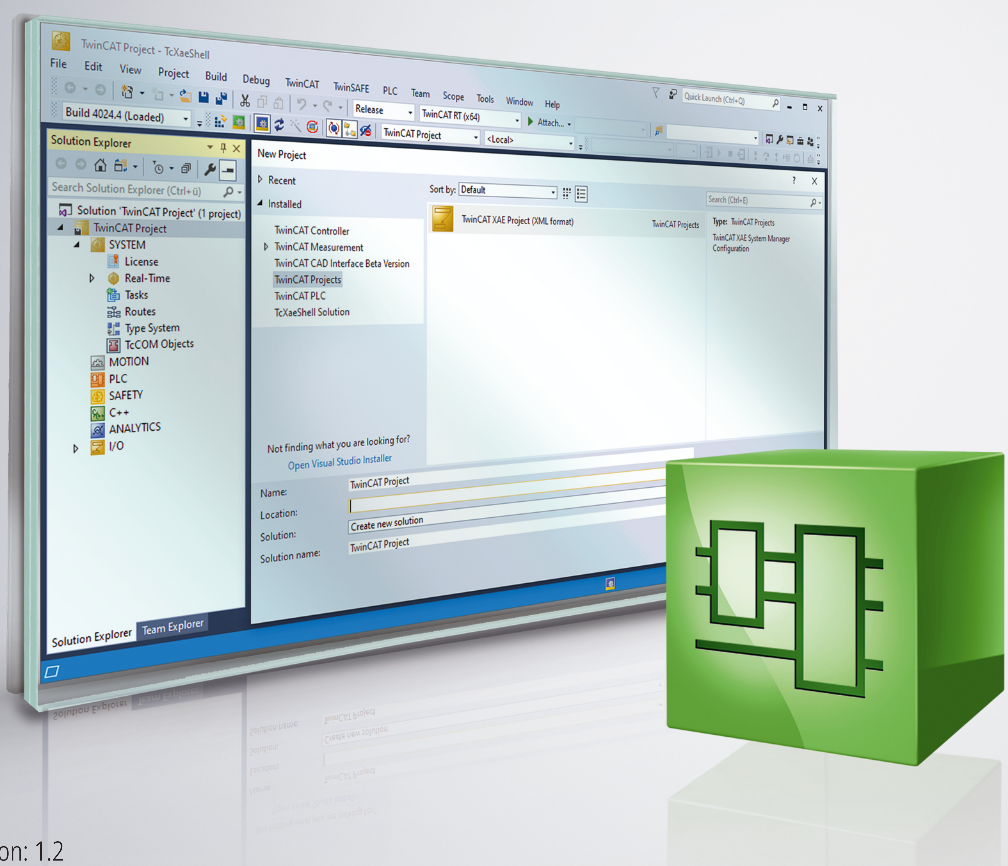
BECKHOFF New Automation Technology

Manual | EN

TE1000

TwinCAT 3 | PLC Library: Tc3_BA_Common



2021-12-15 | Version: 1.2

Table of contents

1 Foreword

1.1 Notes on the documentation

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

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

Disclaimer

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

Trademarks

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

Patent Pending

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

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

Copyright

1.2 Safety instructions

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Description of symbols

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

⚠ DANGER

Serious risk of injury!

Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.

⚠ WARNING

Risk of injury!

Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.

⚠ CAUTION

Personal injuries!

Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.

NOTE

Damage to the environment or devices

Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.

Tip or pointer

This symbol indicates information that contributes to better understanding.

1.3 Notes on information security

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

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

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

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

2 Introduction

The TwinCAT3 Building Automation library (TC3 BA Common) provides controller function blocks and a sequence linker function block.
These function blocks are used by both the TC3_BA library and the Tc2_BACnetRev12 library.

3 General Information

Further libraries required

For PC systems and Embedded PCs (CXxxxx):

- Tc2_Standard
- Tc2_TcBase
- Tc2_TcSystem
- Tc2_TcUtilities

4 Programming

4.1 POUs

4.1.1 Controller

Function blocks

Name	Description
FB_BA_SeqCtrl [▸ 16]	Sequence controller (see Introduction – sequence controller [▸ 10]).
FB_BA_SeqLink [▸ 20]	Sequence controller control function block.
FB_BA_PIDCtrl [▸ 13]	Universal PID controller.

4.1.1.1 Introduction – sequence controller

In heating, ventilation and air-conditioning systems, it is often the case that several actuators, working in a so-called controller sequence, are used in order to achieve a control variable.

In the air conditioning system shown below, three actuators are involved in the regulation of the supply air temperature. In the project a dedicated sequence controller is instanced for each of these actuators.

During active control only one of these sequence controllers is active. The other, non-active controllers fix their control signal so that it is energetically optimal for the tempering of the inlet air temperature.

Depending on the direction of action of the individual controller, this means either the maximum or the minimum for the control value *lrY*.

If the effect of the active actuator (controller) is insufficient when reaching an end position, the active controller switches to the adjacent controller to the left or right.
This then takes over control. The previously active controller remains at the end position of *lrYMax* or *lrYMin*, depending on the direction of action. This is repeated with the remaining actuators until the set value or the left or right end of the sequence is reached.

In the sequence of the illustrated air conditioning system, all actuators that influence the control variable are shown from left to right. At the far left is the actuator that enables the greatest possible increase in the supply air temperature; at the far right is the actuator that effects the greatest possible decrease in the supply air temperature.

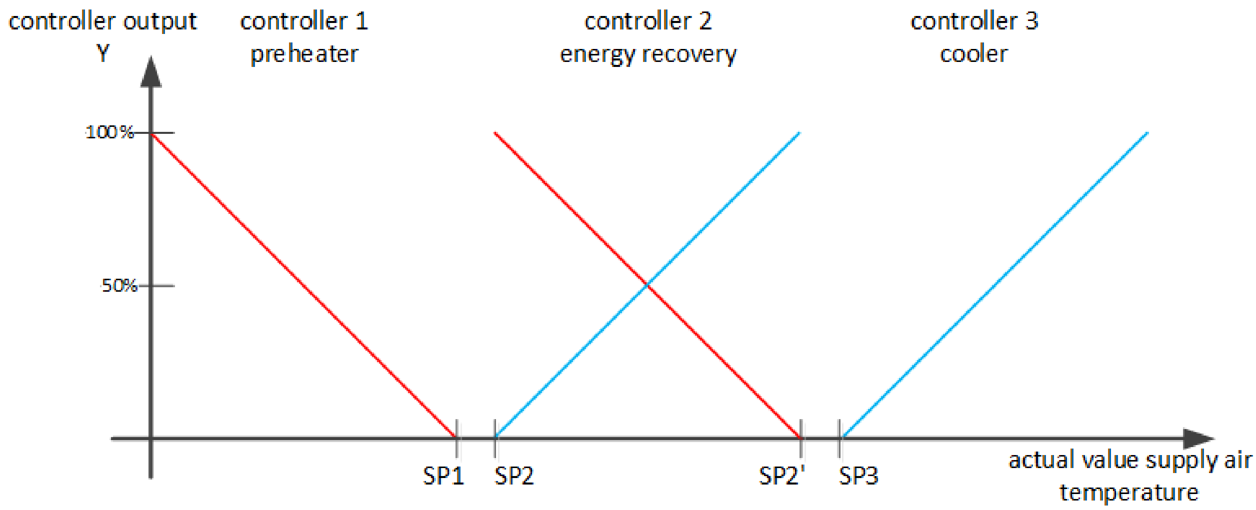
Some actuators, such as a recirculating air flap or a heat recovery unit change their direction of action during operation. (indirect = heating, direct = cooling)

Actuators with varying direction of action, such as outside air flap, recirculating air flap or heat recovery unit, are only listed once.

- 1: Preheater controller
- 2: Mixed air controller
- 3: Cooler

Schematic diagram

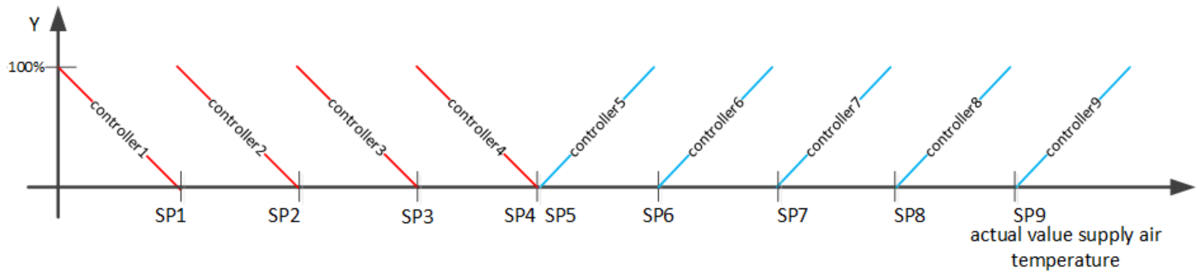
This plant is schematically represented as follows:



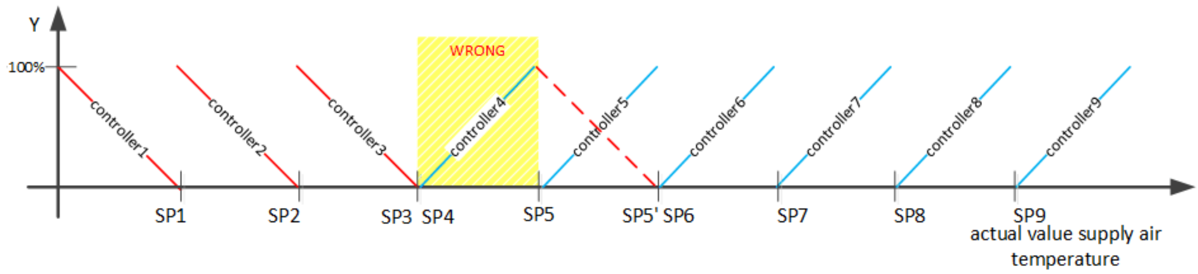
Rules for creating a sequence

The following rules must be followed for creating the sequences; inlet air control is used as reference:

- The sequence controllers are numbered starting with the heating sequences with low ordinal numbers to the cooling sequences with high ordinal numbers.



- A series of heating sequences should not include a cooling sequence. Similarly, a series of cooling sequences should not include a heating sequence. Sequences with reversal of direction of action for a mixed air system or heat recovery should be positioned between the heating and cooling sequences.



In this diagram controller 4 would be placed incorrectly, if controller 5 changed to heating mode. Or: Controller 4 is correct, but controller 5 would have to be a pure cooling controller.
In both cases there would be two switches from heating to cooling.

- The set values within the sequence must be monotonically increasing. This requirement is a result of the switching behavior explained above: If the set value of a controller with a lower number is higher than the next higher one, the result could be continuous switching between the two controllers. As mentioned above, controllers with the same direction of action usually have the same set value.
SP1 ≤ SP2 ≤ SP3 ≤ SP4 ≤ SP5 ≤ SP6 ≤ SP7 ≤ SP8 ≤ SP9

Sequence controllers in the PLC

The *TC3_BA_Common library* provides two function blocks for the implementation of a sequence controller in the PLC program:

The function block FB_BA_SeqCtrl [▶ 16]: This function block provides an individual controller as part of a sequence of up to 16 controllers.

The function block FB_BA_SeqLink [▶ 20]: This function block is the control function block of the sequence and therefore only exists once per sequence. It decides which controller of the sequence is currently active and checks the sequence for certain error states, such as duplicate allocation of ordinal number at the controllers.

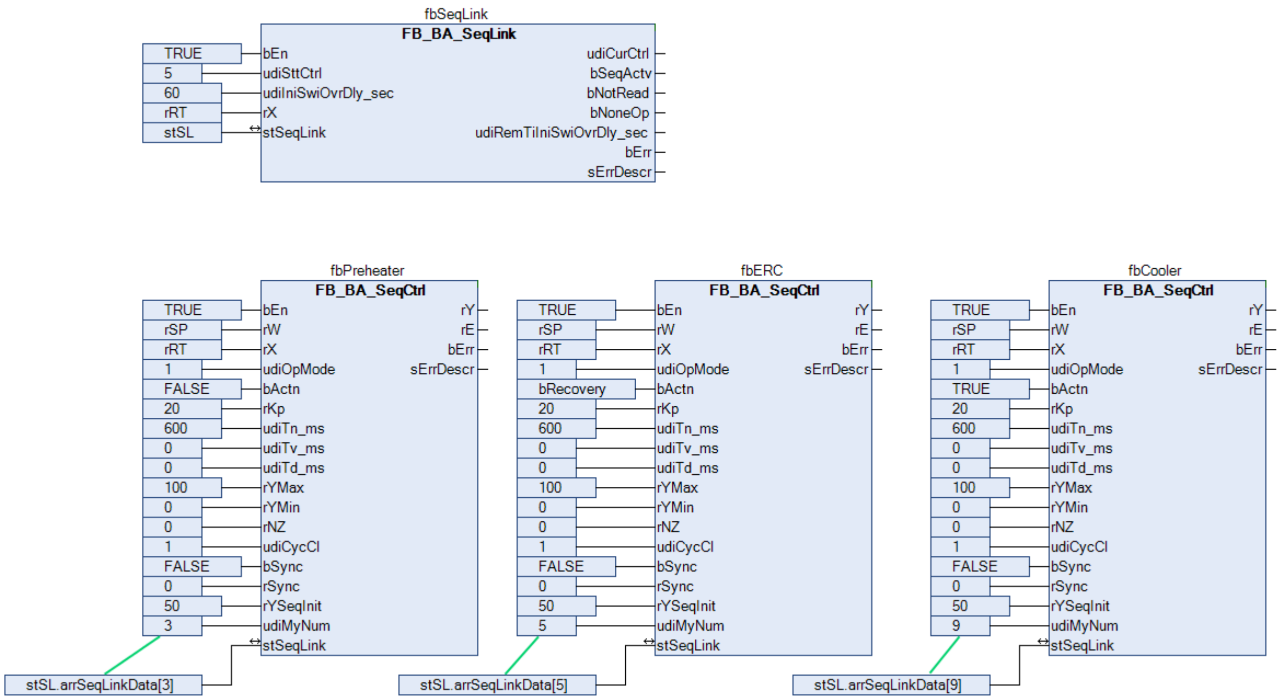
The structure variable ST_BA_SeqLink [▶ 40] is used to link the sequence controllers with the sequence linker FB_BA_SeqLink [▶ 20].

This structure variable has to be declared once per sequence control.

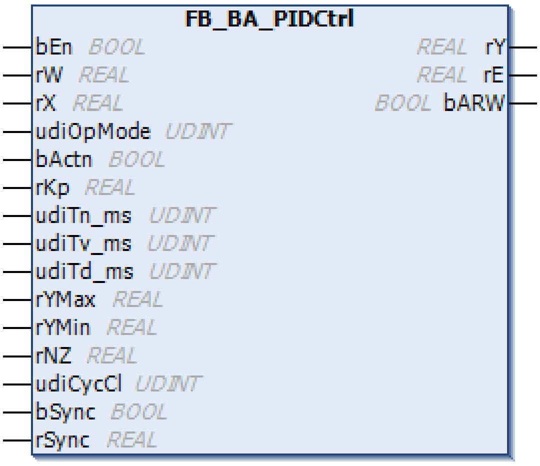
The sequence control is enabled at input *bEn* of the function block FB_BA_SeqLink [▶ 20]. The variable *usiStartCtrl* is used to determine which controller is used as the first one after the start of control mode. In the example, the sequence controller with the no. 5 is assigned as the start controller. Switching from controller 5 to another controller in the sequence after restarting the control is blocked for the value of the input variable *udiIniSwiOvrDly_sec*.

```
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



Universal PID controller, alternatively in parallel structure or with upstream proportional component.

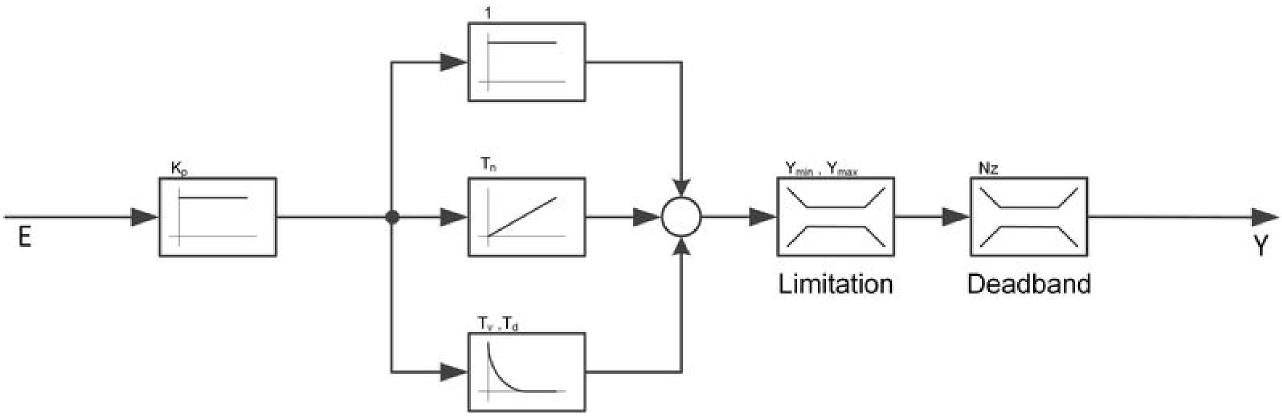
Functional description

This controller is divided internally into two consecutive parts:

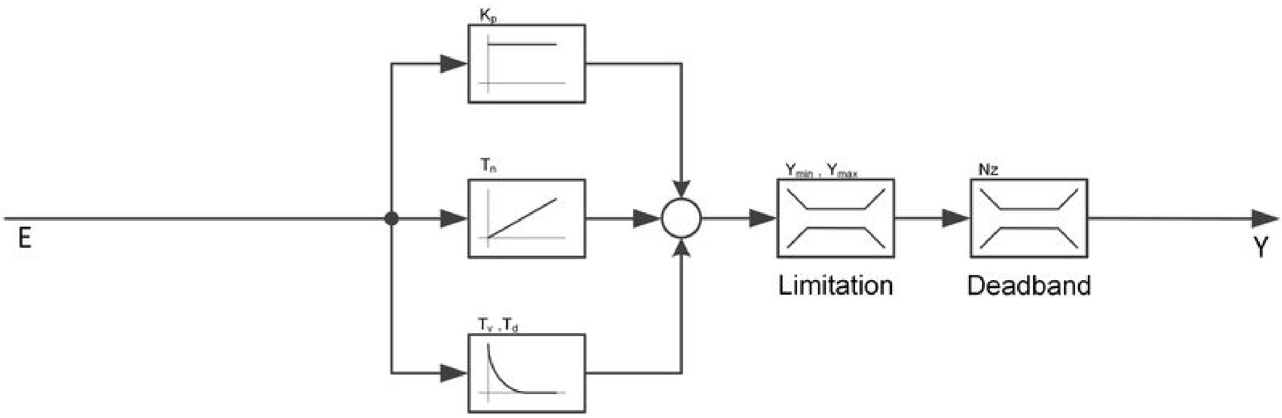
- the controller itself, illustrated in the functional diagrams below as P, I and D component with an output limitation.
- a deadband element that applies a hysteresis to the output changes of the controller.

Functional diagram

udiMode = 0 (upstream proportional component):



udiMode = 1 (parallel structure):



Passive behavior (bEn = FALSE)

The outputs are set as follows:

rY	0.0
rE	0.0
bARW	FALSE

The internal values for the P, I, and D components are set to 0, also the values for the I and D components of the preceding cycle. In case of a restart the control value is thus calculated in the first cycle without past values.

Active behavior (bEn = TRUE)

In the first cycle, the I and D components are calculated "clean", i.e. without historical values, as already mentioned.

Synchronization

A positive signal at *bSync* sets the I component such that the control value assumes the value *rSync*. If *bEn* and *bSync* are set at the same time, this method can be used to set an initial value as a starting point for the control. If the I component is not active, the D component is set accordingly. Note that internally only the rising edge of *bSync* is evaluated, in view of the fact that it is a set action. For a further synchronization, e.g. with a transfer value, a TRUE signal must be set again at input *bSync*.

Anti-Reset-Windup

If the I component is active, the controller ensures that it this maintained if the controller output *rY* should try to go beyond the limits *rYMin* or *rYMax.* A preliminary calculation of the controller output takes place inside the controller in every cycle. If this is smaller than the lower output limit *rYMin* or greater than the upper limit *rYMax*, then the I component is adjusted in such a way that the sum of the P, I and D components results in *rYMin* or *rYMax* respectively. This ensures that the I component is always just large enough so that the control value can immediately assume values within the limits in the case of a corresponding control deviation without an integral component that has become too large having to be reduced first.

Direction of action

bActn = FALSE can be used to reverse the direction of action such that a control deviation of less than 0 results in a change in control value to positive. This is achieved by a negative calculation of the control deviation:

bActn	rXW (control deviation)	Direction of action
TRUE	*rX-rW* (actual value-set value)	direct *(cooling)*
FALSE	*rW-rX* (set value-actual value)	indirect (heating)

Neutral zone

A value of *rNZ* > 0.0 enables the function of the neutral zone (deadband). A value equal to zero deactivates the deadband element and the values at the input are passed directly through.

If, in the active case, the change at the input of the element *rYin* in a PLC cycle is smaller than *rNZ*/2 in comparison with the previous PLC cycle, then the output is held at the value of the previous cycle until the change is larger than or equal to *rNZ*/2.

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

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

This function is intended to avoid an unnecessarily large number of actuating pulses.

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: Controller activation.

rW: Set value.

rX: Actual value.

udiOpMode: *udiMode* = 0: Controller with upstream proportional component, *udiMode* = 1: Controller in parallel structure. Internally limited to the values 0 and 1.

bActn: Direction of action [▶ 14] of the controller.

rKp: Controller gain. Only affects the proportional component. Internally limited to a minimum value of 0.

udiTn_ms: Integral action time of the I component [ms]. A zero value at this parameter disables the I component. Internally limited to a minimum value of 0.

udiTv_ms: Rate time of the D component [ms]. A zero value at this parameter disables the D component. Internally limited to a minimum value of 0.

udiTd_ms: Damping time of the D component [s]. Internally limited to a minimum value of 0.

rYMax: Upper controller output limit. Selectable range: 0..100%.

rYMin: Lower controller output limit [%]. Selectable range: 0..100%. The value *rYMin* is upwardly limited by *rYMax*.

rNZ: Neutral zone.

udiCycCl: Call cycle of the function block as a multiple of the cycle time. Internally limited to a minimum value of 1.

Example: *tTaskCycleTime* = 20ms, *udiCtrlCycleCall* =10 -> The control algorithm is called every 200 ms. Thus the outputs are also updated only every 200 ms.

bSync / rSync: Synchronization command: Set output value *rY* to *rSync.* The value *rSync* is limited internally to values ranging from *rYMin* to *rYMax*.

VAR_OUTPUT

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

rY: Control value. Range limited by *rYMin* and *rYMax.*

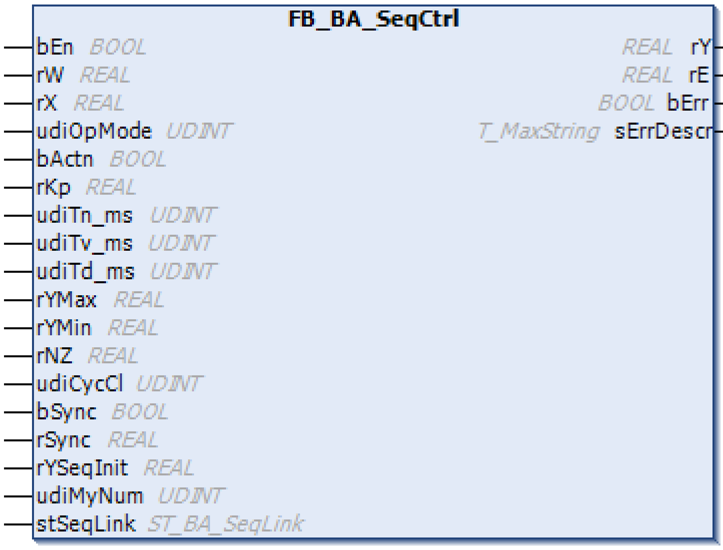
rE: Control deviation (The calculation depends on the direction of action [▸ 14]).

bARW: Anti-Reset-Windup function is active.

Requirements

Development environment	Required library	Necessary function
TwinCAT3.1 4022.16	Tc3Building Automation Common from V1.0.4.3	TF8040 | TwinCAT Building Automation from V1.0.5.0

4.1.1.3 FB_BA_SeqCtrl

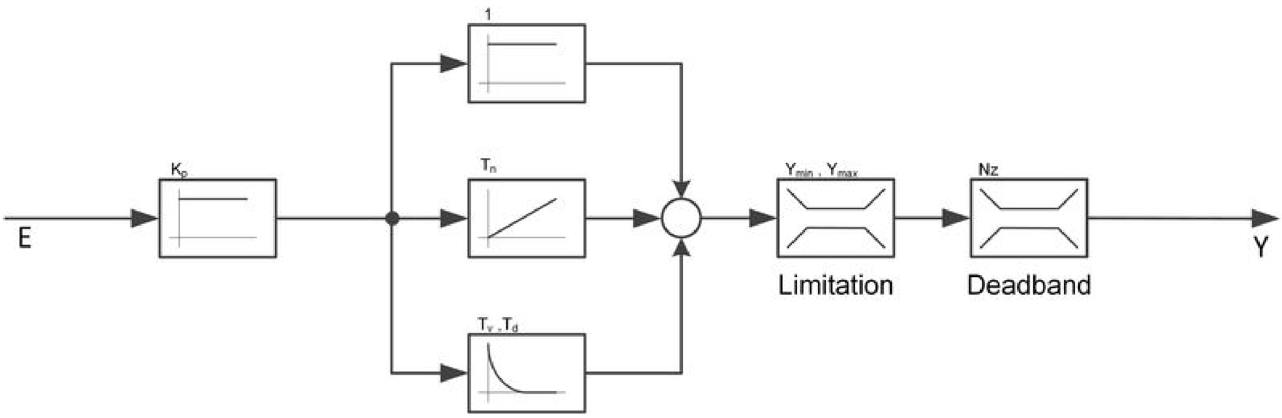


PID controller as part of a sequence.

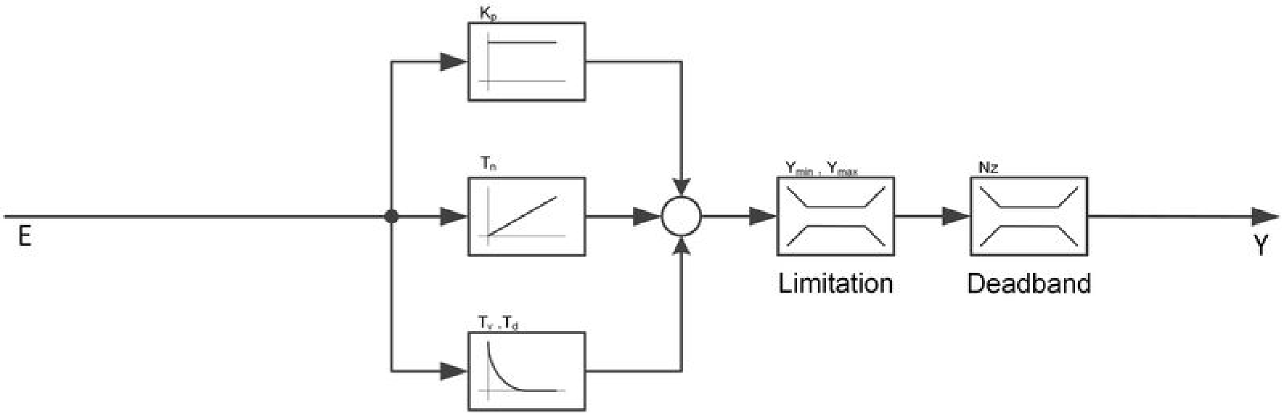
Functional description

The functionalities of this controller are identical to FB_BA_PIDCtrl [▸ 13].

udiOpMode = 0 (upstream proportional component)



udiOpMode = 1 (parallel structure)



In addition the controller, if enabled by *bEn* = TRUE, is controlled via a higher level control function block FB_BA_SeqLink [▸ 20].

The data exchange between the control function block FB_BA_SeqLink [▸ 20] and the sequence controllers *FB_BA_SeqCtrl* takes place via the structure variable stSeqLink [▸ 40].

Heating-cooling sequence

The controller sequence should be configured such that the sequence controller with lower ordinal number are used for heating and the ones with the higher number for cooling. Only one change is permitted:

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

Exclusive programming of cooling and heating controllers is also possible.

Any parameterization that contradicts this convention is detected and indicated as an error at control function block FB_BA_SeqLink [▸ 20].

Controller output

The control function block FB_BA_SeqLink [▸ 20] specifies which sequence controller is active. What is output at the respective control output *rY* is determined inside the individual sequence controllers. Each controller receives the information about the states of the other controllers via the in-out variable *stSeqLink* and evaluates four cases internally.

1. None of the sequence controllers is enabled, whether due to a missing enable signal (*bEn*) at the input or due to an error detected on the control function block FB_BA_SeqLink [▸ 20]
   -> The internal PID controllers are inactive and output 0.0 at the control output *rY*.
2. The sequence controller is enabled and has been set to active by the control function block FB_BA_SeqLink [▸ 20].
   -> The internal PID controller is active. Its output signal is output at the control output *rY*.
3. The sequence controller is enabled, but a sequence controller with a **higher** ordinal number has been set to active by the control function block FB_BA_SeqLink [▸ 20]
   -> The internal PID controller is inactive. If the sequence controller is in heating mode (*bActn*=FALSE), it will output its minimum value *rYMin* at the control output *rY*. Conversely, if it is in cooling mode (*bActn* = TRUE), then it outputs the maximum value *rYM*ax at the control output *rY*.
4. The sequence controller is enabled, but a sequence controller with a **lower** ordinal number has been set to active by the control function block FB_BA_SeqLink [▸ 20]
   -> The internal PID controller is inactive. If the sequence controller is in heating mode (*bActn*=FALSE), it will output its maximum value *rYMax* at the control output *rY*. If it is in cooling mode (*bActn*=*T*RUE), it will output its minimum value *rYMin* at the control output *rY*.

Synchronization

If a sequence controller is activated by the higher-level controller, this always results in synchronization, i.e. the controller starts with a fixed value at the output *rY*. 3 cases are distinguished:

1. The entire sequence control was switched on via the input *bEn* of the higher-level controller FB_BA_SeqLink [▸ 20]. The controller with the ordinal number *udiSttCtrl* at the input of FB_BA_SeqLink [▸ 20] is the start controller.
   -> The sequence controller is synchronized with the value, which is entered at its input *rYSeqInit*.
2. The sequence controller, which has just has been activated, had a higher ordinal number than the "previous" one
   -> If the sequence controller is in heating mode (*bActn* = FALSE), then it is synchronized with its minimum value *rYMin*. If it is in cooling mode (*bActn*=TRUE), the synchronization value is its maximum value *rYMa*x.
3. The sequence controller, which has just has been activated, had a lower ordinal number than the "previous" one
   -> If the sequence controller is in heating mode (*bActn* = FALSE), then it is synchronized with its maximum value *rYMax*. If it is in cooling mode (*bActn*=TRUE), the synchronization value is its minimum value *rYMin*.

Each sequence controller can also be synchronized by specifying a value *rSync* and activating *bSync*, if it has just been activated by the higher-level controller. A constant TRUE signal at the input *bSync* (e.g. accidental) is internally intercepted through edge formation, so that obstruction of the synchronization described above on activation is avoided.

Start-up behavior

In order to enable "sensible" adjustment of the entire control sequence, the start controller is maintained in active state as a minimum for the time *udiIniSwiOvrDly_sec* [s], which is entered at the function block FB_BA_SeqLink [▸ 20]. During this time, no switching takes place to another controller of this sequence. The output *rY* of the start controller is synchronized **once** to its value *rYSeqInit*.

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: Activation of the sequence controller

rW: Set value

rX: Actual value

udiOpMode: *udiOpMode*=0: Controller with upstream proportional component, *udiOpMode* = 1: Controller in parallel structure. The values are limited internally to 0 and 1.

bActn: Direction of action reversal of the controller. For heating/cooling operation: *bActn*=FALSE corresponds to heating mode, *bActn*=TRUE corresponds to cooling mode.

rKp: Controller gain. Only affects the proportional component. Internally limited to a minimum value of 0.

udiTn_ms: Integral action time of the I component [ms]. A zero value at this parameter disables the I component. Internally limited to a minimum value of 0.

udiTv_ms: Rate time of the D component [ms]. A zero value at this parameter disables the D component. Internally limited to a minimum value of 0.

udiTd_ms: Damping time of the D component [s]. Internally limited to a minimum value of 0.

rYMax: Upper controller output limit [%]. Selectable range: 0..100%.

rYMin: Lower controller output limit [%]. Selectable range: 0..100%. The value *lrMin* is upwardly limited by *lrYMax*.

rNZ: neutral zone (see Deadband diagram). Internally limited to a minimum value of 0. Mode of action is same as FB_BA_PID_Ctrl [▸ 13].

udiCycCl: Call cycle of the function block as a multiple of the cycle time. Internally limited to a minimum value of 1.
Example: *tTaskCycleTime* = 20 ms, *udiCycCl* = 10 -> the control algorithm is called every 200 ms. Thus the outputs are also updated only every 200 ms.

bSync / rSync: Synchronization command: Set output value *rY* to *rSync.* The value *rSync* is limited internally to values ranging from *rYMin* to *rYMax*.

rYSeqInit: Starting value of the controller after restart of the whole control sequence.

udiMyNum: Ordinal number of the sequence controller. Internally limited to values ranging from 0 to *gBA_cMaxSeqCtrl*.

VAR_OUTPUT

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

rY: Control value. Section: 0..100%, unless limited further by *rYMin* and *rYMax.*

rE: Control deviation (The calculation depends on the direction of action [▸ 14]).

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

sErrDescr: Contains the error description.

Error description
01: Error: The controller ordinal number *udiMyNum* has been assigned twice
02: Error: The controller ordinal number *udiMyNum* of the enabled controller is 0. That is only allowed for controllers that are not in use and thus not enabled.

VAR_IN_OUT

```
stSeqLink : ST_BA_SeqLink;
```

stSeqLink: Data and command structure (see ST_BA_SeqLink / ST_BA_SeqLinkData [▸ 40]) between the individual sequence controllers and the control function block FB_BA_SeqLink [▸ 20].

If several sequence controllers have the same number (*diMyNum*), this is detected and output as an error at the sequence controller and at the control function block FB_BA_SeqLink [▸ 20].

Requirements

Development environment	Required library	Necessary function
TwinCAT3.1 4022.16	Tc3Building Automation Common from V1.0.4.3	TF8040 | TwinCAT Building Automation from V1.0.5.0

4.1.1.4 FB_BA_SeqLink

This function block represents the higher-level control unit, which specifies which sequence controller is currently active.

The data exchange between the control function block FB_BA_SeqLink and the sequence controllers FB_BA_SeqCtrl [▸ 16] takes place via the structure variable stSeqLink [▸ 40].

Functional description

Start-up behavior

A TRUE signal at input *bEn* activates the entire sequence control. The function block will initially activate the sequence controller named on *udiSttCtrl*. All other sequence controller base their output value on the ranking of the active controller, see FB_BA_SeqCtrl [▸ 16]. The start controller will be set once to its value *rSync* at the start of the sequence.
In order to enable "sensible" adjustment of the entire control sequence, the start controller is maintained in active state as a minimum for the time *udiIniSwiOvrDly_sec* [s]. During this time, no switching takes place to another controller of this sequence.

Switching behavior

When the sequence controller reaches its maximum or minimum value, the next controller in the sequence is activated, depending on the controller direction of action, if the actual value is below or above the set value of the next controller.
4 cases are distinguished:

- The still active controller has direct direction of action (cooling) and is at its maximum value: The next higher controller in the sequence will be selected if the actual value exceeds the set value for this controller.
- The still active controller has direct direction of action (cooling) and is at its minimum value: The next lower controller in the sequence is then selected, if the actual value falls below the set value for this controller.
- The still active controller has indirect direction of action (heating) and is at its maximum value: The next lower controller in the sequence is then selected, if the actual value falls below the set value for this controller.
- The still active controller has indirect direction of action (heating) and is at its minimum value: The next higher controller in the sequence will be selected if the actual value exceeds the set value for this controller.

Switch-off behavior

If the enable status is removed from a controller within the sequence or if it develops a fault, it is no longer available for the whole sequence.
If this is not the previously active controller, a temperature change may occur, depending on which control value this controller has output, which is compensated by the controller sequence, if possible.
However, if it is the active controller whose enable is canceled, the next "sensible" controller must be selected. The sequence link function block uses the following rules:

- The deactivated controller had direct direction of action (cooling)

There is an operational controller with a higher ordinal number → switch to the next higher operational controller.
Only an operational controller with lower ordinal number is available → switch to the next lower operational controller.
No operational controller is available → fault message

- The deactivated controller had indirect direction of action (heating)
- An operational controller with lower ordinal number is available → switch to the next lower operational controller.
  There is an operational controller with a higher ordinal number → switch to the next higher operational controller.
  No operational controller is available → fault message

Sequence behavior

If a controller is added to the sequence, it is in any case initially inactive and will output its minimum or maximum value, depending on the direction of action and positioning within the sequence order. The resulting temperature change is compensated by the controller sequence, if possible.

VAR_INPUT

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

bEn: Activation of the sequence controller.

udiSttCtrl: Ordinal number of the sequence controller that should be the start controller upon general activation. Internally limited to values ranging from 0 to *gBA_cMaxSeqCtrl*.

udiIniSwiOvrDly_sec: The first controller remains active for at least this time [s] in the sequence before other criteria (see FB_BA_SeqLink [▸ 20]) allow switching to a different controller.

rX: Actual value of the control.

VAR_OUTPUT

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

udiCurCtrl: Ordinal number of the currently active sequence controller. If no controller is active, 0 is output here.

bSeqActv: The sequence function block is enabled (*bEn*) and has no error resulting in switch-off, see error detection.

bNotRead: Each sequence controller transfers data to the control function block via the structure *stSeqLink*. This output is TRUE, as long as no data were transmitted - this is the case when the PLC is switched on.

bNoneOp: This output is switched to TRUE, if none of the sequence controller is enabled (*bEn*=TRUE).

udiRemTiIniSwiOvrDly_sec: Remaining initialization time [s] before switching for the first time (see FB_BA_SeqLink [▸ 20]).

bErr: This output is switched to TRUE if the parameters entered are erroneous. This function block may not suspend its execution in the event of an error, see error detection.

sErrDescr: Contains the error description.

Error description
01: Error: The sequence link has been informed that the controller ordinal number *udiMyNum* has been assigned twice.
02: Warning: Direction of action changed twice in the controller sequence.
03: Warning: In the controller sequence, a controller with a higher ordinal number has a lower set value than its "predecessor". No correction takes place; the controller sequence runs with the parameters that were entered.
04: Warning: The sequence controller, which is defined as start controller (*udiSttCtrl*) is not parameterized at all, i.e. it is not present. The controller with the lowest ordinal number is used as start controller.
05: Warning: The ordinal number of the start controller is higher than the maximum permitted number of controllers or zero. The controller with the lowest ordinal number is used as start controller.
06: Warning: The sequence controller, which is defined as start controller (*udiSttCtrl*) is not enabled (present). The controller with the lowest ordinal number is used as start controller.

Only the first error triggers a fault in the sequence link function block and blocks its execution (bSeqActv = FALSE). All associated controllers are then no longer active, and all controllers issue the control value "0". The function block is not active:

VAR_IN_OUT

```
stSeqLink : ST_BA_SeqLink;
```

stSeqLink: Data and command structure (see ST_BA_SeqLink / ST_BA_SeqLinkData [▶ 40]) between the individual sequence controllers and the control function block FB_BA_SeqLink. This structure is used by the sequence link function block to receive all relevant sequence controller data and at the same time to notify the controllers which is the active one.

i If several sequence controllers have the same number (*udiMyNum*), this is detected and output as an error at the sequence controller and at the control function block.

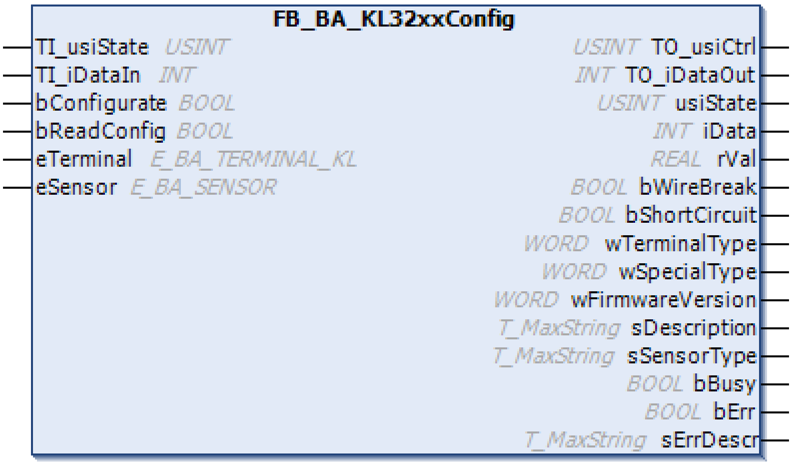
Requirements

Development environment	Required library	Necessary function
TwinCAT3.1 4022.16	Tc3Building Automation Common from V1.0.4.3	TF8040 | TwinCAT Building Automation from V1.0.5.0

4.1.2 Universal

4.1.2.1 Analog inputs/outputs

4.1.2.1.1 FB_BA_KL32xxConfig



Configuration of the Bus Terminals for temperature measurement.

Functional description

The function block is for the configuration of Bus Terminals of the types KL3208_0010, KL3201, KL3202 and KL3204.

VAR_INPUT

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

TI_usiState: Linking with the corresponding status byte of the Bus Terminal in the I/O area of the program.

TI_iDataIn: Linking with the corresponding raw data (Data In) of the Bus Terminal in the I/O area of the program (0 - 32767).

bConfigurate: A rising edge starts the configuration of the Bus Terminal.

bReadConfig: A rising edge starts the reading of the Bus Terminal.

eTerminal: Selection of the respective Bus Terminal (see E_BA_Terminal_KL [▸ 41]).

eSensor: Selection of the sensor type (see 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: Linking with the corresponding control byte of the Bus Terminal in the I/O area of the program.

TO_iDataOut: Linking with the corresponding raw data (Data Out) of the Bus Terminal in the I/O area of the program.

usiState: Output of the present terminal status.

iData: Output of the present process data.

rVal: Scaled output value.

bWireBreak: Display of the channel status, sensor wire breakage.

bShortCircuit: Display of the channel status, sensor short-circuit.

wTerminalType: Display of the terminal type.

wSpecialType: Display of the special version of the terminal.

wFirmwareVersion: Display of the terminal firmware.

sDescription: Display of the terminal type and firmware.

sSensorType: Display of the sensor type.

bErr: Error in the terminal configuration.

sErrDescr: Contains the error description.

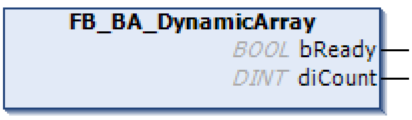
Error description
01: Error: Check the terminal configuration KL32xx *eTerminal/eSensor/TI_usiState/TI_iDataIn/TO_usiCtrl/TO_iDataOut*

Requirements

Development environment	Required library	Necessary function
TwinCAT3.1 4022.16	Tc3Building Automation Common from V1.0.4.3	TF8040 | TwinCAT Building Automation from V1.0.5.0

4.1.2.2 Array

4.1.2.2.1 FB_BA_DynamicArray



The function block generates and deletes memory areas dynamically so that entries can be added and removed at runtime.

As soon as the maximum capacity [▶ 28] of the array is reached, the internal memory area is automatically extended. If the capacity is more than sufficient, the internal memory area is reduced in size.

> The internally used memory is allocated from the router memory pool and is generated via _NEW and released via _DELETE at runtime.
> With each adaptation (i.e. extension or reduction in size) of the internal memory, the pointers to the obsolete/adapted memory are also invalid!

The data type of the entries is not important for the dynamic array!
The user must ensure in every case that the data type is always correctly observed by the application when dealing with contained entries.
Furthermore, all data added to the array must have a uniformly defined size [▸ 25]!

It is recommended to use the dynamic array in particular in cases where the expected memory utilization can be estimated relatively well. Router memory is only available to a limited extent (especially with small controllers) and is to be used as efficiently as possible! If necessary, the amount of router memory available in the target system must additionally be adapted.

VAR_OUTPUT

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

bReady: Status of the allocated memory. (TRUE if at least one entry is contained in the array and memory is thus already generated)

diCount: Current number of entries contained.

VAR

Internal variables that have to be initialized during the declaration.

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

uiEntrySize: Expected size of entries. Used to allocate internal memory and to manage memory areas of recorded entries.

uiMinExpCount: Expected size of the internal memory (specified in [number of entries]) on reaching the maximum capacity [▸ 28].

For further information, see examples [▸ 25] **of initialization during variable declaration.**

Application

Two typical application cases are imaginable:

Case 1) Array contains data sets

In this case the array contains data sets (generic types such as BOOL, INT, STRING or structures) by reserving internal memory in accordance with the size of the type used.

Case 2) Array contains pointers

In this case the array contains pointers to externally declared data and only memory corresponding to the size of memory addresses is reserved.

Instances of the dynamic array are not called cyclically. It is sufficient to use the management functions and properties described here.

Examples

Example 1:

Data sets of the data type *ST_DATA* are stored in an array.

Access to the respective data sets takes place by means of pointers to the internal memory of the array or by means of a copy of a data set.

```
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
```

Example 2:

The addresses of externally declared instances of the function block *FB_Object* are stored in an array.

```
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
```

Error messages

The following error messages may be output in the TwinCAT display window at runtime:

[EDB4] Entry-size of array not defined!
The expected size [▶ 25] of entries was not initialized during the declaration of the array.

[EDB7] Expansion-count of entries not defined!
The expected size [▶ 25] of the internal memory was not initialized during the declaration of the array.

Methods of FB_BA_DynamicArray

Name	Definition location	Description
AddEntry [▸ 28]	Local	Creates a new data set at the end of the array and copies the content of the specified entry to the internal memory
FindEntry [▸ 28]	Local	Determines the position of the specified entry in the array by comparing its content with the data sets of the array.
GetEntry [▸ 29]	Local	Copies the contents of the data set to a certain position in the specified memory area.
GetEntryEx [▸ 29]	Local	Determines a pointer to the internal memory of the specified data set.
RemoveEntry [▸ 29]	Local	Removes the data set at the specified index position from the array.
RemoveEntryEx [▸ 30]	Local	Determines the position of the specified entry and deletes it from the array.
Reset [▸ 30]	Local	Resets the complete content of the array.
SetEntry [▸ 30]	Local	Replaces the existing data set with a new one by overwriting the internal memory area of the existing data set with the value of the new entry.
AddEntryPtr [▸ 31]	Local	Creates a new entry at the end of the array and copies its memory address (i.e. the address to which the pointer *pEntry* points) to the internal memory.
FindEntryPtr [▸ 31]	Local	Determines the position of an entry in the array by comparing its address with the addresses stored in the array.
GetEntryExPtr [▸ 31]	Local	Outputs a pointer to the memory address of the requested entry.
RemoveEntryExPtr [▸ 32]	Local	Determines the position of the specified entry and deletes it from the array.
SetEntryPtr [▸ 32]	Local	Replaces an existing entry with a new one.

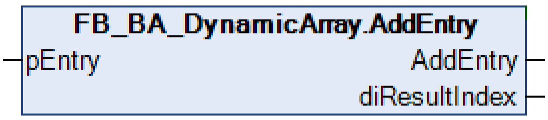
Properties of FB_BA_DynamicArray

Name	Type	Access	Definition location	Initial value	Description
CurCapacity	DINT	Get	Local	-	Current capacity of the array (number of entries). Corresponds to the maximum number of entries that can be accepted by the internal memory.
EntrySize	DINT	Get	Local	uiEntrySize [▸ 25]	Expected size of entries that are stored in the array
LastIndex	DINT	Get	Local	-	Index position of the last entry. This is -1 if no entries exist
UsedMemory	DINT	Get	Local	-	Size of the internal memory consumed [bytes].

Requirements

Development environment	Required library	Necessary function
TwinCAT3.1 4022.16	Tc3Building Automation Common from V1.0.4.3	TF8040 | TwinCAT Building Automation from V1.0.5.0

AddEntry



Creates a new data set at the end of the array and copies the content of the specified entry to the internal memory.

VAR_INPUT

```
pEntry     : PVOID;
```

pEntry: Pointer to the entry to be added.

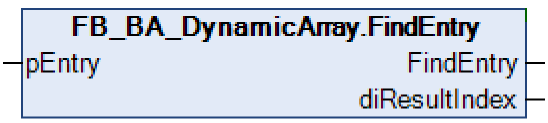
VAR_OUTPUT

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

AddEntry: Result of the function.

diResultIndex: Index position of the entry added.

FindEntry



Determines the position of the specified entry in the array by comparing its content with the data sets of the array.

VAR_INPUT

```
pEntry     : PVOID;
```

pEntry: Pointer to the entry sought.

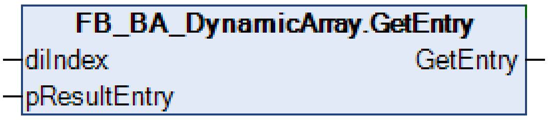
VAR_OUTPUT

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

FindEntry: Result of the function.

diResultIndex: Index position of the entry added.

GetEntry



Copies the contents of the data set to a certain position in the specified memory area.

VAR_INPUT

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

diIndex: Index position of the data set to be output.

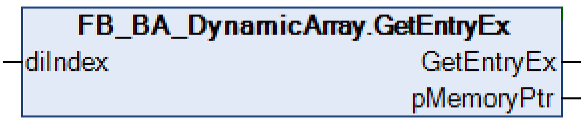
pResultEntry: Pointer to the memory area that is to be used to output the data record.

VAR_OUTPUT

```
GetEntry        : BOOL;
```

GetEntry: Result of the function.

GetEntryEx



Determines a pointer to the internal memory of the specified data set.

VAR_INPUT

```
diIndex        : DINT;
```

diIndex: Index position of the data set to be output.

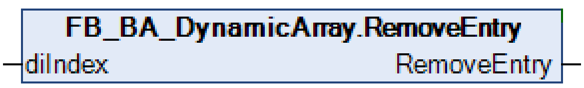
VAR_OUTPUT

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

GetEntry: Result of the function.

pMemoryPtr: Pointer that is to be used to output the data set.

RemoveEntry



Removes the data set at the specified index position from the array.

VAR_INPUT

```
diIndex     : DINT;
```

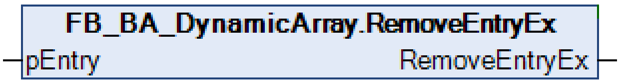
diIndex: Index position of the data set to be removed.

VAR_OUTPUT

```
RemoveEntry     : BOOL;
```

RemoveEntry: Result of the function.

RemoveEntryEx



Determines the position of the specified entry and deletes it from the array.

VAR_INPUT

```
pEntry     : PVOID;
```

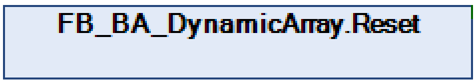
pEntry: Pointer to the entry to be removed.

VAR_OUTPUT

```
RemoveEntryEx     : BOOL;
```

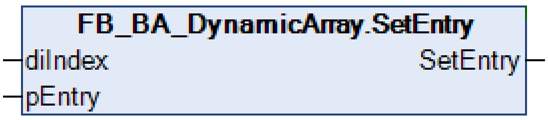
RemoveEntryEx: Result of the function.

Reset



Resets the complete content of the array.

SetEntry



Replaces the existing data set with a new one by overwriting the internal memory area of the existing data set with the value of the new entry.

VAR_INPUT

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

diIndex: Index position of the data set to be replaced.

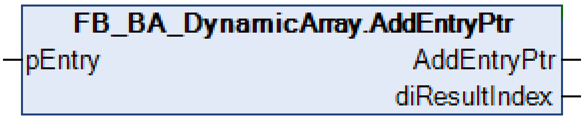
pEntry: Pointer to the entry to be removed.

VAR_OUTPUT

```
SetEntry     : BOOL;
```

SetEntry: Result of the function.

AddEntryPtr



Creates a new entry at the end of the array and copies its memory address (i.e. the address to which the pointer *pEntry* points) to the internal memory.

VAR_INPUT

```
pEntry     : PVOID;
```

pEntry: Pointer to the entry to be added.

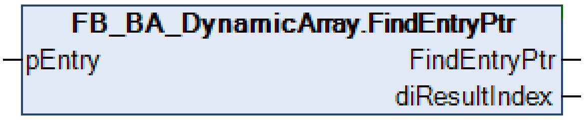
VAR_OUTPUT

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

AddEntryPtr: Result of the function.

diResultIndex: Index position of the entry added.

FindEntryPtr



Determines the position of an entry in the array by comparing its address with the addresses stored in the array.

VAR_INPUT

```
pEntry     : PVOID;
```

pEntry: Pointer to the entry sought.

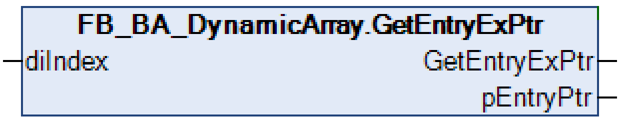
VAR_OUTPUT

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

FindEntryPtr: Result of the function.

diResultIndex: Index position of the entry sought.

GetEntryExPtr



Outputs a pointer to the memory address of the requested entry.

VAR_INPUT

```
diIndex     : DINT;
```

diIndex: Index position of the entry to be output.

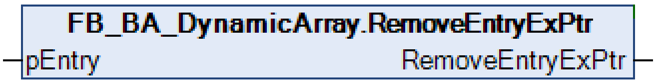
VAR_OUTPUT

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

GetEntryExPtr: Result of the function.

pEntryPtr: Pointer that is to be used to output the entry.

RemoveEntryExPtr



Determines the position of the specified entry and deletes it from the array.

VAR_INPUT

```
pEntry     : PVOID;
```

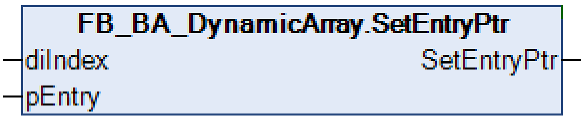
pEntry: Pointer to the entry to be removed.

VAR_OUTPUT

```
RemoveEntryExPtr     : BOOL;
```

RemoveEntryExPtr: Result of the function.

SetEntryPtr



Replaces an existing entry by a new one by overwriting the memory address of the existing entry with the memory address of the new entry.

VAR_INPUT

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

diIndex: Index position of the entry to be replaced.

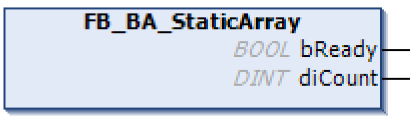
pEntry: Pointer to the entry to be replaced.

VAR_OUTPUT

```
SetEntryPtr     : BOOL;
```

SetEntryPtr: Result of the function.

4.1.2.2.2 FB_BA_StaticArray



The function block is an extension of the function block FB_DynamicArray [▶ 24].

The background to this extension is to avoid the use of router memory and to use static memory instead. This must be declared in the application, where its size can be adapted as desired.

The static memory must be provided by the application, but it must never be changed outside of the array! The management should take place in all cases via the array itself.

It is recommended to always use the static array in cases where the expected memory utilization can be precisely estimated.

For reasons of efficiency the memory size should be dimensioned such that as little memory as possible and as much memory as necessary is reserved. Global constants and parameter lists are suitable for declaring the limits of the memory area.

Application

In principle the application cases are identical to those with the dynamic array [▸ 24]. Only the declaration differs in part, as the external memory area and its size are to be transferred there.

Further information

See examples [▸ 33] of initialization during variable declaration.

Examples

Example declaration of the array, its static memory and corresponding constants.

```
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
```

Further information

Since the static array is to be used in exactly the same way as the dynamic array [▸ 24], appropriate application examples [▸ 25] are documented there.

Methods of FB_BA_StaticArray

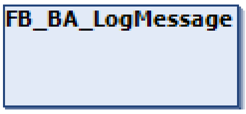
Name	Definition location	Description
AddEntry [▸ 28]	Local	Creates a new data set at the end of the array and copies the content of the specified entry to the internal memory
Reset [▸ 30]	Local	Resets the complete content of the array.

Requirements

Development environment	Required library	Necessary function
TwinCAT3.1 4022.16	Tc3Building Automation Common from V1.0.4.3	TF8040 | TwinCAT Building Automation from V1.0.5.0

4.1.2.3 Log

4.1.2.3.1 FB_BA_LogMessage



The function block outputs messages in the TwinCAT display window.

The symbol path of the function block is inserted for each message to be output so that the user can recognize the calling function block instance by the message.

Instances of this function block **cannot** be called explicitly.

Separate functions are available for various application cases, which are described in the following section of this documentation.

The functionality to output messages is provided by the ADSLOGDINT function, which is used internally.

VAR_OUTPUT

```
sResult     : T_MaxSTRING;
```

sResult: Content of the message last output.

Application

Context-related additional information

The developer has the possibility to output an abbreviation in each message.

Using this abbreviation, messages can be localized more easily in the source code (e.g. when searching for the abbreviation using the search function).

Suppression of cyclically repeated messages

In order to suppress the cyclic output of the same message, the current log code is compared with the log code used last. If the two values correspond, the output of the message is suppressed, which conversely means that different successive messages would be displayed.

This behavior can be influenced with the *bIgnoreBlock* option from the Show function:

TRUE prevents the suppression of a cyclically repeated message.

Example 1:

The example function *DoWork()* outputs a warning in row 150:

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

```
FUNCTION DoWork
```

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

Example 2:

The example function *Init()* outputs an error message in row 80 that could be FB_BA_LogMessage [▶ 34]:

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

```
  iState      : INT := 0;
  sDevice     : STRING := ‘CX9020’;
END_VAR
```

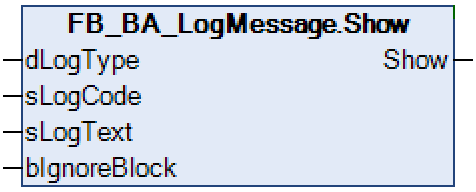
```
FUNCTION Init
```

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

Requirements

Development environment	Required library	Necessary function
TwinCAT3.1 4022.16	Tc3Building Automation Common from V1.0.4.3	TF8040 | TwinCAT Building Automation from V1.0.5.0

Show



Output of a simple message.

VAR_INPUT

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

dLogType: Log type of the message to be displayed.

sLogCode: Optional, context-related additional information [▶ 34].

sLogText: Content of the message.

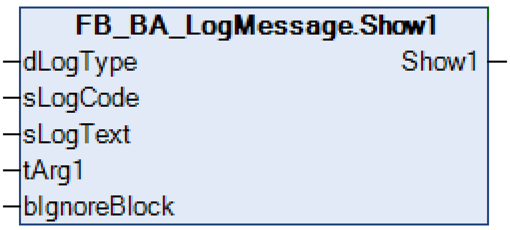
bIgnoreBlock: Prevents the suppression of cyclically repeated messages [▶ 34].

VAR_OUTPUT

```
Show     : BOOL;
```

Show: Indicator of whether a message was output (*TRUE*) or discarded (*FALSE*).

Show1



Output of a message with a value to be formatted.

VAR_INPUT

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

dLogType: Log type of the message to be displayed.

sLogCode: Optional, context-related additional information [▶ 34].

sLogText: Content of the message.

tArg1: Value to be formatted (see T_Arg).

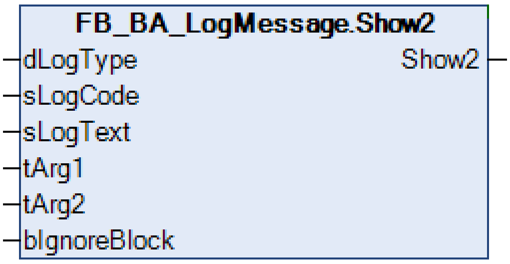
bIgnoreBlock: Prevents the suppression of cyclically repeated messages [▶ 34].

VAR_OUTPUT

```
Show1      : BOOL;
```

Show1: Indicator of whether a message was output (*TRUE*) or discarded (*FALSE*).

Show2



Output of a message with two values to be formatted.

VAR_INPUT

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

dLogType: Log type of the message to be displayed.

sLogCode: Optional, context-related additional information [▶ 34].

sLogText: Content of the message.

tArg1: First value to be formatted (see T_Arg).

tArg2: Second value to be formatted (see T_Arg).

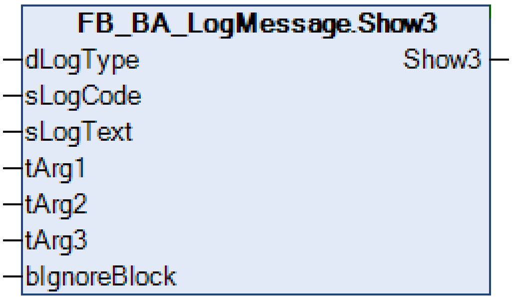
bIgnoreBlock: Prevents the suppression of cyclically repeated messages [▶ 34].

VAR_OUTPUT

```
Show2      : BOOL;
```

Show2: Indicator of whether a message was output (*TRUE*) or discarded (*FALSE*).

Show3



Output of a message with two values to be formatted.

VAR_INPUT

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

dLogType: Log type of the message to be displayed.

sLogCode: Optional, context-related additional information [▶ 34].

sLogText: Content of the message.

tArg1: First value to be formatted (see T_Arg).

tArg2: Second value to be formatted (see T_Arg).

tArg3: Second value to be formatted (see T_Arg).

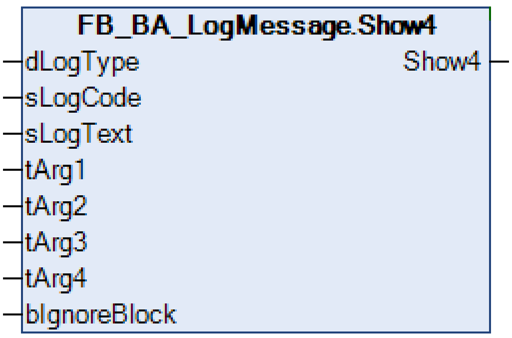
bIgnoreBlock: Prevents the suppression of cyclically repeated messages [▶ 34].

VAR_OUTPUT

```
Show3     : BOOL;
```

Show3: Indicator of whether a message was output (*TRUE*) or discarded (*FALSE*).

Show4



Output of a message with four values to be formatted.

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: Log type of the message to be displayed.

sLogCode: Optional, context-related additional information [▶ 34].

sLogText: Content of the message.

tArg1: First value to be formatted (see T_Arg).

tArg2: Second value to be formatted (see T_Arg).

tArg3: Second value to be formatted (see T_Arg).

tArg4: Second value to be formatted (see T_Arg).

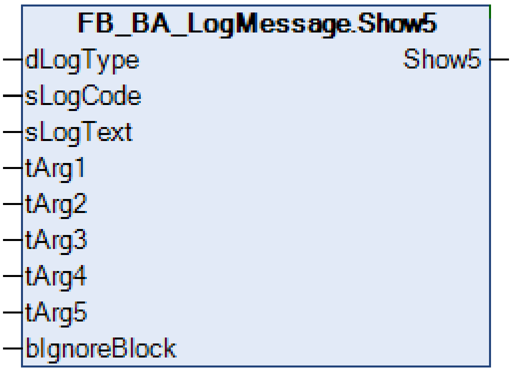
bIgnoreBlock: Prevents the suppression of cyclically repeated messages [▶ 34].

VAR_OUTPUT

```
Show4      : BOOL;
```

Show4: Indicator of whether a message was output (*TRUE*) or discarded (*FALSE*).

Show5



Output of a message with five values to be formatted.

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: Log type of the message to be displayed.

sLogCode: Optional, context-related additional information [▶ 34].

sLogText: Content of the message.

tArg1: First value to be formatted (see T_Arg).

tArg2: Second value to be formatted (see T_Arg).

tArg3: Second value to be formatted (see T_Arg).

tArg4: Second value to be formatted (see T_Arg).

tArg5: Second value to be formatted (see T_Arg).

bIgnoreBlock: Prevents the suppression of cyclically repeated messages [▶ 34].

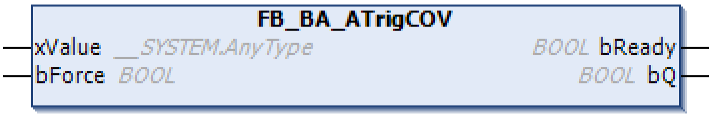
VAR_OUTPUT

```
Show5       : BOOL;
```

Show5: Indicator of whether a message was output (*TRUE*) or discarded (*FALSE*).

4.1.2.4 Trigger

4.1.2.4.1 FB_BA_ATrigCOV



The function block monitors the value *xValue* for changes (Change of Value).

> The monitored value is independent of the data type (ANY).
> For reasons of performance, however, only data types smaller than or equal to 4 bytes are supported!

VAR_INPUT

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

xValue: Value to be monitored.

bForce: Forces a positive comparison ("bQ=TRUE").

VAR_OUTPUT

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

bReady: Indicates operability:

If *xValue* is valid.
Correct value assignment and observance of the permissible data type size.

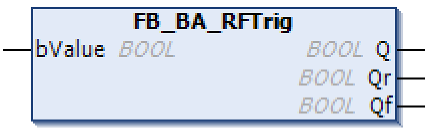
Memory is initialized.
The comparison can be made at the earliest after one cycle, as the internal memory first has to be initialized with the value *xValue*.

bQ: Result of the last comparison (*TRUE* if the value has changed).

Requirements

Development environment	Required library	Necessary function
TwinCAT3.1 4022.16	Tc3Building Automation Common from V1.0.4.3	TF8040 | TwinCAT Building Automation from V1.0.5.0

4.1.2.4.2 FB_BA_RFTrig



Function block for the detection of a rising or falling edge on a Boolean variable. The use of the separate function blocks R_TRIG and F_TRIG can be avoided with the block.

VAR_INPUT

```
bValue     : BOOL;
```

bValue: Value to be monitored.

VAR_OUTPUT

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

Q: *TRUE* if an edge is detected.

Qr: Result of the last comparison (*TRUE* as soon as the monitored value changes from *FALSE* to *TRUE*).

Qf: Result of the last comparison (*TRUE* as soon as the monitored value changes from *TRUE* to *FALSE*).

Requirements

Development environment	Required library	Necessary function
TwinCAT3.1 4022.16	Tc3Building Automation Common from V1.0.4.3	TF8040 | TwinCAT Building Automation from V1.0.5.0

4.2 DUTs

4.2.1 Structures

4.2.1.1 ST_BA_SeqLink / ST_BA_SeqLinkData

Structure of the data and command exchange between the control function block FB_BA_SeqLink [▸ 20] and the sequence controllers FB_BA_SeqCtrl [▸ 16].

This structure has to be created once per sequence control:

```
stSeqLink : ST_BA_SeqLink;
```

Within this structure, a further field structure is declared automatically, through which the sequence link function block and the individual sequence controllers exchange all relevant data. Each sequence controller writes its data into the field element corresponding to its ordinal number (entry at input *diMyNum* at the sequence controller function block). It is always the complete structure with all field elements that is linked to the function blocks.

The structures have the following setup:

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

arrSeqLinkData: Parameters of the individual sequence controllers. See below for a description of the structure *ST_BA_SeqLinkData.*

diCurCtrl: from FB_BA_SeqLink: Specification of current sequence controllers.

bSeqActv: The sequence control is enabled and active.

```
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: from FB_BA_SeqCtrl: Transfer of current control value.

lrYMin: from FB_BA_SeqCtrl: Transfer of minimum control value.

lrYMax: from FB_BA_SeqCtrl: Transfer of maximum control value.

lrW: from FB_BA_SeqCtrl: Transfer of current set value.

bActn: from FB_BA_SeqCtrl: Transfer of inverse direction of action (*bActn* = FALSE: heating mode - *bActn* = TRUE: cooling mode).

bOp: from FB_BA_SeqCtrl: Sequence controller is enabled, i.e. its input *bEn* is set to TRUE.

bPresence: from FB_BA_SeqCtrl: Checkbit, see below

bErrDouble: from FB_BA_SeqCtrl: Error during number verification: Two or more sequence controllers exist with the same ordinal number *diMyNum*.

diCurCtrl: from FB_BA_SeqLink: Specification of current sequence controllers.

Note regarding check bit:

Every sequence controller sets the *bPresence* flag in the structure that is valid for itself. If it is already set, however, then it is mandatory for *diMyNum* to be assigned twice and two sequence controllers access the same structure. After the evaluation, the sequence link function block resets all check bits, so that this test takes place cyclically. This means that an error can automatically be rectified via an online change, and new sequence controllers can be added, if required.

Requirements

Development environment	Required library	Necessary function
TwinCAT3.1 4022.16	Tc3Building Automation Common from V1.0.4.3	TF8040 | TwinCAT Building Automation from V1.0.5.0

4.2.2 Enums

4.2.2.1 E_BA_Terminal_KL

Enumerator for selecting the respective Bus Terminal.

```
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: Temperature sensors with wire breakage and short-circuit detection.

KL320x_0000: Temperature sensors with wire breakage and short-circuit detection.

KL300x: -10 V to 10 V.

KL301x: 0 mA to 20 mA with wire breakage and short-circuit detection.

KL302x: 4 mA to 20 mA with wire breakage and short-circuit detection.

KL304x: 0 mA to 20 mA with wire breakage and short-circuit detection.

KL305x: 4 mA to 20 mA with wire breakage and short-circuit detection.

KL306x: 0 V to 10 V.

KL3132_0000: -10 V to +10 V.

KL3142_0000: 0 mA to 20 mA with wire breakage and short-circuit detection.

KL3152_0000: 4 mA to 20 mA with wire breakage and short-circuit detection.

KL3162_0000: 0 V to +10 V.

KL3172_0000: 0 V to +2 V.

KL3172_0500: 0 V to +0.5 V.

KL3172_1000: 0 V to +1.0 V.

KL3182_0000: -2.0 V to +2.0 V.

KL3404: -10 V to +10 V.

KL3464: 0 V to +10 V.

KL3408: -10 V to +10 V.

KL3468: 0 V to +10 V.

KL3444: 0 mA to 20 mA with wire breakage and short-circuit detection.

KL3454: 4 mA to 20 mA with wire breakage and short-circuit detection.

KL3448: 0 mA to 20 mA with wire breakage and short-circuit detection.

KL3458: 4 mA to 20 mA with wire breakage and short-circuit detection.

Requirements

Development environment	Required library	Necessary function
TwinCAT3.1 4022.16	Tc3Building Automation Common from V1.0.4.3	TF8040 | TwinCAT Building Automation from V1.0.5.0

4.2.2.2 E_BA_Sensor

Enumerator for selecting a sensor type for measuring analog values.

```
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
```

Requirements

Development environment	Required library	Necessary function
TwinCAT3.1 4022.16	Tc3Building Automation Common from V1.0.4.3	TF8040 | TwinCAT Building Automation from V1.0.5.0

4.3 GVLs

4.3.1 Parameter

Global parameters

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

usiMaxSeqCtrl: Maximum number of sequence controllers in a sequence.

5 Appendix

5.1 Support and Service

Beckhoff and their partners around the world offer comprehensive support and service, making available fast and competent assistance with all questions related to Beckhoff products and system solutions.

Beckhoff's branch offices and representatives

Please contact your Beckhoff branch office or representative for local support and service on Beckhoff products!

The addresses of Beckhoff's branch offices and representatives round the world can be found on her internet pages: https://www.beckhoff.com

You will also find further documentation for Beckhoff components there.

Beckhoff Support

Support offers you comprehensive technical assistance, helping you not only with the application of individual Beckhoff products, but also with other, wide-ranging services:

- support
- design, programming and commissioning of complex automation systems
- and extensive training program for Beckhoff system components

Hotline:	+49 5246 963 157
Fax:	+49 5246 963 9157
e-mail:	support@beckhoff.com

Beckhoff Service

The Beckhoff Service Center supports you in all matters of after-sales service:

- on-site service
- repair service
- spare parts service
- hotline service

Hotline:	+49 5246 963 460
Fax:	+49 5246 963 479
e-mail:	service@beckhoff.com

Beckhoff Headquarters

Beckhoff Automation GmbH & Co. KG

Huelshorstweg 20
33415 Verl
Germany

Phone:	+49 5246 963 0
Fax:	+49 5246 963 198
e-mail:	info@beckhoff.com
web:	https://www.beckhoff.com

More Information:
www.beckhoff.com/te1000

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com