anzeige des Sensortyps.

bErr: Fehler in der Klemmenkonfiguration.

sErrDescr: Enthält die Fehlerbeschreibung.

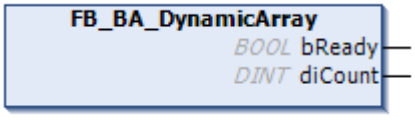
Fehlerbeschreibung
01: Fehler: Klemmenkonfiguration überprüfen KL32xx *eTerminal*/*eSensor*/*TI_usiState*/*TI_iDataIn*/*TO_usiCtrl*/*TO_iDataOut*

Voraussetzungen

Entwicklungsumgebung	Erforderliche Bibliothek	Erforderliche Function
TwinCAT3.1 4022.16	Tc3Building Automation Common ab V1.0.4.3	TF8040 | TwinCAT Building Automation ab V1.0.5.0

4.1.2.2 Array

4.1.2.2.1 FB_BA_DynamicArray



Der Funktionsbaustein erzeugt und löscht Speicherbereiche dynamisch, damit zur Laufzeit Einträge hinzugefügt und entfernt werden können.

Sobald die maximale Kapazität [▶ 28] des Arrays erreicht ist, wird der interne Speicherbereich automatisch erweitert. Ist die Kapazität mehr als ausreichend, wird der interne Speicherbereich verkleinert.

Der intern verwendete Speicher wird aus dem Routerspeicherpool alloziert und zur Laufzeit via _NEW erzeugt und via _DELETE freigegeben.
Bei jeder Anpassung (d.h. Erweiterung oder Verkleinerung) des internen Speichers sind auch die Pointer auf den veralteten / angepassten Speicher ungültig!

Der Datentyp der Einträge spielt für das dynamische Array keine Rolle!
Der Anwender muss in jedem Fall dafür sorgen, dass beim Umgang mit enthaltenen Einträgen stets der Datentyp korrekt von der Applikation beachtet wird.
Des Weiteren müssen alle Daten, die in das Array hinzugefügt werden eine einheitlich definierte Größe [▶ 25] haben!

Es empfiehlt sich das dynamische Array vor allem dann einzusetzen, wenn die zu erwartende Speicherausnutzung relativ gut abgeschätzt werden kann. Routerspeicher steht (vor allem bei kleinen Steuerungen) nur begrenzt zur Verfügung und ist so effizient wie möglich zu verwenden! Gegebenenfalls muss die im Zielsystem verfügbare Menge an Routerspeicher zusätzlich angepasst werden.

VAR_OUTPUT

```
bReady    : BOOL;
discount  : DINT;
```

bReady: Zustand des allozierten Speichers. (TRUE wenn mindestens ein Eintrag im Array enthalten und somit bereits Speicher erzeugt ist)

diCount: Aktuelle Anzahl an enthaltenen Einträgen.

VAR

Interne Variablen, die bei der Deklaration initialisiert werden müssen.

```
uiEntrySize     : UINT;
uiMinExpCount:  : UINT;
```

uiEntrySize: Zu erwartende Größe von Einträgen. Wird verwendet um internen Speicher zu allozieren und Speicherbereiche von aufgenommenen Einträgen zu verwalten.

uiMinExpCount: Zu erweiternde Größe des internen Speichers (angegeben in [Anzahl an Einträgen]) bei Erreichen der maximalen Kapazität [▶ 28].

Weiterführende Informationen, siehe Beispiele [▶ 25] **für Initialisierung bei Variablen-Deklaration.**

Anwendung

Es sind zwei typische Anwendungsfälle vorstellbar:

Fall 1) Array enthält Datensätze

In diesem Fall enthält das Array Datensätze (generische Typen wie z.B. BOOL, INT, STRING oder auch Strukturen), indem interner Speicher entsprechend der Größe des angewendeten Typs reserviert wird.

Fall 2) Array enthält Pointer

In diesem Fall enthält das Array Pointer auf extern deklarierte Daten und es wird nur Speicher entsprechend der Größe von Speicheradressen reserviert.

Instanzen des dynamischen Arrays werden nicht zyklisch aufgerufen. Es ist ausreichend die hier beschriebenen Verwaltungsfunktionen und Eigenschaften anzuwenden.

Beispiele

Beispiel 1:

In einem Array werden Datensäte des Datentyps *ST_DATA* gespeichert.

Ein Zugriff auf die entsprechenden Datensätze geschieht mittels Pointer auf den internen Speicher des Arrays oder mittels einer Kopie eines Datensatzes.

```
VAR
    fbArray : FB_DynamicArray := (uiEntrySize:=SIZEOF(ST_Data), uiMinExpCount:=5);
      stMyDataTmp  : ST_Data;
```

```
       ptrMyDataTmp : POINTER TO ST_Data;
       diIndexTmp   : DINT;
END_VAR
```

```
// 1) Save data in array and remove them with the help of index position:
IF (fbArray.AddEntry(ADR(stMyDataTmp), diResultIndex=>diIndexTmp)) THEN
       fbArray.RemoveEntry(diIndexTmp);
END_IF

// 2) List all data sets consecutively:
FOR diIndexTmp = 0 TO fbArray.LastIndex DO
    IF (fbArray.GetEntryEx(diIndexTmp, pMemoryPtr=>ptrMyDataTemp)) THEN
       ptrMyDataTmp^.diValue := (diIndexTmp+1);
    END_IF
END_FOR

// 3) Get a copy of the first data set:
If (fbArray.GetEntry(0, ADR(stMyDataTmp))) THEN
    // Edit and update data set:
    stMyDataTmp.diValue := 99;

    fbArray.SetEntry(0, ADR(stMyDataTmp));
END_IF
```

Beispiel 2:

In einem Array werden die Adressen von extern deklarierten Instanzen des Funktionsbausteins *FB_Object* gespeichert.

```
VAR
    fbArray : FB_DynamicArray := (uiEntrySize:=SIZEOF(POINTER TO FB_Object), uiMinExpCount:=5);
       fbMyObject1  : FB_Object;
       fbMyObject2  : FB_Object;
       fbObjectTmp  : POINTER TO FB_Object;

       diIndexTmp   : DINT;
END_VAR
```

```
// 1) Add object to array and remove it with the help of index position:
If (fbArray.AddEntryPtr(ADR(fbMyObject1), diResultIndex=>diIndexTmp)) THEN
    fbArray.RemoveEntry(diIndexTmp);
END_IF

// 2) Add object to array and remove subsequently with the use of the pointer:
fbArray.AddEntryPtr(ADR(fbMyObject1));
fbArray.RemoveEntryExPtr(ADR(fbMyObject1));

// 3) Determine the index position of an object within an array:
IF (fbArray.FindEntryPtr(ADR(fbMyObject1), diResultIndex=>diIndexTemp)) THEN
    // Replace entry on position "fbMyObject1" with "fbMyObject2":
    fbArray.SetEntryPtr(diIndexTmp, ADR(fbMyObject2));
ELSE
    // Error handling
    …
END_IF

// 4) Determine first object:
IF (fbArray.GetEntry(0,ADR(fbObjTemp))) THEN
    // …
END_IF

// 5) Remove content of the array if it has more than 10 entries:
IF(fbArray.diCount > 10) THEN
    fb_Array.Reset();
END_IF
```

Fehlermeldungen

Folgende Fehlermeldungen können zur Laufzeit im TwinCAT Ausgabefenster ausgegeben werden:

[EDB4] Entry-size of array not defined!
Die zu erwartende Größe [▶ 25] von Einträgen wurde bei der Deklaration des Arrays nicht initialisiert.

[EDB7] Expansion-count of entries not defined!
Die zu erweiternde Größe [▶ 25] des internen Speichers wurde bei der Deklaration des Arrays nicht initialisiert.

Methods of FB_BA_DynamicArray

Name	Definitionsort	Beschreibung
AddEntry [▶ 28]	Lokal	Erstellt einen neuen Datensatz am Ende des Arrays und kopiert den Inhalt des angegebenen Eintrags in den internen Speicher
FindEntry [▶ 28]	Lokal	Ermittelt die Position des angegebenen Eintrags im Array, indem dessen Inhalt mit den Datensätzen des Arrays vergleichen wird.
GetEntry [▶ 29]	Lokal	Kopiert den Inhalt des Datensatzes an einer bestimmten Position auf den angegebenen Speicherbereich.
GetEntryEx [▶ 29]	Lokal	Ermittelt einen Pointer auf den internen Speicher des angeforderten Datensatzes.
RemoveEntry [▶ 30]	Lokal	Entfernt den Datensatz an der angegebenen Index-Position aus dem Array.
RemoveEntryEx [▶ 30]	Lokal	Ermittelt die Position des angegebenen Eintrags und löscht diesen aus dem Array.
Reset [▶ 30]	Lokal	Setzt den kompletten Inhalt des Arrays zurück.
SetEntry [▶ 30]	Lokal	Ersetzt den existierenden Datensatz durch einen neuen, indem der interne Speicherbereich des existierenden Datensatzes mit dem Wert des neuen Eintrags überschrieben wird.
AddEntryPtr [▶ 31]	Lokal	Erstellt einen neuen Eintrag am Ende des Arrays und kopiert dessen Speicheradresse (d.h. die Adresse auf die der Pointer *pEntry* zeigt) in den internen Speicher.
FindEntryPtr [▶ 31]	Lokal	Ermittelt die Position eines Eintrags im Array, indem dessen Adresse mit den im Array gespeicherten Adressen verglichen wird.
GetEntryExPtr [▶ 31]	Lokal	Gibt einen Pointer auf die Speicheradresse des angeforderten Eintrags aus.
RemoveEntryExPtr [▶ 32]	Lokal	Ermittelt die Position des angegebenen Eintrags und löscht diesen aus dem Array.
SetEntryPtr [▶ 32]	Lokal	Ersetzt einen existierenden Eintrag durch einen Neuen.

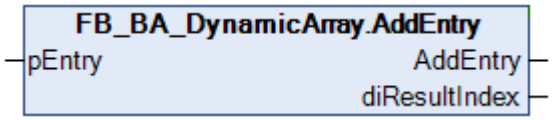
Properties of FB_BA_DynamicArray

Name	Typ	Zugriff	Definitionsort	Initialwert	Beschreibung
CurCapacity	DINT	Get	Lokal	-	Aktuelle Kapazität des Arrays (Anzahl an Einträgen). Entspricht der maximalen Anzahl an Einträgen die vom internen Speicher aufgenommen werden kann.
EntrySize	DINT	Get	Lokal	uiEntrySize [▶ 25]	Zu erwartende Größe von Einträgen welche im Array gespeichert werden
LastIndex	DINT	Get	Lokal	-	Index-Position des letzten Eintrags. Ist -1 wenn keine Einträge vorhanden sind
UsedMemory	DINT	Get	Lokal	-	Größe des verbrauchten internen Speichers [Byte].

Voraussetzungen

Entwicklungsumgebung	Erforderliche Bibliothek	Erforderliche Function
TwinCAT3.1 4022.16	Tc3Building Automation Common ab V1.0.4.3	TF8040 | TwinCAT Building Automation ab V1.0.5.0

AddEntry



Erstellt einen neuen Datensatz am Ende des Arrays und kopiert den Inhalt des angegebenen Eintrags in den internen Speicher.

VAR_INPUT

```
pEntry    : PVOID;
```

pEntry: Pointer auf den hinzuzufügenden Eintrag.

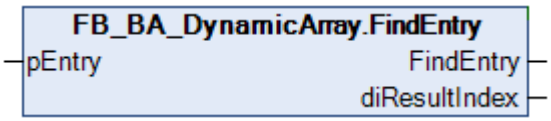
VAR_OUTPUT

```
AddEntry       : BOOL;
diResultIndex  : DINT;
```

AddEntry: Ergebnis der Funktion.

diResultIndex: Index-Position des hinzugefügten Eintrags.

FindEntry



Ermittelt die Position des angegebenen Eintrags im Array, indem dessen Inhalt mit den Datensätzen des Arrays verglichen wird.

VAR_INPUT

```
pEntry    : PVOID;
```

pEntry: Pointer auf den gesuchten Eintrag.

VAR_OUTPUT

```
FindEntry      : BOOL;
diResultIndex  : DINT;
```

FindEntry: Ergebnis der Funktion.

diResultIndex: Index-Position des hinzugefügten Eintrags.

GetEntry

Kopiert den Inhalt des Datensatzes an einer bestimmten Position auf den angegebenen Speicherbereich.

VAR_INPUT

```
diIndex       : DINT;
pResultEntry  : PVOID;
```

diIndex: Index-Position des auszugebenen Datensatzes.

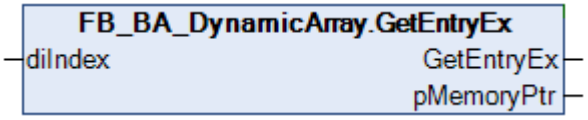
pResultEntry: Pointer auf den Speicherbereich welcher zur Ausgabe des Datensatzes angewendet werden soll.

VAR_OUTPUT

```
GetEntry       : BOOL;
```

GetEntry: Ergebnis der Funktion.

GetEntryEx



Ermittelt einen Pointer auf den internen Speicher des angeforderten Datensatzes.

VAR_INPUT

```
diIndex       : DINT;
```

diIndex: Index-Position des auszugebenen Datensatzes.

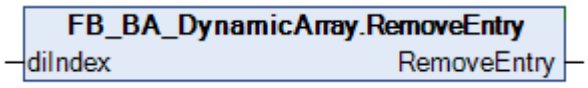
VAR_OUTPUT

```
GetEntryEx     : BOOL;
pMemoryPtr     : POINTER TO PVOID;
```

GetEntry: Ergebnis der Funktion.

pMemoryPtr: Pointer welcher zur Ausgabe des Datensatzes angewendet werden soll.

RemoveEntry



Entfernt den Datensatz an der angegebenen Index-Position aus dem Array.

VAR_INPUT

```
diIndex    : DINT;
```

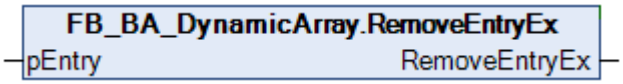
diIndex: Index-Position des zu entfernenden Datensatzes.

VAR_OUTPUT

```
RemoveEntry    : BOOL;
```

RemoveEntry: Ergebnis der Funktion.

RemoveEntryEx



Ermittelt die Position des angegebenen Eintrags und löscht diesen aus dem Array.

VAR_INPUT

```
pEntry    : PVOID;
```

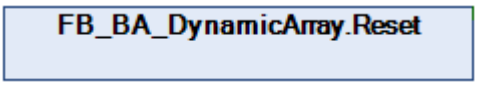
pEntry: Pointer auf zu entfernenden Eintrag.

VAR_OUTPUT

```
RemoveEntryEx    : BOOL;
```

RemoveEntryEx: Ergebnis der Funktion.

Reset



Setzt den kompletten Inhalt des Arrays zurück.

SetEntry

Ersetzt den existierenden Datensatz durch einen neuen, indem der interne Speicherbereich des existierenden Datensatzes mit dem Wert des neuen Eintrags überschrieben wird.

VAR_INPUT

```
diIndex    : DINT;
pEntry    : PVOID;
```

diIndex: Index-Position des zu ersetzenden Datensatzes.

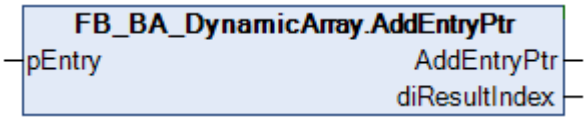
pEntry: Pointer auf zu entfernenden Eintrag.

VAR_OUTPUT

```
SetEntry    : BOOL;
```

SetEntry: Ergebnis der Funktion.

AddEntryPtr



Erstellt einen neuen Eintrag am Ende des Arrays und kopiert dessen Speicheradresse (d.h. die Adresse auf die der Pointer *pEntry* zeigt) in den internen Speicher.

VAR_INPUT

```
pEntry    : PVOID;
```

pEntry: Pointer auf hinzuzufügenden Eintrag.

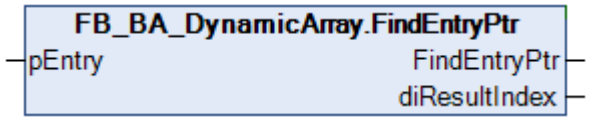
VAR_OUTPUT

```
AddEntryPtr    : BOOL;
diResultIndex  : DINT;
```

AddEntryPtr: Ergebnis der Funktion.

diResultIndex: Index-Position des hinzugefügten Eintrags.

FindEntryPtr



Ermittelt die Position eines Eintrags im Array, indem dessen Adresse mit den im Array gespeicherten Adressen verglichen wird.

VAR_INPUT

```
pEntry    : PVOID;
```

pEntry: Pointer auf den gesuchten Eintrag.

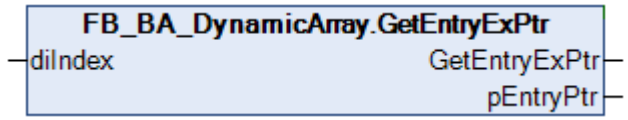
VAR_OUTPUT

```
FindEntryPtr    : BOOL;
diResultIndex  : DINT;
```

FindEntryPtr: Ergebnis der Funktion.

diResultIndex: Index-Position des gesuchten Eintrags.

GetEntryExPtr



Gibt einen Pointer auf die Speicheradresse des angeforderten Eintrags aus.

VAR_INPUT

```
diIndex    : DINT;
```

diIndex: Index-Position des auszugebenden Eintrags.

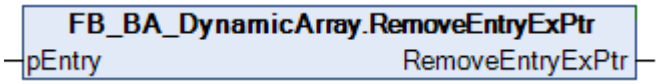
VAR_OUTPUT

```
GetEntryExPtr    : BOOL;
pEntryPtr        : POINTER TO PVOID;
```

GetEntryExPtr: Ergebnis der Funktion.

pEntryPtr: Pointer welcher zur Ausgabe des Eintrags angewendet werden soll.

RemoveEntryExPtr



Ermittelt die Position des angegebenen Eintrags und löscht diesen aus dem Array.

VAR_INPUT

```
pEntry    : PVOID;
```

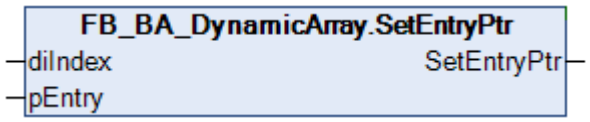
pEntry: Pointer auf zu entfernenden Eintrag.

VAR_OUTPUT

```
RemoveEntryExPtr    : BOOL;
```

RemoveEntryExPtr: Ergebnis der Funktion.

SetEntryPtr



Ersetzt einen existierenden Eintrag durch einen neuen, indem die Speicheradresse des existierenden Eintrags mit der Speicheradresse des neuen Eintrags überschrieben wird.

VAR_INPUT

```
diIndex    : DINT;
pEntry     : PVOID;
```

diIndex: Index-Position des zu ersetzenden Eintrags.

pEntry: Pointer auf zu ersetzenden Eintrag.

VAR_OUTPUT

```
SetEntryPtr    : BOOL;
```

SetEntryPtr: Ergebnis der Funktion.

4.1.2.2.2 FB_BA_StaticArray

Der Funktionsbaustein ist eine Erweiterung des Funktionsbausteins FB_DynamicArray [▶ 24].

Hintergrund dieser Erweiterung ist, den Einsatz von Routerspeicher zu vermeiden, und stattdessen statischen Speicher nutzen. Dieser ist in der Applikation zu deklarieren und kann dort beliebig in dessen Größe angepasst werden.

> Der statische Speicher ist zwar von der Applikation bereitzustellen, jedoch darf dieser niemals außerhalb des Arrays verändert werden! Die Verwaltung sollte in jedem Fall über das Array selbst geschehen.

Es empfiehlt sich das statische Array immer dann einzusetzen, wenn die zu erwartende Speicherausnutzung genau abgeschätzt werden kann.

Auf Gründen der Effizienz sollte die Speichergröße so dimensioniert werden, dass so wenig Speicher wie möglich, und nur so viel Speicher wie nötig reserviert wird. Zur Deklaration der Grenze des Speicherbereichs eignen sich globale Konstanten und Parameterlisten.

Anwendung

Grundsätzlich sind die Anwendungsfälle identisch mit dem dynamischen Array [▶ 24]. Lediglich die Deklaration unterscheidet sich zum Teil, da dort der externe Speicherbereich und dessen Größe zu übergeben sind.

Weiterführende Informationen

Siehe Beispiele [▶ 33] für Initialisierung bei Variablen-Deklaration.

Beispiele

Exemplarische Deklaration des Arrays, dessen statischem Speicher und entsprechender Konstanten.

```
VAR_GLOBAL CONSTANT
    uiObjectCount  : UINT := 100;
    uiArrayMemSize : UINT := TO_UINT(uiObjectCount * SIZEOF(FB_OBJECT));
END_VAR
```

```
VAR
    bArrayMemory   : ARRAY[0.. uiArrayMemSize] OF BYTE;
    fbArray        : FB_StaticArray := (uiEntrySize:=SIZEOF(FB_Object), pExtMemory:=ADR(bArrayMemory
), uiExtMemorySize:=uiArrayMemSize);)
END_VAR
```

Weiterführende Informationen

Da das statische Array genauso anzuwenden ist wie das dynamische Array [▶ 24], sind dort entsprechende Anwendungsbeispiele [▶ 25] dokumentiert.

Methods of FB_BA_StaticArray

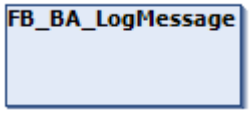
Name	Definitionsort	Beschreibung
AddEntry [▶ 28]	Lokal	Erstellt einen neuen Datensatz am Ende des Arrays und kopiert den Inhalt des angegebenen Eintrags in den internen Speicher
Reset [▶ 30]	Lokal	Setzt den kompletten Inhalt des Arrays zurück.

Voraussetzungen

Entwicklungsumgebung	Erforderliche Bibliothek	Erforderliche Function
TwinCAT3.1 4022.16	Tc3Building Automation Common ab V1.0.4.3	TF8040 | TwinCAT Building Automation ab V1.0.5.0

4.1.2.3 Log

4.1.2.3.1 FB_BA_LogMessage



Der Funktionsbaustein dient zum Ausgeben von Meldungen im Ausgabe-Fenster von TwinCAT.

Für jede auszugebende Meldung wird der Symbolpfad des Funktionsbausteins eingefügt, sodass der Benutzer anhand der Meldung die aufrufende Funktionsbaustein-Instanz erkennen kann.

Instanzen dieses Funktionsbausteins können **nicht** explizit aufgerufen werden.

Es sind separate Funktionen für verschiedene Anwendungsfälle verfügbar, welche im nachfolgenden Teil dieser Dokumentation beschrieben werden.

Die Funktionalität zur Ausgabe von Meldungen wird von der Funktion ADSLOGDINT bereitgestellt, welche intern angewendet wird.

VAR_OUTPUT

```
sResult     : T_MaxSTRING;
```

sResult: Inhalt der zuletzt ausgegebenen Meldung.

Anwendung

Kontextbezogene Zusatzinformationen

Der Entwickler hat die Möglichkeit ein Kürzel in jeder Meldung auszugeben.

Über dieses Kürzel können Meldungen einfacher im Quellcode lokalisiert werden (wenn z.B. mittels Suchfunktion nach dem ausgegebenen Kürzel gesucht wird).

Unterdrücken zyklisch wiederholter Meldungen

Um das zyklische Ausgeben derselben Meldung zu unterdrücken wird der aktuelle Log-Code mit dem zuletzt angewendeten Log-Code verglichen. Wenn beide Werte übereinstimmen wird die Ausgabe der Meldung unterdrückt, was im Umkehrschluss bedeutet, dass unterschiedliche, aufeinanderfolgende Meldungen angezeigt würden.

Dieses Verhalten kann mit der Option *bIgnoreBlock*, aus der Funktion Show, beeinflusst werden:

TRUE verhindert das Unterdrücken einer zyklisch wiederholten Meldung.

Beispiel1:

Die Beispielfunktion *DoWork()* gibt in Zeile 150 eine Warnung aus:

```
FUNCTION_BLOCK FB_TEST
VAR
  fbLogMsg   : FB_LogMessage;
END_VAR
```

```
FUNCTION DoWork
```

```
fbLogMsg.Show(ADSLOG_MSGTYPE_WARN, 'DW150', 'Function not ready.', FALSE;
```

Beispiel2:

Die Beispielfunktion *Init()* gibt in Zeile 80 eine Fehlermeldung aus, welche zyklisch wiederholt [▶ 34] werden könnte:

```
FUNCTION_BLOCK FB_TEST
VAR
  fbLogMsg    : FB_LogMessage;
  iState      : INT := 0;
  sDevice     : STRING := 'CX9020';
END_VAR
```

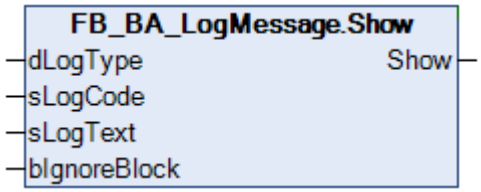
```
FUNCTION Init
```

```
fbLogMsg.Show1(ADSLOG_MSGTYPE_ERROR, 'I80', 'device %s has an
 Invalid state "%d".', F_STRINGEx(sDevice), F_INT(iState), TRUE;
```

Voraussetzungen

Entwicklungsumgebung	Erforderliche Bibliothek	Erforderliche Function
TwinCAT3.1 4022.16	Tc3Building Automation Common ab V1.0.4.3	TF8040 | TwinCAT Building Automation ab V1.0.5.0

Show



Ausgabe einer einfachen Meldung.

VAR_INPUT

```
dLogType      : DWORD;
sLogCode      : T_MaxString;
sLogText      : T_MaxString;
bIgnoreBlock  : BOOL;
```

dLogType: Anzuzeigender Log-Typ der Meldung.

sLogCode: Optionale, kontextbezogene Zusatzinformation [▶ 34].

sLogText: Inhalt der Meldung.

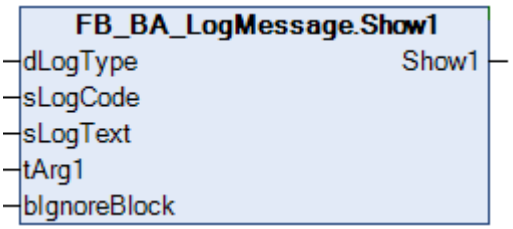
bIgnoreBlock: Verhindert das Unterdrücken von zyklisch wiederholten Meldungen [▶ 34].

VAR_OUTPUT

```
Show     : BOOL;
```

Show: Indikator ob eine Meldung ausgegeben (*TRUE*) oder verworfen (*FALSE*) wurde.

Show1



Ausgabe einer Meldung mit einem zu formatierenden Wert.

VAR_INPUT

```
dLogType      : DWORD;
sLogCode      : T_MaxString;
sLogText      : T_MaxString;
tArg1         : T_Arg;
bIgnoreBlock  : BOOL;
```

dLogType: Anzuzeigeder Log-Typ der Meldung.

sLogCode: Optionale, kontextbezogene Zusatzinformation [▶ 34].

sLogText: Inhalt der Meldung.

tArg1: Zu formatierender Wert (siehe T_Arg).

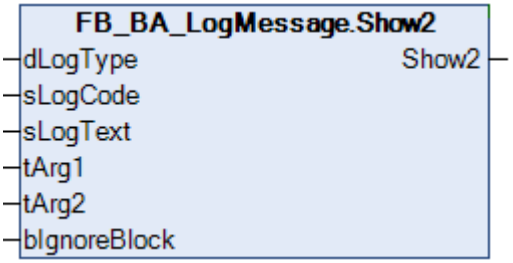
bIgnoreBlock: Verhindert das Unterdrücken von zyklisch wiederholten Meldungen [▶ 34].

VAR_OUTPUT

```
Show1     : BOOL;
```

Show1: Indikator ob eine Meldung ausgegeben (*TRUE*) oder verworfen (*FALSE*) wurde.

Show2



Ausgabe einer Meldung mit zwei zu formatierenden Werten.

VAR_INPUT

```
dLogType      : DWORD;
sLogCode      : T_MaxString;
sLogText      : T_MaxString;
tArg1         : T_Arg;
tArg2         : T_Arg;
bIgnoreBlock  : BOOL;
```

dLogType: Anzuzeigeder Log-Typ der Meldung.

sLogCode: Optionale, kontextbezogene Zusatzinformation [▶ 34].

sLogText: Inhalt der Meldung.

tArg1: Erster zu formatierender Wert (siehe T_Arg).

tArg2: Zweiter zu formatierender Wert (siehe T_Arg).

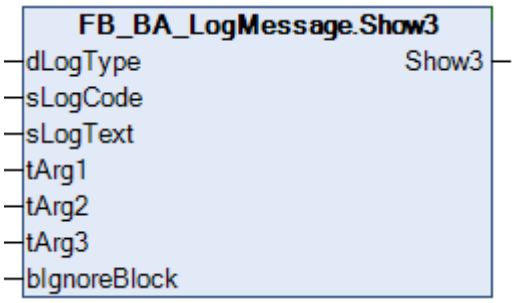
bIgnoreBlock: Verhindert das Unterdrücken von zyklisch wiederholten Meldungen [▶ 34].

VAR_OUTPUT

```
Show2     : BOOL;
```

Show2: Indikator ob eine Meldung ausgegeben (*TRUE*) oder verworfen (*FALSE*) wurde.

Show3



Ausgabe einer Meldung mit drei zu formatierenden Werten.

VAR_INPUT

```
dLogType      : DWORD;
sLogCode      : T_MaxString;
sLogText      : T_MaxString;
tArg1         : T_Arg;
tArg2         : T_Arg;
tArg3         : T_Arg;
bIgnoreBlock  : BOOL;
```

dLogType: Anzuzeigender Log-Typ der Meldung.

sLogCode: Optionale, kontextbezogene Zusatzinformation [▶ 34].

sLogText: Inhalt der Meldung.

tArg1: Erster zu formatierender Wert (siehe T_Arg).

tArg2: Zweiter zu formatierender Wert (siehe T_Arg).

tArg3: Zweiter zu formatierender Wert (siehe T_Arg).

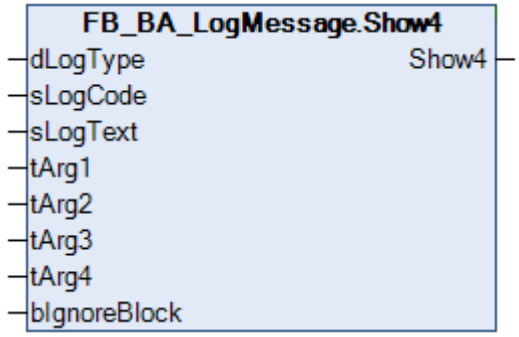
bIgnoreBlock: Verhindert das Unterdrücken von zyklisch wiederholten Meldungen [▶ 34].

VAR_OUTPUT

```
Show3     : BOOL;
```

Show3: Indikator ob eine Meldung ausgegeben (*TRUE*) oder verworfen (*FALSE*) wurde.

Show4



Ausgabe einer Meldung mit vier zu formatierenden Werten.

VAR_INPUT

```
dLogType      : DWORD;
sLogCode      : T_MaxString;
sLogText      : T_MaxString;
tArg1         : T_Arg;
tArg2         : T_Arg;
```

```
tArg3         : T_Arg;
tArg4         : T_Arg;
bIgnoreBlock  : BOOL;
```

dLogType: Anzuzeigender Log-Typ der Meldung.

sLogCode: Optionale, kontextbezogene Zusatzinformation [▶ 34].

sLogText: Inhalt der Meldung.

tArg1: Erster zu formatierender Wert (siehe T_Arg).

tArg2: Zweiter zu formatierender Wert (siehe T_Arg).

tArg3: Zweiter zu formatierender Wert (siehe T_Arg).

tArg4: Zweiter zu formatierender Wert (siehe T_Arg).

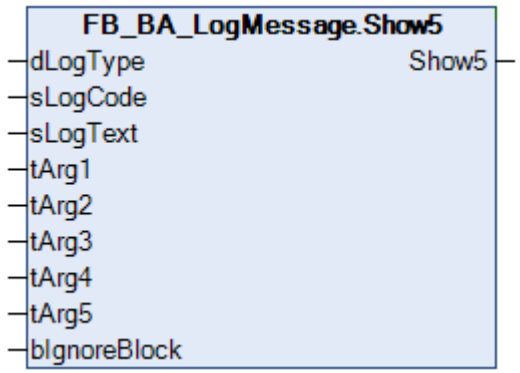
bIgnoreBlock: Verhindert das Unterdrücken von zyklisch wiederholten Meldungen [▶ 34].

VAR_OUTPUT

```
Show4     : BOOL;
```

Show4: Indikator ob eine Meldung ausgegeben (*TRUE*) oder verworfen (*FALSE*) wurde.

Show5



Ausgabe einer Meldung mit fünf zu formatierenden Werten.

VAR_INPUT

```
dLogType      : DWORD;
sLogCode      : T_MaxString;
sLogText      : T_MaxString;
tArg1         : T_Arg;
tArg2         : T_Arg;
tArg3         : T_Arg;
tArg4         : T_Arg;
tArg5         : T_Arg;
bIgnoreBlock  : BOOL;
```

dLogType: Anzuzeigender Log-Typ der Meldung.

sLogCode: Optionale, kontextbezogene Zusatzinformation [▶ 34].

sLogText: Inhalt der Meldung.

tArg1: Erster zu formatierender Wert (siehe T_Arg).

tArg2: Zweiter zu formatierender Wert (siehe T_Arg).

tArg3: Zweiter zu formatierender Wert (siehe T_Arg).

tArg4: Zweiter zu formatierender Wert (siehe T_Arg).

tArg5: Zweiter zu formatierender Wert (siehe T_Arg).

bIgnoreBlock: Verhindert das Unterdrücken von zyklisch wiederholten Meldungen [▶ 34].

VAR_OUTPUT

```
Show5      : BOOL;
```

Show5: Indikator ob eine Meldung ausgegeben (*TRUE*) oder verworfen (*FALSE*) wurde.

4.1.2.4 Trigger

4.1.2.4.1 FB_BA_ATrigCOV

Der Funktionsbaustein überwacht den Wert *xValue* auf Änderung (Change of Value).

> Der überwachte Wert ist Datentyp-unabhängig (ANY).
> Aus Performance-Gründen werden jedoch nur Datentypen unterstützt die kleiner oder gleich 4 Byte sind!

VAR_INPUT

```
xValue      : ANY;
bForce      : BOOL;
```

xValue: Zu überwachender Wert.

bForce: Erzwingt einen positiven Vergleich („bQ=TRUE“).

VAR_OUTPUT

```
bReady      : BOOL;
bQ          : BOOL;
```

bReady: Zeigt Funktionsbereitschaft an:

Bei Gültigkeit von *xValue.*
Korrekte Wertzuweisung und Beachtung der zulässigen Datentyp-Größe.

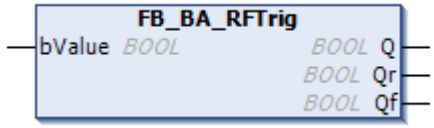
Speicher ist initialisiert.
Der Vergleich kann frühestens nach einem Zyklus ausgeführt werden, da der interne Speicher erst mit dem Wert *xValue* initialisiert werden muss.

bQ: Ergebnis des letzten Vergleichs (*TRUE* bei Wertänderung).

Voraussetzungen

Entwicklungsumgebung	Erforderliche Bibliothek	Erforderliche Function
TwinCAT3.1 4022.16	Tc3Building Automation Common ab V1.0.4.3	TF8040 | TwinCAT Building Automation ab V1.0.5.0

4.1.2.4.2 FB_BA_RFTrig



Funktionsbaustein zum Erkennen einer steigenden oder fallenden Flanke an einer booleschen Variable. Mit dem Baustein kann der Einsatz der separaten Funktionsbausteine R_TRIG und F_TRIG vermieden werden.

VAR_INPUT

```
bValue    : BOOL;
```

bValue: Zu überwachender Wert.

VAR_OUTPUT

```
Q         : BOOL;
Qr        : BOOL;
Qf        : BOOL;
```

Q: *TRUE* wenn eine Flanke erkannt wird.

Qr: Ergebnis des letzten Vergleichs (*TRUE* sobald überwachter Wert von *FALSE* nach *TRUE* wechselt).

Qf: Ergebnis des letzten Vergleichs (*TRUE* sobald überwachter Wert von *TRUE* nach *FALSE* wechselt).

Voraussetzungen

Entwicklungsumgebung	Erforderliche Bibliothek	Erforderliche Function
TwinCAT3.1 4022.16	Tc3Building Automation Common ab V1.0.4.3	TF8040 | TwinCAT Building Automation ab V1.0.5.0

4.2 DUTs

4.2.1 Structures

4.2.1.1 ST_BA_SeqLink / ST_BA_SeqLinkData

Struktur des Daten- und Befehlsaustausches zwischen dem Steuerbaustein FB_BA_SeqLink [▶ 20] und den Sequenzreglern FB_BA_SeqCtrl [▶ 16].

Diese Struktur ist pro Sequenzregelung einmal anzulegen:

```
stSeqLink : ST_BA_SeqLink;
```

Innerhalb dieser Struktur ist automatisch eine weitere Feldstruktur deklariert, über die der Sequenz-Link-Baustein auf der einen Seite und die einzelnen Sequenzregler auf der anderen alle relevanten Daten miteinander austauschen. Jeder Sequenzregler schreibt dabei seine Daten in das seiner Ordnungszahl (Eintrag am Eingang *diMyNum* am Sequenzreglerbaustein) entsprechende Feldelement. An den Bausteinen angelegt wird immer die komplette Struktur mit allen Feldelementen.

Die Strukturen haben den folgenden Aufbau:

```
TYPE ST_BA_SeqLink :
STRUCT
  arrSeqLinkData : ARRAY[1..16] OF ST_BA_SeqLinkData;
  diCurCtrl      : DINT;
  bSeqActv       : BOOL;
END_STRUCT
END_TYPE
```

arrSeqLinkData: Parameter der einzelnen Sequenzregler. Beschreibung der Struktur *ST_BA_SeqLinkData* siehe unten.

diCurCtrl: vom FB_BA_SeqLink: Vorgabe aktueller Sequenzregler.

bSeqActv: Die Sequenzregelung ist freigegeben und aktiv.

```
TYPE ST_BA_SeqLinkData:
STRUCT
  lrY         : LREAL;
  lrYMin      : LREAL;
  lrYMax      : LREAL;
  lrW         : LREAL;
  bActn       : BOOL;
  bOp         : BOOL;
  bPresence   : BOOL;
  bErrDouble  : BOOL;
  diCurCtrl   : DINT;
END_STRUCT
END_TYPE
```

lrY: vom FB_BA_SeqCtrl: Übermittlung aktueller Stellwert.

lrYMin: vom FB_BA_SeqCtrl: Übermittlung minimaler Stellwert.

lrYMax: vom FB_BA_SeqCtrl: Übermittlung maximaler Stellwert.

lrW: vom FB_BA_SeqCtrl: Übermittlung aktueller Sollwert.

bActn: vom FB_BA_SeqCtrl: Übermittlung Wirksinn invers (*bActn* = FALSE: Heizbetrieb - *bActn* = TRUE: Kühlbetrieb).

bOp: vom FB_BA_SeqCtrl:Sequenzregler ist freigegeben, d.h. sein Eingang *bEn* ist auf TRUE gesetzt.

bPresence: vom FB_BA_SeqCtrl: Prüfbit, s.u.

bErrDouble: vom FB_BA_SeqCtrl: Fehler beim Prüfen der Nummern: Es existieren mindestens 2 Sequenzregler gleicher Ordnungszahl *diMyNum.*

diCurCtrl: vom FB_BA_SeqLink: Vorgabe aktueller Sequenzregler.

Bemerkung zum Prüfbit:

Jeder Sequenzcontroller setzt in der für ihn gültigen Struktur das *bPresence*-Flag. Ist dieses jedoch schon gesetzt, so muss *diMyNum* zwangsläufig doppelt vergeben worden sein und es greifen zwei Sequenzregler auf dieselbe Struktur zu. Der Sequenz-Link-Baustein setzt alle Prüfbits nach Auswertung wieder zurück, so dass dieser Test zyklisch erfolgt. Dadurch kann ein Fehler automatisch per online-change behoben, bzw. auch neue Sequenzregler hinzugefügt werden.

Voraussetzungen

Entwicklungsumgebung	Erforderliche Bibliothek	Erforderliche Function
TwinCAT3.1 4022.16	Tc3Building Automation Common ab V1.0.4.3	TF8040 | TwinCAT Building Automation ab V1.0.5.0

4.2.2 Enums

4.2.2.1 E_BA_Terminal_KL

Enumerator zur Auswahl der entsprechenden Busklemme.

```
TYPE E_BA_TERMINAL_KL:
(
  KL3208_0010     := 0,
  KL320x_0000     := 1,
  KL300x          := 2,
  KL301x          := 3,
  KL302x          := 4,
  KL304x          := 5,
  KL305x          := 6,
  KL306x          := 7,
  KL3132_0000     := 8,
  KL3142_0000     := 9,
  KL3152_0000     := 10,
  KL3162_0000     := 11,
```

```
  KL3172_0000     := 12,
  KL3172_0500     := 13,
  KL3172_1000     := 14,
  KL3182_0000     := 15,
  KL3404          := 16,
  KL3464          := 17,
  KL3408          := 18,
  KL3468          := 19,
  KL3444          := 20,
  KL3454          := 21,
  KL3448          := 22,
  KL3458          := 23,
  Undefined       := 16#FFFF
)DINT;
END_TYPE
```

KL3208_0010: Temperatursensoren mit Leitungsbruch und Kurzschluss Erkennung.

KL320x_0000: Temperatursensoren mit Leitungsbruch und Kurzschluss Erkennung.

KL300x: -10V..10V.

KL301x: 0mA..20mA mit Leitungsbruch und Kurzschluss Erkennung.

KL302x: 4mA..20mA mit Leitungsbruch und Kurzschluss Erkennung.

KL304x: 0mA..20mA mit Leitungsbruch und Kurzschluss Erkennung.

KL305x: 4mA..20mA mit Leitungsbruch und Kurzschluss Erkennung.

KL306x: 0V..10V.

KL3132_0000: -10V..+10V.

KL3142_0000: 0mA..20mA mit Leitungsbruch und Kurzschluss Erkennung.

KL3152_0000: 4mA..20mA mit Leitungsbruch und Kurzschluss Erkennung.

KL3162_0000: 0V..+10V.

KL3172_0000: 0V..+2V.

KL3172_0500: 0V..+0,5V.

KL3172_1000: 0V..+1.0V.

KL3182_0000: -2,0V..+2,0V.

KL3404: -10V..+10V.

KL3464: 0V..+10V.

KL3408: -10V..+10V.

KL3468: 0V..+10V.

KL3444: 0mA..20mA mit Leitungsbruch und Kurzschluss Erkennung.

KL3454: 4mA..20mA mit Leitungsbruch und Kurzschluss Erkennung.

KL3448: 0mA..20mA mit Leitungsbruch und Kurzschluss Erkennung.

KL3458: 4mA..20mA mit Leitungsbruch und Kurzschluss Erkennung.

Voraussetzungen

Entwicklungsumgebung	Erforderliche Bibliothek	Erforderliche Function
TwinCAT3.1 4022.16	Tc3Building Automation Common ab V1.0.4.3	TF8040 | TwinCAT Building Automation ab V1.0.5.0

4.2.2.2 E_BA_Sensor

Enumerator zur Auswahl eines Sensortyps zur Erfassung von Analogwerten.

```
TYPE E_BA_SENSOR :
(
  KL3208_0010_PT1000              := 0,
  KL3208_0010_NI1000              := 1,
  KL3208_0010_NI1000_LS           := 2,
  KL3208_0010_NTC1K8              := 3,
  KL3208_0010_NTC1K8_TK           := 4,
  KL3208_0010_NTC2K2              := 5,
  KL3208_0010_NTC3K               := 6,
  KL3208_0010_NTC5K               := 7,
  KL3208_0010_NTC10K              := 8,
  KL3208_0010_NTC10KPRE           := 9,
  KL3208_0010_NTC10K_3204         := 10,
  KL3208_0010_NTC10KTYP2          := 11,
  KL3208_0010_NTC10KTYP3          := 12,
  KL3208_0010_NTC10KDALE          := 13,
  KL3208_0010_NTC10K3A221         := 14,
  KL3208_0010_NTC20K              := 15,
  KL3208_0010_NTC100K             := 16,
  KL3208_0010_Poti_Resolution_01  := 17,
  KL3208_0010_Poti_Resolution_1_1 := 18,
  KL320x_0000_PT1000              := 19,
  KL320x_0000_NI1000              := 20,
  KL320x_0000_PT100               := 21,
  KL320x_0000_PT200               := 22,
  KL320x_0000_PT500               := 23,
  KL320x_0000_NI100               := 24,
  KL320x_0000_NI120               := 25,
  KL320x_0000_Output_10_5000      := 26,
  KL320x_0000_Output_10_1200      := 27,
  Undefined                       := 16#FFFF
)DINT;
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Erforderliche Bibliothek	Erforderliche Function
TwinCAT3.1 4022.16	Tc3Building Automation Common ab V1.0.4.3	TF8040 | TwinCAT Building Automation ab V1.0.5.0

4.3 GVLs

4.3.1 Parameter

Globale Parameter

```
VAR_GLOBAL CONSTANT
  usiMaxSeqCtrl       : USINT := 16;
END_VAR
```

usiMaxSeqCtrl: Maximale Anzahl an Sequenzreglern in einer Sequenz.

5 Anhang

5.1 Support und Service

Beckhoff und seine weltweiten Partnerfirmen bieten einen umfassenden Support und Service, der eine schnelle und kompetente Unterstützung bei allen Fragen zu Beckhoff Produkten und Systemlösungen zur Verfügung stellt.

Beckhoff Niederlassungen und Vertretungen

Wenden Sie sich bitte an Ihre Beckhoff Niederlassung oder Ihre Vertretung für den lokalen Support und Service zu Beckhoff Produkten!

Die Adressen der weltweiten Beckhoff Niederlassungen und Vertretungen entnehmen Sie bitte unseren Internetseiten: https://www.beckhoff.de

Dort finden Sie auch weitere Dokumentationen zu Beckhoff Komponenten.

Beckhoff Support

Der Support bietet Ihnen einen umfangreichen technischen Support, der Sie nicht nur bei dem Einsatz einzelner Beckhoff Produkte, sondern auch bei weiteren umfassenden Dienstleistungen unterstützt:

- Support
- Planung, Programmierung und Inbetriebnahme komplexer Automatisierungssysteme
- umfangreiches Schulungsprogramm für Beckhoff Systemkomponenten

Hotline:	+49(0)5246 963 157
Fax:	+49(0)5246 963 9157
E-Mail:	support@beckhoff.com

Beckhoff Service

Das Beckhoff Service-Center unterstützt Sie rund um den After-Sales-Service:

- Vor-Ort-Service
- Reparaturservice
- Ersatzteilservice
- Hotline-Service

Hotline:	+49(0)5246 963 460
Fax:	+49(0)5246 963 479
E-Mail:	service@beckhoff.com

Beckhoff Firmenzentrale

Beckhoff Automation GmbH & Co. KG

Hülshorstweg 20
33415 Verl
Deutschland

Telefon:	+49(0)5246 963 0
Fax:	+49(0)5246 963 198
E-Mail:	info@beckhoff.com
Internet:	https://www.beckhoff.de

Mehr Informationen:
www.beckhoff.de/te1000

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Deutschland
Telefon: +49 5246 9630
info@beckhoff.de
www.beckhoff.de