Table of contents

1 Foreword .......................................................................................................................... 5
  1.1 Notes on the documentation .................................................................................. 5
  1.2 Safety instructions ............................................................................................... 6
  1.3 Notes on information security .............................................................................. 7

2 Introduction ..................................................................................................................... 8
  2.1 Target groups ....................................................................................................... 8
  2.2 Requirement profile ........................................................................................... 8
  2.3 Hardware requirements ....................................................................................... 8

3 Integration in TwinCAT ................................................................................................. 9
  3.1 System requirements .......................................................................................... 9
  3.2 Installation .......................................................................................................... 9
  3.3 Licensing ............................................................................................................. 12

4 Concepts ....................................................................................................................... 15
  4.1 HMI ...................................................................................................................... 15
    4.1.1 Feedback ....................................................................................................... 15
    4.1.2 Generic ......................................................................................................... 15
    4.1.3 Trending ......................................................................................................... 16
  4.2 PLC ........................................................................................................................ 21
    4.2.1 Objects .......................................................................................................... 21
    4.2.2 Mechanisms .................................................................................................. 26

5 Tutorials ......................................................................................................................... 49
  5.1 PLC ....................................................................................................................... 49
    5.1.1 Starting a project ......................................................................................... 49
  5.2 HMI ....................................................................................................................... 55
    5.2.1 Starting a project ......................................................................................... 55
    5.2.2 Generic HMI .................................................................................................. 61

6 Examples ......................................................................................................................... 71
  6.1 Concept examples .................................................................................................. 72
    6.1.1 PLC .............................................................................................................. 72
    6.1.2 HMI .............................................................................................................. 78
  6.2 Template samples ................................................................................................... 84
    6.2.1 PLC .............................................................................................................. 84
    6.2.2 HMI .............................................................................................................. 85

7 Programming .................................................................................................................. 87
  7.1 PLC ....................................................................................................................... 87
    7.1.1 Libraries ....................................................................................................... 87
    7.1.2 SPS Project Templates ................................................................................. 420
  7.2 HMI ....................................................................................................................... 423
    7.2.1 TcHmiBa ........................................................................................................ 423

8 Tools .............................................................................................................................. 519
  8.1 Symbol Explorer .................................................................................................... 519
    8.1.1 Introduction ................................................................................................... 519

TcHmiBa
SPS Project Templates
  .......................................................................................................................... 420
Libraries
  .......................................................................................................................... 87
SPS Project Templates
  .......................................................................................................................... 420
Trending
  ..............................................................................................................................
Feedback
  .......................................................................................................................... 15
Objects
  ..............................................................................................................................
Mechanisms
  .......................................................................................................................... 26
Starting a project
  .......................................................................................................................... 49
Generic HMI
  .......................................................................................................................... 61
Starting a project
  .......................................................................................................................... 55
Generic HMI
  .......................................................................................................................... 61
Introduction
  ..............................................................................................................................
Symbol Explorer
  .......................................................................................................................... 519
PLC
  ..............................................................................................................................
HMI
  ..............................................................................................................................
Trend
  ..............................................................................................................................
Generic
  ..............................................................................................................................
# Table of contents

8.1.2 Definitions ........................................................................................................................................ 519
8.1.3 Description of the interface ........................................................................................................... 519
8.1.4 Find and replace ............................................................................................................................ 524
8.1.5 Principle of operation ..................................................................................................................... 529
8.1.6 Appendix ........................................................................................................................................ 532

8.2 Template Repository .......................................................................................................................... 532

9 Appendix ............................................................................................................................................... 539

9.1 Third-party components .................................................................................................................. 539
9.2 Support and Service .......................................................................................................................... 539
1 Foreword

1.1 Notes on the documentation

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

It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components.

It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

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

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.

We reserve the right to revise and change the documentation at any time and without prior announcement.

No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.

Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:


with corresponding applications or registrations in various other countries.

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

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.

The reproduction, distribution and utilization of this document as well as the communication of its contents to others without express authorization are prohibited.

Offenders will be held liable for the payment of damages. All rights reserved in the event of the grant of a patent, utility model or design.
1.2 Safety instructions

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Description of symbols

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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="DANGER" /></td>
<td>Serious risk of injury! Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.</td>
</tr>
<tr>
<td><img src="image" alt="WARNING" /></td>
<td>Risk of injury! Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.</td>
</tr>
<tr>
<td><img src="image" alt="CAUTION" /></td>
<td>Personal injuries! Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.</td>
</tr>
<tr>
<td><img src="image" alt="NOTE" /></td>
<td>Damage to the environment or devices Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.</td>
</tr>
</tbody>
</table>

**Tip or pointer**

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

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

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

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

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

By using the function TwinCAT3 Building Automation, all PLC programs, including the central heating plant, the air conditioning plant and the room automation functions can be programmed with TwinCAT PLC Control and are then available as function blocks within the building automation libraries.

The PID controllers, the sequence controllers and the sequence linkers required for the TwinCAT 3 Building Automation library (Tc3_BA) can be found in the pre-installed library Tc3_BA_Common.

The functions and controllers required for the TwinCAT 3 Building Automation library (Tc3_BA2) can be found in the pre-installed library Tc3_BA2_Common.

2.1 Target groups

This software is intended for building automation system partners of Beckhoff Automation GmbH & Co. KG. The system partners operate in the field of building automation and are concerned with the installation, commissioning, expansion, maintenance and service of measurement, control and regulating systems for the technical equipment of buildings.

2.2 Requirement profile

The user requires basic knowledge of the following.

- TwinCAT 3
- PC and network knowledge
- Structure and properties of the Beckhoff Embedded PC and its Bus Terminal system
- Knowledge of heating, ventilation, air conditioning and sanitary systems as well as room automation
- Relevant safety regulations for building technical equipment

2.3 Hardware requirements

The software is usable on all PC-based hardware platforms. The ideal target platforms for heating, ventilation, air conditioning and sanitary applications are the Embedded PCs from the CX series.
3  Integration in TwinCAT

3.1  System requirements

<table>
<thead>
<tr>
<th>Technical data</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>Windows 7/10, Windows Embedded Standard 7, Windows CE7</td>
</tr>
<tr>
<td>Target platform</td>
<td>PC architecture (x86, x64 and ARM)</td>
</tr>
<tr>
<td>TwinCAT version</td>
<td>TwinCAT 3.1 build 4022.17 or higher</td>
</tr>
<tr>
<td>Required TwinCAT setup level</td>
<td>TwinCAT 3 XAE, XAR</td>
</tr>
<tr>
<td>Required TwinCAT license</td>
<td>TF8040 TC3 Building Automation</td>
</tr>
</tbody>
</table>

3.2  Installation

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

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

1. Run the setup file as administrator. To do this, select the command Run as administrator in the context menu of the file.
   ▶ The installation dialog opens.

2. Accept the end user licensing agreement and click Next.
3. Enter your user data.

![Customer Information](image1.png)

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

![Setup Type](image2.png)
5. Select **Next**, then **Install** to start the installation.

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

ços The TwinCAT 3 Function has been successfully installed and can be licensed (see Licensing [12]).

### 3.3 Licensing

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

**Licensing the full version of a TwinCAT 3 Function**

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

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

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

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

![Solution Explorer](image)

- The TwinCAT 3 license manager opens.

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

![Order Information (Runtime)](image)

6. Open the Order Information (Runtime) tab.
   - In the tabular overview of licenses, the previously selected license is displayed with the status "missing".
7. Click **7-Day Trial License**... to activate the 7-day trial license.

![License Management Interface]

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

8. Enter the code exactly as it is displayed and confirm the entry.
9. Confirm the subsequent dialog, which indicates the successful activation.

- In the tabular overview of licenses, the license status now indicates the expiry date of the license.
10. Restart the TwinCAT system.
- The 7-day trial version is enabled.
4 Concepts

4.1 HMI

4.1.1 Feedback

Some controls use feedback. These controls or classes each have an attribute that activates the consideration of the feedback (e.g. UseStateFeedback of the checkbox).

The feedback is intended to notify the user if e.g. a BV is written but the confirmation (feedback) of the PLC is still pending.

Accordingly, the controls always display only the feedback value (e.g. StateFeedback of the checkbox), which ensures that the current value is always displayed in the HMI.

Changing the value of State has no effect on the display in the HMI, since only the value of State-Feedback is displayed.

Procedure

The procedure using the example of the checkbox [452] is as follows:

1. Write value by marking the checkbox
2. Value is written to the State symbol
3. Display of the loading animation
4. Response of the PLC:
   Hiding the loading animation

There are only two possible outcomes.

a. Value of symbol linked to StateFeedback changes to the same value written to State.
b. Value of StateFeedback does not change.

4.1.2 Generic

The creation of an HMI for a building automation system can be very complex and only parts of an HMI can be reused in different projects. Often the budget available for an HMI is also limited. For this reason, it is worthwhile using an HMI that already generates certain parts generically.

This generic is possible with the TF8040 Building Automation solution. This means the Tc3_BA2 [87] library for the PLC and the TcHmiBaServerExtension [508] for the HMI.

The goal of TF8040 is for the integrator to develop his system at only one central point - the PLC.

Due to the way in which systems are implemented with TF8040 and the resulting structures, it is possible to derive generic functions for the HMI.
For more information, see the documentation for Generic HMI [61].

**Project structure**

The generic functions within TcHmiBa are based on the project structure, which is established in the PLC by a child/parent relationship. With this structure it is possible to derive a generic navigation [61] through all objects on a controller.

Further information on the project structure can be found in the documentation.

**Events**

The events of all linked controllers can be collected and displayed centrally in an event list [461]. This function also results from the engineering in the PLC and does not require any further configuration in the HMI.

**Trending**

In addition, the generic design makes it possible to implement various trend functions without having to configure them separately for the HMI.

For more information, see the documentation on Trending [16].

### 4.1.3 Trending

Trends can be used and managed in the HMI. The functions described here can only be used with the generic functions [61] of TcHmiBa.

**Trend collections**

Displaying individual trend curves is often not sufficient, as they are difficult to compare in this way. For this reason there is an option to create trend collections in order to compare the trend curves in a chart.

**Configurator**

The configurator can be opened via the menu in the Parameter window [466] of a view and then only displays objects from this level.
In addition, it can also be opened via the menu of ProjectNavigationTextual [464].

Creating trend collections

Trend collections are created with the configurator. After assigning a name, the trendable objects can be added to the trend collection via the checkboxes.

The menu offers the option to select the objects to be displayed.
A Trendable object must be assigned to a trend object for use in the trend. The status for this can be viewed and also changed in the menu of the Parameter window [466].

If an assignment has not yet been made at the time of selection, this is done automatically by confirming the query.

Subsequently, the trend can be activated directly.

When the trend collection is complete, it will be available in the trend collection view after confirming the dialog.
Observing trend collections

The created trend collections can be viewed and added in the trend collection view.

When the dialog is opened for the first time, an overview appears. A trend collection can be selected from it and then displayed in a Trend Control [476].

The organization is done in tabs and allows you to quickly switch between the different trend collections.

Additional trend collections can be added via the menu.
Further actions are available in the menu for managing the trend collection.

- **Remove collection from view**: removes the trend collection in the active tab from the view (it will not be deleted).
- **Edit collection**: opens the configurator to edit the trend collection in the active tab.
- **New trend collection**: creates a new trend collection. At this point, all objects of the project structure are displayed. The first call may therefore take some time.
- **Delete collection**: opens the trend collection view. The selected trend collection is permanently deleted.

### Live Trends

If the behavior of a value is to be observed only briefly and not assigned to a trend object, it is possible to create a live trend via the menu of the object in the **Parameter window**.

This action is only available if the object itself is trendable (e.g. FB_BA_MV) or is a view with trendable objects.

After pressing the button, the live trend opens, which records either only one trendable object or all trendable objects of the view in a FIFO memory. The functions in the menu are similar to those of the **Trend Control**.

### NOTE

**Loss of data**

When the window is closed, the recorded data is lost.
The menu contains two additional actions:

- **Reset**: deletes all recorded values.
- **Pause**: pauses the recording of new values.

### 4.2 PLC

#### 4.2.1 Objects

#### 4.2.1.1 Events

Events can be generated by event-enabled objects for the following purposes:

- Making the operator aware of relevant events or states.
- Triggering appropriate (e.g. safe) system states.

**Event classes**

An event class describes a very specific configuration (or variant) of an event (e.g. a *simple* alarm event), where properties are defined by means of corresponding parameters.

By providing a fixed set of event classes for each controller, it is possible to define exactly which different configurations are available at all.

The desired event behavior of an event-enabled object is specified by assigning it to an appropriate event class.

**Behavior**

**Event behavior**

The behavior of an event-enabled object is determined by the following parameters:
Event type

The respective type of an event is only used for a more detailed description or representation (e.g. HMI) for the viewer.

<table>
<thead>
<tr>
<th>Type</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alarm</td>
<td>📣</td>
</tr>
<tr>
<td>Fault</td>
<td>💣</td>
</tr>
<tr>
<td>Maintenance</td>
<td>🧣</td>
</tr>
<tr>
<td>Notification</td>
<td>📦</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>🎉</td>
</tr>
</tbody>
</table>

Alarm mode

The alarm mode determines whether an event (or an object) must be acknowledged and / or reset.

A distinction is made between the following modes:

- Simple
  An event is automatically acknowledged after the transition to the normal state.

- Default
  An event continues to be indicated for information purposes after the transition to the normal state. An acknowledgement confirms the event and hides it.

- Extended
  An event is not only still displayed after the transition to the normal state, the fault state is also maintained.
  A simple acknowledgement confirms the event. Only resetting (acknowledging again) puts the event (or the object) back into the normal state and hides it.

Depending on the acknowledgement mode, a simple acknowledgement may be sufficient to trigger the reset.

The default behavior of individual alarm modes can be adjusted by means of global parameters. This may be necessary if, for example, operators require compliance with certain specifications. All settings are carefully preset. Changes have a significant effect on the event behavior of an object and should be implemented carefully.

Transition state

An event-enabled object always assumes one of the following transition states (Transition):

- Normal
  Error-free normal state.

- Abnormal
  A logical error is present (for example, an active event).

- Error
  A physical error is present (for example, defect in the connected hardware).

Representation state

The representation of an event depends on the following circumstances in addition to the event type of an event-enabled object:
• Event type
• Alarm mode
• Acknowledge and reset state

This results in the following possibilities for displaying an event (illustration using the example of an alarm event):

• **Hidden**
  
  The event is not pending.

• **Indicated ***
  
  The event is not (no longer) pending, but is indicated for information purposes until it is acknowledged.

• **Past and acknowledged **
  
  The event is not pending (anymore). However, it has already been acknowledged but not yet reset.

• **Past **
  
  The event is not (no longer) pending. However, it was neither acknowledged nor reset.

• **Pending and acknowledged**
  
  The event is pending and has already been acknowledged.

• **Pending**
  
  The event is pending.

* Only possible with standard alarm mode.
** Only possible with extended alarm mode.

The following representations result for each event type:

<table>
<thead>
<tr>
<th>State</th>
<th>Alarm</th>
<th>Fault</th>
<th>Maintenance</th>
<th>Notification</th>
<th>Miscellaneous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hidden</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Indicated</td>
<td>📣</td>
<td>⚡</td>
<td>⚒</td>
<td>📎</td>
<td>📵</td>
</tr>
<tr>
<td>Past, Acknowledged</td>
<td>📣</td>
<td>⚡</td>
<td>⚒</td>
<td>📎</td>
<td>📵</td>
</tr>
<tr>
<td>Past</td>
<td>📣</td>
<td>⚡</td>
<td>⚒</td>
<td>📎</td>
<td>📵</td>
</tr>
<tr>
<td>Pending, Acknowledged</td>
<td>📣</td>
<td>⚡</td>
<td>⚒</td>
<td>📎</td>
<td>📵</td>
</tr>
<tr>
<td>Pending</td>
<td>📣</td>
<td>⚡</td>
<td>⚒</td>
<td>📎</td>
<td>📵</td>
</tr>
</tbody>
</table>

An event is called **Active** as soon as it is no longer in the **Normal state** (Hidden).
Acknowledging and resetting

The user can interact with active events. He has the following options (depending on the configured alarm mode):

- **Acknowledge**
  Signals (e.g. to maintenance staff) a perceived event. It should be possible to deduce from the understanding that a corresponding action is now required.
  
  **The acknowledgement is therefore for information purposes.**

- **Reset**
  In the extended alarm mode, an event (or an object) must not only be acknowledged, but also reset in order to restore an already past event to the normal state.

  **Resetting therefore prevents the occurrence of undefined states (e.g. uncontrolled restarting of systems) and thus provides additional safety.**

Lock priorities

Define the priority for shutdown events which, for example, have a desired effect on the plant control [p. 26] or the PlantLock [p. 32].

**Local medium**

Enables a local shutdown [p. 33] of medium* priority.

**Local high**

Enables a local shutdown [p. 33] of higher** priority.

**Medium**

Enables a higher-level shutdown [p. 33] of medium* priority.

**High**

Enables a higher-level shutdown [p. 33] of higher** priority.

* Used for system-safe program sections.
** Used for personnel-safe program sections.

Background services

**Notification**

Each event-enabled object reports changes to its parent regarding its event. This results in an overview of the subordinate events of each parent or view object. Furthermore, not only the immediate parent of an object is notified, but all of its higher-level views! As a consequence, the top view object has knowledge of all events.
Programming

Configuration

Simple

```plaintext
FB_BA_EC := {
    nInstanceID := 12,
    eEventType := E_BA_EventType.eDisturb,
    cAlarmMode := E_BA_AlarmMode.cSimple,
    aPriority := [10, 11, 12]
};
```

Disturb

```plaintext
FB_BA_BI_IO := {
    nEventClassID := 51
};
```

Declaration of a simple fault by referencing an appropriately configured event class.

Evaluation

Direct

An active event is indicated by the output bEvent of an event-enabled object.

Example: Switching off by an event by means of conventional interconnection.

Indirect (PlantLock)

Active, shutdown [33] events are output at PlantLock [32].
Example: Shutdown by one or more events using PlantLock.

### 4.2.2 Mechanisms

#### 4.2.2.1 Plant control

The purpose of a plant control is to command individual aggregates depending on certain plant operation modes.

**Use case**

The plant operation mode is determined in the plant template and passed on to the PlantControl:

The PlantControl switches all commanded aggregates into the respective defined states or operation modes:
Commanding an aggregate on priority **Program control**

Which aggregate is commanded with which operation mode is precisely defined in a control matrix.

**Example: Control matrix of 5 plant operation modes and 5 aggregates**

**Plant operation modes**

- Are defined from left to right.
- **Initial state**
  The first plant operation mode, which is commanded with the priority program control, is automatically recognized as initial state (system fault-free and switched off).
  
  This state should be the **idle state** of an aggregate.

- **Switched-on state**
  A plant operating mode is evaluated as **switched-on** if at least one assigned aggregate has an operation mode that is greater in the respective plant operation mode than in the initial state of the plant.
To create a **defined state**, the system is switched to the initial state once (when the PLC is started).

During this initialization phase:

- All aggregates are commanded accordingly.
- No conditions and delays are considered.

**Aggregates**

- Are defined from top to bottom.
- **Aggregate operation modes**
  are each assigned to a plant operation mode.
  
  - If an aggregate does not change its operation mode when switching the plant operation mode, no conditions and delays are taken into account.
  
  **Example:**
  When switching the plant operation mode from *On (Slow)* to *On (Fast)*:
  The aggregate *Conveyor 1* is switched from *On* to *On*.

  - If a plant operation mode is **skipped** when switching forward, delays from all skipped aggregate operation modes are also executed.
  
  **Example:**
  Change plant operation mode from *Off* to *On (Fast)*.

- **Conditions**

  - **Wait for processes**
    As long as an aggregate reports one or more active processes, it waits before switching to the next aggregate.
    The switching is continued only after completion of all processes.

  - **Wait for events**
    As long as an aggregate reports active events, the system waits before switching to the next aggregate.

  Since active events of the lock priorities Local Medium and Local High only switch off locally, i.e. only affect aggregates and not plants, the plant control would wait with the switching forward until the affected events expire.

- **Delays**

---

**Fig. 1:**

Example: Effect of local lock priorities only on aggregates.

*Conditions only affect the plant control* when switching forward.

As soon as the control of an aggregate is completed, *conditions* have no influence (anymore) on the plant control.
Delay step up
Switching to the next aggregate takes place after an adjustable time has elapsed.

Delay step down
Switching back to the previous aggregate takes place after an adjustable time has elapsed.

Control
The control or the command takes place in two directions.

Stepping up
The plant operation mode is switched upwards (from left to right) in case of stepping up or switching forward.
All aggregates are set to the respective operation mode one after the other (from top to bottom):

Stepping down
The plant operation mode is switched upwards (from right to left) in case of stepping down or switching back.
All aggregates are set to the respective operation mode one after the other (from bottom to top):

1. Stepping up the plant operation mode
2. to 5. Commanding individual aggregates by level.
• 1. Stepping down the plant operation mode
• 2. to 5. Commanding individual aggregates by level.

Processes

A process represents a specific workflow within a plant or aggregate.

Example: opening a door. The process Busy is active as long as the door is controlled but its limit switch has not yet responded.
Range

A process can be evaluated **plant-wide**.

This means that:

- The plant control can react accordingly to the processes (see conditions).
- An aggregate A can query the process of aggregate B as long as they are within the same plant.

This enables independent development.

In this way, different aggregates can exchange information with each other via a process without it being known during the development phase whether aggregate A exists in the project planning phase (see example Aggregates and processes).

Examples

- **Plant control and processes**
  The example illustrates the described use case and refers to the commanded aggregates from the control matrix.
  The **plant control** commands the door aggregate:

  When the **door** is switched on, it signals that it is busy by **Process Busy (2)**:

  The plant control waits before switching forward.

  As soon as the end position (1) is reached, the process is completed and switching forward is enabled.

- **Aggregates and processes**
  The example illustrates the two aggregates door and conveyor belt 5 in a plant that is not controlled by a plant control.
  The aggregates are switched on simultaneously:

  The **door** signals that it is busy by **Process Busy (2)**.

  **Conveyor 5** evaluates the same **Process Busy (3)** and goes into a system safe state as long as it remains active.

  As soon as the end position (1) is reached, the process is completed and **conveyor belt 5** is set to program control priority.
This example illustrates the **independent development**:

In the **development phase** it is not known whether the *door* template will be implemented together with the *motor* template.

Nevertheless, both constellations will work:

- If there is no aggregate controlling a *Process Busy*, the *motor* (as desired) will never wait for this *process*.
- However, if one (or more) aggregates exist (such as the *door*) that control the *Process Busy*, the motor will (as desired) automatically wait for this *process*.

**Further Information:**

The requirement to cover as many use cases as possible (which can arise during the project planning phase) with a plant template typically results in a large number of template variants:

Different template variants provide different aggregate constellations.

**Constellation1111**: contains all intended system components.
**Constellation1011**: contains no motor, but door and conveyor belts 1 and 2.
**Constellation1010**: contains no motor, but door and conveyor belt 1.

This problem can be avoided through processes:

Since all aggregates are aware of the relevant processes, all dependencies (e.g. door blocks motor during opening) are clear. This also eliminates the need to program the dependencies.

The example contains all the intended system components. The integrator removes unneeded aggregates without having to consider any dependencies.

**PlantLock**

Provides information about active, *shutdown* events within its own environment.

The environment always refers to the referenced parent (see Notification).
Switch-off

The desired effect can be achieved by appropriate programming.

- The term switch-off arises because the effect of an event is a shutdown. Accordingly, the designation should be Forcing or Holding. This is because an event enforces a certain state via the lock priority.

There are two types of switch-off:

- Higher-level switch-off
  The higher-level Lock priorities Medium and High are usually used to switch-off systems:

- Local switch-off
  The local Lock priorities, Local Medium and Local High are usually used to switch off aggregates:

  The higher-level Lock priorities (Medium and High) should result in the aggregate switch-off.
Examples

- Higher-level switch-off
  The security (1) is configured as lock priority medium.
  In the event of a fault, the following states result:

<table>
<thead>
<tr>
<th>Output</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>TRUE</td>
<td>-</td>
<td>-</td>
<td>TRUE</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

1) Effect on the plant, because it is interconnected.

- Local switch-off
  The fault (6) is configured as a lock priority Local High.
  In the event of a fault, the following states result:

<table>
<thead>
<tr>
<th>Output</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>-</td>
<td>-</td>
<td>TRUE</td>
<td>1</td>
<td>-</td>
<td>TRUE</td>
<td>-</td>
<td>TRUE</td>
<td>2</td>
<td>-</td>
</tr>
</tbody>
</table>

1) No effect on the plant, because it is not interconnected.
2) Effect on aggregate, because it is interconnected.

Programming

Plant control

- Determining the plant operation mode
  The described use case is executed by corresponding programming.
  Determining the plant operation mode and connecting to the PlantControl:

- Configuration
  The described use case is executed by corresponding Initialization.
  Commanding aggregates according to the plant operation mode:
Programmatic creation of a control matrix.

Only Plant operation modes > 0 are detected.

- **Control**
  Only one aggregate is stepped up or down per cycle, because each aggregate must be called in order to have the opportunity to evaluate its conditions.

**Examples**

**Alternative initialization**

As an alternative to initialization in the declaration part (tabular form), *initialization methods* can be used within `FB_init()`:

```plaintext
1. Define available plant operation modes (AddOperationMode()).
   Note the order of the added operation modes.

2. Define aggregates to be commanded (AddAggregate()).
   Note the order of the added aggregates.

3. Assign aggregate operation modes to the respective plant operation modes.
   The order of the assigned aggregate operation modes has no effect.
```

**Processes**

**Configuration**

- Define referenced parent.
Same processes must apply the same process label.

Example

Activating a process

Instantiate process (e.g. in descriptive aggregate A).

The process is active as long as \( bEn \) and \( bVal \) have the value TRUE.

![FB_BA_Process block diagram](image)

Evaluating a process

Instantiate process (e.g. in processing aggregate B).

Query process by \( bActive \).

![FB_BA_Process block diagram](image)

4.2.2.2 Referencing

4.2.2.2.1 Collectors

The FB_BA_Collector collects values and states from one or more referenced objects of uniform types.

In this way, different values and states (such as heat requests or operation mode collection of plants and units) can be collected.

Programming

Use cases

Depending on the data type of the referenced objects, different states and values are output at the function block output:

- **Active**
  
  Consideration of at least one referenced object due to an active value.

- **ActiveCount**
  
  Number of active, referenced objects.

- **MinVal** (separate for Analog and Multistate)
  
  Smallest value of all active, referenced objects.

- **MaxVal** (separate for Analog and Multistate)
  
  Largest value of all active, referenced objects.

- **AvgVal**
Average value of all active, referenced objects.

Depending on the configuration of the object references, the FB operates under one of the following control modes:

- **Analog, Binary, Multistate**
  Collects values from *active* objects of the respective type.

An object is active once the following condition applies:

\[
\text{PresentValue} \neq \text{DefaultValue}
\]

- **En, EnAnalog, EnBinary, EnMultistate**
  Collects values from *released* objects of the respective type.

An object is active once the *release reference* has the value TRUE. The *value reference* is now taken into account in the calculation of the function block outputs.

### Initialization

**Option 1)** Direct referencing

```plaintext
[region 'Analog']
ColAV_Op1 : FB_BA_AV_Op;
ColAV_Op2 : FB_BA_AV_Op;
ColAV_Collect : FB_BA_Collecter := iReferences := [ColAV_Op1, ColAV_Op2 ]);
[endregion]

[region 'Binary']
ColBV_1 : FB_BA_BV;
ColBV_Op2 : FB_BA_BV_Op;
ColBV_Collect : FB_BA_Collecter := iReferences := [ColBV_1, ColBV_Op2 ]);
[endregion]

[region 'Multistate']
ColMV_1 : FB_BA_MV;
ColMV_2 : FB_BA_MV;
ColMV_Collect : FB_BA_Collecter := iReferences := [ColMV_1, ColMV_2 ]);
[endregion]
```

**Option 2)** Indirect referencing

```plaintext
ColEn_View1 : FB_BA_BV_Op
En1 : FB_BA_BV_Op := iParent := ColEn_View1);
Val1 : FB_BA_BV_Op := iParent := ColEn_View1);
ColEn_View2 : FB_BA_BV_Op
En01 : FB_BA_BV_Op := iParent := ColEn_View2);
Val01 : FB_BA_BV_Op := iParent := ColEn_View2);
ColEn_CollectA : FB_BA_Collecter :=
  iReferences := [ColEn_View1, ColEn_View2 ],
  sRefSubjectEn := 'En',
  sRefSubjectVal := 'Val';
```

The subject references of the referenced objects are mapped (depending on the desired control mode) by the corresponding variables `sRefSubjectEn` and `sRefSubjectVal`.

### Samples

**Option 1)** Direct referencing

```plaintext
[region 'Analog']
ColAV_Op1 : FB_BA_AV_Op;
ColAV_Op2 : FB_BA_AV_Op;
ColAV_Collect : FB_BA_Collecter := iReferences := [ColAV_Op1, ColAV_Op2 ]);
[endregion]
```
Option 2) Indirect referencing

ColEn_View1

En1

Val1

ColEn_View2

En01

Val01

ColEn_CollectA

: FB_BA_Collect := (
    iReferences := [ColEn_View1, ColEn_View2],
    sRefSubjectEn := 'En',
    sRefSubjectVal := 'Val'
);

Further information

• Error messages

4.2.2.2 Publisher and Subscriber

Publisher and Subscriber provide a way to easily exchange information.

• Publishers provide information.

• Subscribers subscribe to information from a referenced publisher in order to make it available in a specific program section (e.g. template).

Regardless of where Publisher and Subscriber are implemented (e.g. on different PLCs), information is exchanged in a uniform manner.
Identification

The connection between object and its application purpose is known to the integrator when he sees the object's designation.

This connection is not directly established in the PLC, which exclusively processes the program code.

In order to get access to information of a specific publisher from any place, it is necessary to make the publisher uniquely identifiable.

This is achieved by uniform designation of publishers or objects.

Example: Subscribing to a specific (uniquely designated) object - the outdoor temperature.

By adhering to uniform designations, independent development is made possible.

Thus, the ventilation system illustrated above would also be able to subscribe to values of the object Outdoor temperature if this were not a simple sensor but a weather station, which however provides an object with the same designation.
For the subscriber it is irrelevant where the outdoor temperature is located - but it must be available.

Programming

Subject information

Subject information describes an (arbitrary) publisher or object.

The publisher thus becomes a (very concrete) subject that can be uniquely identified.

The symbol name of a publisher is used to determine identifier and index.

In order to be able to carry out comparison operations more efficiently, a hash is formed via the identifier.

Mapping of different objects with partly the same subject information.

Subject information is generated from the symbol names of the objects as follows:

<table>
<thead>
<tr>
<th>Symbol name</th>
<th>Identifier</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>TWth</td>
<td>TWth</td>
<td>1</td>
</tr>
<tr>
<td>Flwt</td>
<td>Dstb1</td>
<td>1</td>
</tr>
<tr>
<td>Dstb002</td>
<td>Dstb002</td>
<td>2</td>
</tr>
</tbody>
</table>

Now the following options are available:

- **Referencing**
  Referencing specific objects using subject information.

- **Queries**
  Search plants for specific objects.

  ```plaintext
  MyPlant.QueryObjectSbj('Pu', 2)
  ```

- **Case distinctions**
  Iterate over all objects in a plant to perform subject-dependent operations.

  ```plaintext
  WHILE (MyPlant.IterateObjects(_iChild)) DO
  CASE (_iChild.SubjectInfo.sIdentifier) OF
  'TWth': // ...
  'Dstb': // ...
  ```
From this principle, use cases such as the following are possible:

- An HMI or operator workstation searches the PLC for plants and aggregates to automatically generate plant graphics.
- A generic navigation through project structures and systems in applications (e.g. the Terminal Explorer).

### Special cases

The following special characters are not allowed if they are at the end of the identifier (in this case they are ignored or truncated):

```
_ -
```

The following overview compares the symbol name with the expected subject information:

<table>
<thead>
<tr>
<th>Symbol name</th>
<th>Identifier</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISP_10</td>
<td>ISP</td>
<td>10</td>
</tr>
<tr>
<td>_Building05</td>
<td>_Building</td>
<td>5</td>
</tr>
</tbody>
</table>

### Subject references

In order to refer to a Publisher, or Object (example: reference in a Collector or Subscriber), a reference is expressed in an appropriate syntax:

#### Use cases

The use cases listed here illustrate the application of references using subscribers. The same applies to other references (e.g. in Collectors [p. 36]).

- **Simple** Referencing
  A publisher with unique subject information is referenced.

```plaintext
TF1_Sub : FB_BA_SubAI := {
  Subj := 'TF1',
};
```

- **Referencing the same subjects**
  If several publishers with the same subject identifier are subscribed to, their indices must be defined:

```plaintext
TF101 : FB_BA_SubAI := {
  kEnPublish := TRUE
};

TF102 : FB_BA_SubAI := {
  kEnPublish := TRUE
};

TF1_Sub : FB_BA_SubAI := {
  Subj := 'TF12'
};
```

- **Referencing the same, nested subjects**
  If several publishers with the same subject information are subscribed to, their subject paths must be defined:
FUNCTION_BLOCK FB_BA_Pump EXTENDS FB_BA_View
VAR_INPUT CONSTANT
  Dstb  : FB_BA_BI_IO := {
    iParent  := THIS^,
    bEnPublish := TRUE
  };
END_VAR

Pump1  : FB_BA_Pump;
Pump2  : FB_BA_Pump;

Dstb2_Sub  : FB_BA_SubBI := {
  sSubject := 'Pump2.Dstb'
};

The subject Dstb would not be clear enough in this case.

Subject paths always refer to the symbol path of a Publisher, not to its object path within the project structure.

Subject paths should be relative. Paths that are deeper than what is necessary to establish uniqueness should be avoided, since their resolution requires computing capacity.

Publisher

Every object is automatically also a publisher.

Enable publishing

In order to allow publishing (the sharing of information) for one or more subscribers, this must first be enabled.

• Enable

TF1  : FB_BA_AI := {
  bEnPublish := TRUE
};

Example: Enable for the current value of a flow temperature.

• The publishing must be enabled separately for individual publishers.

Subscriber

Setting up a subscription

The subscription (connection from a subscriber to a publisher) can be established if the subject reference of the subscriber matches the subject information of the publisher.

Requirements for a valid subscription are:

• The data type of the subscriber and the subscribed publisher are identical.
• The data of the subscriber and the subscribed publisher have the same size.
Practically, any number of subscribers can be created. The more subscriptions are created, the more computing power is used for the subscription. For more information, refer to Best Practice [46].

Local Subscriptions

Before subscribing (the subscribing of information) to a publisher is possible, the access to its information must be enabled. The subscriber then only needs the subject information of the other side or the subscribed publisher.

- Subject information

```plaintext
TF1_Sub : FB_BA_SubAI := {
  sSubject := 'TF1',
  sTarget := '192.168.10.100.1.1'
};
```

Remote Subscriptions

The procedure for setting up a subscription to publishers on remote ADS devices is the same as for the local subscription.

Only the following information must be provided in addition:

- Target system

```plaintext
TF1_Sub : FB_BA_SubAI := {
  sSubject := 'TF1',
  sTarget := '192.168.10.100.1.1'
};
```

Furthermore, it is possible to define the following (optional) parameters:

- Read interval
- Read tolerance
- Reset in case of read errors

Example: declaration with optional parameters:

```plaintext
TF1_Sub : FB_BA_SubAI := {
  sSubject := 'TF1',
  sTarget := '192.168.10.100.1.1',
  tReadInterval := 5s,
  tReadTolerance := 20s,
  bClearOnReadError := TRUE
};
```

No FB properties of object-related subscribers are supported. These are exclusively for local subscribers.

ADS routes

Creating routes for ADS devices that communicate with each other is a separate step. An integrator first implements the subscriptions and loads the control systems, although the communication (due to missing routes) does not yet work.
These routes can then be set up via the Terminal Explorer (see Device communication). After setting up the routes, all subscriptions will automatically (without any further steps) change from the fault state to the operating state.

**Read errors**

In the event of read errors (e.g. interruption of the physical connection), the affected *Subscriber* switches to the error state *Error*.

- With each further failed reading attempt, the set read interval is temporarily increased by a factor of 1 (up to a maximum of 10) to conserve the network load and reduce error messages.

**States**

The state of a *subscriber* is indicated by the variable eState:

- **Init**
  The *subscription* is initialized.

- **Ready**
  The *subscriber* is ready (Initialization was completed successfully).

- **Error**
  An error has occurred.

**States** that a subscriber can assume once a publisher is subscribed to on the local device:

- **Ready**
  Data is read cyclically by the subscribed publisher in this state.

**States** that a subscriber can assume once a publisher is subscribed to on a remote device:

- **Init Subscription**
  The subscribed publisher is queried on the remote ADS device.

- **Ready**
  The subscriber is idle between individual read operations.

**Suppressing**

A read error has occurred which is suppressed for a set time.

- No error state is displayed during *suppression*.

**Error messages**

At runtime, different error messages can be output in the log.

**Use cases**

Depending on the use case *typeless* [45] or *typed* [45] *publisher* and *subscriber* can be used.

- *Typeless* [45] and *typed* [45] *publishers / subscribers* can be applied mixed.

  For example, a derivative of FB_BA_Publisher can be used to share data, while an FB_BA_Raw-Subscriber is used to read this data.

**Object-related**

*Object-related subscribers* are useful when data is to be read from standard objects.

For all object types there are corresponding prefabricated *subscribers*.

Thus, a subscriber type is compatible with one or more object types.
Example: subscriber that can be applied to all analog objects with priority array:

```
FB_BA_SubAPri
```

**Typeless**

Typeless publishers / subscribers are useful if data can be released or subscribed without any processing or preparation (e.g. program logic for formatting raw data to be sent or received).

**Application**

- **Publisher**

**Declaration:** data to be released and raw publisher:

```
VAR
  stData : ST_BA_MyWeatherData;
  WthSt  : FB_BA_RawPublisher;
END_VAR
```

**Implementation:** assignment of data to be released to the publisher:

```
WthSt
   stData := xData;
```

- **Subscriber**

**Declaration:** data to be received and raw subscriber:

```
VAR
  sData   : ST_BA_MyWeatherData;
  WthSt_Sub: FB_BA_RawSubscriber := (sSubject := 'WthSt');
END_VAR
```

**Implementation:** assignment of data to be received to the subscriber:

```
WthSt_Sub
   sData := xData;
```

**Typed**

Typed publishers / subscribers are useful when data:

- should be processed in an object-oriented manner and encapsulated in an FB.
- are read out (e.g. with linked I/O terminals) before release.
- are prepared before the release, or after the subscription (e.g. by means of appropriate program logic).
- should be validated before release, or after subscription.

**Implementation**

The implementation of one's own publishers and subscribers is illustrated with the example of a weather station.

- **Publisher**
- Derive from FB_BA_Publisher.
- During initialization, call the inherited method `OnInit();`

```pascal
FUNCTION_BLOCK FB_BA_MyWeatherStation EXTENDS FB_BA_Publisher IMPLEMENTS I_BA_MyWeatherStation
VAR
  (region 'Internal')
  eState   : (cInit:=1, cOperation:=2, cError:=3) UDINT := 1;
  stData   : ST_BA_MyWeatherData;
  (endregion)
END_VAR

CASE (eState) OF
  eInit:
    IF (OnInit()) THEN
      eState := cOperation;
    ELSE
      eState := cError;
    END_IF

Override inherited method `GetData()` to make released data retrievable:

```pascal
METHOD GetData
VAR OUTPUT
  pData : VOID; // Pointer to published data.
  nSize : DWORD; // Size of published data.
END_VAR

pData := ADDR(stData);
nSize := SIZEOF(stData);
```

- Subscriber
  - Derive FB_BA_Subscriber.
  - Override inherited method `GetData()` to write subscribed data:

```pascal
METHOD MyWeatherStationSub.GetData
VAR OUTPUT
  pData : VOID; // Pointer to subscribed data.
  nSize : DWORD; // Size of subscribed data.
END_VAR

pData := ADDR(stData);
nSize := SIZEOF(stData);
```

**Best practice**

**Site GVL**

All devices on a site should use a GVL with *site-wide information*.

In this way, all devices can uniformly access the appropriately released information.

```pascal
VAR_GLOBAL
  (region 'Devices')
    Dev_Isp01 : T_BA_MedString := '192.168.10.101.1.1';
    Dev_Isp02 : T_BA_MedString := '192.168.10.102.1.1';
    Dev_Isp03 : T_BA_MedString := '127.168.0.3.1.1';
  (endregion)

  (region 'Subscriptions')
    WthT_Subject   : T_BA_MedString := 'WthT';
    WthT_Target    : T_BA_MedString := NETID_B_ISP02;
    WthSt_Subject  : T_BA_MedString := 'WthSt';
    WthSt_Target   : T_BA_MedString := NETID_B_ISP02;
    Spt_Subject    : T_BA_MedString := 'Spt';
```

46  Version: 1.4  TF8040
### Excerpt from the Document:

#### Example: Site GVL with known devices and subscriptions.

- **Device 1**: provides general setpoints `$Setp$`.
- **Device 2**: provides data of the weather station `$WthSt$` (additionally the outdoor temperature `$WthT$`).

The subscription is independent of the device on which it is implemented.

This is because the correctness of the subject information is ensured in the site GVL, from which it is then extracted.

#### Sample: subscriber on any device that subscribes to weather data:

```plaintext
WthSt   : FB_BA_WeatherStationSub := (  
    #Subject           := SiteGVL.WthSt_Subject,  
    #Target            := SiteGVL.WthSt_Target  
);  
```

> To ensure that all devices are using the same site GVL version, one of the following procedures is recommended:

- **Project library**
  - Outsource GVL to TwinCAT library, which is then integrated in all devices.

- **Template Repository**
  - Keep GVL up-to-date on all devices using Template Repository [p. 532].

### Subscriptions

You should create as few subscribers as possible (e.g. for reasons of clarity or performance), but as many as necessary.

It therefore makes sense to declare subscribers only at plant level. Subscribed data can then be passed (e.g. using `VAR_IN_OUT`) to lower-level or nested program sections (such as aggregates).

Example: subscribing to the outdoor temperature within the (same) "Plant" level; passing on to lower program sections:
Example

Publisher and subscriber [► 72] (Also includes the code sample weather station)

- Local [► 74]
- Remote [► 77]
5     Tutorials

5.1     PLC

5.1.1     Starting a project

This chapter describes how to start a project.

Procedure

The procedure described here includes all settings that are relevant for a functioning project.

When using the TF8040 PLC project templates [420], individual steps are already prepared as required.

5.1.1.1     Updating the runtime

If the runtime on the target device is not up-to-date, it should be updated accordingly:

XAR

Install the latest XAR on desktop Windows systems.

Image

Update the image on other systems (e.g. Windows Compact 7).

5.1.1.2     Creating a TwinCAT project

• Create a new solution.
5.1.1.3  Add a PLC project

- Add the appropriate PLC project template [420]:

5.1.1.4  Choose Target System

- To proceed with the project settings, you must first select a controller as the target system.

For more information refer to the chapter Choose Target System.

5.1.1.5  Project settings

Once you have selected the target system, you can proceed with the following project settings.
System

Real-time

• Reading the existing hardware configuration:

<table>
<thead>
<tr>
<th>Settings</th>
<th>Online</th>
<th>Priorities</th>
<th>C++ Debugger</th>
</tr>
</thead>
<tbody>
<tr>
<td>Router Memory</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configured Size [MB]:</td>
<td>200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allocated / Available:</td>
<td>200 / 199</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Global Task Config</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximal Stack Size [KB]:</td>
<td>64KB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Available Cores | Shared / Isolated: | | |
|-----------------|-------------------|-----------------|
| Core | RT-Core | Base Time | Core Limit |
| 0 | | | |
| 1 | | | |
| 2 | | | |
| 3 | Default | 1 ms | 80 % |

• Selecting a core:

If there are several cores to choose from, the last core is recommended, since the load generated by the operating system tends to be small there.

| Available Cores | Shared / Isolated: | | |
|-----------------|-------------------|-----------------|
| Core | RT-Core | Base Time | Core Limit |
| 0 | | | |
| 1 | | | |
| 2 | | | |
| 3 | Default | 1 ms | 80 % |

**NOTE**

Do not use isolated cores.

• Configuring the router memory

The memory should generally be set to 200 MB:
A restart of the operating system (on the target device) is required to apply the setting.

**Tasks**

**Solution 'MyFirstTF8040' (1 project)**  
**MyFirstTF8040**  
**SYSTEM**  
**License**  
**Real-Time**  
**I/O Idle Task**  
**Tasks**  
**PlcTask**  
**Routes**  
**Type System**  
**TeCCOM Objects**

Make the following settings for the PLC task:

- **Recommended cycle time:** 45 ms

**PLC**

The settings described below are not necessary if a PLC project template [420] is used.

**Libraries**

If you do not use a template, please add the following libraries to the PLC project.

- **Tc3_BA2_Common**
- **Tc3_BACnetRev14**
- **Tc3_BA2 [87]**
In the standard PLC-BA template all the required libraries are already loaded automatically.

I/O

The procedure proposed here refers to the steps required to make the TF8040 function operational in combination with BACnet on the desired hardware.

Further steps for setting up the hardware are not be discussed in detail here.

BACNet

- Add the BACnet device:
Select the appropriate Network Interface Card:

The adapter is set in the BACnet device under the Adapter tab:

- Link the BACnet adapter with the AMS NetID:

  This step is only possible once the project has been compiled without errors.

Select the BACnet adapter to be used:
License

TwinCAT 3 standard licenses are tied to a unique system ID of a TwinCAT 3 license dongle (or IPC). Standard licenses are subject to a fee: The license price depends on the hardware platform level.

For further details please refer to the information system (see Licensing).

- Determine the license status.
- To get started, you can activate a trial license. This unlocks all functions for 7 days.

5.1.1.6 Adapt the PLC template

Adapt the PLC project template you are using according to the recommended procedure.

5.1.1.7 Continue

The PLC project is now set up and you can start the project planning.

5.2 HMI

5.2.1 Starting a project

Description

The following section describes how to create and start a TcHmiBa project.

When using TcHmiBa project templates, individual steps may already be pre-prepared as required.

Procedure

Installing the TwinCAT 3 HMI

TwinCAT 3 HMI Engineering is required for creating TcHmiBa projects. Note the system requirements.
Further information can be found in the documentation for the TwinCAT 3 HMI (TE2000).

After the installation, the project template for a standard HMI should be available in all development environments selected during setup (TcXaeShell, Visual Studio 2019, etc.).

Installing TF8040

To use TcHmiBa projects, the TcHmi and TF8040 must be installed. Among other things, the TF8040 setup includes the NuGet packages, which will be discussed in more detail later.

Create a new TcHmi project

A new TwinCAT HMI project can be created on the start page or via File > New > Project under the category TwinCAT HMI.

During project creation there may be a Windows security alert because the engineering server is accessing the network. Please allow the access request accordingly.

The created project should now have the following structure:
Specific controls for building automation are not included in the standard scope of the TcHmi toolbox.

In order to use these in the created project, the NuGet package `Beckhoff.TwinCAT.HMI.Controls` must be installed. The procedure for doing this is as follows:

- Open the NuGet package management.
At the top right in the package management select the correct package source, TwinCAT Building Automation.

The selection is only available if TF8040 including the TcHmi option (standard) has been installed.

- Install Beckhoff.TwinCAT.HMI.BA.Controls

Beckhoff.TwinCAT.HMI.BA.Controls

Version: Latest stable 1.1.2

Options

Description

Controls for building automation HMIs.

Version: 1.1.2
Owner(s): Beckhoff Automation
Author(s): Beckhoff Automation
License: https://aka.ms/deprecateLicenseUrl
Date published: Tuesday, June 29, 2021 (6/29/2021)
Project URL: https://infosys.beckhoff.de/content/1033/te2000_tcm3_hmi_engineering/index.html
Tags: Beckhoff TwinCAT HMI TcHmi Controls Control BA BaControls TcHmiBa TcHmiBaControls

Dependencies

native, Version = v1.12, Profile = tchmi
Beckhoff.TwinCAT.HMI.BA.Framework (>= 1.1.2)
Beckhoff.TwinCAT.HMI.BA.Icons (>= 1.1.2)
Beckhoff.TwinCAT.HMI.Controls (>= 12.746.3)
Beckhoff.TwinCAT.HMI.Framework (>= 12.746.3)

The packages Beckhoff.TwinCAT.HMI.BA.Framework and Beckhoff.TwinCAT.HMI.BA.Icons are also installed during the installation.

After installing the NuGet packages, the project tree should look like this.
If the Desktop.view and the toolbox are open, the following controls should now be available:
Use

The available controls can be used to create visualizations as well as UserControls. The usage thus corresponds to the standard controls of the TcHmi.
Generic functions

If you want to use the TF8040 as a complete solution with all its advantages during engineering, please continue with the chapter Generic HMI [61].

5.2.2 Generic HMI

Description

The following section describes how the generic functionalities of TcHmiBa are used to minimize the development work for the HMI. As a prerequisite, see the chapter on Starting a project [55].

Procedure

Note the system requirements of the TcHmiBaServerExtension.

Furthermore, a PLC is required that was created with the libraries from TF8040 and is already active.

Description of required terms.

Preparing the server

Before starting to create the visualization with generic functions, the server must be prepared.

Installing the TcHmiBaServerExtension

To support the generic functionalities it is necessary to install the NuGet package Beckhoff.TwinCAT.HMI.BA.ServerExtension.

After installation the project tree should look like this:
Configuring TcHmiBaServerExtension

Open the configuration page for TcHmiBaServerExtension.

All the settings for the TcHmiBaServerExtension can be implemented in the window that opens.

For more information about configuration and functions, see the server extension documentation.

PLC1 is created by default in the runtimes entry. Enter the respective data for the active PLC under AmsNetId and Port.

No change of the settings is required if the PLC is running locally on the development computer.

Afterwards the runtime has to be activated and the dialog has to be confirmed.
In the TwinCAT HMI configuration the PLC should have been created successfully in the server extension.

The runtime is then listed in the open window under All Symbols.
If no HMI project is selected in the project tree, the display in the TwinCAT HMI configuration remains empty.

A mapping for the runtime has also been created already. The mapping is listed under Mapped Symbols.

![TwinCAT HMI Configuration](image)

Note the prerequisites for the required mappings for the generic functionalities.

The configuration of the server is now complete.

**Using generic controls**

The generic controls can only be used with the TcHmiBaServerExtension. A small selection is briefly described below.

Further information can be found in the respective documentation of the individual controls.

**Project navigation**

The quickest way to get started with the HMI is to use the ProjectNavigationTextual control. It can be found in the BA | General category of the toolbox. If the control was placed on the Desktop.view, it should look like this:
Open the Properties window.
The content of the Properties window always depends on the current selection. To see the properties of the project navigation, the control must be selected in Desktop.view.

If the attribute **BaObject** is linked to the ProjectStructure of the runtime, this control can be used to navigate through the entire project structure.
Then start the Live view.

The Live view should now look like this:
**UiIcon**

The **UiIcon** can be used for various applications. It can also be found in the **BA | General** category of the toolbox.

Like all controls from TcHmiBa, the **UiIcon** has the attribute **BaObject**. Any **BaView** or any **BaObject** can be linked to this. Again, the **ProjectStructure** of the runtime (see above) serves as an example.

In Live view, the **UiIcon** then looks like this:

It displays the active events of the linked view/object and enables opening of a window that contains the generic navigation from above.
The same functionalities are thus available from this window.
6 Examples

Examples of all features from TF8040.

Downloads

- TF8040 Samples (see https://infosys.beckhoff.com/content/1033/TF8040_TC3_BuildingAutomation/Resources/zip/11236473867.zip)

Contains

- Samples of PLC (PLC [72], PLC [84])
- Samples of HMI (HMI [78], HMI [85])

Configuration

PLC

Individual settings within the TF8040-Concept-Samples-PLC-Solution must be adapted to the hardware used.

The settings affect the Project settings [50] and the Project settings [53].

All necessary steps are described in Starting a project [49].

Individual settings that have already been made do not need to be taken into account any further.

Preparing the PLC

To run the HMI, the PLC must be configured [71] and started.

Installing the TwinCAT HMI

To load the sample project, the TwinCAT 3 HMI Engineering must first be downloaded and installed.

Further information on the required steps can be found in the documentation for the TwinCAT 3 HMI Engineering in the section Installation.

Download

Once the PLC is activated and running, the sample solution can be opened.

License development system

In order for the HMI samples to be executable, the TC3 HMI server license of the TF2000 must be licensed on the target system.

Server configuration

No further configuration is necessary if the runtime is activated on the PC on which the HMI project is also started.

If not, the configuration must be adjusted. The procedure for this is described in the tutorial Generic HMI [61].

Please note the general comments.
Further information
  • PLC programming

6.1 Concept examples

6.1.1 PLC

6.1.1.1 Demonstration

6.1.1.1.1 Objects

6.1.1.1.2 Reference

Collectors

Groups

Publisher and Subscriber
Contains examples around the concept of referencing using publish and subscribe.
All examples also include:
  • Publisher [42]
  • Object-related subscribers [44]
  • Type-less subscribers [45]
  • Typed subscribers [45]

Example

Weather station
The distribution of information (using typed publishers [42] and subscribers [42]) is illustrated using the example of a weather station.

Use case
Building 1 has a weather station and distributes information (e.g. outdoor temperature) to all rooms in several buildings.
Structure

- 4 PLCs control rooms on a total of 4 floors (2 x ground floor, 2 x upper floor).
- The weather station is on PLC 1.
- Each room is implemented by one instance of a room template. The access to data of the weather station is established by means of a Subscriber [42].

Principle of operation

PLC 1 makes weather data available (publishing) through the typed publisher "FB_BA_MyWeatherStation" (physically connected weather station).

The rooms on the different PLCs each contain the typed subscriber of type "FB_BA_MyWeatherStationSub" in order to read the weather data (corresponding to local and remote subscriptions).
Implementation

- On PLC 1 the weather station is declared at any position of the PLC program:

```plaintext
VAR
  MyWthSt : FB_BA_MyWeatherStation := (bEnPublish := TRUE);
END_VAR
```

- The subject information required to initialize the subscriber must exist on all PLCs (but usually in the site GVL):

```plaintext
VAR_CONSTANT
  WthSt_Target : T_BA_MedString := '192.168.10.101.1.1';
  WthSt_Subject : T_BA_MedString := 'MyWthSt';
END_VAR
```

- The declaration of the subscriber in the room template of the respective PLC would then look like the following:

```plaintext
WthStSub : FB_BA_MyWeatherStationSub := (
  $Subject := WthSt_Subject,
  $Target := WthSt_Target
);
```

- **PLC 1**
  - As the weather data are stored on the local PLC, the data can be read directly (within the PLC) (Local Subscriptions).

- **PLC 2**
  - As the weather data is stored on a remote PLC, the data is read from PLC 1 via ADS (Remote Subscriptions).

**Local**

This documentation describes the use of local subscribers located on the device.
Examples

Providing and reading

Providing information via publisher \[42\] and reading provided information via subscriber \[42\] (on the same device) is illustrated by implementing a pair of publisher \[42\] and subscriber \[42\] and configuring them accordingly.

Configuration

- Enable publishing \[42\]
- Setting up a subscription \[42\]

Object-related

For all available object types, a sample is provided to illustrate how object type-dependent information (e.g. current value or event status) is provided and read.
Sample: publisher and subscriber of an "Analog Value" Object

```c
AVOp_Sub : FB_BA_SubAV_Op := (sSubject:=[AVOp_Obj]);
AVOp_Obj : FB_BA_AV_Op := {
  iParent := A,
  iProfile := MyProfiles.AVOp_Prf,
  bEnPublish := TRUE,
  fValueRm := 10
};
```

Typeless

The sample illustrates how any information (e.g. raw data in the form of a structure) is provided and read.
Sample: publisher and subscriber for raw data of a weather station:
Typed

The sample illustrates how typed information is provided and read by implementing custom derived publisher and subscriber.

Sample: publisher and subscriber of the user-defined type of a weather station:

```plaintext
MyRawWthSt : FB BA_MyWeatherStation := (bEnPublish := TRUE);
MyRawWthSt Sub: FB BA_MyWeatherStationSub := (sSubject := "MyRawWthSt");
```

Tests

Subject path

The sample illustrates the relationship between subject information and references.

The code snippet below generates the following project structure.

Two view objects, each with two further subobjects:
Examples

Since two objects with the subject Value exist, this information alone is no longer sufficient to uniquely identify one of these two objects!

The uniqueness is established by additionally specifying subject information of the parent.

Remote

This documentation describes how to use remote subscribers on other devices.

Examples

The samples described here refer entirely to the samples for local subscribers [74]:

For each local sample, there is an analogous sample of a remote subscriber.

This document is intended to illustrate the difference between local [43] and remote [43] subscribers.

Communication

Setting up the routes [43] is a separate step.
NOTE

Communication

Communication between publishers [42] and subscribers [42] (on different devices) can only work if the ADS routes of both devices are set up correctly.

Application

The targets, of the subscribers [42] illustrated in these samples are initialized with predefined AMS NetIDs.

Here, the placeholder variable `sDevice1_NetID` is used to initialize one half, while the other half is initialized with the placeholder variable `sDevice2_NetID`:

```plaintext
AVOp_Sub : F5_BA_SubAV_Op := {
  sSubject := 'AVOp_Obj',
  sTarget := sDevice1_NetID,
  tReadInterval := T#5S
};

AI_Sub  : F5_BA_SUBAI := {
  sSubject := 'AI_Obj',
  sTarget := sDevice2_NetID,
  tReadInterval := T#5S
};
```

By commenting in and out and adjusting these placeholder variables accordingly, the sample program can be adapted to your own test environment:

```plaintext
VAR CONSTANT
{region 'Test Settings'}
// Use one (remote) device:
// sDevice1_NetID : T_EA_MedString := '192.168.10.100.1.1';
// sDevice2_NetID : T_EA_MedString := '192.168.10.100.1.1';

// Use one (loopback) device:
// sDevice1_NetID : T_EA_MedString := '127.0.0.1.1.1';
// sDevice2_NetID : T_EA_MedString := '127.0.0.1.1.1';

// Use two (remote and loopback) devices:
// sDevice1_NetID : T_EA_MedString := '192.168.2.200.1.1';
// sDevice2_NetID : T_EA_MedString := '127.0.0.1.1.1';

// Use two (remote) devices:
// sDevice1_NetID : T_EA_MedString := '192.168.10.100.1.1';
// sDevice2_NetID : T_EA_MedString := '192.168.2.200.1.1';
{endregion}
END_VAR
```

The following test settings are available for selection:

- A remote device
- A loopback device
- A remote and a loopback device
- Two remote devices

6.1.2 HMI

Explanation of the sample project TF8040-Concept-Samples-HMI.
For more information on the required steps, refer to the sample documentation in the HMI section.

Contents

The individual sample pages of the project are described below. It is advisable to open the live view in order to be able to follow the execution more easily.

Header

The header is provided with various functions (from left to right).

- Logo
- Responsive navigation
- User settings and further information
- Event list
- Building information
- Outdoor temperature
- Date and Time

Further information on the functions can be found in the documentation for the header.
Project navigation

The generic project navigation was defined as the start page of the visualization. The content of the Live-View should look like this after the start:

In the project navigation you can navigate through the project structure and display the parameters of individual views or objects.

For more information on project navigation, see the documentation for ProjectNavigationTextual.

The following pages are located under the entry Content\Samples.

BasicComponents

This page displays all controls that are stored in the Toolbox category BA | Base.

The controls are not linked to variables from the PLC.

Further information can be found in the documentation for Controls.


BaObjects

On this page, a corresponding control is stored for each primitive data type (*Analog*, *Binary* and *Multistate*).

The controls are connected to objects from the PLC and the values are written to the PLC and read again.

The following information is displayed per line:

- **The Description** of the object.
- **The value of the object** (writable or read-only).
- **Button to open the project navigation of the object** (only one entry is visible, because single objects are concerned).

**Event**

The simulation of the different event types is possible on this page.
The respective event can be activated via the checkboxes and the behavior can be observed in the event list below. A *UiIcon* to display the events is also positioned on the page. The view containing the sample events is linked to both the event list and the *UiIcon*. Therefore, no events outside the view are displayed on this page.

### Trend

This page shows the trend control for displaying various trend curves.

In this case the complete project structure was linked to the control. This filters the project structure for all available trends and displays them. On the right-hand side there is an option to select and deselect trends.

For more information, see the documentation on Trend [476].

### Schedule

The schedule displays the *current schedule* and the *weekly schedule*. The *Calendar* tab contains the entries from the linked calendar references and the local exceptions.

The buttons at the bottom of the page allow you to switch to other schedule types. In addition, the alignment and accuracy of the schedule can be set.
For more information, see the documentation on the Schedule [468].

### RoomAutomation

The following list shows the available controls for room automation:

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunblind [499]</td>
<td>Displays and controls the position and angle of a sunblind.</td>
</tr>
<tr>
<td>Light [495]</td>
<td>Displays and controls the brightness value of a lamp (dimmable or on/off).</td>
</tr>
<tr>
<td>AirConditioning [495]</td>
<td>Displays and controls the air conditioning of a room.</td>
</tr>
<tr>
<td>Window [504]</td>
<td>Shows and controls the position of a window (percentage or open/close).</td>
</tr>
<tr>
<td>RoomControl [485]</td>
<td>This control can combine all of the above controls.</td>
</tr>
</tbody>
</table>

The controls are for demonstration purposes only and are therefore not linked to variables from the PLC.
ColorPicker

On this page, ColorPicker is shown in its different versions.

Further information can be found in the documentation for the ColorPicker [454]

6.2 Template samples

The samples in this package are intended to show different templates in the PLC and HMI and how to use them.

6.2.1 PLC

Explanation of the procedure for starting the sample project TF8040 Template Samples PLC and its contents.
Procedure
The procedure can be found in the documentation for the concept sample.

Contents
The individual templates used in the project are described below.

Heating circuit
The template FB_BA_H_HtgCir01 is used here.

![](image)

Ventilation system
The template FB_BA_AHU_1St_PreHtr_ErcPl_Col is used here.

![](image)

6.2.2 HMI
Explanation of the sample project TF8040 Template Samples HMI.

For more information on the required steps, refer to the sample documentation in the HMI section.

Components
The individual sample pages of the project are described below.

The open live view shows many errors. This is because objects in the PLC signal an error if no physical inputs are connected.

Project navigation
The generic navigation is displayed on the content of this page.

Plants
HZK01
Displays a heating circuit.
AHU01

Displays a ventilation system.
7 Programming

7.1 PLC

7.1.1 Libraries

7.1.1.1 Tc3_BA2

For new projects please use the library Tc3_BA2

7.1.1.1.1 DUTs

Enumerations

Frame

DPAD

E_BA_ConcatDPADMode

TYPE E_BA_ConcatDPADMode :
{
  Invalid := 0,
  eNone := 1,
  eEntryPoint := 2,
  eParent := 3
)
END_TYPE

eNone: No linking takes place (instance title only).

eEntryPoint: Linking up to the instance entry point (e.g. plant).

eParent: Linking up to the parent instance.

E_BA_DPADMode

TYPE E_BA_DPADMode :
{
  Invalid := 0,
  Undefined := 1,
  eExclude := 2,
  eInclude := 3,
  eIncludeObjectName := 5,
  eIncludeDescription := 4
)
END_TYPE

eInclude: Apply object name and description.

eIncludeObjectName: Apply object name.

eIncludeDescription: Apply description.
Events

**E_BA_AcknowledgeMode**

TYPE E_BA_AcknowledgeMode:
{
  eSingle := 1,
  eEntire := 2
} INT;
END_TYPE

eSingle: Confirms only the next outstanding event transition.
eEntire: Confirms all unconfirmed event transitions with a single confirmation.

**E_BA_AlarmMode**

TYPE E_BA_AlarmMode :
{
  Invalid := 0,
  eSimple := 1,
  eStandard := 2,
  eExtended := 3
} BYTE;
END_TYPE

eSimple: The message goes out when the alarm has gone.
eStandard: The message goes out when the gone alarm has been acknowledged.
eExtended: The message goes out when the gone alarm has been acknowledged and reset.

**E_BA_EventType**

TYPE E_BA_EventType :
{
  Invalid := 0,
  eAlarm := 1,
  eDisturb := 2,
  eMaintenance := 3,
  eNotification := 5,
  eOther := 4
} BYTE;
END_TYPE

**E_BA_ObjectStateFlags**

TYPE E_BA_ObjectStateFlags :
{
  eOutOfService := 1,
  eOverridden := 2
} BYTE;
END_TYPE

Groups

**E_BA_AValCalcMode**

TYPE E_BA_AValCalcMode :
{
  eUndefined := 0,
  eMin := 1,
  eMax := 2,
  eAverage := 3
} BYTE;
END_TYPE
E_BA_BValCalcMode

TYPE E_BA_BValCalcMode :
{
eUndefined := 0,
eAnd := 1,
eOr := 2,
eXOr := 3
} BYTE;
END_TYPE

E_BA_MValCalcMode

TYPE E_BA_MValCalcMode :
{
eUndefined := 0,
eMin := 1,
eMax := 2,
} BYTE;
END_TYPE

Priority

E_BA_LockPriority

TYPE E_BA_LockPriority :
{
Invalid := 0,
eNoLock := 1,
eLocalMedium := 2,
eLocalHigh := 3,
eMedium := 4,
eHigh := 5
} BYTE;
END_TYPE

eLocalMedium: Locking units via the medium priority (lower in the project structure).
eLocalHigh: Locking units via the high priority (lower in the project structure).
eMedium: Locking units via the medium priority (higher in the project structure).
eHigh: Locking units via the high priority (higher in the project structure).

E_BA_Priority

TYPE E_BA_Priority :
{
Invalid := 0,
eDefault := 1,
eProgram := 2,
eManualRemote := 3,
eManualLocal := 4,
eCritical := 5,
eLifeSafety := 6
} BYTE;
END_TYPE

eDefault: Default if no priority is used.
eManualLocal: Manual override e.g. by means of physical switches.
References

E_BA_AssignRefMode

TYPE E_BA_AssignRefMode:
{
  eNone := 1,
  eAssignNow := 2,
  eInitByProfile := 3
} BYTE;
END_TYPE

eAssignNow: Immediately assign a reference.
eInitByProfile: Assignment of a reference during initialization if there is a release in the associated object profile.

Types

E_BA_NodeType

TYPE E_BA_NodeType:
{
  Invalid := 0,
  Automatic := 1,
  eUnknown := 10,
  eOther := 11,
  eGeneral := 12,
  eLocation := 13,
  eBuilding := 14,
  eBuildingElement := 15,
  eInformationFocus := 16,
  eControlCabinet := 17,
  eTrade := 18,
  eFloor := 19,
  eRoom := 20,
  ePlant := 21,
  eComponent := 22,
  eAggregate := 23,
  eFunction := 24
} BYTE;
END_TYPE

E_BA_ObjectType

TYPE E_B_A_ObjectType:
{
  Invalid := 0,
  Undefined := 1,
  eAnalogInput := 10,
  eAnalogOutput := 11,
  eAnalogValue := 12,
  eBinaryInput := 15,
  eBinaryOutput := 16,
  eBinaryValue := 17,
  eMultistateInput := 20,
  eMultistateOutput := 21,
  eMultistateValue := 22,
  eStructuredView := 25,
  eProject := 26,
  eEventClass := 27,
  eCalendar := 28,
  eSchedule := 29,
  eLoop := 30,
  eTrend := 31
} BYTE;
END_TYPE
Room automation

Heating, cooling

**E_BA_Medium**

```plaintext
TYPE E_BA_Medium :
  
  {  
    Invalid : 0,
    eNoMedium : 1,
    eHeatMedium : 2,
    eCoolMedium : 3
  } INT;

END_TYPE
```

**E_BA_PipeSys**

```plaintext
TYPE E_BA_PipeSys :
  
  {  
    Invalid : 0,
    e2Pipe : 1,
    e4Pipe : 2
  } UDINT;

END_TYPE
```

Lighting

**E_BA_ConstLightMode**

```plaintext
TYPE E_BA_Medium :
  
  {  
    eFullAutomatic : 0,
    eSemiAutomatic : 1
  } INT;

END_TYPE
```

- **eFullAutomatic**: In fully automatic mode, the function can be switched on and off via the input \( bPrc \): a positive edge switches the function on. If there is no new rising edge after a falling edge over the time \( nAbcDetc \), the constant light control is switched off.

- **eSemiAutomatic**: In semi-automatic mode, the function is ONLY switched off via the input \( bPrc \), as described under *FullAutomatic*.

Shading

**E_BA_PosMod**

```plaintext
ENUMERATOR for the definition of the positioning mode.

TYPE E_BA_PosMod :
  
  {  
    Invalid := 0,
    eFix := 1,
    eTab := 2,
    eMaxIndc := 3
  } BYTE;

END_TYPE
```

- **PosModFix**: The blind height is a fixed value, which is set at function block \( FB_BA_SunPrtc \) via the value \( fFixPos \) [%].
PosModTab: The height positioning takes place with the help of a table of 6 interpolation points, 4 of which are parameterizable. A blind position is then calculated from these points by linear interpolation, depending on the position of the sun (see FB_BA_BldPosEntry[142]).

PosModMaxIndc: The positioning takes place with specification of the maximum desired incidence of light.

E_BA_ShdObjType

Enumerator for selecting the shading object type.

```plaintext
TYPE E_BA_ShdObjType :
{
  Invalid := 0,
  eTetragon := 1,
  eGlobe := 2
} BYTE;
END_TYPE
```

eTetragon: Object type is a rectangle.
eGlobe: Object type is a ball.

E_BA_SunBldPrio

```plaintext
TYPE E_BA_SunBldPrio :
{
  Invalid := 0,
  eFire := 1,
  eStorm := 2,
  eIce := 3,
  eCommError := 4,
  eBurglary := 5,
  eMaintenance := 6,
  eReferencing := 7,
  eManualActuator := 8,
  eManualGroup := 9,
  eAllDown := 10,
  eAllUp := 11,
  eScene1 := 12,
  eFacadeThermoAutomatic := 13,
  eFacadeTwilightAutomatic := 14,
  eParkPosition := 15,
  eScene2 := 16,
  eScene3 := 17,
  eScene4 := 18,
  eScene5 := 19,
  eSunProtection := 20,
  eGroupThermoAuto := 21,
  eTwilightAuto := 22
} BYTE;
END_TYPE
```

Universal

E_BA_AntBlkgMode

```plaintext
TYPE E_BA_AntBlkgMedium :
{
  eOff := 1,
  eExternalRequest := 2,
  eOffTime := 3
} UDINT;
END_TYPE
```
Types

Room automation

Heating, cooling

ST_BA_SpRmT

Room temperature setpoints.

```plaintext
TYPE ST_BA_SpRmT :
  STRUCT
    fPrtcHtg : REAL := 12.0;
    fEcoHtg  : REAL := 15.0;
    fPreCmfHtg : REAL := 19.0;
    fCmfHtg : REAL := 21.0;
    fPrtcCol : REAL := 40.0;
    fEcoCol : REAL := 35.0;
    fPreCmfCol : REAL := 28.0;
    fCmfCol : REAL := 24.0;
  END_STRUCT
END_TYPE
```

The values in the structure are defined with the preset values.

The variables have the following meaning:

- rPrtcHtg: Protection Heating.
- rEcoHtg: Economy Heating.
- rPreCmfHtg: Pre-Comfort Heating.
- rCmfHtg: Comfort Heating.
- rPrtcCol: Protection Cooling.
- rEcoCol: Economy Cooling.
- rPreCmfCol: Pre-Comfort Cooling.
- rCmfCol: Comfort Cooling.

Shading

ST_BA_BldPosTab

Structure of the interpolation point entries for the height adjustment of the blind.

```plaintext
TYPE ST_BA_BldPosTab :
  STRUCT
    aSunElv : ARRAY[0..5] OF REAL;
    aPos : ARRAY[0..5] OF REAL;
    bVld : BOOL;
  END_STRUCT
END_TYPE
```

aSunElv / aPos: The 6 interpolation points that are transferred, wherein the array elements 0 and 5 represent the automatically generated edge elements mentioned above.

bVld: Validity flag for the function block FB_BA_SunPrtc [187]. It is set to TRUE by the function block FB_BA_BldPosEntry [142] if the data entered correspond to the validity criteria described.
**ST_BA_FcdElem**

List entry for a facade element (window).

```plaintext
TYPE ST_BA_FcdElem:
STRUCT
  fWdwWdth : REAL;
  fWdwHght : REAL;
  aCnr      : ARRAY [1..4] OF ST_BA_Cnr;
  nGrp      : DINT;
  bVld      : BOOL;
END_STRUCT
END_TYPE
```

- **fWdwWdth**: Width of the window.
- **fWdwHght**: Height of the window.
- **aCnr**: Coordinates of the window corners and information as to whether this corner point is in the shade (see `ST_BA_Cnr` [94]).
- **bVld**: Plausibility of the data entered: bVld=TRUE: Data are plausible.

**ST_BA_Cnr**

Information about window corners.

```plaintext
TYPE ST_BA_Cnr :
STRUCT
  fx   : REAL;
  fy   : REAL;
  bShdd: BOOL;
END_STRUCT
END_TYPE
```

- **fx**: X-coordinate of the window (on the facade).
- **fy**: Y-coordinate of the window (on the facade).
- **bShdd**: Information as to whether this corner is in the shade: bShdd=TRUE: Corner is in the shade.

**ST_BA_SunBld**

Structure of the blind positioning telegram.

```plaintext
TYPE ST_BA_SunBld :
STRUCT
  fPos   : REAL;
  fAngl  : REAL;
  bManUp : BOOL;
  bManDwn: BOOL;
  bManMod: BOOL;
  bActv  : BOOL;
END_STRUCT
END_TYPE
```

- **fPos**: Transferred blind height [%].
- **fAngl**: Transferred slat position [°].
- **bManUp**: Manual command: blind up.
- **bManDwn**: Manual command: blind down.
- **bManMod**: TRUE: Manual mode is active. FALSE: Automatic mode is active.
- **bActv**: The sender of the telegram is active. This bit is only evaluated by the priority control. The sun protection actuators `FB_BA_SunBldActr [171]` and `FB_BA_RolBldActr [163]` ignore it.
**ST_BA_ShdoB**

List entry for a shading object.

```plaintext
TYPE ST_BA_ShdObj :
  STRUCT
    fP1x       : REAL;
    fP1y       : REAL;
    fP1z       : REAL;
    fP2x       : REAL;
    fP2y       : REAL;
    fP2z       : REAL;
    fP3x       : REAL;
    fP3y       : REAL;
    fP3z       : REAL;
    fP4x       : REAL;
    fP4y       : REAL;
    fP4z       : REAL;
    fMx        : REAL;
    fMy        : REAL;
    fMz        : REAL;
    fRads      : REAL;
    nBegMth    : USINT;
    nEndMth    : USINT;
    eType      : E_BA_ShdObjType;
    bVld       : BOOL;
  END_STRUCT
END_TYPE
```

- **fP1x...fP4z**: Corner coordinates. Of importance only if the element is a rectangle.
- **fMx...fMz**: Center coordinates. Of importance only if the element is a ball.
- **fRads**: Radius of the ball. Of importance only if the element is a ball.
- **nBegMth**: Beginning of the shading period (month).
- **nEndMth**: End of the shading period (month).
- **eType**: Object type (see **E_BA_ShdObjType [92]**).
- **bVld**: Plausibility of the data: 
  \[ bVld = \text{TRUE}: \text{Data are plausible.} \]

**Remark about the shading period:**

The entries for the months may not be 0 or greater than 12, otherwise all combinations are possible.

**Examples:**

- Start=1, End=1: shading in January.
- Start=1, End=5: shading from the beginning of January to the end of May.
- Start=11, End=5: shading from the beginning of November to the end of May (the following year).

**ST_BA_SunBldScn**

Table entry for a blind scene.

```plaintext
TYPE ST_BA_SunBldScn :
  STRUCT
    fPos       : REAL;
    fAngl      : REAL;
  END_STRUCT
END_TYPE
```

- **fPos**: Blind height [%].
- **fAngl**: Slat position [°].
Function Blocks

Air conditioning equipment

FB_BA_FrstPrtc

The function block is used for frost monitoring of a heating coil in an air conditioning system.

A frost risk is present, if the input \( b_{Frst} \) is TRUE. The frost alarm must be linked in the plant program such that the plant is switched off immediately, the heater valve opens, and the heater pump is switched on.

If there is risk of frost, the output \( b_{On} \) is set, and \( nT1 \) (seconds) is started. If the frost risk remains \( (b_{Frst} = TRUE) \) after \( nT1 \) has elapsed, \( b_{On} \) remains set. It can only be reset at input \( b_{Rst} \).

If the frost alarm ceases due to activation of the heating coil within the time \( nT1 \) \( (b_{Frst} = FALSE) \), the plant automatically restarts. For the plant restart \( b_{On} \) becomes FALSE, and at output \( b_{HW}Rst \) a pulse for acknowledgement of a latching circuit in the control cabinet is issued. With the restart a second monitoring period \( nT2 \) (seconds) is initiated. If another frost alarm occurs within this period, the plant is permanently locked. \( b_{On} \) remains set until the frost alarm has been eliminated and \( b_{Rst} \) has been acknowledged.

In a scenario where frost alarms recur with time offsets that are greater than \( nT2 \), theoretically the plant would keep restarting automatically. In order to avoid this, the restarts within the function block are counted. The parameter \( n_{AlmCnt} \) can be used to set the number of possible automatic restart between 0 and 4.

An acknowledgement at input \( b_{Rst} \) resets the alarm memory within the function block to zero.

Example:

\( t_0 = \) frost alarm at input \( b_{Frst} \), alarm message at output \( b_{On} \), start of timer \( T1 \) \( (nT1 \ [s]) \)
\( t_1 = \) frost alarm off, resetting of \( b_{On} \), output of hardware pulse, start of timer \( T2 \) \( (nT2 \ [s]) \), plant restart
\( t_2 = \) further frost alarm within \( T2 \), alarm message at \( b_{On} \), start of timer \( T1 \), locking of the frost alarm
\( t_3 = \) frost alarm off.
\( t_4 = \) acknowledgement of the alarm at \( b_{Rst} \), resetting of \( b_{On} \).
VAR_INPUT

bFrst : BOOL;
nT1 : UDINT;
nT2 : UDINT;
nAlmCnt : UDINT;
bRst : BOOL;

bFrst: Connection for frost events on the air and water side.

nT1: Timer for restart delays [s]. Internally limited to a minimum value of 0.

nT2: Timer monitoring time [s]. Internally limited to a minimum value of 0.

nAlmCnt: Maximum number of automatic plant restarts without reset. Internally limited to values between 0 and 4.

bRst: Resetting and acknowledgement of the frost alarm.

VAR_OUTPUT

bOn : BOOL;
bHWRst : BOOL;
nRemTi1 : UDINT;
nRemTi2 : UDINT;
bAlmLck : BOOL;
nStaCnt : UDINT;

bOn: Frost alarm active.

bHWRst: Output of a pulse for acknowledgement of the frost protection hardware.

nRemTi1: Time remaining to plant restart after frost alarm.

nRemTi2: Remaining monitoring time.

bAlmLck: Alarm lock - stored alarm.

nStaCnt: Status counter – current number of unacknowledged false starts.
**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

**VAR_INPUT**

- **fT**: Temperature [°C].
- **fHumRel**: Relative humidity [%].
- **fAP**: Hydrostatic air pressure at 1013.25 hPa.

**VAR_OUTPUT**

- **fHumAbs**: Absolute humidity g water per kg dry air [g/Kg].
- **fDewPnt**: Dew point temperature [°C].
- **fE**: Enthalpy [kJ/kg].
- **fDHA**: Density of moist air ρ [kg mixture/m³].
- **fSpecV**: Specific volume [m³/kg].
- **fTWet**: Wet bulb temperature [°C].

**FB_BA_HX**

This function block is used to calculate the dew point temperature, the specific enthalpy and the absolute humidity. The temperature, the relative humidity and the barometric pressure are required for calculating these parameters. The enthalpy is a measure for the energy of a thermodynamic system.
This function block is used to cool down rooms that have been heated up during the day with cool outside air at night. The summer night cooling function serves to improve the quality of the air and to save electrical energy. Electrical energy for cooling is saved during the first hours of the next summer day.

The start conditions for the summer night cooling are defined by parameterizing the FB_BA_NgtCol function block. The function block can be used to open motor-driven windows or to switch air conditioning systems to summer night cooling mode outside their normal hours of operation.

The following conditions must be met for activation of summer night cooling:

- The function block itself is enabled (bEn=TRUE).
- The outside temperature is not too low (fTOts > fTOtsLoLmt).
- The outside temperature is sufficiently low compared to the room temperature (fTRm - fTOts) > fSwiOnDiffT.
- The room temperature is such that it is not worth switching on summer night cooling. fTRm > fSpRm + fTRmHys.

Under the following conditions the summer night cooling is disabled:

- The function block itself is disabled (bEn = FALSE).
- The outside temperature is too low (fTOts < fTOtsLoLmt).
- The outside temperature is too high compared to the room temperature (fTRm - fTOts) < fSwiOffDiffT.
- The room temperature is lower than the setpoint. fTRm ≤ fSpRm.

VAR_INPUT

bEn : BOOL;
fTOts : REAL;
fTRm : REAL;
fSpRm : REAL;
fTOtsLoLmt : REAL;
fTOtsHys : REAL;
fTRmHys : REAL;
fSwiOnDiffT : REAL;
fSwiOffDiffT : REAL;

bEn: Enable function block.
fTOts: Outside temperature [°C].
fTRm: Outside temperature [°C].
fSpRm: Room temperature setpoint.
fTOtsLoLmt: Lower outside temperature limit [°C]; prevents excessive cooling.
fTOtsHys: Hysteresis for minimum outside temperature [°K]. This hysteresis, which at the lower end is internally limited to 0.5 °K, is intended to prevent jitter in bQ, if the outside temperature fluctuates precisely around the value of rTOtsLoLmt.
fTRmHys: Hysteresis for the room temperature [K]. This hysteresis, which at the lower end is internally limited to 0.5 K, is intended to prevent unnecessary fluctuation of bQ, if the room temperature fluctuates precisely around the setpoint rSpRm.

fSwiOnDiffT: Difference between the room temperature and the outside temperature, from which summer night cooling is enabled [K].

fSwiOffDiffT: Difference between the room temperature and the outside temperature, from which summer night cooling is locked [K].

VAR_OUTPUT
bQ : BOOL;

bQ: Summer night cooling on.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

FB_BA_RcvMonit

The function block is used for calculating the efficiency of an energy recovery system.

The function block requires the following measured temperature values for calculating the efficiency (heat recovery rate):

- Outside air temperature fTOts
- Exhaust air temperature fTExh
- Air temperature of the energy recovery system in the inlet air duct (alternatively: in the outlet air duct) fTAftRcv

The function block logs the temperature values every 10 seconds and forms minutely averages from 6 consecutive values. The results are used to check whether the plant has reached a "stable" state.

- This is the case when the recorded temperatures of outside air, exhaust air and air after energy recovery are almost constant, i.e. none of the 6 individual values deviate by more than 0.5 K from the respective mean value.
- The temperature difference between outside air and exhaust air is at least 5 K.

If this is the case, this measuring cycle is acknowledged with a TRUE signal at output bStblOp, and the calculated efficiency is output at fEffc. If the state is not "stable", a FALSE signal appears at output bStblOp, and fEffc is set to 0.

In any case, each measuring and analysis cycle is marked as completed with a trigger (a TRUE signal lasting one PLC cycle) at bNewVal.
Enable \((bEn)\) and Reset \((bRst)\)

The function block is only active if a TRUE signal is present at \(bEn\). Otherwise its execution stops, and all outputs are set to FALSE or 0.0.

An active measuring and evaluation cycle can be terminated at any time by a TRUE signal at \(bRst\). All outputs are set to FALSE or 0.0, and the measuring cycle restarts automatically.

Selection of the temperature value "after recovery" \((bSnsRcvTExh)\)

A FALSE entry at \(bSnsRcvTExh\) means that the temperature measurement after the heat recovery in the supply air duct is used for calculating the efficiency.

To use the temperature measurement after the heat recovery in the exhaust air duct, TRUE must be applied at \(bSnsRcvTExh\).

Limit value exceeded \((fContrVar, fLmtEffc, bLmtRchd)\)

A limit violation has occurred, if the calculated efficiency is less than the specified limit value \(fLmtEffc\), and at the same time the control value for the heat recovery is at 100%. To this end the control value must be linked to input \(fContrVar\).

The limit violation message can be delayed by an entry at \(nLmtVioDly\_sec\): If the two criteria, violation and override, are met for longer than \(nLmtVioDly\_sec\) [s], this is indicated with a TRUE signal at \(bLmtRchd\).

A warning message, which may have occurred, is canceled if a complete measuring cycle provides "good" values, or with a rising edge at \(bRst\) or deactivation of the function block.

\[
\text{This warning message only occurs if the plant is in a stable operation mode (} bStblOp = \text{TRUE).}
\]

Taking into account the temperature increase of the exhaust air due to the fan motor \((fTIncFan)\)

It is possible that the outlet air is warmed by a fan motor, resulting in distortion of the measurement. This temperature increase can be specified through \(fTIncFan\). Internally, the measured outlet air temperature is then reduced by this value.

### VAR_INPUT

\[
\text{bEn: Function block enable.}
\]

\[
bRst: \text{Reset - all determined values are deleted.}
\]

\[
fContrVar: \text{Control value for the heat recovery, i.e. the actual value.}
\]

\[
fTOts: \text{Outside temperature.}
\]

\[
fTExh: \text{Exhaust air temperature.}
\]

\[
fTArtRcv: \text{Temperature after energy recovery.}
\]

\[
bSnsRcvTExh: \text{Temperature at the measuring point after energy recovery: FALSE -> in inlet air duct (SupplyAir) - TRUE -> in outlet air duct (ExhaustAir).}
\]

\[
fTIncFan: \text{Temperature increase due to fan.}
\]

\[
fLmtEffc: \text{Limit value efficiency.}
\]
nLmtVioDly: Limit violation delay [s]. Internally limited to a minimum value of 0.

VAR_OUTPUT

<table>
<thead>
<tr>
<th>bNewVal</th>
<th>BOOL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>fEffc</td>
<td>REAL;</td>
</tr>
<tr>
<td>bLmtRchd</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bStblOp</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>

bNewVal: Output trigger for new value fEffc.

fEffc: Efficiency

bLmtRchd: Limit value reached

bStblOp: Stable operation.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

FB_BA_SpSupVis

Function block for processing and checking the lower and upper setpoint of a supply air humidity or temperature control

Checks and limits for the setpoints

The function block limits the setpoints. The following two tables show which parameters are checked and what the response is in the event of an error.

<table>
<thead>
<tr>
<th>Checking</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>fSpLo &gt; fSpHi</td>
<td>last valid values of fSpLo and fSpHi are used</td>
</tr>
<tr>
<td>fSpMin &gt;= fSpMax</td>
<td>last valid values of fSpMin and fSpMax are used</td>
</tr>
<tr>
<td>fSpHi &gt; fSpMax</td>
<td>fPrSpHi = fSpMax</td>
</tr>
<tr>
<td>fSpLo &lt; fSpMin</td>
<td>fPrSpLo = fSpMin</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Checking</th>
<th>bErr</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>fSpMin &gt;= fSpMax</td>
<td>TRUE</td>
<td>fSpErr = ((fSpMin + fSpMax) / 2)</td>
</tr>
<tr>
<td>fSpHi &lt; fSpMin</td>
<td></td>
<td>fPrSpHi = fSpLo = fPrRcv = fSpErr</td>
</tr>
</tbody>
</table>

The difference between the setpoints describes an energy-neutral zone. With inlet air control, no heating or cooling would take place within the neutral zone.
The checked and, if necessary, limited setpoints are output at the function block output as $f_{PrSpHi}$ and $f_{PrSpLo}$ (Present Setpoint).

**Setpoint for heat recovery**

For heat recovery, the setpoint $f_{SpRcv}$ is optionally calculated from the mean value of the upper and lower setpoint, $f_{SpHi}$ and $f_{SpLo}$, or depending on the control direction of the heat recovery system. The method is defined through the input variable $b_{SlcnSpRcv}$:

<table>
<thead>
<tr>
<th>$b_{SlcnSpRcv}$</th>
<th>$f_{SpRcv}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>Mean value of $f_{SpLo}$ and $f_{SpHi}$</td>
</tr>
<tr>
<td>FALSE</td>
<td>Depends on control direction, defined through input $b_{ActRcv}$</td>
</tr>
</tbody>
</table>

If the setpoint is defined depending on the control direction, the following applies:

<table>
<thead>
<tr>
<th>$b_{ActRcv}$</th>
<th>Control direction</th>
<th>$f_{SpRcv}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>direct (cooling)</td>
<td>$f_{SpHi}$</td>
</tr>
<tr>
<td>FALSE</td>
<td>indirect (heating)</td>
<td>$f_{SpLo}$</td>
</tr>
</tbody>
</table>

**Heat recovery**

- For room temperature > outside temperature, controller direction reverse (heating)
- For room temperature < outside temperature, controller direction direct (cooling)

$$f_{SpHi} \text{ or } \frac{(f_{SpHi}-f_{SpLo})}{2}$$

$$f_{SpLo} \text{ or } \frac{(f_{SpHi}-f_{SpLo})}{2}$$
VAR_INPUT

bEn : BOOL;
fSpHi : REAL;
fSpLo : REAL;
fSpMax : REAL;
fSpMin : REAL;
bActnRcv : BOOL;
bSlcnSpRcv : BOOL;

bEn: Function block enable. If bEn = FALSE, all output parameters are 0.0.
fSpHi: Upper setpoint input value to be checked.
fSpLo: Lower setpoint input value to be checked.
fSpMax: Maximum setpoint.
fSpMin: Minimum setpoint.
bActnRcv: Direction of action of the downstream heat recovery.
bSlcnSpRcv: Setpoint selection of the downstream heat recovery system.

VAR_OUTPUT

fPrSpHi : REAL;
fPrSpLo : REAL;
fSpRcv : REAL;
bErr : BOOL;
sErrDescr : T_MAXSTRING;

fPrSpHi: Output value for the upper setpoint.
fPrSpLo: Output value for the lower setpoint.
fSpRcv: Output value for the resulting heat recovery setpoint.
bErr: This output is switched to TRUE if the parameters entered are erroneous.
sErrDescr: Contains the error description.

Error description

01: Warning: The setpoints are not in a logical order: Either (fSpMin >= fSpMax) OR (fSpHi < fSpMin) OR (fSpLo > fSpMax)

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

Provision of hot water

FB_BA_DHW2P
This function block controls the charging (heating) of a hot water tank via an on-off controller. Tank heating is activated at input \( bEn \). If tank heating is active the output \( bLd \) is TRUE. The variable \( fSp \) is used to transfer the setpoint for the service water temperature to the function block. At input \( fTMin \) a minimum selection of all temperature sensors for the hot water tank is connected, at input \( fTMax \) a maximum selection of all temperature sensors.

Due to the thermal stratification in the hot water tank, the sensor at the top is generally the one showing the highest temperature, the one at the bottom the lowest.

The tank can be charged in two ways via the variables \( bKepFul \):

**\( bKepFul = FALSE \)**

Charging is requested if \( fTMax \) falls below the value of \( fSp-fSpHys \). The charge request is disabled if \( fTMin \) is above the setpoint for \( fSp \).

Due to the fact that the sensor at the top generally measures the highest temperature, the heating is not switched on until the hot water tank has been discharged.

**\( bKepFul = TRUE \)**

Charging is requested if \( fTMin \) falls below the value of \( fSp-fSpHys \). The charge request is disabled once \( fTMin \) is above the setpoint again.

Selecting the minimum of all tank temperatures ensures that the coldest point of the tank is used for control purposes. Recharging takes place when the tank is no longer full.
**VAR_INPUT**

bEn : BOOL;
fSp : REAL;
fSpHys : REAL;
fTMax : REAL;
fTMin : REAL;
bKepFul : BOOL;

**bEn**: Enable boiler charging.

**fSp**: Service water temperature setpoint [°C].

**fSpHys**: Hysteresis, recommended 1°K to 5°K.

**fTMax**: Maximum selection of all tank temperatures [°C].

**fTMin**: Minimum selection of all tank temperatures [°C].

**bKepFul**: Control temperature selection:

FALSE = $fTMax$ is used to request $bLd$, $fTMin$ to switch off

TRUE = $fTMin$ alone controls switching on/off of $bLd$

**VAR_OUTPUT**

bLd : BOOL;
fSpOut : REAL;

**bLd**: Enable charging mode.

**fSpOut**: Setpoint transfer to charging circuit:

- $fSpOut = fSp$ (input) if the function block is enabled
- $fSpOut = 0$ if the function block is not enabled

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

**FB_BA_LglPrev**

This function block is used for disinfection of the service water and for killing off Legionella. Disinfection mode is activated at input $bEnLglPrev$ via a timer program. It is advisable to run the disinfection at least once per week (during the night). The temperature should be at least 70 °C. The activation interval at $bEnLglPrev$ must be adequately long. The output $bLd$ activates tank heating.

For hot water tanks with several temperature sensors, a minimum selection feature for all sensors must be connected at $fTMin$.

If $fTMin$ exceeds the value of $fSp$, a monitoring timer is started with a time of $nTi\_sec$ [s]. If the minimum tank temperature $fTMin$ remains above $fSp - fSpHys$ while the timer is active, the tank was heated adequately. If circulation is active, the output $bLd$ must be linked to enabling of the circulation pump, to ensure that the water pipe within the service water system is included in the disinfection. If the temperature has fallen below $fSp-fSpHys$ during the disinfection process, this process must be restarted and run until the time $nTi$ has fully
If the disinfection was successful, the output \( bLd \) is reset. If the disinfection process was incomplete during the function block activation \( (bEnLglPrev) \), this is indicated with the output \( bAlm \). The output must be reset with \( bRst \).

**Explanation of the diagram:**

\( t_0 \): Start of the legionella program and switching of output \( bLd \). Heating of the hot water tank.

\( t_1 \): The tank has reached the temperature \( fSp \). The timer for the heating time is started.

\( t_2 \): The minimum tank temperature has fallen below \( fSp - fSpHys \). The timer for the heating time is reset.

\( t_3 \): The temperature exceeds \( fSp \) again, and the heating timer is started again.

\( t_4 \): The minimum tank temperature was above the limit \( fSp - fSpHys \) over the period \( nTi \); the disinfection was successful. \( bLd \) is reset, and the hot water tank switches back to normal operation.

**VAR_INPUT**

- \( bEnLglPrev \): BOOL;
- \( fTMin \): REAL;
- \( fSp \): REAL;
- \( fSpHys \): REAL;
- \( nTi \): UDINT;
- \( bRst \): BOOL;

- \( bEnLglPrev \): Enabling of disinfection operation via a timer program.

- \( fTMin \): Minimum tank temperature \(^\circ C\). Minimum selection of temperature sensors at the top and bottom.

- \( fSp \): Setpoint for disinfection \(^\circ C\).

- \( fSpHys \): Temperature difference \(^\circ K\) lower limit; always calculated absolute.
nTi: Monitoring period [s].

bRst: Resetting of the legionella alarm.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bLd</td>
<td>BOOL</td>
<td>Anti-legionella mode active.</td>
</tr>
</tbody>
</table>
| fSpOut   | REAL  | Setpoint transfer to charging circuit:
|          |       | - fSp (input) if the function block is enabled |
|          |       | - 0 if the function block is not enabled |
| nRTi     | UDINT | Disinfection mode timer countdown. |
| nSta     | UDINT | Disinfection program status: |
|          |       | 1. The disinfection operation was successful. |
|          |       | 2. The disinfection was completed successfully. After the disinfection, and to reactivate legionella prevention, bEnLglPrev must be FALSE. |
|          |       | 3. The disinfection operation is active. |
|          |       | 4. Disinfection was not successful. Alarm is pending. |
|          |       | 5. Disinfection was not successful, the alarm was acknowledged. |
|          |       | 6. Controller restart, or legionella mode has not yet been requested. |
| bAlm     |       | The temperature setpoint was not reached consistently over the interval nTi, so that adequate disinfection is not guaranteed. |

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

**Room automation**

**Heating, cooling**

**FB_BA_FnctSel**

![FB_BA_FnctSel diagram]
The function block is used for enabling heating or cooling mode in a room. The distribution network type plays a significant role:

In a two-pipe system, all rooms served by the plant can either be heated or cooled at any one time.
In a four-pipe system, the room conditioning can be demand-based, i.e. some rooms can be heated, while other rooms can be cooled by the same plant.

The function block used for each room, as already mentioned, selects its controllers, depending on which type of piping system is available:

**Two-pipe network**

The two-pipe system is selected if $ePipeSys.e2Pipe$ is set at the input of the function block. Since all rooms served by the plant can only either be heated or cooled, the choice is specified centrally for all rooms via the input $eMedium$. If $eMedium$ is FALSE, the room heating controller is selected. If the input is TRUE the cooling controller is selected. The controller enable states $bEnHtg$ and $bEnCol$ are always issued with a delay of $nChgOvrDly$ [s]. In other words, heating cannot be enabled until the cooling enable state $bEnCol$ for $nChgOvrDly$ is FALSE, and vice versa. In addition to the elapsing of this changeover time, the system checks that the output from controller to be switched off is 0.0. This is based on feedback at the inputs $fCtrlValHtg$ and $fCtrlValCol$. In this way, a drastic change from heating to cooling and vice versa is avoided.

**Four-pipe network**

The four-pipe system is selected if $ePipeSys.e4Pipe$ is set at the input of the function block. In this case, the choice of controller can be different for the individual rooms as required, based on the room temperature $fRmT$ and the setpoints $fSpHtg$ for heating and $fSpCol$ for cooling. If the room temperature exceeds the setpoint, the cooling controller is activated ($bEnCol$), if it falls below the heating setpoint, the heating controller is activated ($bEnHtg$). If the temperature is between the two setpoints, both controllers are switched off (energy-neutral zone). Here too, the output of the controller enable states $bEnHtg$ and $bEnCol$ is delayed by $nChgOvrDly$ [s] (see **two-pipe network**). In addition to the elapsing of this switching time, the system checks that the output from controller to be switched off is 0.0. This is based on feedback at the inputs $fCtrlValHtg$ and $fCtrlValCol$. In this way, a drastic change from heating to cooling and vice versa is avoided, if the changeover time is inadequate.

**Dew-point monitor** ($bDewPnt$)

In both systems (two- and four-pipe) the dew-point monitor has the task of deactivating cooling immediately, if required.

**Program sequence**

The function block can have 3 possible states:

1. Waiting for heating or cooling enable
2. Heating enable
3. Cooling enable

In the first step, the function block waits for compliance with the conditions required for heating or cooling:
If a chain of conditions is met, the function block switches to the respective state (heating or cooling) and remains in this state until the corresponding controller issues 0 at the function block input \((fCtrlValHtg/ fCtrlValCol)\). This ensures that only one controller is active at any one time, even if a high heating controller output, for example, would call for a brief cooling intervention (overshoot). Heating or cooling continues until there is no longer a demand.

There are 3 exceptions, for which heating or cooling is immediately interrupted:

1. In the two-pipe system \((ePipeSys = E\_BA\_PipeSys.2Pipe)\) heating is active \((bEnHtg)\) but the system has been switched to cooling medium \((eMedium = E\_BA\_Medium.eCoolMedium)\)
2. In the two-pipe system \((ePipeSys = E\_BA\_PipeSys.2Pipe)\) cooling is active \((bEnCol)\) but the system has been switched to heating medium \((eMedium = E\_BA\_Medium.eHeatMedium)\)
3. The dew-point monitor was triggered \((bDewPnt=TRUE)\) in cooling mode (two- or four-pipe system)

In these cases the heating or cooling enable states are canceled, and the plant switches to standby.

**Demand message \((eReqdMedium)\)**

To notify the plant of the current demand for heating or cooling, a demand ID is issued at the function block output, i.e. for each room, depending on the actual and set temperature. These can be collected and evaluated centrally. The evaluation always takes place, irrespective of the network type (two- or four-pipe).

<table>
<thead>
<tr>
<th>eReqdMedium</th>
<th>Medium</th>
<th>Room temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No medium is requested</td>
<td>(fRmT &gt; fSpHtg \text{ AND } fRmT &lt; fSpCol)</td>
</tr>
<tr>
<td>2</td>
<td>Heating medium is requested</td>
<td>(fRmT &lt; fSpHtg)</td>
</tr>
<tr>
<td>3</td>
<td>Cooling medium is requested</td>
<td>(fRmT &gt; fSpCol)</td>
</tr>
</tbody>
</table>

**Error handling**

The heating setpoint must not be greater than or equal to the cooling setpoint, since this would result in a temperature range with simultaneous heating and cooling demand. However, since the function block only issues one enable state at a time (i.e. heating or cooling), the case is harmless from a plant engineering perspective. In this case only a warning message is issued \((bErr = TRUE, sErrDescr = \text{warning message});\) the function block does not interrupt its cycle.

**VAR_INPUT**

- \(ePipeSys\): Pipe system (2Pipe, 4Pipe) of the plant (see \(E\_BA\_PipeSys\) [91]).
- \(eMedium\): Selection of the medium for the entire two-pipe network (NoMedium, HeatMedium, CoolMedium) (see \(E\_BA\_Medium\) [91]).
- \(bDewPnt\): Dew-point monitor: If \(bDewPnt = \text{FALSE}\), the cooling controller is locked.
- \(fTRm\): Room temperature.
- \(fSpHtg\): Heating setpoint.
- \(fSpCol\): Cooling setpoint.
- \(fCtrlValHtg\): Current output value of the heating controller. Used internally as switching criterion from heating to cooling: \(fCtrlValHtg\) must be 0.
- \(fCtrlValCol\): Current output value of the cooling controller. Used internally as switching criterion from cooling to heating: \(fCtrlValCol\) must be 0.
nChgOvrDel: Switchover delay [s] from heating to cooling or vice versa. Internally limited to a minimum value of 0.

VAR_OUTPUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEnHtg</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEnCol</td>
<td>BOOL</td>
</tr>
<tr>
<td>eReqdMedium</td>
<td>E_BA_Medium</td>
</tr>
<tr>
<td>nRemTiChgOvrDlyHtg</td>
<td>UDINT</td>
</tr>
<tr>
<td>nRemTiChgOvrDlyCol</td>
<td>UDINT</td>
</tr>
<tr>
<td>bErr</td>
<td>BOOL</td>
</tr>
<tr>
<td>sErrDescr</td>
<td>T_MAXSTRING</td>
</tr>
</tbody>
</table>

bEnHtg: Heating controller enable.

bEnCol: Cooling controller enable.

eReqdMedium: Requested medium (see Determination of needs [p. 110]).

nRemTiChgOvrDlyHtg: Countdown [s] for switchover delay from cooling to heating.

RemTiChgOvrDlyCol: Countdown [s] for switchover delay from heating to cooling.

bErr: In case of a fault, e.g. if warning stages are active, this output is set to TRUE.

sErrDescr: Contains the error description.

Error description
01: Warning: The heating setpoint is higher than or equal to the cooling setpoint

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

FB_BA_RmTAdj

The function block FB_BA_RmTAdj is used for user adjustment of the room temperature setpoint. It shifts the setpoints at the input of a function block depending on an offset \( f_{RmTAdj} \), as shown in the following diagram. At the \( f_{RmTAdj} \) input, the value of a resistance potentiometer or a bus-capable field device can be used for the setpoint correction.
If the set value \( fRmTAdj \) is greater than zero, room heating is requested: The Comfort Heating value is increased by \( fRmTAdj \). At the same time, the values for Comfort Cooling and PreComfort Cooling are increased. If the value \( fRmTAdj \) is less than zero, a lower room temperature is requested. Analogous to the heating case, the values for Comfort Cooling, Comfort Heating and PreComfort Heating are now reduced by the value \( fRmTAdj \).

**Auto-correction**

The temperature adjustment is intended for small corrections of the values. Although it is possible to enter any input values, a heating system will only work in a meaningful manner if the setpoints have ascending values in the following order:
• Protection Heating
• Economy Heating
• Precomfort Heating
• Comfort Heating
• Comfort Cooling
• Precomfort Cooling
• Economy Cooling
• Protection Cooling

Auto-correction works according to the following principle: Starting with the value Economy Heating, the system checks whether this value is smaller than the lower value of Protection Heating. If this is the case, the value for Economy Heating is adjusted to match the value for Protection Heating. The system then checks whether the value for Precomfort Heating is less than Economy Heating and so on, until the value for Protection Cooling is compared with the value for Economy Cooling. If one or several values were corrected, this is indicated with a TRUE signal at output $bValCorr$.

**VAR_INPUT**

| fRmTAdj     : REAL; |
| stSp        : ST_BA_SpRmT; |

fRmTAdj: Room temperature offset value.

stSp: Input structure for the setpoints (see $ST_BA_SpRmT$).

**VAR_OUTPUT**

| bValCorr   : BOOL; |
| fPrPrtcHtg : REAL; |
| fPrEcoHtg  : REAL; |
| fPrPreCmfHtg : REAL; |
| fPrCmfHtg  : REAL; |
| fPrPrtcCol : REAL; |
| fPrEcoCol  : REAL; |
| fPrPreCmfCol : REAL; |
| fPrCmfCol  : REAL; |
| stPrSp     : ST_BA_SpRmT; |

bValCorr: Autocorrection for the values was performed, see above.

rPrPrtcHtg: Resulting Protection Heating setpoint.

rPrEcoHtg: Resulting Economy Heating setpoint.

rPrPreCmfHtg: Resulting PreComfort Heating setpoint.

rPrCmfHtg: Resulting Comfort Heating setpoint.

rPrCmfCol: Resulting Comfort Cooling setpoint.

rPrPreCmfCol: Resulting PreComfort Cooling setpoint.

rPrEcoCol: Resulting Economy Cooling setpoint.

rPrPrtcCol: Resulting Protection Cooling setpoint.

stPrSp: Consolidated output of the resulting values in a structure (see $ST_BA_SpRmT$).

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
The function block FB_BA_SpRmT assigns setpoints for cooling and heating operation to each of the energy levels Protection, Economy, PreComfort and Comfort. The following graphics illustrates the behavior of the function block; the entered values should be regarded as examples:
The parameter \( f_{\text{ShiftHtg}} \) is applied to the Comfort and Precomfort values for the heating mode as central setpoint shift. In addition, winter compensation \( f_{\text{WinCpsn}} \) is applied.

Similarly, the following applies for the cooling mode: The parameter \( f_{\text{ShiftCol}} \) is applied to the Comfort and Precomfort values. In addition, the summer compensation value \( f_{\text{SumCpsn}} \) is applied.
Auto-correction

The temperature adjustment is intended for small corrections of the values. Although it is possible to enter any input values, a heating system will only work in a meaningful manner if the setpoints have ascending values in the following order:

- Protection Heating
- Economy Heating
- Precomfort Heating
- Comfort Heating
- Comfort Cooling
- Precomfort Cooling
- Economy Cooling
- Protection Cooling

Auto-correction works according to the following principle: Starting with the value Economy Heating, the system checks whether this value is smaller than the lower value of Protection Heating. If this is the case, the value for Economy Heating is adjusted to match the value for Protection Heating. The system then checks whether the value for Precomfort Heating is less than Economy Heating and so on, until the value for Protection Cooling is compared with the value for Economy Cooling. If one or several values were corrected, this is indicated with a TRUE signal at output \( bValCorr \).

VAR_INPUT

\[
\begin{align*}
&fPrtcHtg &: \text{REAL;} \\
&fEcoHtg &: \text{REAL;} \\
&fPreCmfHtg &: \text{REAL;} \\
&fCmfHtg &: \text{REAL;} \\
&fPreCmfCol &: \text{REAL;} \\
&fCmfCol &: \text{REAL;} \\
&fPrtcCol &: \text{REAL;} \\
&fShiftHtg &: \text{REAL;} \\
&fShiftCol &: \text{REAL;} \\
&fSumCpsn &: \text{REAL;} \\
&fWinCpsn &: \text{REAL;}
\end{align*}
\]

- \( fPrtcHtg \): Basic Protection Heating setpoint.
- \( fEcoHtg \): Basic Economy Heating setpoint.
- \( fPreCmfHtg \): Basic PreComfort Heating setpoint.
- \( fCmfHtg \): Basic Comfort Heating setpoint.
- \( fCmfCol \): Basic Comfort Cooling setpoint.
- \( fPreCmfCol \): Basic PreComfort Cooling setpoint.
- \( fEcoCol \): Basic Economy Cooling setpoint.
- \( fPrtcCol \): Basic Protection Cooling setpoint.
- \( fShiftHtg \): Heating setpoint value shift.
- \( fShiftCol \): Cooling setpoint value shift.
- \( fSumCpsn \): Summer compensation value.
- \( fWinCpsn \): Winter compensation value.

VAR_OUTPUT

\[
\begin{align*}
&bValCorr &: \text{BOOL;} \\
&fPrPrtcHtg &: \text{REAL;} \\
&fPrEcoHtg &: \text{REAL;} \\
&fPrPreCmfHtg &: \text{REAL;} \\
&fPrCmfHtg &: \text{REAL;} \\
&fPrCmfCol &: \text{REAL;}
\end{align*}
\]
bValCorr: Autocorrection: At least one of the resulting setpoints was adjusted such that the values continue to monotonically increase.

fPrPrtcHtg: Resulting Protection Heating setpoint.

fPrEcoHtg: Resulting Economy Heating setpoint.

fPrPreCmfHtg: Resulting PreComfort Heating setpoint.

fPrCmfHtg: Resulting Comfort Heating setpoint.

fPrCmfCol: Resulting Comfort Cooling setpoint.

fPrPreCmfCol: Resulting PreComfort Cooling setpoint.

rPrEcoCol: Resulting Economy Cooling setpoint.

fPrPrtcCol: Resulting Protection Cooling setpoint.

stPrSp: Consolidated output of the resulting values in a structure (see ST_BA_SpRmT).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

Lighting

FB_BA_AutoLgt

Automatic light with output between 0% and 100%; the control value is subject to hysteresis. The function block is typically used for corridor lighting.

Switching the function on and off

Basically, the control can always be switched on and off alternately via keystrokes at bSwi, via bOn only on and via bOff only off.

The input bPrc is used to integrate an additional occupancy sensor: a positive edge at this input switches the function on. If there is no new rising edge after a falling edge during the time nAbcDetc [s], the function is deactivated and the light is switched off.

The occupancy sensor function is subordinate to the other switching options.
Even if it is defective and either permanently on or off, the function can always be activated or deactivated via the inputs \texttt{bOn} or \texttt{bOff}.

Since the automatic light can also output the light value "0", an additional output \texttt{bActive} indicates the on-state of the function.

\textbf{Active behavior}

When switching on, the light is switched to the value present at \texttt{fCtrlLgtVal}.

The light is switched with a ramp that is specified at \texttt{nSwiTi} in seconds relative to 0 to 100%.

The light control value then continues to follow the input \texttt{fCtrlLgtVal}, but now with a different ramp time, which can be set in seconds at \texttt{nRampTi}.

To prevent the ramp function block from constantly specifying new light values, a hysteresis range can be specified at the \texttt{fHys} input. The function block only readjusts if the default value \texttt{fCtrlLgtVal} is $\frac{1}{2}\times fHys$ above or $\frac{1}{2}\times fHys$ below the output control value \texttt{fOut}.

The function block will in any case pass the values \texttt{fCtrlLgtVal} = 0.0 and \texttt{fCtrlLgtVal} = 100.0 to \texttt{fOut}, even if the hysteresis would not allow this. This ensures that the maximum value and the minimum value are always available.

\textbf{VAR_INPUT}

\begin{verbatim}
VAR_INPUT
bSwi           : BOOL;
bOn            : BOOL;
bOff           : BOOL;
bPrc           : BOOL;
fCtrlLgtVal    : REAL;
nSwiTi         : UDINT;
nRampTi        : UDINT;
fHys           : REAL;
nAbcDetc       : UDINT;
\end{verbatim}

\texttt{bSwi}: A rising edge at this input switches the function block and the light on and off alternately.

\texttt{bOn}: A rising edge at this input switches on the function block and the light on.

\texttt{bOff}: A rising edge at this input switches off the light and the overall function of the function block.

\texttt{bPrc}: A positive edge at the occupancy detection input switches the function on. If there is no new rising edge after a falling edge over the time \texttt{nAbcDetc}, the light control function is switched off.

If this input remains unassigned, it has no function.

\texttt{fCtrlLgtVal}: light control value. When the function block is active, the light control output \texttt{fOut} is ramped to this input (\texttt{nRampTi}) and controlled based on hysteresis.

\texttt{nSwiTi}: dimming control ramp. Instead of setting the light directly to a specific value when switching it on or off, it is perceived as more agreeable if the light is controlled via a dimming ramp. The time [s] entered here is required to switch the light from 0 to 100\% and vice versa.

\texttt{nRampTi}: light dimming control ramp. The time [s] refers to dimming from 0 to 100\% or vice versa.

\texttt{fHys}: hysteresis range around the light control value. A hysteresis range can be specified to prevent the function block from constantly specifying new light values (see \texttt{FB_BA_AutoLgt}).

\texttt{nAbcDetc}: occupancy sensor hold time. If the presence input \texttt{bPrc} is used, the input \texttt{bPrc} must be set to FALSE for this time before absence is detected and the system is switched off.

\textbf{VAR_OUTPUT}

\begin{verbatim}
VAR_OUTPUT
fOut           : REAL;
bLgt           : BOOL;
bActive        : BOOL;
\end{verbatim}

\texttt{fOut}: Control output for light actuators (0...100\%).

\texttt{bLgt}: Binary output for light control: \texttt{fOut} = 0 \rightarrow \texttt{bLgt} = FALSE, otherwise \texttt{bLgt} = TRUE.
bActive: Automatic light is active. However, the light output value may well be "0".

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

**FB_BA_ConstLgtCtrl**

Constant light control with analog output 0 to 100%.

**Switching the control on and off**

Basically, the control can always be switched on and off alternately via short keystrokes at bSwi, bSwiUp and bSwiDwn, via bOn only on and via bOff only off. In addition, a occupancy sensor can also be active at the input bPrc; the behavior depends on eMode (see E_BA_ConstLightMode[91]):

**FullAutomatic**: In fully automatic mode, the function can be switched on and off via the input bPrc: a positive edge switches the function on. If there is no new rising edge after a falling edge over the time nAbcDetc, the constant light control is deactivated and the light is switched off.

**SemiAutomatic**: In semi-automatic mode, the function is ONLY switched off via the input bPrc, as described under FullAutomatic.

The occupancy sensor function is subordinate to the other switching options.

Even if it is defective and either permanently on or off, the control can always be activated or deactivated via the options referred to above.

Since an active constant light control can also output the light value "0", an additional output bActive indicates the on-state of the control.

**Switch-on behavior**

The switch-on value of the light control depends on whether memory mode is selected (bMemMod=TRUE) or not.
In memory mode, the light control will initially assume the value it had before the last switch-off. However, this value is limited to the minimum control value $f_{MinVal}$.

Without memory mode, the value is initially switched to $f_{OnVal}$.

The light is switched with a ramp that is specified at $n_{SwiTi} [s]$ relative to 0 to 100%.

Once the selected light value is reached, the control pauses for 1 second (this value is fixed in the software), so that the light sensor under $f_{Brtns}$ can "get used to" the light. The brightness that is then measured is used as the setpoint for the control.

Control behavior

The control tries to maintain the setpoint: if the incident daylight increases, the light control value at the output $f_{Out}$ is reduced; if the overall brightness decreases, the light control value is increased. The change follows a ramp, which is specified at $n_{RampTi} [s]$ in relation to 0 to 100%. The constant light control is a simple I-controller.

To prevent the ramp function block from constantly specifying new light values, a hysteresis range can be specified at the $f_{Hys}$ input. The function block only becomes active when the total brightness is $\frac{1}{2} \times f_{Hys}$ above or $\frac{1}{2} \times f_{Hys}$ below the previously stored setpoint.

While the control is active, the brightness value can be dimmed up by pressing and holding $b_{SwiDwn}$ / $b_{SwiUp}$ / $b_{Swi}/b_{SwiUp}$. Dimming takes place via a ramp, which is specified at $n_{DimTi}$. The control is interrupted for the duration of this manual intervention. The input $b_{DimCtrlMod}$ now determines whether control is resumed after manual intervention or whether the light remains fixed at the set value:

$b_{DimCtrlMod} = TRUE$:

After completion of the manual intervention and a fixed waiting time of 1 second, the function block returns to normal operation. Dimming is thus used to set a new light setpoint. The waiting time of 1 s is necessary so that a light sensor can reliably detect the new total brightness.

$b_{DimCtrlMod} = FALSE$:

After the end of the manual intervention, the function block remains at the set light value. The control is now no longer active. A positive edge at $b_{DimCtrlMod}$ causes the function block to resume control until the next dimming operation.

Minimum control value

Changes in the lower brightness range are often perceived as irritating.

It can therefore be useful to limit the light control to a minimum control value and then switch off the light completely when the outdoor brightness is very high.

In this case, when the control has reached its minimum control value $f_{MinVal}$, instead of reducing further it remains at the minimum value for the time $n_{OffDly} [s]$ and then switches $f_{Out}$ to 0.

The measured brightness value is stored before and after switching. The difference resulting from this is the brightness increase by switching on to the minimum value.

The new threshold for switching on again is now the control setpoint minus the brightness increase. If the measured brightness reaches or falls below this threshold value for the time $n_{OnDly} [s]$, the light is switched on again and then controlled to the setpoint.

The following diagram is intended to illustrate the choice of threshold value:
1. The measured brightness \( f_{Brtns} \) initially moves around the internal setpoint \( f_{SP} \) within the tolerance limits. Towards the end of this phase, the brightness exceeds the tolerance range.

2. The light control dims the light (control value \( f_{Out} \)) until the light setpoint is reached or the light level falls just below in the PLC cycle. The measured light value \( f_{Brtns} \) can therefore be slightly below the setpoint, but this is accepted in order to avoid continuous readjustment.

3. The measured brightness \( f_{Brtns} \) is again within the tolerance limits, as in phase 1.

4. It is controlled again, this time to the minimum output control value, which brings the total measured brightness back into the tolerance range.

5. The measured brightness increases again due to outdoor brightness and exceeds the tolerance range.

6. The measured brightness \( f_{Brtns} \) remains above the tolerance range, the internal time \( n_{OffDly} \) [s] expires.

7. The light is switched off (\( f_{Out} = 0 \)). The difference between the measured brightness before and after switching off is recorded internally as a brightness gain by switching on the light to the minimum control value. The new threshold that must now be undershot is the control setpoint minus the brightness gain. The choice of this threshold is based on the following considerations:
   - Switching the light on again must not result in the measured brightness subsequently being above the tolerance range again, as this could result in the light continuously being switched on and off.
   - This threshold is nevertheless realistic, i.e. > 0: at the start of phase 5, the outdoor brightness was already so high that the control was set to the minimum value. Before the light is switched off in phase 7, more outdoor brightness was added. In the diagram it is clear that the new threshold is reached when the increase in outdoor brightness from the start of phase 5 to the end of phase 6 has passed again – a realistic scenario. To be on the safe side, however, an internal check is made not only for falling below the threshold value but also for reaching it (less than or equal to).

8. The threshold value is undershot for the time \( n_{OnDly} \) [s].

9. The light is again first switched to the minimum value and then dimmed up until the measured brightness corresponds to the setpoint value \( f_{SP} \).

10. The measured brightness \( f_{Brtns} \) again moves around the internal setpoint \( f_{SP} \) within the tolerance limits.

### VAR_INPUT

- \( b_{Swi} \) : BOOL;
- \( b_{SwiUp} \) : BOOL;
- \( b_{SwiDwn} \) : BOOL;
- \( b_{On} \) : BOOL;
- \( b_{Off} \) : BOOL;
- \( b_{Prc} \) : BOOL;
bSwi: A rising edge at this input switches the function block and the light on and off alternately. Either the value \( f_\text{OnVal} \) is used or, if memory mode is activated (\( b\text{MemMod}=\text{TRUE} \)), the value that was valid before the last switch-off is used, with the lower limit set to the minimum control value \( f\text{MinVal} \).

Long button presses alternately dim and brighten the light. If the dimming value reaches the maximum value of 100%, it remains there until the button is pressed again; there is no automatic change. The same applies to the minimum value for \( f\text{MinVal} \).

bSwiUp: Short pulses at this input have the same switch-on / switch-off behavior as at the input \( b\text{Swi} \). Long button presses, however, only dim up the light in a targeted manner. If the dimming value reaches the maximum value of 100%, it remains there.

bSwiDwn: Short pulses at this input have the same switch-on / switch-off behavior as at the input \( b\text{Swi} \). Long button presses, however, only dim down the light in a targeted manner. If the dimming value reaches the minimum control value \( f\text{MinVal} \), it remains there.

bOn: A rising edge at this input switches the control on if it was previously switched off (output \( b\text{Active} \)).

If the control was previously switched off, the output value changes either to \( f\text{OnVal} \) or, if memory mode is activated (\( b\text{MemMod} = \text{TRUE} \)) to the value that was valid before the last switch-off, limited downwards to the minimum control value \( f\text{MinVal} \).

If the control was already active before, the light output value does not change.

If the control had the light output value \( fOut = 0 \), it will remain at \( fOut = 0 \) after the rising edge at \( b\text{On} \).

bOff: A rising edge at this input switches off the light and the overall function of the function block.

bPrc: A positive edge at the occupancy detection input switches the function on. If there is no new rising edge after a falling edge over the time \( n\text{AbcDetc} \), the constant light control is switched off.

In semi-automatic mode the function is ONLY switched off.

If this input remains unassigned, it has no function.

fBrtns: Measured total light value. The unit of this variable is irrelevant in that after switching on the light and transitioning to ramp control mode, the value present is remembered and then kept constant by ramping the light up and down, as long as no further commands are sent to the inputs of the function block; see Introduction.

nSwiOvrTi: Switching time [ms] for detecting a short or long keypress.

nSwiTi: dimming control ramp. Instead of setting the light directly to a specific value when switching it on or off, it is perceived as more agreeable if the light is controlled via a dimming ramp. The time [s] entered here is used to change the brightness from 0 to 100% and vice versa.

nRampTi: light dimming control ramp. The time [s] refers to dimming from 0 to 100% or vice versa.

eMode: Selection of the operation mode (see \( \text{E_BA_ConstLightMode} \ [91] \)).
bDimCtrlMod: While the control is active, the brightness value can be dimmed up by pressing and holding bSwi/bSwiUp/bSwiDwn. The control is interrupted for the duration of this manual intervention. The input bDimCtrlMod now determines whether control is resumed after manual intervention (bDimCtrlMod= TRUE) or whether the light remains fixed at the set value (bDimCtrlMod= FALSE).

bMemMod / fOnVal: If bMemMod = FALSE, the output is switched to fOnVal when the light is switched on.
If bMemMod = TRUE (memory-mode), the last value is stored before the light is switched off and used as the resetting value.

fMinVal: Minimum light control value before the control switches off the light.

nOffDly: If the light control wants to output a light control value below the entered minimum value fMinVal over a longer period of time nOffDly [s], the light is switched off.

nOnDly: If the measured total brightness falls below the setpoint of the control for an extended period, this is the time in seconds that the system waits until the control switches on the light and controls it to the appropriate value.

fHys: Hysteresis range around the total brightness to be controlled. A hysteresis range can be specified to prevent the function block from constantly specifying new light values (see FB_BA_AutoLgt [118]).

nPrc: occupancy sensor hold time. If the presence input bPrc is used, the input bPrc must be set to FALSE for this time before absence is detected and the system is switched off. This only happens if the input eMode is configured accordingly.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>fOut</th>
<th>REAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>fSP</td>
<td>REAL</td>
</tr>
<tr>
<td>bLgt</td>
<td>BOOL</td>
</tr>
<tr>
<td>bActive</td>
<td>BOOL</td>
</tr>
<tr>
<td>bCtrlOn</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

fOut: Control output for light actuators (0...100%).

fSp: Light control setpoint. The unit is the same as that of the brightness input fBrtns.

bLgt: Binary output for light control: fOut = 0 -> bLgt = FALSE, otherwise bLgt = TRUE.

bActive: The function block is active, either in control mode or in the set dimming mode (see bDimCtrlMod). However, the light output value may well be "0".

bCtrlOn: Constant light control is active if the output is TRUE. However, the light output value may well be "0".

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
FB_BA_DimSwi

Function block for switching and dimming light actuators with analog output (0...100%).

Short keypresses (less than \(n\text{SwiOvrTi}\) [ms]) at \(b\text{Swi}\), \(b\text{SwiUp}\) and \(b\text{SwiDwn}\) switch the light on if it was previously off \((f\text{Out}= 0)\) and off if it was previously on \((f\text{Out}> 0)\). With \(b\text{On}\) and \(b\text{Off}\), targeted switching on or off occurs (central commands).

Switching on is possible to two different values: In memory mode (parameter \(b\text{MemMod}= \text{TRUE}\)), the value that the light had before it was last switched off is saved. It is used as control value when the light is switched on again. If the parameter \(b\text{MemMod}\) is FALSE, the light is controlled to the parameter \(f\text{OnVal}\).

Switching on or off always takes place with a ramp defined with the time duration \(n\text{SwiTi}\) in relation to a dimming range of 0 to 100%.

Long button presses on \(b\text{Swi}\), \(b\text{SwiUp}\) and \(b\text{SwiDwn}\) can be used to dim the light; \(b\text{Swi}\) is intended for on-switch operation and alternate dimming up and down.

Dimming follows a ramp which is defined with the time duration \(n\text{DimTi}\) in relation to a dimming range of 0 to 100%.

\(b\text{SetVal}\) controls \(f\text{Out}\) to a value stored under \(f\text{SetVal}\).

---

*Once the light has been switched on and adjusted, the light value cannot be set again to \(r\text{On}\) by re-triggering the input \(b\text{On}\) or to the last stored value in memory mode. \(b\text{On}\) means central switching on when the light is off.*

---

**VAR_INPUT**

```plaintext
bSwi       : BOOL;
bSwiUp     : BOOL;
bSwiDwn    : BOOL;
bSetVal    : BOOL;
bOn        : BOOL;
bOff       : BOOL;
nSwiOvrTi  : UDINT;
nSwiTi     : UDINT;
nDimTi     : UDINT;
bMemMod    : BOOL;
fOnVal     : REAL;
fSetVal    : REAL;
```

\(b\text{Swi}\): A rising edge at this input switches the light on and off alternately.

\(f\text{Out}\) assumes the value \(f\text{OnVal}\). If \(b\text{MemMod}= \text{TRUE}\), then \(f\text{Out}\) assumes the value that applied before the last switch-off.

A long keypress alternately dims the light up and down. If the dimming value reaches the maximum value of 100%, it remains there until the button is pressed again; there is no automatic change. The same applies to the minimum value at 0%.
bSwiUp: A rising edge at this input switches the light on.

\( fOut \) assumes the value \( fOnVal \). If \( bMemMod = \text{TRUE} \), then \( fOut \) assumes the value that applied before the last switch-off.

A long keypress increases the brightness. If the dimming value reaches the maximum value of 100 %, it remains there.

bSwiDwn: A rising edge at this input switches the light on.

\( fOut \) assumes the value \( fOnVal \). If \( bMemMod = \text{TRUE} \), then \( fOut \) assumes the value that applied before the last switch-off.

A long keypress decreases the brightness. If the dimming value reaches the minimum value of 0 %, it remains there.

bSetVal: A rising edge at this input switches the light value to \( fSetVal \).

bOn: A rising edge at this input switches \( fOut \) to the value \( fOnVal \). If \( bMemMod = \text{TRUE} \), then \( fOut \) assumes the value that applied before the last switch-off.

This input only switches on. If the light is already set to a value greater than zero, this input has no effect.

bOff: A rising edge at this input switches off the light value and the overall function of the function block.

nSwiOvrTi: Switching time [ms] for detecting a short or long keypress.

nSwiTi: dimming control ramp. Instead of setting the light directly to a specific value when switching it on or off, it is perceived as more agreeable if the light is controlled via a dimming ramp. The time [s] entered here is used to change the brightness from 0 to 100% and vice versa.

nDimTi: Dimming ramp for dimming up or down via a long keypress at \( bSwi, bSwiUp \) and \( bSwiDwn \). This time [s] is required to dim the light from 0 to 100% and vice versa.

bMemMod / fOnVal: If \( bMemMod = \text{FALSE} \), the output is switched to \( fOnVal \) when the light is switched on.

If \( bMemMod = \text{TRUE} \) (memory-mode), the last value is stored before the light is switched off and used as the resetting value.

fSetVal: With a rising edge at \( bSetVal \) this value is output at \( fOut \).

**VAR_OUTPUT**

\[
\begin{align*}
  fOut & : \text{REAL}; \\
  bLgt & : \text{BOOL};
\end{align*}
\]

**fOut**: Control output for light actuators (0...100%).

**bLgt**: Binary output for light control: \( fOut = 0 \Rightarrow bLgt = \text{FALSE} \), otherwise \( bLgt = \text{TRUE} \).

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
FB_BA_StairwellAnlg

Stairwell lighting control with analog output (0...100 %) and parameterizable switch-on and pre-warning value.

The function can be started via a rising edge at \(bSwi\). Either a push button or an occupancy sensor is connected to this input.

With a rising edge at \(bSwi\) the light control output \(fOut\) is initially set to the value \(fOnVal\). If the signal at \(bSwi\) drops again, the switch-on time of the light \(nOnTi\) [s] is counted down.

After the switch-on time has elapsed, the light is set to the prewarning value \(fPrewarnVal\) and a second timer \(nPrewarnTi\) [s] elapses.

When this phase is finished, the light is switched off.

The procedure can be restarted at any time by a rising edge at \(bSwi\).

An edge at \(bOff\) switches the light off, an edge at \(bOn\) sets the light level to the value \(fOnVal\).

The stairwell function can still be operated via \(bSwi\).

VAR_INPUT

\[\begin{align*}
\text{bSwi} & : \text{BOOL}; \\
\text{bOn} & : \text{BOOL}; \\
\text{bOff} & : \text{BOOL}; \\
\text{nOnTi} & : \text{UDINT}; \\
\text{nSwiTi} & : \text{UDINT}; \\
\text{nPrewarnTi} & : \text{UDINT}; \\
\text{fOnVal} & : \text{REAL}; \\
\text{fPrewarnVal} & : \text{REAL}; \\
\end{align*}\]

**bSwi**: A rising edge at this input switches the light on and off alternately.

**bOn**: A rising edge at this input switches \(fOut\) to the value \(fOnVal\).

**bOff**: A rising edge at this input switches off the light value and the overall function of the function block.

**nOnTi**: Switch-on time [s]. This timer is started after a falling edge at \(bSwi\). During the switch-on time, the light output value \(fOut\) is at \(fOnVal\).

**nPrewarnTi**: Prewarning time [s] for which the light remains at the value \(fPrewarnVal\) after the switch-on time has elapsed.

**fOnVal**: Switch-on value for the lighting (0...100 %).

**fPrewarnVal**: Dimming value of the prewarning time (0...100%).

VAR_OUTPUT

\[\begin{align*}
\text{fOut} & : \text{REAL}; \\
\text{bLgt} & : \text{BOOL}; \\
\end{align*}\]

**fOut**: Control output for light actuators (0...100%).

**bLgt**: Binary output for light control: \(fOut = 0 \rightarrow bLgt = \text{FALSE}, \text{otherwise } bLgt = \text{TRUE} \).
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

FB_BA_StairwellBin

Function block for stairwell lighting control with binary light output and prewarning flashing sequence.

The function can be started via a rising edge at bSwi. This input is intended for connecting either a push button or an occupancy sensor to it.

With a rising edge at bSwi, the light control output bLgt is initially set to TRUE. If the signal at bSwi drops again, the light-on time (nOnTi), in seconds, is counted down.

At the end of this phase, a flash sequence of off-on pulses begins, the number of which is defined by nPrewarnBlinks. Furthermore, the switch-on and switch-off times can be specified via the parameters nBlinkOnTi and nBlinkOffTi, both in milliseconds.

When the flashing sequence is finished, the light switches off (bLgt = FALSE).

This procedure can be restarted at any time by a rising edge at bSwi.

An edge at bOff switches the light off, an edge at bOn switches it on.

The stairwell function can still be operated via bSwi.

VAR_INPUT

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bSwi</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bOn</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bOff</td>
<td>BOOL;</td>
</tr>
<tr>
<td>nOnTi</td>
<td>UDINT;</td>
</tr>
<tr>
<td>nSwiTi</td>
<td>UDINT;</td>
</tr>
<tr>
<td>nPrewarnBlinks</td>
<td>UDINT;</td>
</tr>
<tr>
<td>nBlinkOffTi</td>
<td>UDINT;</td>
</tr>
<tr>
<td>nBlinkOnTi</td>
<td>UDINT;</td>
</tr>
</tbody>
</table>

bSwi: A rising edge at this input switches the light on. After a falling edge the switch-off delay starts.

bOn: A rising edge at this input switches the light on and resets the active timers.

bOff: A rising edge at this input switches the light off and resets the active timers.

nOnTi: Switch-on time [s]. This timer is started after a falling edge at bSwi. During the switch-on time, the light output value bLgt is TRUE.

nPrewarnBlinks: Number of off-on pulses of the prewarning flashing sequence.

nBlinkOffTi: Duration [ms] of the switch-off phase of a prewarning pulse.

nBlinkOnTi: Duration [ms] of the switch-on phase of a prewarning pulse.

VAR_OUTPUT

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bLgt</td>
</tr>
</tbody>
</table>

An edge at bSwi switches the light on. After a falling edge the switch-off delay starts.
bLgt: Binary output for light control.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

FB_BA_Swi

The function block is used to switch an actuator on or off.

A positive edge at the input bOn results in setting of output bQ. The output is reset by a positive edge at the bOff input. If a positive edge is presented to bToggle, the output is negated; i.e., if On it goes Off, and if Off it goes On.

VAR_INPUT

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bOn</td>
<td>BOOL</td>
</tr>
<tr>
<td>bOff</td>
<td>BOOL</td>
</tr>
<tr>
<td>bToggle</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

bOn: Switches the output on.
bOff: Switches the output off.
bToggle: Negates the current output state.

VAR_OUTPUT

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bLgt</td>
</tr>
</tbody>
</table>

bLgt: Switching output.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
Shading

Overview of shading correction

### meaning of colours
- **Blue**: effective for the whole building
- **Red**: effective for one facade
- **Green**: effective for a room/group of rooms

### outside-brightness with hysteresis and delay
- **FB_BA_BrtnsHysDly**
- **bOut**

### check if the facade is in sunlight (azimuth-angle)
- **FB_BA_InRngAzm**
- **bOut**

### check, if the height of the sun (elevation-angle) is in valid range
- **FB_BA_InRngElv**
- **bOut**

### input of shading objects
- **FB_BA_ShdojBentry**
- **FB_BA_RdShdObjLst**

### input of facade elements
- **FB_BA_FcElemEntry**
- **FB_BA_RdFcElemLst**

### shading-correction of a group of windows
- **FB_BA_ShdojCorr**
- **bGrpNotShdd**

### sun-protection
- **FB_BA_SunPrtc**
- **bEn**
- **positioning-telegram**

### Shading correction: Basic principles and definitions

The shading correction can be used in conjunction with the automatic sun function or slat adjustment. The function checks whether a window or a window group that is assigned to a room, for example, is temporarily placed in the shade by surrounding buildings or parts of its own building. Sun shading for windows that stand in the shadow of surrounding buildings or trees is not necessary and may even be disturbing under certain circumstances. On the basis of data entered regarding the facade and its surroundings, the shading correction...
correction determines which parts of the front are in the shade. Hence, it is then possible to decide whether the sun protection should be active for individual windows or window groups.

Apart from the current position of the sun, the shading of the individual windows depends on three things:

- the orientation of the facade
- the position of the windows
- the positioning of the shading objects

The following illustrations are intended to describe these interrelationships and to present the parameters to be entered.

**Orientation of the facade**

**Observation from above**

A two-dimensional coordinate system is required for observing the shadow cast on the facade, which is why the x- and y-axis were placed on the facade. The zero point is thereby at the bottom left on the base, as if one were regarding the facade from the front. For the calculation of the shading objects the Z component is then also added. Its axis points from away the facade and has the same zero point as the X and Y axis.

In the northern hemisphere, the horizontal sun position (azimuth angle) is determined from the north direction by definition. The facade orientation is likewise related to north, wherein the following applies to the line of sight from a window in the facade:

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

In the southern hemisphere is the sun path is reversed: Although it also rises in the east, at midday it is in the north. The facade orientation is adjusted to this path:

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

For convenience, the other explanations refer to the northern hemisphere. The calculations for the southern hemisphere are analogous. When the function block **FB_BA_ShdCorr** ([311](#)) (shading correction) is parameterized they are activated through a boolean input, `bSouth`.

The following two illustrations are intended to further clarify the position of the point of origin P₀ as well as the orientation of the coordinate system:
Observation from the side

The angle of elevation (height of the sun) can be represented using this illustration: by definition this is 0° at sunrise (horizontal incidence of light) and can reach maximally 90°, but this applies only to places within the Tropic of Cancer and the Tropic of Capricorn.

Observation from the front

Here, the position of the point of origin, $P_0$, at the bottom left base point of the facade is once more very clear. Beyond that the X-Y orientation is illustrated, which is important later for the entry of the window elements.

Position of the windows

The position of the windows is defined by the specification of their bottom left corner in relation to the facade coordinate system. Since a window lies flat on the facade, the entry is restricted to the X and Y coordinates.
In addition, the window width and the window height have to be specified.

The position of each window corner on the facade is determined internally from the values entered. A window is considered to be in the shade if all corners lie in the shade.

**Positioning of the shading objects**

When describing the shading objects, distinction is made between angular objects (building, column) and objects that are approximately spherical (e.g. trees). Angular objects can be subdivided into square shadow-casting facades according to their shadows, noting which cast the main shadow throughout the day:
In the morning and around noon, the shadow is mainly cast by the sides $S_1$ and $S_4$. $S_2$ and $S_3$ do not have to be considered, unless they are higher.
In the afternoon and evening, the total shade can be determined solely through $S_1$ and $S_2$. In this case it is therefore sufficient to specify $S_1$, $S_2$, and $S_3$ as shadow casters. The entry is made on the basis of the four corners or their coordinates in relation to the zero point of the facade:

In this sketch only the upper points, $P_2$ and $P_3$, are illustrated due to the plan view. The lower point $P_1$ lies underneath $P_2$ and $P_4$ lies underneath $P_3$. 
The input of shadow-casting ball elements is done by entering the center of the ball and its radius:

A "classification" of the ball element as in the case of the angular building is of course unnecessary, since the shadow cast by a ball changes only its direction, but not its size.
Overview of automatic sun protection

### Meaning of colors
- **Blue**: effective for the whole building
- **Red**: effective for one facade
- **Green**: effective for a room/group of rooms

#### Priority selection
- **FB_BA_SunBldPrioSwi**

#### Inputs
- **Prio1-input**
- **Prio2-input**
- **Prio3-input**
- **Prio4-input**
- **Prio5-input**
- **Prio6-input**
- **Prio7-input**
- **Prio8-input**

#### Outputs
- **sunblind-actuator**

#### Actions
- **Ice protection** (`FB_BA_SunBldIcePrtc`)
- **Positioning telegram** (`ST_BA_SunBld`)
- **Wind protection** (`FB_BA_SunBldWndPrtc`)
- **Maintenance position** (`FB_BA_SunBldEvt`)
- **Manual mode** (`FB_BA_SunBldSwi`)
- **Twilight automatic** (`FB_BA_SunBldTwLgtAuto`)
- **Thermo automatic** (`FB_BA_SunBldEvt`)
- **Sun protection** (`FB_BA_SunPrtc`)
- **Parking position** (`FB_BA_SunBldEvt`)

---

Version: 1.4
TF8040
Sun protection: Basic principles and definitions

The direct incidence of daylight is regarded as disturbing by persons in rooms. On the other hand, however, people perceive natural light to be more pleasant in comparison with artificial light. Two options for glare protection are to be presented here:

- Slat adjustment
- Height adjustment

Slat adjustment

A louvered blind that can be adjusted offers the option of intelligent sun protection here. The position of the slats is cyclically adapted to the current position of the sun, so that no direct daylight enters through the blinds, but as much diffuse daylight can be utilized as possible.

The illustration shows that diffuse light can still enter from underneath, whereas no further direct daylight, or theoretically only a single ray, can enter. The following parameters are necessary for the calculation of the slat angle:

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

Effective elevation angle

If the blind is viewed in section as above, the angle of incidence does not depend solely on the solar altitude (elevation), but also on the direction of the sun:

- If the facade orientation and the sun position (azimuth) are the same, i.e. the sunlight falls directly onto the facade, the effective light incidence angle is the same as the current elevation angle.
- However, if the sunlight falls at an angle onto the facade as seen from the sun direction, the effective angle is larger for the same angle of elevation.

This relationship can easily be illustrated with a set square positioned upright on the table: Viewed directly from the side you can see a triangle with two 45° angles and one 90° angle. If the triangle is rotated, the side on the table appears to become shorter and the two original 45° angles change. The triangle appears to be getting steeper.

We therefore refer to the "effective elevation angle", which describes the proportion of light that falls directly onto the blind.

The following three images illustrate the relationship between the effective elevation angle and the blind dimensions, and how the resulting slat angle $\lambda$ changes during the day:
louvre-angle

louvre at an angle of $\lambda=0$

Louvre-angle in the morning and in the evening

Louvre angle at noon

$\lambda = \text{louvre-angle, in this drawing: } \lambda < 0$

$\varepsilon_{\text{eff}} = \text{effective sun-elevation}$

$\lambda = \text{louvre-angle, in this drawing: } \lambda > 0$

$\varepsilon_{\text{eff}} = \text{effective sun-elevation}$
Height adjustment

With a high position of the sun at midday, the direct rays of sunlight do not penetrate into the full depth of the room. If direct rays of sunlight in the area of the window sill are regarded as uncritical, the height of the sun protection can be adapted automatically in such a way that the rays of sunlight only ever penetrate into the room up to an uncritical depth.

In order to be able to calculate at any time the appropriate blind height that guarantees that the incidence of sunlight does not exceed a certain value, the following values are necessary.

Required for the calculation of the respective blind height:

- Height of the sun (elevation)
- Window height
- Distance between the window and the floor

The following illustration shows where these parameters are to be classified:
Influence of the facade inclination

In both of the methods of sun protection described, it was assumed that the facade and thus the windows are perpendicular to the ground. In the case of an inclined facade, however, the incidence of light changes such that this influence will also be taken into account. The facade inclination is defined as follows:
facade angle: $\varphi = 0^\circ$

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

facade angle: $\varphi > 0^\circ$
List of shading elements

The data of all shading objects (building components, trees, etc.) per facade are stored in a field of structure elements of type `ST_BA_ShdObj` within the program.

The shading correction `FB_BA_ShdCorr` reads the information from this list. The management function block `FB_BA_ShdObjEntry` reads and writes it as input/output variable.

It is therefore advisable to declare this list globally:

```plaintext
VAR_GLOBAL
  aShdObj : ARRAY[1..BA_Param.nSunPrt_MaxShdObj] OF ST_BA_ShdObj;
END_VAR
```

The variable `nSunPrt_MaxShdObj` represents the upper limit of the available elements and is defined as a global constant within the program library:

```plaintext
VAR_GLOBAL CONSTANT
  nSunPrt_MaxShdObj : UINT := 20;
END_VAR
```

List of facade elements

The data of all windows (facade elements) per facade are saved within the program in a field of structure elements of the type `ST_BA_FcdElem`.

The management function block `FB_BA_FcdElemEntry` and the shading correction `FB_BA_ShdCorr` read and write to this list (the latter sets the shading information); they access this field as input/output variables.

It is therefore advisable to declare this list globally:

```plaintext
VAR_GLOBAL
  aFcdElem : ARRAY[1..BA_Param.nSunPrt_MaxRowFcd, 1..BA_Param.nSunPrt_MaxColumnFcd] OF ST_BA_FcdElem;
END_VAR
```

The variables `nSunPrt_MaxColumnFcd` and `nSunPrt_MaxRowFcd` define the upper limit of the available elements and are declared as global constants within the program library:

```plaintext
VAR_GLOBAL CONSTANT
  nSunPrt_MaxRowFcd : UINT := 10;
nSunPrt_MaxColumnFcd : UINT := 20;
END_VAR
```

**FB_BA_BldPosEntry**

This function block is used for entering interpolation points for the function block `FB_BA_SunPrtc`, if this function block is operated in height positioning mode with the aid of a table (see `E_BA_PosMod`).

In addition to the operation modes "Fixed blind height" and "Maximum incidence of light", the function block `FB_BA_SunPrtc` also offers the possibility to control the blind height in relation to the position of the sun by means of table entries. By entering several interpolation points, the blind height relative to the respective sun position is calculated by linear interpolation. However, since incorrectly entered values can
lead to malfunctions in FB_BA_SunPrtc [187], this function block is to be preceded by the function block FB_BA_BldPosEntry. Four interpolation points can be parameterized on this function block, whereby a missing entry is evaluated as a zero entry.
The function block does not sort the values entered independently, but instead ensures that the positions of the sun entered in the respective interpolation points are entered in ascending order. Unintentional erroneous entries are noticed faster as a result.
The values chosen for fSunElv1 .. fSunElv4 must be unique; for example, the following situation must be avoided:
[fSunElv1 = 10 ; fPos1 = 50 ] and simultaneously [fSunElv2 = 10 ; fPos2 = 30 ].
This would mean that there would be two different target values for one and the same value, which does not allow a unique functional correlation to be established.
On top of that the entries for the position of the sun and blind height must lie within the valid range.
Mathematically this means that the following conditions must be satisfied:

- fSunElv1 < fSunElv2 < fSunElv3 < fSunElv4 - (values ascending and not equal)
- 0 ≤ fSunElv ≤ 90 (° - scope source values)
- 0 ≤ fPos ≤ 100 (in percent - scope target values)

The function block checks the entered values for these conditions and issues an error message if they are not met. In addition, the control value bVld of ST_BA_BldPosTab [93] is set to FALSE.
Furthermore the function block independently ensures that the boundary areas are filled out: Internally, another interpolation point is set at fSunElv = 0 with fPos1 and another one above fSunElv4 at fSunElv = 90 with fPos4. This ensures that a meaningful target value is available for all valid input values 0 ≤ fSunElv ≤ 90, without the user having to enter fSunElv = 0 and fSunElv = 90:

This increases the actual number of interpolation points transferred to the function block FB_BA_SunPrtc [187] to 6; see ST_BA_BldPosTab [93].

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

VAR INPUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>fSunElv1</td>
<td>REAL</td>
</tr>
<tr>
<td>fPos1</td>
<td>REAL</td>
</tr>
<tr>
<td>fSunElv2</td>
<td>REAL</td>
</tr>
<tr>
<td>fPos2</td>
<td>REAL</td>
</tr>
<tr>
<td>fSunElv3</td>
<td>REAL</td>
</tr>
<tr>
<td>fPos3</td>
<td>REAL</td>
</tr>
<tr>
<td>fSunElv4</td>
<td>REAL</td>
</tr>
<tr>
<td>fPos4</td>
<td>REAL</td>
</tr>
</tbody>
</table>

fSunElv1: Position of the sun at the first interpolation point [°] (0...90).
fPos1: Blind position (degree of closure) at the first interpolation point [%] (0…100).
fSunElv2: Position of the sun at the second interpolation point [°] (0…90).
fPos2: Blind position (degree of closure) at the second interpolation point [%] (0…100).
fSunElv3: Position of the sun at the third interpolation point [°] (0…90).
fPos3: Blind position (degree of closure) at the third interpolation point [%] (0…100).
fSunElv4: Position of the sun at the fourth interpolation point [°] (0…90).
fPos4: Blind position (degree of closure) at the fourth interpolation point [%] (0…100).

VAR_OUTPUT

```plaintext
stBldPosTab : ST_BA_BldPosTab;
bErr : BOOL;
sErrDescr : T_MAXSTRING;
```

**stBldPosTab**: Transfer structure of the interpolation points (see ST_BA_BldPosTab [93]).

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

**sErrDescr**: Contains the error description.

**Error description**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Error: The x-values (elevation values) in the table are either not listed in ascending order, or they are duplicated.</td>
</tr>
<tr>
<td>02</td>
<td>Error: An elevation value that was entered is outside the valid range of 0°…90°.</td>
</tr>
<tr>
<td>03</td>
<td>Error: A position value that was entered is outside the valid range of 0%…100%.</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

**FB_BA_CalcSunPos**

Calculation of sun position based on the date, time, longitude and latitude.

The position of the sun for a given point in time can be calculated according to common methods with a defined accuracy. For applications with moderate requirements, the present function block is sufficient. As the basis for this, the SUNAE algorithm was used, which represents a favorable compromise between accuracy and computing effort.

The position of the sun at a fixed observation point is normally determined by specifying two angles. One angle indicates the height above the horizon; 0° means that the sun is in the horizontal plane of the location; a value of 90° means that the is perpendicular to the observer. The other angle indicates the direction at which the sun is positioned. The SUNAE algorithm is used to distinguish whether the observer is standing on the northern hemisphere (longitude > 0 degrees) or on the southern hemisphere (longitude < 0 degrees) of the earth. If the observation point is in the northern hemisphere, a value of 0° is assigned for the northern direction of the sun and then moves clockwise around the compass, i.e. 90° is east, 180° is south, 270° west, etc. If the observation point is in the southern hemisphere, 0° corresponds to the southern direction and moves counterclockwise, i.e. 90° is east, 180° is north, 270° is west, etc.
The time has to be specified as coordinated world time (UTC, Universal Time Coordinated, previously referred to as GMT, Greenwich Mean Time).

The latitude is the northerly or southerly distance of a location on the Earth’s surface from the equator, in degrees [°]. The latitude can assume values between 0° (at the equator) and ±90° (at the poles). A positive sign thereby indicates a northern direction and a negative sign a southern direction. The longitude is an angle that can assume values up to ±180° starting from the prime meridian 0° (an artificially determined North-South line). A positive sign indicates a longitude in an eastern direction and a negative sign in a western direction. Examples:

<table>
<thead>
<tr>
<th>Location</th>
<th>Longitude</th>
<th>Latitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sydney, Australia</td>
<td>151.2°</td>
<td>-33.9°</td>
</tr>
<tr>
<td>New York, USA</td>
<td>-74.0°</td>
<td>40.7°</td>
</tr>
<tr>
<td>London, England</td>
<td>-0.1°</td>
<td>51.5°</td>
</tr>
<tr>
<td>Moscow, Russia</td>
<td>37.6°</td>
<td>55.7°</td>
</tr>
<tr>
<td>Beijing, China</td>
<td>116.3°</td>
<td>39.9°</td>
</tr>
<tr>
<td>Dubai, United Arab Emirates</td>
<td>55.3°</td>
<td>25.4°</td>
</tr>
<tr>
<td>Rio de Janeiro, Brazil</td>
<td>-43.2°</td>
<td>-22.9°</td>
</tr>
<tr>
<td>Hawaii, USA</td>
<td>-155.8°</td>
<td>20.2°</td>
</tr>
<tr>
<td>Verl, Germany</td>
<td>8.5°</td>
<td>51.9°</td>
</tr>
</tbody>
</table>

If the function block `FB_BA_CalcSunpos` returns a negative value for the solar altitude `fSunElv`, the sun is invisible. This can be used to determine sunrise and sunset.

**VAR_INPUT**

- `fDegLndg`: Longitude [°].
- `fDegLatd`: Latitude [°].
- `tUTC`: Input of the current time as coordinated world time (see `TIMESTRUCT`). The function block `FB_BA_GetTime` can be used to read this time from a target system.

**VAR_OUTPUT**

- `fSunAzm`: REAL;
- `fSunElv`: REAL;
fSunAzm: Direction of the sun (northern hemisphere: 0° north ... 90° east ... 180° south ... 270° west ... / southern hemisphere: 0° south ... 90° east ... 180° north ... 270° west ...).

fSunElv: Height of the sun (0° horizontal ... 90° vertical).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

FB_BA_FcdElemEntry

This function block serves the administration of all facade elements (windows) in a facade, which are saved globally in a list of facade elements [142]. It is intended to facilitate inputting element information - not least with regard to using the TC3 PLC HMI. A schematic illustration of the objects with description of the coordinates is given in Shading correction: principles and definitions [129].

The facade elements are declared in the global variables as a two-dimensional field above the window columns and rows:

```plaintext
VAR_GLOBAL
  aFcdElem : ARRAY[1..Param.nSunPrt_MaxColumnFcd, 1..Param.nSunPrt_MaxRowFcd] OF ST_BA_FcdElem;
END_VAR
```

Each element `arrFcdElem[x,y]` contains the information for an individual facade element (`ST_BA_FcdElem [94]`). The information includes the group membership, the dimensions (width, height) and the coordinates of the corners. The function block thereby accesses this field directly via the IN-OUT variable `aFcdElem`.

Note: The fact that the coordinates of corners C2 to C4 are output values arises from the fact that they are formed from the input parameters and are to be available for use in a visualization:
Programming

All entries in [m]!

\[ \begin{align*}
  f\text{Cnr2}X &= f\text{Cnr1}X \\
  f\text{Cnr2}Y &= f\text{Cnr1}Y + f\text{WdwHght} \text{ (window height)} \\
  f\text{Cnr3}X &= f\text{Cnr1}X + f\text{WdwWdth} \text{ (window width)} \\
  f\text{Cnr3}Y &= f\text{Cnr2}Y \\
  f\text{Cnr4}X &= f\text{Cnr1}X + f\text{WdwWdth} \text{ (window width)} \\
  f\text{Cnr4}Y &= f\text{Cnr1}Y
\end{align*} \]

The function block is used in three steps:

- Read
- Change
- Write

Read

With the entries at \( n\text{Column} \) and \( n\text{Row} \) the corresponding element is selected from the list, \( a\text{FcdElem}[n\text{Column}, n\text{Row}] \). A rising edge on \( b\text{Rd} \) reads the following data from the list element:

- \( n\text{Grp} \) group membership,
- \( f\text{Cnr1}X \) x-coordinate of corner point 1 [m]
- \( f\text{Cnr1}Y \) y-coordinate of corner point 1 [m]
- \( f\text{WdwWdth} \) window width [m]
- \( f\text{WdwHght} \) window height [m]

These are then assigned to the corresponding input variables of the function block, which uses them to calculate the coordinates of corners C2-C4 as output variables in accordance with the correlation described above. It is important here that the input values are not overwritten in the reading step. Hence, all values can initially be displayed in a visualization.
Change

In a next program step the listed input values can then be changed. The values entered are constantly checked for plausibility. The output bErr indicates whether the values are valid (bErr=FALSE). If the values are invalid, a corresponding error message is issued at output sErrDescr. See also "Error (bErr=TRUE)" below.

Write

With a positive edge at bWrt the parameterized data are written into the field of the array aFcdElem dependent on nRow and nColumn, regardless of whether they represent valid values or not. The element structure ST_BA_FcdElem [94] therefore also contains a plausibility bit bVld, which forwards precisely this information to the function block FB_BA_ShdCorr [165] to prevent miscalculations.

This approach is to be regarded only as a proposal. It is naturally also possible to parameterize the function block quite normally in one step and to write the values entered to the corresponding list element with a rising edge on bWrt.

Error (bErr=TRUE)

The function block FB_BA_ShdCorr [165], which judges whether all windows in a group are shaded, will only perform its task if all windows in the examined group have valid entries. This means:

- nGrp must be greater than 0
- fCnr1X must greater than or equal to 0.0
- fCnr1Y must greater than or equal to 0.0
- fWdwWdth must be greater than 0
- fWdwHght must be greater than 0

If one of these criteria is not met, it is interpreted as incorrect input, and the error output bErr is set at the function block output of FB_BA_FcdElemEntry. Within the window element ST_BA_FcdElem [94], the plausibility bit bVld is set to FALSE.

If on the other hand all entries of a facade element are zero, it is regarded as a valid, deliberately omitted facade element:
In the case of a facade of 6x4 windows, the elements window (2.1), window (3.5) and window (4.4) would be empty elements here.

**VAR_INPUT**

```plaintext
nColumn   : UDINT;
nRow      : UDINT;
bWrt      : BOOL;
bRd       : BOOL;
nGrp      : UDINT;
fCnr1X     : REAL;
fCnr1Y     : REAL;
fWdwWdth   : REAL;
fWdwHght   : REAL;
```

- **nColumn**: Column index of the selected component on the facade. This refers to the selection of a field element of the array stored in the IN-OUT variable `aFcdElem`.

- **nRow**: ditto. Row index. *nRow and nColumn must not be zero!* This is due to the field definition, which always starts with 1; see above.
bRd: A positive edge at this input causes the information of the selected element, $aFcdElem[nColumn, nRow]$, to be read into the function block and assigned to the input variables $nGrp$ to $fWdwHght$. The resulting output variables are $fCnr2X$ to $fCnr4Y$. If data are already present on the inputs $nGrp$ to $fWdwHght$ at time of reading, then the data previously read are immediately overwritten with these data.

bWrt: A positive edge writes both the entered values and the calculated values into the selected field element $aFcdElem[nColumn, nRow]$.

$nGrp$: Group membership. Internally limited to a minimum value of 0.

$fCnr1X$: X-coordinate of corner point 1 [m].

$fCnr1Y$: Y-coordinate of corner point 1 [m].

$fWdwWdth$: Window width [m].

$fWdwHght$: Window height [m].

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>fCnr2X : REAL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>fCnr2Y : REAL;</td>
</tr>
<tr>
<td>fCnr3X : REAL;</td>
</tr>
<tr>
<td>fCnr3Y : REAL;</td>
</tr>
<tr>
<td>fCnr4X : REAL;</td>
</tr>
<tr>
<td>fCnr4Y : REAL;</td>
</tr>
<tr>
<td>bErr : BOOL;</td>
</tr>
<tr>
<td>sErrDesc : T_MAXSTRING;</td>
</tr>
</tbody>
</table>

$fCnr2X$: x-coordinate determined for corner point 2 of the window [m] (see function description [146]).

$fCnr2Y$: y-coordinate determined for corner point 2 of the window [m] (see function description [146]).

$fCnr3X$: x-coordinate determined for corner point 3 of the window [m] (see function description [146]).

$fCnr3Y$: y-coordinate determined for corner point 3 of the window [m] (see function description [146]).

$fCnr4X$: x-coordinate determined for corner point 4 of the window [m] (see function description [146]).

$fCnr4Y$: y-coordinate determined for corner point 4 of the window [m] (see function description [146]).

$bErr$: Result verification for the entered values.

$sErrDesc$: Contains the error description.

**Error description**

<table>
<thead>
<tr>
<th>Error number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Error: index error! nColumn and/or nRow are outside the permissible limits 1.. nSunPrt_MaxColumnFcd or 1.. nSunPrt_MaxRowFcd. See list of facade elements.</td>
</tr>
<tr>
<td>02</td>
<td>Error: The group index is 0, but at the same time another entry of the facade element is not zero. Only if all entries of a facade element are zero is it considered to be a valid, deliberately omitted facade component, otherwise it is interpreted as an incorrect entry. NOTE: Group entries less than zero are internally limited to zero.</td>
</tr>
<tr>
<td>03</td>
<td>Error: The X-component of the first corner point (Corner1) is less than zero.</td>
</tr>
<tr>
<td>04</td>
<td>Error: The Y-component of the first corner point (Corner1) is less than zero.</td>
</tr>
<tr>
<td>05</td>
<td>Error: The window width is less than or equal to zero.</td>
</tr>
<tr>
<td>06</td>
<td>Error: The window height is less than or equal to zero.</td>
</tr>
</tbody>
</table>

**VAR_IN_OUT**

$aFcdElem$ : ARRAY[1..Param.nSunPrt_MaxColumnFcd, 1..Param.nSunPrt_MaxRowFcd] OF ST_BA_FcdElem;

$aFcdElem$: List of facade elements (see list of facade elements [142]).
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

**FB_BA_InRngAzm**

This function block checks whether the current azimuth angle (horizontal position of the sun) lies within the limits entered. As can be seen in the overview [136], the function block provides an additionally evaluation as to whether the sun shading of a window group should be activated. Therefore the observations in the remainder of the text always apply to one window group.

The sun incidence azimuth angle on a smooth facade will always be facade orientation-90°...facade orientation+90°.

If the facade has lateral projections, however, this range is limited. This limitation can be checked with the help of this function block. However, the position of the window group on the facade also plays a role. If it lies centrally, this gives rise to the following situation (the values are only examples):
The values change for a group at the edge:

The start of the range $fSttRng$ may be greater than the end $fEndRng$, in which case values beyond 0° are considered:

Example

<table>
<thead>
<tr>
<th>$fAzm$</th>
<th>10.0°</th>
</tr>
</thead>
<tbody>
<tr>
<td>$fSttRng$</td>
<td>280.0°</td>
</tr>
<tr>
<td>$fEndRng$</td>
<td>20.0°</td>
</tr>
<tr>
<td>$bOut$</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

However, the range regarded may not be greater than 180° or equal to 0° – this would be unrealistic. Such entries result in an error on the output $bErr$ – the test output $bOut$ is then additionally set to FALSE.

**VAR_INPUT**

- $fAzm$: Current azimuth angle.
- $fSttRng$: Start of range [°].
- $fEndRng$: End of range [°].

**VAR_OUTPUT**

- $bOut$: The facade element is in the sun if the output is TRUE.
- $bErr$: This output is switched to TRUE if the parameters entered are erroneous.
- $sErrDescr$: Contains the error description.

**Error description**

01: Error: $fSttRng$ or $fEndRng$ less than 0° or greater than 360°.

02: Error: The difference between $fSttRng$ and $fEndRng$ is greater than 180°. This range is too large for analyzing the insolation on a facade.
**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

**FB_BA_InRngElv**

This function block checks whether the current angle of elevation (vertical position of the sun) lies within the limits entered. As can be seen in the overview [136], the function block provides an additionally evaluation as to whether the sun shading of a window group should be activated. Therefore the observations in the remainder of the text always apply to one window group.

A normal vertical facade is irradiated by the sun at an angle of elevation of 0° to maximally 90°.

If the facade has projections, however, this range is limited. This limitation can be checked with the help of this function block. However, the position of the window group on the facade also plays a role. If it lies in the lower range, this gives rise to the following situation (the values are only examples):
The values change for a group below the projection:
The lower observation limit, \( f_{LoLmt} \), may thereby not be greater than or equal to the upper limit, \( f_{HiLmt} \). Such entries result in an error on the output \( b_{Err} \) – the test output \( b_{Out} \) is then additionally set to FALSE.

**VAR_INPUT**

- \( fElv \) : REAL;
- \( fLoLmt \) : REAL;
- \( fHiLmt \) : REAL;

\( fElv \): Current elevation angle ['°].

\( fLoLmt \): Lower limit value ['°].

\( fHiLmt \): Upper limit value ['°].

**VAR_OUTPUT**

- \( b_{Out} \) : BOOL;
- \( b_{Err} \) : BOOL;
- \( s_{ErrDescr} \) : T_MAXSTRING;

\( b_{Out} \): The facade element is in the sun

\( b_{Err} \): This output is switched to TRUE if the parameters entered are erroneous.

\( s_{ErrDescr} \): Contains the error description.

**Error description**

01: Error: \( f_{HiLmt} \) less than or equal to \( f_{LoLmt} \).

02: Error: \( f_{LoLmt} \) is less than 0° or \( f_{HiLmt} \) is greater than 90°.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

**FB_BA_RdFcdElemLst**

With the help of this function block, data for facade elements (windows) can be imported from a pre-defined Excel table in csv format into the List of facade elements [142]. In addition the imported data are checked for plausibility and errors are written to a log file.

The following example shows the Excel table with the entries of the window elements. All text fields are freely writable. Important are the fields marked in green. Each row there identifies a data set.

The following rules are to be observed:

- A data set must always start with a '@'.
- The indices \( IndexColumn \) and \( IndexRow \) must lie within the defined limits (see List of facade elements [142]). These indices directly describe the facade element in the list \( a_{FcdElem} \) to which the data from the set are saved.
- Window width and window height must be greater than zero
- The corner coordinates \( P1x \) and \( P1y \) must be greater than or equal to zero.
- Each window element must be assigned to a group 1...255.
- For system-related reasons the total size of the table may not exceed 65534 bytes.
- This must have been saved in Excel as file type "CSV (comma-separated values) (*.csv)".
It is not necessary to describe all window elements that would be possible by definition or declaration. Before the new list is read in, the function block deletes the entire old list in the program. All elements that are not described by entries in the Excel table then have pure zero entries and are thus marked as non-existent and also non-evaluable, since the function block for shading correction, \texttt{FB\_BA\_ShdCorr}\footnote{165}, does not accept elements with the group entry ‘0’.

### Log file

Each time the reading function block is restarted, the log file is rewritten and the old contents are deleted. If there is no log file, it will be automatically created first. The log file then contains either an OK message or a list of all errors that have occurred. Errors connected with the opening, writing or closing of the log file itself.
cannot be written at the same time. Therefore, always note the output sErrDescr of the reading function block that indicates the last error code. Since the log file is always closed last during the reading process, a corresponding alarm is ensured in the event of an error.

**Program sample**

```plaintext
PROGRAM ReadFacadeElements
VAR
  blInit : BOOL;
  rRead : R_TRIG;
  fbReadFacadeElementList : FB_BA_RdFacadeElementList;
  aFacadeElement : ARRAY [...] OF BA_Param.nFunStrMaxColumnFod;
  bStt : BOOL;
  nAmountOfRead : UINT;
  bErr : BOOL;
  sErrDescr : T_MaxString;
  bBusy : BOOL;
  bErrDataSet : BOOL;
  tNetID = "local";
  aFcdElem : T_FacadeElement;
END_VAR

In this sample the variable blInit is initially set to TRUE when the PLC starts. Hence, the input bStt on the function block fbReadFacadeElementList receives a once-only rising edge that triggers the reading process. The file FacadeElements.csv is read, which is located in the folder C:\Projekte. The log file "Logfile.txt" is then saved in the same folder. If this log file does not yet exist it will be created, otherwise the existing contents are overwritten. Reading and writing take place on the same computer on which the PLC is located. This is defined by the input tNetID = "local". All data are written to the list aFcdElem declared in the program. The output bBusy is set to TRUE during reading and writing. The last file handling error that occurred is displayed at sErrDescr; bErr is then TRUE. If an error is detected in the data set, this is displayed at bErrDataSet and described in more detail in the log file. The number of found and read data rows is displayed at nAmtSetsRd for verification purposes.

The errors marked were "built into" the following Excel list. This gives rise to the log file shown:
The first error is in data set 2 and is an index error, since "0" is not permitted.

The next error in data set 6 was found after validation of the data with the internally used function block `FB_BA_ShDObjEntry` and allocated an error description. The third and the fourth errors likewise occurred after the internal validation.

Important here it that the data set numbers (in this case 22 and 24) do not go by the numbers entered in the list, but by the actual sequential numbers: only 30 data sets were read in here.

### VAR_INPUT

<table>
<thead>
<tr>
<th>bStt</th>
<th>sDataFile</th>
<th>sLogFile</th>
<th>tNetId</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>STRING</td>
<td>STRING</td>
<td>T_AmsNetId</td>
</tr>
</tbody>
</table>

**bStt**: A TRUE edge on this input starts the reading process.

**sDataFile**: Contains the path and file name for the data file to be opened. This must have been saved in Excel as file type "CSV (comma-separated values) (*.csv)". If the file is opened with a simple text editor, the values must be separated by semicolons. Example of an entry: `sDataFile := 'C:\Projekte\FacadeElements.csv'`

**sLogFile**: ditto. Log file for the accumulating errors. This file is overwritten each time the function block is activated, so that only current errors are contained.
**tNetId**: A string can be entered here with the AMS Net ID of the TwinCAT computer on which the files are to be written/read. An empty string can be specified for the local computer (see T_AmsNetId).

The data can be saved only on the control computer itself and on the computers that are connected by ADS to the control computer. Links to local hard disks in this computer are possible, but not to connected network hard drives.

---

**VAR_OUTPUT**

*bBusy*: This output is TRUE as long as elements are being read from the file.

*nAmtSetsRd*: Number of data sets read.

*bErr*: This output is switched to TRUE, if a file write or read error has occurred.

*sErrMsg*: Contains the error description.

<table>
<thead>
<tr>
<th>Error description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01: File handling error: Opening the log file - the ADS error number is stated.</td>
</tr>
<tr>
<td>02: File handling error: Open the data file - the ADS error number is stated.</td>
</tr>
<tr>
<td>03: File handling error: Reading the data file - the ADS error number is stated.</td>
</tr>
<tr>
<td>04: Error: During reading of the data file it was determined that the file is too large (number of bytes larger than nMaxDataFileSize)</td>
</tr>
<tr>
<td>05: File handling error: Writing to the log file - the ADS error number is stated.</td>
</tr>
<tr>
<td>06: File handling error: Closing the data file - the ADS error number is stated.</td>
</tr>
<tr>
<td>07: File handling error: Writing to the log file (OK message if no errors were detected) - the ADS error number is stated.</td>
</tr>
<tr>
<td>08: File handling error: Closing the log file - the ADS error number is stated.</td>
</tr>
</tbody>
</table>

*bErrDataSet*: This output is set to TRUE, if the read data sets are faulty. Further details are entered in the log file.

**VAR_IN_OUT**

*aFcdElem*: List of facade elements [142].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

**FB BA RdShdObjLst**

With the help of this function block, data for shading objects can be imported from a pre-defined Excel table in csv format into the list of shading objects [142]. In addition the imported data are checked for plausibility and errors are written to a log file.
The following example shows the Excel table with the entries of the window elements. All text fields are freely writable. The fields marked in green are important; each line in these fields identifies a data set. The columns G to J have a different meaning depending on whether the type rectangle or ball is concerned. The columns K to M are to be left empty in the case of balls. With regard to the rectangle coordinates, only the relevant data are entered and the remainder are internally calculated (see FB_BA_ShObjEntry [168]).

The following rules are to be observed:

- A data set must always start with a '@'.
- The month entries must not be 0 and not be greater than 12, all other combinations are possible.
- Examples:
  - Start=1, End=1: Shading in January.
  - Start=1, End=5: Shading from the beginning of January to the end of May.
  - Start=11, End=5: Shading from the beginning of November to the end of May (of the following year).
- Window width and window height must be greater than zero.
- The z-coordinates P1z and P3z or Mz must be greater than zero.
- The radius must be greater than zero.
- For system-related reasons the total size of the table may not exceed 65534 bytes.
- This must have been saved in Excel as file type "CSV (comma-separated values) (*.csv)".

It is not necessary to describe all shading objects that are possible per facade. Only those contained in the list ultimately take effect.

### Log file

Each time the reading function block is restarted, the log file is rewritten and the old contents are deleted. If there is no log file, it will be automatically created first. The log file then contains either an OK message or a list of all errors that have occurred. Errors connected with the opening, writing or closing of the log file itself cannot be written at the same time. Therefore, always note the output sErrDescr of the reading function block that indicates the last error code. Since the log file is always closed last during the reading process, a corresponding alarm is ensured in the event of an error.
Program sample

```plaintext
PROGRAM ReadShadingObjects
VAR
    bInit      : BOOL;
    rtRead     : R_TRIG;
    fbReadShadingObjects : FB_BA_RdShdObj;
    aShadingObject : ARRAY [1..BA_Param.nSunPrt_MaxShdObj] OF ST_BA_ShdObj;
    bBusy      : BOOL;
    nAmountOfSetsRead : UDINT;
    bError     : BOOL;
    sErrorDescr : T_MaxString;
    bErrDataSet : BOOL;
END_VAR
```

In this sample the variable `bInit` is initially set to TRUE when the PLC starts. Hence, the input `bStt` on the function block `fbReadShadingObjects` receives a once-only rising edge that triggers the reading process. The file `ShadingObjects.csv` is read, which is located in the folder `C:\Projekte\`. The log file `Logfile.txt` is then saved in the same folder. If this log file does not yet exist it will be created, otherwise the existing contents are overwritten. Reading and writing take place on the same computer on which the PLC is located. This is defined by the input `tNetID = " (=local). All data are written to the list `aShdObj` declared in the program. The output `bBusy` is set to TRUE during reading and writing. The last file handling error that occurred is displayed at `sErrDescr`; `bErr` is then TRUE. If an error is detected in the data set, this is displayed at `bErrDataSet` and described in more detail in the log file. The number of found and read data rows is displayed at `nAmtSetsRd` for verification purposes.

The errors marked were built into the following Excel list. This gives rise to the log file shown:
The first error is in data set 3 and is a type error, since "2" is not defined. The next error in data set 6 was found after validation of the data with the internally used function block FB_BA_ShdObjEntry [168] and allocated an error description. The third error likewise occurred after the internal validation.

Important here it that the data set number (in this case 11) does not go by the numbers entered in the list, but by the actual sequential number: only 16 data sets were read in here.

VAR_INPUT:

bStt : BOOL;
sDataFile : STRING;
sLogFile : STRING;
tNetId : T_AmsNetId;

bStt: A TRUE edge on this input starts the reading process.

dsDataFile: Contains the path and file name for the data file to be opened. This must have been saved in Excel as file type "CSV (comma-separated values) (*.csv)". If the file is opened with a simple text editor, the values must be separated by semicolons. Example of an entry: sDataFile := 'C:\Projects\ShadingObjects.csv'

sLogFile: Ditto. Log file for the accumulating errors. This file is overwritten each time the function block is activated, so that only current errors are contained.

tNetId: A string can be entered here with the AMS Net ID of the TwinCAT computer on which the files are to be written/read. An empty string can be specified for the local computer (see T_AmsNetId).

The data can be saved only on the control computer itself and on the computers that are connected by ADS to the control computer. Links to local hard disks in this computer are possible, but not to connected network hard drives.

VAR_OUTPUT:

bBusy : BOOL;
numSetsRead : UDINT;
bErr : BOOL;
sErrDescr : T_MAXSTRING;
bErrDataSet : BOOL;
bBusy: This output is TRUE as long as elements are being read from the file.

nAmtSetsRd: Number of data sets read.

bErr: This output is switched to TRUE, if a file write or read error has occurred.

sErrDescr: Contains the error description.

**Error description**

<table>
<thead>
<tr>
<th>Error number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>File handling error: Opening the log file - the ADS error number is stated.</td>
</tr>
<tr>
<td>02</td>
<td>File handling error: Open the data file - the ADS error number is stated.</td>
</tr>
<tr>
<td>03</td>
<td>File handling error: Reading the data file - the ADS error number is stated.</td>
</tr>
<tr>
<td>04</td>
<td>Error: During reading of the data file it was determined that the file is too large (number of bytes larger than nMaxDataFileSize)</td>
</tr>
<tr>
<td>05</td>
<td>File handling error: Writing to the log file - the ADS error number is stated.</td>
</tr>
<tr>
<td>06</td>
<td>File handling error: Closing the data file - the ADS error number is stated.</td>
</tr>
<tr>
<td>07</td>
<td>File handling error: Writing to the log file (OK message if no errors were detected) - the ADS error number is stated.</td>
</tr>
<tr>
<td>08</td>
<td>File handling error: Closing the log file - the ADS error number is stated.</td>
</tr>
</tbody>
</table>

bErrDataSet: This output is set to TRUE, if the read data sets are faulty. Further details are entered in the log file.

**VAR_IN_OUT**

aShdObj: ARRAY[1..BA_Param.nSunPrt_MaxShdObj] OF ST_BA_Shdobj;

aShdObj: List of shading objects [142].

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

**FB_BA_RolBldActr**

This function block is used to position a roller shutter over two outputs: up and down. The positioning telegram stSunBld [94] can be used to move the roller shutter to any position. In addition, the positioning telegram stSunBld [94] offers manual commands, which can be used to move the roller shutter to particular positions. These manual commands are controlled by the function block FB_BA_SunBldSwi [182].

The current height position is not read in by an additional encoder; it is determined internally by the runtime of the roller shutter. The two runtime parameters nTiUp (roller shutter travel-up time [ms]) and nTiDwn (roller shutter travel-down time [ms]) take account of the different movement characteristics.
As a rule, the function block controls the roller shutter based on the information from the positioning telegram $stSunBld$. If automatic mode is active ($bManMod = FALSE$), the roller shutter always moves to the current position; changes are reflected immediately. In manual mode ($bManMod = TRUE$), the roller shutter is controlled by the commands $bManUp$ and $bManDwn$.

**Referencing**

Safe referencing refers to a situation when the roller shutter is upwards-controlled for longer than its complete travel-up time. The position is then always "0". Since roller shutter positioning without encoder is always error-prone, it is important to use automatic referencing whenever possible: Whenever "0" is specified as the target position, the roller shutter initially moves upwards normally, based on continuous position calculation. Once the calculated position value 0% is reached, the output $bUp$ continues to be held for the complete travel-up time + 5s.

For reasons of flexibility there are now two possibilities to interrupt the referencing procedure: Until the calculated 0% position is reached, a change in position continues to be assumed and executed. Once this 0% position is reached, the roller shutter can still be moved with the manual "travel-down" command. These two sensible restrictions make it necessary for the user to ensure that the roller shutter is referenced safely whenever possible.

After a system restart, the function block executes a reference run. Completion of the initial referencing is indicated through a TRUE signal at output $bInitRefCmpl$. The initial referencing can also be terminated through a manual "travel-down" command.

**VAR_INPUT**

- $bEn$: Enable input for the function block. As long as this input is TRUE, the actuator function block accepts and executes commands as described above. A FALSE signal on this input resets the control outputs $bUp$ and $bDwn$ and the function block remains in a state of rest.
- $stSunBld$: Positioning telegram (see $ST\_BA\_Sunblind$).
- $nTiUp$: Complete time for driving up [ms].
- $nTiDwn$: Complete time for driving down [ms].

**VAR_OUTPUT**

- $bUp$: Roller shutter control output up.
- $bDwn$: Roller shutter control output down.
- $fActlPos$: Current position in percent.
- $bRef$: The roller shutter is in referencing mode, i.e. the output $bUp$ is set for the complete travel-up time + 5s. Only a manual "down" command can move the roller shutter in the opposite direction and terminate this mode.
- $nRefTi$: Referencing countdown display [s].
- $bInitRefCompl$: Initial referencing process complete.
- $bBusy$: A positioning or a referencing procedure is in progress.
- $bErr$: This output is switched to TRUE if the parameters entered are erroneous.
sErrDescr: Contains the error description.

<table>
<thead>
<tr>
<th>Error description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01: Error: The total travel-up or travel-down time (nTiUp / nTiDwn) is zero.</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

**FB_BA_ShdCorr**

The function block is used to assess the shading of a group of windows on a facade.

The function block *FB_BA_ShdCorr* calculates whether a window group lies in the shadow of surrounding objects. The result, which is output at the output *bGrpNotShdd*, can be used to judge whether sun shading makes sense for this window group.

The function block thereby accesses two lists, which are to be defined:

- The parameters that describe the shading elements that are relevant to the facade on which the window group is located. This list of shading objects [142] is used as input variable *aShdObj* for the function block, since the information is read only.

- The data of the elements (window) of the facade in which the group to be regarded is located. This list of facade elements [142] is accessed via the IN/OUT variable *aFcdElem*, since not only the window coordinates are read, but the function block *FB_BA_ShdCorr* also stores the shading information for each window corner in this list. In this way, the information can also be used in other parts of the application program.

On the basis of the facade orientation (*fFcdOrtn*), the direction of the sun (*fAzm*) and the height of the sun (*fElv*), a calculation can be performed for each corner of a window to check whether this lies in a shaded area. A window group is considered to be completely shaded if all corners are shaded.

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

<table>
<thead>
<tr>
<th>Line of sight</th>
<th>Facade orientation</th>
</tr>
</thead>
<tbody>
<tr>
<td>North</td>
<td>$\beta=0^\circ$</td>
</tr>
<tr>
<td>East</td>
<td>$\beta=90^\circ$</td>
</tr>
<tr>
<td>South</td>
<td>$\beta=180^\circ$</td>
</tr>
<tr>
<td>West</td>
<td>$\beta=270^\circ$</td>
</tr>
</tbody>
</table>

The function block performs its calculations only if the sun is actually shining on the facade. Considering the drawing presented in the introduction, this is the case if:

Facade orientation $<$ azimuth angle $<$ facade orientation $+$ 180°
In addition, a calculation is also not required, if the sun has not yet risen, i.e. the solar elevation is below 0°. In both cases the output $bFcdSunlit$ is set to FALSE.

The situation is different for the southern hemisphere. The following applies to the facade orientation (looking out the window):

<table>
<thead>
<tr>
<th>Line of sight</th>
<th>Facade orientation</th>
</tr>
</thead>
<tbody>
<tr>
<td>South</td>
<td>$\beta=0^\circ$</td>
</tr>
<tr>
<td>East</td>
<td>$\beta=90^\circ$</td>
</tr>
<tr>
<td>North</td>
<td>$\beta=180^\circ$</td>
</tr>
<tr>
<td>West</td>
<td>$\beta=270^\circ$</td>
</tr>
</tbody>
</table>

The internal calculation or the relationship between facade and sunbeam also changes:
To distinguish between the situation in the northern and southern hemisphere, set the input parameter `bSouth` to FALSE (northern hemisphere) or TRUE (southern hemisphere)

**VAR_INPUT**

- `tTiAct1` : TIMESTRUCT;
- `fFcdOrtn` : REAL;
- `fAzm` : REAL;
- `fElv` : REAL;
- `nGrpID` : DINT;
- `bSouth` : BOOL;
- `aShdObj` : ARRAY[1..BA_Param.nSunPrt_MaxShdObj] OF ST_BA_ShdObj;

`tTiAct1`: input of the current time - local time in this case, since this time takes into account the shaded months. If the UTC time (or GMT) is used, the month may change in the middle of the day, depending on the location on the earth (see TIMESTRUCT).

`fFcdOrtn`: facade orientation, see illustration above.

`fAzm`: direction of the sun at the time of observation [°].

`fElv`: height of the sun (elevation) at the time of observation [°].

`nGrpID`: window group regarded. The group 0 is reserved here for unused window elements (see FB_BA_FcdElemEntry [146]). A 0-entry would lead to an error output (bErr=TRUE). The function block is then not executed any further and `bGrpNotShdd` is set to FALSE.

`bSouth`: FALSE: calculations refer to conditions in the northern hemisphere - TRUE: in the southern hemisphere

`aShdObj`: list of shading objects [142].

**VAR_OUTPUT**

- `bGrpNotShdd` : BOOL;
- `bFcdSunlit` : BOOL;
- `bErr` : BOOL;
- `sErrDescr` : T_MAXSTRING

`bGrpNotShdd`: is TRUE as long as the window group is not calculated as shaded.

`bFcdSunlit`: this output is set to TRUE when the sun shines on the facade (see description above).
bErr: this output is switched to TRUE if the parameters entered are erroneous.

sErrMsg: contains the error description.

### Error description

| 01: Error: the index of the window group nGrpID under consideration is 0. |
| 02: Error: an element of the facade list is invalid. This is specified in the error description sErrMsg as arrFcdElem[nColumn,nRow]. |

### Requirements

**Development environment**

TwinCAT from v3.1.4024.17

**Required PLC library**

Tc3_BA2 from v4.8.9.0

### FB_BA_ShObjEntry

This function block serves for the administration of all shading elements in a facade, which is globally saved in a list of shading elements [142]. It is intended to facilitate the input of the element information - also with regard to the use of the visualization. A schematic illustration of the objects with description of the coordinates is given in Shading correction: principles and definitions [129].

The shading elements are declared in the global variables:

```plaintext
VAR_GLOBAL
  arrShdObj : ARRAY[1..BA_Param.nSunPrt.nMaxShdObj] OF ST_BA_ShObj;
END_VAR
```

Each individual element aShdObj[1] to aShdObj[nMaxShdObj] carries the information for an individual shading element (ST_BA_ShObj [95]). This information consists of the selected type of shading (rectangle or ball) and the respectively associated coordinates. For a rectangle, these are the corner points (fP1x, fP1y, fP1z), (fP2x, fP2y, fP2z), (fP3x, fP3y, fP3z) and (fP4x, fP4y, fP4z), for a sphere these are the center point (fMx, fMy, fMz) and the radius fRads. In addition, the phase of the shading can be defined via the inputs nBegMth and nEndMth, which is important in the case of objects such as trees that bear no foliage in winter.
The function block directly accesses the array of this information via the IN-OUT variable \( aShdObj \).

Note: The fact that the rectangle coordinates \( fP2x \), \( fP2z \), \( fP4x \), \( fP4y \), and \( fP4z \) are output values results from the fact that they are formed from the input parameters:

\[
\begin{align*}
fP2x &= fP1x; & fP2z &= fP1z; & fP4x &= fP3x; & fP4y &= fP1y; & fP4z &= fP3z; \\
\end{align*}
\]

That limits the input of a rectangle to the extent that the lateral edges stand vertically on the floor (\( fP2x = fP1x \) and \( fP4x = fP3x \)), that the square has no inclination (\( fP2z = fP1z \) and \( fP4z = fP3z \)) and can only have a different height "upwards", i.e. in the positive y-direction (\( fP4y = fP1y \)).

The function block is used in three steps:

- **Read**
  - Selection of the element from the list \( aShdObj[nId] \) is based on the entry at \( nId \). A rising edge on \( bRd \) reads the data. These values are assigned to the input and output variables of the function block. These are the input values \( fP1x \), \( fP1y \), \( fP1z \), \( fP2y \), \( fP3x \), \( fP3y \), \( fP3z \), \( fMx \), \( fMy \), \( fMz \), \( fRads \), the object enumerator \( eType \) and the output values \( fP2x \), \( fP2z \), \( fP4x \), \( fP4y \) and \( fP4z \). It is important here that the input values are not overwritten in the reading step. Hence, all values can initially be displayed in a visualization.

- **Change**
  - In a next program step the listed input values can then be changed. If a rectangle is preselected at input \( eType \) via the value "\( eObjectTypeTetragon \)”, the output values \( rP2x \), \( rP2z \), \( rP4x \), \( rP4y \) and \( rP4z \) result from the rectangle coordinates that were entered (see above).

- **Write**
  - The parameterized data are written to the list element with the index \( niId \) upon a positive edge on \( bWrt \), regardless of whether they represent valid values or not. The element structure \( ST_BA_ShdObj[95] \) therefore contains a plausibility bit \( bVld \), which forwards precisely this information to the function block \( FB_BA_ShdCorr[165] \) to prevent miscalculations.

This approach is to be regarded only as a proposal. It is also possible to parameterize the function block quite normally in one step and to write the values entered to the corresponding list element with a rising edge on \( bWrt \).

VAR_INPUT

\[
\begin{align*}
nId & : \text{UDINT}; \\
bRd & : \text{BOOL}; \\
bWrt & : \text{BOOL}; \\
fP1x & : \text{REAL}; \\
fP1y & : \text{REAL}; \\
fP1z & : \text{REAL}; \\
fP2y & : \text{REAL}; \\
fP3x & : \text{REAL}; \\
fP3y & : \text{REAL}; \\
fP3z & : \text{REAL}; \\
fMx & : \text{REAL}; \\
fMy & : \text{REAL}; \\
fMz & : \text{REAL}; \\
fRads & : \text{REAL}; \\
nBegMth & : \text{UDINT}; \\
nEndMth & : \text{UDINT}; \\
eType & : \text{E_BA_ShdObjType};
\end{align*}
\]
**nId**: Index of the selected element. This refers to the selection of a field element of the array saved in the IN-OUT variable \( aShdObj \). The variable **nId must not be zero**! This is due to the field definition, which starts with 1. However, an incorrect input is recognized and displayed as such at **bErr/sErrDescr**.

**bRd**: The information of the selected element, \( aShdObj[nId] \), is read into the function block with a positive edge at this input and assigned to the input variables \( fP1x \) to \( eType \) and the output variables \( fP2x \) to \( fP4z \). If data are already present on the inputs \( fP1x \) to \( eType \) at this time, then the data previously read are immediately overwritten with these data.

**bWrt**: A positive edge writes the values applied to the inputs \( fP1x \) to \( eType \) and the values determined and assigned to the outputs \( fP2x \) to \( fP4z \) to the selected field element \( aShdObj[nId] \).

**fP1x**: X-coordinate of point 1 of the shading element (rectangle) [m].

**fP1y**: Y-coordinate of point 1 of the shading element (rectangle) [m].

**fP1z**: Z-coordinate of point 1 of the shading element (rectangle) [m].

**fP2y**: Y-coordinate of point 2 of the shading element (rectangle) [m].

**fP3x**: X-coordinate of point 3 of the shading element (rectangle) [m].

**fP3y**: Y-coordinate of point 3 of the shading element (rectangle) [m].

**fP3z**: Z-coordinate of point 3 of the shading element (rectangle) [m].

**fMx**: X-coordinate of the center of the shading element (ball) [m].

**fMy**: Y-coordinate of the center of the shading element (ball) [m].

**fMz**: Z-coordinate of the center of the shading element (ball) [m].

**fRads**: Radius of the shading element (ball) [m].

**nBegMth**: Beginning of the shading period (month).

**nEndMth**: End of the shading period (month).

**eType**: Selected type of element: Rectangle or sphere (see \( E\_BA\_ShdObjType \)).

**Remark about the shading period:**
The entries for the months may not be 0 or greater than 12, otherwise all combinations are possible.

**Samples:**
Start=1, End=1: shading in January.
Start=1, End=5: shading from the beginning of January to the end of May.
Start=11, End=5: shading from the beginning of November to the end of May (the following year).

**VAR_OUTPUT**

```
fP2x     : REAL;
fP2z     : REAL;
fP4x     : REAL;
fP4y     : REAL;
fP4z     : REAL;
bErr     : BOOL;
sErrDescr : T_MAXSTRING;
```

**fP2x**: determined X-coordinate of point 2 of the shading element (rectangle) [m] (see "note [168]" above).

**fP2z**: determined Z-coordinate of point 2 of the shading element (rectangle) [m] (see "note [168]" above).

**fP4x**: determined X-coordinate of point 4 of the shading element (rectangle) [m] (see "note [168]" above).

**fP4y**: determined Y-coordinate of point 4 of the shading element (rectangle) [m] (see "note [168]" above).

**fP4z**: determined Z-coordinate of point 4 of the shading element (rectangle) [m] (see "note [168]" above).
bErr: result of the plausibility check for the values entered. For a rectangle, the internal angle is 360° and the points are in a plane in front of the facade under consideration. In the case of a ball the center must likewise lie in front of the facade and the radius must be greater than zero.

sErrDescr: contains the error description.

**Error description**

<table>
<thead>
<tr>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01: Error: the input <code>nId</code> is outside the permissible limits 1...<code>nMaxShdObj</code>.</td>
</tr>
<tr>
<td>02: Error: the sum of the angles of the rectangle is not 360°. This means that the corners are not in the order P1, P2, P3 and P4 but rather P1, P3, P2 and P4. This results in a crossed-over rectangle.</td>
</tr>
<tr>
<td>03: Error: the corners of the rectangle are not in the same plane.</td>
</tr>
<tr>
<td>04: Error: the z-component of P1 is less than zero. This corner would thus lie behind the facade.</td>
</tr>
<tr>
<td>05: Error: the z-component of P3 is less than zero. This corner would thus lie behind the facade.</td>
</tr>
<tr>
<td>06: Error: P1 is equal to P2. The object entered is thus not a rectangle.</td>
</tr>
<tr>
<td>07: Error: P1 is equal to P3. The object entered is thus not a rectangle.</td>
</tr>
<tr>
<td>08: Error: P1 is equal to P4. The object entered is thus not a rectangle.</td>
</tr>
<tr>
<td>09: Error: P2 is equal to P3. The object entered is thus not a rectangle.</td>
</tr>
<tr>
<td>10: Error: P2 is equal to P4. The object entered is thus not a rectangle.</td>
</tr>
<tr>
<td>11: Error: P3 is equal to P4. The object entered is thus not a rectangle.</td>
</tr>
<tr>
<td>12: Error: the radius entered is zero.</td>
</tr>
<tr>
<td>13: Error: The z-component of the ball center is less than zero. This point would thus lie behind the facade.</td>
</tr>
<tr>
<td>14: Error: error object type <code>eType</code> - neither rectangle nor ball.</td>
</tr>
<tr>
<td>15: Error: month input error.</td>
</tr>
</tbody>
</table>

**VAR_IN_OUT**

```
aShdObj : ARRAY[1..BA_Param.nSunPrt.nMaxShdObj] OF ST_BA_ShdObj;  
```

aShdObj: List of shading objects [142].

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

**FB_BA_SunBldActr**

This function block is used for positioning of a louvered blind via two outputs: drive up and drive down. The blind can be driven to any desired (height) position and slat angle via the positioning telegram `stSunBld` [94]. On top of that, the positioning telegram `stSunBld` [94] also contains manual commands with which the blind can be moved individually to certain positions. These manual commands are controlled by the function block `FB_BA_SunBldSwi` [182].
The current height position and the slat angle are not read in by an additional encoder, but determined internally by the travel time of the blind. The calculation is based on the following travel profile (regarded from the highest and lowest position of the blind):

**Downward travel profile:**

1) The blind is in the uppermost position
2) The backlash was moved out. The blind was driven down a little bit without turning the lamellas.
3) The lamellas are turned to the lowest angle.
4) The blind is completely driven down.

More detailed explanations of the terms “backlash” and “turning” are given here in the downward movement:

The blind normally describes its downward movement with the slat low point directed outwards, as in fig. 3). If the blind is in an initial position with the low point directed inwards (i.e. after the conclusion of an upward movement), then a certain time elapses after a new downward movement begins before the slats start to turn from the "inward low point" to the "outward low point". During this time the slat angle does not change; the blind only drives downward (fig.1 and fig. 2). This time is an important parameter for the movement calculation and is entered in the function block under \( nBckLshTiDwn \,[\text{ms}] \). Since it is not known at any point after a blind movement of any length whether backlash has already taken effect, the backlash of the downward movement or its travel time can be measured most reliably if the blind was first raised fully. A further important parameter is the time interval of the subsequent turning of the slats from the "inward low point" to the "Outward low point". This time should be entered as \( nTurnTiDwn \,[\text{ms}] \) at the function block.
Upward travel profile:

1) The blind is in the lowestmost position
2) The backlash was moved out. The blind was driven up a little bit without turning the lamellae.
3) The lamellae are turned to the highest angle.
4) The blind is completely driven up.

More detailed explanations of the terms "backlash" and "turning" are given here in the upward movement:

The circumstances are similar to the downward movement described above: The blind normally describes its upward movement with the slat low point directed inwards, as in fig. 3).
If the blind is in an initial position with the low point directed outwards (i.e. after the conclusion of a downward movement), then a certain time elapses after a new upward movement begins before the slats start to turn from the "Outward low point" to the "Inward low point". During this time the slat angle does not change; the blind only drives upward (fig. 1 and fig. 2). Also this time is an important parameter for the movement calculation and is entered in the function block under \( nBckLshTiUp \) [ms]. Since it is not known at any point after a blind movement of any length whether backlash has already taken effect, the backlash of the upward movement or its travel time can be measured most reliably if the blind was first driven fully downward. A further important parameter is the time interval of the subsequent turning of the slats from the "Outward low point" to the "Inward low point". This time should be entered as \( nTurnTiUp \) [ms] at the function block.

Parameterization

For the calculation of the (height) position and the slat angle, the following times now have to be determined for both the upward and downward movement:

- the backlash duration (\( nBckLshTiUp \) / \( nBckLshTiDwn \) [ms])
- the turning duration (\( nTurnTiUp \) / \( nTurnTiDwn \) [ms])
- the total travel time (\( nTiUp \) / \( nTiDwn \) [ms])

Furthermore the following are required for the calculation:

- the highest slat angle after turning upwards (\( fAnglLmtUp \) [°])
- the lowest slat angle after turning downwards (\( fAnglLmtDwn \) [°])

The slat angle \( \lambda \) is defined by a notional straight line through the end points of the slat to the horizontal.
Functioning

As a rule, the function block controls the blind based on the information from the positioning telegram \texttt{stSunBld [416]}. If automatic mode is active (\texttt{bManMod = FALSE}), then the current position and slat angle are always driven to, wherein changes are immediately accounted for. The height positioning takes priority: First the entered height and afterwards the slat angle are driven to. For reasons of the simplicity the position error due to the angle movement is disregarded. In manual mode (\texttt{bManMod = TRUE}), the blind is controlled by the commands \texttt{bManUp} and \texttt{bManDwn}.

An automatic movement command is triggered whenever a change from manual to automatic mode occurs.

Referencing

Secure referencing is ensured if the blind is driven upward for longer than its complete drive-up time. The position is then in any case "0" and the louvre angle is at its maximum. Since blind positioning without an encoder is naturally always susceptible to error, it is important to automatically reference as often as possible: each time the "0" position is to be driven to (the angle is unimportant), the blind initially drives upward quite normally with continuous position calculation. Once the calculated position value 0% is reached, the output \texttt{bUp} continues to be held for the complete travel-up time + 5 s.

For reasons of flexibility, there are two ways to interrupt the referencing process: Until the calculated 0% position is reached, a change in position continues to be assumed and executed. Once this 0% position is reached, the blind can still be moved with the manual "travel-down" command. These two sensible limitations make it necessary for the user to ensure that the blind is securely referenced as often as possible.

After a system restart, the function block executes a reference run. Completion of the initial referencing is indicated through a TRUE signal at output \texttt{bInitRefCmpl}. The initial referencing can also be terminated through a manual "travel-down" command.

Target accuracy

Since the function block determines the blind position solely via travel times, the cycle time of the PLC task plays a crucial role for positioning accuracy. If the switching time for a slat angle range of -70° to 10° is 1 second, for example, the accuracy at a cycle time of 50 ms is +/-4°.

\begin{Verbatim}
VAR_INPUT
\begin{tabular}{ll}
  bEn & : BOOL;
  stCmd & : \texttt{ST_BA_SunBld};
  nTiUp & : UDINT;
  nTiDwn & : UDINT;
  nTurnTiUp & : UDINT;
  nTurnTiDwn & : UDINT;
  nBckLshTiUp & : UDINT;
  nBckLshTiDwn & : UDINT;
  fAnglLmtUp & : REAL;
  fAnglLmtDwn & : REAL;
\end{tabular}
\end{Verbatim}

\texttt{bEn}: Enable input for the function block. As long as this input is TRUE, the actuator function block accepts and executes commands as described above. A FALSE signal on this input resets the control outputs \texttt{bUp} and \texttt{bDwn} and the function block remains in a state of rest.

\texttt{stCmd}: Positioning telegram, (see \texttt{ST_BA_SunBld [94]}).
nTiUp: Complete time for driving up [ms].

nTiDwn: Complete time for driving down [ms].

nTurnTiUp: Time for turning the slats in the upward direction [ms].

nTurnTiDwn: Time for turning the slats in the downward direction [ms].

nBckLshTiUp: Time to traverse the backlash in the upward direction [ms]. This input is internally limited to a minimum value of 0.

nBckLshTiDwn: Time to traverse the backlash in the downward direction [ms]. This input is internally limited to a minimum value of 0.

fAnglLmtUp: Highest position of the slats [°].

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

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

fAnglLmtDwn: Lowest position of the slats [°].

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

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

VAR_OUTPUT

| bUp     | BOOL;  |
| bDwn    | BOOL;  |
| fAct1Pos | REAL;  |
| fAct1Angl| REAL;  |
| bRef    | BOOL;  |
| nRefTi  | UDINT; |
| bInitRefCompl | BOOL;  |
| bBusy   | BOOL;  |
| bErr    | BOOL;  |
| sErrDesc | T_MAXSTRING; |
bUp: Control output for blind up.

bDwn: Control output for blind down.

fActlPos: Current position in percent.

fActlAngl: Current slat angle [°].

bRef: The blind is referencing, i.e. the output bUp is set for the complete travel-up time + 5s. Only a manual "down" command can move the blind in the opposite direction and terminate this mode.

nRefTi: Referencing countdown display [s].

bInitRefCompl: Initial referencing process complete.

bBusy: A positioning or a referencing procedure is in progress.

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

sErrDesc: Contains the error description.

<table>
<thead>
<tr>
<th>Error description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01: Error: Up/Down timer = 0.</td>
</tr>
<tr>
<td>02: Error: Turning timer = 0.</td>
</tr>
<tr>
<td>03: Error: Slat angle limits: The upper limit is less than or equal to the lower limit (fAnglLmtUp &lt;= fAnglLmtDwn).</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

**FB_BA_SunBldEvt**

This function block serves to preset the position and angle for any desired event. It can be used, for example, in order to drive to a parking position or to drive the blind upward for maintenance.

The function is activated via the input bEn. If this is the case, the Active flag in the positioning telegram (bActv in stSunBld) at output stSunBld [94] is set, and the values entered for the In/Out variables fPos for the blind height [%] and fAngl for the slat angle [°] are passed on in this telegram. If the function is no longer active due to the resetting of bEn, then the active flag in the positioning telegram stSunBld [94] is reset and the positions for height and angle are set to "0". The priority function block (e.g. FB_BA_SunBldPrioSwi4 [325]) enables a function with lower priority to take over the control by resetting.

**VAR_INPUT**

bEn : BOOL;
fPos : REAL;
fAngl : REAL;
ePrio : E_BA_SunBldPrio := E_BA_SunBldPrio.eSunProtection;

bEn: a TRUE signal on this input activates the function block and transfers the entered setpoint values together with the active flag in the positioning telegram ST_BA_SunBld [94]. A FALSE signal resets the active flag again and sets position and angle to zero.

fPos: height position of the blind [%] in case of activation.

fAngl: slat angle of the blind [°] in case of activation.
ePrio: priority of the active telegram (see E_BA_SunBldPrio [92]).

VAR_OUTPUT

| stSunBld  | ST_BA_SunBld; |
| bActv     | BOOL;        |

bActv: corresponds to the boolean value bActv in the blind telegram ST_BA_SunBld [94] and is solely used to indicate whether the function block sends an active telegram.

stSunBld: output structure of the blind positions (see ST_BA_SunBld [94]).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

FB_BA_SunBldIcePrtc

The function block FB_BA_SunBldIcePrtc deals with direction-independent anti-icing.

The weather protection has the highest priority in the blind controller (see overview [136]) and is intended to ensure that the blind is not damaged by ice or wind.

Impending icing up is detected by the fact that, during precipitation detection at bRainSns, the measured outside temperature fOtsT is below the frost limit fFrstT. This event is saved internally and remains active until it is ensured that the ice has melted again. In addition, the outside temperature must have exceeded the frost limit value for the entered deicing time nDeiceTi [s]. For safety reasons the icing event is persistently saved, i.e. also beyond a PLC failure. Thus, if the controller fails during the icing up or deicing period, the blind is considered to be newly iced up when then the controller restarts and the deicing timer starts from the beginning again.

If there is a risk of icing, the blind is moved to the protection position specified by fPosProt (height position [%]) and fAnglProt (slat angle [°]).

VAR_INPUT

| bEn        | BOOL; |
| fOtsT      | REAL; |
| bRainSns   | BOOL; |
| fFrstT     | REAL; |
| nDeiceTi   | UDINT; |
| fPosProt   | REAL; |
| fAnglProt  | REAL; |
| ePrio      | E_BA_SunBldPrio := E_BA_SunBldPrio.eSunProtection; |

bEn: the function block has no function if this input is FALSE. In the positioning telegram ST_BA_SunBld [94], 0 is output for the position and the angle, and bActv is FALSE. This means that another function takes over control of the blind via the priority controller.

fOtsT: outdoor temperature [°C].

bRainSns: input for a rain sensor.
**fFrstT:** icing-up temperature limit value [°] Celsius. This value may not be greater than 0. Otherwise an error is output.

**nDeiceTi:** time until the deicing of the blind after icing up [s]. After that the icing-up alarm is reset.

**fPosProt:** height position of the blind [%] in the case of protection.

**fAnglProt:** slat angle of the blind [°] in the case of protection.

**ePrio:** priority of the active telegram (see E BA_SunBldPrio [92]).

### VAR_OUTPUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stSunBld</td>
<td>output structure of the blind positions (see ST_BA_SunBld [94])</td>
</tr>
<tr>
<td>bActv</td>
<td>corresponds to the boolean value bActv in the blind telegram ST_BA_SunBld [94] and is solely used to indicate whether the function block sends an active telegram.</td>
</tr>
<tr>
<td>bIceAlm</td>
<td>displays the icing-up alarm.</td>
</tr>
<tr>
<td>nRemTiIceAlm</td>
<td>in the case of impending icing up (bIceAlm = TRUE), this second counter is set to the deicing time. As soon as the temperature lies above the frost point entered (fFrstT), the remaining number of seconds until the ‘all-clear’ signal is given (bIceAlm = FALSE) is displayed here. This output is 0 as long as no countdown of the time is taking place.</td>
</tr>
</tbody>
</table>

If an error occurs, this automatic control is deactivated and the position and angle are set to 0. This means that if a priority controller is in use, another function with a lower priority (see Overview [136]) automatically takes over control of the blind. In the case of a direct connection, conversely, the blind will drive to position/angle 0.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

### FB_BA_SunBldPosDly

This function block delays changes in position based on automatic commands.

If an event, e.g. weather protection, results in too many blind drives being started at the same time, fuses may be triggered by motor starting current peaks. It is therefore advisable to start the blind drives slightly staggered, in order to avoid excessive total current values.

This function block relays automatic commands from the input telegram stIn [94] to the output telegram stOut [94] with a delay. A distinction is made between three cases

1. the blind position rPos has changed in automatic mode (bManMode = FALSE in telegram stIn)
2. the slat angle rAngl has changed in automatic mode (bManMode = FALSE in telegram stIn)
3. manual mode has just been exited, i.e. automatic mode has just become active (falling edge bManMode in telegram stIn)

The output telegram stOut is always a direct copy of the input telegram stIn. However, in these three cases the output telegram stOut is set for the time nDly [ms].
This ensures that the blind controlled via the function block FB_BA_SunBldActr [171] is kept at its position during the delay period. Each further change based on the criteria mentioned above within the delay time restarts the timer.

However, a change to manual in the input telegram (bManMode = TRUE) cancels the delay timer immediately. The (manual) telegram is passed on without delay. In this way, only automatic telegrams are delayed.

Application

Preferably directly before the blind actuator function block:

![Function Block Diagram]

**VAR_INPUT**

<table>
<thead>
<tr>
<th>var</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stIn</td>
<td>ST_BA_Sunblind;</td>
<td>input positioning telegram (see ST_BA_SunBld [94]).</td>
</tr>
<tr>
<td>nDly</td>
<td>UDINT;</td>
<td>delay time of the active bit in the positioning telegram [ms].</td>
</tr>
</tbody>
</table>

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>var</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stOut</td>
<td>ST_BA_Sunblind;</td>
<td>output positioning telegram (see ST_BA_SunBld [94]).</td>
</tr>
<tr>
<td>nRemTiDly</td>
<td>UDINT;</td>
<td>display output for elapsed delay time [s].</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

FB_BA_SunBldScn

![Function Block Diagram]
This function block represents an extension of the manual controller FB_BA_SunBldSwi [182] by a scene memory and a call function. The blind control FB_BA_SunBldActr [171] or the roller blind control FB_BA_RolBldActr [163] can be active in manual mode and also directly target previously stored positions (scenes). Up to 21 scenes can be saved.

**Operation**

In manual mode, the function block controls the blind function block FB_BA_SunBldActr [171] or the roller shutter function block FB_BA_RolBldActr [163] via the command inputs bUp and bDwn; bUp has priority. The commands are passed on to the respective commands bManUp and bManDwn of the positioning telegram. If a command input is activated for longer than the entered time nSwiOvrTi [ms], the corresponding control command latches. Activating a command input again clears this latching.

A rising edge at bSavScn saves the current position and the slat angle in the scene selected at nSlcdScn. This procedure is possible at any time, even during active positioning. With bCIScn the selected scene is called up, i.e. the stored values of position and angle are driven to.

If the function block is activated by input bEn = TRUE, bit bActv is set immediately in the positioning telegram. In this way the function block signals its priority over lower priorities at a priority switch (see FB_BA_SunBldTgmSel4 or FB_BA_SunBldTgmSel8). If the command "Call Scene” is not active (bCIScn = TRUE), the bit bManMod is also set in the positioning telegram to notify the connected actuator function blocks that they should respond to manual commands.

If the function block is deactivated by bEn = FALSE, both bits, bActv and bManMod, are set to FALSE again.

**Linking to the blind function block**

Like the "normal" manual mode function block FB_BA_SunBldSwi [182], the scene selection function block can be connected either via an upstream priority control FB_BA_SunBldPrioSwi4 or FB_BA_SunBldPrioSwi8, or directly via the blind function block. The connection is established via the positioning telegram ST_BA_Sunbld [94]. Furthermore the scene function block requires the current positions from the blind function block for the reference blind:

**Use of a priority controller:**

**Direct connection:**
VAR_INPUT

bEn : BOOL;
bUp : BOOL;
bDwn : BOOL;
nSwiOvrTi : UDINT;
nSlcdScn : UDINT;
bClScn : BOOL;
bSavScn : BOOL;
fSpPos : REAL;
fSpAngl : REAL;

bEn: The function block has no function if this input is FALSE. In the positioning telegram ST_BA_Sunbld [94], 0 is output for the position and the angle - bManMod and bActv are FALSE. For a connection with priority controller this means that another functionality takes over control of the blind. Conversely, a direct connection allows the blind to drive directly to the 0 position, i.e. fully up, since the actuator function block does not evaluate the bit bActv itself.

bUp: Command input for blind up.

bDwn: Command input for blind down.

nSwiOvrTi: Time [ms] until the corresponding manual command in the positioning telegram ST_BA_Sunbld [94] switches to latching mode, if the command input is activated permanently. Internally limited to a minimum value of 0.

nSlcdScn: Selected scene which should either be saved (bSavScn) or called (bClScn). Internally limited to a minimum value from 0 to BA_Param.nSunPrt_MaxSunBldScn.

bClScn: Call selected scene.

bSavScn: Save selected scene.

fSpPos: Set position [%] that is to be saved in the selected scene. This must be linked to the actual position of the actuator function block FB_BA_SunBldActr [171] or FB_BA_RolBldActr [163] of the reference blind/roller shutter, in order to be able to save a position that was previously approached manually. Internally limited to values between 0 and 100.

fSpAngl: ditto. Slat angle [°].

VAR_OUTPUT

stSunBld : ST_BA_SunBld;
bActv : BOOL;
fActlScnPos : REAL;
fActlScnAngl : REAL;

stSunBld: Positioning telegram (see ST_BA_SunBld [94]).

bActv: Corresponds to the boolean value bActv in the blind telegram ST_BA_SunBld [94] and is solely used to indicate whether the function block sends an active telegram.

fActlScnPos: Indicates the saved relative blind height position [%] for the currently selected scene.

fActlScnAngl: ditto. Slat angle [°].

If an error occurs, this automatic control is deactivated, and the position and angle are set to 0. This means that if a priority controller is in use, another function with a lower priority (see Overview [136]) automatically takes over control of the blind. In the case of a direct connection, conversely, the blind will drive to position/angle 0.

VAR_IN_OUT

aSunBldScn : ARRAY[0..BA_Param.nSunPrt_MaxSunBldScn] OF ST_BA_SunBldScn;

aSunBldScn: Table with the scene entries of the type ST_BA_SunBldScn [95].
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

FB_BA_SunBldSwi

This function block can be used to control the blind FB_BA_SunBldActr [171] or roller shutter FB_BA_RolBldActr [163] in manual mode. The connection takes place via the positioning telegram ST_BA_Sunbld [94] either directly or with an additional priority controller.

Operation

In manual mode, the function block controls the blind function block FB_BA_SunBldActr [171] or the roller shutter function block FB_BA_RolBldActr [163] via the command inputs bUp and bDwn; bUp has priority. The commands are passed on to the respective commands bManUp and bManDwn of the positioning telegram. If a command input is activated that is longer than the entered time nSwiOvrTi [ms], then the corresponding control command latches. Activating a command input again releases this latch. If the function block is activated by input bEn = TRUE, bit bActv is set immediately in the positioning telegram. The function block uses this to indicate its priority over low priorities at the priority switch (see ....). At the same time, the bit bManMod is set in the positioning telegram to notify the connected actuator function blocks that they should respond to manual commands.

If the function block is deactivated by bEn = FALSE, both bits, bActv and bManMod, are set to FALSE again.

Linking to the blind function block

The manual mode function block can be connected either via an upstream priority control FB_BA_... or directly at the blind function block. The connection is established via the positioning telegram ST_BA_Sunbld [94].

Use of a priority controller:

![Diagram of SunblindSwitch and PrioritySwitch connections]
Direct connection:

**VAR_INPUT**

<table>
<thead>
<tr>
<th>bEn</th>
<th>BOOL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bUp</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bDwn</td>
<td>BOOL;</td>
</tr>
<tr>
<td>nSwiOvrTi</td>
<td>UDINT;</td>
</tr>
</tbody>
</table>

**bEn**: The function block has no function if this input is FALSE. In the positioning telegram ST BA Sunbld [94], 0 is output for the position and the angle - bManMod and bActv are FALSE. For a connection with priority controller this means that another functionality takes over control of the blind. Conversely, a direct connection allows the blind to drive directly to the 0 position, i.e. fully up, since the actuator function block does not evaluate the bit bActv itself.

**bUp**: Command input for blind up.

**bDwn**: Command input for blind down.

**nSwiOvrTi**: Time [ms] until the corresponding manual command in the positioning telegram ST BA Sunbld [94] switches to latching mode, if the command input is activated permanently. Internally limited to a minimum value of 0.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>stSunBld</th>
<th>ST BA SunBld;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bActv</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>

**stSunBld**: positioning telegram (see ST BA SunBld [94]).

**bActv**: corresponds to the boolean value bActv in the blind telegram ST BA SunBld [94] and is solely used to indicate whether the function block sends an active telegram.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3 BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
FB_BA_SunBldTwiLgtAuto

This function block controls the blind when the outdoor brightness has fallen below a limit value.

The automatic twilight function operates with both a value hysteresis and a temporal hysteresis: If the outdoor brightness value \( f_{Brtns} \) falls below the value \( f_{ActvVal} \) for the time \( n_{ActvDly} \) [s], the function block is active and will provide the blind positions \( f_{PosTwiLgt} \) (height [%]) and \( f_{AnglTwiLgt} \) (slat angle [°]) specified for the input variables at the output in the positioning telegram \( ST\_BA\_SunBld \) [94]. If the outdoor brightness exceeds the value \( f_{DctvVal} \) for the time \( n_{DctvDly} \) [s], automatic mode is no longer active. The active flag in the positioning telegram \( ST\_BA\_SunBld \) [94] is reset and the positions for height and angle are set to "0". A function with a lower priority can then take over control.

VAR_INPUT

\begin{verbatim}
VAR_INPUT
bEn : BOOL;
fBrtns : REAL;
fActvVal : REAL;
fDctvVal : REAL;
nActvDly : UDINT;
nDctvDly : UDINT;
fPosTwiLgt : REAL;
fAnglTwiLgt : REAL;
ePrio : E_BA_SunBldPrio := E_BA_SunBldPrio.eSunProtection;
\end{verbatim}

- \( bEn \): the function block has no function if this input is FALSE. In the positioning telegram \( ST\_BA\_SunBld \) [416], 0 is output for the position and the angle, and \( bActv \) is FALSE. This means that another function takes over control of the blind via the priority controller.

- \( fBrtns \): outdoor brightness [lx].

- \( fActvVal \): activation limit value [lx]. The value \( fActvVal \) is internally limited to values from 0 to \( rDctvVal \).

- \( fDctvVal \): deactivation limit value [lx]. Internally limited to a minimum value of 0.

- \( nActvDly \): activation delay [s]. Internally limited to a minimum value of 0.

- \( nDctvDly \): deactivation delay [s]. Internally limited to a minimum value of 0.

- \( fPosTwiLgt \): vertical position of the blind [%] if the automatic twilight function is active. Internally limited to values between 0 and 100.

- \( fAnglTwiLgt \): slat angle of the blind [%] if the automatic twilight function is active.

- \( ePrio \): priority of the active telegram (see \( E\_BA\_SunBldPrio \) [92]).

VAR_OUTPUT

\begin{verbatim}
VAR_OUTPUT
stSunBld : ST_BA_SunBld;
bActv : BOOL;
nRemTiActv : UDINT;
nRemTiDctv : UDINT;
\end{verbatim}

- \( stSunBld \): output structure of the blind positions (see \( ST\_BA\_SunBld \) [94]).

- \( bActv \): corresponds to the boolean value \( bActv \) in the blind telegram \( ST\_BA\_SunBld \) [94] and is solely used to indicate whether the function block sends an active telegram.
nRemTiActv: shows the time remaining [s] after falling below the switching value \(f_{ActvVal}\) until automatic mode is activated. This output is 0 as long as no countdown of the time is taking place.

nRemTiDctv: shows the time remaining [s] after exceeding the switching value \(f_{DctvVal}\) until automatic mode is disabled. This output is 0 as long as no countdown of the time is taking place.

If an error occurs, this automatic control is deactivated and the position and angle are set to 0. This means that if a priority controller is in use, another function with a lower priority (see Overview [136]) automatically takes over control of the blind. In the case of a direct connection, conversely, the blind will drive to position/angle 0.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

FB_BA_SunBldWndPrtc

The function block \(FB_{-}BA\_{-}SunBldWndPrtc\) deals with the direction-dependent wind protection.

The weather protection has the highest priority in the blind controller (see overview [136]) and is intended to ensure that the blind is not damaged by ice or wind.

If the measured wind speed is above the value \(f_{WndSpdStrmOn}\) for the time \(nDlyStrmOn\) [s], it is assumed that high winds are imminent. The storm is regarded as having subsided, so that the blind can be moved safely, once the wind speed falls below the value \(f_{WndSpdStrmOff}\) for the time \(nDlyStrmOff\) [s]. For safety reasons the storm event is also persistently saved. Thus, if the controller fails during a storm, the sequence timer is started again from the beginning when the controller is restarted.

If there is a risk of high wind, the blind is moved to the protection position specified by \(f_{PosProt}\) (height position in percent) and \(f_{AnglProt}\) (slat angle [°]).

VAR_INPUT

\[
\begin{align*}
bEn & : \text{BOOL}; \\
f_{WndSpd} & : \text{REAL}; \\
f_{WndSpdStrmOn} & : \text{REAL}; \\
f_{WndSpdStrmOff} & : \text{REAL}; \\
nDlyStrmOn & : \text{UDINT}; \\
nDlyStrmOff & : \text{UDINT}; \\
f_{PosProt} & : \text{REAL}; \\
f_{AnglProt} & : \text{REAL}; \\
ePrio & : \text{E_{-}BA\_{-}SunBldPrio} := \text{E_{-}BA\_{-}SunBldPrio}.eSunProtection;
\end{align*}
\]

**bEn:** the function block has no function if this input is FALSE. In the positioning telegram \(ST_{-}BA\_{-}SunBld\) [94], 0 is output for the position and the angle, and \(b\_\text{Actv}\) is FALSE. This means that another function takes over control of the blind via the priority controller.

**fWndSpd:** wind speed. The unit of the entry is arbitrary, but it is important that no value is smaller than 0 and that the values become larger with increasing speed.
**fWndSpdStrmOn:** wind speed limit value for the activation of the storm alarm. This value may not be smaller than 0 and must lie above the value for the deactivation. Otherwise an error is output. The unit of the entry must be the same as that of the input fWndSpd. A value greater than this limit value triggers the alarm after the entered time nDlyStrmOn.

**fWndSpdStrmOff:** wind speed limit value for the deactivation of the storm alarm. This value may be not smaller than 0 and must lie below the value for the activation. Otherwise an error is output. The unit of the entry must be the same as that of the input fWndSpd. A value smaller than or equal to this limit value resets the alarm after the entered time nDlyStrmOff.

**nDlyStrmOn:** time delay until the storm alarm is triggered [s].

**nDlyStrmOff:** time delay until the storm alarm is reset [s].

**fPosProt:** height position of the blind [%] in the case of protection.

**fAnglProt:** slat angle of the blind [°] in the case of protection.

**ePrio:** priority of the active telegram (see E_BA_SunBldPrio [92]).

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stSunBld</td>
<td>output structure of the blind positions (see ST_BA_SunBld [94]).</td>
</tr>
<tr>
<td>bActv</td>
<td>corresponds to the boolean value bActv in the blind telegram ST_BA_SunBld [94] and is solely used to indicate whether the function block sends an active telegram.</td>
</tr>
<tr>
<td>bStrmAlm</td>
<td>displays the storm alarm.</td>
</tr>
<tr>
<td>nRemTiStrmDetc</td>
<td>in an uncritical case this second counter constantly displays the alarm delay time nDlyStrmOn. If the measured wind speed fWndSpd is above the activation limit value fWndSpdStrmOn, the seconds to the alarm are counted down. This output is 0 as long as no countdown of the time is taking place.</td>
</tr>
<tr>
<td>nRemTiStrmAlm</td>
<td>as soon as the storm alarm is triggered, this second counter first constantly displays the deactivation time delay of the storm alarm nDlyStrmOff. If the measured wind speed fWndSpd falls below the deactivation limit value fWndSpdStrmOff, the seconds to the all-clear signal (bStrmAlm=FALSE) are counted down. This output is 0 as long as no countdown of the time is taking place.</td>
</tr>
</tbody>
</table>

---

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

---

If an error occurs, this automatic control is deactivated and the position and angle are set to 0. This means that if a priority controller is in use, another function with a lower priority (see Overview [136]) automatically takes over control of the blind. In the case of a direct connection, conversely, the blind will drive to position/angle 0.
The function block is used for glare protection with the aid of a slatted blind. Glare protection is realized through variation of the slat angle and positioning of the blind height. The slat angle is set as a function of the sun position such that direct glare is prevented, while letting as much natural light through as possible.

Three different operation modes are available for varying the blind height.

1. When sun protection is active, the blind moves to a fixed height. The height value is specified with the variable \textit{fFixPos}.
2. The blind position is varied as a function of the sun position. The position is specified in the table \textit{(ST\_BA\_BldPosTab [93])}. See also description of \textit{FB\_BA\_BldPosEntry [287]}.
3. The high of the blind is calculated based on the window geometry such that the sun’s rays reach a specified depth in the room. The incidence depth of the sun's rays is defined with the variable \textit{fMaxLgtIndc}.

In order to avoid excessive repositioning of the slat angle, the variable \textit{nPosIntval [Min]} can be used to specify a time interval, within which the slat angle is not adjusted. In order to avoid glare, the angle is always changed sufficiently for the respective time interval.

The following conditions must be met for positioning the blind and setting the slat angle.

- 1. The input \textit{bEn} must be \textbf{TRUE}.
- 2. The sun must have risen. (elevation > 0)
- 3. The function block is parameterized correctly (\textit{bErr} = FALSE)

\textbf{VAR\_INPUT}

\begin{verbatim}
VAR_INPUT
bEn     : BOOL;
stUTC   : TIMESTRUCT;
nPosIntval : UDINT;
fDegLLngd : REAL;
fDegLatd : REAL;
fFcdOrtn : REAL;
fFcdAng1 : REAL;
flamWdth : REAL;
flamDStc : REAL;
fFcdOrtn : REAL;
fWdhtFr : REAL;
fDtcWdFr : REAL;
stBldPosTab : ST\_BA\_BldPosTab;
ePosMod : E\_BA\_PosMod := E\_BA\_PosMod.eFix;
ePrio : E\_BA\_SunBldPrio := E\_BA\_SunBldPrio.eSunProtection;
\end{verbatim}
**EN:** if this input is set to FALSE, the positioning is inactive, i.e. the active bit \((bActv)\) is reset in the positioning structure \(stSunBld\) of the type \(ST\_BA\_SunBld\) and the function block itself remains in a standstill mode. If on the other hand the function block is activated, then the active bit is TRUE and the function block outputs its control values \((fPos, fAngl)\) in the positioning structure at the appropriate times.

**stUTC:** input of current time as coordinated world time (UTC - Coordinates Universal Time, previously referred to as GMT, Greenwich Mean Time) (see \(TIMESTRUCT\)). The function block \(FB\_BA\_GetTime\) can be used to read this time from a target system.

A jump of more than 300 seconds leads to immediate repositioning, if the blind is in the sun and glare protection is active, based on the above criteria. This functionality was added to ensure a reproducible program execution.

**nPosIntval:** positioning interval in minutes - time between two blind position outputs. Valid range: 1 min...720 min.

**fDegLngd:** longitude \([°]\). Valid range: -180°...180°.

**fDegLatd:** latitude \([°]\). Valid range: -90°...90°.

**fFcdOrtn:** facade orientation \([°]:\)

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

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

The following applies for the southern hemisphere:

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

**fFcdAngl:** facade inclination \([°]\) (see \(Facade inclination\) \([°]\).)

**fLamWdth:** width of the slats (see \(sketch\) \([°]\)).

**fLamDstc:** slat spacing in mm (see \(sketch\) \([°]\)).

**fFixPos:** fixed (constant) blind height \([0...100\%]\). Applies if \(ePosMod = ePosModFix\) (see \(E\_BA\_PosMod\) \([°]\)).

**fMaxLgtIndc:** maximum desired light incidence in mm measured from the outside of the wall (see \(Sun protection: Basic principles and definitions\) \([°]\)). The parameters \(fWdwHght\) and \(fDstcWdwFlr\) are used to calculate how high the blinds must be, depending on the position of the sun, such that the incidence of light does not exceed the value \(fMaxLgtIndc\). Applies if \(ePosMod = ePosModeMaxIncidence\) (see \(E\_BA\_PosMod\) \([°]\)).

**fWdwHght:** window height in mm for the calculation of the blind height if the mode "maximum desired incidence of light" is selected.

**fDstcWdwFlr:** distance between the floor and the window sill in mm for the calculation of the blind height if the mode "maximum desired incidence of light" is selected.

**stBldPosTab:** table of 6 interpolation points, 4 of which are parameterizable, from which a blind position is then given in relation to the position of the sun by linear interpolation. Applies if \(ePosMod = ePosModFix\) (see \(E\_BA\_PosMod\) \([°]\)). For a more detailed description please refer to \(FB\_BA\_BldPosEntry\) \([°]\).
ePosMod: selection of the positioning mode (see E\_BA\_PosMod [91]).

ePrio: priority of the active telegram (see E\_BA\_SunBldPrio [92]).

VAR OUTPUT

| stSunBld     : ST\_BA\_SunBld; |
| bActv        : BOOL;         |
| bErr          : BOOL;        |
| sErrorDescr   : T\_MAXSTRING; |

stSunBld: output structure of the blind positions (ST\_BA\_SunBld [94]).

bActv: the function block is in the active state, i.e. no error is pending, the function block is enabled and the sun position is in the specified facade area (the facade is sunlit).

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

sErrorDescr: contains the error description.

<table>
<thead>
<tr>
<th>Error description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01: Error: the duration of the positioning interval is less than or equal to zero, or it exceeds 720 min.</td>
</tr>
<tr>
<td>02: Error: the value entered for the longitude is not in the valid range of -180°...180°.</td>
</tr>
<tr>
<td>03: Error: the value entered for the latitude is not in the valid range of -90°...90°.</td>
</tr>
<tr>
<td>04: Error: the value entered for the facade inclination ( fFcdAngl ) is outside the valid range of -90°..90°.</td>
</tr>
<tr>
<td>05: Error: the value for the slat spacing ( fLamDstc ) is greater than or equal to the value for the slat width ( fLamWdth ). This does not represent a &quot;valid&quot; blind, since the slats cannot close fully. Mathematically, this would lead to errors.</td>
</tr>
<tr>
<td>06: Error: the value entered for the slat width ( fLamWdth ) is zero.</td>
</tr>
<tr>
<td>07: Error: the value entered for the slat spacing ( fLamDstc ) is zero.</td>
</tr>
<tr>
<td>08: Error: the value entered for the fixed blind height ( fFixPos ) is greater than 100 or less than 0. At the same time, positioning &quot;fixed blind height&quot; is selected - ePosMod = ePosModFix.</td>
</tr>
<tr>
<td>09: Error: the &quot;Values valid&quot; bit ( bVld ) in the ( stBldPosTab ) positioning table is not set - invalid values, see FB_BA_BldPosEntry. At the same time, &quot;Table&quot; positioning is selected – ePosMod = ePosModTab.</td>
</tr>
<tr>
<td>10: Error: the value entered for the maximum required light incidence ( fMaxLgtIndc ) is less than or equal to zero. At the same time, &quot;maximum light incidence&quot; is selected – ePosMod = ePosModMaxIndc.</td>
</tr>
<tr>
<td>11: Error: the value entered for the window height ( fWdwHght ) is less than or equal to zero. At the same time, &quot;maximum light incidence&quot; is selected – ePosMod = ePosModMaxIndc.</td>
</tr>
<tr>
<td>12: Error: the distance between lower window edge and floor ( fDstcWdwFlr ) that was entered is less than zero. At the same time, &quot;maximum light incidence&quot; is selected – ePosMod = ePosModMaxIndc.</td>
</tr>
<tr>
<td>13: Error: an invalid positioning mode is entered at input ePosMod.</td>
</tr>
</tbody>
</table>

If an error occurs, this automatic control is deactivated and the position and angle are set to 0. This means that if a priority controller is in use, another function with a lower priority (see Overview [136]) automatically takes over control of the blind. In the case of a direct connection, conversely, the blind will drive to position/angle 0.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
**System**

**FB_BA_CnvTiSt**

The function block FB_BA_CnvTiSt can be used to combine the individual components of a time structure into a single structure.

The function block does not check for incorrect entries, such as an hour entry of 99. It makes sense to check this in the connected function blocks, which have to check the time structure in any case. The limit values are shown as part of the variable explanations.

**VAR_INPUT**

- `nYear` : WORD;
- `nMonth` : WORD;
- `nDay` : WORD;
- `nHour` : WORD;
- `nMinute` : WORD;
- `nSecond` : WORD;
- `nMilliseconds` : WORD;

`nYear`: The year (1970...2106).

`nMonth`: The month (1...12).

`nDay`: The day of the month (1...31).

`nHour`: The hour (0...23).

`nMinute`: The minutes (0...59).

`nSecond`: The seconds (0...59).

`nMilliseconds`: The milliseconds (0...999).

**VAR_OUTPUT**

- `bNewData` : BOOL;
- `stTi` : TIMESTRUCT;

`bNewData`: The output is TRUE in the cycle in which the input variables have changed.

`stTi`: Output time structure (see TIMESTRUCT)

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
FB_BA_ExtTiSt

The function block FB_BA_ExtTiSt resolves a time structure into the different components, so that it can be used for time conditions, for example.

**VAR_INPUT**

tTi : TIMESTRUCT;

tTi: Input time structure (see TIMESTRUCT)

**VAR_OUTPUT**

nYear : WORD;
nMonth : WORD;
nDayOfWeek : WORD;
nDay : WORD;
nHour : WORD;
nMinute : WORD;
nSecond : WORD;
nMilliseconds : WORD;

nYear: The year (1970...2106).
nMonth: The month (1...12).
nDayOfWeek: The day of the week (0 (Sun)...0 (Sat)).
nDay: The day of the month (1...31).
nHour: The hour (0...23).
nMinute: The minutes (0...59).
nSecond: The seconds (0...59).
nMilliseconds: The milliseconds (0...999).

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
With this function block an internal clock (Real Time Clock RTC) can be implemented in the TwinCAT PLC. When the function block is enabled via \( bEn \), the RTC clock is initialized with the current NT system time. One system cycle of the CPU is used to calculate the current RTC time. The function block must be called once per PLC cycle in order for the current time to be calculated. Within the function block, an instance of the function blocks NT_GetTime, FB_GetTimeZoneInformation and RTC_EX2 is called. The time is output at the outputs \( tSysTi \) for the read system time and \( tUtc \) for the Coordinated Universal Time (UTC). This is determined internally from the system time and the time zone. If the system time and/or the time zone was entered incorrectly, the UTC time will also be wrong.

The system time is read cyclically via the timer to be set, \( nUpdRTC \) [s]; it is used to synchronize the internal RTC clock. The time information (time zone, time shift relative to UTC, summer/winter time) is read in the same cycle. The output \( nRemTiUpd \) indicates the seconds remaining to the next read cycle. The time structures that are output, \( dtSysTi \) and \( dtUtc \), can be resolved with the aid of the function block FB_BA_ExtTiSt into the components day, month, hour, minute etc.

### Information on the read/wait cycle

During the read cycle, the outputs \( bRdySysTi \) and \( bRdyTiZolInfo \) change to FALSE, and the enumerator \( eTiZId \) shows \( 0 = eTimeZoneID_Unknown \). If the read operation was successful, the outputs switch back to TRUE or show the respective information for summer or winter time, if available. If the read operation was unsuccessful - internally the system waits for a response for 5 seconds - the outputs remain at FALSE or 0, and another wait cycle is started before the next read cycle. Although the internal RTC clock is not synchronized in the event of an error and may still show the right time, the time information may be wrong, and therefore also the UTC time. Errors during the read cycle will, any case, show up in \( bErr \) and \( sErrDescr \). The countdown output \( nRemTiUpd \) is not restarted until the wait cycle starts.

### VAR_INPUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEn</td>
<td>BOOL</td>
</tr>
<tr>
<td>tNetId</td>
<td>T_AmsNetId;</td>
</tr>
<tr>
<td>nUpdRtc</td>
<td>UDINT;</td>
</tr>
<tr>
<td>bUpdRtc</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>

**bEn**: Enables the function block. If \( bEn = \) TRUE, then the RTC clock is initialized with the NT system time.

**tNetId**: This parameter can be used to specify the AmsNetId (see T_AmsNetId) of the TwinCAT computer whose NT system time is to be read as timebase. If it is to be run on the local computer, an empty string can be entered.

**nUpdRtc**: Time specification [s] with which the RTC clock is regularly synchronized with the NT system time. Internally this value is limited to a minimum of 5 seconds, in order to ensure correct processing of the internal function blocks.

**bUpdRtc**: In parallel with the time \( nUpdRtc \), the RTC clock can be synchronized via a positive edge at this input.
VAR_OUTPUT

bRdySysTi : BOOL;
bRdyTiZoInfo : BOOL;
bRdyRTC : BOOL;
nRemTiUpd : UDINT;
tSysTi : TIMESTRUCT;
tUTC : TIMESTRUCT;
dtSysTi : DT;
dTUTC : DT;
nCurrentTime_ms : UDINT;
eTiZId : E_TimeZoneID;
bErr : BOOL;
sErrDescr : T_MAXSTRING;

bRdySysTi: The system time was read successfully from the target system.

bRdyTiZoInfo: The additional time information (time zone, time shift relative to UTC and summer/winter time) was read successfully.

bRdyRTC: This output is set if the function block has been initialized at least once. If this output is set, then the values for date, time and milliseconds at the outputs are valid.

nRemTiUpd: Countdown to next synchronization/update of the time information.

stSysTi: System time of the read target system (see TIMESTRUCT). The time structure can be resolved with the aid of the function block FB_BA_ExtTiSt into its components: day, month, hour, minute etc. Note: If the function block is not enabled (bEn = FALSE), the output stSysTi and its subelements (day month, etc.) show 0.

tUTC: Coordinated world time (see TIMESTRUCT). This is determined internally from the system time and the time information read from the target system. The time structure can be resolved with the aid of the function block FB_BA_ExtTiSt into its components: day, month, hour, minute etc. Note: If the function block is not enabled (bEn = FALSE), the output tUTC and its subelements (day month, etc.) show 0.

dtSysTi / dtUTC: As stSysTi / tUTC, but in DATE-AND-TIME format: year-month-day-hours-minutes-seconds. Note: If the function block is not enabled (bEn = FALSE), the outputs dtSysTi and dtUTC show DT#1970-01-01-00:00, since this is the lower limit, which corresponds to the zeros in the structure representation of dtSysTi / dtUTC.

nCurrentTime: Current time of day [ms].
eTiZId: Enumerator for summer/winter time information (see E_TimeZoneID).
bErr: This output is switched to TRUE if the parameters entered are erroneous.
sErrDescr: Contains the error description.

<table>
<thead>
<tr>
<th>Error description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01: Warning: ADS error when reading the time (NT_GetTime). The ADS error number is stated.</td>
</tr>
<tr>
<td>02: Warning: ADS error when reading the time zone information (FB_GetTimeZoneInformation). The ADS error number is stated.</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

FB_BA_SetTime
The function block FB_BA_SetTime can be used to set the local NT system time and the date for a TwinCAT system (the local NT system time is shown in the taskbar). The system time is specified via the structure tSysTi.

Internally, an instance of the function block NT_SetLocalTime from the TcUtilities library is called in the function block.

The local NT system time can also be synchronized with a reference time with the aid of the SNTP protocol. For further information please refer to the Beckhoff Information System under: Beckhoff Information System > Embedded PC > Operating systems > CE > SNTP: Simple Network Time Protocol

VAR_INPUT

bSet : BOOL;
tNetId : T_AmsNetId;
tSysTi : TIMESTRUCT;
nTiOut : UDINT;

bSet: Activation of the function block with a rising edge.

tNetId: This parameter can be used to specify the AmsNetID of the TwinCAT computer, whose local NT system time is to be set. An empty string tNetId := ""; can also be specified for the local computer (see T_AmsNetId).

tSysTi: Structure with the new local NT system time (see TIMESTRUCT). If the time is not available as structure, it is advisable to use the function block FB_BA_CnvtTiSt[190], which brings the subvariables of date and time in a structure together.

nTiOut: Indicates the timeout time [s], which must not be exceeded during execution.

VAR_OUTPUT

bBusy : BOOL;
bError : BOOL;
sErrorDescr : T_MAXSTRING;

bBusy: If the function block is activated via a rising edge at bSet, this output is set and remains set until feedback occurs.

bError: This output is set to TRUE, if either the system time to be transferred is incorrect or an ADS error occurs during the transfer.
sErrDescr: Contains the error description.

<table>
<thead>
<tr>
<th>Error description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01: Error: Error: range exceeded year</td>
</tr>
<tr>
<td>02: Error: Error: range exceeded month</td>
</tr>
<tr>
<td>03: Error: Error: range exceeded day of the month</td>
</tr>
<tr>
<td>04: Error: Error: range exceeded hour</td>
</tr>
<tr>
<td>05: Error: Error: range exceeded minute</td>
</tr>
<tr>
<td>06: Error: Error: range exceeded second</td>
</tr>
<tr>
<td>07: Error: Error: range exceeded millisecond</td>
</tr>
<tr>
<td>08: Warning: An ADS error occurred while setting the time (NT_SetLocalTime). The ADS error number is stated.</td>
</tr>
</tbody>
</table>

Time specification limits

The time structure stUtcTi that was created is internally checked for limits (see TIMESTRUCT)

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

FB_BA_WrtPersistDat

When activated, the function block FB_BA_WrtPersistDat first saves the persistent data in the Port_xxx.bootdata file. It is not necessary to explicitly specify at which port or runtime system the PLC is located; this is determined internally. Once the data has been written, the content of the file Port_xxx.bootdata is copied to the backup file Port_xxx.bootdata-old. Thus both files are always synchronized. In case the original file with the persistent data is not readable, the backup copy, which is then read, contains the same data.

In any case, the checkmark for "Clear Invalid Persistent Data" must be removed (see Description of persistent data handling under TwinCAT3).

The function block can be started in two ways:

Via a positive edge at input bStt, if the function block is not in the set start-up phase.

Initially once the start-up phase is completed after a reset or TwinCAT restart. The duration is set at nInitSttDly in seconds. If "0" is entered there, the duration of the start-up phase is 0 and an initial execution of the function block is ignored.

No commands are accepted at bStt during the start-up phase.

If errors occur while reading, writing, opening or closing the files, this is indicated with a corresponding error message at bErr/sErrDescr. After an internally fixed waiting time of two seconds, the function block automatically attempts to execute the command (read, write, open or close) again.

It is therefore advisable to keep an eye on the error outputs or to evaluate them.
It is also important to note whether the backup file for the persistent data was loaded during the TwinCAT restart or after a reset. This indicates that the original file cannot be read and that the memory card of the controller is defective. It can be queried for each runtime system with the Boolean assignment of `TwinCAT_SystemInfoVarList._AppInfo.OldBootData` (see PlcAppSystemInfo).

Sample in ST:

```plaintext
PROGRAM Example_ST
VAR
  boLDData  :  BOOL;
END_VAR

boLDData:=TwinCAT_SystemInfoVarList._AppInfo.OldBootData;
```

Sample in CFC:

```plaintext
PROGRAM Example_CFC
VAR
  boLDData  :  BOOL;
END_VAR

TwinCAT_SystemInfoVarList._AppInfo.BootDataLoaded boLDData
```

**NOTE**

**File handle conflicts**

Make sure that only this function block and only one instance of it accesses the persistent data. If several function blocks open a file and do not close it again, unforeseen file handle conflicts can occur which cannot be intercepted. The persistent data will then no longer be updated in the xxx.bootdata file.

**Description of persistent data handling under TwinCAT 3**

TwinCAT saves the persistent data for each runtime system in a file during each orderly shutdown, i.e. when switching from Run to Config or Stop mode.

The file name consists of the ADS port name of the runtime system with the file extension .bootdata, e.g.: `Port_851.bootdata` and is stored in the TwinCAT directory under `TwinCAT\3.1\Boot\PLC`.

When the system is restarted, i.e. when switching to run mode, this file is read and then saved as `Port_xxx.bootdata-old`.

If the file `Port_xxx.bootdata-old` already exists, it is overwritten.

The original file `Port_xxx.bootdata` then no longer exists. It is created again automatically when switching to Stop mode or by the function block `FB_WritePersistentData` from the TC2_Utilities library.

This behavior applies to each runtime system; each system has its own files with persistent data.

If the file is defective when the TwinCAT system is restarted, the system automatically accesses the backup file `Port_xxx.bootdata-old`. However, this behavior only applies if the **Clear Invalid Persistent Data** checkmark is unchecked in the runtime settings. If it is checked and the original file is defective, no data will be read.
The Port_xxx.bootdata-old backup file is also used when the controller is de-energized. In this case, too, the current persistent data is not stored in Port_xxx.bootdata. When the system is restarted, only the old data is available, unless a more up-to-date file was created by the function block FB_WritePersistDat before the system was switched off.
VAR_INPUT

bStt : BOOL;
nInitSttDly : UDINT;

bStt: A rising edge at this input starts the function block if it is not in the start-up phase.
nInitSttDly: Start-up phase after a reset or TwinCAT restart. The duration is set in seconds. Once the start-up phase has elapsed, the function block is automatically started once. No commands are accepted at bStt during the start-up phase. If “0” is set at nInitSttDly, the start-up phase is skipped. This input is preconfigured with 10 s.

VAR OUTPUT

VAR_OUTPUT

bBusy : BOOL;
nRemTiInitSttDly : UDINT;
bErr : BOOL;
sErrDescr : T_MaxString;

bBusy: The function block is being executed.

nRemTiInitSttDly: Countdown of the set startup phase.

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

sErrDescr: Contains the error description.

Error description

<table>
<thead>
<tr>
<th>Error number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Error: The number of the ADS port issued by the PLC is “0“</td>
</tr>
<tr>
<td>02</td>
<td>Warning: Error when writing the persistent data via the internal function block FB_WritePersistentData. Additionally its error number.</td>
</tr>
<tr>
<td>03</td>
<td>Warning: Error when opening the backup file (xxx.bootdata-old) via the internal function block FB_FileOpen. Additionally its error number.</td>
</tr>
<tr>
<td>04</td>
<td>Warning: Error when reading the original file (xxx.bootdata) via the internal function block FB_FileRead. Additionally its error number.</td>
</tr>
<tr>
<td>05</td>
<td>Warning: Error when writing to the backup file (xxx.bootdata-old) via the internal function block FB_FileWrite. Additionally its error number.</td>
</tr>
<tr>
<td>06</td>
<td>Warning: Error when closing the original file (xxx.bootdata) via the internal function block FB_FileClose. Additionally its error number.</td>
</tr>
<tr>
<td>07</td>
<td>Warning: Error when closing the backup file (xxx.bootdata-old) via the internal function block FB_FileClose. Additionally its error number.</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

Universal

actuators

FB_BA_Actuator_3Point

The function block is used to control a 3-point actuator, e.g. a 3-point flap or a 3-point valve.

The command for opening the actuator is connected to output bOpen.

The command for closing the actuator is connected to output bClose.
In automatic mode (\( nOpMode = 0 \)) the control commands of \( bCmdOpen \) and \( bCmdClose \) are forwarded directly to the outputs \( bOpen \) and \( bClose \).

The \( nOpMode \) input is used to determine the operation mode of the 3-point actuator:

- 0 = Automatic
- 1 = Stop ( \( bOpen = bClose = FALSE \) )
- 2 = Close
- 3 = Open

### VAR_INPUT

- \( bEn \): General enable of the function block.
- \( bAutoOpen \): Command to open the actuator.
- \( bAutoClose \): Command to close the actuator.
- \( nOpMode \): Select operation mode (0 = Automatic, 1 = Stop ( \( bOpen = bClose = FALSE \) ), 2 = Close, 3 = Open)

### VAR_OUTPUT

- \( bOpen \): Open control output.
- \( bClose \): Close control output.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

### FB_BA_Anlg3Pnt

The function block is intended for control of three-point actuators for valves or dampers.

A continuous control signal for positioning an actuator is converted into binary commands for opening and closing.

If the deviation between the set position value \( fIn \) and the calculated actual position value \( fPos \) of the actuator exceeds the set threshold value \( fHys / 2 \), the function block starts to correct the position by switching the outputs \( bOpn \) or \( bCls \), depending on the magnitude of the control deviation:
<table>
<thead>
<tr>
<th>bOpn</th>
<th>bCls</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

If the function block reaches an end position \( fOut = 0 \) or \( fOut = 100 \) through a corresponding input value \( fIn \), the corresponding switching output remains permanently set in order to safely reach this end position at the valve or damper:

<table>
<thead>
<tr>
<th>bOpn</th>
<th>bCls</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>permanently TRUE</td>
</tr>
<tr>
<td>permanently TRUE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Any deactivation of the continuous signal must be implemented by the user through external programming.

The input \( fIn \) is automatically limited to the range \( 0..100\% \) internally.

This also applies to the entries \( fHys \) and \( fRefVal \). The travel times \( nTiCls \) and \( nTiOpn \) both have a lower limit value of 10 (milliseconds).

A rising edge at \( bRef \) triggers a referencing command (the calculated actual position is set to \( fRefVal \)).

If the drive has limit switches, they can be sampled directly via the digital input and used for referencing at \( bRef \).

**VAR_INPUT**

\[
\begin{align*}
\text{fIn} &: \text{REAL}; \\
\text{fHys} &: \text{REAL}; \\
\text{nTiCls} &: \text{UDINT}; \\
\text{nTiOpn} &: \text{UDINT}; \\
\text{bRef} &: \text{BOOL}; \\
\text{fRefVal} &: \text{REAL}; \\
\text{bCloseInit} &: \text{BOOL};
\end{align*}
\]

- \( \text{fIn} \): Setpoint for the actuator position \([0…100 \%]\). Internally limited to values between 0 and 100.
- \( \text{fHys} \): Hysteresis for the actuator position \([0…100 \%]\). Internally limited to values between 0 and 100.
- \( \text{nTiCls} \): Run time of the actuator from open to closed [ms]. Internally limited to values between 0 and 100.
- \( \text{nTiOpn} \): Run time of the actuator from closed to open [ms]. Internally limited to values between 0 and 100.
- \( \text{bRef} \): Edge references the internal position memory of the drive to value of \( fRefVal \) \([0…100 \%]\).
- \( \text{fRefVal} \): Value for referencing the actuator with \( bRef \) \([0…100 \%]\). Internally limited to values between 0 and 100.
- \( \text{bCloseInit} \): If this input is TRUE, output \( bCls \) is TRUE for the time \( udiTiOpn_{\text{ms}} \)

**VAR_OUTPUT**

\[
\begin{align*}
\text{bCls} &: \text{BOOL}; \\
\text{bOpn} &: \text{BOOL}; \\
\text{fPos} &: \text{REAL};
\end{align*}
\]

- \( \text{bCls} \): Output for closing the actuator.
- \( \text{bOpn} \): Output for opening the actuator.
- \( \text{fPos} \): Current calculated actuator position \([0…100 \%]\).

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

TF8040 Version: 1.4
This function block prevents blocking of pumps or actuators after prolonged idle periods by issuing a switch-on pulse. The maximum idle period before such a pulse is issued is determined by the value of \( nTiOffMin \). For logging the idle time, the input \( bFdb \) must be linked to the operating feedback from the unit. The pulse length is parameterized with \( nTiImplLngt \). The input \( bExe \) should be used if the blocking protection pulses are to be issued cyclically based on a switching schedule, rather than depending on the idle times. A rising edge at \( bExe \) immediately triggers output of a pulse to \( bQ \). Generally, a pulse output only occurs if the function block at \( bEn \) is enabled.

**VAR_INPUT**

\[
\begin{align*}
  bEn & : \text{BOOL}; \\
  bFdb & : \text{BOOL}; \\
  bExe & : \text{BOOL}; \\
  nTiOffMin & : \text{UDINT}; \\
  nTiImplLngt & : \text{UDINT};
\end{align*}
\]

- **bEn**: Enable function block.
- **bFdb**: Input for connecting the feedback signal of a motor or valve.
- **bExe**: Rising edge forces a pulse output.
- **nTiOffMin**: Minimum switch-off time [s]: A pulse is issued once the time \( nTiOffMin \) has elapsed without movement of the unit.
- **nTiImplLngt**: Length of the blocking protection pulse [s] at \( bQ \).

**VAR_OUTPUT**

\[
\begin{align*}
  bQ & : \text{BOOL}; \\
  nRTiOffMin & : \text{UDINT}; \\
  nRTiImplLngt & : \text{UDINT};
\end{align*}
\]

- **bQ**: Pulse output.
- **nRTiOffMin**: Remaining time [s] before the next pulse is issued in the absence of movement.
- **nRTiImplLngt**: Remaining residual time [s] of the pulse at \( bQ \).

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

**FB_BA_Motor1St**

Function block for controlling a simple single-stage motor.
The input \textit{bEn} is used for general enabling of the motor.

The input \textit{nOpMode} is used to set the operation mode of the motor:

- 0 = Automatic
- 1 = Manual off
- 2 = Manual on

In automatic mode (\textit{nOpMode} = 0) the motor can be operated via the input \textit{bAuto} (\textit{bAuto} = \textit{bQ} = TRUE).

The collection of all possible malfunctions of a motor is connected to \textit{bDst}.

\textbf{VAR\_INPUT}

\begin{verbatim}
bEn : BOOL;
bAuto : BOOL;
bDst : BOOL;
nOpMode : UDI NT;
\end{verbatim}

\textbf{bEn}: Enable motor.

\textbf{bAuto}: Request of the actuator in automatic mode (\textit{nOpMode} = 0).

\textbf{bDst}: Input for collecting the possible motor malfunctions.

\textbf{nOpMode}: Select the operation mode (0 = Automatic, 1 = Manual off, 2 = Manual on).

\textbf{VAR\_OUTPUT}

\begin{verbatim}
bQ : BOOL;
\end{verbatim}

\textbf{bQ}: Control output.

\textbf{Requirements}

\begin{tabular}{|l|l|}
\hline
Development environment & Required PLC library \\
\hline
TwinCAT from v3.1.4024.17 & Tc3\_BA2 from v4.8.9.0 \\
\hline
\end{tabular}

\textbf{FB\_BA\_Motor2St}

Function block for controlling a simple two-stage motor.

The input \textit{bEn} is used for general enabling of the motor.

The input \textit{nOpMode} is used to set the operation mode of the motor:

- 0 = Automatic
- 1 = Manual off
- 2 = Manual stage 1
- 3 = Manual stage 2

In automatic mode (\textit{nOpMode} = 0) the desired level can be set via the inputs \textit{bAutoSt1} (stage 1) and \textit{bAutoSt2} (stage 2).

The collection of all possible malfunctions of a motor is connected to \textit{bDst}. 
VAR_INPUT
bEn : BOOL;
bAutoSt1 : BOOL;
bAutoSt2 : BOOL;
bDst : BOOL;
nOpMode : UDINT;

bEn: Enable motor.
bAutoSt1: Actuator request at level 1 in automatic mode (nOpMode = 0).
bAutoSt2: Actuator request at level 2 in automatic mode (nOpMode = 0).
bDst: Input for collecting the possible motor malfunctions.
nOpMode: Select the operation mode (0 = Automatic, 1 = Manual off, 2 = Manual stage 1, 3 = Manual stage 2).

VAR_OUTPUT
bQ1 : BOOL;
bQ2 : BOOL;

bQ1: Control output stage 1.
bQ2: Control output stage 2.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

FB_BA_PWM

The function block calculates switch-on and switch-off times fActTiOn and fActTiOff [s] from an analog input signal rIn (0..100%, internally limited) and a period nPrd [s].

The following relationships apply:

- 100% at the input of a switch-on time fActTiOn of the total period nPrd and a switch-off time fActTiOff of 0 s
- 0% at the input of a switch-on time fActTiOn of the total period nPrd and a switch-off time fActTiOff of 0 s.

In addition, nMinSwiTi [s] can be used to set a lower limit for the switching time, in order to prevent damage to drives caused by too short actuating pulses. This behavior is only valid for 0 > fin >100.

If fin = 0 or 100, the output bQ remains deleted or set. After the period time has elapsed, the current input signal is evaluated again. If it is still set to 0 or 100, there is no change of state of bQ.

Switching characteristics

1. A FALSE signal at input bEn disables the function block and sets bQ to FALSE. Only the switch-on and switch-off times are continuously calculated and displayed at the outputs fActTiOn fActTiOff [s].
2. A rising edge at input $bEn$ enables the function block: It will initially jump to a decision step. Depending on the previous state of the switching output $bQ$, the switching step is now accessed. However, if the input $fIn$ is set to 0, an immediate jump occurs to the Off step ($bQ$=FALSE), or to the On step if $fIn$=100 ($bQ$=TRUE), irrespective of the previous state of $bQ$. The minimum switching time is deactivated for these two cases.

3. A countdown timer with the current calculated starting value runs in the respective active step (ON or OFF), which is based on the pulse/pause ratio. The on- or off-step is completed with the calculated time, irrespective of whether the pulse/pause ratio changes in the meantime. The respective countdown is displayed at the outputs $nRemTiOn$ / $nRemTiOff$ in full seconds.

4. Completion of the on- or off-step is followed by a jump back to the decision step (point 2).

**VAR_INPUT**

<table>
<thead>
<tr>
<th>bEn</th>
<th>BOOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>fIn</td>
<td>REAL</td>
</tr>
<tr>
<td>nPrd</td>
<td>UDINT</td>
</tr>
<tr>
<td>nMinSwiTi</td>
<td>UDINT</td>
</tr>
</tbody>
</table>

- **bEn**: Activation of pulse width modulation.
- **fIn**: Input signal, internally limited to 0...100%.
- **nPrd**: Period time[s]. Internally limited to a minimum value of 0.
- **nMinSwiTi**: Minimum switch-on time [s], to avoid too short pulses. Internally limited to values between 0 and $nPrd$.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>bQ</th>
<th>BOOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>bLmtSwiTi</td>
<td>BOOL</td>
</tr>
<tr>
<td>fActTiOn</td>
<td>REAL</td>
</tr>
<tr>
<td>fActTiOff</td>
<td>REAL</td>
</tr>
<tr>
<td>nRemTiOn</td>
<td>UDINT</td>
</tr>
<tr>
<td>nRemTiOff</td>
<td>UDINT</td>
</tr>
</tbody>
</table>

- **bQ**: PWM output.
- **bLmtSwiTi**: Information output to indicate that the input signal is so low that the minimum switch-on time is used as limit.
- **fActTiOn**: Information output: calculated switch-on time [s].
- **fActTiOff**: Information output: calculated switch-off time [s].
- **nRemTiOn**: Switch-on timer countdown [s].
- **nRemTiOff**: Switch-off timer countdown [s].

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
Demultiplexer function blocks exist for different variable types (BOOL, INT, LREAL, REAL, USINT, UINT, UDINT and DINT) and in different output values (4, 8, 12 and 16), but they all have the same functionality. The function block FB_BA_DMUX_LR16 is described as an example.

In active state \((bEn=\text{TRUE})\), the function block outputs the value at input \(fIn\) at the output \(fQ01..fQ16\) whose number is entered at input \(nSel\). All other outputs are set to 0 (for boolean demultiplexers to FALSE).

Sample:

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>(bEn = \text{TRUE})</td>
<td>(fQ01 = 0.0)</td>
</tr>
<tr>
<td>(nSel = 5)</td>
<td>(fQ02 = 0.0)</td>
</tr>
<tr>
<td>(fIn = 32.5)</td>
<td>(fQ03 = 0.0)</td>
</tr>
<tr>
<td></td>
<td>(fQ04 = 0.0)</td>
</tr>
<tr>
<td></td>
<td>(fQ05 = 32.5)</td>
</tr>
<tr>
<td></td>
<td>(fQ06 = 0.0)</td>
</tr>
<tr>
<td></td>
<td>(fQ07 = 0.0)</td>
</tr>
<tr>
<td></td>
<td>(fQ08 = 0.0)</td>
</tr>
<tr>
<td></td>
<td>(fQ09 = 0.0)</td>
</tr>
<tr>
<td></td>
<td>(fQ10 = 0.0)</td>
</tr>
<tr>
<td></td>
<td>(fQ11 = 0.0)</td>
</tr>
<tr>
<td></td>
<td>(fQ12 = 0.0)</td>
</tr>
<tr>
<td></td>
<td>(fQ13 = 0.0)</td>
</tr>
<tr>
<td></td>
<td>(fQ14 = 0.0)</td>
</tr>
<tr>
<td></td>
<td>(fQ15 = 0.0)</td>
</tr>
<tr>
<td></td>
<td>(fQ16 = 0.0)</td>
</tr>
</tbody>
</table>

If the value entered at \(nSel\) is greater than the number of outputs, the value of \(fIn\) is output at the "highest" output.
Inputs | Outputs
---|---
`bEn = TRUE` | `fQ01 = 0.0`
`nSel = 25` | `fQ02 = 0.0`
`fIn = 32.5` | `fQ03 = 0.0`

If `bEn = FALSE`, 0.0 is output at all outputs, or FALSE for boolean demultiplexers.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bEn</code></td>
<td>Activation of the block function.</td>
</tr>
<tr>
<td><code>nSel</code></td>
<td>Number of the output <code>fQ01...fQ16</code>, which is to assume the value of input <code>fIn</code>.</td>
</tr>
<tr>
<td><code>fIn</code></td>
<td>Value to be output.</td>
</tr>
</tbody>
</table>

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fQ01</code></td>
<td>Value outputs.</td>
</tr>
<tr>
<td><code>fQ02</code></td>
<td></td>
</tr>
<tr>
<td><code>fQ03</code></td>
<td></td>
</tr>
<tr>
<td><code>fQ04</code></td>
<td></td>
</tr>
<tr>
<td><code>fQ05</code></td>
<td></td>
</tr>
<tr>
<td><code>fQ06</code></td>
<td></td>
</tr>
<tr>
<td><code>fQ07</code></td>
<td></td>
</tr>
<tr>
<td><code>fQ08</code></td>
<td></td>
</tr>
<tr>
<td><code>fQ09</code></td>
<td></td>
</tr>
<tr>
<td><code>fQ10</code></td>
<td></td>
</tr>
<tr>
<td><code>fQ11</code></td>
<td></td>
</tr>
<tr>
<td><code>fQ12</code></td>
<td></td>
</tr>
<tr>
<td><code>fQ13</code></td>
<td></td>
</tr>
<tr>
<td><code>fQ14</code></td>
<td></td>
</tr>
<tr>
<td><code>fQ15</code></td>
<td></td>
</tr>
<tr>
<td><code>fQ16</code></td>
<td></td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
The function block activates an input value on the output, depending on a selector and the corresponding input selector condition.

Multiplexer function blocks exist for different variable types (BOOL, INT, LREAL, REAL, USINT, UINT, UDINT and DINT) and in different input values (4, 8, 12, 16 and 24), but they all have the same functionality. The function block FB_BA_MMUX_R16 is described as an example.

In active state \( bEn = \text{TRUE} \), the function block activates one of the input values \( fValxx \) at output \( fVal \), depending on a selector \( nSel \) and the corresponding input selector condition \( nEnxx \).

If several input selector conditions \( nEn01...nEn16 \) are identical and the selector \( nSel \) matches a condition, the input value \( fVal01...fVal16 \) of the lowest active selector condition is activated at output \( fVal \). \( nEn01 \) is the lowest selector condition, \( nEn16 \) the highest.

The output variable \( bQ \) indicates that the selector \( nSel \) matches an input selector condition \( nEnxx \).

The output variable \( nActvPrio \) indicates the active selector condition.
If no selector condition is active, $f\text{ReplVal}$ is output at $f\text{Val}$. In this case $bQ$ is FALSE and $n\text{ActvPrio}$ shows 255.

Sample:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b\text{En}$</td>
<td>TRUE</td>
<td>$b\text{Q}$</td>
<td>TRUE</td>
</tr>
<tr>
<td>$n\text{Sel}$</td>
<td>5</td>
<td>$f\text{Val}$</td>
<td>1.123</td>
</tr>
<tr>
<td>$n\text{En01}$</td>
<td>4</td>
<td>$n\text{ActvPrio}$</td>
<td>7</td>
</tr>
<tr>
<td>$f\text{Val01}$</td>
<td>123</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n\text{En02}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f\text{Val02}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n\text{En03}$</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f\text{Val03}$</td>
<td>321</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n\text{En04}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f\text{Val04}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n\text{En05}$</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f\text{Val05}$</td>
<td>345</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n\text{En06}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f\text{Val06}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n\text{En07}$</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f\text{Val07}$</td>
<td>1.123</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n\text{En08}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f\text{Val08}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n\text{En09}$</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f\text{Val09}$</td>
<td>5.4321</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n\text{En10}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f\text{Val10}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n\text{En11}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f\text{Val11}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n\text{En12}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f\text{Val12}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n\text{En13}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f\text{Val13}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n\text{En14}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f\text{Val14}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n\text{En15}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f\text{Val15}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n\text{En16}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f\text{Val16}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f\text{ReplVal}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If no priority is active, the value of the global constant $\text{BA\_Globals.nNoActivePrio}$ is output at $n\text{ActvPrio}$.

VAR_INPUT

$b\text{En}$ : BOOL
$n\text{Sel}$ : UDINT
$n\text{En01}$ : UDINT := $\text{BA\_Globals.nNoActvPrio}$
$f\text{Val01}$ : REAL
$n\text{En02}$ : UDINT := $\text{BA\_Globals.nNoActvPrio}$
$f\text{Val02}$ : REAL
$n\text{En03}$ : UDINT := $\text{BA\_Globals.nNoActvPrio}$
$f\text{Val03}$ : REAL
$n\text{En04}$ : UDINT := $\text{BA\_Globals.nNoActvPrio}$
VAR_INPUT

fVal01  :  REAL;

nEn01   :  UDINT := BA_Globals.nNoActvPrio;
fVal02  :  REAL;
nEn02   :  UDINT := BA_Globals.nNoActvPrio;
fVal03  :  REAL;
nEn03   :  UDINT := BA_Globals.nNoActvPrio;
fVal04  :  REAL;
nEn04   :  UDINT := BA_Globals.nNoActvPrio;
fVal05  :  REAL;
nEn05   :  UDINT := BA_Globals.nNoActvPrio;
fVal06  :  REAL;
nEn06   :  UDINT := BA_Globals.nNoActvPrio;
fVal07  :  REAL;
nEn07   :  UDINT := BA_Globals.nNoActvPrio;
fVal08  :  REAL;
nEn08   :  UDINT := BA_Globals.nNoActvPrio;
fVal09  :  REAL;
nEn09   :  UDINT := BA_Globals.nNoActvPrio;
fVal10  :  REAL;
nEn10   :  UDINT := BA_Globals.nNoActvPrio;
fVal11  :  REAL;
nEn11   :  UDINT := BA_Globals.nNoActvPrio;
fVal12  :  REAL;
nEn12   :  UDINT := BA_Globals.nNoActvPrio;
fVal13  :  REAL;
nEn13   :  UDINT := BA_Globals.nNoActvPrio;
fVal14  :  REAL;
nEn14   :  UDINT := BA_Globals.nNoActvPrio;
fVal15  :  REAL;
nEn15   :  UDINT := BA_Globals.nNoActvPrio;
fVal16  :  REAL;
nEn16   :  UDINT := BA_Globals.nNoActvPrio;

VAR_OUTPUT

bQ      :  BOOL;

fVal    :  REAL;
nActvPrio: UDINT;

bQ: TRUE if the selector nSel matches an input selector condition nEnxx.

fVal: Value of the selected input selector condition.

nActvPrio: Indicates which input selector condition is active. If no priority is active, the value of the global constant BA_Globals.nNoActivePrio is output at nActvPrio.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
Multiplexer function blocks exist for different variable types (BOOL, INT, LREAL, REAL, USINT, UINT, UDINT and DINT) and in different input values (4, 8, 12 and 16), but they all have the same functionality. The function block FB_BA_MUX_LR16 is described as an example.

In active state \((bEn=\text{TRUE})\), the function block outputs the input value \(fIn01..fIn16\) at output \(fQ\), whose number is entered at input \(nSel\).

Sample:

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEn = TRUE</td>
<td>fQ = 16.5</td>
</tr>
<tr>
<td>nSel = 5</td>
<td></td>
</tr>
<tr>
<td>fIn01 = 15.9</td>
<td></td>
</tr>
<tr>
<td>fIn02 = 32.5</td>
<td></td>
</tr>
<tr>
<td>fIn03 = 17.4</td>
<td></td>
</tr>
<tr>
<td>fIn04 = 5.84</td>
<td></td>
</tr>
<tr>
<td>fIn05 = 9.56</td>
<td></td>
</tr>
<tr>
<td>fIn06 = 16.5</td>
<td></td>
</tr>
<tr>
<td>fIn07 = 32.781</td>
<td></td>
</tr>
<tr>
<td>fIn08 = 25.4</td>
<td></td>
</tr>
<tr>
<td>fIn09 = 44.5</td>
<td></td>
</tr>
<tr>
<td>fIn10 = 66.1</td>
<td></td>
</tr>
<tr>
<td>fIn11 = 45.5</td>
<td></td>
</tr>
<tr>
<td>fIn12 = 83.3</td>
<td></td>
</tr>
<tr>
<td>fIn13 = 54.56</td>
<td></td>
</tr>
<tr>
<td>fIn14 = 33.8</td>
<td></td>
</tr>
<tr>
<td>fIn15 = 98.5</td>
<td></td>
</tr>
<tr>
<td>fIn16 = 71.3</td>
<td></td>
</tr>
</tbody>
</table>

If the value entered at \(nSel\) is greater than the number of inputs, the "highest-ranking" input is output at \(fQ\):
### Inputs

<table>
<thead>
<tr>
<th>bEn = TRUE</th>
<th>fQ = 2.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSel = 25</td>
<td></td>
</tr>
<tr>
<td>fIn01 = 15.9</td>
<td></td>
</tr>
<tr>
<td>fIn02 = 32.5</td>
<td></td>
</tr>
<tr>
<td>fIn03 = 17.4</td>
<td></td>
</tr>
<tr>
<td>fIn04 = 5.84</td>
<td></td>
</tr>
<tr>
<td>fIn05 = 9.56</td>
<td></td>
</tr>
<tr>
<td>fIn06 = 16.5</td>
<td></td>
</tr>
<tr>
<td>fIn07 = 32.781</td>
<td></td>
</tr>
<tr>
<td>fIn08 = 25.4</td>
<td></td>
</tr>
<tr>
<td>fIn09 = 44.5</td>
<td></td>
</tr>
<tr>
<td>fIn10 = 66.1</td>
<td></td>
</tr>
<tr>
<td>fIn11 = 45.5</td>
<td></td>
</tr>
<tr>
<td>fIn12 = 83.3</td>
<td></td>
</tr>
<tr>
<td>fIn13 = 54.56</td>
<td></td>
</tr>
<tr>
<td>fIn14 = 33.8</td>
<td></td>
</tr>
<tr>
<td>fIn15 = 98.5</td>
<td></td>
</tr>
<tr>
<td>fIn16 = 71.3</td>
<td></td>
</tr>
</tbody>
</table>

If \( bEn=\text{FALSE} \), 0.0 is output at \( fQ \), or FALSE for boolean multiplexers.

### VAR_INPUT

<table>
<thead>
<tr>
<th>bEn : BOOL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSel : UDINT;</td>
</tr>
<tr>
<td>fIn01 : LREAL;</td>
</tr>
<tr>
<td>fIn02 : LREAL;</td>
</tr>
<tr>
<td>fIn03 : LREAL;</td>
</tr>
<tr>
<td>fIn04 : LREAL;</td>
</tr>
<tr>
<td>fIn05 : LREAL;</td>
</tr>
<tr>
<td>fIn06 : LREAL;</td>
</tr>
<tr>
<td>fIn07 : LREAL;</td>
</tr>
<tr>
<td>fIn08 : LREAL;</td>
</tr>
<tr>
<td>fIn09 : LREAL;</td>
</tr>
<tr>
<td>fIn10 : LREAL;</td>
</tr>
<tr>
<td>fIn11 : LREAL;</td>
</tr>
<tr>
<td>fIn12 : LREAL;</td>
</tr>
<tr>
<td>fIn13 : LREAL;</td>
</tr>
<tr>
<td>fIn14 : LREAL;</td>
</tr>
<tr>
<td>fIn15 : LREAL;</td>
</tr>
<tr>
<td>fIn16 : LREAL;</td>
</tr>
</tbody>
</table>

**bEn**: Activation of the block function.

**nSel**: Number of the input, whose value is to be output at \( fQ \).

**f01...f16**: Input values to select from.

### VAR_OUTPUT

| fQ : LREAL; |

**fQ**: Value of the selected input.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
FB_BA_PrioSwi_XX

The priority switches exist for different variable types (BOOL, INT, LREAL, REAL, USINT, UINT, UDINT and DINT) and in different output values (4, 8, 12 and 16 or 24), but they all have the same functionality. The function block FB_BA_PrioSwi_LR08 is described as an example.

Priority switches are available for selecting different values. At output \( fVal \) the value with the highest priority is applied whose input \( bEnxx \) is TRUE.

Sample:

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( bEn01 )</td>
<td>FALSE</td>
</tr>
<tr>
<td>( fVal01 )</td>
<td>32.5</td>
</tr>
<tr>
<td>( bEn02 )</td>
<td>FALSE</td>
</tr>
<tr>
<td>( fVal02 )</td>
<td>17.4</td>
</tr>
<tr>
<td>( bEn03 )</td>
<td>TRUE</td>
</tr>
<tr>
<td>( fVal03 )</td>
<td>5.84</td>
</tr>
<tr>
<td>( bEn04 )</td>
<td>TRUE</td>
</tr>
<tr>
<td>( fVal04 )</td>
<td>9.56</td>
</tr>
<tr>
<td>( bEn05 )</td>
<td>FALSE</td>
</tr>
<tr>
<td>( fVal05 )</td>
<td>16.5</td>
</tr>
<tr>
<td>( bEn06 )</td>
<td>TRUE</td>
</tr>
<tr>
<td>( fVal06 )</td>
<td>32.781</td>
</tr>
<tr>
<td>( bEn07 )</td>
<td>FALSE</td>
</tr>
<tr>
<td>( fVal07 )</td>
<td>25.4</td>
</tr>
<tr>
<td>( bEn08 )</td>
<td>TRUE</td>
</tr>
<tr>
<td>( fVal08 )</td>
<td>44.5</td>
</tr>
</tbody>
</table>

If none of the priorities is enabled, the output \( bQ \) switches to FALSE. 0 is output at \( fVal \) and \( nActvPrio \). For a boolean priority switch, FALSE is then output at \( bVal \).
If no priority is active, the value of the global constant is output at `nActvPrio`. 

**VAR_INPUT**

<table>
<thead>
<tr>
<th>bEn01</th>
<th>bEn02</th>
<th>bEn03</th>
<th>bEn04</th>
<th>bEn05</th>
<th>bEn06</th>
<th>bEn07</th>
<th>bEn08</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>bQ</th>
<th>fVal</th>
<th>nActvPrio</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>0.0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
FB_BA_Blink

This function block is an oscillator with configurable pulse and pause time, \( nTiOn \) and \( nTiOff \) [ms]. It is enabled with a TRUE signal at \( bEn \) and starts with the pulse phase.

\[ nTiNextSwi \] is a countdown [s] to the next change of \( bQ \).

**VAR_INPUT**

- \( bEn \) : BOOL;
- \( nTiOn \) : UDINT;
- \( nTiOff \) : UDINT;

\( bEn \): Function block enable.
\( nTiOn \): pulse time [ms].
\( nTiOff \): pause time [ms].

**VAR_OUTPUT**

- \( bQ \) : BOOL;
- \( nTiNextSwi \) : UDINT;

\( bQ \): Oscillator output.
\( nTiNextSwi \): Countdown to next change of \( bQ \) [s].

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
The function block FB_BA_FIFO04 enables sequential control of up to four units, with automatic switching of the switch-on sequence based on operating hours. The function block is available in two versions: for a sequence of four or eight units.

Units with fewer operating hours take precedence in the sequence over units with more operating hours.

A rising edge at \texttt{bChg} forces a sequence change. The units with the fewest operating hours are set to the top of the FIFO and thus given priority for switching on. In the sequence only units are entered, which are enabled at inputs \texttt{bEn01}...\texttt{bEn04}. \texttt{nNum} indicates the number of requested units.

The operating hours of the units are entered at inputs \texttt{nActvTi01} to \texttt{nActvTi04}. If all these inputs are set to a constant value of zero, the sequence change is controlled cyclically, depending on \texttt{bChg}.

The first unit is removed from the FIFO, the other units are advanced, and the first unit is appended at the end of the FIFO again. As a result is an alternating sequence of units.

**VAR_INPUT**

- \texttt{bEn}: Enables the function block.
- \texttt{nNum}: Number of units.
- \texttt{bChg}: Force sequence change.
- \texttt{bEn01}...\texttt{bEn04}: Enable unit 1...enable unit 4.
- \texttt{nActvTi01}...\texttt{nActvTi04}: Operating hours unit 1...operating hours unit 4.

**VAR_OUTPUT**

- \texttt{bQ01}...\texttt{bQ04}: Switches unit 1...4.
- \texttt{nNextOn}: Number of the unit that is switched on next.
- \texttt{nNextOff}: Number of the unit that is switched on next.

\texttt{VAR_INPUT}

\begin{verbatim}
  bEn         : BOOL;
  nNum        : UDINT;
  bChg        : BOOL;
  bEn01       : BOOL;
  bEn02       : BOOL;
  bEn03       : BOOL;
  bEn04       : BOOL;
  nActvTi01   : UDINT;
  nActvTi02   : UDINT;
  nActvTi03   : UDINT;
  nActvTi04   : UDINT;
\end{verbatim}

\texttt{VAR_OUTPUT}

\begin{verbatim}
  bQ01         : BOOL;
  bQ02         : BOOL;
  bQ03         : BOOL;
  bQ04         : BOOL;
  nNextOn      : UDINT;
  nNextOff     : UDINT;
  aFIFO        : ARRAY [1..4] OF UDINT;
  nNumOfEn     : UDINT;
\end{verbatim}
aFIFO: FIFO buffer as a field.
nNumOfEn: Number of devices, depending on the individual enable states.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

**FB_BA_FIFO08**

The function block `FB_BA_FIFO08` enables sequential control of up to eight units, with automatic switching of the switch-on sequence based on operating hours.

The function block is available in two versions: for a sequence of four [216] or eight units.

Units with fewer operating hours take precedence in the sequence over units with more operating hours.

A rising edge at `bChg` forces a sequence change. The units with the fewest operating hours are set to the top of the FIFO and thus given priority for switching on.

In the sequence only units are entered, which are enabled at inputs `bEn01…bEn08`. `nNum` indicates the number of requested units.

The operating hours of the units are entered at inputs `nActvTi01` to `nActvTi08`. If all these inputs are set to a constant value of zero, the sequence change is controlled cyclically, depending on `bChg`.

The first unit is removed from the FIFO, the other units are advanced, and the first unit is appended at the end of the FIFO again. As a result is an alternating sequence of units.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEn</td>
<td>BOOL</td>
</tr>
<tr>
<td>nNum</td>
<td>UDINT</td>
</tr>
<tr>
<td>bChg</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEn01</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEn02</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEn03</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEn04</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEn05</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEn06</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEn07</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEn08</td>
<td>BOOL</td>
</tr>
<tr>
<td>nActvTi01</td>
<td>UDINT</td>
</tr>
<tr>
<td>nActvTi02</td>
<td>UDINT</td>
</tr>
<tr>
<td>nActvTi03</td>
<td>UDINT</td>
</tr>
<tr>
<td>nActvTi04</td>
<td>UDINT</td>
</tr>
<tr>
<td>nActvTi05</td>
<td>UDINT</td>
</tr>
<tr>
<td>nActvTi06</td>
<td>UDINT</td>
</tr>
<tr>
<td>nActvTi07</td>
<td>UDINT</td>
</tr>
<tr>
<td>nActvTi08</td>
<td>UDINT</td>
</tr>
</tbody>
</table>
**bEn**: Enables the function block.

**nNum**: Number of units.

**bChg**: Force sequence change.

**bEn01...bEn08**: Enable unit 1...enable unit 8.

**nActvTi01...nActvTi08**: Operating hours unit 1...operating hours unit 8.

### VAR_OUTPUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bQ01</td>
<td>BOOL</td>
</tr>
<tr>
<td>bQ02</td>
<td>BOOL</td>
</tr>
<tr>
<td>bQ03</td>
<td>BOOL</td>
</tr>
<tr>
<td>bQ04</td>
<td>BOOL</td>
</tr>
<tr>
<td>bQ05</td>
<td>BOOL</td>
</tr>
<tr>
<td>bQ06</td>
<td>BOOL</td>
</tr>
<tr>
<td>bQ07</td>
<td>BOOL</td>
</tr>
<tr>
<td>bQ08</td>
<td>BOOL</td>
</tr>
<tr>
<td>nNextOn</td>
<td>UDINT</td>
</tr>
<tr>
<td>nNextOff</td>
<td>UDINT</td>
</tr>
<tr>
<td>aFIFO</td>
<td>ARRAY [1..8]  OF UDINT</td>
</tr>
<tr>
<td>nNumOfEn</td>
<td>UDINT</td>
</tr>
</tbody>
</table>

**bQ01...bQ08**: Switches unit 1...8.

**nNextOn**: Number of the unit that is switched on next.

**nNextOff**: Number of the unit that is switched on next.

**aFIFO**: FIFO buffer as a field.

**nNumOfEn**: Number of devices, depending on the individual enable states.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
The function block is used for issuing sequential control commands. A typical application for this function block is startup of an air conditioning system. \( bEn \) is used for general enable of the function block. If \( bEn = \) FALSE, all outputs of \( bQ01 \) to \( bQ08 \) are set to FALSE. The control sequence starts at input \( bEvt01 \). Once the timer \( nDlyOn01 \) (see Parameters) has elapsed, the corresponding output \( bQ01 \) is set. Further stages are activated after a rising edge at the inputs \( bEvt02 \) to \( bEvt08 \), in each case delayed via the timers \( nDlyOn02 \) to \( nDlyOn08 \). If \( bEvt01 \) becomes FALSE once the control chain is up and running, the control sequence switches back in reverse order. Switching off of the outputs is delayed by the timers \( nDlyOff01 \) to \( nDlyOff08 \); see Parameters.

The outputs \( bUp \) and \( bDwn \) indicate whether the control chain is in ascending or descending state. The variable \( nActvEvt \) indicates the current step of the control chain. "0" means the step sequence is not active.

The output \( nRemTiDlyOn \) indicates the time remaining to the next step during up-switching of the control chain. The output \( nRemTiDlyOff \) indicates the time remaining to the next lower step during down-switching of the control chain.
Sample

- t0 step sequence switch-on
- t1 switch on step 1 \( nDlyOn01 = t1 - t0 \)
- t2 event enable step 2, switch on step 2, \( nDlyOn02 = 0 \)
- t3 event enable step 3, switch on step 3, \( nDlyOn03 = 0 \)
- t4 event enable step 4, switch on step 4, \( nDlyOn04 = 0 \)
- t5 event enable step 5, switch on step 5, \( nDlyOn05 = 0 \)
- t6 disable the step sequence, disable step 5, disable step 4; \( nDlyOff05 = 0, nDlyOff04 = 0 \)
- t7 switch off step 3, \( nDlyOff03 = t7 - t6 \)
- t8 switch off step 2, \( nDlyOff02 = t8 - t7 \)
- t9 switch off step 1, \( nDlyOff01 = t9 - t8 \)

**VAR_INPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEn</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEvt01</td>
<td>BOOL</td>
</tr>
<tr>
<td>nDlyOn01</td>
<td>UDIINT</td>
</tr>
<tr>
<td>nDlyOff01</td>
<td>UDIINT</td>
</tr>
<tr>
<td>bEvt02</td>
<td>BOOL</td>
</tr>
<tr>
<td>nDlyOn02</td>
<td>UDIINT</td>
</tr>
<tr>
<td>nDlyOff02</td>
<td>UDIINT</td>
</tr>
<tr>
<td>bEvt03</td>
<td>BOOL</td>
</tr>
<tr>
<td>nDlyOn03</td>
<td>UDIINT</td>
</tr>
<tr>
<td>nDlyOff03</td>
<td>UDIINT</td>
</tr>
<tr>
<td>bEvt04</td>
<td>BOOL</td>
</tr>
<tr>
<td>nDlyOn04</td>
<td>UDIINT</td>
</tr>
<tr>
<td>nDlyOff04</td>
<td>UDIINT</td>
</tr>
<tr>
<td>bEvt05</td>
<td>BOOL</td>
</tr>
<tr>
<td>nDlyOn05</td>
<td>UDIINT</td>
</tr>
<tr>
<td>nDlyOff05</td>
<td>UDIINT</td>
</tr>
<tr>
<td>bEvt06</td>
<td>BOOL</td>
</tr>
<tr>
<td>nDlyOn06</td>
<td>UDIINT</td>
</tr>
<tr>
<td>nDlyOff06</td>
<td>UDIINT</td>
</tr>
<tr>
<td>bEvt07</td>
<td>BOOL</td>
</tr>
<tr>
<td>nDlyOn07</td>
<td>UDIINT</td>
</tr>
</tbody>
</table>
nDlyOff07 : UDINT;
bEvt08 : BOOL;
nDlyOn08 : UDINT;
nDlyOff08 : UDINT;

bEn: Enable function block.

bEvt01...08: Switch-on command for steps 1 to 8.

nDlyOn01...08: Start-up delay for output bQ01...08 [s].

nDlyOff01...08: Switch-off delay for output bQ01...08 [s].

VAR_OUTPUT

bQ01 : BOOL;
bQ02 : BOOL;
bQ03 : BOOL;
bQ04 : BOOL;
bQ05 : BOOL;
bQ06 : BOOL;
bQ07 : BOOL;
bQ08 : BOOL;
bUp : BOOL;
bDwn : BOOL;
nActvEvt : UDINT;
nRemTiDlyOn : UDINT;
nRemTiDlyOff : UDINT;

bQ01...08: Step 1 to 8 On.

bUp: Control chain is in ascending state.

bDwn: Control chain is in descending state.

nActvEvt: Active step, display 0...8; "0" represents a non-active step sequence.

nRemTiDlyOn: Time remaining to up-switching to the next step [s].

nRemTiDlyOff: Time remaining to down-switching to the previous step [s].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
The function block is used for issuing sequential control commands. A typical application for this function block is startup of an air conditioning system. \( b\text{En} \) is used for general enable of the function block. If \( b\text{En} = \text{FALSE} \), all outputs of \( b\text{Q01} \) to \( b\text{Q12} \) are set to \text{FALSE}. The control sequence starts at input \( b\text{Evt01} \). Once the timer \( n\text{DlyOn01} \) (see Parameters) has elapsed, the corresponding output \( b\text{Q01} \) is set. Further stages are activated after a rising edge at the inputs \( b\text{Evt02} \) to \( b\text{Evt12} \), in each case delayed via the timers \( n\text{DlyOn02} \) to \( n\text{DlyOn12} \). If \( b\text{Evt01} \) becomes \text{FALSE} once the control chain is up and running, the control sequence switches back in reverse order. Switching off of the outputs is delayed by the timers \( n\text{DlyOff01} \) to \( n\text{DlyOff12} \); see Parameters.

The outputs \( b\text{Up} \) and \( b\text{Dwn} \) indicate whether the control chain is in ascending or descending state. The variable \( n\text{ActvEvt} \) indicates the current step of the control chain. "0" means the step sequence is not active.

The output \( n\text{RemTiDlyOn} \) indicates the time remaining to the next step during up-switching of the control chain. The output \( n\text{RemTiDlyOff} \) indicates the time remaining to the next lower step during down-switching of the control chain.
Sample

- t0 step sequence switch-on
- t1 switch on step 1 \( nDlyOn01 = t1 - t0 \)
- t2 event enable step 2, switch on step 2, \( nDlyOn02 = 0 \)
- t3 event enable step 3, switch on step 3, \( nDlyOn03 = 0 \)
- t4 event enable step 4, switch on step 4, \( nDlyOn04 = 0 \)
- t5 event enable step 5, switch on step 5, \( nDlyOn05 = 0 \)
- t6 disable the step sequence, disable step 5, disable step 4; \( nDlyOff05 = 0, nDlyOff04 = 0 \)
- t7 switch off step 3, \( nDlyOff03 = t7 - t6 \)
- t8 switch off step 2, \( nDlyOff02 = t8 - t7 \)
- t9 switch off step 1, \( nDlyOff01 = t9 - t8 \)

VAR_INPUT

\[
\begin{align*}
\text{bEn} & \quad : \text{BOOL}; \\
\text{bEvt01} & \quad : \text{BOOL}; \\
\text{nDlyOn01} & \quad : \text{UDINT}; \\
\text{nDlyOff01} & \quad : \text{UDINT}; \\
\text{bEvt02} & \quad : \text{BOOL}; \\
\text{nDlyOn02} & \quad : \text{UDINT}; \\
\text{nDlyOff02} & \quad : \text{UDINT}; \\
\text{bEvt03} & \quad : \text{BOOL}; \\
\text{nDlyOn03} & \quad : \text{UDINT}; \\
\text{nDlyOff03} & \quad : \text{UDINT}; \\
\text{bEvt04} & \quad : \text{BOOL}; \\
\text{nDlyOn04} & \quad : \text{UDINT}; \\
\text{nDlyOff04} & \quad : \text{UDINT}; \\
\text{bEvt05} & \quad : \text{BOOL}; \\
\text{nDlyOn05} & \quad : \text{UDINT}; \\
\text{nDlyOff05} & \quad : \text{UDINT}; \\
\text{bEvt06} & \quad : \text{BOOL}; \\
\text{nDlyOn06} & \quad : \text{UDINT}; \\
\text{nDlyOff06} & \quad : \text{UDINT}; \\
\text{bEvt07} & \quad : \text{BOOL}; \\
\text{nDlyOn07} & \quad : \text{UDINT}; \\
\end{align*}
\]
Programming

nDlyOff07  : UDINT;
bEvt08    : BOOL;
nDlyOn08  : UDINT;
nDlyOff08 : UDINT;
bEvt09    : BOOL;
nDlyOn09  : UDINT;
nDlyOff09 : UDINT;
bEvt10    : BOOL;
nDlyOn10  : UDINT;
nDlyOff10 : UDINT;
bEvt11    : BOOL;
nDlyOn11  : UDINT;
nDlyOff11 : UDINT;
bEvt12    : BOOL;
nDlyOn12  : UDINT;
nDlyOff12 : UDINT;

bEn: Enable function block.

bEvt01...12: Switch-on command for steps 1 to 12.

nDlyOn01...12: Start-up delay for output bQ01...12 [s].

nDlyOff01...12: Switch-off delay for output bQ01...12 [s].

VAR_OUTPUT

bQ01      : BOOL;
bQ02      : BOOL;
bQ03      : BOOL;
bQ04      : BOOL;
bQ05      : BOOL;
bQ06      : BOOL;
bQ07      : BOOL;
bQ08      : BOOL;
bQ09      : BOOL;
bQ10      : BOOL;
bQ11      : BOOL;
bQ12      : BOOL;
bUp       : BOOL;
bDwn      : BOOL;
nActvEvt  : UDINT;
nRemTiDlyOn: UDINT;
nRemTiDlyOff: UDINT;

bQ01...12: Step 1 to 12 On.

bUp: Control chain is in ascending state.

bDwn: Control chain is in descending state.

nActvEvt: Active step, display 0...12; "0" represents a non-active step sequence.

nRemTiDlyOn: Time remaining to up-switching to the next step [s].

nRemTiDlyOff: Time remaining to down-switching to the previous step [s].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
FB_BA_FIFO04_XX

This function block is used to evaluate the FiFo memory from FB_BA_FIFO04. The inputs are linked according to the FIFO table to the corresponding outputs of the function block FB_BA_FIFO04_BOOL or FB_BA_FIFO04_REAL.

Sample:
In the sample the array contains: 4,3,1,2,0,0,0,0. The following result is output in FB_BA_FIFO04_REAL:

- fIn01 to output fVal04
- fIn02 to output fVal03
- fIn03 to output fVal01
- fIn04 to output fVal02

VAR_INPUT

<table>
<thead>
<tr>
<th>aFIFO</th>
<th>: Array [1..4] OF UDINT;</th>
</tr>
</thead>
<tbody>
<tr>
<td>fin01…fin04</td>
<td>: REAL;</td>
</tr>
</tbody>
</table>

aFIFO: Contains the assignment table with a maximum of eight values. The first value indicates where the first input will be copied to, the second value indicates where the second input will be copied to, etc. No assignment is made for "0.

fin01…fin04: Setpoints to be linked.

VAR_OUTPUT

| fVal01…fVal04 | : REAL; |

fVal01…fVal04: actuator setpoint, input value linked according to FIFO table.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

FB_BA_FIFO08_XX

This function block is used to evaluate the FiFo memory from FB_BA_FIFO08. The inputs are linked according to the FIFO table to the corresponding outputs of the function block FB_BA_FIFO08_BOOL or FB_BA_FIFO08_REAL.
This function block is used to evaluate the FiFo memory from `FB_BA_FIFO08[217]`. The inputs are linked according to the FIFO table to the corresponding outputs of the function block `FB_BA_FIFO08_BOOL` or `FB_BA_FIFO08_REAL`.

Sample:
In the sample the array contains: 4,3,1,2,0,0,0,0. The result in `FB_BA_FIFO08_REAL` is

- `fIn01` to output `fVal04`
- `fIn02` to output `fVal03`
- `fIn03` to output `fVal01`
- `fIn04` to output `fVal02`

### VAR_INPUT

```plaintext
VAR_INPUT
aFIFO : Array [1..8] OF UDINT;
fIn01...fIn08 : REAL;
```

**aFIFO**: Contains the assignment table with a maximum of eight values. The first value indicates where the first input will be copied to, the second value indicates where the second input will be copied to, etc. No assignment is made for "0.

**fIn01...fIn08**: Setpoints to be linked.

### VAR_OUTPUT

```plaintext
VAR_OUTPUT
fVal01...fVal08 : REAL;
```

**fVal01...fVal08**: Actuator setpoint, input value linked according to FIFO table.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
Hysteresis, 2-point control

FB_BA_Cont4Stp01

The function block determines the resulting switching stages of a multi-level unit, depending on the input signal. Four switch-on thresholds and four hysteresis values can be parameterized.

Diagram 01

Control direction of parameter $bActn = \text{FALSE} = \text{Reverse} = \text{Heating}$
Diagram 02

Control direction parameter \( bActn = \text{TRUE} = \text{Direct} = \text{Cooling} \)

<table>
<thead>
<tr>
<th>nStp</th>
<th>nNumOfStp</th>
<th>fSwiOn</th>
<th>fSwiOff</th>
<th>nRemTi</th>
<th>nRemTi</th>
<th>bQ01</th>
<th>bQ02</th>
<th>bQ03</th>
<th>bQ04</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>fSwiOn01</td>
<td>fSwiOn01 - fHys01</td>
<td>nDlyOn01</td>
<td>0</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>1</td>
<td>&gt;= 1</td>
<td>fSwiOn02</td>
<td>fSwiOn01 - fHys01</td>
<td>nDlyOn02</td>
<td>nDlyOff01</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>2</td>
<td>&gt;= 2</td>
<td>fSwiOn03</td>
<td>fSwiOn02 - fHys02</td>
<td>nDlyOn03</td>
<td>nDlyOff02</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>3</td>
<td>&gt;= 3</td>
<td>fSwiOn04</td>
<td>fSwiOn03 - fHys03</td>
<td>nDlyOn04</td>
<td>nDlyOff03</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>4</td>
<td>&gt;= 4</td>
<td>fSwiOn04</td>
<td>fSwiOn04 - fHys04</td>
<td>0</td>
<td>nDlyOff04</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>
Diagram 03
Timing of the switch-on and switch-off delays

At time t₁ $fln$ jumps from $fSwiOn01$ to $fSwiOn04$

At time t₂ $fln$ jumps from $fSwiOn04$ to $fSwiOn01 - fHys01$
VAR_INPUT

bEn : BOOL;

fIn : REAL;

fSwiOn01 : REAL;

fHys01 : REAL;

nDlyOn01 : UDINT;

nDlyOff01 : UDINT;

fSwiOn02 : REAL;

fHys02 : REAL;

nDlyOn02 : UDINT;

nDlyOff02 : UDINT;

fSwiOn03 : REAL;

fHys03 : REAL;

nDlyOn03 : UDINT;

nDlyOff03 : UDINT;

fSwiOn04 : REAL;

fHys04 : REAL;

nDlyOn04 : UDINT;

nDlyOff04 : UDINT;

nNumOfStp : UDINT;

bActn : BOOL;

**bEn**: General enable of the function block. If `bEn` is FALSE, all outputs are set to 0.

**fIn**: Input value, from which the switching state is derived.

**fSwiOn01**: Switch-on point stage 01

**fHys01**: Absolute value hysteresis stage 01

**nDlyOn01**: Start-up delay stage 01

**nDlyOff01**: Switch-off delay stage 01

**fSwiOn02**: Switch-on point stage 02

**fHys02**: Absolute value hysteresis stage 02

**nDlyOn02**: Start-up delay stage 02

**nDlyOff02**: Switch-off delay stage 02

**fSwiOn03**: Switch-on point stage 03
fHys03: Absolute value hysteresis stage 03

nDlyOn03: Start-up delay stage 03

nDlyOff03: Switch-off delay stage 03

fSwiOn04: Switch-on point stage 04

fHys04: Absolute value hysteresis stage 04

nDlyOn04: Start-up delay stage 04

nDlyOff04: Switch-off delay stage 04

nNumOfStp: Number of stages that are required.
The input is limited to a range from 0 to 4

bActn: Input variable used to determine the control direction of the step switch.
TRUE = direct = cooling; FALSE = reverse = heating

VAR_OUTPUT

bQ01 : BOOL;
bQ02 : BOOL;
bQ03 : BOOL;
bQ04 : BOOL;
nStp : UDINT;
fSwiOn : REAL;
fSwiOff : REAL;
nRemTiDlyOn : UDINT;
nRemTiDlyOff : UDINT;

bQ01: Shows status level 01
TRUE = ON; FALSE = OFF
nStp >= 1

bQ02: Shows status level 02
TRUE = ON; FALSE = OFF
nStp >= 2

bQ03: Shows status level 03
TRUE = ON; FALSE = OFF
nStp >= 3

bQ04: Shows status level 04
TRUE = ON; FALSE = OFF
nStp >= 4

nStp: Shows the current step of the step switch

fSwiOn: Shows the next switch-on point

fSwiOff: Shows the next switch-off point

nRemTiDlyOn: If the switch-on point for switching to the next level is met, the progress of the switch-on delay time is displayed here.

nRemTiDlyOff: If the switch-off point for switching down to the next level is met, the progress of the switch-off delay time is displayed here.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
The function block **FB_BA_Swi2P** is a two-point switch with one switch-on point and one switch-off point.

A general function block enable can be implemented at input \( bEn \). The output \( bQ \) is FALSE as long as \( bEn \) is FALSE. The control direction of the function block depends on the relative position of the switch-on/switch-off points.

If the switch-on point is greater than the switch-off point, the control direction is direct/synchronous (cooling mode).

If the switch-off point is greater than the switch-on point, the control direction is indirect/reversed (heating mode).
VAR_INPUT

bEn : BOOL;
fIn : REAL;
fOn : REAL;
fOff : REAL;
nDlyOn : UDINT;
nDlyOff : UDINT;

bEn: General enable of the function block.
fIn: Input value.
fOn: Switch-on point.
fOff: Switch-off point.
nDlyOn: Start-up delay [s].
nDlyOff: Switch-off delay [s].

VAR_OUTPUT

bQ : BOOL;
nRemTiDlyOn : UDINT;
nRemTiDlyOff : UDINT;

bQ: Control output.
nRemTiDlyOn: Remaining time of the start-up delay [s].
nRemTiDlyOff: Remaining time of the switch-off delay [s].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
The function block FB_BA_SwiHys2P is a two-point switch with adjustable hysteresis and hysteresis offset.

A general function block enable can be implemented at input bEn. If the function block is locked, the output bQ is FALSE. The setpoint for the two-point switch is connected at input fSp. The control direction of the function block depends on the input variable bActn.

The active switching points result from the setpoint, the hysteresis and the hysteresis offset. They are output at fSwiHi and fSwiLow.

- The upper switching point is given by fSp + fHys/2 + fHysOffs.
- The lower switching point is given by fSp - fHys/2 + fHysOffs.

If bActn TRUE, the result is direct/synchronous control direction (cooling mode).

If bActn FALSE, the result is indirect/reversed control direction (heating mode).
VAR_INPUT

bEn : BOOL;
rIn : REAL;
rSp : REAL;
rHys : REAL;
rHysOffs : REAL;
udiDlyOn_sec : UDINT;
udiDlyOff_sec : UDINT;
bActn : BOOL;

bEn: General enable of the function block.

fIn: Input value.

fSp: Setpoint input.

fHys: Hysteresis.

fHysOffs: Hysteresis offset.

nDlyOn: Start-up delay [s].

nDlyOff: Switch-off delay [s].

bActn: Control direction.

VAR_OUTPUT

bQ : BOOL;
fSwiHi : REAL;
fSwiLow : REAL;
nRemTiDlyOn : UDINT;
nRemTiDlyOff : UDINT;

bQ: Output.

fSwiHi: Upper switching point.

fSwiLow: Lower switching point.

nRemTiDlyOn: Time remaining before switching on [s].

nRemTiDlyOff: Time remaining before switching off [s].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

Mathematical functions

FB_BA_EnAvrg0X
VAR_INPUT
bEn01..bEn04 : BOOL;
fVal01..fVal04 : REAL;

VAR_OUTPUT
bQ : BOOL;
fAvgVal : REAL;
fMinVal : REAL;
fMaxVal : REAL;
nActvCnt : UDINT;

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

FB_BA_MultiCalcREAL32

The multi-calculation function block exists for the variable type REAL.

In enabled state \((bEn=TRUE)\), the function block determines the following from the input values \(aVal\):

- the maximum value of all inputs \(fMax\)
- the input at which this maximum value is present \(nMaxActv\)
- the minimum value of all inputs \(fMin\)
- the input at which this minimum value is present \(nMinActv\)
- the mean value of all inputs \(fAvrg\)
- the sum of all inputs \(fSum\)
- the difference between the maximum and minimum value \(fDiff\)

If not all inputs are to be used for the calculation, the number can be limited via an entry at \(nNumOfElem\): with \(nNumOfElem = 6\), for example, the calculations are performed only for the first six entries of \(aVal\). Any entry greater than 32 is automatically limited to 32, any entry less than 1 is automatically set to 1.

Sample:
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEn = TRUE</td>
<td>fMax = 32</td>
</tr>
<tr>
<td>aVal[1] = 32</td>
<td>nMaxActv = 1</td>
</tr>
<tr>
<td>aVal[2] = 17</td>
<td>fMin = 5</td>
</tr>
<tr>
<td>aVal[3] = 5</td>
<td>nMinActv = 3</td>
</tr>
<tr>
<td>aVal[4] = 9</td>
<td>fAvg = 18.5</td>
</tr>
<tr>
<td>aVal[5] = 16</td>
<td>fSum = 111</td>
</tr>
<tr>
<td>aVal[6] = 32</td>
<td>fDiff = 27</td>
</tr>
<tr>
<td>aVal[7] = 25</td>
<td></td>
</tr>
<tr>
<td>aVal[8] = 44</td>
<td></td>
</tr>
<tr>
<td>nNumOfElem = 6</td>
<td></td>
</tr>
</tbody>
</table>

If $bEn = \text{FALSE}$, 0 is output at all outputs.

**VAR_INPUT**

- **bEn**: Activation of the block function.
- **aVal**: Field with the values to be calculated.
- **nNumOfElem**: Number of input values to be used for the calculation.

**VAR_OUTPUT**

- **fMax**: Maximum value of all inputs.
- **nMaxActv**: Input at which the maximum value is present.
- **fMin**: Minimum value of all inputs.
- **nMinActv**: Input at which the minimum value is present.
- **fAvg**: Average value of all inputs.
- **fSum**: Sum of all inputs.
- **fDiff**: Difference between maximum and minimum value.
- **bErr**: This output is switched to TRUE if the parameters entered are erroneous.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
The function block FB_BA_Chrtc02 represents a linear interpolation with 2 interpolation points and can be used to generate a characteristic curve. The characteristic curve is determined by the interpolation points \([fX01/fY01]\) and \([fX02/fY02]\).

**Error handling**

The input values for \(fX[n+1]\) must always be at least 0.0000001 greater than the values for \(fX[n]\). In the event of an error the variable \(sErrDescr\) indicates that at one point of the characteristic curve the values are not monotonically increasing.

**VAR_INPUT**

- \(fX\) : REAL;
- \(fX01\) : REAL;
- \(fX02\) : REAL;
- \(fY01\) : REAL;
- \(fY02\) : REAL;
- \(fYMin\) : REAL;
- \(fYMax\) : REAL;

\(fx\): Input value of the characteristic curve.

\(fX01\): X-value for interpolation point P1.

\(fX02\): X-value for interpolation point P2.

\(fY01\): Y-value for interpolation point P1.

\(fY02\): Y-value for interpolation point P2.

\(fYMin\): Smallest y-value for an interpolation point.

\(fYMax\): Largest y-value for an interpolation point.

**VAR_OUTPUT**

- \(fCalcVal\) : REAL;
- \(bErr\) : BOOL;

\(fCalcVal\): Calculated value of the characteristic curve.

\(bErr\): Indicates whether an error has occurred.
**fCalcVal**: Calculated output value of the characteristic curve.

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

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

**FB_BA_Chrtc03**

**Error handling**

The input values for \( fX[n+1] \) must always be at least 0.0000001 greater than the values for \( fX[n] \).

In the event of an error the variable \( sErrDescr \) indicates that at one point of the characteristic curve the values are not monotonically increasing.

**VAR_INPUT**

- \( fX \): Input value of the characteristic curve.
- \( fX01 \): X-value for interpolation point P1.
- \( fX02 \): X-value for interpolation point P2.
- \( fX03 \): X-value for interpolation point P3.
- \( fY01 \): Y-value for interpolation point P1.
- \( fY02 \): Y-value for interpolation point P2.
- \( fY03 \): Y-value for interpolation point P3.
- \( fYMin \): Smallest y-value for an interpolation point.
- \( fYMax \): Largest y-value for an interpolation point.

**VAR_OUTPUT**

- \( fCalcVal \): Calculated output value of the characteristic curve.
bErr: This output is switched to TRUE if the parameters entered are erroneous.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

**FB_BA_Chrt04**

The function block FB_BA_Chrt04 represents a linear interpolation with 4 interpolation points and can be used to generate a characteristic curve. The characteristic curve is determined by the interpolation points \([fX1/fY1]\) to \([fX4/fY4]\). If the input variable \(bLmt\) is TRUE, \(fY\) is limited by \(fY01\) and \(fY04\). If \(bLmt\) is FALSE, \(fY\) is not limited.

Error handling

The input values for \(fX[n+1]\) must always be at least 0.0000001 greater than the values for \(fX[n]\). In the event of an error the variable \(sErrDescr\) indicates that at one point of the characteristic curve the values are not monotonically increasing.

**VAR_INPUT**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fX</strong></td>
<td>REAL;</td>
</tr>
<tr>
<td><strong>fX01</strong></td>
<td>REAL;</td>
</tr>
<tr>
<td><strong>fX02</strong></td>
<td>REAL;</td>
</tr>
<tr>
<td><strong>fX03</strong></td>
<td>REAL;</td>
</tr>
<tr>
<td><strong>fX04</strong></td>
<td>REAL;</td>
</tr>
<tr>
<td><strong>fY01</strong></td>
<td>REAL;</td>
</tr>
<tr>
<td><strong>fY02</strong></td>
<td>REAL;</td>
</tr>
<tr>
<td><strong>fY03</strong></td>
<td>REAL;</td>
</tr>
<tr>
<td><strong>fY04</strong></td>
<td>REAL;</td>
</tr>
<tr>
<td><strong>fYMin</strong></td>
<td>REAL;</td>
</tr>
<tr>
<td><strong>fYMax</strong></td>
<td>REAL;</td>
</tr>
</tbody>
</table>
fX: Input value of the characteristic curve.
fX01: X-value for interpolation point P1.
fX02: X-value for interpolation point P2.
fX03: X-value for interpolation point P3.
fX04: X-value for interpolation point P4.
fY01: Y-value for interpolation point P1.
fY02: Y-value for interpolation point P2.
fY03: Y-value for interpolation point P3.
fY04: Y-value for interpolation point P4.
fYMin: Smallest y-value for an interpolation point.
fYMax: Largest y-value for an interpolation point.

VAR_OUTPUT
fCalcVal : REAL;
bErr : BOOL;

fCalcVal: Calculated output value of the characteristic curve.
bErr: This output is switched to TRUE if the parameters entered are erroneous.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

FB_BA_Chrc07

The function block FB_BA_Chrc07 represents a linear interpolation with 7 interpolation points and can be used to generate a characteristic curve. The characteristic curve is determined by the interpolation points [fX01/fY01] to [fX07/fY07]. If the input variable bLmt is TRUE, fY is limited by fY01 and fY07. If bLmt is FALSE, fY is not limited.
Error handling

The input values for $fX[n+1]$ must always be at least 0.0000001 greater than the values for $fX[n]$. In the event of an error the variable sErrDescr indicates that at one point of the characteristic curve the values are not monotonically increasing.

VAR_INPUT

| fX   | : REAL; |
| fX01 | : REAL; |
| fX02 | : REAL; |
| fX03 | : REAL; |
| fX04 | : REAL; |
| fX05 | : REAL; |
| fX06 | : REAL; |
| fY01 | : REAL; |
| fY02 | : REAL; |
| fY03 | : REAL; |
| fY04 | : REAL; |
| fY05 | : REAL; |
| fY06 | : REAL; |
| fY07 | : REAL; |
| fYMin| : REAL; |
| fYMax| : REAL; |

rX: Input value of the characteristic curve.

rX01: X-value for interpolation point P1.

rX02: X-value for interpolation point P2.

rX03: X-value for interpolation point P3.

rX04: X-value for interpolation point P4.

rX05: X-value for interpolation point P5.

rX06: X-value for interpolation point P6.

rX07: X-value for interpolation point P7.

rY01: Y-value for interpolation point P1.

rY02: Y-value for interpolation point P2.

rY03: Y-value for interpolation point P3.

rY04: Y-value for interpolation point P4.

rY05: Y-value for interpolation point P5.

rY06: Y-value for interpolation point P6.

rY07: Y-value for interpolation point P7.

fYMin: Smallest y-value for an interpolation point.
**VAR_OUTPUT**

- fCalcVal : REAL;
- bErr : BOOL;

**fCalcVal:** Calculated output value of the characteristic curve.

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

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

**FB_BA_Chrc32**

The function block FB_BA_Chrc32 represents a linear interpolation with up to 32 interpolation points and can be used to generate a characteristic curve. In contrast to the "smaller" interpolation function blocks FB_BA_Chrc02 [238], FB_BA_Chrc04 [240] and FB_BA_Chrc07 [241], and in the interest of clarity, the interpolation points are determined via field variables [aX[1]/aY[1]] to [aX[n]/aY[n]]. If the input variable bLmt is TRUE, fY is limited by aY[1] and aY[n]. If bLmt is FALSE, fY is not limited.

**Error handling**

The input values for fX[n+1] must always be at least 0.0000001 greater than the values for fX[n].

**VAR_INPUT**

- fX : REAL;
- aX : ARRAY [1..cBA_NumOfElem] OF REAL;
- aY : ARRAY [1..cBA_NumOfElem] OF REAL;
- nNumOfElem : DINT(2..32);
- fYMin : REAL;
- fYMax : REAL;

**fX:** Input value of the characteristic curve

**aX:** Field with the X-values for the interpolation points.
aY: Field with the Y-values for the interpolation points.

nNumOfElem: Number of interpolation points. Internally limited to values between 2 and 32.

fYMin: Smallest y-value for an interpolation point.

fYMax: Largest y-value for an interpolation point.

VAR_OUTPUT

fCalcVal : REAL;
bErr : BOOL;

fCalcVal: Calculated output value of the characteristic curve.

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

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

FB_BA_TiAvrg

The function block FB_BA_TiAvrg calculates the arithmetic mean value of an analog value that was logged over a certain period. Discrete values are written into a FIFO buffer. nIntval specifies the time interval [s] over which the values are logged and written into the FIFO. Values are written if the input bEn is TRUE. The variable nNumOfElem is used to determine the size of the FIFO buffer. It is limited to 1…512.

The function block can be used for calculating an hourly mean outside temperature over a day, for example. In this case nNumOfElem would be 24, and nIntval would be 3600 seconds. bEn is the general enable of the function block. If bEn = FALSE, the FIFO buffer within the function block is deleted completely, and no data are recorded.

Sample:

nNumOfElem = 5

<table>
<thead>
<tr>
<th>First cycle</th>
<th>Second cycle</th>
<th>Third cycle</th>
<th>Fourth cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>t1</td>
<td>4</td>
<td>(2+4)/2 = 3</td>
<td>5</td>
</tr>
<tr>
<td>t2</td>
<td>6</td>
<td>(2+4+6)/3 = 4</td>
<td>4</td>
</tr>
<tr>
<td>t3</td>
<td>7</td>
<td>(2+4+6+7)/4 = 4.75</td>
<td>2</td>
</tr>
<tr>
<td>t4</td>
<td>7</td>
<td>(2+4+6+7+7)/5 = 5.2</td>
<td>1</td>
</tr>
</tbody>
</table>
VAR_INPUT

bEn : BOOL;
fIn : REAL;
nIntVal : UDINT;
nNumOfElem : UDINT;

bEn: Enables the function block.
fIn: Input value for averaging.

nIntVal: Time interval [s] for writing new values into the FIFO. Internally limited to a value between 1 and 2147483.

nNumOfElem: Size of the FIFO buffer. A change resets the previous averaging. Internally limited to a value between 1 and 512.

VAR_OUTPUT

fOut : REAL;
fMax : REAL;
fMin : REAL;

fOut: Calculated mean value.
fMax: Largest value in the FIFO buffer.
fMin: Smallest value in the FIFO buffer.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

Monitoring functions

FB_BA_FdbCtrlBinary

The function block is used for feedback monitoring of an actuator by means of digital feedback. Application examples of the function block are, for example, an operation feedback monitoring, a process feedback monitoring or the run monitoring of a drive by means of limit switches.

The input bEn is used for enabling the function block. If bEn is FALSE, the message output bQ will always be FALSE.

The switching actuator output of the unit to be monitored is connected to the input bActuator. The bSwitch input is used to connect the feedback signal (e.g. differential pressure switch, flow monitor or limit switch).

The timer nFdbDelay [s] is used to a response delay of the feedback control after the start of the unit.

The second timer nInterruptionDelay [s] is used for a response delay of the feedback control after reaching the end state.
VAR_INPUT

bEn : BOOL;
bActuator : BOOL;
bSwitch : BOOL;
nFdbDelay : UDINT;
nInterruptionDelay : UDINT;

bEn: Function block enable.

bActuator: Feedback of the switching output.

bSwitch: Feedback signal from the process.

nFdbDelay: Response delay [s] of the monitoring function when the actuator is started. Internally limited to values from 0 to BACmn_Global.udiMaxSecInMilli.

nInterruptionDelay: Response delay [s] of the monitoring function when the actuator has already been started successfully (e.g. pressure fluctuations when monitoring the running of a fan). Internally limited to values from 0 to BACmn_Global.udiMaxSecInMilli.

VAR_OUTPUT

bQ : BOOL;
nRemTiFdbDelay : UDINT;
nRemTiInterruptionDelay : UDINT;

bQ: Output an error message if the feedback signal is not present within the parameterized time of nFdbDelay, or the feedback signal has been interrupted longer than after nInterruptionDelay.

nRemTiFdbDelay: Remaining time [s] until output bErrOpn is set.

nRemTiInterruptionDelay: Remaining time [s] until output bErrSwi is set.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

FB_BA_FixedLimitCtrl

Function block for monitoring a fixed limit value.

The input bEn is used for enabling the function block.

A tolerance range is defined around the value fIn to be monitored.

The tolerance range results from an upper limit value fHighLimit and a lower limit value fLowLimit.

If the value fIn exceeds the upper limit value of the tolerance range, then the output bHighLimit becomes TRUE. A response delay of the output bHighLimit is to be parameterized with the timer nDelay.

If the value fIn falls below the lower limit of the tolerance range, output bLowLimit becomes TRUE. A response delay of the output bLowLimit is to be parameterized with the timer nDelay.
bEn: Function block enable.

fHighLimit: Default upper limit value, preset to 32.

fLowLimit: Default lower limit value, preset to 16.

fIn: Input value to be monitored.

nDelay: Output response delay [s]. Internally limited to values from 0 to BAComn_Global.udiMaxSecInMilli.

bHighLimit: Upper limit value reached.

bLowLimit: Lower limit value reached.

nRemTiDelay: Time remaining after a limit value has been exceeded until either the output bHighLimit or bLowLimit responds.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

**FB_BA_SlidingLimitCtrl**

Function block for monitoring a floating setpoint.

The input bEn is used for enabling the function block.

To check the function of a control system, the actual value is compared with the setpoint of the controlled system.

If the deviation of setpoint and actual value is within the tolerance range fHys, then the control is OK. If the actual value deviates from the setpoint by an amount outside this tolerance range over an extended period, the timer nDelay is started. After the timer has expired, if the control deviation is permanent, either the output bLowLimit or bHighLimit TRUE of the function block outputs a message.
VAR_INPUT

bEn : BOOL;
fW : REAL;
fX : REAL;
fHys : REAL;
nDelay : UDINT;

bEn: Function block enable.
fW: Setpoint.
fX: Actual value.
fHys: Hysteresis.
nDelay: Output response delay [s]. Internally limited to values from 0 to BACmn_Global.udIMaxSecInMilli.

VAR_OUTPUT

bHighLimit : BOOL;
bLowLimit : BOOL;
fHighLimit : REAL;
fLowLimit : REAL;
nRemTiDelay : UDINT;

bHighLimit: Upper limit value reached.
bLowLimit: Lower limit value reached.
fHighLimit: Output of the upper limit value.
fLowLimit: Output of the lower limit value.
nRemTiDelay: Time remaining after a limit value has been exceeded until either the output bHighLimit or bLowLimit responds.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

Ramps, filters, controllers

FB_BA_FltrPT1

First order filter.

When the function block is first called (system startup), the output fOut is automatically set (once) to the input fin.

VAR_INPUT

| fin | REAL |
| nDampConst | UDINT |
| bSetAct1 | BOOL |

fin: Input signal

nDampConst: Filter time constant [s]. Internally limited to values between 0 and 86400.

bSetAct1: A rising edge at this input sets the output value fOut to the input value fin.

VAR_OUTPUT

| fOut | REAL |

fOut: Filtered output signal.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>

FB_BA_RampLmt

Requirements
The function block limits the increase or decrease speed of an input signal. When \( f_{\text{In}} \) increases, the output \( f_{\text{Out}} \) is limited to the slope of \((f_{\text{Hi}} - f_{\text{Lo}})/n_{\text{TiUp}}\).

When \( f_{\text{In}} \) decreases, the output \( f_{\text{Out}} \) is limited to the slope of \((f_{\text{Hi}} - f_{\text{Lo}})/n_{\text{TiDwn}}\).

**VAR_INPUT**

- \( b_{\text{En}} \): Enable function block if FALSE, in which case \( f_{\text{Out}} = 0.0 \).
- \( b_{\text{EnRamp}} \): Enable ramp limitation if FALSE, in which case \( f_{\text{Out}} = f_{\text{In}} \).
- \( f_{\text{In}} \): Input value of the ramp function
- \( f_{\text{Hi}} \): Upper interpolation point for calculating the ramps.
- \( f_{\text{Lo}} \): Lower interpolation point for calculating the ramps. \( f_{\text{Hi}} \) must be greater than \( f_{\text{Lo}} \), otherwise an error is output.
- \( n_{\text{TiUp}} \): Rise time [s].
- \( n_{\text{TiDwn}} \): Fall time [s]

**VAR_OUTPUT**

- \( f_{\text{Out}} \): Output signal, slope-limited through the ramps.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.17</td>
<td>Tc3_BA2 from v4.8.9.0</td>
</tr>
</tbody>
</table>
7.1.1.2 Archive

7.1.1.2.1 Tc3_BA

This library is used for the maintenance and servicing of existing projects. For new projects please use the library Tc3_BA2.

POUs

Air conditioning equipment

Function blocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_BA_FrstPrtc</td>
<td>Monitoring of frost alarm and emergency heating</td>
</tr>
<tr>
<td>FB_BA_HX</td>
<td>Calculation of dew point temperature, specific enthalpy and absolute humidity</td>
</tr>
<tr>
<td>FB_BA_NgtCol</td>
<td>Summer night cooling</td>
</tr>
<tr>
<td>FB_BA_RcvMonit</td>
<td>Function block for calculating the efficiency of an energy recovery system</td>
</tr>
<tr>
<td>FB_BA_SPSupvis</td>
<td>Function block for processing and checking the lower and upper setpoint of a supply air humidity or temperature control</td>
</tr>
</tbody>
</table>

FB_BA_FrstPrtc

The function block is used for frost monitoring of a heating coil in an air conditioning system.

A frost risk is present, if the input bFrst is TRUE. The frost alarm must be linked in the plant program such that the plant is switched off immediately, the heater valve opens, and the heater pump is switched on.

If there is risk of frost, the output bOn is set, and udiT1_sec (seconds) is started. If the frost risk remains (bFrst=TRUE) after udiT1_sec has elapsed, bOn remains set. It can only be reset at input bRst.

If the frost alarm ceases due to activation of the heating coil within the time udiT1_sec (bFrst=FALSE), the plant automatically restarts. For the plant restart bOn becomes FALSE, and at output bHWRst a pulse for acknowledgement of a latching circuit in the control cabinet is issued. With the restart a second monitoring period udiT2_sec (seconds) is initiated. If another frost alarm occurs within this period, the plant is permanently locked. bOn remains set until the frost alarm has been eliminated and bRst has been acknowledged.

In a scenario where frost alarms recur with time offsets that are greater than udiT2_sec, theoretically the plant would keep restarting automatically. In order to avoid this, the restarts within the function block are counted. The parameter udiAlmCnt can be used to set the number of possible automatic restart between 0 and 4.
An acknowledgement at input $bRst$ resets the alarm memory within the function block to zero.

**Sample:**

$t0 =$ frost alarm at input $bFrst$, alarm message at output $bOn$, start of timer T1 ($udiT1_{sec}$ [$s$])
$t1 =$ frost alarm off, resetting of $bOn$, output of hardware pulse, start of timer T2 ($udiT2_{sec}$ [$s$]), plant restart
$t2 =$ further frost alarm within T2, alarm message at $bOn$, start of timer T1, locking of the frost alarm
$t3 =$ frost alarm off.
$t4 =$ acknowledgement of the alarm at $bRst$, resetting of $bOn$.

**VAR_INPUT**

$bFrst: Connection for frost events on the air and water side.

udiT1_{sec}: Timer for restart delays [$s$]. Internally limited to a minimum value of 0.

udiT2_{sec}: Timer monitoring time [$s$]. Internally limited to a minimum value of 0.

udiAlmCnt: Maximum number of automatic plant restarts without reset. Internally limited to values between 0 and 4.

$bRst: Resetting and acknowledgement of the frost alarm.

**VAR_OUTPUT**

$bOn: Frost alarm active.

bHWRst: Output of a pulse for acknowledgement of the frost protection hardware.

udiRemTi1_{sec}: Time remaining to plant restart after frost alarm.
udiRemTi2_sec: Remaining monitoring time.

bAlmLck: Alarm lock - stored alarm.

udiStaCnt: Status counter – current number of unacknowledged false starts.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**FB_BA_HX**

This function block is used to calculate the dew point temperature, the specific enthalpy and the absolute humidity. The temperature, the relative humidity and the barometric pressure are required for calculating these parameters. The enthalpy is a measure for the energy of a thermodynamic system.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>rT</th>
<th>REAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>rHumRel</td>
<td>REAL</td>
</tr>
<tr>
<td>rAP</td>
<td>REAL</td>
</tr>
</tbody>
</table>

rT: Temperature [°C].

rHumRel: Relative humidity [%].

rAP: Hydrostatic air pressure at 1013.25 hPa.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>lrHumAbs</th>
<th>LREAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>lrDewPnt</td>
<td>LREAL</td>
</tr>
<tr>
<td>lrE</td>
<td>LREAL</td>
</tr>
<tr>
<td>lrDHA</td>
<td>LREAL</td>
</tr>
<tr>
<td>lrSpecV</td>
<td>LREAL</td>
</tr>
<tr>
<td>lrTWet</td>
<td>LREAL</td>
</tr>
</tbody>
</table>

lrHumAbs: Absolute humidity g water per kg dry air [g/Kg].

lrDewPnt: Dew point temperature [°C].

lrE: Enthalpy [kJ/kg].

lrDHA: Density of moist air ρ [kg mixture/m³].

lrSpecV: Specific volume [m³/kg].

lrTWet: Wet bulb temperature [°C].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
With this function block, rooms that were heated up on the day before can be cooled down during the night using cool outside air. The summer night cooling function serves to improve the quality of the air and to save electrical energy. Electrical energy for cooling is saved during the first hours of the next summer day.

The start conditions for the summer night cooling are defined by parameterizing the FB_BA_NgtCol function block. The function block can be used to open motor-driven windows or to switch air conditioning systems to summer night cooling mode outside their normal hours of operation.

The following conditions must be met for activation of summer night cooling:

- The function block itself is enabled (bEn=TRUE).
- The outside temperature is not too low (rTOts > rTOtsLoLmt).
- The outside temperature is sufficiently low compared with the room temperature (rTRm - rTOts) > rSwiOnDiffT.
- The room temperature is high enough to justify activating summer night cooling. rTRm > rSpRm + rTRmHys.

Under the following conditions the summer night cooling is disabled:

- The function block itself is disabled (bEn = FALSE).
- The outside temperature is too low (rTOts < rTOtsLoLmt).
- The outside temperature is too high compared with the room temperature (rTRm - rTOts) < rSwiOffDiffT.
- The room temperature is lower than the setpoint. rTRm < rSpRm.

### VAR_INPUT

- **bEn**: Enable function block.
- **rTOts**: Outside temperature [°C].
- **rTRm**: Outside temperature [°C].
- **rSpRm**: Room temperature setpoint.
- **rTOtsLoLmt**: Lower outside temperature limit [°C]; prevents excessive cooling.
- **rSwiOnDiffT**: Hysteresis for minimum outside temperature [K]. This hysteresis, which at the lower end is internally limited to 0.5 K, is intended to prevent jitter in bQ, if the outside temperature fluctuates precisely around the value of rTOtsLoLmt.
rTRmHys: Hysteresis for the room temperature [K]. This hysteresis, which at the lower end is internally limited to 0.5 K, is intended to prevent unnecessary fluctuation of bQ, if the room temperature fluctuates precisely around the setpoint rSpRm.

rSwiOnDiffT: Difference between the room temperature and the outside temperature, from which summer night cooling is enabled [K].

rSwiOffDiffT: Difference between the room temperature and the outside temperature, from which summer night cooling is locked [K].

**VAR_OUTPUT**

bQ : BOOL;

bQ: Summer night cooling on.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**FB_BA_RcvMonit**

The function block is used for calculating the efficiency of an energy recovery system.

The function block requires the following measured temperature values for calculating the efficiency (heat recovery rate):

- Outside air temperature rTOts
- Exhaust air temperature rTEXh
- Air temperature of the energy recovery system in the inlet air duct (alternatively: in the outlet air duct) rTAftRcv
The function block logs the temperature values every 10 seconds and forms minutely averages from 6 consecutive values. The results are used to check whether the plant has reached a "stable" state.

- This is the case when the recorded temperatures of outside air, exhaust air and air after energy recovery are almost constant, i.e. none of the 6 individual values deviate by more than 0.5 K from the respective mean value.
- The temperature difference between outside air and exhaust air is at least 5 K.

If this is the case, this measuring cycle is acknowledged with a TRUE signal at output \( bStblOp \), and the calculated efficiency is output at \( rEffc \). If the state is not "stable", a FALSE signal appears at output \( bStblOp \), and \( rEffc \) is set to 0.

In any case, each measuring and analysis cycle is marked as completed with a trigger (a TRUE signal lasting one PLC cycle) at \( bNewVal \).

**Enable (\( bEn \)) and Reset (\( bRst \))**

The function block is only active if a TRUE signal is present at \( bEn \). Otherwise its execution stops, and all outputs are set to FALSE or 0.0.

An active measuring and evaluation cycle can be terminated at any time by a TRUE signal at \( bRst \). All outputs are set to FALSE or 0.0, and the measuring cycle restarts automatically.

**Selection of the temperature value "after recovery" (\( bSnsRcvTExh \))**

A FALSE entry at \( bSnsRcvTExh \) means that the temperature measurement after the heat recovery in the supply air duct is used for calculating the efficiency.

To use the temperature measurement after the heat recovery in the exhaust air duct, TRUE must be applied at \( bSnsRcvTExh \).

**Limit violation (\( rContrVar \), \( rLmtEffc \), \( bLmtRchd \))**

A limit violation has occurred, if the calculated efficiency is less than the specified limit value \( rLmtEffc \), and at the same time the control value for the heat recovery is at 100%. To this end the control value must be linked to input \( rContrVar \).

The limit violation message can be delayed by an entry at \( udiLmtVioDly\_sec \) [s]: If the two criteria, violation and override, are met for longer than \( udiLmtVioDly\_sec \) [s], this is indicated with a TRUE signal at \( bLmtRchd \).

A warning message, which may have occurred, is canceled if a complete measuring cycle provides "good" values, or with a rising edge at \( bRst \) or deactivation of the function block.
This warning message only occurs if the plant is in a stable operating mode (bStblOp=TRUE).

**Taking into account the temperature increase of the outlet air due to the fan motor (rTIncFan)**

It is possible that the outlet air is warmed by a fan motor, resulting in distortion of the measurement. This temperature increase can be specified through rTIncFan. Internally, the measured outlet air temperature is then reduced by this value.

**VAR_INPUT**

- bEn: Function block enable.
- bRst: Reset - all determined values are deleted.
- rContrVar: Control value for the heat recovery, i.e. the actual value.
- rTOts: Outside temperature.
- rTEExh: Exhaust air temperature.
- rTAftRcv: Temperature after energy recovery.
- bSnsRcvTEExh: Temperature at the measuring point after energy recovery: FALSE -> in inlet air duct (SupplyAir) - TRUE -> in outlet air duct (ExhaustAir).
- rTIncFan: Temperature increase due to fan.
- rLmtEffc: Limit value efficiency.
- udiLmtVioDly_sec: Limit violation delay [s]. Internally limited to a minimum value of 0.

**VAR_OUTPUT**

- bNewVal: Output trigger for new value rEffc.
- rEffc: Efficiency
- bLmtRchd: Limit value reached
- bStblOp: Stable operation.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
FB_BA_SpSupvis

Function block for processing and checking the lower and upper setpoint of an inlet air humidity or temperature control.

Checking and limitation of the setpoints

The function block limits the setpoints. The following two tables show which parameters are checked and what the response is in the event of an error.

<table>
<thead>
<tr>
<th>Checking</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>rSpLo &gt; rSpHi</td>
<td>last valid values of rSpLo and rSpHi are used</td>
</tr>
<tr>
<td>rSpMin &gt;= rSpMax</td>
<td>last valid values of rSpMin and rSpMax are used</td>
</tr>
<tr>
<td>rSpHi &gt; rSpMax</td>
<td>rPrSpHi = rSpMax</td>
</tr>
<tr>
<td>rSpLo &lt; rSpMin</td>
<td>rPrSpLo = rSpMin</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Checking</th>
<th>bErr</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>rSpMin &gt;= rSpMax</td>
<td>TRUE</td>
<td>rSpErr = ((rSpMin + rSpMax) / 2)</td>
</tr>
<tr>
<td>rSpHi &lt; rSpMin</td>
<td></td>
<td>rPrSpHi = rPrSpLo = rPrRcv = rSpErr</td>
</tr>
<tr>
<td>rSpLo &gt; rSpMax</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The difference between the setpoints describes an energy-neutral zone. With inlet air control, no heating or cooling would take place within the neutral zone.

![Diagram](image)

The checked and, if necessary, limited setpoints are output at the function block output as rPrSpHi and rPrSpLo (Present Setpoint).

Setpoint for heat recovery

For heat recovery, the setpoint rSpRcv is optionally calculated from the mean value of the upper and lower setpoint, rSpHi and rSpLo, or depending on the control direction of the heat recovery system. The method is defined through the input variable bSlcnSpRcv.
<table>
<thead>
<tr>
<th>b SlcnSpRcv</th>
<th>rSpRcv</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>Mean value of rSpLo and rSpHi</td>
</tr>
<tr>
<td>FALSE</td>
<td>Depends on direction of action, defined through input bActRcv</td>
</tr>
</tbody>
</table>

If the setpoint is defined depending on the direction of action, the following applies:

<table>
<thead>
<tr>
<th>bActRcv</th>
<th>Control direction</th>
<th>rSpRcv</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>direct (cooling)</td>
<td>rSpHi</td>
</tr>
<tr>
<td>FALSE</td>
<td>indirect (heating)</td>
<td>rSpLo</td>
</tr>
</tbody>
</table>

**Heat recovery**

![Heat recovery diagram]

**VAR_INPUT**

- **bEn**: function block enable. If $bEn = FALSE$, all output parameters are 0.0.
- **rSpHi**: Upper setpoint input value to be checked.
- **rSpLo**: Lower setpoint input value to be checked.
- **rSpMax**: Maximum setpoint.
- **rSpMin**: Minimum setpoint.
- **bActnRcv**: Direction of action of the downstream heat recovery.
- **bSlcnSpRcv**: Setpoint selection of the downstream heat recovery system.

**VAR_OUTPUT**

- **rPrSpHi**: Output value for the upper setpoint.
**Programming**

- **rPrSpLo**: Output value for the lower setpoint.
- **rSpRcv**: Output value for the resulting heat recovery setpoint.
- **bErr**: This output is switched to TRUE if the parameters entered are erroneous.
- **sErrDescr**: Contains the error description.

<table>
<thead>
<tr>
<th>Error description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01: Warning: The setpoints are not in a logical order: Either (rSpMin \geq rSpMax) OR (rSpHi &lt; rSpMin) OR (rSpLo &gt; rSpMax)</td>
</tr>
</tbody>
</table>

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

### Room automation

#### Heating, cooling

#### Function blocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_BA_FnctSel [260]</td>
<td>Function selection (heating and/or cooling) in two- or four-pipe network.</td>
</tr>
<tr>
<td>FB_BA_RmTAdj [263]</td>
<td>Adjustment of the room temperature setpoint.</td>
</tr>
<tr>
<td>FB_BA_SpRmT [266]</td>
<td>Adjustment of the room temperature setpoint</td>
</tr>
</tbody>
</table>

**FB_BA_FnctSel**

The function block is used for enabling heating or cooling mode in a room.

The distribution network type plays a significant role:
In a two-pipe system, all rooms served by the plant can either be heated or cooled at the same time.
In a four-pipe system, the room conditioning can be demand-based, i.e. some rooms can be heated, while other rooms can be cooled by the same plant.

The function block used for each room, as already mentioned, selects its controllers, depending on which type of piping system is available:
Two-pipe network

The two-pipe system is selected if the function block has a FALSE entry at input bPipeSys. Since all rooms served by the plant can only either be heated or cooled, the choice is specified centrally for all rooms via the input bMedium. If bMedium is FALSE, the room heating controller is selected. If the input is TRUE the cooling controller is selected. The controller enable states bEnHtg and bEnCol are always issued with a delay of udiChgOvrDly_sec [s]. In other words, heating cannot be enabled until the cooling enable state bEnCol for udiChgOvrDly_sec is FALSE, and vice versa. In addition to the elapsing of this changeover time, the system checks that the output from controller to be switched off is 0.0. This is based on feedback at the inputs rCtrlValHtg and rCtrlValCol. In this way, a drastic change from heating to cooling and vice versa is avoided.

Four-pipe network

The four-pipe system is selected if the function block has a TRUE entry at input bPipeSys. In this case, the choice of controller can be different for the individual rooms as required, based on the room temperature rRmT and the setpoints rSpHtg for heating and rSpCol for cooling. If the room temperature exceeds the cooling setpoint, the cooling controller is activated (bEnCol), if it falls below the heating setpoint, the heating controller is activated (bEnHtg). If the temperature is between the two setpoints, both controllers are switched off (energy-neutral zone). Here too, the output of the controller enable states bEnHtg and bEnCol is delayed by udiChgOvrDly_sec [s] (see two-pipe network). In addition to the elapsing of this changeover time, the system checks that the output from controller to be switched off is 0.0. This is based on feedback at the inputs rCtrlValHtg and rCtrlValCol. In this way, a drastic change from heating to cooling and vice versa is avoided, if the changeover time is inadequate.

Dew-point monitor (bDewPnt)

In both systems (two- and four-pipe) the dew-point monitor has the task of deactivating cooling immediately, if required.

Program sequence

The function block can have 3 possible states:

1. Waiting for heating or cooling enable
2. Heating enable
3. Cooling enable

In the first step, the function block waits for compliance with the conditions required for heating or cooling:

<table>
<thead>
<tr>
<th>Heating</th>
<th>Cooling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cooling controller output = 0 (rCtrlValCol)</td>
<td>Heating controller output = 0 (rCtrlValHtg)</td>
</tr>
<tr>
<td>Room temperature (rRmT) &lt; heating setpoint (rSpHtg)</td>
<td>Room temperature (rRmT) &gt; cooling setpoint (rSpCol)</td>
</tr>
<tr>
<td>Cooling controller enable (bEnCol) is FALSE over at least the changeover time udiChgOvrDly_sec [s]</td>
<td>Heating controller enable (bEnHtg) is FALSE over at least the changeover time udiChgOvrDly_sec [s]</td>
</tr>
<tr>
<td>Four-pipe system is selected (bPipeSys=TRUE) OR two-pipe system is selected and heating medium is available (bPipeSys=FALSE AND bMedium=FALSE)</td>
<td>Four-pipe system is selected (bPipeSys=TRUE) OR two-pipe system is selected and cooling medium is available (bPipeSys=FALSE AND bMedium=TRUE)</td>
</tr>
<tr>
<td>The dew-point monitor does not respond (bDewPnt=TRUE)</td>
<td></td>
</tr>
</tbody>
</table>

If a chain of conditions is met, the function block switches to the respective state (heating or cooling) and remains in this state until the corresponding controller issues 0 at the function block input (rCtrlValHtg/rCtrlValCol). This ensures that only one controller is active at any one time, even if a high heating controller output, for example, would call for a brief cooling intervention (overshoot). Heating or cooling continues until there is no longer a demand.

There are 3 exceptions, for which heating or cooling is immediately interrupted:
1. A two-pipe system (\(b\text{PipeSys}=\text{FALSE}\)) is in heating mode (\(b\text{EnHtg}\)), but a switch to cooling medium occurred \(b\text{Medium}=\text{TRUE}\).
2. A two-pipe system (\(b\text{PipeSys}=\text{FALSE}\)) is in cooling mode (\(b\text{EnCol}\)), but a switch to heating medium occurred \(b\text{Medium}=\text{FALSE}\).
3. The dew-point monitor was triggered (\(b\text{DewPnt}=\text{TRUE}\)) in cooling mode (two- or four-pipe system).

In these cases the heating or cooling enable states are canceled, and the plant switches to standby.

**Demand message (\(\text{udiReqdMedium}\))**

To notify the plant of the current demand for heating or cooling, a demand ID is issued at the function block output, i.e. for each room, depending on the actual and set temperature. These can be collected and evaluated centrally. The evaluation always takes place, irrespective of the network type (two- or four-pipe).

<table>
<thead>
<tr>
<th>(\text{udiReqdMedium} )</th>
<th>Medium</th>
<th>Room temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No medium is requested</td>
<td>(r\text{RmT} &gt; r\text{SpHtg AND rRmT} &lt; r\text{SpCol})</td>
</tr>
<tr>
<td>2</td>
<td>Heating medium is requested</td>
<td>(r\text{RmT} &lt; r\text{SpHtg})</td>
</tr>
<tr>
<td>3</td>
<td>Cooling medium is requested</td>
<td>(r\text{RmT} &gt; r\text{SpCol})</td>
</tr>
</tbody>
</table>

**Error handling**

The heating setpoint must not be greater than or equal to the cooling setpoint, since this would result in temperature range with simultaneous heating and cooling demand. However, since the function block only issues one enable state at a time (i.e. heating or cooling), the case is harmless from a plant engineering perspective. In this case only a warning message is issued (\(b\text{Err}=\text{TRUE}\), \(s\text{ErrDescr}=\text{warning message}\)); the function block does not interrupt its cycle.

**VAR_INPUT**

- \(b\text{PipeSys} : \text{BOOL};\)
- \(b\text{Medium} : \text{BOOL};\)
- \(b\text{DewPnt} : \text{BOOL};\)
- \(r\text{RmT} : \text{REAL};\)
- \(r\text{SpHtg} : \text{REAL};\)
- \(r\text{SpCol} : \text{REAL};\)
- \(r\text{CtrlValHtg} : \text{REAL};\)
- \(r\text{CtrlValCol} : \text{REAL};\)
- \(\text{udiChgOvrDel_sec} : \text{UDINT};\)

- \(b\text{PipeSys}: \) In two-pipe system \(b\text{PipeSys}\) is FALSE, in four-pipe systems it is TRUE.
- \(b\text{Medium}: \) Current supply of the whole two-pipe network with cooling or heating medium. If heating medium is active, \(b\text{Medium}\) is FALSE.
- \(b\text{DewPnt}: \) Dew-point monitor: If \(b\text{DewPnt} = \text{FALSE}\), the cooling controller is locked.
- \(r\text{RmT}: \) Room temperature.
- \(r\text{SpHtg}: \) Heating setpoint.
- \(r\text{SpCol}: \) Cooling setpoint.
- \(r\text{CtrlValHtg}: \) Current output value of the heating controller. Used internally as switching criterion from heating to cooling: \(r\text{CtrlValHtg}\) must be 0.
- \(r\text{CtrlValCol}: \) Current output value of the cooling controller. Used internally as switching criterion from cooling to heating: \(r\text{CtrlValCol}\) must be 0.
- \(\text{udiChgOvrDel_sec}: \) Switchover delay [s] from heating to cooling or vice versa. Internally limited to a minimum value of 0.

**VAR_OUTPUT**

- \(b\text{EnHtg} : \text{BOOL};\)
- \(b\text{EnCol} : \text{BOOL};\)
- \(\text{udiReqdMedium} : \text{UDINT};\)
bEnHtg: Heating controller enable.

bEnCol: Cooling controller enable.

<table>
<thead>
<tr>
<th>udiReqdMedium</th>
<th>Medium</th>
<th>Room temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No medium is requested</td>
<td>$r_{RmT} &gt; r_{SpHtg}$ AND $r_{RmT} &lt; r_{SpCol}$</td>
</tr>
<tr>
<td>2</td>
<td>Heating medium is requested</td>
<td>$r_{RmT} &lt; r_{SpHtg}$</td>
</tr>
<tr>
<td>3</td>
<td>Cooling medium is requested</td>
<td>$r_{RmT} &gt; r_{SpCol}$</td>
</tr>
</tbody>
</table>

udiRemTiChgOvrDlyHtg_sec: Countdown [s] for switchover delay from cooling to heating.

udiRemTiChgOvrDlyCol_sec: Countdown [s] for switchover delay from heating to cooling.

bErr: In case of a fault, e.g. if warning stages are active, this output is set to TRUE.

sErrDescr: Contains the error description.

Error description
01: Warning: The heating setpoint is higher than or equal to the cooling setpoint

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**FB_BA_RmTAdj**

The function block **FB_BA_RmTAdj** is used for user adjustment of the room temperature setpoint. It shifts the setpoints at the input of a function block depending on an offset $r_{RmTAdj}$, as shown in the following diagram. At the $r_{RmTAdj}$ input, the value of a resistance potentiometer or a bus-capable field device can be used for the setpoint correction.
If the set value $rRmTAdj$ is greater than zero, room heating is requested: The Comfort Heating value is increased by $rRmTAdj$. At the same time, the values for Comfort Cooling and PreComfort Cooling are increased. If the value $rRmTAdj$ is less than zero, a lower room temperature is requested. Analog to the heating case, the values for Comfort Cooling, Comfort Heating and PreComfort Heating are now reduced by the value $rRmTAdj$.

**Auto-correction**

The temperature adjustment is intended for small corrections of the values. Although it is possible to enter any input values, a heating system will only work in a meaningful manner if the setpoints have ascending values in the following order:
- Protection Heating
- Economy Heating
- Precomfort Heating
- Comfort Heating
- Comfort Cooling
- Precomfort Cooling
- Economy Cooling
- Protection Cooling

Auto-correction works according to the following principle: Starting with the value Economy Heating, the system checks whether this value is smaller than the lower value of Protection Heating. If this is the case, the value for Economy Heating is adjusted to match the value for Protection Heating. The system then checks whether the value for Precomfort Heating is less than Economy Heating and so on, until the value for Protection Cooling is compared with the value for Economy Cooling. If one or several values were corrected, this is indicated with a TRUE signal at output $bValCorr$.

**VAR_INPUT**

| rRmTAdj    : REAL; |
| stSp       : ST_BA_SpRmT; |

$rRmTAdj$: Room temperature offset value.

$stSp$: Input structure for the setpoints (see $ST_BA_SpRmT$).

**VAR_OUTPUT**

| bValCorr   : BOOL; |
| rPrPrtcHtg : REAL; |
| rPrEcoHtg  : REAL; |
| rPrPreCmfHtg : REAL; |
| rPrCmfHtg  : REAL; |
| rPrPrtcCol : REAL; |
| rPrEcoCol  : REAL; |
| rPrPreCmfCol : REAL; |
| rPrCmfCol  : REAL; |
| stPrSp     : ST_BA_SpRmT; |

$bValCorr$: Autocorrection for the values was performed, see above.

$rPrPrtcHtg$: Resulting Protection Heating setpoint.

$rPrEcoHtg$: Resulting Economy Heating setpoint.

$rPrPreCmfHtg$: Resulting PreComfort Heating setpoint.

$rPrCmfHtg$: Resulting Comfort Heating setpoint.

$rPrCmfCol$: Resulting Comfort Cooling setpoint.

$rPrPreCmfCol$: Resulting PreComfort Cooling setpoint.

$rPrEcoCol$: Resulting Economy Cooling setpoint.

$rPrPrtcCol$: Resulting Protection Cooling setpoint.

$stPrSp$: Consolidated output of the resulting values in a structure (see $ST_BA_SpRmT$).

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
The function block \textit{FB\_BA\_SpRmT} assigns setpoints for cooling and heating operation to each of the energy levels Protection, Economy, PreComfort and Comfort.

The following graphics illustrates the behavior of the function block; the entered values should be regarded as examples:
The parameter $r_{\text{ShiftHtg}}$ is applied to the Comfort and Precomfort values for the heating mode as central setpoint shift. In addition, winter compensation $r_{\text{WinCpsn}}$ is applied.

Similarly, the following applies for the cooling mode: The parameter $r_{\text{ShiftCol}}$ is applied to the Comfort and Precomfort values. In addition, the summer compensation value $r_{\text{SumCpsn}}$ is applied.
Auto-correction

The setpoint shift is intended for small corrections of the values. Although it is possible to enter any input values, a heating system will only work in a meaningful manner if the setpoints have ascending values in the following order:

- Protection Heating
- Economy Heating
- Precomfort Heating
- Comfort Heating
- Precomfort Cooling
- Comfort Cooling
- Economy Cooling
- Protection Cooling

Auto-correction works according to the following principle: Starting with the value Economy Heating, the system checks whether this value is smaller than the lower value of Protection Heating. If this is the case, the value for Economy Heating is adjusted to match the value for Protection Heating. The system then checks whether the value for Precomfort Heating is less than Economy Heating and so on, until the value for Protection Cooling is compared with the value for Economy Cooling. If one or several values were corrected, this is indicated with a TRUE signal at output \( b\text{ValCorr} \).

VAR_INPUT

\[
\begin{align*}
\text{rSumCpsn} &: \text{REAL}; \\
\text{rWinCpsn} &: \text{REAL};
\end{align*}
\]

- \( r\text{SumCpsn} \): Summer compensation value
- \( r\text{WinCpsn} \): Winter compensation value

VAR_OUTPUT

\[
\begin{align*}
\text{bValCorr} &: \text{BOOL}; \\
\text{rPrPrtcHtg} &: \text{REAL}; \\
\text{rPrEcoHtg} &: \text{REAL}; \\
\text{rPrPreCmfHtg} &: \text{REAL}; \\
\text{rPrCmfHtg} &: \text{REAL}; \\
\text{rPrPreCmfCol} &: \text{REAL}; \\
\text{rPrCmfCol} &: \text{REAL}; \\
\text{rPrEcoCol} &: \text{REAL}; \\
\text{rPrPrtcCol} &: \text{REAL}; \\
\text{stPrSp} &: \text{ST_BA_SpRmT};
\end{align*}
\]

- \( b\text{ValCorr} \): Autocorrection: At least one of the resulting setpoints was adjusted such that the values continue to monotonically increase.
- \( r\text{PrPrtcHtg} \): Resulting Protection Heating setpoint.
- \( r\text{PrEcoHtg} \): Resulting Economy Heating setpoint.
- \( r\text{PrPreCmfHtg} \): Resulting PreComfort Heating setpoint.
- \( r\text{PrCmfHtg} \): Resulting Comfort Heating setpoint.
- \( r\text{PrPreCmfCol} \): Resulting PreComfort Cooling setpoint.
- \( r\text{PrCmfCol} \): Resulting Comfort Cooling setpoint.
- \( r\text{PrEcoCol} \): Resulting Economy Cooling setpoint.
- \( r\text{PrPrtcCol} \): Resulting Protection Cooling setpoint.
- \( st\text{PrSp} \): Consolidated output of the resulting values in a structure (see \( \text{ST_BA_SpRmT} \) [416]).
VAR_INPUT_CONSTANT_PERSISTENT (Parameter)

\[ \begin{align*}
\text{rShiftCol} & : \text{REAL} := 0; \\
\text{rShiftHtg} & : \text{REAL} := 0; \\
\text{rPrtcCol} & : \text{REAL} := 35; \\
\text{rEcoCol} & : \text{REAL} := 28; \\
\text{rPreCmfCol} & : \text{REAL} := 25; \\
\text{rCmfCol} & : \text{REAL} := 23; \\
\text{rCmfHtg} & : \text{REAL} := 21; \\
\text{rPreCmfHtg} & : \text{REAL} := 18; \\
\text{rEcoHtg} & : \text{REAL} := 14; \\
\text{rPrtcHtg} & : \text{REAL} := 6; \\
\end{align*} \]

- \text{rShiftCol}: Cooling setpoint value shift.
- \text{rShiftHtg}: Heating setpoint value shift.
- \text{rPrtcCol}: Basic Protection Cooling setpoint.
- \text{rEcoCol}: Basic Economy Cooling setpoint.
- \text{rPreCmfCol}: Basic PreComfort Cooling setpoint.
- \text{rCmfCol}: Basic Comfort Cooling setpoint.
- \text{rCmfHtg}: Basic Comfort Heating setpoint.
- \text{rPreCmfHtg}: Basic PreComfort Heating setpoint.
- \text{rEcoHtg}: Basic Economy Heating setpoint.
- \text{rPrtcHtg}: Basic Protection Heating setpoint.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

Lighting

Function blocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_BA_Toggle</td>
<td>Switching of lamps.</td>
</tr>
</tbody>
</table>

FB_BA_Toggle

The function block is used to switch an actuator on or off.

The input \( bEn \) is used for enabling the function block.

A positive edge at the input \( bOn \) results in setting of output \( bQ \). The output is reset by a rising edge at the \( bOff \) input. If a rising edge is presented to \( bToggle \), the output is negated; i.e., if On it goes Off, and if Off it goes On.
VAR_INPUT

bEn : BOOL;
bOn : BOOL;
bOff : BOOL;
bToggle : BOOL;

bEn: Function block enable.
bOn: Switches the output on.
bOff: Switches the output off.
bToggle: Negates the current output state.

VAR_OUTPUT

bQ : BOOL;

bQ: Control output.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

Shading

Overview of shading correction [272]

Shading correction: Basic principles and definitions [272]

Overview of automatic sun protection [280]

Sun protection: Basic principles and definitions [282]

List of shading elements [287]

List of facade elements [287]
### Function blocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_BA_BldPosEntry [287]</td>
<td>Sun protection function: Input of blind positions.</td>
</tr>
<tr>
<td>FB_BA_CalcSunPos [289]</td>
<td>Calculation of sun position</td>
</tr>
<tr>
<td>FB_BA_InRngAzm [296]</td>
<td>Verification of valid sun position and sun direction range (azimuth angle)</td>
</tr>
<tr>
<td>FB_BA_InRngElv [298]</td>
<td>Verification of valid sun position and sun elevation range (elevation angle)</td>
</tr>
<tr>
<td>FB_BA_RdFcdElemLst [301]</td>
<td>Shading correction: Input of facade elements via data list (csv).</td>
</tr>
<tr>
<td>FB_BA_RdShdObjLst [305]</td>
<td>Shading correction: Input of shading objects via data list (csv).</td>
</tr>
<tr>
<td>FB_BA_RolBldActr [309]</td>
<td>Roller shutter actuator</td>
</tr>
<tr>
<td>FB_BA_ShdCorr [311]</td>
<td>Shading correction function block</td>
</tr>
<tr>
<td>FB_BA_ShdObjEntry [314]</td>
<td>Shading correction: Input of shading objects per function block.</td>
</tr>
<tr>
<td>FB_BA_SunBldActr [317]</td>
<td>Blind actuator</td>
</tr>
<tr>
<td>FB_BA_SunBldEvt [322]</td>
<td>Output of a specified blind position and angle in percent</td>
</tr>
<tr>
<td>FB_BA_SunBldIcePrtc [323]</td>
<td>Anti-icing</td>
</tr>
<tr>
<td>FB_BA_SunBldPosDly [324]</td>
<td>Switch-on delay for blinds/groups of blinds</td>
</tr>
<tr>
<td>FB_BA_SunBldPrioSwi4 [325]</td>
<td>Priority control, 4 inputs</td>
</tr>
<tr>
<td>FB_BA_SunBldPrioSwi8 [326]</td>
<td>Priority control, 8 inputs</td>
</tr>
<tr>
<td>FB_BA_SunBldScn [327]</td>
<td>Manual operation with scene selection and programming</td>
</tr>
<tr>
<td>FB_BA_SunBldTwilightAuto [332]</td>
<td>Automatic twilight function</td>
</tr>
<tr>
<td>FB_BA_SunBldWndPrtc [333]</td>
<td>Protection against wind damage</td>
</tr>
<tr>
<td>FB_BA_SunPrtc [335]</td>
<td>Sun protection function, see Overview of automatic sun protection (shading correction)</td>
</tr>
</tbody>
</table>
Overview of shading correction

Shading correction: Basic principles and definitions

The shading correction can be used in conjunction with the automatic sun function or louvre adjustment. The function checks whether a window or a window group that is assigned to a room, for example, is temporarily placed in the shade by surrounding buildings or parts of its own building. Sun shading for windows that stand in the shadow of surrounding buildings or trees is not necessary and may even be disturbing under certain circumstances. On the basis of data entered regarding the facade and its surroundings, the shading correction determines which parts of the front are in the shade. Hence, it is then possible to decide whether the sun protection should be active for individual windows or window groups. Apart from the current position of the sun, the shading of the individual windows depends on three things:
• the orientation of the facade
• the position of the windows
• the positioning of the shading objects

The following illustrations are intended to describe these interrelationships and to present the parameters to be entered.

**Orientation of the facade**

**Observation from above**

For the pure observation of the shadow thrown on the facade, a two-dimensional coordinate system is ultimately required, therefore the X and Y axis were placed on the facade. The zero point is thereby at the bottom left on the base, as if one were regarding the facade from the front. For the calculation of the shading objects the Z component is then also added. Its axis points from away the facade and has the same zero point as the X and Y axis.

In the northern hemisphere, the horizontal sun position (azimuth angle) is determined from the north direction by definition. The facade orientation is likewise related to north, wherein the following applies to the line of sight from a window in the facade:

<table>
<thead>
<tr>
<th>Line of sight</th>
<th>Facade orientation</th>
</tr>
</thead>
<tbody>
<tr>
<td>North</td>
<td>( \beta = 0^\circ )</td>
</tr>
<tr>
<td>East</td>
<td>( \beta = 90^\circ )</td>
</tr>
<tr>
<td>South</td>
<td>( \beta = 180^\circ )</td>
</tr>
<tr>
<td>West</td>
<td>( \beta = 270^\circ )</td>
</tr>
</tbody>
</table>

In the southern hemisphere is the sun path is the other way round: Although it also rises in the east, at midday it is in the north. The facade orientation is adjusted to this path:

<table>
<thead>
<tr>
<th>Line of sight</th>
<th>Facade orientation</th>
</tr>
</thead>
<tbody>
<tr>
<td>South</td>
<td>( \beta = 0^\circ )</td>
</tr>
<tr>
<td>East</td>
<td>( \beta = 90^\circ )</td>
</tr>
<tr>
<td>North</td>
<td>( \beta = 180^\circ )</td>
</tr>
<tr>
<td>West</td>
<td>( \beta = 270^\circ )</td>
</tr>
</tbody>
</table>

For convenience, the other explanations refer to the northern hemisphere. The calculations for the southern hemisphere are analogous. When the function block `FB_BA_ShdCorr` (shading correction) is parameterized they are activated through a boolean input, `bSouth`

The following two illustrations are intended to further clarify the position of the point of origin \( P_0 \) as well as the orientation of the coordinate system:
Observation from the side

The angle of elevation (height of the sun) can be represented using this illustration: by definition this is 0° at sunrise (horizontal incidence of light) and can reach maximally 90°, but this applies only to places within the Tropic of Cancer and the Tropic of Capricorn.

Observation from the front

Here, the position of the point of origin, \( P_0 \), at the bottom left base point of the facade is once more very clear. Beyond that the X-Y orientation is illustrated, which is important later for the entry of the window elements.

Position of the windows

The position of the windows is defined by the specification of their bottom left corner in relation to the facade coordinate system. Since a window lies flat on the facade, the entry is restricted to the X and Y coordinates.
In addition, the window width and the window height have to be specified.

The position of each window corner on the facade is determined internally from the values entered. A window is considered to be in the shade if all corners lie in the shade.

**Positioning of the shading objects**

When describing the shading objects, distinction is made between angular objects (building, column) and objects that are approximately spherical (e.g. trees). Angular objects can be subdivided into rectangular shadow-casting facades depending on their shadow projection; you should consider which surfaces cast the main shadows over the day:
Morning/noon

In the morning and around noon, the shadow is mainly cast by the sides $S_1$ and $S_4$. $S_2$ and $S_3$ do not have to be considered, unless they are higher.
In the afternoon and evening, the total shade can be determined solely through $S_1$ and $S_2$. In this case it is therefore sufficient to specify $S_1$, $S_2$, and $S_4$ as shadow casters. The entry is made on the basis of the four corners or their coordinates in relation to the zero point of the facade:
In this sketch only the upper points, P₂ and P₃, are illustrated due to the plan view. The lower point P₁ lies underneath P₂ and P₄ lies underneath P₃.

The input of shadow-casting ball elements is done by entering the center of the ball and its radius:

**Ball elements**

<table>
<thead>
<tr>
<th>Meaning of colors</th>
<th>Priority selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>effective for the whole building</td>
<td>FB_BA_SunBldPrioSwi</td>
</tr>
<tr>
<td>effective for one facade</td>
<td>Prio1-input</td>
</tr>
<tr>
<td>effective for a room/group of rooms</td>
<td>Prio2-input</td>
</tr>
</tbody>
</table>

- **Ice protection**
  - FB_BA_SunBldIcePrtc

- **Wind protection**
  - FB_BA_SunBldWndPrtc

- **Maintenance position**
  - FB_BA_SunBldEvt

- **Manual mode**
  - FB_BA_SunBldSwi
  - or
  - FB_BA_SunBldScn

- **Twilight-automatic**
  - FB_BA_SunBldTwLgtAuto

- **Thermo-automatic**
  - FB_BA_SunBldEvt

- **Sun-protection**
  - FB_BA_SunPrtc

- **Parking-position**
  - FB_BA_SunBldEvt

- **Positioning telegram**
  - ST_BA_SunBld

- **Sunblind-actuator**
  - FB_BA_SunBldActr
A "classification" of the ball element as in the case of the angular building is of course unnecessary, since the shadow cast by a ball changes only its direction, but not its size.
Overview of automatic sun protection
Meaning of colors
- effective for the whole building
- effective for one facade
- effective for a room/group of rooms

Priority selection
- FB_BA_SunBldPrioSwi

icet protection
- FB_BA_SunBldIcePrtc

wind protection
- FB_BA_SunBldWndPrtc

maintenance position
- FB_BA_SunBldEvt

manual mode
- FB_BA_SunBldSwi
  or
- FB_BA_SunBldScn

twilight-automatic
- FB_BA_SunBldTwLgtAuto

thermo-automatic
- FB_BA_SunBldEvt

sun-protection
- FB_BA_SunPrtc

parking-position
- FB_BA_SunBldEvt

positioning telegram
- ST_BA_SunBld

sunblind-actuator
- FB_BA_SunBldActr

output

TF8040
Version: 1.4
Sun protection: Basic principles and definitions

The direct incidence of daylight is regarded as disturbing by persons in rooms. On the other hand, however, people perceive natural light to be more pleasant in comparison with artificial light. Two options for glare protection are to be presented here:

- Louvre adjustment
- Height adjustment

Louvre adjustment

A louvered blind that can be adjusted offers the option of intelligent sun protection here. The position of the louvres is cyclically adapted to the current position of the sun, so that no direct daylight enters through the blinds, but as much diffuse daylight can be utilized as possible.

The illustration shows that diffuse light can still enter from underneath, whereas no further direct daylight, or theoretically only a single ray, can enter. The following parameters are necessary for the calculation of the louvre angle:

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

Effective elevation angle

If the blind is viewed in section as above, the angle of incidence does not depend solely on the solar altitude (elevation), but also on the direction of the sun:

- If the facade orientation and the sun position (azimuth) are the same, i.e. the sunlight falls directly onto the facade, the effective light incidence angle is the same as the current elevation angle.
- However, if the sunlight falls at an angle onto the facade as seen from the sun direction, the effective angle is larger for the same angle of elevation.

This relationship can easily be illustrated with a set square positioned upright on the table: Viewed directly from the side you can see a triangle with two 45° angles and one 90° angle. If the triangle is rotated, the side on the table appears to become shorter and the two original 45° angles change. The triangle appears to be getting steeper.

We therefore refer to the "effective elevation angle", which describes the proportion of light that falls directly onto the blind.
The following three images illustrate the relationship between the effective elevation angle and the blind dimensions, and how the resulting louvre angle $\lambda$ changes during the day:

**louvre-angle**

**louvre at an angle of $\lambda=0$**

**Louvre-angle in the morning and in the evening**

$\lambda$ = louvre-angle, in this drawing: $\lambda<0$

$\varepsilon_{\text{eff}}$ = effective sun-elevation

**Louvre angle at noon**

$\lambda$ = louvre-angle, in this drawing: $\lambda>0$

$\varepsilon_{\text{eff}}$ = effective sun-elevation
**Height adjustment**

With a high position of the sun at midday, the direct rays of sunlight do not penetrate into the full depth of the room. If direct rays of sunlight in the area of the window sill are regarded as uncritical, the height of the sun protection can be adapted automatically in such a way that the rays of sunlight only ever penetrate into the room up to an uncritical depth.

![Diagram of sunlight incidence and uncritical range]

In order to be able to calculate at any time the appropriate blind height that guarantees that the incidence of sunlight does not exceed a certain value, the following values are necessary.

Required for the calculation of the respective blind height:

- Height of the sun (elevation)
- Window height
- Distance between the window and the floor

The following illustration shows where these parameters are to be classified:
Influence of the facade inclination

In both of the methods of sun protection described, it was assumed that the facade and thus the windows are perpendicular to the ground. In the case of an inclined facade, however, the incidence of light changes such that this influence will also be taken into account. The facade inclination is defined as follows:
facade angle: $\varphi = 0^\circ$

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

facade angle: $\varphi > 0^\circ$
List of shading elements

The data of all shading objects (building components, trees, etc.) per facade are stored in a field of structure elements of type ST_BA_ShdObj [415] within the program.

The shading correction FB_BA_ShdCorr [311] reads the information from this list. The management function block FB_BA_ShdObjEntry [314] reads and writes it as input/output variable. It is therefore advisable to declare this list globally:

VAR_GLOBAL
   arrShdObj : ARRAY[1..gBA_cMaxShdObj] OF ST_BA_ShdObj;
END_VAR

The variable gBA_cMaxShdObj represents the upper limit of the available elements and is defined as a global constant within the program library:

VAR_GLOBAL CONSTANT
   gBA_cMaxShdObj : INT := 20;
END_VAR

List of facade elements

The data of all windows (facade elements) per facade are saved within the program in a field of structure elements of the type ST_BA_FcdElem [414].

The management function block FB_BA_FcdElemEntry [291] and the shading correction FB_BA_ShdCorr [311] read and write to this list (the latter sets the shading information); they access this field as input/output variables. It is therefore advisable to declare this list globally:

VAR_GLOBAL
   arrFcdElem : ARRAY[1..uiMaxRowFcd, 1..uiMaxColumnFcd] OF ST_BA_FcdElem;
END_VAR

The variables uiMaxColumnFcd and uiMaxRowFcd define the upper limit of the available elements and are declared as global constants within the program library:

VAR_GLOBAL CONSTANT
   uiMaxRowFcd : UINT :=10;
   uiMaxColumnFcd : UINT :=20;
END_VAR

FB_BA_BldPosEntry

This function block is used for entering interpolation points for the function block FB_BA_SunPrtc [335], if this function block is operated in height positioning mode with the aid of a table, see E_BA_PosMod [412].

In addition to the operating modes “Fixed shutter height” and “Maximum incidence of light”, the function block FB_BA_SunPrtc [335] also offers the possibility to control the shutter height in relation to the position of the sun by means of table entries. By entering several interpolation points, the shutter height relative to the respective sun position is calculated by linear interpolation. However, since incorrectly entered values
can lead to malfunctions in $\text{FB\_BA\_SunPrtc}[335]$, this function block is to be preceded by the function block $\text{FB\_BA\_BldPosEntry}$. Four interpolation points can be parameterized on this function block, whereby a missing entry is evaluated as a zero entry.

The function block does not sort the values entered independently, but instead ensures that the positions of the sun entered in the respective interpolation points are entered in ascending order. Unintentional erroneous entries are noticed faster as a result.

The values chosen for $r\text{SunElv1}..r\text{SunElv4}$ must be unique; for example, the following situation must be avoided:

\[
[r\text{SunElv1} = 10; r\text{Pos1} = 50] \text{ and at the same time } [r\text{SunElv2} = 10; r\text{Pos2} = 30].
\]

This would mean that there would be two different target values for one and the same value, which does not allow a unique functional correlation to be established.

On top of that the entries for the position of the sun and shutter height must lie within the valid range.

Mathematically this means that the following conditions must be satisfied:

- $r\text{SunElv1} < r\text{SunElv2} < r\text{SunElv3} < r\text{SunElv4}$ - (values ascending and not equal)
- $0 \leq r\text{SunElv} \leq 90$ ([$^\circ$] - scope source values)
- $0 \leq r\text{Pos} \leq 100$ (in percent - scope target values)

The function block checks the entered values for these conditions and issues an error message if they are not met. In addition, the value $b\text{Valid}$ of $\text{ST\_BA\_BldPosTab}[413]$ is set to FALSE.

Furthermore the function block independently ensures that the boundary areas are filled out: Internally, a further interpolation point is set at $r\text{SunElv} = 0$ with $r\text{BldPos1}$ and another one above $r\text{SunElv4}$ at $r\text{SunElv} = 90$ with $r\text{BldPos4}$. This ensures that a meaningful target value is available for all valid input values $0 \leq r\text{SunElv} \leq 90$, without the user having to enter $r\text{SunElv} = 0$ and $r\text{SunElv} = 90$:

This increases the actual number of interpolation points transferred to the function block $\text{FB\_BA\_SunPrtc}[335]$ to 6; see $\text{ST\_BA\_BldPosTab}[413]$.

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

**VAR INPUT**

\[
\begin{align*}
r\text{SunElv1} &: \text{REAL} \\
r\text{Pos1} &: \text{REAL} \\
r\text{SunElv2} &: \text{REAL} \\
r\text{Pos2} &: \text{REAL} \\
r\text{SunElv3} &: \text{REAL} \\
r\text{Pos3} &: \text{REAL} \\
r\text{SunElv4} &: \text{REAL} \\
r\text{Pos4} &: \text{REAL}
\end{align*}
\]

$r\text{SunElv1}$: Sun position at the first interpolation point ($0^\circ$..$90^\circ$).
rPos1: Blind position (degree of closure) at the first interpolation point (0%..100%).

rSunElv2: Sun position at the second interpolation point (0°..90°).

rPos2: Blind position (degree of closure) at the second interpolation point (0%..100%).

rSunElv3: Sun position at the third interpolation point (0°..90°).

rPos3: Blind position (degree of closure) at the third interpolation point (0%..100%).

rSunElv4: Sun position at the fourth interpolation point (0°..90°).

rPos4: Blind position (degree of closure) at the fourth interpolation point (0%..100%).

VAR_OUTPUT

stBldPosTab : ST_BA_BldPosTab;
bErr : BOOL;
sErrDescr : T_MAXSTRING;

stBldPosTab: Transfer structure of the interpolation points, see ST_BA_BldPosTab [413].
bErr: This output is switched to TRUE if the parameters entered are erroneous.
sErrDescr: Contains the error description.

Error description

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Error: The x-values (elevation values) in the table are either not listed in ascending order, or they are duplicated.</td>
</tr>
<tr>
<td>02</td>
<td>Error: An elevation value that was entered is outside the valid range of 0°...90°.</td>
</tr>
<tr>
<td>03</td>
<td>Error: An position value that was entered is outside the valid range of 0°...100%.</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

FB_BA_CalcSunPos

Calculation of sun position based on the date, time, longitude and latitude.

The position of the sun for a given point in time can be calculated according to common methods with a defined accuracy. For applications with moderate requirements, the present function block is sufficient. As the basis for this, the SUNAE algorithm was used, which represents a favorable compromise between accuracy and computing effort.

The position of the sun at a fixed observation point is normally determined by specifying two angles. One angle indicates the height above the horizon; 0° means that the sun is in the horizontal plane of the location; a value of 90° means that the is perpendicular to the observer. The other angle indicates the direction at which the sun is positioned. The SUNAE algorithm is used to distinguish whether the observer is standing on the northern hemisphere (longitude > 0 degrees) or on the southern hemisphere (longitude < 0 degrees) of the earth. If the observation point is in the northern hemisphere, then a value of 0° is assigned for the northern sun direction and it then runs in the clockwise direction around the compass, i.e. 90° is east, 180° is south, 270° is west etc. If the point of observation is in the southern hemisphere, then 0° corresponds to the southern direction and it then runs in the counter clockwise direction, i.e. 90° is east, 180° is north, 270° is west etc.
The time has to be specified as coordinated world time (UTC, Universal Time Coordinated, previously referred to as GMT, Greenwich Mean Time).

The latitude is the northerly or southerly distance of a location on the Earth’s surface from the equator, in degrees [°]. The latitude can assume a value from 0° (at the equator) to ±90° (at the poles). A positive sign thereby indicates a northern direction and a negative sign a southern direction. The longitude is an angle that can assume values up to ±180° starting from the prime meridian 0° (an artificially determined North-South line). A positive sign indicates a longitude in an eastern direction and a negative sign in a western direction.

Examples:

<table>
<thead>
<tr>
<th>Location</th>
<th>Longitude</th>
<th>Latitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sydney, Australia</td>
<td>151.2°</td>
<td>-33.9°</td>
</tr>
<tr>
<td>New York, USA</td>
<td>-74.0</td>
<td>40.7°</td>
</tr>
<tr>
<td>London, England</td>
<td>-0.1°</td>
<td>51.5°</td>
</tr>
<tr>
<td>Moscow, Russia</td>
<td>37.6°</td>
<td>55.7°</td>
</tr>
<tr>
<td>Peking, China</td>
<td>116.3°</td>
<td>39.9°</td>
</tr>
<tr>
<td>Dubai, United Arab Emirates</td>
<td>55.3°</td>
<td>25.4°</td>
</tr>
<tr>
<td>Rio de Janeiro, Brazil</td>
<td>-43.2°</td>
<td>-22.9°</td>
</tr>
<tr>
<td>Hawaii, USA</td>
<td>-155.8°</td>
<td>20.2°</td>
</tr>
<tr>
<td>Verl, Germany</td>
<td>8.5°</td>
<td>51.9°</td>
</tr>
</tbody>
</table>

If the function block `FB_BA_CalcSunPos` returns a negative value for the solar altitude `rSunElv`, the sun is invisible. This can be used to determine sunrise and sunset.

**VAR_INPUT**

rDegLngd : REAL;
rDegLatd : REAL;
stUTC : TIMESTRUCT;

rDegLngd: Longitude [°].

rDegLatd: Latitude [°].

stUTC: Input of the current time as coordinated world time (see TIMESTRUCT). The function block `FB_BA_GetTime` can be used to read this time from a target system.

**VAR_OUTPUT**

rSunAzm : REAL;
rSunElv : REAL;
rSunAzm: Direction of the sun (northern hemisphere: 0° north ... 90° east ... 180° south ... 270° west ... / southern hemisphere: 0° south ... 90° east ... 180° north ... 270° west ...).

rSunElv: Height of the sun (0° horizontal - 90° vertical).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

FB_BA_FcdElemEntry

This function block serves the administration of all facade elements (windows) in a facade, which are saved globally in a list of facade elements [287]. It is intended to facilitate inputting element information - not least with regard to using the TC3 PLC HMI. A schematic illustration of the objects with description of the coordinates is given in Shading correction: principles and definitions [272].

The facade elements are declared in the global variables as a two-dimensional field above the window columns and rows:

```plaintext
VAR_GLOBAL
  arrFcdElem : ARRAY[1..Param.uiMaxColumnFcd, 1..Param.uiMaxRowFcd] OF ST_BA_FcdElem;
END_VAR
```

Each element arrFcdElem[x,y] contains the information for an individual facade element (ST_BA_FcdElem [414]). The information includes the group affiliation, the dimensions (width, height) and the coordinates of the corners. The function block thereby accesses this field directly via the IN-OUT variable arrFcdElem.

Note: The fact that the coordinates of corners C2 to C4 are output values arises from the fact that they are formed from the input parameters and are to be available for use in a visualization:
All entries in [m]!

- \( r_{Cnr2X} = r_{Cnr1X} \)
- \( r_{Cnr2Y} = r_{Cnr1Y} + r_{WdwHght} \) (window height)
- \( r_{Cnr3X} = r_{Cnr1X} + r_{WdwWdth} \) (window width)
- \( r_{Cnr3Y} = r_{Cnr2Y} \)
- \( r_{Cnr4X} = r_{Cnr1X} + r_{WdwWdth} \) (window width)
- \( r_{Cnr4Y} = r_{Cnr1Y} \)

The function block is used in three steps:

- Read
- Change
- Write

Read

The entries \textit{udiColumn} and \textit{udiRow} are used to select the corresponding element from the list, \textit{arrFcdElem[udiColumn, udiRow]}. A rising edge on \textit{bRd} reads the following data from the list element:

- \textit{usiGrp} group membership,
- \( r_{Cnr1X} \) x-coordinate of corner point 1 [m]
- \( r_{Cnr1Y} \) y-coordinate of corner point 1 [m]
- \( r_{WdwWdth} \) window width [m]
- \( r_{WdwHght} \) window height [m]

These are then assigned to the corresponding input variables of the function block, which uses them to calculate the coordinates of corners C2-C4 as output variables in accordance with the correlation described above. It is important here that the input values are not overwritten in the reading step. Hence, all values can initially be displayed in a visualization.
Change

In a next program step the listed input values can then be changed. The values entered are constantly checked for plausibility. The output \( bErr \) indicates whether the values are valid (\( bErr=FALSE \)). If the values are invalid, a corresponding error message is issued at output \( sErrDescr \). See also "Error (\( bErr=TRUE \))" below.

Write

The parameterized data are written to the list element with the index \( nId \) upon a rising edge on \( bWrt \), regardless of whether they represent valid values or not. The element structure \( \text{ST\_BA\_FcdElem} [414] \) therefore also contains a plausibility bit \( bVld \), which forwards precisely this information to the function block \( \text{FB\_BA\_ShdCorr} [311] \) to prevent miscalculations.

This approach is to be regarded only as a proposal. It is naturally also possible to parameterize the function block quite normally in one step and to write the values entered to the corresponding list element with a rising edge on \( bWrt \).

Error (\( bErr=TRUE \))

The function block \( \text{FB\_BA\_ShdCorr} [311] \), which judges whether all windows in a group are shaded, will only perform its task if all windows in the examined group have valid entries. This means:

- \( \text{usiGrp} \) must be greater than 0
- \( rCnr1X \) must be greater than or equal to 0.0
- \( rCnr1Y \) must be greater than or equal to 0.0
- \( rWdwWdth \) must be greater than 0
- \( rWdwHght \) must be greater than 0

If one of these criteria is not met, it is interpreted as incorrect input, and the error output \( bErr \) is set at the function block output of \( \text{FB\_BA\_FcdElemEntry} \). Within the window element \( \text{ST\_BA\_FcdElem} [414] \), the plausibility bit \( bVld \) is set to FALSE.

If on the other hand all entries of a facade element are zero, it is regarded as a valid, deliberately omitted facade element:
In the case of a facade of 6x4 windows, the elements window (2.1), window (3.5) and window (4.4) would be empty elements here.

**VAR_INPUT**

- udiColumn: UDINT;
- udiRow: UDINT;
- bWrt: BOOL;
- bRd: BOOL;
- udiGrp: UDINT;
- rCnr1X: REAL;
- rCnr1Y: REAL;
- rWdwWdth: REAL;
- rWdwHght: REAL;

**udiColumn**: Column index of the selected component on the facade. This refers to the selection of a field element of the array stored in the IN-OUT variable `arrFcdElem`.

**udiRow**: ditto. row index. **udiRow and udiColumn must not be zero**! This is due to the field definition, which always starts with 1; see above.
**bRd:** A positive edge at this input causes the information of the selected element, `arrFcdElem[udiColumn,udiRow]`, to be read into the function block and assigned to the input variables `diGrp` to `rWdwHght`. The resulting output variables are `rCnr2X` to `rCnr4Y`. If data are already present on the inputs `diGrp` to `rWdwHght` at time of reading, then the data previously read are immediately overwritten with these data.

**bWrt:** A positive edge writes the entered and calculated values into the selected field element `arrFcdElem[udiColumn, udiRow]`.

**udiGrp:** Group membership. Internally limited to a minimum value of 0.

**rCnr1X:** X-coordinate of corner point 1 [m].

**rCnr1Y:** Y-coordinate of corner point 1 [m].

**rWdwWdth:** Window width [m].

**rWdwHght:** Window height [m].

**VAR_OUTPUT**

```plaintext
rCnr2X : REAL;
rCnr2Y : REAL;
rCnr3X : REAL;
rCnr3Y : REAL;
rCnr4X : REAL;
rCnr4Y : REAL;
bErr : BOOL;
sErrDesc : T_MAXSTRING;
```

**rCnr2X:** Calculated X-coordinate of corner point 2 of the window [m]. See "Note [291]" above.

**rCnr2Y:** Calculated Y-coordinate of corner point 2 of the window [m]. See "Note [291]" above.

**rCnr3X:** Calculated X-coordinate of corner point 3 of the window [m]. See "Note [291]" above.

**rCnr3Y:** Calculated Y-coordinate of corner point 3 of the window [m]. See "Note [291]" above.

**rCnr4X:** Calculated X-coordinate of corner point 4 of the window [m]. See "Note [291]" above.

**rCnr4Y:** Calculated Y-coordinate of corner point 4 of the window [m]. See "Note [291]" above.

**bErr:** Result verification for the entered values.

**sErrDesc:** Contains the error description.

### Error description

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Error: Index error! udiColumn and/or udiRow are outside the permitted limits, 1.. uiMaxColumnFcd and 1.. uiMaxColumnFcd, respectively. See list of facade elements.</td>
</tr>
<tr>
<td>02</td>
<td>Error: The group index is 0, but at the same time another entry of the facade element is not zero. Only if all entries of a facade element are zero is it considered to be a valid, deliberately omitted facade component, otherwise it is interpreted as an incorrect entry. NOTE: Group entries less than zero are internally limited to zero.</td>
</tr>
<tr>
<td>03</td>
<td>Error: The X-component of the first corner point (Corner1) is less than zero.</td>
</tr>
<tr>
<td>04</td>
<td>Error: The Y-component of the first corner point (Corner1) is less than zero.</td>
</tr>
<tr>
<td>05</td>
<td>Error: The window width is less than or equal to zero.</td>
</tr>
<tr>
<td>06</td>
<td>Error: The window height is less than or equal to zero.</td>
</tr>
</tbody>
</table>

**VAR_IN_OUT**

```plaintext
arrFcdElem : ARRAY[1..Param.uiMaxColumnFcd, 1..Param.uiMaxRowFcd] OF ST_BA_FcdElem;
```

**arrFcdElem:** List of facade elements (see List of facade elements [287]).
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

FB_BA_InRngAzm

This function block checks whether the current azimuth angle (horizontal position of the sun) lies within the limits entered. As can be seen in the overview [280], the function block provides an additionally evaluation as to whether the sun shading of a window group should be activated. Therefore the observations in the remainder of the text always apply to one window group.

A smooth facade is always irradiated by the sun at an azimuth angle of Facade orientation-90° to Facade orientation+90°.

If the facade has lateral projections, however, this range is limited. This limitation can be checked with the help of this function block. However, the position of the window group on the facade also plays a role. If it lies centrally, this gives rise to the following situation (the values are only examples):
The values change for a group at the edge:

The start of the range \( rSttRng \) may be greater than the end \( rEndRng \), in which case values beyond 0° are considered:

Sample

<table>
<thead>
<tr>
<th>( rAzm )</th>
<th>10.0°</th>
</tr>
</thead>
<tbody>
<tr>
<td>( rSttRng )</td>
<td>280.0°</td>
</tr>
<tr>
<td>( rEndRng )</td>
<td>20.0°</td>
</tr>
<tr>
<td>( bOut )</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

However, the range regarded may not be greater than 180° or equal to 0° – this would be unrealistic. Such entries result in an error on the output \( bErr \) – the test output \( bOut \) is then additionally set to FALSE.

**VAR_INPUT**

- \( rAzm \): Current azimuth angle.
- \( rSttRng \): Start of range [°].
rEndRng: End of range [°].

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bOut</td>
<td>BOOL</td>
<td>The facade element is in the sun if the output is TRUE.</td>
</tr>
<tr>
<td>bErr</td>
<td>BOOL</td>
<td>This output is switched to TRUE if the parameters entered are erroneous.</td>
</tr>
<tr>
<td>sErrDescr</td>
<td>T_MAXSTRING</td>
<td>Contains the error description.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Error description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01: Error: rSttRng or rEndRng less than 0° or greater than 360°.</td>
</tr>
<tr>
<td>02: Error: The difference between rSttRng and rEndRng is greater than 180°. This range is too large for analyzing the insolation on a facade.</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**FB_BA_InRngElv**

This function block checks whether the current angle of elevation (vertical position of the sun) lies within the limits entered. As can be seen in the overview, the function block provides an additionally evaluation as to whether the sun shading of a window group should be activated. Therefore the observations in the remainder of the text always apply to one window group.

A normal vertical facade is irradiated by the sun at an angle of elevation of 0° to maximally 90°.
If the facade has projections, however, this range is limited. This limitation can be checked with the help of this function block. However, the position of the window group on the facade also plays a role. If it lies in the lower range, this gives rise to the following situation (the values are only examples):
The values change for a group below the projection:
The lower observation limit, \( r_{LoLmt} \), may thereby not be greater than or equal to the upper limit, \( r_{HiLmt} \). Such entries result in an error on the output \( bErr \) – the test output \( bOut \) is then additionally set to FALSE.

**VAR_INPUT**

| \( rElv \)   | REAL |
| \( rLoLmt \) | REAL |
| \( rHiLmt \) | REAL |

\( rElv \): Current elevation angle [°].

\( rLoLmt \): Lower limit value [°].

\( rHiLmt \): Upper limit value [°].

**VAR_OUTPUT**

| \( bOut \)   | BOOL |
| \( bErr \)   | BOOL |
| \( sErrDescr \) | T_MAXSTRING |

\( bOut \): The facade element is in the sun

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

\( sErrDescr \): Contains the error description.

**Error description**

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Error: ( r_{HiLmt} ) less than or equal to ( r_{LoLmt} ).</td>
</tr>
<tr>
<td>02</td>
<td>Error: ( r_{LoLmt} ) is less than 0° or ( r_{HiLmt} ) is greater than 90°.</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**FB_BA_RdFcdElemLst**

With the help of this function block, data for facade elements (windows) can be imported from a pre-defined Excel table in csv format into the list of facade elements [p. 287]. In addition the imported data are checked for plausibility and errors are written to a log file.

The following example shows the Excel table with the entries of the window elements. All text fields are freely writable. The fields marked in green are important; each line in these fields identifies a data set.

The following rules are to be observed:

- A data set must always start with a '@'.
- The indices \( \text{IndexColumn} \) and \( \text{IndexRow} \) must lie within the defined limits, see List of facade elements [p. 287]. These indices directly describe the facade element in the list \( \text{arrFcdElem} \) to which the data from the set are saved.
- Window width and window height must be greater than zero
- The corner coordinates \( P1x \) and \( P1y \) must be greater than or equal to zero.
- Each window element must be assigned to a group 1..255.
- For system-related reasons the total size of the table may not exceed 65534 bytes.
- This must have been saved in Excel as file type "CSV (comma-separated values) (*.csv)".
It is not necessary to describe all window elements that would be possible by definition or declaration. Before the new list is read in, the function block deletes the entire old list in the program. All elements that are not described by entries in the Excel table then have pure zero entries and are thus marked as non-existent and also non-evaluable, since the function block for shading correction, `FB_BA_ShdCorr` [311], does not accept elements with the group entry '0'.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>Description</td>
<td>IndexColumn</td>
<td>IndexRow</td>
<td>Window-Width</td>
<td>Window-Height</td>
<td>P1x</td>
<td>P1y</td>
<td>Group</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>Description</td>
<td>@</td>
<td>1</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>1.5</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>Description</td>
<td>@</td>
<td>0</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>2.7</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>Description</td>
<td>@</td>
<td>3</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>4.4</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>Description</td>
<td>@</td>
<td>4</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>6.1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>Description</td>
<td>@</td>
<td>5</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>7.8</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>Description</td>
<td>@</td>
<td>6</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>9.5</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>Description</td>
<td>@</td>
<td>7</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>11.2</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>8</td>
<td>Description</td>
<td>@</td>
<td>8</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>12.9</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>9</td>
<td>Description</td>
<td>@</td>
<td>9</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>14.6</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>10</td>
<td>Description</td>
<td>@</td>
<td>10</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>16.3</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>11</td>
<td>Description</td>
<td>@</td>
<td>1</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>1.5</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>12</td>
<td>Description</td>
<td>@</td>
<td>0</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>2.7</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>13</td>
<td>Description</td>
<td>@</td>
<td>3</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>4.4</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>14</td>
<td>Description</td>
<td>@</td>
<td>4</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>6.1</td>
<td>4</td>
</tr>
<tr>
<td>18</td>
<td>15</td>
<td>Description</td>
<td>@</td>
<td>5</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>7.8</td>
<td>4</td>
</tr>
<tr>
<td>19</td>
<td>16</td>
<td>Description</td>
<td>@</td>
<td>6</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>9.5</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>17</td>
<td>Description</td>
<td>@</td>
<td>7</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>11.2</td>
<td>4</td>
</tr>
<tr>
<td>21</td>
<td>18</td>
<td>Description</td>
<td>@</td>
<td>8</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>12.9</td>
<td>4</td>
</tr>
<tr>
<td>22</td>
<td>19</td>
<td>Description</td>
<td>@</td>
<td>9</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>14.6</td>
<td>4</td>
</tr>
<tr>
<td>23</td>
<td>20</td>
<td>Description</td>
<td>@</td>
<td>10</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>16.3</td>
<td>4</td>
</tr>
<tr>
<td>24</td>
<td>21</td>
<td>Description</td>
<td>@</td>
<td>1</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>1.5</td>
<td>7</td>
</tr>
<tr>
<td>25</td>
<td>22</td>
<td>Description</td>
<td>@</td>
<td>0</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>2.7</td>
<td>7</td>
</tr>
<tr>
<td>26</td>
<td>23</td>
<td>Description</td>
<td>@</td>
<td>3</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>4.4</td>
<td>7</td>
</tr>
<tr>
<td>27</td>
<td>24</td>
<td>Description</td>
<td>@</td>
<td>4</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>6.1</td>
<td>7</td>
</tr>
<tr>
<td>28</td>
<td>25</td>
<td>Description</td>
<td>@</td>
<td>5</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>7.8</td>
<td>7</td>
</tr>
<tr>
<td>29</td>
<td>26</td>
<td>Description</td>
<td>@</td>
<td>6</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>9.5</td>
<td>7</td>
</tr>
<tr>
<td>30</td>
<td>27</td>
<td>Description</td>
<td>@</td>
<td>7</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>11.2</td>
<td>7</td>
</tr>
<tr>
<td>31</td>
<td>28</td>
<td>Description</td>
<td>@</td>
<td>8</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>12.9</td>
<td>7</td>
</tr>
<tr>
<td>32</td>
<td>29</td>
<td>Description</td>
<td>@</td>
<td>9</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>14.6</td>
<td>7</td>
</tr>
<tr>
<td>33</td>
<td>30</td>
<td>Description</td>
<td>@</td>
<td>10</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>16.3</td>
<td>7</td>
</tr>
<tr>
<td>34</td>
<td>31</td>
<td>Description</td>
<td>@</td>
<td>1</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>1.5</td>
<td>10</td>
</tr>
<tr>
<td>35</td>
<td>32</td>
<td>Description</td>
<td>@</td>
<td>0</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>2.7</td>
<td>10</td>
</tr>
<tr>
<td>36</td>
<td>33</td>
<td>Description</td>
<td>@</td>
<td>3</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>4.4</td>
<td>10</td>
</tr>
<tr>
<td>37</td>
<td>34</td>
<td>Description</td>
<td>@</td>
<td>4</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>6.1</td>
<td>10</td>
</tr>
<tr>
<td>38</td>
<td>35</td>
<td>Description</td>
<td>@</td>
<td>5</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>7.8</td>
<td>10</td>
</tr>
<tr>
<td>39</td>
<td>36</td>
<td>Description</td>
<td>@</td>
<td>6</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>9.5</td>
<td>10</td>
</tr>
<tr>
<td>40</td>
<td>37</td>
<td>Description</td>
<td>@</td>
<td>7</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>11.2</td>
<td>10</td>
</tr>
<tr>
<td>41</td>
<td>38</td>
<td>Description</td>
<td>@</td>
<td>8</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>12.9</td>
<td>10</td>
</tr>
<tr>
<td>42</td>
<td>39</td>
<td>Description</td>
<td>@</td>
<td>9</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>14.6</td>
<td>10</td>
</tr>
<tr>
<td>43</td>
<td>40</td>
<td>Description</td>
<td>@</td>
<td>10</td>
<td>1</td>
<td>1.2</td>
<td>1.3</td>
<td>16.3</td>
<td>10</td>
</tr>
</tbody>
</table>

Log file

Each time the reading function block is restarted, the log file is rewritten and the old contents are deleted. If there is no log file, it will be automatically created first. The log file then contains either an OK message or a list of all errors that have occurred. Errors connected with the opening, writing or closing of the log file itself
cannot be written at the same time. Therefore, always note the output \textit{sErrDescr} of the reading function block that indicates the last error code. Since the log file is always closed last during the reading process, a corresponding alarm is ensured in the event of an error.

**Program sample**

```plaintext
PROGRAM ReadFacadeElements
VAR
  bInit : BOOL;
  sRead : R_TRIG;
  fbReadFacadeElementList : FB_BA_RdFacadeElementList;
  arrFacadeElement : ARRAY (1..uiMaxColumnFcd, 1..uiMaxRowFcd) OF ST_BA_FcdElem;

  bbInit : BOOL;
  udiAmountOfSetsRead : UDINT;
  bError : BOOL;
  sErrorDescr : T_MaxString;
  bErrDataSet : BOOL;
END_VAR
```

In this sample the variable \textit{bInit} is initially set to TRUE when the PLC starts. Hence, the input \textit{bStt} of the function block \textit{fbReadFacadeElementList} receives a once-only rising edge that triggers the reading process. The file "FacadeElements.csv" is read, which is located in the folder "C:\Projects\". The log file "Logfile.txt" is then saved in the same folder. If this log file does not yet exist it will be created, otherwise the existing contents are overwritten. Reading and writing take place on the same computer on which the PLC is located. This is defined by the input \textit{sNetID} = " (=local). All data are written to the list \textit{arrFcdElem} declared in the program. During reading and writing the output \textit{bBusy} is set to TRUE. The last file handling error that occurred is displayed at \textit{sErrDescr}; \textit{bErr} is TRUE. If an error is detected in the data set, this is displayed at \textit{bErrDataSet} and described in more detail in the log file. The number of found and read data rows is displayed at \textit{udiAmtSetsRd} for verification purposes.

The errors marked were "built into" the following Excel list. This gives rise to the log file shown:
The first error is in data set 2 and is an index error, since "0" is not permitted. The next error in data set 6 was found after validation of the data with the internally used function block FB_BA_ShdoObjEntry [314] and allocated an error description. The third and the fourth errors likewise occurred after the internal validation.

Important here is that the data set numbers (in this case 22 and 24) do not go by the numbers entered in the list, but by the actual sequential numbers: only 30 data sets were read in here.

VAR_INPUT

bStt : BOOL;
sDataFile : STRING;
sLogFile : STRING;

VAR_INPUT

bStt: A TRUE edge on this input starts the reading process.

sDataFile: Contains the path and file name for the data file to be opened. This must have been saved in Excel as file type "CSV (comma-separated values) (*.csv)". If the file is opened with a simple text editor, the values must be separated by semicolons. Example of an entry: sDataFile:=
'C:\Projekte\FacadeElements.csv'

sLogFile: ditto. Log file for the accumulating errors. This file is overwritten each time the function block is activated, so that only current errors are contained.
sNetId: A string can be entered here with the AMS Net ID of the TwinCAT computer on which the files are to be written/read. An empty string can be specified for the local computer (see T_AmsNetId).

The data can be saved only on the control computer itself and on the computers that are connected by ADS to the control computer. Links to local hard disks in this computer are possible, but not to connected network hard drives.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
</tr>
<tr>
<td>udiAmtSetsRd</td>
<td>UDINT</td>
</tr>
<tr>
<td>bErr</td>
<td>BOOL</td>
</tr>
<tr>
<td>sErrDescr</td>
<td>T_MAXSTRING</td>
</tr>
<tr>
<td>bErrDataSet</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

bBusy: This output is TRUE as long as elements are being read from the file.

udiAmtSetsRd: Number of data sets read.

bErr: This output is switched to TRUE, if a file write or read error has occurred.

sErrDescr: Contains the error description.

**Error description**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>File handling error: Opening the log file - the ADS error number is stated.</td>
</tr>
<tr>
<td>02</td>
<td>File handling error: Open the data file - the ADS error number is stated.</td>
</tr>
<tr>
<td>03</td>
<td>File handling error: Reading the data file - the ADS error number is stated.</td>
</tr>
<tr>
<td>04</td>
<td>Error: During reading of the data file it was determined that the file is too large (number of bytes larger than udiMaxDataFileSize)</td>
</tr>
<tr>
<td>05</td>
<td>File handling error: Writing to the log file - the ADS error number is stated.</td>
</tr>
<tr>
<td>06</td>
<td>File handling error: Closing the data file - the ADS error number is stated.</td>
</tr>
<tr>
<td>07</td>
<td>File handling error: Writing to the log file (OK message if no errors were detected) - the ADS error number is stated.</td>
</tr>
<tr>
<td>08</td>
<td>File handling error: Closing the log file - the ADS error number is stated.</td>
</tr>
</tbody>
</table>

bErrDataSet: This output is set to TRUE, if the read data sets are faulty. Further details are entered in the log file.

**VAR_IN_OUT**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrFcdElem</td>
<td>ARRAY[1..Param.uiMaxColumnFcd, 1..Param.uiMaxRowFcd] OF ST_BA_FcdElem;</td>
</tr>
</tbody>
</table>

arrFcdElem: List of facade elements (287).

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**FB_BA_RdShdObjLst**

```
FB_BA_RdShdObjLst
```

- bStt: BOOL
- sDataFile: STRING
- sLogFile: STRING
- sNetId: T_AmsNetId
- arrShdObj: ARRAY [1..Param.uiMaxShdObj] OF ST_BA_ShdObj
With the help of this function block, data for shading objects can be imported from a pre-defined Excel table in csv format into the list of shading objects \([287]\). In addition the imported data are checked for plausibility and errors are written to a log file.

The following example shows the Excel table with the entries of the window elements. All text fields are freely writable. The fields marked in green are important; each line in these fields identifies a data set. The columns G to J have a different meaning depending on whether the type rectangle or ball is concerned. The columns K to M are to be left empty in the case of balls. With regard to the rectangle coordinates, only the relevant data are entered and the remainder are internally calculated, see \(\text{FB\_BA\_ShdObjEntry}\ [314]\).

The following rules are to be observed:

- A data set must always start with a ‘@’.
- The monthly entries must not be 0 or greater than 12; all other combinations are possible.

**Examples:**
- Start=1, End=1: shading in January.
- Start=1, End=5: shading from the beginning of January to the end of May.
- Start=11, End=5: shading from the beginning of November to the end of May (the following year).

- Window width and window height must be greater than zero
- The z-coordinates P1z and P3z or Mz must be greater than zero.
- The radius must be greater than zero.
- For system-related reasons the total size of the table may not exceed 65534 bytes.
- This must have been saved in Excel as file type "CSV (comma-separated values) (*.csv)".

Is not necessary to describe all shading objects that are possible per facade. Only those contained in the list ultimately take effect.

### Log file

Each time the reading function block is restarted, the log file is rewritten and the old contents are deleted. If there is no log file, it will be automatically created first. The log file then contains either an OK message or a list of all errors that have occurred. Errors connected with the opening, writing or closing of the log file itself
cannot be written at the same time. Therefore, always note the output \textit{sErrDescr} of the reading function block that indicates the last error code. Since the log file is always closed last during the reading process, a corresponding alarm is ensured in the event of an error.

\textbf{Program sample}

\begin{verbatim}
PROGRAM ReadShadingObjects
VAR
  bInit : BOOL;
  rRead : R_TRIG;
  fbReadShadingObjects : FB_BA_RdShdObjList;
  arrShadingObject : ARRAY [1..ulMaxShdObj] OF ST_BA_ShObj;

  bBusy : BOOL;
  ulAmountOfSetsRead : UDI_INT;
  bError : BOOL;
  sErrorDescr : T_MaxString;
  bErrDataSet : BOOL;
END_VAR

rRead

\end{verbatim}

In this sample the variable \textit{bInit} is initially set to TRUE when the PLC starts. Hence, the input \textit{bStt} on the function block \textit{fbReadShadingObjects} receives a once-only rising edge that triggers the reading process. The file "ShadingObjects.csv" is read, which is located in the folder "C:\Projects\". The log file "Logfile.txt" is then saved in the same folder. If this log file does not yet exist it will be created, otherwise the existing contents are overwritten. Reading and writing take place on the same computer on which the PLC is located. This is defined by the input \textit{sNetID} = " (=local). All data are written to the list \textit{arrShdObj} declared in the program. During reading and writing the output \textit{bBusy} is set to TRUE. The last file handling error that occurred is displayed at \textit{sErrDescr}; \textit{bErr} is TRUE. If an error is detected in the data set, this is displayed at \textit{bErrDataSet} and described in more detail in the log file. The number of found and read data rows is displayed at \textit{ulAmountOfSetsRead} for verification purposes.

The errors marked were built into the following Excel list. This gives rise to the log file shown:
The first error is in data set 3 and is a type error, since "2" is not defined.
The next error in data set 6 was found after validation of the data with the internally used function block FB_BA_ShdObjEntry [314] and allocated an error description. The third error likewise occurred after the internal validation.

Important here it that the data set number (in this case 11) does not go by the number entered in the list, but by the actual sequential number: only 16 data sets were read in here.

**VAR_INPUT**

- **bStt**: A TRUE edge on this input starts the reading process.
- **sDataFile**: Contains the path and file name for the data file to be opened. This must have been saved in Excel as file type "CSV (comma-separated values) (*.csv)". If the file is opened with a simple text editor, the values must be separated by semicolons. Example of an entry: `sDataFile := 'C:\Projects\ShadingObjects.csv'
- **sLogFile**: ditto. Log file for the accumulating errors. This file is overwritten each time the function block is activated, so that only current errors are contained.
- **sNetId**: A string can be entered here with the AMS Net ID of the TwinCAT computer on which the files are to be written/read. An empty string can be specified for the local computer (see T_AmsNetId).

The data can be saved only on the control computer itself and on the computers that are connected by ADS to the control computer. Links to local hard disks in this computer are possible, but not to connected network hard drives.

**VAR_OUTPUT**

- **bBusy**: BOOL;
- **udiAmtSetsRd**: UDINT;
- **bErr**: BOOL;
- **sErrDescr**: T_MAXSTRING;
- **bErrDataSet**: BOOL;
bBusy: This output is TRUE as long as elements are being read from the file.

udiAmtSetsRd: Number of data sets read.

bErr: This output is switched to TRUE, if a file write or read error has occurred.

sErrMsg: Contains the error description.

### Error description

<table>
<thead>
<tr>
<th>Error description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>File handling error: Opening the log file - the ADS error number is stated.</td>
</tr>
<tr>
<td>02</td>
<td>File handling error: Open the data file - the ADS error number is stated.</td>
</tr>
<tr>
<td>03</td>
<td>File handling error: Reading the data file - the ADS error number is stated.</td>
</tr>
<tr>
<td>04</td>
<td>Error: During reading of the data file it was determined that the file is too large (number of bytes larger than udiMaxDataFileSize)</td>
</tr>
<tr>
<td>05</td>
<td>File handling error: Writing to the log file - the ADS error number is stated.</td>
</tr>
<tr>
<td>06</td>
<td>File handling error: Closing the data file - the ADS error number is stated.</td>
</tr>
<tr>
<td>07</td>
<td>File handling error: Writing to the log file (OK message if no errors were detected) - the ADS error number is stated.</td>
</tr>
<tr>
<td>08</td>
<td>File handling error: Closing the log file - the ADS error number is stated.</td>
</tr>
</tbody>
</table>

bErrDataSet: This output is set to TRUE, if the read data sets are faulty. Further details are entered in the log file.

### VAR_IN_OUT

arrShdObj: ARRAY[1..Param.uiMaxShdObj] OF ST_BA_ShdObj;

arrShdObj: List of shading objects [287].

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

### FB_BA_RolBldActr

This function block is used to position a roller shutter over two outputs: up and down. The positioning telegram stSunBld [416] can be used to move the roller shutter to any position. In addition, the positioning telegram stSunBld [416] offers manual commands, which can be used to move the roller shutter to particular positions. These manual commands are controlled by the function block FB_BA_SunBldSwi [330].

Structure of the blind positioning telegram stSunBld [416].

```plaintext
TYPE ST_BA_SunBld:
STRUCT
  rPos      : REAL;
  rAngl     : REAL;
  bManUp    : BOOL;
  bManDwn   : BOOL;
END_STRUCT
```

TF8040 Version: 1.4 309
The current height position is not read in by an additional encoder; it is determined internally by the runtime of the roller shutter.

The two runtime parameters \( \text{udiTiUp} \) (roller shutter travel-up time [ms]) and \( \text{udiTiDwn} \) (roller shutter travel-down time [ms]) take account of the different movement characteristics.

As a rule, the function block controls the roller shutter based on the information from the positioning telegram \( \text{stSunBld} \), If automatic mode is active \((b\text{ManMod}=\text{FALSE})\), the roller shutter always moves to the current position; changes are reflected immediately. In manual mode \((b\text{ManMod}=\text{TRUE})\), the roller shutter is controlled by the commands \( \text{bManUp} \) and \( \text{bManDwn} \).

### Referencing

Safe referencing refers to a situation when the roller shutter is upwards-controlled for longer than its complete travel-up time. The position is then always "0". Since roller shutter positioning without encoder is always error-prone, it is important to use automatic referencing whenever possible: Whenever "0" is specified as the target position, the roller shutter initially moves upwards normally, based on continuous position calculation. Once the calculated position value 0% is reached, the output \( \text{bUp} \) continues to be held for the complete travel-up time + 5s.

For reasons of flexibility there are now two possibilities to interrupt the referencing procedure: Until the calculated 0% position is reached, a change in position continues to be assumed and executed. Once this 0% position is reached, the roller shutter can still be moved with the manual "travel-down" command. These two sensible restrictions make it necessary for the user to ensure that the roller shutter is referenced safely whenever possible.

After a system restart, the function block executes a reference run. Completion of the initial referencing is indicated through a TRUE signal at output \( \text{bInitRefCmpl} \). The initial referencing can also be terminated through a manual "travel-down" command.

### VAR_INPUT

- \( \text{bEn} \): Enable input for the function block. As long as this input is TRUE, the actuator function block accepts and executes commands as described above. A FALSE signal on this input resets the control outputs \( \text{bUp} \) and \( \text{bDwn} \) and the function block remains in a state of rest.
- \( \text{stSunBld} \): Positioning telegram, see \( \text{ST\_BA\_SunBld} \).
- \( \text{udiTiUp} \): Complete time for driving up [ms].
- \( \text{udiTiDwn} \): Complete time for driving down in ms.

### VAR_OUTPUT

- \( \text{bUp} \): Roller shutter control output up.
- \( \text{bDwn} \): Roller shutter control output down.
- \( \text{rActlPos} \): Current position in percent.
bRef: The roller shutter is in referencing mode, i.e. the output $b_{Up}$ is set for the complete travel-up time + 5s. Only a manual "down" command can move the roller shutter in the opposite direction and terminate this mode.

udiRefTi_sec: Referencing countdown display [s].

bInitRefCompl: Initial referencing process complete.

bBusy: A positioning or a referencing procedure is in progress.

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

sErrDescr: Contains the error description.

**Error description**

01: Error: The total travel-up or travel-down time (udiTiUp_ms/udiTiDwn_ms) is zero.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**FB_BA_ShdCorr**

The function block is used to assess the shading of a group of windows on a facade.

The function block FB_BA_ShdCorr calculates whether a window group lies in the shadow of surrounding objects. The result, which is output at the output $b_{GrpNotShdd}$, can be used to judge whether sun shading makes sense for this window group.

The function block thereby accesses two lists, which are to be defined:

- The parameters that describe the shading elements that are relevant to the facade on which the window group is located. This list of shading objects [287] is used as input variable $arrShdObj$ for the function block, since the information is read only.

- The data of the elements (window) of the facade in which the group to be regarded is located. This list of facade elements [287] is accessed via the IN/OUT variable $arrFcdElem$, since not only the window coordinates are read, but the function block FB_BA_ShdCorr also stores the shading information for each window corner in this list. In this way, the information can also be used in other parts of the application program.

On the basis of the facade orientation ($rFcdOrtn$), the direction of the sun ($rAzm$) and the height of the sun ($rElv$), a calculation can be performed for each corner of a window to check whether this lies in a shaded area. A window group is considered to be completely shaded if all corners are shaded.

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

<table>
<thead>
<tr>
<th>Line of sight</th>
<th>Facade orientation</th>
</tr>
</thead>
<tbody>
<tr>
<td>North</td>
<td>$\beta=0^\circ$</td>
</tr>
<tr>
<td>East</td>
<td>$\beta=90^\circ$</td>
</tr>
<tr>
<td>South</td>
<td>$\beta=180^\circ$</td>
</tr>
<tr>
<td>West</td>
<td>$\beta=270^\circ$</td>
</tr>
</tbody>
</table>
The function block performs its calculations only if the sun is actually shining on the facade. Considering the drawing presented in the introduction, this is the case if:

Facade orientation < azimuth angle < facade orientation + 180°

In addition, a calculation is also not required, if the sun has not yet risen, i.e. the solar elevation is below 0°. In both cases the output $bFcdSunlit$ is set to FALSE.

The situation is different for the southern hemisphere. The following applies to the facade orientation (looking out the window):

<table>
<thead>
<tr>
<th>Line of sight</th>
<th>Facade orientation</th>
</tr>
</thead>
<tbody>
<tr>
<td>South</td>
<td>$\beta=0^\circ$</td>
</tr>
<tr>
<td>East</td>
<td>$\beta=90^\circ$</td>
</tr>
<tr>
<td>North</td>
<td>$\beta=180^\circ$</td>
</tr>
<tr>
<td>West</td>
<td>$\beta=270^\circ$</td>
</tr>
</tbody>
</table>

The internal calculation or the relationship between facade and sunbeam also changes:
To distinguish between the situation in the northern and southern hemisphere, set the input parameter 
$bSouth$ to FALSE (northern hemisphere) or TRUE (southern hemisphere)

**VAR_INPUT**

- `stTiActl : TIMESTRUCT;`
- `rFcdOrtn : REAL;`
- `rAzm : REAL;`
- `rElv : REAL;`
- `diGrpID : DINT;`
- `bSouth : BOOL;`
- `arrShdObj : ARRAY[1..Param.uiMaxShdObj] OF ST_BA_ShdObj;`

`stTiActl`: Input of the current time - local time in this case, since this time takes into account the shaded months. If the UTC time (or GMT) is used, the month may change in the middle of the day, depending on the location on the earth (see TIMESTRUCT).

`rFcdOrtn`: Facade orientation, see illustration above.

`rAzm`: Direction of the sun at the time of observation ['°].

`rElv`: Solar altitude at the time of observation ['°].

`diGrpID`: Window group regarded. The group 0 is reserved here for unused window elements, see FB_BA_FcdElemEntry [urence]. A 0-entry would lead to an error output (bErr=TRUE). The function block is then not executed any further and `bGrpNotShdd` is set to FALSE.

`bSouth`: FALSE: Calculations refer to conditions in the northern hemisphere - TRUE: In the southern hemisphere

`arrShdObj`: List of shading objects [urence].

**VAR_OUTPUT**

- `bGrpNotShdd : BOOL;`
- `bFcdSunlit : BOOL;`
- `bErr : BOOL;`
- `sErrDescr : T_MAXSTRING`

`bGrpNotShdd`: Is TRUE as long as the window group is not calculated as shaded.

`bFcdSunlit`: This output is set to TRUE if the sun is shining on the facade. See description above.
bErr: This output is switched to TRUE if the parameters entered are erroneous.

sErrDescr: Contains the error description.

<table>
<thead>
<tr>
<th>Error description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01: Error: The index of the window group usiGrpId under consideration is 0.</td>
</tr>
<tr>
<td>02: Error: An element of the facade list is invalid. This is specified in the error description sErrDescr as arrFcdElem[nColumn,nRow].</td>
</tr>
</tbody>
</table>

VAR_IN_OUT

arrFcdElem : ARRAY[1..Param.uiMaxColumnFcd, 1..Param.uiMaxRowFcd] OF ST_BA_FcdElem;

arrFcdElem: List of facade elements [> 287].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

FB_BA_ShdojObjEntry

This function block serves for the administration of all shading elements in a facade, which is globally saved in a list of shading elements [> 287]. It is intended to facilitate the input of the element information - also with regard to the use of a visualization. A schematic illustration of the objects with description of the coordinates is given in Shading correction: principles and definitions [> 272].

The shading elements are declared in the global variables:

VAR_GLOBAL

arrShdObj : ARRAY[1..Param.uiMaxShdObj] OF ST_BA_ShdojObj;

END_VAR

Each individual element arrShdObj[1] to arrShdObj [uiMaxShdObj] carries the information for an individual shading element (ST_BA_ShdojObj [> 415]). This information consists of the selected type of shading (rectangle or ball) and the respectively associated coordinates. For a rectangle, these are the corner points (rP1x, rP1y, rP1z), (rP2x, rP2y, rP2z), (rP3x, rP3y, rP3z) and (rP4x, rP4y, rP4z), for a sphere these are the center point (rMx, rMy, rMz) and the radius rRads. In addition, the phase of the shading can be defined via the inputs udiBegMth and udiEndMth, which is important in the case of objects such as trees that bear no foliage in winter.
The function block directly accesses the array of this information via the IN-OUT variable arrShdObj.

**Note:** The fact that the rectangle coordinates \( rP2x, rP2z, rP4x, rP4y \) and \( rP4z \) are output values results from the fact that they are formed from the input parameters:

\[
\begin{align*}
  rP2x &= rP1x; \\
  rP2z &= rP1z; \\
  rP4x &= rP3x; \\
  rP4y &= rP1y; \\
  rP4z &= rP3z;
\end{align*}
\]

That limits the input of a square to the extent that the lateral edges stand vertically on the floor \( (rP2x = rP1x \) and \( rP4x = rP3x) \), that the square has no inclination \( (rP2z = rP1z \) and \( rP4z = rP3z) \) and can only have a different height "upwards", i.e. in the positive y-direction \( (rP4y = rP1y) \).

The function block is used in three steps:

- **Read**
- **Change**
- **Write**

**Read**

Selection of the element from the list \( arrShdObj[iId] \) is based on the entry at \( udlid \). A rising edge on \( bRd \) reads the data. These values are assigned to the input and output variables of the function block. These are the input values \( rP1x, rP1y, rP1z, rP2y, rP3x, rP3y, rP3z, rMx, rMy, rMz, rRads \) and the object enumerator \( eType \) and the output values \( rP2x, rP2z, rP4x, rP4y \) and \( rP4z \). It is important here that the input values are not overwritten in the reading step. Hence, all values can initially be displayed in a visualization.

**Change**

In a next program step the listed input values can then be changed. If a rectangle is preselected at input \( eType \) via the value "eObjectTypeTetragon", the output values \( rP2x, rP2z, rP4x, rP4y \) and \( rP4z \) result from the rectangle coordinates that were entered (see above).

The values entered are constantly checked for plausibility. The output \( bErr \) indicates whether the values are valid \( (bErr=FALSE) \). If the value is invalid, a corresponding error message is issued at output \( sErrDescr \). If a rectangle is defined, only the inputs \( rP1x, rP1y, rP1z, rP2y, rP3x, rP3y, rP3z \) have to be described; the inputs \( rMx, rMy, rMz \) and \( rRads \) do not have to be linked. For a sphere definition, only \( rMx, rMy, rMz \) and \( rRads \) have to be described; the rectangle coordinates can remain unlinked.

**Write**

The parameterized data are written to the list element with the index \( udlid \) upon a rising edge on \( bWrt \), regardless of whether they represent valid values or not. The element structure \( ST_BA_ShdoObj | 415 \) therefore contains a plausibility bit \( bVld \), which forwards precisely this information to the function block \( FB_BA_ShdoCorr | 311 \) to prevent miscalculations.

This approach is to be regarded only as a proposal. It is naturally also possible to parameterize the function block quite normally in one step and to write the values entered to the corresponding list element with a rising edge on \( bWrt \).

**VAR_INPUT**

\[
\begin{align*}
  udlid & : UDINT; \\
  bRd & : BOOL; \\
  bWrt & : BOOL; \\
  rP1x & : REAL; \\
  rP1y & : REAL; \\
  rP1z & : REAL; \\
  rP2y & : REAL; \\
  rP3x & : REAL; \\
  rP3y & : REAL; \\
  rP3z & : REAL; \\
  rMx & : REAL; \\
  rMy & : REAL; \\
  rMz & : REAL; \\
  rRads & : REAL; \\
  udlBegMth & : UDINT; \\
  udlEndMth & : UDINT; \\
  eType & : E_BA_ShdoObjType;
\end{align*}
\]
udiId: Index of the selected element. This refers to the selection of a field element of the array saved in the IN-OUT variable arrShdObj. The variable udiId must not be zero! This is due to the field definition, which starts with 1. However, an incorrect input is recognized and displayed as such at bErr/sErrDescr.

bRd: The information of the selected element, arrShdObj[udiId], is read into the function block with a positive edge at this input and assigned to the input variables rP1x to eType and the output variables rP2x to rP4z. If data are already present on the inputs rP1x to eType at this time, then the data previously read are immediately overwritten with these data.

bWrt: A positive edge writes the values applied to the inputs rP1x to eType and the values determined and assigned to the outputs rP2x to rP4z to the selected field element arrShdObj[udiId].

rP1x: X-coordinate of point 1 of the shading element (rectangle) [m].

rP1y: Y-coordinate of point 1 of the shading element (rectangle) [m].

rP1z: Z-coordinate of point 1 of the shading element (rectangle) [m].

rP2y: Y-coordinate of point 2 of the shading element (rectangle) [m].

rP3x: X-coordinate of point 3 of the shading element (rectangle) [m].

rP3y: Y-coordinate of point 3 of the shading element (rectangle) [m].

rP3z: Z-coordinate of point 3 of the shading element (rectangle) [m].

rMx: X-coordinate of the center of the shading element (ball) [m].

rMy: Y-coordinate of the center of the shading element (ball) [m].

rMz: Z-coordinate of the center of the shading element (ball) [m].

rRads: Radius of the shading element (ball) [m].

udiBegMth: Beginning of the shading period (month).

udiEndMth: End of the shading period (month).

eType: Selected type of element: Rectangle or sphere (see E_BA_ShdObjType [412]).

Remark about the shading period:

The entries for the months may not be 0 or greater than 12, otherwise all combinations are possible.

Examples:
Start=1, End=1: shading in January.
Start=1, End=5: shading from the beginning of January to the end of May.
Start=11, End=5: shading from the beginning of November to the end of May (the following year).

VAR_OUTPUT
rP2x : REAL;
rP2z : REAL;
rP4x : REAL;
rP4y : REAL;
rP4z : REAL;
bErr : BOOL;
sErrDescr : T_MAXSTRING;

rP2x: Calculated X-coordinate of point 2 of the shading element (rectangle) [m]. See "Note [314]" above.

rP2z: Calculated Z-coordinate of point 2 of the shading element (rectangle) [m]. See "Note [314]" above.

rP4x: Calculated X-coordinate of point 4 of the shading element (rectangle) [m]. See "Note [314]" above.

rP4y: Calculated Y-coordinate of point 4 of the shading element (rectangle) [m]. See "Note [314]" above.

rP4z: Calculated Z-coordinate of point 4 of the shading element (rectangle) [m]. See "Note [314]" above.
bErr: Result of the plausibility check for the values entered. For a rectangle, the internal angle is 360° and the points are in a plane in front of the facade under consideration. In the case of a ball the center must likewise lie in front of the facade and the radius must be greater than zero.

sErrDescr: Contains the error description.

<table>
<thead>
<tr>
<th>Error description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01: Error: The input uIdld is outside the permissible limits 1..uiMaxShdObj.</td>
</tr>
<tr>
<td>02: Error: The sum of the angles of the rectangle is not 360°. This means that the corners are not in the order P1, P2, P3 and P4 but rather P1, P3, P2 and P4. This results in a crossed-over rectangle.</td>
</tr>
<tr>
<td>03: Error: The corners of the square are not on the same level.</td>
</tr>
<tr>
<td>04: Error: The z-component of P1 is less than zero. This corner would thus lie behind the facade.</td>
</tr>
<tr>
<td>05: Error: The z-component of P3 is less than zero. This corner would thus lie behind the facade.</td>
</tr>
<tr>
<td>06: Error: P1 is equal to P2. The object entered is thus not a rectangle.</td>
</tr>
<tr>
<td>07: Error: P1 is equal to P3. The object entered is thus not a rectangle.</td>
</tr>
<tr>
<td>08: Error: P1 is equal to P4. The object entered is thus not a rectangle.</td>
</tr>
<tr>
<td>09: Error: P2 is equal to P3. The object entered is thus not a rectangle.</td>
</tr>
<tr>
<td>10: Error: P2 is equal to P4. The object entered is thus not a rectangle.</td>
</tr>
<tr>
<td>11: Error: P3 is equal to P4. The object entered is thus not a rectangle.</td>
</tr>
<tr>
<td>12: Error: The radius entered is zero.</td>
</tr>
<tr>
<td>13: Error: The z-component of the ball center is less than zero. This point would thus lie behind the facade.</td>
</tr>
<tr>
<td>14: Error: Error object type eType - neither rectangle nor ball.</td>
</tr>
<tr>
<td>15: Error: Month input error.</td>
</tr>
</tbody>
</table>

VAR_IN_OUT

arrShdObj : ARRAY[1..Param.uiMaxShdObj] OF ST_BA_ShdObj;

arrShdObj: List of shading objects [287].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

FB_BA_SunBldActr

This function block is used for positioning of a louvered blind via two outputs: drive up and drive down. The blind can be driven to any desired (height) position and louver angle via the positioning telegram stSunBld [416]. On top of that, the positioning telegram stSunBld [416] also contains manual commands with which the blind can be moved individually to certain positions. These manual commands are controlled by the function block FB_BA_SunBldSwi [330].
Structure of the blind positioning telegram stSunBld [416].

```plaintext
TYPE ST_BA_SunBld:
  STRUCT
    rPos : REAL;
    rAngl : REAL;
    bManUp : BOOL;
    bManDwn : BOOL;
    bManMod : BOOL;
    bActv : BOOL;
  END_STRUCT
END_TYPE
```

The current height position and the louvre angle are not read in by an additional encoder, but determined internally by the travel time of the blind. The calculation is based on the following travel profile (regarded from the highest and lowest position of the blind):

**Downward travel profile:**

<table>
<thead>
<tr>
<th>1) The blind is in the uppermost position</th>
<th>2) The backlash was moved out. The blind was driven down a little bit without turning the lamellas.</th>
<th>3) The lamellas are turned to the lowest angle.</th>
<th>4) The blind is completely driven down.</th>
</tr>
</thead>
</table>

More detailed explanations of the terms “backlash” and “turning” are given here in the downward movement:

The blind normally describes its downward movement with the louvre low point directed outwards, as in fig. 3).

If the blind is in an initial position with the low point directed inwards (i.e. after the conclusion of an upward movement), then a certain time elapses after a new downward movement begins before the louvres start to turn from the "inward low point" to the "outward low point". During this time the louvre angle does not change; the blind only drives downward (fig.1 and fig. 2). This time is an important parameter for the movement calculation and is entered in the function block under *udiBckLshTiDwn_ms* [ms]. Since it is not known at any point after a blind movement of any length whether backlash has already taken effect, the backlash of the downward movement or its travel time can be measured most reliably if the blind was first raised fully. A further important parameter is the time interval of the subsequent turning of the louvres from the "Outward low point" to the "Inward low point". This time should be entered as *udiTurnTiDwn_ms* [ms] at the function block.
Upward travel profile:

1) The blind is in the lowermost position

2) The backlash was moved out. The blind was driven up a little bit without turning the lamellas.

3) The lamellas are turned to the highest angle.

4) The blind is completely driven up.

More detailed explanations of the terms “backlash” and “turning” are given here in the upward movement:

The circumstances are similar to the downward movement described above: The blind normally describes its upward movement with the louvre low point directed inwards, as in fig. 3).

If the blind is in an initial position with the low point directed outwards (i.e. after the conclusion of a downward movement), then a certain time elapses after a new upward movement begins before the louvres start to turn from the "Outward low point" to the "Inward low point". During this time the louvre angle does not change; the blind only drives upward (fig. 1 and fig. 2). Also this time is an important parameter for the movement calculation and is entered in the function block under $udiBckLshTiUp ms [ms]$. Since it is not known at any point after a blind movement of any length whether backlash has already taken effect, the backlash of the upward movement or its travel time can be measured most reliably if the blind was first driven fully downward. A further important parameter is the time interval of the subsequent turning of the louvres from the "Outward low point" to the "Inward low point". This time should be entered as $udiTurnTiUp ms [ms]$ at the function block.

Parameterization

For the calculation of the (height) position and the louvre angle, the following times now have to be determined for both the upward and downward movement:

- the travel time of the backlash ($udiBckLshTiUp ms / udiBckLshTiDwn ms [ms]$)
- the turning duration ($udiTurnTiUp ms / udiTurnTiDwn ms [ms]$)
- the total travel time ($udiTiUp ms / udiTiDwn ms [ms]$)

Furthermore the following are required for the calculation:

- the highest louvre angle after turning upwards ($rAngLmtUp [°]$)
- the lowest louvre angle after turning downwards ($rAngLmtDwn [°]$)

The louvre angle $\lambda$ is defined by a notional straight line through the end points of the louvre to the horizontal.
Functioning

As a rule, the function block controls the blind based on the information from the positioning telegram stSunBld. If automatic mode is active (bManMod=FALSE), then the current position and louvre angle are always driven to, wherein changes are immediately accounted for. The height positioning takes priority: First the entered height and afterwards the louvre angle are driven to. For reasons of the simplicity the position error due to the angle movement is disregarded. In manual mode (bManMod=TRUE), the blind is controlled by the commands bManUp and bManDwn.

An automatic movement command is triggered whenever a change from manual to automatic mode occurs.

Referencing

Secure referencing is ensured if the blind is driven upward for longer than its complete drive-up time. The position is then in any case "0" and the louvre angle is at its maximum. Since blind positioning without an encoder is naturally always susceptible to error, it is important to automatically reference as often as possible: each time the "0" position is to be driven to (the angle is unimportant), the blind initially drives upward quite normally with continuous position calculation. Once the calculated position value 0% is reached, the output bUp continues to be held for the complete travel-up time + 5 s.

For reasons of flexibility, there are two ways to interrupt the referencing process: Until the calculated 0% position is reached, a change in position continues to be assumed and executed. Once this 0% position is reached, the blind can still be moved with the manual "travel-down" command. These two sensible limitations make it necessary for the user to ensure that the blind is securely referenced as often as possible.

After a system restart, the function block executes a reference run. Completion of the initial referencing is indicated through a TRUE signal at output bInitRefCmpl. The initial referencing can also be terminated through a manual "travel-down" command.

Target accuracy

Since the function block determines the blind position solely via run times, the cycle time of the PLC task plays a crucial role for positioning accuracy. If the switching time for a louvre angle range of -70° to 10° is 1 second, for example, the accuracy at a cycle time of 50 ms is +/-4°.

VAR_INPUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEn</td>
<td>BOOL</td>
</tr>
<tr>
<td>stSunBld</td>
<td>ST_BA_SunBld;</td>
</tr>
<tr>
<td>udiTiUp_ms</td>
<td>UDINT;</td>
</tr>
<tr>
<td>udiTiDwn_ms</td>
<td>UDINT;</td>
</tr>
<tr>
<td>udiTurnTiUp_ms</td>
<td>UDINT;</td>
</tr>
<tr>
<td>udiTurnTiDwn_ms</td>
<td>UDINT;</td>
</tr>
<tr>
<td>udiBckLshTiUp_ms</td>
<td>UDINT;</td>
</tr>
<tr>
<td>udiBckLshTiDwn_ms</td>
<td>UDINT;</td>
</tr>
<tr>
<td>rAngLmtUp</td>
<td>REAL;</td>
</tr>
<tr>
<td>rAngLmtDwn</td>
<td>REAL;</td>
</tr>
</tbody>
</table>

bEn: Enable input for the function block. As long as this input is TRUE, the actuator function block accepts and executes commands as described above. A FALSE signal on this input resets the control outputs bUp and bDwn and the function block remains in a state of rest.

stSunBld: Positioning telegram, (see ST_BA_SunBld).
udiTiUp_ms: Complete time for driving up [ms].

udiTiDwn_ms: Complete time for driving down [ms].

udiTurnTiUp_ms: Time for turning the louvres in the upward direction [ms].

udiTurnTiDwn_ms: Time for turning the louvres in the downward direction [ms].

udiBckLshTiUp_ms: Time to traverse the backlash in the upward direction [ms]. This input is internally limited to a minimum value of 0.

udiBckLshTiDwn_ms: Time to traverse the backlash in the downward direction [ms]. This input is internally limited to a minimum value of 0.

rAnglLmtUp: Highest position of the louvres [°].

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

The louvre angle $\lambda$, as defined above, is then typically greater than zero.

<table>
<thead>
<tr>
<th>Inside</th>
<th>Outside</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Diagram" /></td>
<td></td>
</tr>
</tbody>
</table>

rAnglLmtDwn: Lowest position of the louvres [°].

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

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

<table>
<thead>
<tr>
<th>Inside</th>
<th>Outside</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image2.png" alt="Diagram" /></td>
<td></td>
</tr>
</tbody>
</table>

VAR_OUTPUT

bUp : BOOL;
bDwn : BOOL;
rActlPos : REAL;
rActlAngl : REAL;
bRef : BOOL;
udiRefTi_sec : UDINT;
bInitRefCompl : BOOL;
bBusy : BOOL;
bErr : BOOL;
sErrDesc : T_MAXSTRING;

bUp: Control output for blind up.

bDwn: Control output for blind down.

rActlPos: Current position in percent.

rActlAngl: Current louvre angle [''].

bRef: The blind is referencing, i.e. the output bUp is set for the complete travel-up time + 5s. Only a manual "down" command can move the blind in the opposite direction and terminate this mode.

udiRefTi_sec: Referencing countdown display [s].

bInitRefCompl: Initial referencing process complete.

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

sErrDesc: Contains the error description.

<table>
<thead>
<tr>
<th>Error description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01: Error: Up/Down timer = 0.</td>
</tr>
<tr>
<td>02: Error: Turning timer = 0.</td>
</tr>
<tr>
<td>03: Error: Louvre angle limits: The upper limit is less than or equal to the lower limit ($rAnglLimUp\leq rAnglLimDwn$).</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

FB_BA_SunBldEvt

This function block serves to preset the position and angle for any desired event. It can be used, for example, in order to drive to a parking position or to drive the blind upward for maintenance.

The function is activated via the input bEn. If this is the case, the active flag in the positioning telegram (bActv in stSunBld) at output stSunBld [416] is set, and the values entered for the In/Out variables rPos for the blind height [%] and rAngl the louvre angle [''] are passed on in this telegram. If the function is no longer active due to the resetting of bEn, then the active flag in the positioning telegram stSunBld [416] is reset and the positions for height and angle are set to "0". The priority function block (e.g. FB_BA_SunBldPrioSwi4 [325]) enables a function with lower priority to take over the control by resetting.

VAR_INPUT

bEn : BOOL;
rPos : REAL;
rAngl : REAL;

bEn: A TRUE signal on this input activates the function block and transfers the entered setpoint values together with the active flag in the positioning telegram ST_BA_SunBld [416]. A FALSE signal resets the active flag again and sets position and angle to zero.

rPos: Height position of the blind [%] in case of activation.
rAngl: Louvre angle of the blind [°] in case of activation.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>stSunBld</th>
<th>ST_BA_SunBld;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bActv</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>

bActv: Corresponds to the boolean value bActv in the blind telegram ST_BA_SunBld [416] and is solely used to indicate whether the function block sends an active telegram.

stSunBld: Output structure of the blind positions, see ST_BA_SunBld [416]

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**FB_BA_SunBldIcePrtc**

The function block FB_BA_SunBldIcePrtc deals with direction-independent anti-freezing.

The weather protection has the highest priority in the blind controller (see overview [280]) and is intended to ensure that the blind is not damaged by ice or wind.

Impending icing up is detected by the fact that, during precipitation detection at bRainSns, the measured outside temperature rOtsT is below the frost limit rFrstT. This event is saved internally and remains active until it is ensured that the ice has melted again. In addition, the outside temperature must have exceeded the frost limit value for the entered deicing time udiDeiceTi_sec [s]. For safety reasons the icing event is persistently saved, i.e. also beyond a PLC failure. Thus, if the controller fails during the icing up or deicing period, the blind is considered to be newly iced up when then the controller restarts and the deicing timer starts from the beginning again.

If there is a risk of icing, the blind is moved to the protective position specified by rPosProt (height position in percent) and rAnglProt (louvre angle [°]).

**VAR_INPUT**

<table>
<thead>
<tr>
<th>bEn</th>
<th>BOOL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>rOtsT</td>
<td>REAL;</td>
</tr>
<tr>
<td>bRainSns</td>
<td>BOOL;</td>
</tr>
<tr>
<td>rFrstT</td>
<td>REAL;</td>
</tr>
<tr>
<td>udiDeiceTi_sec</td>
<td>UDINT;</td>
</tr>
<tr>
<td>rPosProt</td>
<td>REAL;</td>
</tr>
<tr>
<td>rAnglProt</td>
<td>REAL;</td>
</tr>
</tbody>
</table>

bEn: The function block has no function if this input is FALSE. In the positioning telegram ST_BA_Sunbld [416] 0 is output for the position and the angle, and bActv is FALSE. This means that another function takes over control of the blind via the priority controller.

rOtsT: Outside temperature [°C].

bRainSns: Input for a rain sensor.

rFrstT: Icing up temperature limit value [°] Celsius. This value may not be greater than 0. Otherwise an error is output.
udiDeiceTi_sec: Time until the deicing of the blind after icing up [s]. After that the icing up alarm is reset.

rPosProt: Height position of the blind [%] in the case of protection.

rAnglProt: Louvre angle of the blind [°] in the case of protection.

VAR_OUTPUT

| stSunBld | : ST_BA_SunBld;  |
| bActv | : BOOL;  |
| bIceAlm | : BOOL;  |
| udiRemTilIceAlm_sec | : UDINT;  |

stSunBld: Output structure of the blind positions, see ST_BA_SunBld [416].

bActv: Corresponds to the boolean value bActv in the blind telegram ST_BA_SunBld [416] and is solely used to indicate whether the function block sends an active telegram.

bIceAlm: Indicates the icing up alarm.

udiRemTilIceAlm_sec: In the case of impending icing up (bIceAlm=TRUE), this second counter is set to the deicing time. As soon as the temperature lies above the frost point entered (rFrstT), the remaining number of seconds until the 'all-clear' signal is given (bIceAlm=FALSE) are indicated here. This output is 0 as long as no countdown of the time is taking place.

If an error occurs, this automatic control is deactivated, and the position and angle are set to 0. This means that if a priority controller is in use, another function with a lower priority (see Overview) automatically takes over control of the blind. In the case of a direct connection, conversely, the blind will drive to position/angle 0.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

FB_BA_SunBldPosDly

This function block delays changes in position based on automatic commands.

If an event, e.g. weather protection, results in too many blind drives being started at the same time, fuses may be triggered by motor starting current peaks. It is therefore advisable to start the blind drives slightly staggered, in order to avoid excessive total current values.

This function block relays automatic commands from the input telegram stIn [416] to the output telegram stOut [416] with a delay. A distinction is made between three cases

1. the blind position rPos has changed in automatic mode (bManMode=FALSE in telegram stIn)
2. the louvre angle rAngl has changed in automatic mode (bManMode=FALSE in telegram stIn)
3. manual mode has just been exited, i.e. automatic mode has just become active (falling edge bManMode in telegram stIn)

The output telegram stOut is always a direct copy of the input telegram stIn. In these three cases, however, the output telegram stOut is set for the time udiDly_ms [ms].

This ensures that the blind controlled via the function block FB_BA_SunBldActr [317] is kept at its position during the delay period. Each further change based on the criteria mentioned above within the delay time restarts the timer.
However, a change to manual in the input telegram (bManMode = TRUE) cancels the delay timer immediately. The (manual) telegram is passed on without delay. In this way, only automatic telegrams are delayed.

Application

Preferably directly before the blind actuator function block:

```
VAR_INPUT
stIn : ST_BA_Sunblind;
udiDly_ms : UDINT;

stIn: Input positioning telegram, see ST_BA_SunBld [416].
udiDly_ms: Delay time of the active bit in the positioning telegram [ms].

VAR_OUTPUT
stOut : ST_BA_Sunblind;
udiRemTiDly_sec : UDINT;

stOut: Output positioning telegram, see ST_BA_SunBld [416].
udiRemTiDly_sec: Display output for elapsed delay time [s].
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**FB_BA_SunBldPrioSwi4**

The function block is used for priority control for up to 4 positioning telegrams (stSunBld_Prio1 ... stSunBld_Prio4) of type ST_BA_SunBld [416].

Structure of the blind positioning telegram ST_BA_SunBld [416]:

```
TYPE ST_BA_SunBld:
STRUCT
  rPos : REAL;
  rAngl : REAL;
  bManUp : BOOL;
  bManDwn : BOOL;
  bManMod : BOOL;
```
Programming

bActv : BOOL;
END_STRUCT
END_TYPE

Up to 4 positioning telegrams from different control function blocks can be applied to this function block. The telegram on \textit{stSunBld\_Prio1} has the highest priority and that on \textit{stSunBld\_Prio4} the lowest. The active telegram with the highest priority is output at the output \textit{stSunBld}. "Active" means that the variable \textit{bActv} is set within the structure of the positioning telegram.

This function block is to be programmed in such a way that one of the applied telegrams is always active. If no telegram is active, an empty telegram is output, i.e. \textit{rPos}=0, \textit{rAngl}=0, \textit{bManUp}=FALSE, \textit{bManDwn}=FALSE, \textit{bManMod}=FALSE, \textit{bActv}=FALSE. Since the blind function block \textit{FB\_BA\_SunBldActr} \[317\] or the roller blind function block \textit{FB\_BA\_RolBldActr} \[309\] does not take account of the flag \textit{bActv}, this telegram would be interpreted as movement command to position "0", i.e. fully open. The absence of an active telegram therefore does not represent a safety risk for the blind.

\textbf{VAR\_INPUT}

\begin{verbatim}
stSunBld\_Prio1 : ST_BA_SunBld;
stSunBld\_Prio2 : ST_BA_SunBld;
stSunBld\_Prio3 : ST_BA_SunBld;
stSunBld\_Prio4 : ST_BA_SunBld;
\end{verbatim}

\textit{stSunBld\_Prio1...stSunBld\_Prio4}: Positioning telegrams available for selection. \textit{stSunBld\_Prio1} has the highest priority and \textit{stSunBld\_Prio4} the lowest.

\textbf{VAR\_OUTPUT}

\begin{verbatim}
stSunBld : ST_BA_SunBld;
udiActvPrio : UDINT;
\end{verbatim}

\textit{stSunBld}: Resulting positioning telegram.
\textit{udiActvPrio}: Active positioning telegram. If none is active, "0" is output.

\textbf{Requirements}

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

\textbf{FB\_BA\_SunBldPrioSwi8}

The function block is used for priority control for up to 8 positioning telegrams (\textit{stSunBld\_Prio1} ... \textit{stSunBld\_Prio8}) of type \textit{ST\_BA\_SunBld} \[416\].

Structure of the blind positioning telegram \textit{ST\_BA\_Sunbld} \[416\].

\begin{verbatim}
TYPE ST_BA_SunBld:
STRUCT
 rPos : REAL;
 rAngl : REAL;
 bManUp : BOOL;
 bManDwn : BOOL;
 bManMod : BOOL;
\end{verbatim}
Up to 8 positioning telegrams from different control function blocks can be applied to this function block. The telegram on stSunBld_Prio1 has the highest priority and that on stSunBld_Prio8 the lowest. The active telegram with the highest priority is output at the output stSunBld. "Active" means that the variable bActv is set within the structure of the positioning telegram.

This function block is to be programmed in such a way that one of the applied telegrams is always active. If no telegram is active, an empty telegram is output, i.e. rPos=0, rAngl=0, bManUp=FALSE, bManDwn=FALSE, bManMod=FALSE, bActv=FALSE. Since the blind function block FB_BA_SunBldActr [317] or the roller blind function block FB_BA_RolBldActr [309] does not take account of the flag bActv, this telegram would be interpreted as movement command to position "0", i.e. fully open. The absence of an active telegram therefore does not represent a safety risk for the blind.

VAR_INPUT
stSunBld_Prio1 : ST_BA_SunBld;
stSunBld_Prio2 : ST_BA_SunBld;
stSunBld_Prio3 : ST_BA_SunBld;
stSunBld_Prio4 : ST_BA_SunBld;
stSunBld_Prio5 : ST_BA_SunBld;
stSunBld_Prio6 : ST_BA_SunBld;
stSunBld_Prio7 : ST_BA_SunBld;
stSunBld_Prio8 : ST_BA_SunBld;

stSunBld_Prio1..stSunBld_Prio8: Positioning telegrams available for selection. stSunBld_Prio1 has the highest priority and stSunBld_Prio8 the lowest.

VAR_OUTPUT
stSunBld : ST_BA_SunBld;
udiActvPrio : UDINT;

stSunBld: Resulting positioning telegram.
udiActvPrio: Active positioning telegram. If none is active, "0" is output.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

FB_BA_SunBldScn

This function block represents an extension of the manual controller FB_BA_SunBldSwi [330] by a scene memory and a call function. The blind control FB_BA_SunBldActr [317] or the roller blind control FB_BA_RolBldActr [309] can be active in manual mode and also directly target previously stored positions (scenes). Up to 21 scenes can be saved.

Structure of the blind positioning telegram ST_BA_SunBld [416].
**Operation**

In manual mode, the function block controls the blind function block `FB_BA_SunBldActr` or the roller shutter function block `FB_BA_RolBldActr` via the command inputs `bUp` and `bDwn`; `bUp` has priority. The commands are passed on to the respective commands `bManUp` and `bManDwn` of the positioning telegram. If a command input is activated that is longer than the entered time `udiSwiOvrTi_ms` [ms], then the corresponding control command latches. Activating a command input again releases this latch.

A rising edge on `bSavScn` saves the current position and louvre angle in the scene selected in `udiSlcdScn`. This procedure is possible at any time, even during active positioning. The selected scene is called with `bClScn`, i.e. the saved position and angle values are driven to.

If the function block is activated by input `bEn=TRUE`, bit `bActv` is set immediately in the positioning telegram. The function block uses this to notify a priority switch (`FB_BA_SunBldPrioSwi4` or `FB_BA_SunBldPrioSwi8`) of its priority over lower priorities. If the command "Call Scene" is not active (`bClScn =TRUE`), the bit `bManMod` is also set in the positioning telegram to notify the connected actuator function blocks that they should respond to manual commands.

If the function block is deactivated by `bEn=FALSE`, both bits, `bActv` and `bManMod`, are set to FALSE again.

**Linking to the blind function block**

Like the "normal" manual mode function block `FB_BA_SunBldSwi`, the scene selection function block can be connected either via an upstream priority control `FB_BA_SunBldPrioSwi4` or `FB_BA_SunBldPrioSwi8`, or directly via the blind function block. The connection is established via the positioning telegram `ST_BA_Sunbld`. Furthermore the scene function block requires the current positions from the blind function block for the reference blind:

**Use of a priority controller:**

![Diagram](image)

---

**TYPE ST_BA_SunBld:**

```plaintext
STRUCT
    rPos    : REAL;
    rAngl   : REAL;
    bManUp  : BOOL;
    bManDwn : BOOL;
    bManMod : BOOL;
    bActv   : BOOL;
END_STRUCT
END_TYPE
```
VAR_INPUT

bEn : BOOL;
bUp : BOOL;
bDwn : BOOL;
udiSwiOvrTi_ms : UDINT;
udiSlcdScn : UDINT;
bClScn : BOOL;
bSavScn : BOOL;
rSpPos : REAL;
rSpAngl : REAL;

bEn: The function block has no function if this input is FALSE. In the positioning telegram ST_BA_SunBld [416], 0 is output for the position and the angle - bManMod and bActv are FALSE. For a connection with priority controller this means that another functionality takes over control of the blind. Conversely, a direct connection allows the blind to drive directly to the 0 position, i.e. fully up, since the actuator function block does not evaluate the bit bActv itself.

bUp: Command input for blind up.

bDwn: Command input for blind down.

udiSwiOvrTi_ms: Time [ms] until the corresponding manual command in the positioning telegram ST_BA_SunBld [416] switches to latching mode, if the command input is activated permanently. Internally limited to a minimum value of 0.

udiSlcdScn: Selected scene which should either be saved (bSavScn) or called (bClScn). Internally limited to a minimum value of 0 to cMaxSunBldScn.

bClScn: Call selected scene.

bSavScn: Save selected scene.

rSpPos: Set position [%] that is to be saved in the selected scene. This must be linked to the actual position of the actuator function block FB_BA_SunBldActr [317] or FB_BA_RolBldActr [309] of the reference blind/roller shutter, in order to be able to save a position that was previously approached manually. Internally limited to values between 0 and 100.

rSpAngl: ditto. Louvre angle [°].

VAR_OUTPUT

stSunBld : ST_BA_SunBld;
bActv : BOOL;
rActlScnPos : REAL;
rActlScnAngl : REAL;

stSunBld: Positioning telegram, see ST_BA_SunBld [416].

bActv: Corresponds to the boolean value bActv in the blind telegram ST_BA_SunBld [416] and is solely used to indicate whether the function block sends an active telegram.
**rActlScnPos:** Indicates the saved relative blind height position [%] for the currently selected scene.

**rActlScnAngl:** ditto. Louvre angle [°].

If an error occurs, this automatic control is deactivated, and the position and angle are set to 0. This means that if a priority controller is in use, another function with a lower priority (see Overview) automatically takes over control of the blind. In the case of a direct connection, conversely, the blind will drive to position/angle 0.

**VAR_IN_OUT**

| arrSunBldScn | ARRAY[0..Param.usiMaxSunBldScn] OF ST_BA_SunBldScn; |

**arrSunBldScn:** Table with the scene entries of the type **ST_BA_SunBldScn** [417].

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**FB_BA_SunBldSwi**

This function block can be used to control the blind **FB_BA_SunBldActr** [317] or roller shutter **FB_BA_RolBldActr** [309] in manual mode. The connection takes place via the positioning telegram **ST_BA_Sunbld** [416] either directly or with an additional priority controller.

**Structure of the blind positioning telegram ST_BA_Sunbld** [416].

```plaintext
TYPE ST_BA_SunBld:
  STRUCT
    rPos    : REAL;
    rAngl   : REAL;
    bManUp  : BOOL;
    bManDwn : BOOL;
    bManMod : BOOL;
    bActv   : BOOL;
  END_STRUCT
END_TYPE
```

**Operation**

In manual mode, the function block controls the blind function block **FB_BA_SunBldActr** [317] or the roller shutter function block **FB_BA_RolBldActr** [309] via the command inputs **bUp** and **bDwn**; **bUp** has priority. The commands are passed on to the respective commands **bManUp** and **bManDwn** of the positioning telegram. If a command input is activated that is longer than the entered time **udiSwiOvrTi_ms** [ms], then the corresponding control command latches. Activating a command input again releases this latch. If the function block is activated by input **bEn**=TRUE, bit **bActv** is set immediately in the positioning telegram. The function block uses this to notify a priority switch (**FB_BA_SunBldPrioSwi4** [325] or **FB_BA_SunBldPrioSwi8** [326]) of its priority over lower priorities. At the same time, the bit **bManMod** is set in the positioning telegram to notify the connected actuator function blocks that they should respond to manual commands.

If the function block is deactivated by **bEn**=FALSE, both bits, **bActv** and **bManMod**, are set to FALSE again.
Linking to the blind function block

The manual mode function block can be connected either via an upstream priority control \texttt{FB\_BA\_SunBldPrioSwi4 [325]} or \texttt{FB\_BA\_SunBldPrioSwi8 [326]}, or directly at the blind function block. The connection is established via the positioning telegram \texttt{ST\_BA\_Sunbld [416]}.

Use of a priority controller:

Direct connection:

\begin{verbatim}
VAR_INPUT
  bEn : BOOL;
  bUp : BOOL;
  bDwn : BOOL;
  udiSwiOvrTi_ms : UDINT;

bEn: The function block has no function if this input is FALSE. In the positioning telegram \texttt{ST\_BA\_Sunbld [416]}, 0 is output for the position and the angle - \textit{bManMod} and \textit{bActv} are FALSE. For a connection with priority controller this means that another functionality takes over control of the blind. Conversely, a direct connection allows the blind to drive directly to the 0 position, i.e. fully up, since the actuator function block does not evaluate the bit \textit{bActv} itself.

bUp: Command input for blind up.

bDwn: Command input for blind down.

udiSwiOvrTi_ms: Time [ms] until the corresponding manual command in the positioning telegram \texttt{ST\_BA\_Sunbld [416]} switches to latching mode, if the command input is activated permanently. Internally limited to a minimum value of 0.

VAR_OUTPUT
  stSunBld : ST\_BA\_SunBld;
  bActv : BOOL;

stSunBld: Positioning telegram, see \texttt{ST\_BA\_SunBld [416]}.
\end{verbatim}
bActv: Corresponds to the boolean value bActv in the blind telegram ST_BA_SunBld [416] and is solely used to indicate whether the function block sends an active telegram.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3 BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

FB_BA_SunBldTwiLgtAuto

This function block controls the blind when the outdoor brightness has fallen below a limit value.

The automatic twilight function operates with both a value hysteresis and a temporal hysteresis: If the outdoor brightness value \( rBrtns \) [lux] falls below the value \( rActvVal \) [lux] for the time \( udiActvDly\_sec \) [s], the function block is active and will provide the blind positions \( rPosTwiLgt \) (height [%]) and \( rAnglTwiLgt \) (louvre angle [°]) specified for the input variables at the output in the positioning telegram ST_BA_Sunbld [416]. If the outdoor brightness exceeds the value \( rActvVal \) [lux] for the time \( udiDctvDly\_sec \) [s], automatic mode is no longer active. The active flag in the positioning telegram ST_BA_Sunbld [416] is reset and the positions for height and angle are set to "0". A function with a lower priority can then take over control.

VAR_INPUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEn</td>
<td>BOOL</td>
<td>Function block has no function if this input is FALSE. In the positioning telegram ST_BA_SunBld [416] 0 is output for the position and the angle, and bActv is FALSE. This means that another function takes over control of the blind via the priority controller.</td>
</tr>
<tr>
<td>rBrtns</td>
<td>REAL</td>
<td>Outdoor brightness [lx].</td>
</tr>
<tr>
<td>rActvVal</td>
<td>REAL</td>
<td>Activation limit value [lx]. The value rActvVal is internally limited to values from 0 to rDctvVal.</td>
</tr>
<tr>
<td>rDctvVal</td>
<td>REAL</td>
<td>Deactivation limit value [lx]. Internally limited to a minimum value of 0.</td>
</tr>
<tr>
<td>udiActvDly_sec</td>
<td>UDINT</td>
<td>Activation delay [s]. Internally limited to a minimum value of 0.</td>
</tr>
<tr>
<td>udiDctvDly_sec</td>
<td>UDINT</td>
<td>Deactivation delay [s]. Internally limited to a minimum value of 0.</td>
</tr>
<tr>
<td>rPosTwiLgt</td>
<td>REAL</td>
<td>Vertical position of the blind [%] if the automatic twilight function is active. Internally limited to values between 0 and 100.</td>
</tr>
<tr>
<td>rAnglTwiLgt</td>
<td>REAL</td>
<td>Louvre angle of the blind [°] if the automatic twilight function is active.</td>
</tr>
</tbody>
</table>
VAR_OUTPUT

stSunBld : ST_BA_SunBld;
bActv : BOOL;
udiRemTiActv_sec : UDINT;
udiRemTiDctv_sec : UDINT;

stSunBld: Output structure of the blind positions, see ST_BA_SunBld [416].

bActv: Corresponds to the boolean value bActv in the blind telegram ST_BA_SunBld [416] and is solely used to indicate whether the function block sends an active telegram.

udiRemTiActv_sec: Shows the time remaining [s] after falling below the switching value rActvVal until automatic mode is activated. This output is 0 as long as no countdown of the time is taking place.

udiRemTiDctv_sec: Shows the time remaining [s] after exceeding of the switching value rDctvVal until automatic mode is disabled. This output is 0 as long as no countdown of the time is taking place.

If an error occurs, this automatic control is deactivated, and the position and angle are set to 0. This means that if a priority controller is in use, another function with a lower priority (see Overview) automatically takes over control of the blind. In the case of a direct connection, conversely, the blind will drive to position/angle 0.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

FB_BA_SunBldWndPrtc

The function block FB_BA_SunBldWndPrtc deals with the direction-dependent wind protection.

The weather protection has the highest priority in the blind controller (see overview [280]) and is intended to ensure that the blind is not damaged by ice or wind.

If the measured wind speed is above the value rWndSpdStrmOn for the time udiDlyStrmOn_sec [s], it is assumed that high winds are imminent. The storm is regarded as having subsided, so that the blind can be moved safely, once the wind speed falls below the value rWndSpdStrmOff for the time udiDlyStrmOff_sec [s]. For safety reasons the storm event is also persistently saved. Thus, if the controller fails during a storm, the sequence timer is started again from the beginning when the controller is restarted.

If there is a risk of high wind, the blind is moved to the protection position specified by rPosProt (height position in percent) and rAnglProt (louvre angle [°]).

VAR_INPUT

bEn : BOOL;
rWndSpd : REAL;
rWndSpdStrmOn : REAL;
rWndSpdStrmOff : REAL;
udiDlyStrmOn_sec : UDINT;
udiDlyStrmOff_sec : UDINT;
rPosProt : REAL;
rAnglProt : REAL;
bEn: The function block has no function if this input is FALSE. In the positioning telegram ST_BA_Sunbld [416] 0 is output for the position and the angle, and bActv is FALSE. This means that another function takes over control of the blind via the priority controller.

rWndSpd: Wind speed. The unit of entry is arbitrary, but it is important that no value is smaller than 0 and that the values become larger with increasing speed.

rWndSpdStrmOn: Wind speed limit value for the activation of the storm alarm. This value may be not smaller than 0 and must lie above the value for the deactivation. Otherwise an error is output. The unit of entry must be the same as that of the input rWndSpd. A value greater than this limit value triggers the alarm after the specified time udiDlyStrmOn_sec.

rWndSpdStrmOff: Wind speed limit value for the deactivation of the storm alarm. This value may be not smaller than 0 and must lie below the value for the activation. Otherwise an error is output. The unit of entry must be the same as that of the input rWndSpd. A value smaller than or equal to this limit value resets the alarm after the specified time udiDlyStrmOff_sec.

udiDlyStrmOn_sec: Time delay until the storm alarm is triggered [s].

udiDlyStrmOff_sec: Time delay until the storm alarm is reset [s].

rPosProt: Height position of the blind [%] in the case of protection.

rAnglProt: Louvre angle of the blind [°] in the case of protection.

VAR_OUTPUT

stSunBld : ST_BA_SunBld;
bActv : BOOL;
bStrmAlm : BOOL;
udiRemTiStrmDetc_sec : UDINT;
udiRemTiStrmAlm_sec : UDINT;

stSunBld: Output structure of the blind positions, see ST_BA_SunBld [416].

bActv: Corresponds to the boolean value bActv in the blind telegram ST_BA_SunBld [416] and is solely used to indicate whether the function block sends an active telegram.

bStrmAlm: Indicates the storm alarm.

udiRemTiStrmDetc_sec: In the non-critical case, this second counter continuously shows the alarm delay time udiDlyStrmOn_sec. If the measured wind speed rWndSpd is above the activation limit value rWndSpdStrmOn, the seconds to the alarm are counted down. This output is 0 as long as no countdown of the time is taking place.

udiRemTiStrmAlm_sec: As soon as the storm alarm is initiated, this second counter initially constantly indicates the deactivation time delay of the storm alarm udiDlyStrmOff_sec. If the measured wind speed rWndSpd falls below the deactivation limit value rWndSpdStrmOff, the seconds to the all-clear signal (bStrmAlm=FALSE) are counted down. This output is 0 as long as no countdown of the time is taking place.

If an error occurs, this automatic control is deactivated, and the position and angle are set to 0. This means that if a priority controller is in use, another function with a lower priority (see Overview) automatically takes over control of the blind. In the case of a direct connection, conversely, the blind will drive to position/angle 0.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
The function block is used for glare protection with the aid of a slatted blind.

Glare protection is realized through variation of the louvre angle and positioning of the blind height.

The louvre angle is set as a function of the sun position such that direct glare is prevented, while letting as much natural light through as possible.

Three different operating modes are available for varying the blind height.

1. When sun protection is active, the blind moves to a fixed height. The height value is specified with the variable \( r_{FixPos} \).

2. The blind position is varied as a function of the sun position. The position is specified in the table \((ST_{-}B_{A}\_BldPosTab)\). See also description of \(FB_{-}B_{A}\_BldPosEntry\).

3. The height of the blind is calculated based on the window geometry such that the sun's rays reach a specified depth in the room. The incidence depth of the sun's rays is defined with the variable \( r_{MaxLgtIndc} \).

In order to avoid excessive repositioning of the louvre angle, the variable \( udi\_PosIntval\_min \) can be used to specify a time interval, within which the louvre angle is not adjusted. In order to avoid glare, the angle is always changed sufficiently for the respective time interval.

The following conditions must be met for positioning the blind and setting the louvre angle.

- 1. The input \( bEn \) must be TRUE.
- 2. The sun must have risen. (elevation > 0)
- 3. The function block is parameterized correctly (\( bErr = FALSE \))

**VAR_INPUT**

```
bEn             : BOOL;
stUTC           : TIMESTRUCT;
uDiPosIntval_min : UDINT;
rDegLngd        : REAL;
rDegLatd        : REAL;
rFcdOrtn        : REAL;
rFcdAngl        : REAL;
FcdOrtn          : REAL;
FcdAngl          : REAL;
FcdOrtn          : REAL;
FcdAngl          : REAL;
FcdOrtn          : REAL;
FcdAngl          : REAL;
windHgt          : REAL;
IndcWdwFr        : REAL;
stBldPosTab      : ST_{-}B_{A}\_BldPosTab;
ePosMod          : E_{-}B_{A}\_PosMod;
```
bEn: If this input is set to FALSE the positioning is inactive, i.e. the active bit (bActv) is reset in the positioning structure stSunBld of the type ST_BA_SunBld [416] and the function block itself remains in a standstill mode. If on the other hand the function block is activated, then the active bit is TRUE and the function block outputs its control values (rPos, rAngl) in the positioning structure at the appropriate times.

stUTC: Input of current time as coordinated world time (UTC - Universal Time Coordinated, previously referred to as GMT, Greenwich Mean Time) (see TIMESTRUCT). The function block FB_BA_GetTime [344] can be used to read this time from a target system.

A jump of more than 300 seconds leads to immediate repositioning, if the blind is in the sun and glare protection is active, based on the above criteria. This functionality was added to ensure a reproducible program execution.

udiPosIntval_min: Positioning interval in minutes - time between two blind position outputs. Valid range: 1 min...720 min.

rDegLngd: Longitude [°]. Valid range: -180°...180°.

rDegLatd: Latitude [°]. Valid range: -90°...90°.

rFcdOrtn: Facade orientation [°]:

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

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

The following applies for the southern hemisphere:

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

rFcdAngl: Facade inclination [°]. See facade inclination [285].

rLamWdth: Width of the louvres in mm, see sketch [282].

rLamDstc: Louvre spacing in mm, see sketch [282].

rFixPos: Fixed (constant) shutter height [0..100%]. Applies if ePosMod = ePosModFix (see enumerator E_BA_PosMod [412]).

rMaxLgtIndc: Maximum desired light incidence in mm measured from the outside of the wall (see Sun protection: Basic principles and definitions [284]). The parameters rWdwHght and rDstcWdwFlr are used to calculate how high the blinds must be, depending on the position of the sun, such that the incidence of light does not exceed the value rMaxLgtIndc. Applies if ePosMod = ePosModeMaxIncidence (see enumerator E_BA_PosMod [412]).

rWdwHght: Window height in mm for the calculation of the shutter height if the mode "maximum desired incidence of light" is selected.

rDstcWdwFlr: Distance between the floor and the window sill in mm for the calculation of the shutter height if the mode "maximum desired incidence of light" is selected.
**stBldPosTab**: Table of 6 interpolation points, 4 of which are parameterizable, from which a blind position is then given in relation to the position of the sun by linear interpolation. Applies if `ePosMod = ePosModFix` (see enumerator `E_BA_PosMod[▸412]`). For a more detailed description please refer to `FB_BA_BldPosEntry[▸287].`

**ePosMod**: Selection of the positioning mode, see enumerator `E_BA_PosMod[▸412].`

**VAR_OUTPUT**

```plaintext
VAR_OUTPUT
  stSunBld   : ST_BA_SunBld;
  bActv      : BOOL;
  bErr       : BOOL;
  sErrDescr  : T_MAXSTRING;
```

**stSunBld**: Output structure of the blind positions, see `ST_BA_SunBld[▸416]`

**bActv**: The function block is in active state, i.e. no error is pending, the function block is enabled, and the sun position is in the specified facade area (the facade is sunlit).

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

**sErrDescr**: Contains the error description.

### Error description

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Error: The duration of the positioning interval is less than or equal to zero, or it exceeds 720 min.</td>
</tr>
<tr>
<td>02</td>
<td>Error: The longitude entered is not within the valid range from -180°..180°.</td>
</tr>
<tr>
<td>03</td>
<td>Error: The latitude entered is not within the valid range from -90°..90°.</td>
</tr>
<tr>
<td>04</td>
<td>Error: The value entered for the facade inclination (r\text{FcdAngl}) is outside the valid range of -90°..90°.</td>
</tr>
<tr>
<td>05</td>
<td>Error: The value for the louvre spacing (r\text{LamDstc}) is greater than or equal to the value for the louvre width (r\text{LamWdth}). This does not represent a &quot;valid&quot; blind, since the louvres cannot close fully. Mathematically, this would lead to errors.</td>
</tr>
<tr>
<td>06</td>
<td>Error: The value entered for the louvre width (r\text{LamWdth}) is zero.</td>
</tr>
<tr>
<td>07</td>
<td>Error: The value entered for the louvre spacing (r\text{LamDstc}) is zero.</td>
</tr>
<tr>
<td>08</td>
<td>Error: The value entered for the fixed blind height (r\text{FixPos}) is greater than 100 or less than 0. At the same time, &quot;fixed blind height&quot; positioning is selected - (e\text{PosMod}=e\text{PosModFix}).</td>
</tr>
<tr>
<td>09</td>
<td>Error: The bit &quot;values valid&quot; (b\text{Vld}) in the positioning table (st\text{BldPosTab}) is not set - invalid values: see <code>FB_BA_BldPosEntry</code>. At the same time, &quot;Table&quot; positioning is selected - (e\text{PosMod}=e\text{PosModTab}).</td>
</tr>
<tr>
<td>10</td>
<td>Error: The value entered for the maximum required light incidence (r\text{MaxLgtIndc}) is less than or equal to zero. At the same time, &quot;maximum light incidence&quot; is selected - (e\text{PosMod}=e\text{PosModMaxIndc}).</td>
</tr>
<tr>
<td>11</td>
<td>Error: The value entered for the window height (r\text{WdwHght}) is less than or equal to zero. At the same time, &quot;maximum light incidence&quot; is selected - (e\text{PosMod}=e\text{PosModMaxIndc}).</td>
</tr>
<tr>
<td>12</td>
<td>Error: The distance between lower window edge and floor (r\text{DstcWdwFlr}) that was entered is less than zero. At the same time, &quot;maximum light incidence&quot; is selected - (e\text{PosMod}=e\text{PosModMaxIndc}).</td>
</tr>
<tr>
<td>13</td>
<td>Error: An invalid positioning mode is entered at input (e\text{PosMod}).</td>
</tr>
</tbody>
</table>

If an error occurs, this automatic control is deactivated, and the position and angle are set to 0. This means that if a priority controller is in use, another function with a lower priority (see Overview) automatically takes over control of the blind. In the case of a direct connection, conversely, the blind will drive to position/angle 0.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
**Provision of hot water**

**Function blocks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_BA_DHW2P</td>
<td>Charge control for a hot water tank via an on-off controller.</td>
</tr>
<tr>
<td>FB_BA_LglPrev</td>
<td>Function block for disinfecting service water and destroying legionella.</td>
</tr>
</tbody>
</table>

**FB_BA_DHW2P**

This function block controls the heating of a hot water tank via an on-off controller. Tank heating is activated at input `bEn`. If tank heating is active the output `bLd` is TRUE. The variable `rSp` is used to transfer the setpoint for the hot water temperature to the function block. At input `rTMin` a minimum selection of all temperature sensors for the hot water tank is connected, at input `rTMax` a maximum selection of all temperature sensors.

Due to the thermal stratification in the hot water tank, the sensor at the top is generally the one showing the highest temperature, the one at the bottom the lowest.

The tank can be charged in two ways via the variables `bKepFul`:

**bKepFul = FALSE**

Charging is requested if `rTMax` falls below the value of `rSp-rSpHys`. The charge request is disabled if `rTMin` is above the setpoint of `rSp`.

Due to the fact that the sensor at the top generally measures the highest temperature, the heating is not switched on until the hot water tank has been discharged.
bKepFul = TRUE

Charging is requested if $rTMin$ falls below the value of $rSp-rSpHys$. The charge request is disabled once $rTMin$ is above the setpoint again.

Selecting the minimum of all tank temperatures ensures that the coldest point of the tank is used for control purposes. Recharging takes place when the tank is no longer full.

*VAR_INPUT*

<table>
<thead>
<tr>
<th>bEn</th>
<th>BOOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>rSp</td>
<td>REAL</td>
</tr>
<tr>
<td>rSpHys</td>
<td>REAL</td>
</tr>
<tr>
<td>rTMax</td>
<td>REAL</td>
</tr>
<tr>
<td>rTMin</td>
<td>REAL</td>
</tr>
<tr>
<td>bKepFul</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

- **bEn**: Enable boiler charging.
- **rSp**: Service water temperature setpoint [°C].
- **rSpHys**: Hysteresis, recommended 1°K to 5°K.
- **rTMax**: Maximum selection of all tank temperatures [°C].
- **rTMin**: Minimum selection of all tank temperatures [°C].
- **bKepFul**: Control temperature selection:
  - FALSE = $rTMax$ is used to request $bLd$, $rTMin$ to switch off
  - TRUE = $rTMin$ alone controls switching on/off of $bLd$

*VAR_OUTPUT*

<table>
<thead>
<tr>
<th>bLd</th>
<th>BOOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>rSpOut</td>
<td>REAL</td>
</tr>
</tbody>
</table>

- **bLd**: Enable charging mode.
- **rSpOut**: Setpoint transfer to charging circuit:
  - $rSpOut = rSp$ (input) if the function block is enabled
  - $rSpOut = 0$ if the function block is not enabled

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
This function block is used for disinfection of the service water and for killing off Legionella. Disinfection mode is activated at input \( bEnLglPrev \) via a timer program. It is advisable to run the disinfection at least once per week (during the night). The temperature should be at least 70 °C. The activation interval at \( bEnLglPrev \) must be adequately long. The output \( bLd \) activates tank heating.

For hot water tanks with several temperature sensors, a minimum selection feature for all sensors must be connected at \( rTMin \).

If \( rTMin \) exceeds the value of \( rSp \), a monitoring timer is started with a time of \( udiTi\_sec \) [s]. If the minimum tank temperature \( rTMin \) remains above \( rSp - rSpHys \) while the timer is active, the tank was heated adequately. If circulation is active, the output \( bLd \) must be linked to enabling of the circulation pump, to ensure that the water pipe within the service water system is included in the disinfection. If the temperature has fallen below \( rSp - rSpHys \) during the disinfection process, the process must be restarted and run until the time \( udiTi\_sec \) has fully elapsed. If the disinfection was successful, the output \( bLd \) is reset.

If the disinfection process was incomplete during the function block activation (\( bEnLglPrev \)), this is indicated with the output \( bAlm \). The output must be reset with \( bRst \).
Explanation of the diagram:

t0 Start of the legionella program and switching of output bLd. Heating of the hot water tank.

t1 The tank has reached the temperature rSp. The timer for the heating time is started.

t2 The minimum tank temperature has fallen below rSp - rSpHys. The timer for the heating time is reset.

t3 The temperature exceeds rSp again, and the heating timer is started again.

t4 The minimum tank temperature was above the limit rSp - rSpHys over the period udiTi_sec; the disinfection was successful. bLd is reset, and the hot water tank switches back to normal operation.

VAR_INPUT

bEnLglPrev : BOOL;
rTMin : REAL;
rSp : REAL;
rSpHys : REAL;
udiTi_sec : UDINT;
bRst : BOOL;

bEnLglPrev: Enabling of disinfection operation via a timer program.

rTMin: Minimum tank temperature [°C]. Minimum selection of temperature sensors at the top and bottom.

rSp: Setpoint for disinfection [°C].

rSpHys: Temperature difference [K] lower limit; always calculated absolute.

udiTi_sec: Monitoring period [s].

bRst: Resetting of the legionella alarm;

VAR_OUTPUT

bLd : BOOL;
rSpOut : REAL;
udiRTi : UDINT;
udiSta : UDINT;

bLd: Anti-legionella mode active.

rSpOut: Setpoint transfer to charging circuit:

• rSp (input) if the function block is enabled
• 0 if the function block is not enabled

udiRTi: Disinfection mode timer countdown.

udiSta: Disinfection program status:

1. The disinfection operation was successful.
2. The disinfection was completed successfully. After the disinfection, and to reactivate legionella prevention, bEnLglPrev must be FALSE.
3. The disinfection operation is active.
4. Disinfection was not successful. Alarm is pending.
5. Disinfection was not successful, the alarm was acknowledged.
6. Controller restart, or legionella mode has not yet been requested.

bAlm: The temperature setpoint was not reached consistently over via the interval udiTi_sec, so that adequate disinfection is not guaranteed.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
### System

#### Function blocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_BA_CnvTiSt [<em>342]</em></td>
<td>Conversion year, month, day, hour, minute and second in time structure</td>
</tr>
<tr>
<td>FB_BA_ExtTiSt [<em>343]</em></td>
<td>Conversion time structure in year, month, day, hour, minute and second</td>
</tr>
<tr>
<td>FB_BA_GetTime [<em>344]</em></td>
<td>Internal clock with time information - can be synchronized with system time</td>
</tr>
<tr>
<td>FB_BA_SetTime [<em>345]</em></td>
<td>Setting the system time</td>
</tr>
<tr>
<td>FB_BA_WrtPersistDat [*347]</td>
<td>Writes persistent data</td>
</tr>
</tbody>
</table>

**FB_BA_CnvTiSt**

The function block **FB_BA_CnvTiSt** can be used to consolidate the different components of a time structure.

- **Note:**
  - The function block does not check for incorrect entries, such as an hour entry of 99. It makes sense to check this in the connected function blocks, which have to check the time structure in any case. The limit values are shown as part of the variable explanations.

**VAR_INPUT**

- **wYear**: The year (1970..2106).
- **wMonth**: The month (1..12).
- **wDay**: The day of the month (1..31).
- **wHour**: The hour (0..23).
- **wMinute**: The minutes (0..59).
- **wSecond**: The seconds (0..59).
- **wMilliseconds**: The milliseconds (0..999).

**VAR_OUTPUT**

- **stTi**: Output time structure (see TIMESTRUCT)
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

FB_BA_ExtTiSt

The function block `FB_BA_ExtTiSt` resolves a time structure into the different components, so that it can be used for time conditions, for example.

**VAR_INPUT**

```
stTi : TIMESTRUCT;
```

`stTi`: Input time structure (see TIMESTRUCT)

**VAR_OUTPUT**

```
wYear       : WORD;
wMonth      : WORD;
wDayOfWeek  : WORD;
wDay        : WORD;
wHour       : WORD;
wMinute     : WORD;
wSecond     : WORD;
wMilliseconds : WORD;
```

- **wYear**: The year (1970..2106).
- **wMonth**: The month (1..12).
- **wDayOfWeek**: The day of the week (0(Sun)..0(Sat)).
- **wDay**: The day of the month (1..31).
- **wHour**: The hour (0..23).
- **wMinute**: The minutes (0..59).
- **wSecond**: The seconds (0..59).
- **wMilliseconds**: The milliseconds (0..999).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
With this function block an internal clock (Real Time Clock RTC) can be implemented in the TwinCAT PLC.

When the function block is enabled via \texttt{bEn}, the RTC clock is initialized with the current NT system time. One system cycle of the CPU is used to calculate the current RTC time. The function block must be called once per PLC cycle in order for the current time to be calculated. Within the function block, an instance of the function blocks \texttt{NT_GetTime}, \texttt{FB_GetTimeZoneInformation} and \texttt{RTC_EX2} is called. The time is output at the outputs \texttt{stSysTi} for the read system time and \texttt{stUtc} for the Coordinated Universal Time (UTC). This is determined internally from the system time and the time zone. If the system time and/or the time zone was entered incorrectly, the UTC time will also be wrong.

The system time is read cyclically via the timer to be set \texttt{udiUpdRTC_sec} [s]; it is used to synchronize the internal RTC clock. The time information (time zone, time shift relative to UTC, summer/winter time) is read in the same cycle. The output \texttt{udiRemTiUpd_sec} indicates the seconds remaining to the next read cycle.

The time structures that are output, \texttt{stSysTi} and \texttt{stUtc}, can be resolved with the aid of the function block \texttt{FB_BA_ExtTiSt [343]} into the components day, month, hour, minute etc.

**Notes regarding read/wait cycle**

During the read cycle, the outputs \texttt{bRdySysTi} and \texttt{bRdyTiZInfo} change to FALSE, and the enumerator \texttt{eTimeZoneId} shows \texttt{0 = eTimeZoneID\_Unknown}. If the read operation was successful, the outputs switch back to TRUE or show the respective information for summer or winter time, if available. If the read operation was unsuccessful - internally the system waits for a response for 5 seconds - the outputs remain at FALSE or 0, and another wait cycle is started before the next read cycle. Although the internal RTC clock is not synchronized in the event of an error and may still show the right time, the time information may be wrong, and therefore also the UTC time. Errors during the read cycle will, any case, show up in \texttt{bErr} and \texttt{sErrMsg}. The countdown output \texttt{udiRemTiUpd_sec} is not restarted until the wait cycle starts.

**VAR_INPUT**

\begin{verbatim}
 bEn        : BOOL;
 sNetId     : T_AmsNetId;
 udiUpdRtc_sec : UDINT;
 bUpdRtc    : BOOL;
\end{verbatim}

\texttt{bEn:} Enables the function block. If \texttt{bEn = TRUE}, then the RTC clock is initialized with the NT system time.

\texttt{sNetId:} This parameter can be used to specify the AmsNetId (see \texttt{T_AmsNetId}) of the TwinCAT computer whose NT system time is to be read as timebase. If it is to be run on the local computer, an empty string can be entered.

\texttt{udiUpdRtc_sec:} Time specification [s] with which the RTC clock is regularly synchronized with the NT system time. Internally this value is limited to a minimum of 5 seconds, in order to ensure correct processing of the internal function blocks.

\texttt{bUpdRtc:} In parallel with the time \texttt{udiUpdRtc_sec}, the RTC clock can be synchronized via a positive edge at this input.
VAR_OUTPUT

bRdySysTi : BOOL;
bRdyTiZoInfo : BOOL;
bRdyRTC : BOOL;
udiRemTiUpd_sec : UDINT;
stSysTi : TIMESTRUCT;
stUTC : TIMESTRUCT;
dtSysTi : DT;
dtUTC : DT;
udiCurrentTime_ms : UDINT

eTiZId : E_TimeZoneID;
bErr : BOOL;
sErrDescr : T_MAXSTRING;

bRdySysTi: The system time was read successfully from the target system.

bRdyTiZoInfo: The additional time information (time zone, time shift relative to UTC and summer/winter
time) was read successfully.

bRdyRTC: This output is set if the function block has been initialized at least once. If this output is set, then
the values for date, time and milliseconds at the outputs are valid.

udiRemTiUpd_sec: Countdown to next synchronization/update of the time information.

stSysTi: System time of the read target system (see TIMESTRUCT). The time structure can be resolved
with the aid of the function block FB_BA_ExtTiSt [343] into its components: day, month, hour, minute etc.
Note: If the function block is not enabled (bEn=FALSE), the output stSysTi and its subelements (day month,
etc.) show 0.

stUTC: Coordinated world time (see TIMESTRUCT). This is determined internally from the system time and
the time information read from the target system. The time structure can be resolved with the aid of the
function block FB_BA_ExtTiSt [343] into its components: day, month, hour, minute etc. Note: If the function
block is not enabled (bEn=FALSE), the output stUTC and its subelements (day month, etc.) show 0.

dtSysTi / dtUTC: As stSysTi / stUTC, but in DATE-AND-TIME format: year-month-day-hours-minutes-
seconds. Note: If the function block is not enabled (bEn=FALSE), the outputs dtSysTi and dtUTC show
DT#1970-01-01-00:00, since this is the lower limit, which corresponds to the zeros in the structure
representation of stSysTi / stUTC.

udiCurrentTime_ms: Current time of day in ms.

eTiZId: Enumerator for summer/winter time information (see E_TimeZoneID).

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

sErrDescr: Contains the error description.

Error description

01: Warning: ADS error when reading the time (FB_NT_GetTime). The ADS error number is stated.

02: Warning: ADS error when reading the time zone information (FB_GetTimeZoneInformation). The ADS
error number is stated.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

FB_BA_SetTime
The function block \textit{FB_BA_SetTime} can be used to set the local NT system time and the date for a TwinCAT system (the local NT system time is shown in the taskbar). The system time is specified via the structure \textit{stSysTi}.

Internally, an instance of the function block \textit{NT_SetLocalTime} from the TcUtilities library is called in the function block.

The local NT system time can also be synchronized with a reference time with the aid of the SNTP protocol. For further information please refer to the Beckhoff Information System under: Beckhoff Information System > Embedded-PC > Operating systems > CE > SNTP: Simple Network Time Protocol.

\begin{verbatim}
VAR_INPUT
bSet : BOOL;
sNetId : T_AmsNetId;
stSysTi : TIMESTRUCT;
udiTiOut_sec : UDINT;

bSet: Activation of the function block with a rising edge.

sNetId: This parameter can be used to specify the AmsNetID of the TwinCAT computer, whose local NT system time is to be set. An empty string \textit{sNetId} := ""; can also be specified for the local computer (see \textit{T_AmsNetId}).

stSysTi: Structure with the new local NT system time (see TIMESTRUCT). If the time is not available as structure, it is advisable to use the function block \textit{FB_BA_CnvtTiSt} \cite{342}, which brings the subvariables of date and time in a structure together.

udiTiOut_sec: Indicates the timeout time [s], which must not be exceeded during execution.

VAR_OUTPUT
bBusy : BOOL;
bError : BOOL;
sErrorDescr : T_MAXSTRING;

bBusy: If the function block is activated via a rising edge at \textit{bSet}, this output is set and remains set until feedback occurs.

bErr: This output is set to TRUE, if either the system time to be transferred is incorrect or an ADS error occurs during the transfer.
\end{verbatim}
sErrDescr: Contains the error description.

<table>
<thead>
<tr>
<th>Error description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01: Error: Error: range exceeded year</td>
</tr>
<tr>
<td>02: Error: Error: range exceeded month</td>
</tr>
<tr>
<td>03: Error: Error: range exceeded day of the month</td>
</tr>
<tr>
<td>04: Error: Error: range exceeded hour</td>
</tr>
<tr>
<td>05: Error: Error: range exceeded minute</td>
</tr>
<tr>
<td>06: Error: Error: range exceeded second</td>
</tr>
<tr>
<td>07: Error: Error: range exceeded millisecond</td>
</tr>
<tr>
<td>08: Warning: An ADS error occurred while setting the time (FB NT_SetLocalTime). The ADS error number is stated.</td>
</tr>
</tbody>
</table>

Time specification limits

The time structure stUtcTi that was created is internally checked for limits (see TIMESTRUCT)

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**FB_BA_WrtPersistDat**

When activated, the function block `FB_BA_WrtPersistDat` first saves the persistent data in the Port_xxx.bootdata file. It is not necessary to explicitly specify the port or runtime system at which the PLC is located; this is determined internally. Once the data has been written, the content of the file Port_xxx.bootdata is copied to the backup file Port_xxx.bootdata-old. Thus both files are always synchronized. In case the original file with the persistent data is not readable, the backup copy, which is then read, contains the same data.

In any case, the checkmark "Clear Invalid Persistent Data" must be removed (see `FB_BA_WrtPersistDat`)

The function block can be started in two ways:

Via a positive edge at input bStt, if the function block is not in the set start-up phase.

Initially once the start-up phase is completed after a reset or TwinCAT restart. The duration is set at udiInitSttDly_sec in seconds. If "0" is entered there, the duration of the start-up phase is 0 and an initial execution of the function block is skipped.

No commands are accepted at bStt during the start-up phase.

If errors occur while reading, writing, opening or closing the files, this is indicated by a corresponding error message at bErr/sErrDescr. After an internally fixed waiting time of two seconds, the function block automatically attempts to execute the command (read, write, open or close) again.

It is therefore advisable to keep an eye on the error outputs or to evaluate them.
It is also important to note whether the backup file for the persistent data was loaded during the TwinCAT restart or after a reset. This indicates that the original file cannot be read and that the memory card of the controller is defective. It can be queried for each runtime system with the Boolean assignment of `TwinCAT_SystemInfoVarList._AppInfo.OldBootData` (see PlcAppSystemInfo).

Sample in ST:

```
PROGRAM Example_ST

VAR

  bOldData : BOOL;

END_VAR

bOldData:=TwinCAT_SystemInfoVarList._AppInfo.OldBootData;
```

Sample in CFC:

```
PROGRAM Example_CFC

VAR

  bOldData : BOOL;

END_VAR

TwinCAT_SystemInfoVarList._AppInfo.BootDataLoaded bOldData
```

### File handle conflict

Make sure that only this function block and only one instance of it accesses the persistent data. If several function blocks open a file and do not close it again, unforeseen file handle conflicts can occur which cannot be intercepted. The persistent data will then no longer be updated in the xxx.bootdata file.

### Description of persistent data handling under TwinCAT 3

TwinCAT saves the persistent data for each runtime system in a file during each orderly shutdown, i.e. when switching from Run to Config or Stop mode.

The file name consists of the ADS port name of the runtime system with the file extension `.bootdata`, e.g.: `Port_851.bootdata` and is stored in the TwinCAT directory under `TwinCAT\3.1\Boot\PLC`.

When the system is restarted, i.e. when switching to run mode, this file is read and then saved as `Port_xxx.bootdata-old`. If the file `Port_xxx.bootdata-old` already exists, it is overwritten.

The original file `Port_xxx.bootdata` then no longer exists. It is created again automatically when switching to Stop mode or by the function block `FB_WritePersistentData` from the TC2_Utilities library.

This behavior applies to each runtime system; each system has its own files with persistent data.

If the file is defective when the TwinCAT system is restarted, the system automatically accesses the backup file `Port_xxx.bootdata-old`. However, this behavior only applies if the Clear Invalid Persistent Data checkmark is unchecked in the runtime settings. If it is checked and the original file is defective, no data will be read.
The Port_xxx.bootdata-old backup file is also used when the controller is de-energized. In this case, too, the current persistent data is not stored in Port_xxx.bootdata. When the system is restarted, only the old data is available, unless a more up-to-date file was created by the FB_WritePersistentData function block before the system was switched off.
VAR_INPUT

bStt : BOOL;
udiInitSttDly_sec : UDINT;

bStt: A rising edge at this input starts the function block if it is not in the start-up phase.
udiInitSttDly_sec: Start-up phase after a reset or TwinCAT restart. The duration is set in seconds. Once the start-up phase has elapsed, the function block is automatically started once. No commands are accepted at bStt during the start-up phase. If "0" is set at udiInitSttDly_sec, the start-up phase is skipped. This input is preconfigured with 10 s.

**VAR OUTPUT**

bBusy: The function block is being executed.

udiRemTiInitSttDly_sec: Countdown of the set startup phase.

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

sErrDescr: Contains the error description.

### Error description

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Error: The number of the ADS port issued by the PLC is &quot;0&quot;</td>
</tr>
<tr>
<td>02</td>
<td>Warning: Error when writing the persistent data via the internal function block FB_WritePersistentData. Additionally its error number.</td>
</tr>
<tr>
<td>03</td>
<td>Warning: Error when opening the backup file (xxx.bootdata-old) via the internal function block FB_FileOpen. Additionally its error number.</td>
</tr>
<tr>
<td>04</td>
<td>Warning: Error when reading the original file (xxx.bootdata) via the internal function block FB_FileRead. Additionally its error number</td>
</tr>
<tr>
<td>05</td>
<td>Warning: Error when writing to the backup file (xxx.bootdata-old) via the internal function block FB_FileWrite. Additionally its error number.</td>
</tr>
<tr>
<td>06</td>
<td>Warning: Error when closing the original file (xxx.bootdata) via the internal function block FB_FileClose. Additionally its error number.</td>
</tr>
<tr>
<td>07</td>
<td>Warning: Error when closing the backup file (xxx.bootdata-old) via the internal function block FB_FileClose. Additionally its error number.</td>
</tr>
</tbody>
</table>

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**Universal**

**Actuators**

**Function blocks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_BA Actuator 3Point</td>
<td>Control of three-point dampers or valves</td>
</tr>
<tr>
<td>FB_BA Anlg3Pnt</td>
<td>Analog value for three-point converters</td>
</tr>
<tr>
<td>FB_BA AntBlkg</td>
<td>Blocking protection for pump or actuators</td>
</tr>
<tr>
<td>FB_BA Motor1St</td>
<td>Control of single-speed drives</td>
</tr>
<tr>
<td>FB_BA Motor2St</td>
<td>Control of two-speed drives</td>
</tr>
<tr>
<td>FB_BA PWM</td>
<td>Pulse width modulation function block</td>
</tr>
</tbody>
</table>
FB_BA_Actuator_3Point

The function block is used to control a 3-point actuator, e.g., a 3-point flap or a 3-point valve.

The command for opening the actuator is connected to output \textit{bOpen}.

The command for closing the actuator is connected to output \textit{bClose}.

In automatic mode (\textit{udiOpMode}=0) the control commands of \textit{bCmdOpen} and \textit{bCmdClose} are forwarded directly to the outputs \textit{bOpen} and \textit{bClose}.

The \textit{udiOpMode} input is used to determine the operating mode of the 3-point actuator:

\begin{itemize}
  \item 0 = Automatic
  \item 1 = Stop (\textit{bOpen} = \textit{bClose} = FALSE)
  \item 2 = Close
  \item 3 = Open
\end{itemize}

\textbf{VAR_INPUT}

\begin{itemize}
  \item \textit{bEn}: General enable of the function block.
  \item \textit{bAutoOpen}: Command to open the actuator.
  \item \textit{bAutoClose}: Command to close the actuator.
  \item \textit{udiOpMode}: Select operating mode (0 = Automatic, 1 = Stop (\textit{bOpen} = \textit{bClose} = FALSE), 2 = Close, 3 = Open)
\end{itemize}

\textbf{VAR_OUTPUT}

\begin{itemize}
  \item \textit{bOpen}: Open control output.
  \item \textit{bClose}: Close control output.
\end{itemize}

\textbf{Requirements}

\begin{table}[h]
\begin{tabular}{|l|l|}
\hline
\textbf{Development environment} & \textbf{Required PLC library} \\
\hline
TwinCAT from v3.1.4024.7 & Tc3_BA from v1.1.6.0 \\
\hline
\end{tabular}
\end{table}
The function block is intended for control of three-point actuators for valves or dampers.

A continuous control signal for positioning an actuator is converted into binary commands for opening and closing.

If the deviation between the set position value \( r_{In} \) and the calculated actual position value \( r_{Pos} \) of the actuator exceeds the set threshold value \( r_{Hys}/2 \), the function block starts to correct the position by switching the outputs \( b_{Opn} \) or \( b_{Cls} \), depending on the magnitude of the control deviation:

<table>
<thead>
<tr>
<th>Condition</th>
<th>( b_{Opn} )</th>
<th>( b_{Cls} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_{In} - r_{Pos} &gt; r_{Hys}/2 )</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>( r_{In} - r_{Pos} &lt; - r_{Hys}/2 )</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

If the function block reaches an end position \( r_{Out} = 0 \) or \( r_{Out} = 100 \) through a corresponding input value \( r_{In} \), the corresponding switching output remains permanently set in order to safely reach this end position at the valve or damper:

<table>
<thead>
<tr>
<th>Condition</th>
<th>( b_{Opn} )</th>
<th>( b_{Cls} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_{Out} = 0 )</td>
<td>FALSE</td>
<td>permanently TRUE</td>
</tr>
<tr>
<td>( r_{Out} = 100 )</td>
<td>permanently TRUE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Any deactivation of the continuous signal must be implemented by the user through external programming.

The input \( r_{In} \) is automatically limited to the range 0..100% internally.

This also applies to the entries \( r_{Hys} \) and \( r_{RefVal} \). The travel times \( udiTiCls_{ms} \) and \( udiTiOpn_{ms} \) both have a lower limit value of 10 (milliseconds).

A rising edge at \( b_{Ref} \) triggers a referencing command (the calculated actual position is set to \( r_{RefVal} \)).

If the drive has limit switches, they can be sampled directly via the digital input and used for referencing at \( b_{Ref} \).

**VAR_INPUT**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_{In} )</td>
<td>Setpoint for the actuator position [0 - 100%]. Internally limited to values between 0 and 100.</td>
</tr>
<tr>
<td>( r_{Hys} )</td>
<td>Hysteresis for the actuator position [0 - 100%]. Internally limited to values between 0 and 100.</td>
</tr>
<tr>
<td>( udiTiCls_{ms} )</td>
<td>Run time of the actuator from open to closed [ms]. Internally limited to values between 0 and 100.</td>
</tr>
<tr>
<td>( udiTiOpn_{ms} )</td>
<td>Run time of the actuator from closed to open [ms]. Internally limited to values between 0 and 100.</td>
</tr>
<tr>
<td>( b_{Ref} )</td>
<td>Edge references the internal position memory of the drive to value of ( r_{RefVal} ) [0 - 100%].</td>
</tr>
<tr>
<td>( r_{RefVal} )</td>
<td>Value to which the actual position is set upon a rising edge at ( b_{Ref} ).</td>
</tr>
<tr>
<td>( b_{CloseInit} )</td>
<td>Boolean indicating whether the actuator is closed initially.</td>
</tr>
</tbody>
</table>
rRefVal: Value for referencing the actuator with \texttt{bRef} [0 - 100\%]. Internally limited to values between 0 and 100.

bCloseInit: If this input is TRUE, output \texttt{bCls} is TRUE for the time \texttt{udiTiOpn_ms}

\begin{verbatim}
VAR_OUTPUT
bCls  : BOOL;
bOpn  : BOOL;
rPos  : REAL;
\end{verbatim}

\textbf{bCls}: Output for closing the actuator.

\textbf{bOpn}: Output for opening the actuator.

\textbf{rPos}: Current calculated actuator position [0 - 100\%].

\textbf{Requirements}

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
Development environment & Required PLC library \\
\hline
TwinCAT from v3.1.4024.7 & Tc3_BA from v1.1.6.0 \\
\hline
\end{tabular}
\end{table}

\textbf{FB_BA_AntBlkg}

This function block prevents blocking of pumps or actuators after prolonged idle periods by issuing a switch-on pulse.

The maximum idle period before such a pulse is issued is determined by the value of \texttt{udiTiOffMin_sec}. For logging the idle time, the input \texttt{bFdb} must be linked to the operating feedback from the unit. The length of the pulse is parameterized with \texttt{udiTiImpLntg_sec}. The input \texttt{bExe} should be used if the blocking protection pulses are to be issued cyclically based on a switching schedule, rather than depending on the idle times. A rising edge at \texttt{bExe} immediately triggers output of a pulse to \texttt{bQ}. Generally, a pulse output only occurs if the function block at \texttt{bEn} is enabled.

\begin{verbatim}
VAR_INPUT
bEn     : BOOL;
bFdb    : BOOL;
bExe    : BOOL;
udiTiOffMin_sec : UDINT;
udiTiImpLntg_sec : UDINT;
\end{verbatim}

\textbf{bEn}: Enable function block.

\textbf{bFdb}: Input for connecting the feedback signal of a motor or valve.

\textbf{bExe}: Rising edge forces a pulse output.

\textbf{udiTiOffMin_sec}: Minimum switch-off time [s]: a pulse is issued once the time \texttt{udiTiOffMin_sec} has elapsed without movement of the unit.

\textbf{udiTiImpLntg_sec}: Length of the blocking protection pulse [s] at \texttt{bQ}.

\begin{verbatim}
VAR_OUTPUT
bQ         : BOOL;
udiTiOffMin_sec : UDINT;
udiTiImpLntg_sec : UDINT;
\end{verbatim}
bQ: Pulse output.

udiRTIffMin_sec: Remaining time [s] before the next pulse is issued in the absence of movement.

udiTIImpLntgLongt_sec: Remaining residual time [s] of the pulse at bQ.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**FB_BA_Motor1St**

Function block for controlling a simple single-stage motor.

The input bEn is used for general enabling of the motor.

The input udiOpMode is used to set the operating mode of the motor:

- 0 = Automatic
- 1 = Manual off
- 2 = Manual on

In automatic mode (udiOpMode= 0) the motor can be operated via the input bAuto (bAuto = bQ = TRUE).

The collection of all possible malfunctions of a motor is connected to bDst.

**VAR_INPUT**

- bEn: Enable motor.
- bAuto: Request of the actuator in automatic mode (udiOpMode = 0).
- bDst: Input for collecting the possible motor malfunctions.
- udiOpMode: Select the operating mode (0 = Automatic, 1 = Manual off, 2 = Manual on).

**VAR_OUTPUT**

- bQ: Control output.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
FB_BA_Motor2St

Function block for controlling a simple two-stage motor.

The input \textit{bEn} is used for general enabling of the motor.

The input \textit{udiOpMode} is used to set the operating mode of the motor:

- 0 = Automatic
- 1 = Manual off
- 2 = Manual stage 1
- 3 = Manual stage 2

In automatic mode (\text{udiOpMode} = 0) the desired stage can be set via the inputs \textit{bAutoSt1} (stage 1) and \textit{bAutoSt2} (stage 2).

The collection of all possible malfunctions of a motor is connected to \textit{bDst}.

\begin{verbatim}
VAR_INPUT
  bEn      : BOOL;
  bAutoSt1 : BOOL;
  bAutoSt2 : BOOL;
  bDst     : BOOL;
  udiOpMode : UDINT;
\end{verbatim}

\textit{bEn}: Enable motor.

\textit{bAutoSt1}: Request of the actuator at stage 1 in automatic mode (\textit{udiOpMode} = 0).

\textit{bAutoSt2}: Request of the actuator at stage 2 in automatic mode (\textit{udiOpMode} = 0).

\textit{bDst}: Input for collecting the possible motor malfunctions.

\textit{udiOpMode}: Select the operating mode (0 = Automatic, 1 = Manual off, 2 = Manual stage 1, 3 = Manual stage 2).

\begin{verbatim}
VAR_OUTPUT
  bQ1      : BOOL;
  bQ2      : BOOL;
\end{verbatim}

\textit{bQ1}: Control output stage 1.

\textit{bQ2}: Control output stage 2.

Requirements

\begin{tabular}{|l|l|}
\hline
Development environment & Required PLC library \\
\hline
TwinCAT from v3.1.4024.7 & Tc3_BA from v1.1.6.0 \\
\hline
\end{tabular}
FB_BA_PWM

The function block calculates switch-on and switch-off times \( rActTiOn\_sec \) and \( rActTiOff\_sec \) [s] from an analog input signal \( rIn \) (0..100\%, internally limited) and the period \( udiPrd\_sec \) [s].

The following relationships apply:

- 100\% at the input of a switch-on time \( rActTiOn\_sec \) of the total period \( udiPrd\_sec \) and a switch-off time \( rActTiOff\_sec \) of 0 s
- 0\% at the input of a switch-on time \( rActTiOn\_sec \) of 0 s and a switch-off time \( rActTiOff\_sec \) of the total period \( udiPrd\_sec \).

In addition, \( udiMinSwiTi\_sec \) [s] can be used to set a lower limit for the switching time, in order to prevent damage to drives caused by too short actuating pulses. This behavior is only valid for \( 0>rIn>100! \)

If \( rIn=0 \) or 100, the output \( bQ \) remains deleted or set. After the period time has elapsed, the current input signal is evaluated again. If it is still set to 0 or 100, there is no change of state of \( bQ \).

Switching characteristics

1. A FALSE signal at input \( bEn \) disables the function block and sets \( bQ \) to FALSE. Only the switch-on and switch-off times are continuously calculated and displayed at the outputs \( rActTiOn\_sec/rActTiOff\_sec \) [s].
2. A rising edge at input \( bEn \) enables the function block: It will initially jump to a decision step. Depending on the previous state of the switching output \( bQ \), the switching step is now accessed. However, if the input \( rIn \) is set to 0, an immediate jump occurs to the Off step (\( bQ=FALSE \)), or to the On step if \( rIn=100 \) (\( bQ=TRUE \)), irrespective of the previous state of \( bQ \). The minimum switching time is deactivated for these two cases.
3. A countdown timer with the current calculated starting value runs in the respective active step (ON or OFF), which is based on the pulse/pause ratio. The on- or off-step is completed with the calculated time, irrespective of whether the pulse/pause ratio changes in the meantime. The respective countdown is displayed at the outputs \( udiRemTiOn\_sec/udiRemTiOff\_sec \) in full seconds.
4. Completion of the on- or off-step is followed by a jump back to the decision step (point 2).

**VAR_INPUT**

- \( bEn \): Activation of pulse width modulation.
- \( rIn \): Input signal, internally limited to 0..100\%.
- \( udiPrd\_sec \): Period time[s]. Internally limited to a minimum value of 0.
- \( udiMinSwiTi\_sec \): Minimum switch-on time [s], to avoid too short pulses. Internally limited to values between 0 and \( udiPrd\_sec \).

**VAR_OUTPUT**

- \( bQ \): Boolean value.
- \( bLmtSwiTi \): Boolean value.
- \( rActTiOn\_sec \): REAL;
bQ: PWM output.

bLmtSwiTi: Information output to indicate that the input signal is so low that the minimum switch-on time is used as limit.

rActTiOn_sec: Information output: Calculated switch-on time.

rActTiOff_sec: Information output: Calculated switch-off time.

udiRemTiOn_sec: Switch-on timer countdown.

udiRemTiOff_sec: Switch-off timer countdown.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

Analog inputs/outputs

Function blocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_BA_AI [358]</td>
<td>Acquisition of analog input signals</td>
</tr>
<tr>
<td>FB_BA_AO [361]</td>
<td>Control of analog actuators with integrated scaling function</td>
</tr>
<tr>
<td>FB_BA_KL32xxConfig</td>
<td>Parameterization of the connected sensor type on an input channel from the PLC</td>
</tr>
</tbody>
</table>

FB_BA_AI

The function block is used for measured data processing and terminal configuration of all standard K-bus analog input terminals.
Terminal configuration

The first step when using this function block is to select the corresponding terminal type eTerminal. Correct functioning of the function block is only guaranteed if the terminal selected with the eTerminal variable matches the terminal actually inserted and linked.

With the terminals for resistance temperature measurement of type KL3208_0010 and KL320x_0000, the temperature sensor used at the terminal input is additionally selected with the enumeration eSensor.

A rising edge at bConfigure writes the terminal settings to the terminal using the variables TO_usCtrl and TO_iDataOut.

Measured value scaling

At the analog input terminals of types: KL300x, KL306x, KL3132_0000, KL3162_0000, KL3172_0000, KL3172_0500, KL3172_1000, KL3182_0000, KL3404, KL3464, KL3408, KL3468, the raw value TI_iDataIn of the terminal is scaled by the internal function block FB_BA_Chrcct02 [396]. (See diagram). The parameterization of the scaling function is carried out using parameters X(01/02) and Y(01/02). A signal offset that may be required for measured value correction can be parameterized with the variable rOffset. rSmoothingFactor attenuates the scaled measurement signal, including the offset.

Signal flow:
With the temperature measurement terminals of types KL3208_0010 and KL320x_0000, the measuring signal is not scaled within the function block, but directly in the terminal. Attenuation of excessively fluctuating measuring signals takes place with the factor $\text{rSmoothingFactor}$. If an error is detected at the terminal, such as $\text{bShortCircuit}$ or $\text{bWireBreak}$, the last valid value is output until the error is eliminated at the output of the function block.

Signal flow:

Measuring range monitoring

The variables $\text{rHighLimit}$ and $\text{rLowLimit}$ are used to monitor the measured value $\text{rVal}$ at the output of the function block for compliance with a valid range. If the measured value $\text{rVal}$ is above $\text{rHighLimit}$ or below $\text{rLowLimit}$, this is displayed at the outputs $\text{bHighLimit}$ or $\text{bLowLimit}$. Measuring range monitoring is deactivated if the input variable $\text{bEnLimitCtrl}$ is FALSE or if an error (e.g. short circuit) is detected by means of the terminal status $\text{TI_usiState}$.

The response of the measuring range monitoring can be delayed by the variable $\text{udiDelayLimitCtrl_sec}$.

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{TI_usiState}$ : USINT;</td>
</tr>
<tr>
<td>$\text{TI_iDataIn}$ : INT;</td>
</tr>
<tr>
<td>$\text{eTerminal}$ : E_BA_TERMINAL_KL;</td>
</tr>
<tr>
<td>$\text{eSensor}$ : E_BA_SENSOR;</td>
</tr>
<tr>
<td>$\text{bConfigure}$ : BOOL;</td>
</tr>
<tr>
<td>$\text{rX01}$ : REAL;</td>
</tr>
<tr>
<td>$\text{rX02}$ : REAL;</td>
</tr>
<tr>
<td>$\text{rY01}$ : REAL;</td>
</tr>
<tr>
<td>$\text{rY02}$ : REAL;</td>
</tr>
<tr>
<td>$\text{rOffset}$ : REAL;</td>
</tr>
<tr>
<td>$\text{rSmoothingFactor}$ : REAL;</td>
</tr>
<tr>
<td>$\text{udiDecimalPlace}$ : UDINT(1..6);</td>
</tr>
<tr>
<td>$\text{bEnLimitCtrl}$ : BOOL;</td>
</tr>
<tr>
<td>$\text{rHighLimit}$ : REAL;</td>
</tr>
<tr>
<td>$\text{rLowLimit}$ : REAL;</td>
</tr>
<tr>
<td>$\text{udiDelayLimitCtrl_sec}$ : UDINT;</td>
</tr>
</tbody>
</table>

$\text{TI_usiState}$: Linking with the corresponding status byte of the Bus Terminal in the I/O area of the program.

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

$\text{eTerminal}$: Selection of the respective Bus Terminal (see E_BA_Terminal_KL).

$\text{eSensor}$: Selection of the sensor type (see E_BA_Sensor).

$\text{bConfigure}$: A rising edge starts the configuration of the Bus Terminal.

$\text{rX01}$: x-value for the interpolation point P1.

$\text{rX02}$: x-value for the interpolation point P2.

$\text{rY01}$: y-value for the interpolation point P1.

$\text{rY02}$: y-value for the interpolation point P2.

$\text{rOffset}$: Offset.

$\text{rSmoothingFactor}$: attenuation factor. Internally limited to between 1 and 10000.

$\text{udiDecimalPlace}$: Specifies the decimal places for the value $\text{rVal}$. Preset to 1.

$\text{bEnLimitCtrl}$: Enable limit value monitoring.
**rHighLimit:** Upper limit value.

**rLowLimit:** Lower limit value.

**udiDelayLimitCtrl_sec:** Time delay until limit value monitoring is activated.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>TO_usiCtrl</td>
<td>USINT;</td>
</tr>
<tr>
<td>TO_iDataOut</td>
<td>INT;</td>
</tr>
<tr>
<td>rVal</td>
<td>REAL;</td>
</tr>
<tr>
<td>bWireBreak</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bShortCircuit</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bHighLimit</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bLowLimit</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bErr</td>
<td>BOOL;</td>
</tr>
<tr>
<td>sErrDescr</td>
<td>T_MAXSTRING;</td>
</tr>
</tbody>
</table>

**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.

**rVal:** Scaled output value.

**bWireBreak:** Broken wire at the sensor.

**bShortCircuit:** Short-circuit at the sensor.

**bHighLimit:** Upper limit value exceeded.

**bLowLimit:** Value below lower limit.

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

**sErrDescr:** Contains the error description.

**Error description**

<table>
<thead>
<tr>
<th>Error ID</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Error: Incorrect scaling parameter $rX01/rX02/rY01/rY02$</td>
</tr>
<tr>
<td>02</td>
<td>Error: Check the terminal configuration KL32xx eTerminal/eSensor/TI_usiState/TI_iDataIn/TO_usiCtrl/IO_iDataOut</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**FB_BA_AO**

The function block **FB_BA_AO** is used for output and scaling of an analog output. The input value at $rX$ is converted into an output value from 0 to 32767 or 0-10 Volt or 4-20 mA by means of a linear equation.
The operating mode is set via the input \textit{udiOpMod}:

- \texttt{0} = Automatic
- \texttt{1} = Manual

In automatic mode (\textit{udiOpMod}= 0) the input value \texttt{rX} is passed on.

In manual mode (\textit{udiOpMod}= 1) the input value \texttt{rXManual} is passed on.

\textbf{VAR_INPUT}
\begin{verbatim}
rX         : REAL;
rX01       : REAL;
rX02       : REAL;
rY01       : REAL;
rY02       : REAL;
udiOpMod   : UDINT(0..1);
rXManual   : REAL;
\end{verbatim}

\texttt{rX}: Input value of the process in automatic mode.

\texttt{rX01}: x-value for the interpolation point P1.

\texttt{rX02}: x-value for the interpolation point P2.

\texttt{rY01}: y-value for the interpolation point P1.

\texttt{rY02}: y-value for the interpolation point P2.

\texttt{udiOpMod}: Selection of the operating mode (0 = Automatic, 1 = Manual).

\texttt{rXManual}: Input value for manual operation.

\textbf{VAR_OUTPUT}
\begin{verbatim}
rY         : REAL;
TO_iY      : INT;
\end{verbatim}
**rY**: Output signal as floating point number.

**TQ_iY**: Output signal as integer value for linking to the output value of the bus terminal in the I/O section of the program.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

### General control functions

**Function blocks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_BA_DMUX_XX [363]</td>
<td>Demultiplexer function blocks</td>
</tr>
<tr>
<td>FB_BA_MMUX_XX [366]</td>
<td>The function blocks activate an input value on the output, depending on a selector and the corresponding input selector condition</td>
</tr>
<tr>
<td>FB_BA_MUX_XX [369]</td>
<td>Multiplexer function blocks</td>
</tr>
<tr>
<td>FB_BA_PrioSwi_XX [371]</td>
<td>Priority switch</td>
</tr>
<tr>
<td>FB_BA_Blink [373]</td>
<td>Simple oscillator function block</td>
</tr>
<tr>
<td>FB_BA_FIFO04 [374]</td>
<td>Sequential control of up to four units</td>
</tr>
<tr>
<td>FB_BA_FIFO08 [376]</td>
<td>Sequential control of up to eight units</td>
</tr>
<tr>
<td>FB_BA_StepCtrl08 [378]</td>
<td>Step sequence function block, 8 steps</td>
</tr>
<tr>
<td>FB_BA_StepCtrl12 [381]</td>
<td>Step sequence function block, 12 steps</td>
</tr>
<tr>
<td>FB_BA_FIFO04_XX [384]</td>
<td></td>
</tr>
<tr>
<td>FB_BA_FIFO08_XX [385]</td>
<td></td>
</tr>
</tbody>
</table>

#### FB_BA_DMUX_XX

```
FB_BA_DMUX_LR16

- bEn: BOOL
- uSel: UDIINT
- IrIn: LREAL
- IrQ0, IrQ1, ..., IrQ16: LREAL
```
Demultiplexer function blocks exist for different variable types (BOOL, INT, LREAL, REAL, USINT, UINT, UDINT and DINT) and in different output parameters (4, 8, 12 and 16), but they all have the same functionality.

The function block FB_BA_DMUX_LR16 is described as an example.

In active state \((bEn=\text{TRUE})\), the function block outputs the value at input \(lrIn\) at the output \((lrQ01..lrQ16)\) whose number is entered at input \(udiSel\). All other outputs are set to 0 (for boolean demultiplexers to FALSE).

Example:

\[
\begin{array}{|c|c|}
\hline
\text{Inputs} & \text{Outputs} \\
\hline
bEn = \text{TRUE} & lrQ01 = 0.0 \\
udiSel = 5 & lrQ02 = 0.0 \\
lrIn = 32.5 & lrQ03 = 0.0 \\
& lrQ04 = 0.0 \\
& lrQ05 = 32.5 \\
& lrQ06 = 0.0 \\
& lrQ07 = 0.0 \\
& lrQ08 = 0.0 \\
& lrQ09 = 0.0 \\
& lrQ10 = 0.0 \\
& lrQ11 = 0.0 \\
& lrQ12 = 0.0 \\
& lrQ13 = 0.0 \\
& lrQ14 = 0.0 \\
& lrQ15 = 0.0 \\
& lrQ16 = 0.0 \\
\hline
\end{array}
\]

If the value entered at \(udiSel\) is greater than the number of outputs, the value of \(lrIn\) is output at the "highest" output:

\[
\begin{array}{|c|c|}
\hline
\text{Inputs} & \text{Outputs} \\
\hline
bEn = \text{TRUE} & lrQ01 = 0.0 \\
udiSel = 25 & lrQ02 = 0.0 \\
lrIn = 32.5 & lrQ03 = 0.0 \\
& lrQ04 = 0.0 \\
& lrQ05 = 0.0 \\
& lrQ06 = 0.0 \\
& lrQ07 = 0.0 \\
& lrQ08 = 0.0 \\
& lrQ09 = 0.0 \\
& lrQ10 = 0.0 \\
& lrQ11 = 0.0 \\
& lrQ12 = 0.0 \\
& lrQ13 = 0.0 \\
& lrQ14 = 0.0 \\
& lrQ15 = 0.0 \\
& lrQ16 = 32.5 \\
\hline
\end{array}
\]

If \(bEn = \text{FALSE}\), 0.0 is output at all outputs, or FALSE for boolean demultiplexers.
VAR_INPUT

bEn : BOOL;
udiSel : UDINT;
rIn : LREAL;

bEn: Activation of the block function.

udiSel: Number of the output (lrQ00...lrQ16), which is to take on the value of input lrIn.

lrIn: Value to be output.

VAR_OUTPUT

lrQ00 : LREAL;
lrQ01 : LREAL;
lrQ02 : LREAL;
lrQ03 : LREAL;
lrQ04 : LREAL;
lrQ05 : LREAL;
lrQ06 : LREAL;
lrQ07 : LREAL;
lrQ08 : LREAL;
lrQ09 : LREAL;
lrQ10 : LREAL;
lrQ11 : LREAL;
lrQ12 : LREAL;
lrQ13 : LREAL;
lrQ14 : LREAL;
lrQ15 : LREAL;
lrQ16 : LREAL;

lrQ00...lrQ16: Value outputs.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
The function block activates an input value on the output, depending on a selector and the corresponding input selector condition.

Multiplexer function blocks exist for different variable types (BOOL, INT, LREAL, REAL, USINT, UINT, UDINT and DINT) and in different input parameters (4, 8, 12, 16 and 24), but they all have the same functionality.

The function block FB_BA_MMUX_R16 is described as an example.

In active state \( bEn = \text{TRUE} \), the function block activates one of the input values \( rVal_{xx} \) at output \( rVal \), depending on a selector \( udiSel \) and the corresponding input selector condition \( udiEn_{xx} \).

If several input selector conditions \( udiEn_{01} \ldots udiEn_{16} \) are identical and the selector \( udiSel \) matches a condition, the input value \( rVal_{01} \ldots rVal_{16} \) of the lowest active selector condition is activated at output \( rVal \). 

\( udiEn_{01} \) is the lowest selector condition, \( udiEn_{16} \) the highest.

The output variable \( bQ \) indicates that the selector \( udiSel \) matches the input selector condition \( udiEn_{xx} \).

The output variable \( udiActvPrio \) indicates the active selector condition.
If no selector condition is active, \( rReplVal \) is output at \( rVal \). \( bQ \) is then FALSE and \( udiActvPrio \) shows 255.

Example:

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>Value</td>
</tr>
<tr>
<td>bEn</td>
<td>TRUE</td>
</tr>
<tr>
<td>udiSel</td>
<td>5</td>
</tr>
<tr>
<td>udiEn01</td>
<td>4</td>
</tr>
<tr>
<td>rVal01</td>
<td>123</td>
</tr>
<tr>
<td>udiEn02</td>
<td></td>
</tr>
<tr>
<td>rVal02</td>
<td></td>
</tr>
<tr>
<td>udiEn03</td>
<td>3</td>
</tr>
<tr>
<td>rVal03</td>
<td>321</td>
</tr>
<tr>
<td>udiEn04</td>
<td></td>
</tr>
<tr>
<td>rVal04</td>
<td></td>
</tr>
<tr>
<td>udiEn05</td>
<td>8</td>
</tr>
<tr>
<td>rVal05</td>
<td>345</td>
</tr>
<tr>
<td>udiEn06</td>
<td></td>
</tr>
<tr>
<td>rVal06</td>
<td></td>
</tr>
<tr>
<td>udiEn07</td>
<td>5</td>
</tr>
<tr>
<td>rVal07</td>
<td>1.123</td>
</tr>
<tr>
<td>udiEn08</td>
<td></td>
</tr>
<tr>
<td>rVal08</td>
<td></td>
</tr>
<tr>
<td>udiEn09</td>
<td>5</td>
</tr>
<tr>
<td>rVal09</td>
<td>5.4321</td>
</tr>
<tr>
<td>udiEn10</td>
<td></td>
</tr>
<tr>
<td>rVal10</td>
<td></td>
</tr>
<tr>
<td>udiEn11</td>
<td></td>
</tr>
<tr>
<td>rVal11</td>
<td></td>
</tr>
<tr>
<td>udiEn12</td>
<td></td>
</tr>
<tr>
<td>rVal12</td>
<td></td>
</tr>
<tr>
<td>udiEn13</td>
<td></td>
</tr>
<tr>
<td>rVal13</td>
<td></td>
</tr>
<tr>
<td>udiEn14</td>
<td></td>
</tr>
<tr>
<td>rVal14</td>
<td></td>
</tr>
<tr>
<td>udiEn15</td>
<td></td>
</tr>
<tr>
<td>rVal15</td>
<td></td>
</tr>
<tr>
<td>udiEn16</td>
<td></td>
</tr>
<tr>
<td>rVal16</td>
<td></td>
</tr>
<tr>
<td>rReplVal</td>
<td></td>
</tr>
</tbody>
</table>

If no priority is active, the value of the global constant \( \text{Const.udiNoActvPrio} \) is output at \( udiActvPrio \).

VAR_INPUT

\[
\begin{align*}
\text{bEn} & : \text{BOOL}; \\
\text{udiSel} & : \text{UDINT}; \\
\text{udiEn01} & : \text{UDINT} := \text{Const.udiNoActvPrio}; \\
\text{rVal01} & : \text{REAL}; \\
\text{udiEn02} & : \text{UDINT} := \text{Const.udiNoActvPrio}; \\
\text{rVal02} & : \text{REAL}; \\
\text{udiEn03} & : \text{UDINT} := \text{Const.udiNoActvPrio}; \\
\text{rVal03} & : \text{REAL}; \\
\text{udiEn04} & : \text{UDINT} := \text{Const.udiNoActvPrio}; \\
\end{align*}
\]
rVal04 : REAL;
udiEn05 : UDINT := Const.udiNoActvPrio;
rVal05 : REAL;
udiEn06 : UDINT := Const.udiNoActvPrio;
rVal06 : REAL;
udiEn07 : UDINT := Const.udiNoActvPrio;
rVal07 : REAL;
udiEn08 : UDINT := Const.udiNoActvPrio;
rVal08 : REAL;
udiEn09 : UDINT := Const.udiNoActvPrio;
rVal09 : REAL;
udiEn10 : UDINT := Const.udiNoActvPrio;
rVal10 : REAL;
udiEn11 : UDINT := Const.udiNoActvPrio;
rVal11 : REAL;
udiEn12 : UDINT := Const.udiNoActvPrio;
rVal12 : REAL;
udiEn13 : UDINT := Const.udiNoActvPrio;
rVal13 : REAL;
udiEn14 : UDINT := Const.udiNoActvPrio;
rVal14 : REAL;
udiEn15 : UDINT := Const.udiNoActvPrio;
rVal15 : REAL;
udiEn16 : UDINT := Const.udiNoActvPrio;
rVal16 : REAL;
rReplVal : REAL;

bEn: Activation of the block function.

udiSel: Selector. Internally limited to values between 0 and 4294967294.

udiEn01..udiEn16: Input selector condition.
The input variables are pre-initialized to the value 255.

rVal01...rVal16: Input values to select from.

rReplVal: Substitute value, if no input selector condition is active

VAR_OUTPUT

bQ : BOOL;
rVal : REAL;
udiActvPrio : UDINT;

bQ: TRUE, if the selector udiSel matches an input selector condition udiEnxx.

rVal: Value of the selected input selector condition.

udiActvPrio: Indicates which input selector condition is active. If no priority is active, the value of the global constant Const.udiNoActvPrio [419] is output at udiActvPrio.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
Multiplexer function blocks exist for different variable types (BOOL, INT, LREAL, REAL, USINT, UINT, UDINT and DINT) and in different input parameters (4, 8, 12 and 16), but they all have the same functionality.

The function block FB_BA_MUX_LR16 is described as an example.

In active state \( bEn = \text{TRUE} \), the function block outputs the input value \( lrIn01..lrIn16 \) at output \( lrQ \), whose number is entered at input \( udiSel \).

Example:
If the value entered at `udiSel` is greater than the number of inputs, the "highest-ranking" input is output at `lrQ`:

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEn = TRUE</td>
<td>lrQ = 16.5</td>
</tr>
<tr>
<td>udiSel = 5</td>
<td></td>
</tr>
<tr>
<td>lrIn01 = 15.9</td>
<td></td>
</tr>
<tr>
<td>lrIn02 = 32.5</td>
<td></td>
</tr>
<tr>
<td>lrIn03 = 17.4</td>
<td></td>
</tr>
<tr>
<td>lrIn04 = 5.84</td>
<td></td>
</tr>
<tr>
<td>lrIn05 = 9.56</td>
<td></td>
</tr>
<tr>
<td>lrIn06 = 16.5</td>
<td></td>
</tr>
<tr>
<td>lrIn07 = 32.781</td>
<td></td>
</tr>
<tr>
<td>lrIn08 = 25.4</td>
<td></td>
</tr>
<tr>
<td>lrIn09 = 44.5</td>
<td></td>
</tr>
<tr>
<td>lrIn10 = 66.1</td>
<td></td>
</tr>
<tr>
<td>lrIn11 = 45.5</td>
<td></td>
</tr>
<tr>
<td>lrIn12 = 83.3</td>
<td></td>
</tr>
<tr>
<td>lrIn13 = 54.56</td>
<td></td>
</tr>
<tr>
<td>lrIn14 = 33.8</td>
<td></td>
</tr>
<tr>
<td>lrIn15 = 98.5</td>
<td></td>
</tr>
<tr>
<td>lrIn16 = 71.3</td>
<td></td>
</tr>
</tbody>
</table>

If `bEn` = FALSE, 0.0 is output at `lrQ`, or FALSE for boolean multiplexers.

**VAR INPUT**

```plaintext
VAR_INPUT
bEn : BOOL;
udiSel : UDINT;
lrIn00 : LREAL;
lrIn01 : LREAL;
lrIn02 : LREAL;
lrIn03 : LREAL;
lrIn04 : LREAL;
lrIn05 : LREAL;
lrIn06 : LREAL;
lrIn07 : LREAL;
lrIn08 : LREAL;
lrIn09 : LREAL;
lrIn10 : LREAL;
lrIn11 : LREAL;
lrIn12 : LREAL;
lrIn13 : LREAL;
lrIn14 : LREAL;
lrIn15 : LREAL;
lrIn16 : LREAL;
```
bEn: Activation of the block function.

udiSel: Number of the input, whose value is to be output at lrQ.

lr00...lr16: Input values to select from.

VAR_OUTPUT

lrQ : LREAL;

lrQ: Value of the selected input.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

FB_BA_PrioSwi_XX

The priority switches exist for different variable types (BOOL, INT, LREAL, REAL, USINT, UINT, UDINT and DINT) and in different output sizes (4, 8, 12 and 16 or 24), but they all have the same functionality. The function block FB_BA_PrioSwi_LR08 is described as an example.

Priority switches are available for selecting different values. At output lrVal the value with the highest priority is applied whose input bEnxx is TRUE.

Example:
If none of the priorities is enabled, the output bQ switches to FALSE. 0 is output at lrVal and udiActvPrio. For a boolean priority switch, FALSE is then output at bVal.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEn01</td>
<td>FALSE</td>
</tr>
<tr>
<td>lrVal01</td>
<td>32.5</td>
</tr>
<tr>
<td>bEn02</td>
<td>FALSE</td>
</tr>
<tr>
<td>lrVal02</td>
<td>17.4</td>
</tr>
<tr>
<td>bEn03</td>
<td>TRUE</td>
</tr>
<tr>
<td>lrVal03</td>
<td>5.84</td>
</tr>
<tr>
<td>bEn04</td>
<td>TRUE</td>
</tr>
<tr>
<td>lrVal04</td>
<td>9.56</td>
</tr>
<tr>
<td>bEn05</td>
<td>FALSE</td>
</tr>
<tr>
<td>lrVal05</td>
<td>16.5</td>
</tr>
<tr>
<td>bEn06</td>
<td>TRUE</td>
</tr>
<tr>
<td>lrVal06</td>
<td>32.781</td>
</tr>
<tr>
<td>bEn07</td>
<td>FALSE</td>
</tr>
<tr>
<td>lrVal07</td>
<td>25.4</td>
</tr>
<tr>
<td>bEn08</td>
<td>TRUE</td>
</tr>
<tr>
<td>lrVal08</td>
<td>44.5</td>
</tr>
</tbody>
</table>

If no priority is active, the value of the global constant ConstudiNoActvPrio is output at udiActvPrio.

VAR_INPUT

bEn01 : BOOL;
lrVal01 : LREAL;
bEn02 : BOOL;
lrVal02 : LREAL;
bEn03 : BOOL;
lrVal03 : LREAL;
bEn04 : BOOL;
lrVal04 : LREAL;
bEn05 : BOOL;
lrVal05 : LREAL;
bEn06 : BOOL;
lrVal06 : LREAL;
bEn07 : BOOL;
Programming

lrVal07 : LREAL;
bEn08 : BOOL;
lrVal08 : LREAL;

bEn01...bEn08: Enabling the priority value.
lrVal01...lrVal08: Priority value.

VAR_OUTPUT

bQ : BOOL;
lrVal : LREAL;
udiActvPrio : UDINT;

bQ: Output to indicate whether a priority is enabled.
lrVal: Output of the value of the current (highest) priority that is enabled.
udiActvPrio: Current (highest) priority that is enabled.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

FB_BA_Blink

This function block is an oscillator with adjustable pulse and pause time, udiTiOn_ms and udiTiOff_ms [ms]. It is enabled with a TRUE signal at bEn and starts with the pulse phase.

udiTiNextSwi_sec is a countdown [s] to the next change of bQ.

VAR_INPUT

bEn : BOOL;
udiTiOn_ms : UDINT;
udiTiOff_ms : UDINT;

bEn: Function block enable.
udiTiOn_ms: pulse time [ms].
udiTiOff_ms: pause time [ms].
VAR_OUTPUT

_VAR_OUTPUT_

<table>
<thead>
<tr>
<th>bQ</th>
<th>BOOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>udiTiNextSwi_sec</td>
<td>UDINT</td>
</tr>
</tbody>
</table>

bQ: Oscillator output.

udiTiNextSwi_sec: Countdown to next change of bQ [s].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

FB_BA_FIFO04

The function block FB_BA_FIFO04 enables sequential control of up to four units, with automatic switching of the switch-on sequence based on operating hours.

The function block is available in two versions: for a sequence of four or eight units.

Units with fewer operating hours take precedence in the sequence over units with more operating hours.

A rising edge at bChg forces a sequence change. The units with the fewest operating hours are set to the top of the FIFO and thus given priority for switching on.

In the sequence only units that are enabled at inputs bEn01..bEn04. udiNum indicates the number of requested units.

The operating hours of the units are entered at inputs udiActvTi01_h to udiActvTi04_h. If all these inputs are set to a constant value of zero, the sequence change is controlled cyclically, depending on bChg.

The first unit is removed from the FIFO, the other units are advanced, and the first unit is appended at the end of the FIFO again. As a result is an alternating sequence of units.

If more units are requested at input udiNum than are available at inputs bEn01 to bEn04, this is indicated with TRUE at bErr.

Error handling

If more units are requested at input udiNum than are available at inputs bEn01 to bEn04, this is indicated with TRUE at bErr.

VAR_INPUT

_VAR_INPUT_

<table>
<thead>
<tr>
<th>bEn</th>
<th>BOOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>udiNum</td>
<td>UDINT</td>
</tr>
<tr>
<td>bChg</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEn01</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEn02</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEn03</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEn04</td>
<td>BOOL</td>
</tr>
<tr>
<td>udiActvTi01_h</td>
<td>UDINT</td>
</tr>
<tr>
<td>udiActvTi02_h</td>
<td>UDINT</td>
</tr>
<tr>
<td>udiActvTi03_h</td>
<td>UDINT</td>
</tr>
<tr>
<td>udiActvTi04_h</td>
<td>UDINT</td>
</tr>
</tbody>
</table>
bEn: Enables the function block.

udiNum: Number of units.

bChg: Force sequence change.

bEn01...bEn04: Enable unit 1...enable unit 4.

udiActvTi01_h...udiActvTi04_h: Operating hours unit 1...operating hours unit 4.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>bQ01</th>
<th>: BOOL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bQ02</td>
<td>: BOOL;</td>
</tr>
<tr>
<td>bQ03</td>
<td>: BOOL;</td>
</tr>
<tr>
<td>bQ04</td>
<td>: BOOL;</td>
</tr>
<tr>
<td>udiNextOn</td>
<td>: UDINT;</td>
</tr>
<tr>
<td>udiNextOff</td>
<td>: UDINT;</td>
</tr>
<tr>
<td>arrFIFO</td>
<td>: ARRAY [1..4] OF UDINT;</td>
</tr>
<tr>
<td>udiNumOfEn</td>
<td>: UDINT;</td>
</tr>
<tr>
<td>bErr</td>
<td>: BOOL;</td>
</tr>
<tr>
<td>sErrDesc</td>
<td>: STRING;</td>
</tr>
</tbody>
</table>

bQ01...bQ04: Switches unit 1..4.

udiNextOn: Number of the unit that is switched on next.

udiNextOff: Number of the unit that is switched on next.

arrFIFO: FIFO buffer as a field.

udiNumOfEn: Number of devices, depending on the individual enable states.

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

sErrDesc: Contains the error description.

**Error description**

01: Warning: More than 4 devices are entered at input udiNum. The number is limited to the number enabled at inputs bEn01..bEn04.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
The function block *FB_BA_FIFO08* enables sequential control of up to eight units, with automatic switching of the switch-on sequence based on operating hours. The function block is available in two versions: for a sequence of four or eight units. Units with fewer operating hours take precedence in the sequence over units with more operating hours.

A rising edge at *bChg* forces a sequence change. The units with the fewest operating hours are set to the top of the FIFO and thus given priority for switching on. In the sequence only units that are enabled at inputs *bEn01..bEn08*. *udiNum* indicates the number of requested units. The operating hours of the units are entered at inputs *udiActvTi01_h* to *udiActvTi08_h*. If all these inputs are set to a constant value of zero, the sequence change is controlled cyclically, depending on *bChg*. The first unit is removed from the FIFO, the other units are advanced, and the first unit is appended at the end of the FIFO again. As a result is an alternating sequence of units.

If more units are requested at input *udiNum* than are available at inputs *bEn01* to *bEn08*, this is indicated with TRUE at *bErr*.

**Error handling**

If more units are requested at input *udiNum* than are available at inputs *bEn01* to *bEn08*, this is indicated with TRUE at *bErr*.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>Variables</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEn</td>
<td>BOOL</td>
</tr>
<tr>
<td>udiNum</td>
<td>UDINT</td>
</tr>
<tr>
<td>bChg</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEn01</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEn02</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEn03</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEn04</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEn05</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEn06</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEn07</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEn08</td>
<td>BOOL</td>
</tr>
<tr>
<td>udiActvT101_h</td>
<td>UDINT</td>
</tr>
<tr>
<td>udiActvT102_h</td>
<td>UDINT</td>
</tr>
<tr>
<td>udiActvT103_h</td>
<td>UDINT</td>
</tr>
<tr>
<td>udiActvT104_h</td>
<td>UDINT</td>
</tr>
<tr>
<td>udiActvT105_h</td>
<td>UDINT</td>
</tr>
<tr>
<td>udiActvT106_h</td>
<td>UDINT</td>
</tr>
<tr>
<td>udiActvT107_h</td>
<td>UDINT</td>
</tr>
<tr>
<td>udiActvT108_h</td>
<td>UDINT</td>
</tr>
</tbody>
</table>
bEn: Enables the function block.

udiNum: Number of units.

bChg: Force sequence change.

bEn01...bEn08: Enable unit 1...enable unit 8.

udiActvTi01_h...udiActvTi08_h: Operating hours unit 1...operating hours unit 8.

VAR_OUTPUT

bQ01 : BOOL;
bQ02 : BOOL;
bQ03 : BOOL;
bQ04 : BOOL;
bQ05 : BOOL;
bQ06 : BOOL;
bQ07 : BOOL;
bQ08 : BOOL;
udiNextOn : UDINT;
udiNextOff : UDINT;
arrFIFO : ARRAY [1..8] OF UDINT;
udiNumOfEn : UDINT;
bErr : BOOL;
sErrDescr : STRING;

bQ01...bQ08: Switches unit 1..8.

udiNextOn: Number of the unit that is switched on next.

udiNextOff: Number of the unit that is switched on next.

arrFIFO: FIFO buffer as a field.

udiNumOfEn: Number of devices, depending on the individual enable states.

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

sErrDescr: Contains the error description.

Error description

01: Warning: More than 8 devices are entered at input udiNum. The number is limited to the number enabled at inputs bEn01..bEn08.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
The function block is used for issuing sequential control commands. A typical application for this function block is startup of an air conditioning system. \( bEn \) is used for general enable of the function block. If \( bEn = \text{FALSE} \), all outputs of \( bQ01 \) to \( bQ08 \) are set to \( \text{FALSE} \). The control sequence starts at input \( bEvt01 \). Once the timer \( udiDlyOn01\_sec \) (see Parameters) has elapsed, the corresponding output \( bQ01 \) is set. Further stages are activated after a rising edge at the inputs \( bEvt02 \) to \( bEvt08 \), in each case delayed via the timers \( udiDlyOn02\_sec \) to \( udiDlyOn08\_sec \). If \( bEvt01 \) becomes \( \text{FALSE} \) once the control chain is up and running, the control sequence switches back in reverse order. Switching off of the outputs is delayed by the timers \( udiDlyOff01\_sec \) to \( udiDlyOff08\_sec \); see Parameters.

The outputs \( bUp \) and \( bDwn \) indicate whether the control chain is in ascending or descending state. The variable \( udiActvEvt \) indicates the current step of the control chain. "0" means the step sequence is not active.

The output \( udiRemTiDlyOn\_sec \) indicates the time remaining to the next step during up-switching of the control chain. The output \( udiRemTiDlyOff\_sec \) indicates the time remaining to the next lower step during down-switching of the control chain.
Example

- t0 step sequence switch-on
- t1 switch on step 1 $udiDlyOn01\_sec = t1 - t0$
- t2 event enable step 2, switch on step 2, $udiDlyOn02\_sec = 0$
- t3 event enable step 3, switch on step 3, $udiDlyOn03\_sec = 0$
- t4 event enable step 4, switch on step 4, $udiDlyOn04\_sec = 0$
- t5 event enable step 5, switch on step 5, $udiDlyOn05\_sec = 0$
- t6 disable the step sequence, disable step 5, disable step 4; $udiDlyOff05\_sec = 0, udiDlyOff04\_sec = 0$
- t7 disable step 3, $udiDlyOff03\_sec = t7 - t6$
- t8 disable step 2, $udiDlyOff02\_sec = t8 - t7$
- t9 disable step 1, $udiDlyOff01\_sec = t9 - t8$

VAR_INPUT

- bEn : BOOL;
- bEvt01 : BOOL;
- udiDlyOn01\_sec : UDINT;
- udiDlyOff01\_sec : UDINT;
- bEvt02 : BOOL;
- udiDlyOn02\_sec : UDINT;
- udiDlyOff02\_sec : UDINT;
- bEvt03 : BOOL;
- udiDlyOn03\_sec : UDINT;
- udiDlyOff03\_sec : UDINT;
- bEvt04 : BOOL;
- udiDlyOn04\_sec : UDINT;
- udiDlyOff04\_sec : UDINT;
- bEvt05 : BOOL;
- udiDlyOn05\_sec : UDINT;
- udiDlyOff05\_sec : UDINT;
- bEvt06 : BOOL;
- udiDlyOn06\_sec : UDINT;
- udiDlyOff06\_sec : UDINT;
- bEvt07 : BOOL;
- udiDlyOn07\_sec : UDINT;
udiDlyOff07_sec : UDINT;
bEvt08 : BOOL;
udiDlyOn08_sec : UDINT;
udiDlyOff08_sec : UDINT;

**bEn**: Enable function block.

**bEvt01..08**: Switch-on command for steps 1 to 8.

**udiDlyOn01..08_sec**: Switch-on delay for output \(bQ01 \ldots 08\) [s].

**udiDlyOff01..08_sec**: Switch-off delay for output \(bQ01 \ldots 08\) [s].

**VAR_OUTPUT**

\[\begin{align*}
bQ01 & : BOOL; \\
bQ02 & : BOOL; \\
bQ03 & : BOOL; \\
bQ04 & : BOOL; \\
bQ05 & : BOOL; \\
bQ06 & : BOOL; \\
bQ07 & : BOOL; \\
bQ08 & : BOOL; \\
bUp & : BOOL; \\
bDwn & : BOOL; \\
udiActvEvt & : UDINT; \\
udiRemTiDlyOn_sec & : UDINT; \\
udiRemTiDlyOff_sec & : UDINT;
\end{align*}\]

**bQ01..08**: Step 1 to 8 On.

**bUp**: Control chain is in ascending state.

**bDwn**: Control chain is in descending state.

**udiActvEvt**: Active step, display 0..8; "0" represents an active step sequence.

**udiRTiDlyOn**: Time remaining to up-switching to the next step [s].

**udiRTiDlyOff**: Time remaining to down-switching to the previous step [s].

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
The function block is used for issuing sequential control commands. A typical application for this function block is startup of an air conditioning system. If $bEn$ = FALSE, all outputs of $bQ01$ to $bQ12$ are set to FALSE. The control sequence starts at input $bEvt01$. Once the timer $udiDlyOn01\_sec$ (see Parameters) has elapsed, the corresponding output $bQ01$ is set. Further stages are activated after a rising edge at the inputs $bEvt02$ to $bEvt12$, in each case delayed via the timers $udiDlyOn02\_sec$ to $udiDlyOn12\_sec$. If $bEvt01$ becomes FALSE once the control chain is up and running, the control sequence switches back in reverse order. Switching off of the outputs is delayed by the timers $udiDlyOff01\_sec$ to $udiDlyOff12\_sec$; see Parameters.

The outputs $bUp$ and $bDwn$ indicate whether the control chain is in ascending or descending state. The variable $udiActvEvt$ indicates the current step of the control chain. "0" means the step sequence is not active.

The output $udiRemTiDlyOn\_sec$ indicates the time remaining to the next step during up-switching of the control chain. The output $udiRemTiDlyOff\_sec$ indicates the time remaining to the next lower step during down-switching of the control chain.
Example

- t0 step sequence switch-on
  - t1 switch on step 1 \( udiDlyOn01\_sec = t1 - t0 \)
- t2 event enable step 2, switch on step 2, \( udiDlyOn02\_sec = 0 \)
- t3 event enable step 3, switch on step 3, \( udiDlyOn03\_sec = 0 \)
- t4 event enable step 4, switch on step 4, \( udiDlyOn04\_sec = 0 \)
- t5 event enable step 5, switch on step 5, \( udiDlyOn05\_sec = 0 \)
- t6 disable the step sequence, disable step 5, disable step 4; \( udiDlyOff05\_sec = 0, udiDlyOff04\_sec = 0 \)
- t7 disable step 3, \( udiDlyOff03\_sec = t7 - t6 \)
- t8 disable step 2, \( udiDlyOff02\_sec = t8 - t7 \)
- t9 disable step 1, \( udiDlyOff01\_sec = t9 - t8 \)

**VAR_INPUT**

- \( bEn \) : BOOL;
- \( bEvt01 \) : BOOL;
- \( udiDlyOn01\_sec \) : UDINT;
- \( udiDlyOff01\_sec \) : UDINT;
- \( bEvt02 \) : BOOL;
- \( udiDlyOn02\_sec \) : UDINT;
- \( udiDlyOff02\_sec \) : UDINT;
- \( bEvt03 \) : BOOL;
- \( udiDlyOn03\_sec \) : UDINT;
- \( udiDlyOff03\_sec \) : UDINT;
- \( bEvt04 \) : BOOL;
- \( udiDlyOn04\_sec \) : UDINT;
- \( udiDlyOff04\_sec \) : UDINT;
- \( bEvt05 \) : BOOL;
- \( udiDlyOn05\_sec \) : UDINT;
- \( udiDlyOff05\_sec \) : UDINT;
- \( bEvt06 \) : BOOL;
- \( udiDlyOn06\_sec \) : UDINT;
- \( udiDlyOff06\_sec \) : UDINT;
- \( bEvt07 \) : BOOL;
- \( udiDlyOn07\_sec \) : UDINT;
bEn: Enable function block.

bEvt01..12: Switch-on command for steps 1 to 12.

udiDlyOn01..12_sec: Switch-on delay for output bQ01 .. 12 [s].

udiDlyOff01..12_sec: Switch-off delay for output bQ01 .. 12 [s].

VAR_OUTPUT

bQ01 : BOOL;
bQ02 : BOOL;
bQ03 : BOOL;
bQ04 : BOOL;
bQ05 : BOOL;
bQ06 : BOOL;
bQ07 : BOOL;
bQ08 : BOOL;
bQ09 : BOOL;
bQ10 : BOOL;
bQ11 : BOOL;
bQ12 : BOOL;
bUp : BOOL;
bDwn : BOOL;
udiActvEvt : UDINT;
udiRemTiDlyOn_sec : UDINT;
udiRemTiDlyOff_sec : UDINT;

bQ01..12: Step 1 to 12 On.

bUp: Control chain is in ascending state.

bDwn: Control chain is in descending state.

udiActvEvt: Active step, display 0..12; "0" represents an active step sequence.

udiRTiDlyOn: Time remaining to up-switching to the next step [s].

udiRTiDlyOff: Time remaining to down-switching to the previous step [s].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
FB_BA_FIFO04_XX

This function block is used to evaluate the FiFo memory from FB_BA_FIFO04.[374]. The inputs are linked according to the FIFO table to the corresponding outputs of the function block FB_BA_FIFO04_BOOL or FB_BA_FIFO04_REAL.

Sample:
In the sample the array contains: 4,3,1,2,0,0,0,0. The following result is output in FB_BA_FIFO04_REAL:

- rIn01 at output rVal04
- rIn02 at output rVal03
- rIn03 at output rVal01
- rIn04 at output rVal02

VAR_INPUT

arrFIFO : Array [1..4] OF UDINT;
rIn01 - rIn04 : REAL;

arrFIFO: Contains the assignment table with a maximum of eight values. The first value indicates where the first input will be copied to, the second value indicates where the second input will be copied to, etc. No assignment takes place with "0".

rIn01 - rIn04: Setpoints to be linked.

VAR_OUTPUT

rVal01 - rVal04 : REAL;

rVal01 - rVal04: set actuator value, input value linked according to FIFO table.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>


This function block is used to evaluate the FiFo memory from FB_BA_FIFO08_XX. The inputs are linked according to the FIFO table to the corresponding outputs of the function block FB_BA_FIFO08_BOOL or FB_BA_FIFO08_REAL.

Sample:
In the sample the array contains: 4,3,1,2,0,0,0,0. The result output in FB_BA_FIFO08_REAL is

- rln01 at output rVal04
- rln02 at output rVal03
- rln03 at output rVal01
- rln04 at output rVal02

**VAR_INPUT**

```
arrFIFO : Array [1..8] OF UDINT;
rIn01 – rIn08 : REAL;
```

**arrFIFO:** Contains the assignment table with a maximum of eight values. The first value indicates where the first input will be copied to, the second value indicates where the second input will be copied to, etc. No assignment takes place with "0".

**rln01 – rln08:** Setpoints to be linked.

**VAR_OUTPUT**

```
rVal01 – rVal08 : REAL;
```

**rVal01 – rVal08:** set actuator value, input value linked according to FIFO table.
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**Hysteresis, 2-point control**

**Function blocks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_BA_Cont4Stp01</td>
<td>Step switch with four stages</td>
</tr>
<tr>
<td>FB_BA_Swi2P</td>
<td>Two-point switch</td>
</tr>
<tr>
<td>FB_BA_SwiHys2P</td>
<td>Two-point switch with one switching point</td>
</tr>
</tbody>
</table>

**FB_BA_Cont4Stp01**

```
-- bEn  BOOL  bQ01
-- rIn  REAL  bQ02
-- rSwiOn01 REAL  bQ03
-- rHys01 REAL  bQ04
-- udiDlyOn01_sec UINT udiStp
-- udiDlyOff01_sec UINT
-- rSwiOn02 REAL  rSwiOff
-- rHys02 REAL  udiRemTIDlyOn_sec
-- udiDlyOn02_sec UINT udiRemTIDlyOff_sec
-- udiDlyOff02_sec UINT
-- rSwiOn03 REAL
-- rHys03 REAL
-- udiDlyOn03_sec UINT
-- udiDlyOff03_sec UINT
-- rSwiOn04 REAL
-- rHys04 REAL
-- udiDlyOn04_sec UINT
-- udiDlyOff04_sec UINT
-- udiNumStp UINT
-- bActn BOOL
```

The function block determines the resulting switching stages of a multi-level unit, depending on the input signal. Four switch-on thresholds and four hysteresis values can be parameterized.

**Diagram 01**

Control direction of parameter bActn = FALSE = Reverse = Heating
Diagram 02
Control direction parameter \( bActn = \text{TRUE} = \text{Direct} = \text{Cooling} \)
Diagram 03

Timing of the switch-on and switch-off delays

At time t1, rIn jumps from rSwiOn01 to rSwiOn04
At time t2, \( rIn \) jumps from \( rSwiOn04 \) to \( rSwiOn01 - rHys01 \)

**VAR_INPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEn</td>
<td>BOOL;</td>
</tr>
<tr>
<td>rIn</td>
<td>REAL;</td>
</tr>
<tr>
<td>rSwiOn01</td>
<td>REAL</td>
</tr>
<tr>
<td>rHys01</td>
<td>REAL</td>
</tr>
<tr>
<td>udiDlyOn01_sec</td>
<td>UDINT;</td>
</tr>
<tr>
<td>udiDlyOff01_sec</td>
<td>UDINT;</td>
</tr>
<tr>
<td>rSwiOn02</td>
<td>REAL;</td>
</tr>
<tr>
<td>rHys02</td>
<td>REAL;</td>
</tr>
<tr>
<td>udiDlyOn02_sec</td>
<td>UDINT;</td>
</tr>
<tr>
<td>udiDlyOff02_sec</td>
<td>UDINT;</td>
</tr>
<tr>
<td>rSwiOn03</td>
<td>REAL;</td>
</tr>
<tr>
<td>rHys03</td>
<td>REAL;</td>
</tr>
<tr>
<td>udiDlyOn03_sec</td>
<td>UDINT;</td>
</tr>
<tr>
<td>udiDlyOff03_sec</td>
<td>UDINT;</td>
</tr>
<tr>
<td>rSwiOn04</td>
<td>REAL;</td>
</tr>
<tr>
<td>rHys04</td>
<td>REAL;</td>
</tr>
<tr>
<td>udiDlyOn04_sec</td>
<td>UDINT;</td>
</tr>
<tr>
<td>udiDlyOff04_sec</td>
<td>UDINT;</td>
</tr>
<tr>
<td>udiNumOfStp</td>
<td>UDINT;</td>
</tr>
<tr>
<td>bActn</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>

**bEn**: General enable of the function block. If \( bEn \) is FALSE, all outputs are set to 0.

**rIn**: Input value, from which the switching state is derived.

**rSwiOn01**: Switch-on point stage 01

**rHys01**: Absolute value hysteresis stage 01

**udiDlyOn01_sec**: Switch-on delay stage 01

**udiDlyOff01_sec**: Switch-off delay stage 01

**rSwiOn02**: Switch-on point stage 02

**rHys02**: Absolute value hysteresis stage 02

**udiDlyOn02_sec**: Switch-on delay stage 02

**udiDlyOff02_sec**: Switch-off delay stage 02
rSwiOn03: Switch-on point stage 03
rHys03: Absolute value hysteresis stage 03
udiDlyOn03_sec: Switch-on delay stage 03
udiDlyOff03_sec: Switch-off delay stage 03
rSwiOn04: Switch-on point stage 04
rHys04: Absolute value hysteresis stage 04
udiDlyOn04_sec: Switch-on delay stage 04
udiDlyOff04_sec: Switch-off delay stage 04
udiNumOfStp: Number of stages that are required.
The input is limited to a range from 0 to 4
bActn: Input variable used to determine the control direction of the step switch.
TRUE = direct = cooling; FALSE = reverse = heating

VAR_OUTPUT
bQ01 : BOOL;
bQ02 : BOOL;
bQ03 : BOOL;
bQ04 : BOOL;
udiStp : UDINT;
rSwiOn : REAL;
rSwiOff : REAL;
udiRemTiDlyOn_sec : UDINT;
udiRemTiDlyOff_sec : UDINT;

bQ01: Display of status step 01
TRUE = ON; FALSE = OFF
udiStp >= 1

bQ02: Display of status step 02
TRUE = ON; FALSE = OFF
udiStp >= 2

bQ03: Display of status step 03
TRUE = ON; FALSE = OFF
udiStp >= 3

bQ04: Display of status step 04
TRUE = ON; FALSE = OFF
udiStp >= 4

udiStp: Shows the current step of the step switch
rSwiOn: Shows the next switch-on point
rSwiOff: Shows the next switch-off point

udiRemTiDlyOn_sec: If the switch-on point for switching to the next level is met, the progress of the switch-
on delay time is displayed here.
udiRemTiDlyOff_sec: If the switch-off point for switching down to the next level is met, the progress of the
switch-off delay time is displayed here.

Requirements
<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
The function block \textit{FB\_BA\_Swi2P} is a two-point switch with one switch-on point and one switch-off point.

A general function block enable can be implemented at input \textit{bEn}. The output \textit{bQ} is FALSE as long as \textit{bEn} is FALSE. The control direction of the function block depends on the relative position of the switch-on/switch-off points.

If the switch-on point is greater than the switch-off point, the control direction is direct/synchronous (cooling mode).

If the switch-off point is greater than the switch-on point, the control direction is indirect/reversed (heating mode).
VAR_INPUT

bEn : BOOL;
rIn : REAL;
rOn : REAL;
rOff : REAL;
udiDlyOn_sec : UDINT;
udiDlyOff_sec : UDINT;

bEn: General enable of the function block.
rIn: Input value.
rOn: Switch-on point.
rOff: Switch-off point.
udiDlyOn_sec: Switch-on delay.
udiDlyOff_sec: Switch-off delay.

VAR_OUTPUT

bQ : BOOL;
udiRemTiDlyOn_sec : UDINT;
udiRemTiDlyOff_sec : UDINT;

bQ: Control output.
udiRemTiDlyOn_sec: Remaining time of the switch-on delay.
udiRemTiDlyOff_sec: Remaining time of the switch-off delay.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
The function block **FB_BA_SwiHys2P** is a two-point switch with adjustable hysteresis and hysteresis offset.

A general function block enable can be implemented at input **bEn**. If the function block is locked, the output **bQ** is FALSE. The setpoint for the two-point switch is connected at input **rSp**. The control direction of the function block depends on the input variable **bActn**.

The active switching points result from the setpoint, the hysteresis and the hysteresis offset. They are output at **rSwiHi** and **rSwiLo**.

- The upper switching point results from \( rSp + rHys/2 + rHysOffs \).
- The lower switching point results from \( rSp + rHys/2 + rHysOffs \).

If **bActn** TRUE, the result is direct/synchronous control direction (cooling mode).

If **bActn** FALSE, the result is indirect/reversed control direction (heating mode).
### VAR_INPUT

- **bEn**: General enable of the function block.
- **rIn**: Input value.
- **rSp**: Setpoint input.
- **rHys**: Hysteresis.
- **rHysOffs**: Hysteresis offset.
- **udiDlyOn_sec**: Switch-on delay
- **udiDlyOff_sec**: Release delay
- **bActn**: Control direction.

### VAR_OUTPUT

- **bQ**: Output.
- **rSwiHi**: Upper switching point.
- **rSwiLo**: Lower switching point.
- **udiRemTiDlyOn_sec**: Time remaining before switching on.
- **udiRemTiDlyOff_sec**: Time remaining before switching off.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

### Mathematical functions

#### Function blocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_BA_MultiCalc_XX [395]</td>
<td>Multi-calculation function blocks</td>
</tr>
<tr>
<td>FB_BA_Chrc02 [396]</td>
<td>Linear interpolation for 2 interpolation points</td>
</tr>
<tr>
<td>FB_BA_Chrc04 [398]</td>
<td>Linear interpolation for 4 interpolation points</td>
</tr>
<tr>
<td>FB_BA_Chrc07 [399]</td>
<td>Linear interpolation for 7 interpolation points</td>
</tr>
<tr>
<td>FB_BA_Chrc32 [401]</td>
<td>Linear interpolation for 32 interpolation points</td>
</tr>
<tr>
<td>FB_BA_TiAvrg [402]</td>
<td>Arithmetic mean value over time</td>
</tr>
</tbody>
</table>
The multi-calculation function blocks exist for the variable types LREAL and REAL, although they all have the same functionality. The function block FB_BA_R08 is described as an example.

In enabled state \((bEn=TRUE)\), the function block determines the following from the 8 input values \(r01\ldots r08\):
- the maximum value of all inputs \(rMax\)
- the input at which this maximum value occurs \(udiMaxActv\)
- the minimum value of all inputs \(rMin\)
- the input at which this minimum value occurs \(udiMinActv\)
- the mean value of all inputs \(rAvrg\)
- the sum of all inputs \(rSum\)
- the difference between the maximum and minimum value \(rDiff\)

If not all inputs are used for the calculation, the number can be limited via an entry at \(udiNum\): \(udiNum=6\), for example, can be used to limit the calculations to inputs \(r01\) to \(r06\). Any entry greater than 8 is automatically limited to 8, any entry less than 1 is automatically set to 1.

Sample:

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEn = TRUE</td>
<td>rMax = 32</td>
</tr>
<tr>
<td>r01 = 32</td>
<td>udiMaxActv = 1</td>
</tr>
<tr>
<td>r02 = 17</td>
<td>rMin = 5</td>
</tr>
<tr>
<td>r03 = 5</td>
<td>udiMinActv = 3</td>
</tr>
<tr>
<td>r04 = 9</td>
<td>rAvrg = 18.5</td>
</tr>
<tr>
<td>r05 = 16</td>
<td>rSum = 111</td>
</tr>
<tr>
<td>r06 = 32</td>
<td>rDiff = 27</td>
</tr>
<tr>
<td>r07 = 25</td>
<td></td>
</tr>
<tr>
<td>r08 = 44</td>
<td></td>
</tr>
<tr>
<td>udiNum = 6</td>
<td></td>
</tr>
</tbody>
</table>

If \(bEn=FALSE\), 0 is output at all outputs.

### VAR_INPUT

\[
\begin{align*}
\text{bEn} & : \text{ BOOL;} \\
\text{r01} & : \text{ REAL;} \\
\text{r02} & : \text{ REAL;} \\
\text{r03} & : \text{ REAL;} \\
\text{r04} & : \text{ REAL;} \\
\text{r05} & : \text{ REAL;} \\
\text{r06} & : \text{ REAL;} \\
\end{align*}
\]
bEn: Activation of the block function.

r01...r08: Input values to be used for the calculation.

udiNum: Number of input values to be used for the calculation.

VAR_OUTPUT

rMax: Maximum value of all inputs.

udiMaxActv: Input at which the maximum value is present.

rMin: Minimum value of all inputs.

udiMinActv: Input at which the minimum value is present.

rAvrg: Mean value of all inputs.

rSum: Sum of all inputs.

rDiff: Difference between maximum and minimum value.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

FB_BA_Chrc02

The function block FB_BA_Chrc02 represents a linear interpolation with 2 interpolation points and can be used to generate a characteristic curve. The characteristic curve is determined by the interpolation points \([rX1/rY1]\) and \([rX2/rY2]\). If the input variable \(bLmt\) is TRUE, \(rY\) is limited by \(rY01\) and \(rY02\). If \(bLmt\) is FALSE, \(rY\) is not limited.
Error handling

The input values for \( rX[n+1] \) must always be at least 0.0000001 greater than the values for \( rX[n] \).

In the event of an error the variable \( sErrDescr \) indicates that at one point of the characteristic curve the values are not monotonically increasing.

**VAR_INPUT**

rX : REAL;
rX01 : REAL;
rX02 : REAL;
rY01 : REAL;
rY02 : REAL;
bLmt : BOOL;

- **rX**: Input value of the characteristic curve.
- **rX01**: X-value for interpolation point P1.
- **rX02**: X-value for interpolation point P2.
- **rY01**: Y-value for interpolation point P1.
- **rY02**: Y-value for interpolation point P2.
- **bLmt**: Limit for the output value \( rY \).

**VAR_OUTPUT**

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

- **rY**: Calculated output value of the characteristic curve.
- **bErr**: This output is switched to TRUE if the parameters entered are erroneous.
- **sErrDescr**: Contains the error description.

**Error description**

01: Error: \( rX01 \) must not be equal to \( rX02 \).

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
The function block FB_BA_Chrct04 represents a linear interpolation with 4 interpolation points and can be used to generate a characteristic curve. The characteristic curve is determined by the interpolation points \([rX1/rY1]\) to \([rX4/rY4]\). If the input variable \(bLmt\) is TRUE, \(rY\) is limited by \(rY01\) and \(rY04\). If \(bLmt\) is FALSE, \(rY\) is not limited.

**Error handling**

The input values for \(rX[n+1]\) must always be at least 0.0000001 greater than the values for \(rX[n]\). In the event of an error the variable \(sErrDescr\) indicates that at one point of the characteristic curve the values are not monotonically increasing.

**VAR_INPUT**

\[
\begin{align*}
\text{rX} & : \text{REAL;} \\
\text{rX01} & : \text{REAL;} \\
\text{rX02} & : \text{REAL;} \\
\text{rX03} & : \text{REAL;} \\
\text{rX04} & : \text{REAL;} \\
\text{rY01} & : \text{REAL;} \\
\text{rY02} & : \text{REAL;} \\
\text{rY03} & : \text{REAL;} \\
\text{rY04} & : \text{REAL;} \\
\text{bLmt} & : \text{BOOL;} \\
\end{align*}
\]

- \(rX\): Input value of the characteristic curve.
- \(rX01\): X-value for interpolation point P1.
- \(rX02\): X-value for interpolation point P2.
- \(rX03\): X-value for interpolation point P3.
- \(rX04\): X-value for interpolation point P4.
- \(rY01\): Y-value for interpolation point P1.
rY02: Y-value for interpolation point P2.

rY03: Y-value for interpolation point P3.

rY04: Y-value for interpolation point P4.

bLmt: Limit for the output value rY.

VAR_OUTPUT

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

rY: Calculated output value of the characteristic curve.

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

sErrDescr: Contains the error description.

Error description

01: Error: at the specified element. The sequence must always be rX01 > rX02 > rXn or rX01 < rX02 < rXn.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

FB_BA_Chrc07

The function block FB_BA_Chrc07 represents a linear interpolation with 7 interpolation points and can be used to generate a characteristic curve. The characteristic curve is determined by the interpolation points [rX1/rY1] to [rX7/rY7]. If the input variable bLmt is TRUE, rY is limited by rY01 and rY07. If bLmt is FALSE, rY is not limited.
Error handling

The input values for \( rX[n+1] \) must always be at least 0.0000001 greater than the values for \( rX[n] \). In the event of an error the variable \( sErrDescr \) indicates that at one point of the characteristic curve the values are not monotonically increasing.

**VAR_INPUT**

- \( rX \): Input value of the characteristic curve.
- \( rX01 \): X-value for interpolation point P1.
- \( rX02 \): X-value for interpolation point P2.
- \( rX03 \): X-value for interpolation point P3.
- \( rX04 \): X-value for interpolation point P4.
- \( rX05 \): X-value for interpolation point P5.
- \( rX06 \): X-value for interpolation point P6.
- \( rX07 \): X-value for interpolation point P7.
- \( rY01 \): Y-value for interpolation point P1.
- \( rY02 \): Y-value for interpolation point P2.
- \( rY03 \): Y-value for interpolation point P3.
- \( rY04 \): Y-value for interpolation point P4.
- \( rY05 \): Y-value for interpolation point P5.
- \( rY06 \): Y-value for interpolation point P6.
- \( rY07 \): Y-value for interpolation point P7.
- \( bLmt \): Limit for the output value \( rY \).
VAR_OUTPUT

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

rY : Calculated output value of the characteristic curve.

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

sErrDescr : Contains the error description.

Error description

01: Error: at the specified element. The sequence must always be rX01 > rX02 > rXn or rX01 < rX02 < rXn.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

FB_BA_Chrct32

The function block FB_BA_Chrct32 represents a linear interpolation with up to 32 interpolation points and can be used to generate a characteristic curve. In contrast to the “smaller” interpolation function blocks FB_BA_Chrct02 [396], FB_BA_Chrct04 [398] and FB_BA_Chrct07 [399], and in the interest of clarity, the interpolation points are determined via field variables [arrX[1]/arrY[1]] to [arrX[n]/arrY[n]]. If the input variable bLmt is TRUE, rY is limited by arrY[1] and arrY[n]. If bLmt is FALSE, rY is not limited.

Error handling

The input values for rX[n+1] must always be at least 0.0000001 greater than the values for rX[n].

In the event of an error the variable sErrDescr indicates that at one point of the characteristic curve the values are not monotonically increasing.

The parameter for the number of interpolation points, diNumOfElem, must be in the range 2..32.

VAR_INPUT

rX : REAL;
arrX : ARRAY [1..cBA_NumOfElem] OF REAL;
arrY : ARRAY [1..cBA_NumOfElem] OF REAL;
diNumOfElem : DINT(2..32);
bLmt : BOOL;
rX: Input value of the characteristic curve

arrX: Field with the X-values for the interpolation points.

arrY: Field with the Y-values for the interpolation points.

diNumOfElem: Number of interpolation points. Internally limited to values between 2 and 32.

bLmt: Limit for the output value rY.

VAR_OUTPUT

<table>
<thead>
<tr>
<th>rY</th>
<th>: REAL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bErr</td>
<td>: BOOL;</td>
</tr>
<tr>
<td>sErrDescr</td>
<td>: T_MAXSTRING;</td>
</tr>
</tbody>
</table>

rY: Calculated output value of the characteristic curve.

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

sErrDescr: Contains the error description.

Error description

01: Error: at the specified element. The sequence must always be rX01 > rX02 > rXn or rX01 < rX02 < rXn.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

FB_BA_TiAvrg

The function block FB_BA_TiAvrg calculates the arithmetic mean value of an analog value that was logged over a certain period. Discrete values are written into a FIFO buffer. udiIntval_sec specifies the time interval [s] over which the values are logged and written into the FIFO. Values are written if the input bEn is TRUE. The variable udiNumOfElem is used to determine the size of the FIFO buffer. It is limited to 1..512.

The function block can be used for calculating an hourly mean outside temperature over a day, for example. In this case udiNumOfElem would be 24 and udiIntval_SEC would be 3600 seconds. bEn is the general enable of the function block. If bEn = FALSE, the FIFO buffer within the function block is deleted completely, and no data are recorded.

Example:

udiNumOfElem = 5
<table>
<thead>
<tr>
<th>First cycle</th>
<th>Second cycle</th>
<th>Third cycle</th>
<th>Fourth cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>rIn</td>
<td>rOut</td>
<td>rIn</td>
<td>rOut</td>
</tr>
<tr>
<td>t0</td>
<td>2</td>
<td>(2/1 = 2)</td>
<td>6</td>
</tr>
<tr>
<td>t1</td>
<td>4</td>
<td>((2+4)/2 = 3)</td>
<td>5</td>
</tr>
<tr>
<td>t2</td>
<td>6</td>
<td>((2+4+6)/3 = 4)</td>
<td>4</td>
</tr>
<tr>
<td>t3</td>
<td>7</td>
<td>((2+4+6+7)/4 = 4.75)</td>
<td>2</td>
</tr>
<tr>
<td>t4</td>
<td>7</td>
<td>((2+4+6+7+7)/5 = 5.2)</td>
<td>1</td>
</tr>
</tbody>
</table>

**VAR_INPUT**

bEn : BOOL;
rIn : REAL;
udiIntVal_SEC : UDINT;
udiNumOfElem : UDINT;

bEn: Enables the function block.

rIn: Input value for averaging.

udiIntVal_SEC: Time interval [s] for writing new values into the FIFO. Internally limited to a value between 1 and 2147483.

udiNumOfElem: Size of the FIFO buffer. A change resets the previous averaging. Internally limited to a value between 1 and 512.

**VAR_OUTPUT**

rOut : REAL;
rMax : REAL;
rMin : REAL;

rOut: Calculated mean value.

rMax: Largest value in the FIFO buffer.

rMin: Smallest value in the FIFO buffer.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**Monitoring functions**

**Function blocks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_BA_FdbCtrlBinary</td>
<td>Feedback monitoring of an actuator by means of digital feedback.</td>
</tr>
<tr>
<td>FB_BA_FixedLimitCtrl</td>
<td>Limit value monitoring of a fixed value.</td>
</tr>
<tr>
<td>FB_BA_SlidingLimitCtrl</td>
<td>Sliding limit value monitoring.</td>
</tr>
</tbody>
</table>
FB_BA_FixedLimitCtrl

Function block for monitoring a fixed limit value. The input \( bEn \) is used for enabling the function block. A tolerance range is defined around the value \( rIn \) to be monitored. The tolerance range results from an upper limit value \( rInHighLimit \) and a lower limit value \( rInLowLimit \). If the value \( rIn \) exceeds the upper limit value of the tolerance range, then the output \( bHighLimit \) becomes TRUE. A response delay of the output \( bHighLimit \) must be parameterized with the timer \( udiDelay_sec \). If the value \( rIn \) falls below the lower limit of the tolerance range, output \( bLowLimit \) becomes TRUE. A response delay of the output \( bLowLimit \) must be parameterized with the timer \( udiDelay_sec \).

VAR_INPUT

\begin{verbatim}
VAR_INPUT
bEn               : BOOL;
rHighLimit       : REAL := 32;
rLowLimit        : REAL := 16;
rIn              : REAL;
udiDelay_sec     : UDINT;
\end{verbatim}

\( bEn \): Function block enable.
\( rHighLimit \): Default upper limit value, preset to 32.
\( rLowLimit \): Default lower limit value, preset to 16.
\( rIn \): Input value to be monitored.
\( udiDelay_sec \): Output response delay [s]. Internally limited to values between 0 and Const.udiTiSec [\ref{419}].

VAR_OUTPUT

\begin{verbatim}
VAR_OUTPUT
bHighLimit       : BOOL;
bLowLimit        : BOOL;
udiRemTiDelay_sec : UDINT;
\end{verbatim}

\( bHighLimit \): Upper limit value reached.
\( bLowLimit \): Lower limit value reached.
\( udiRemTiDelay_sec \): Time remaining after a limit value has been exceeded until either the output \( bHighLimit \) or \( bLowLimit \) responds.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
The function block is used for feedback monitoring of an actuator by means of digital feedback. Application examples of the function block are, for example, an operation feedback monitoring, a process feedback monitoring or the run monitoring of a drive by means of limit switches.

The input bEn is used for enabling the function block. If bEn is FALSE, the message output bQ will always be FALSE.

The switching actuator output of the unit to be monitored is connected to the input bActuator. The bSwitch input is used to connect the feedback signal (e.g. differential pressure switch, flow monitor or limit switch).

By means of the timer udiFdbDelay_sec [s] a response delay of the feedback control after the start of the unit is set.

The second timer udiInterruptionDelay_sec [s] serves for a response delay of the feedback control after reaching the final state.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEn</td>
<td>Function block enable.</td>
</tr>
<tr>
<td>bActuator</td>
<td>Feedback of the switching output.</td>
</tr>
<tr>
<td>bSwitch</td>
<td>Feedback signal from the process.</td>
</tr>
<tr>
<td>udiFdbDelay_sec</td>
<td>Response delay [s] of the monitoring function when the actuator is started. Internally limited to values between 0 and Const.udiTiSec [419].</td>
</tr>
<tr>
<td>udiInterruptionDelay_sec</td>
<td>Response delay [s] of the monitoring function when the actuator has already been started successfully (e.g. pressure fluctuations when monitoring the running of a fan). Internally limited to values between 0 and Const.udiTiSec [419].</td>
</tr>
</tbody>
</table>

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bQ</td>
<td>Output an error message if the feedback signal is not present within the parameterized time of udiFdbDelay_sec, or the feedback signal has been interrupted longer than after udiInterruptionDelay_sec.</td>
</tr>
<tr>
<td>udiRemTiFdbDelay</td>
<td>Remaining time [s] until output bErrOpn is set.</td>
</tr>
<tr>
<td>udiRemTiInterruptionDelay</td>
<td>Remaining time [s] until output bErrSwi is set.</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
**FB_BA_SlidingLimitCtrl**

Function block for monitoring a floating setpoint.

The input `bEn` is used for enabling the function block.

To check the function of a control system, the actual value is compared with the setpoint of the controlled system.

If the deviation of setpoint and actual value is within the tolerance range `rHys`, then the control is OK. If the actual value deviates from the setpoint by an amount outside this tolerance range over a longer period of time, the timer `udiDelay_sec` is started. After the timer has expired, if the control deviation is permanent, either the output `bLowLimit` or `bHighLimit` TRUE of the function block outputs a message.

**VAR_INPUT**

- `bEn` : BOOL;
- `rW` : REAL;
- `rX` : REAL;
- `rHys` : REAL;
- `udiDelay_sec` : UDINT;

`bEn`: Function block enable.
rW: Setpoint.

rX: Actual value.

rHys: Hysteresis.

udiDelay_sec: Output response delay [s]. Internally limited to values between 0 and Const.udiTiSec[419].

VAR_OUTPUT

bHighLimit : BOOL;
bLowLimit : BOOL;
rHighLimit : REAL;
rLowLimit : REAL;
udiRemTiDelay_sec : UDINT;

bHighLimit: Upper limit value reached.

bLowLimit: Lower limit value reached.

rHighLimit: Output of the upper limit value.

rLowLimit: Output of the lower limit value.

udiRemTiDelay_sec: Time remaining after a limit value has been exceeded until either the output bHighLimit or bLowLimit responds.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

Ramps, filters, controllers

Function blocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_BA_FltrPT1</td>
<td>First order filter</td>
</tr>
<tr>
<td>FB_BA_RampLmt</td>
<td>Ramp limitation</td>
</tr>
<tr>
<td>FB_BA_SeqCtrl</td>
<td>Sequence controller (function block in Tc3_BA_Common)</td>
</tr>
<tr>
<td>FB_BA_SeqLink</td>
<td>Sequence linker (function block in Tc3_BA_Common)</td>
</tr>
<tr>
<td>FB_BA_PIDCtrl</td>
<td>PID controller (function block in Tc3_BA_Common)</td>
</tr>
</tbody>
</table>

FB_BA_FltrPT1

FB_BA_FltrPT1

rIn REAL

rOut REAL

udiDampConst_sec UDINT

bSetActl BOOL

First order filter.

- When the function block is first called (system start), the output rOut is automatically set (once) to the input rIn.
VAR_INPUT

rIn : REAL;
udiDampConst_sec : UDINT;
bSetActl : BOOL;

rIn: Input signal

udiDampConst_sec: Filter time constant [s]. Internally limited to values between 0 and 86400.
bSetActl: A rising edge at this input sets the output value rOut to the input value rIn.

VAR_OUTPUT

rOut : REAL;

rOut: Filtered output signal.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

FB_BA_RampLmt

The function block limits the increase or decrease speed of an input signal.
An increase of rIn results in the output rOut to be limited to the slope of (rHi-rLo)/udiTiUp.
A decrease of rIn results in the output rOut to be limited to the slope of (rHi-rLo)/udiTiUp.
VAR_INPUT

bEn : BOOL;
bEnRamp : BOOL;
rIn : REAL;
rHi : REAL;
rLo : REAL;
udiTiUp_sec : UDINT;
udiTiDwn_sec : UDINT;

bEn: Enable function block if FALSE, in which case rOut = 0.0.
bEnRamp: Enable ramp limitation if FALSE, in which case rOut = rIn.

rIn: Input value of the ramp function
rHi: Upper interpolation point for calculating the ramps.
rLo: Lower interpolation point for calculating the ramps. rHi must be greater than rLo, otherwise an error is output!

udiTiUp_sec: Rise time [s].
udiTiDwn_sec: Fall time [s]

VAR_OUTPUT

rOut : REAL;

rOut: Output signal, slope-limited through the ramps

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
Calendar

Function blocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_BA_SchedulerWeeklyXXCh [410]</td>
<td>Weekly scheduler</td>
</tr>
<tr>
<td>FB_BA_CalendarXXCh [411]</td>
<td>Yearly scheduler</td>
</tr>
</tbody>
</table>

FB_BA_SchedulerWeeklyXXCh

Weekly timer with 1, 7 or 28 timer channels.

The function block FB_BA_SchedulerWeekly07Ch is described as an example.

The function block is used to enter a total of up to 7 switch-on periods.
Each switch-on period can be assigned an switch-on time [hh:mm:ss] and a switch-off time [hh:mm:ss].
The variables bMonday to bSunday can be used to select on which days of the week the switch-on period should be active.
A switch-on period is only active if the variable bEn of the channel is set to TRUE.
For irregular but recurring events, the variable bResetAfterOn can be set to TRUE. This will automatically reset the enable of channel bEn to FALSE after the event has finished.
To facilitate data entry, a rising edge at bAllActive sets bEn and all days of the week (bMonday to bSunday) to TRUE.

The function block is only active if a TRUE signal is present at bEn.

For demand-dependent switch-on optimization, switching on of the output bQ can be brought forward by the time of the variable udiPredictTime_sec.

The switch-on and switch-off points of a channel must be in the same year. The switch-off point must not be earlier than the switch-on point. Otherwise the switch-off point is automatically corrected and set to the same value as the switch-on point.
If the switch-on point is equal to the switch-off point, the channel remains off.

VAR_INPUT

<table>
<thead>
<tr>
<th>bEn</th>
<th>BOOL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>stSysTi</td>
<td>TIMESTRUCT;</td>
</tr>
<tr>
<td>udiPredictTime_sec</td>
<td>UDINT;</td>
</tr>
</tbody>
</table>

bEn: General function block enable.

stSysTi: Structure with the local NT system time (see TIMESTRUCT).

udiPredictTime_sec: Precalculated switch-on time. Internally limited to values between 0 and 43200.

VAR_OUTPUT

<table>
<thead>
<tr>
<th>bQ</th>
<th>BOOL;</th>
</tr>
</thead>
</table>

bQ: Switching output

VAR_IN_OUT

arrChannel: ARRAY [1..cBA_NumOfChannels] OF ST_BA_SchedulerWeeklyChannel;
arrChannel: Weekly scheduler; with the single-channel function block, the name of the variable is stChannel (see ST_BA_SchedulerWeeklyChannel [418]). Internally limited to the respective number of possible channels via the variable cBA_NumOfChannels.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3 BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

FB_BA_CalenderXXCh

Yearly scheduler with 1, 7 or 28 channels.

The function block FB_BA_Calender07Ch is described as an example.

This function block is used to enter periods such as school holidays or company holidays. The function block is enabled by the input variable bEnable. The input stSysTi is linked to the current system time. If the time switching condition is fulfilled, output bQ is set. Within the calendar, a time period is described by a switch-on date [day, month, hour, minute] and a switch-off date [day, month, hour, minute]. A switch-on period is only active if the variable bEn of the channel is set to TRUE. For irregular but recurring periods, the variable bResetAfterOn can be set to TRUE. The enable parameter bEn is then automatically reset to FALSE after the time has elapsed.

The switch-on and switch-off points of a channel must be in the same year. The switch-off point must not be earlier than the switch-on point. Otherwise the switch-off point is automatically corrected and set to the same value as the switch-on point. If the switch-on point is equal to the switch-off point, the channel remains off.

VAR_INPUT

<table>
<thead>
<tr>
<th>bEn</th>
<th>BOOL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>stSysTi</td>
<td>TIMESTRUCT;</td>
</tr>
</tbody>
</table>

bEn: General function block enable.

stSysTi: Structure with the local NT system time (see TIMESTRUCT).

VAR_IN_OUT

arrChannel: ARRAY [1..7] OF ST_BA_CalendarChannel;

arrChannel: Yearly scheduler; with the single-channel function block, the name of the variable is stChannel (see ST_BA_CalendarChannel [417]). Internally limited to the respective number of possible channels via the variable cBA_NumOfChannels.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3 BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
### DUTs

#### Enums

**Enumerations**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_BA_PosMod [412]</td>
<td>Enumerator for the definition of the positioning mode.</td>
</tr>
<tr>
<td>E_BA_ShdObjType [412]</td>
<td>Enumerator for selecting the shading object type.</td>
</tr>
<tr>
<td>E_BA_Sensor</td>
<td>Enumerator for selecting a sensor type for measuring analog values.</td>
</tr>
<tr>
<td>E_BA_Terminal_KL</td>
<td>Enumerator for selecting the respective Bus Terminal.</td>
</tr>
</tbody>
</table>

**E_BA_PosMod**

Enumerator for the definition of the positioning mode.

```plaintext
TYPE E_BA_PosMod :
{
  PosModFix:= 0,
  PosModTab,
  PosModMaxIndc
};
END_TYPE
```

**PosModFix**: The shutter height is a fixed value, which is set at function block FB_BA_SunPrtc [335] via the value IrFixPos [%].

**PosModTab**: The height positioning takes place with the help of a table of 6 interpolation points, 4 of which are parameterizable. A blind position in relation to the position of the sun is then calculated from these points by linear interpolation. For a more detailed description please refer to FB_BA_BldPosEntry [287].

**PosModMaxIndc**: The positioning takes place with specification of the maximum desired incidence of light.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**E_BA_ShdObjType**

Enumerator for selecting the shading object type.

```plaintext
TYPE E_BA_ShdObjType :
{
  ObjTypeTetragon := 0,
  ObjTypeGlobe
};
END_TYPE
```

**ObjTypeTetragon**: Object type is a rectangle.

**ObjTypeGlobe**: Object type is a ball.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_BA_BldPosTab</td>
<td>Structure of the interpolation point entries for the height adjustment of the blind.</td>
</tr>
<tr>
<td>ST_BA_Cnr</td>
<td>Information about window corners.</td>
</tr>
<tr>
<td>ST_BA_Fcd</td>
<td>Facade-specific data for activating the automatic functions.</td>
</tr>
<tr>
<td>ST_BA_FcdElem</td>
<td>List entry for a facade element (window).</td>
</tr>
<tr>
<td>ST_BA_ShdoObj</td>
<td>List entry for a shading object</td>
</tr>
<tr>
<td>ST_BA_SpRmT</td>
<td>Room temperature setpoints.</td>
</tr>
<tr>
<td>ST_BA_SunBld</td>
<td>Structure of the blind positioning telegram.</td>
</tr>
<tr>
<td>ST_BA_SunBldScn</td>
<td>Table entry for a blind scene.</td>
</tr>
<tr>
<td>ST_BA_CalendarChannel</td>
<td>Input of calendar entries.</td>
</tr>
<tr>
<td>ST_BA_SchedulerWeeklyChannel</td>
<td>Input of time switch entries.</td>
</tr>
</tbody>
</table>

**ST_BA_BldPosTab**

Structure of the interpolation point entries for the height adjustment of the blind.

```plaintext
TYPE ST_BA_BldPosTab:
STRUCT
  rSunElv   : ARRAY[0..5] OF REAL;
  rPos      : ARRAY[0..5] OF REAL;
  bVld      : BOOL;
END_STRUCT
END_TYPE
```

`rSunElv / rPos`: The 6 interpolation points that are transferred, wherein the array elements 0 and 5 represent the automatically generated edge elements mentioned above.

`bVld`: Validity flag for the function block `FB_BA_SunPrtc [335]`. It is set to TRUE by the function block `FB_BA_BldPosEntry [287]` if the data entered correspond to the validity criteria described.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**ST_BA_Cnr**

Information about window corners.

```plaintext
TYPE ST_BA_Cnr :
STRUCT
  rX   : REAL;
  rY   : REAL;
  bShdd : BOOL;
END_STRUCT
END_TYPE
```

`rX`: X-coordinate of the window (on the facade).

`rY`: Y-coordinate of the window (on the facade).

`bShdd`: Information as to whether this corner is in the shade: `bShdd=TRUE`: Corner is in the shade.
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**ST_BA_Fcd**

Facade-specific data at room level for activating the automatic functions.

```plaintext
TYPE ST_BA_Fcd:
  STRUCT
    rSunPrtcAngl : REAL;
    rSunPrtcPos  : REAL;
    rFcdThAutoPos : REAL;
    rFcdThAutoAngl : REAL;
    bFcdThAutoEn  : BOOL;
    bThAutoEn     : BOOL;
    bTwilLgtAutoEn : BOOL;
    bSunPrtcEn    : BOOL;
  END_STRUCT
END_TYPE
```

- **rSunPrtcAngl**: Sun protection: Current calculated position [%] for the blinds.
- **rSunPrtcPos**: Sun protection: Current calculated louvre angle [°] for the blinds.
- **rFcdThAutoPos**: Thermo-automatic function for whole facade: Currently valid position [%] for the blinds (heating or cooling position).
- **rFcdThAutoAngl**: Thermo-automatic function for whole facade: Currently valid louvre angle [°] for the blinds (heating or cooling position).
- **bFcdThAutoEn**: Thermo-automatic function for whole facade enabled.
- **bThAutoEn**: Thermo-automatic function enabled.
- **bTwilLgtAutoEn**: Automatic twilight function enabled.
- **bSunPrtcEn**: Automatic sun protection enabled.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**ST_BA_FcdElem**

List entry for a facade element (window).

```plaintext
TYPE ST_BA_FcdElem:
  STRUCT
    rWdwWdth  : REAL;
    rWdwHght  : REAL;
    stCnr     : ARRAY [1..4] OF ST_BA_Cnr;
    diGrp     : DINT;
    bVld      : BOOL;
  END_STRUCT
END_TYPE
```

- **rWdwWdth**: Width of the window.
- **rWdwHght**: Height of the window.
- **stCnr**: Coordinates of the window corners and information as to whether this corner point is in the shade; see `ST_BA_Cnr[413]`.
- **bVld**: Plausibility of the data entered: bVld=TRUE: Data are plausible.
diGrp: Group membership of the element.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**ST_BA_ShdObj**

List entry for a shading object.

```plaintext
TYPE ST_BA_ShdObj :
STRUCT
  rP1x : REAL;
  rP1y : REAL;
  rP1z : REAL;
  rP2x : REAL;
  rP2y : REAL;
  rP2z : REAL;
  rP3x : REAL;
  rP3y : REAL;
  rP3z : REAL;
  rP4x : REAL;
  rP4y : REAL;
  rP4z : REAL;
  rMx : REAL;
  rMy : REAL;
  rMz : REAL;
  rRads : REAL;
  diBegMth : UINT;
  diEndMth : UINT;
  eType : E_BA_ShdObjType;
  bVld : BOOL;
END_STRUCT
END_TYPE
```

- **rP1x .. rP4z**: Corner coordinates. Of importance only if the element is a square.
- **rMx .. rMz**: Center coordinates. Of importance only if the element is a ball.
- **rRads**: Radius of the ball. Of importance only if the element is a ball.
- **diBegMth**: Beginning of the shading period (month).
- **diEndMth**: End of the shading period (month).
- **eType**: Object type, see `E_BA_ShdObjType`.
- **bVld**: Plausibility of the data: `bVld=TRUE`: Data are plausible.

Remark about the shading period:

The entries for the months may not be 0 or greater than 12, otherwise all combinations are possible.

Examples:

- Start=1, End=1: shading in January.
- Start=1, End=5: shading from the beginning of January to the end of May.
- Start=11, End=5: shading from the beginning of November to the end of May (the following year).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
**ST_BA_SpRmT**

Room temperature setpoints.

```plaintext
TYPE ST_BA_SpRmT :
  STRUCT
    rPrtcHtg : REAL := 12.0;
    rEcoHtg  : REAL := 15.0;
    rPreCmfHtg : REAL := 19.0;
    rCmfHtg  : REAL := 21.0;
    rPrtcCol : REAL := 40.0;
    rEcoCol  : REAL := 35.0;
    rPreCmfCol : REAL := 28.0;
    rCmfCol  : REAL := 24.0;
  END_STRUCT
END_TYPE
```

The values in the structure are defined with the preset values.

The variables have the following meaning:

- **rPrtcHtg**: Protection Heating.
- **rEcoHtg**: Economy Heating.
- **rPreCmfHtg**: Pre-Comfort Heating.
- **rCmfHtg**: Comfort Heating.
- **rPrtcCol**: Protection Cooling.
- **rEcoCol**: Economy Cooling.
- **rPreCmfCol**: Pre-Comfort Cooling.
- **rCmfCol**: Comfort Cooling.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**ST_BA_Sunbld**

Structure of the blind positioning telegram.

```plaintext
TYPE ST_BA_SunBld:
  STRUCT
    rPos        : REAL;
    rAngl       : REAL;
    bManUp      : BOOL;
    bManDwn     : BOOL;
    bManMod     : BOOL;
    bActv       : BOOL;
  END_STRUCT
END_TYPE
```

- **rPos**: Transferred shutter height [%].
- **rAngl**: Transferred louvre position [°].
- **bManUp**: Manual command: blind up.
- **bManDwn**: Manual command: blind down.
- **bManMod**: TRUE: Manual mode is active. FALSE: Automatic mode is active.
- **bActv**: The sender of the telegram is active. This bit is only evaluated by the priority control FB_BA_SunBldPrioSwi4 [325] or FB_BA_SunBldPrioSwi8 [326]. The sun protection actuators FB_BA_SunBldActr [317] and FB_BA_RolBldActr [309] ignore it.
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**ST_BA_SunBldScn**

Table entry for a blind scene.

```
TYPE ST_BA_SunBldScn:
STRUCT
  rPos : REAL;
  rAngl : REAL;
END_STRUCT
END_TYPE
```

- **rPos**: Shutter height [%].
- **rAngl**: Louvre position [°].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**ST_BA_CalenderChannel**

Structure for entering calendar entries.

```
TYPE ST_BA_CalenderChannel:
STRUCT
  udiOn_Day : UDINT(1..31);
  udiOn_Month : UDINT(1..12);
  udiOn_hh : UDINT(0..23);
  udiOn_mm : UDINT(0..59);
  udiOff_Day : UDINT(1..31);
  udiOff_Month : UDINT(1..12);
  udiOff_hh : UDINT(0..23);
  udiOff_mm : UDINT(0..59);
  bEn : BOOL;
  bResetAfterOn : BOOL;
  bQ : BOOL;
END_STRUCT
END_TYPE
```

- **udiOn_Day**: Switch-on point for day.
- **udiOn_Month**: Switch-on point for month.
- **udiOn_hh**: Switch-on point for hour.
- **udiOn_mm**: Switch-on point for minute.
- **udiOff_Day**: Switch-off point for day.
- **udiOff_Month**: Switch-off point for month.
- **udiOff_hh**: Switch-off point for hour.
- **udiOff_mm**: Switch-off point for minute.
- **bEn**: TRUE -> Enable channel, FALSE -> bQ = FALSE
- **bResetAfterOn**: One-time and non-recurring switching-on.
- **bQ**: Channel status output.
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

**ST_BA_SchedulerWeeklyChannel**

Structure for entering time switch entries.

```plaintext
TYPE ST_BA_SchedulerWeeklyChannel:
  STRUCT
    udiOn_hh : UDINT(0..23);
    udiOn_mm : UDINT(0..59);
    udiOn_ss : UDINT(0..59);
    udiOff_hh : UDINT(0..23);
    udiOff_mm : UDINT(0..59);
    udiOff_ss : UDINT(0..59);
    bAllActive : BOOL;
    bEn      : BOOL;
    bMonday  : BOOL;
    bTuesday : BOOL;
    bWednesday : BOOL;
    bThursday : BOOL;
    bFriday  : BOOL;
    bSaturday : BOOL;
    bSunday  : BOOL;
    bResetAfterOn : BOOL;
    bQ       : BOOL;
  END_STRUCT
END_TYPE
```

- **udiOn_hh**: Switch-on point for hour.
- **udiOn_mm**: Switch-on point for minute.
- **udiOn_ss**: Switch-on point for second.
- **udiOff_hh**: Switch-off point for hour.
- **udiOff_mm**: Switch-off point for minute.
- **udiOff_ss**: Switch-off point for second.
- **bAllActive**: Activation of the timer condition for all weekdays.
- **bEn**: TRUE -> Enable channel, FALSE -> bQ = FALSE.
- **bMonday**: Switch-on point for Monday.
- **bTuesday**: Switch-on point for Tuesday.
- **bWednesday**: Switch-on point for Wednesday.
- **bThursday**: Switch-on point for Thursday.
- **bFriday**: Switch on point for Friday.
- **bSaturday**: Switch-on point for Saturday.
- **bSunday**: Switch-on point for Sunday.
- **bResetAfterOn**: One-time and non-recurring switching-on.
- **bQ**: Channel status output.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>
GVLs

Constants

Global constants

VAR_GLOBAL CONSTANT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>rClsZero</td>
<td>REAL</td>
<td>0.00001</td>
</tr>
<tr>
<td>udiNoActvPrio</td>
<td>UDINT</td>
<td>4294967295</td>
</tr>
<tr>
<td>udiTiSec</td>
<td>UDINT</td>
<td>4294967295</td>
</tr>
<tr>
<td>wSUNDAY</td>
<td>WORD</td>
<td>0</td>
</tr>
<tr>
<td>wMONDAY</td>
<td>WORD</td>
<td>1</td>
</tr>
<tr>
<td>wTUESDAY</td>
<td>WORD</td>
<td>2</td>
</tr>
<tr>
<td>wWEDNESDAY</td>
<td>WORD</td>
<td>3</td>
</tr>
<tr>
<td>wTHURSDAY</td>
<td>WORD</td>
<td>4</td>
</tr>
<tr>
<td>wFRIDAY</td>
<td>WORD</td>
<td>5</td>
</tr>
<tr>
<td>wSATURDAY</td>
<td>WORD</td>
<td>6</td>
</tr>
<tr>
<td>TimeValue24h_ms</td>
<td>UDINT</td>
<td>864000000</td>
</tr>
</tbody>
</table>

END_VAR

rClsZero: Reference value to avoid division by zero.

udiNoActvPrio: The value of the constants indicates that no priority is active.

udiTiSec: Constant for specifying a time in seconds.

wSUNDAY: Constant value for Sunday.

wMONDAY: Constant value for Monday.

wTUESDAY: Constant value for Tuesday.

wWEDNESDAY: Constant value for Wednesday.

wTHURSDAY: Constant value for Thursday.

wFRIDAY: Constant value for Friday.

wSATURDAY: Constant value for Saturday.

TimeValue24h_ms: Time value for 24 hours in milliseconds.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

Parameter

Global parameters

VAR_GLOBAL CONSTANT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>usiMaxSunBldScn</td>
<td>USINT</td>
<td>20</td>
</tr>
<tr>
<td>uiMaxRowFcd</td>
<td>UINT</td>
<td>10</td>
</tr>
<tr>
<td>uiMaxColumnFcd</td>
<td>UINT</td>
<td>20</td>
</tr>
<tr>
<td>uiMaxShdObj</td>
<td>UINT</td>
<td>20</td>
</tr>
<tr>
<td>uiMaxDataFileSize</td>
<td>UDINT</td>
<td>1000000</td>
</tr>
</tbody>
</table>

END_VAR

usiMaxSunBldScn: Maximum number of scenes that are processed by the function block FB BA_SunBldScn [327].

uiMaxRowFcd: Maximum number of floors for which the shading correction applies (horizontal arrangement of windows).
uiMaxColumnFcd: Maximum number of axes for which the shading correction applies (vertical arrangement of windows).

uiMaxShdObj: Maximum number of shading objects that cast shadows on the facade.

udiMaxDataFileSize: Maximum file size for the Excel list (in bytes), which is read by the function blocks FB_BA_RdFcdElemLst [301] and FB_BA_RdShdObjLst [305].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4024.7</td>
<td>Tc3_BA from v1.1.6.0</td>
</tr>
</tbody>
</table>

7.1.1.2.2 Tc2_BA

This library was imported from TwinCAT 2 because of its proven functionalities from the TS8040 supplement. Therefore, the documentation is identical (see Tc BA).

7.1.2 SPS Project Templates

7.1.2.1 Standard PLC BA template

The standard PLC BA Template is a PLC template for a standard TF8040 project.

The standard PLC BA template is used to load all the libraries and project settings required to make a start with TF8040. The template is very suitable for making a start with a TF8040 project.

Structure

The standard project template contains all the declarations, FB calls and libraries required for the initial commissioning of a TF8040 controller.

Template for a building automation PLC Project with the following basic content:

Settings

- Task
  - PLC cycle time: 45 ms

References

- Tc3_BA2
- Tc3_BA2_Common
- Tc3_BACnetRev14

Programs

- MAIN
  - FB_BA_DPAD
    Contains preconfigured layers for small projects.
  - Project structure
    Contains declarations and FB calls for:
    - Control cabinet
    - device

Templates

Contains all the templates required for compiling the project template without errors.
Structure in the solution tree

All the libraries required for a TF8040 project are loaded here.

- **References**
  - Tc2_MDP
  - Tc2_Standard
  - Tc2_SUPS
  - Tc2_System
  - Tc2.Utilities
  - Tc3_BA2
  - Tc3_BA2_Common
  - Tc3_BACnetRev14
  - Tc3_Module

- **MAIN POU**

The standard project structure is called in the MAIN POU.

---

This template provides a project structure as an application example with 8 levels according to an example from VDI 3814.
• Templates

All templates required to get started are located in this folder. They enable error-free compilation of the PLC project template. Further templates for the realization of project-specific applications can be found in the Template Repository [532].

• PLC task

The cycle time of the PLC task is set to 45 ms and should not be set smaller for performance reasons.
Project-specific programming

• Control

Once the project can be compiled without errors, the preparation is complete.

• Importing templates

Project-specific programming begins with the integration of templates from the Template Repository [532].

Now start with the project-specific programming.

7.2 HMI

7.2.1 TcHmiBa

7.2.1.1 Framework

The framework contains various classes, enumerations, interfaces and types that are used to create the controls [445]. Various helper methods facilitate server-client communication, access to different content and positioning in a control. The package contains no controls, because it is intended for use in framework control projects.

The framework is written in TypeScript.

If the complete solution TF8040 [8] is used, the framework also handles a range of management functions in order to realize the generics [15].

Installation

In order to use the framework, the NuGet package Beckhoff.TwinCAT.HMI.BA.Framework must be installed.

Additional functions are required, which are included in the following NuGet packages, which are installed automatically:

• Beckhoff.TwinCAT.HMI.BA.Icons [424]
• Beckhoff.TwinCAT.HMI.Framework

Ensure that the files from the framework are loaded in the project. When used in a TcHmi framework project, the appropriate packages must be entered in Manifest.json.
IntelliSense support for the framework in Visual Studio can be achieved by adding the following entries to `tsconfig.tpl.json`:

```json
{
  "strictFunctionTypes": false,
  "strictPropertyInitialization": false,
  "alwaysStrict": false,
  "include": [
    "$(Beckhoff.TwinCAT.HMI.Framework).InstallPath/Tchmi.d.ts",
    "$(Beckhoff.TwinCAT.HMI.BA.Icons).InstallPath/Source.d.ts",
    "$(Beckhoff.TwinCAT.HMI.BA.Framework).InstallPath/TchmiBaFramework.d.ts"
  ]
}
```

### 7.2.1.2 Icons

TcHmiBa contains various icons that are necessary for the realization of visualizations for building automation. The icons are created in `*.svg` format and are intended for use on the web.

**Installation**

In order to use the icons, the NuGet package `Beckhoff.TwinCAT.HMI.BA.Icons` must be installed.
Since the Beckhoff.TwinCAT.HMI.Framework is required, this is also installed.

**Use**

There are three different application options.

**ZIP archive**

A NuGet package is basically a ZIP archive that allows direct use of the icons after unpacking. The unpacked content looks like this:

```
rel

Icons

package
runtimes
tools
[Content_Types].xml
Beckhoff.TwinCAT.HMI.BA.Icons.nuspec
READMe.txt
```

Only the folder *Icons* is relevant. It contains the various icons divided according to content categories.

**GalleryExplorer**

After installing the NuGet package in a TwinCAT HMI project, the icons integrate into the GalleryExplorer.
The icons here are arranged according to function. To use an icon from the GalleryExplorer, drag and drop it into a folder in the project.

- The icon is created as a copy in the project directory, not as a reference.

**Use as reference**

When using the icons as reference, their extended functionalities can be used. In addition, the icons benefit directly from updates to the NuGet package.

**TcHmi project**

The function `TcHmi.Functions.GetBalconPath()` can be used to reference the icons in a TcHmi project.
The following steps are required:

1. Drag a control (e.g. a button) to the content/view.
2. Create the **Function Binding**.

![Function Binding](image)

3. In the **Backgroundimage** field, enter **GetBalconPath** (function is suggested).

![Backgroundimage](image)

4. Enter the path to the icon, e.g. "HVAC/fan" (note the quotation marks).

5. Set the size and position.

   The path information can be taken from the structure of the *GalleryExplorer*.

   - The button should then look like this:

   ![Button](image)

The extended icon functions can only be used with controls from the NuGet package *Beckhoff.TwinCAT.HMI.BA.Controls*.

The controls from the *Icon* category have extended setting options.

As an example, the button from the category **BA | Base** is used in the following section.

![Icon](image)

The Icon attribute can again be set using the function *TcHmi.Functions.GetBalconPath()*.

This form of embedding allows the icon to change dynamically. The following attributes are available:

- IconRotation
In the code

For using the icons in the code, e.g. when developing framework controls, the icon paths can be accessed even more easily. There are constants in the namespace `TcHmi.BuildingAutomation.Icons` that point to the respective icons in the NuGet package (e.g. `TcHmi.BuildingAutomation.Icons.HVAC.fan.svg`).

HVAC symbols

Symbols for P&I diagrams.

- The icons are drawn with appropriate size ratios for a P&I diagram and are only shown differently in this listing.

Event symbols

Symbols to represent alarms, events or notifications.
Event

Flag

Lock

Priority

Object Type
Symbols for the different object types.
Room Automation
Symbols for room automation.

Standard
Standard symbols for visualizations.
7.2.1.3 Controls

The following pages describe the content and functions of the NuGet package Beckhoff.TwinCAT.HMI.BA.Controls. The main focus is on the attributes and functions of the controls that can be used in the Designer.

System requirements

- TF8040 (current version)
- TE2000
- Google Chrome / Chromium (support for other browsers to follow)

Installation

The NuGet package Beckhoff.TwinCAT.HMI.BA.Controls must be installed in order to use the controls and functions.
The package depends on the following packages and therefore installs them as well:

- Beckhoff.TwinCAT.HMI.Framework
- Beckhoff.TwinCAT.HMI.Controls
- Beckhoff.TwinCAT.HMI.BA.Framework [423]
- Beckhoff.TwinCAT.HMI.BA.Icons [424]

Contents

The content includes controls [445] and templates.

7.2.1.3 Basis Controls

The BasisControls are controls that are not included in the toolbox. They perform various background functions and have attributes that are inherited by other controls.

BaseControl

Description

The BaseControl passes on the generic functionalities [61] to controls that inherit from them. It thus forms the interface between the server extension [508] and the client. In addition, it contains properties that other controls also require, thus reducing redundant implementations.

Use

The BaseControl is only used for inheritance and is therefore not available in the toolbox.
Special features

The BaseControl implements functionalities that run in the background and deal with the management of various tasks. These include, for example:

- Busy handling
- Logging off various watches
- Handling of BaObjects from the TcHMiBaServerExtension [508]

Attributes

The control inherits from TcHmiControl and thus has the same attributes. In addition, there are the following attributes.

BA

BaObject
tchmi:framework#/definitions/Symbol

To use the generic functionalities of TcHmiBa (see Symbol). It links a single object or a complete view (including children) to the control.

More detailed information about the generic capabilities of TcHmiBa can be found here.

The attribute is not applicable to all controls.
Common

ReadOnly
tcimi:general#/definitions/Boolean

Determines whether the user has read-only or write access (see Boolean).

- The attribute is not applicable to all controls.

Layout

ContentPadding
tcimi:framework#/definitions/Padding

Sets the padding for the content of the control (see Padding).

- The attribute is not applicable to all controls.

Enumerations

DateTimeChoice

"TcHmi.BuildingAutomation.DateTimeChoice": {
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "DateTimeChoice",
  "type": "integer",
  "enum": [0, 1, 2],
  "options": [
    { "label": "Date", "value": 0 },
    { "label": "Time", "value": 1 },
    { "label": "DateTime", "value": 2 }
  ]
}

PositionEx

"TcHmi.BuildingAutomation.PositionEx": {
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "PositionEx",
  "type": "string",
  "enum": [0, 1],
}
Position

"TcHmi.BuildingAutomation.Position": {
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Position",
  "type": "string",
  "enum": [0, 1, 2, 3],
  "options": [
    { "label": "left", "value": 0 },
    { "label": "top", "value": 1 },
    { "label": "right", "value": 2 },
    { "label": "bottom", "value": 3 }
  ]
}
Direction

"TcHmi.BuildingAutomation.Direction": {
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "string",
  "enum": [0, 1],
  "options": [
    {"label": "left", "value": 0 },
    {"label": "right", "value": 1 }
  ]
}

Orientation

"TcHmi.BuildingAutomation.Orientation": {
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "string",
  "enum": [0, 1],
  "options": [
    {"label": "Vertical", "value": 0 },
    {"label": "Horizontal", "value": 1 }
  ]
}

Types

FourSidedColor

Scheme for attributes that define colors for four sides.

"TcHmi.BuildingAutomation.FourSidedColor": {
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "object",
  "title": "FourSidedColor",
  "propertiesMeta": [
    {"name": "left", "displayName": "Left", "defaultValue": null, "defaultValueInternal": null },
    {"name": "top", "displayName": "Top", "defaultValue": null, "defaultValueInternal": null },
    {"name": "right", "displayName": "Right", "defaultValue": null, "defaultValueInternal": null },
    {"name": "bottom",}]}
NumberOrBoolean

Scheme for attributes that can have number or boolean as data type.

```
"TcHmi.BuildingAutomation.NumberOrBoolean": {
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": [ "number", "boolean" ]
}
```

ICoordinateXY

Scheme for an xy-coordinate pair.

```
"TcHmi.BuildingAutomation.ICoordinateXY": {
  "type": "object",
  "title": "ICoordinateXY",
  "description": "Defines a XY coordinate",
  "engineeringColumns": [ "x", "y" ],
  "propertiesMeta": [
    {
      "name": "x",
      "category": "Common",
      "displayName": "X Value"
    },
    {
      "name": "y",
      "category": "Common",
      "displayName": "Y Value"
    }
  ],
  "properties": {
    "x": {
      "type": "number"
    },
    "y": {
      "type": "number"
    }
  },
  "required": [ "x", "y" ]
}
```

DialogWindowOptions

Scheme for editing the options for a dialog box.

```
"TcHmi.BuildingAutomation.DialogWindowOptions": {
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "object",
  "propertiesMeta": [
  ]
}
```
"name": "content",
"displayName": "Content",
"description": "The content that should be displayed in the dialog window.",
"defaultValue": null
},

"name": "buttons",
"displayName": "Buttons",
"description": "Buttons for the dialog.",
"defaultValue": 0
},

"name": "modal",
"displayName": "Modal",
"description": "Selects if the dialog is opened modal or not.",
"defaultValue": true
},

"name": "headline",
"displayName": "Headline",
"description": "The headline of the dialog.",
"defaultValue": "Dialog headline"
},

"name": "width",
"displayName": "Width",
"defaultValue": 1000,
"defaultValueInternal": 1000
},

"name": "widthUnit",
"displayName": "Width unit",
"refTo": "width",
"defaultValue": "px",
"defaultValueInternal": "px"
},

"name": "height",
"displayName": "Height",
"defaultValue": 500,
"defaultValueInternal": 500
},

"name": "heightUnit",
"displayName": "Height unit",
"refTo": "height",
"defaultValue": "px",
"defaultValueInternal": "px"
}
],

"properties": {
"content": {
"$ref": "tchmi:framework#/definitions/ContentPath"
},

"buttons": {
"$ref": "tchmi:framework#/definitions/TcHmi.BuildingAutomation.Controls.DialogWindowButtons"
},

"modal": {
"type": "boolean"
},

"headline": {
"type": "string"
},

"width": {
"type": "number"
},

"widthUnit": {
"$ref": "tchmi:framework#/definitions/MeasurementUnit"
},

"height": {
"type": "number"
},

"heightUnit": {
"$ref": "tchmi:framework#/definitions/MeasurementUnit"
}
UserContents

Scheme for an editor entry that assigns a content to a user name.

"TcHmi.BuildingAutomation.UserContents": {
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "array",
  "items": {
    "type": "object",
    "title": "UserContent",
    "description": "Defines a relation between a user and a content.",
    "engineeringColumns": [
      "userName",
      "content"
    ],
    "propertiesMeta": [
      { "name": "userName",
        "displayName": "UserName",
        "defaultValue": null,
        "defaultValueInternal": null
      },
      { "name": "content",
        "displayName": "Content",
        "defaultValue": null,
        "defaultValueInternal": null
      }
    ],
    "properties": {
      "userName": { "type": "string" },
      "content": { "$ref": "tchmi:framework#/definitions/ContentPath" }
    },
    "required": [ "userName", "content" ]
  }
}

DateTime

BaDateTime

Scheme for time and date in BA format.

"TcHmi.BuildingAutomation.BaDateTime": {
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "object",
  "propertiesMeta": [
    { "name": "stDate",
      "defaultValue": null,
      "defaultValueInternal": null
    },
    { "name": "stTime",
      "defaultValue": null,
      "defaultValueInternal": null
    }
  ],
  "properties": {
    "stDate": { "$ref": "tchmi:framework#/definitions/TcHmi.BuildingAutomation.BaDate" },
    "stTime": { "$ref": "tchmi:framework#/definitions/TcHmi.BuildingAutomation.BaTime" }
  }
}
**BaDate**

Scheme for a date in BA format.

```
"TcHmi.BuildingAutomation.BaDate": {  
  "$schema": "http://json-schema.org/draft-04/schema",  
  "type": "object",  
  "propertiesMeta": [  
    {  
      "name": "eDayOfWeek",  
      "defaultValue": null,  
      "defaultValueInternal": null  
    },  
    {  
      "name": "eMonth",  
      "defaultValue": null,  
      "defaultValueInternal": null  
    },  
    {  
      "name": "nDay",  
      "defaultValue": null,  
      "defaultValueInternal": null  
    },  
    {  
      "name": "nYear",  
      "defaultValue": null,  
      "defaultValueInternal": null  
    }  
  ],  
  "properties": {  
    "eDayOfWeek": {  
      "$ref": "tchmi:framework#/definitions/TcHmi.BuildingAutomation.Weekday"  
    },  
    "eMonth": {  
      "$ref": "tchmi:framework#/definitions/TcHmi.BuildingAutomation.Month"  
    },  
    "nDay": {  
      "type": "number"  
    },  
    "nYear": {  
      "type": "number"  
    }  
  }  
}
```

**BaTime**

Scheme for a time in BA format.

```
"TcHmi.BuildingAutomation.BaTime": {  
  "$schema": "http://json-schema.org/draft-04/schema",  
  "type": "object",  
  "propertiesMeta": [  
    {  
      "name": "nHour",  
      "defaultValue": null,  
      "defaultValueInternal": null  
    },  
    {  
      "name": "nMinute",  
      "defaultValue": null,  
      "defaultValueInternal": null  
    },  
    {  
      "name": "nSecond",  
      "defaultValue": null,  
      "defaultValueInternal": null  
    }  
  ],  
  "properties": {  
    "nHour": {  
      "type": "number"  
    },  
    "nMinute": {  
      "type": "number"  
    },  
    "nSecond": {  
      "number"  
    }  
  }  
}
```
"type": "number"
}
]

Month

Enumeration for a date specification in BACnet format.

"TcHmi.BuildingAutomation.Month": {
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "string",
  "enum": [0, 255, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
  "options": [
    {"label": "Unspecified", "value": 255},
    {"label": "eJanuary", "value": 1},
    {"label": "eFebruary", "value": 2},
    {"label": "eMarch", "value": 3},
    {"label": "eApril", "value": 4},
    {"label": "eMay", "value": 5},
    {"label": "eJune", "value": 6},
    {"label": "eJuly", "value": 7},
    {"label": "eAugust", "value": 8},
    {"label": "eSeptember", "value": 9},
    {"label": "eOctober", "value": 10},
    {"label": "eNovember", "value": 11}]
}
Weekday

Scheme for specifying a day of the week in BACnet format.

```json
"TcHmi.BuildingAutomation.Weekday": {
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "string",
  "enum": [255, 1, 2, 3, 4, 5, 6, 7],
  "options": [
    {
      "label": "Unspecified",
      "value": 255
    },
    {
      "label": "eMonday",
      "value": 1
    },
    {
      "label": "eTuesday",
      "value": 2
    },
    {
      "label": "eWednesday",
      "value": 3
    },
    {
      "label": "eThursday",
      "value": 4
    },
    {
      "label": "eFriday",
      "value": 5
    },
    {
      "label": "eSaturday",
      "value": 6
    },
    {
      "label": "eSunday",
      "value": 7
    }
  ]
}
```

TextControl

The TextControl offers various attributes, all of which apply to text manipulation.

Use

This is only used for inheritance and is therefore not available in the toolbox.

Special features

The following text manipulations are possible:
- Change position horizontally and vertically.
• Set font type, size and thickness.
• Apply various styles to the text (e.g. underline).
• Define how the text should be displayed if the available space is not sufficient.

**Attributes**

The control inherits from `BaseControl` and thus has the same attributes. In addition, there are the following attributes.

**Colors**

**TextColor**

tchmi:framework#/definitions/SolidColor

Text colors (see `SolidColor`).

**TextDecorationColor**

tchmi:framework#/definitions/SolidColor

Color of text decorations (see `SolidColor`).

**Text**

**TextVerticalAlignment**

tchmi:framework#/definitions/VerticalAlignment

Vertical alignment of texts (see `VerticalAlignment`).

**TextHorizontalAlignment**

tchmi:framework#/definitions/HorizontalAlignment

Horizontal alignment of texts (see `HorizontalAlignment`).

**TextFontSize**

tchmi:framework#/definitions/MeasurementValue

Font size of texts. Percentages are relative to the font size of the parent element (see `number`).

**TextFontSizeUnit**

tchmi:framework#/definitions/measurementUnit

Unit for the font size of texts. Can be absolute (px) or relative (%) (see `DimensionUnit`).

**TextFontFamily**

tchmi:framework#/definitions/FontFamily

Font of texts (see `FontFamily`).

**TextFontStyle**

tchmi:framework#/definitions/FontStyle

Font style of texts (see `FontStyle`).

**TextFontWeight**

tchmi:framework#/definitions/FontWeight

Font weight of texts (see `FontWeight`).

**TextDecorationLine**


Position of the text decoration (see `TextDecorationLine`).
TextDecorationStyle


Text decoration style (see TextDecorationStyle [444]).

UserSelect


Behavior when selecting the text of a user (see UserSelect [444]).

TextUtils


Defines how to display text that is wider than the control (see TextOverflow [444]).

Enumerations

TextDecorationLine

Scheme for the attribute TextDecorationLine of TextControls.

  "$schema": "http://json-schema.org/draft-04/schema",  
  "type": "string",  
  "enum": [ "none", "underline", "overline", "line-through", "initial", "inherit" ]
}

TextDecorationStyle

Scheme for the attribute TextDecorationStyle of TextControls.

  "$schema": "http://json-schema.org/draft-04/schema",  
  "type": "string",  
  "enum": [ "solid", "double", "dotted", "dashed", "wavy", "initial", "inherit" ]
}

UserSelect

Scheme for the UserSelect attribute of TextControls.

  "$schema": "http://json-schema.org/draft-04/schema",  
  "type": "string",  
  "enum": [ "auto", "none", "text", "all" ]
}

TextOverflow

Scheme for the attribute TextOverflow of TextControls.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>The text is not truncated and is displayed beyond the control.</td>
</tr>
<tr>
<td>ellipse</td>
<td>The text is truncated (e.g. &quot;A long te...&quot;).</td>
</tr>
<tr>
<td>marquee</td>
<td>The text moves back and forth in the available space (scrolling).</td>
</tr>
</tbody>
</table>

  "$schema": "http://json-schema.org/draft-04/schema",  
  "type": "integer",  
  "enum": [ 0, 1, ]
}
7.2.1.3.2 Controls

The controls in this category are all included in the toolbox.

BuildingInformation

The BuildingInformation control initially appears like a normal button [447]. It is designed to display various building or facade specific information.

Use

Use on any page (e.g. in the header).

Features

If the window is not open, the amount of relevant information is displayed in the corner of the button. This includes:

- Fire alarm
- Burglar alarm
- Ice alarm
- Maintenance
- Storm alarm
The status of the sun protection or the thermal automatic can only be viewed when the window is open.

When an alarm becomes active, all controls that depend on building information are notified. This applies to:

- **Sunblind** [499]
- **Window** [504]

**Attributes**

The control inherits from the button [447] and thus has the same attributes. In addition, there are the following attributes.

**Common**

**FireAlarm**

tchmi:general#/definitions/Boolean

If TRUE, the fire alarm is displayed.

**BurglarAlarm**

tchmi:general#/definitions/Boolean

If TRUE, the burglar alarm is displayed.

**IceAlarm**

tchmi:general#/definitions/Boolean

If TRUE, the ice alarm is displayed.

**ThermalAutomatic**

tchmi:general#/definitions/Boolean

If TRUE, the thermal automatic is displayed.

**Facades**


Create different facades that can then be referenced by corresponding controls (see above).

**Types**

**Facades**

Collection of **FacadeInfos** [447].

```json
"TcHmi.BuildingAutomation.Controls.BuildingInformation.Facades": {
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "array",
  "title": "Facades",
  "items": {
    "$ref": "tchmi:framework#/definitions/TcHmi.BuildingAutomation.Controls.BuildingInformation.FacadeInfo"
  }
}
```
FacadeInfo

Scheme for an editor that allows the creation of facade information for the control BuildingInformation

```
  "$schema": "http://json-schema.org/draft-04/schema",  
  "type": "object",  
  "title": "FacadeInfo",  
  "description": "Defines a facade",  
  "engineeringColumns": [  
    "name"  
  ],  
  "propertiesMeta": [  
    {  
      "name": "name",  
      "category": "Common",  
      "displayName": "Name",  
      "description": "The name of the facade (e.g. west).",  
      "defaultValue": null,  
      "defaultValueInternal": null  
    },  
    {  
      "name": "maintenance",  
      "category": "Common",  
      "displayName": "Maintenance",  
      "description": "Flag if maintenance is active.",  
      "defaultValue": null,  
      "defaultValueInternal": null  
    },  
    {  
      "name": "stormProtection",  
      "category": "Common",  
      "displayName": "Storm Protection",  
      "description": "Flag if storm protection is active.",  
      "defaultValue": null,  
      "defaultValueInternal": null  
    },  
    {  
      "name": "sunProtection",  
      "category": "Common",  
      "displayName": "Sun Protection",  
      "description": "Flag if sun protection is active.",  
      "defaultValue": null,  
      "defaultValueInternal": null  
    }  
  ],  
  "properties": {  
    "name": {  
      "type": "string"  
    },  
    "maintenance": {  
      "type": "boolean"  
    },  
    "stormProtection": {  
      "type": "boolean"  
    },  
    "sunProtection": {  
      "type": "boolean"  
    }  
  }  
}
```

Button

Description

The button is essentially the same as the TcHmiButton. The only difference is extended functionalities for the icon, because the options of the icon package can be used here.
Use
Can be used wherever a button with extended icon functionality is required.

Special features
Provides advanced functionality for icons from the NuGet package TcHmiBalcons.

Attributes
The control inherits from BaseControl and thus has the same attributes. In addition, there are the following attributes.

Icon
Path to the icon.

IconWidth
tchmi:general#/definitions/Number
Width of the icon (see Number).

IconWidthUnit
tchmi:general#/definitions/MeasurementUnit
Unit of the icon width (see DimensionUnit).

IconHeight
tchmi:general#/definitions/Number
Height of the icon (see Number).

IconHeightUnit
tchmi:general#/definitions/MeasurementUnit
Unit of the icon height (see DimensionUnit).

IconHorizontalAlignment
tchmi:general#/definitions/HorizontalAlignment
Definition of the horizontal alignment of the icon within the button (see HorizontalAlignment).

IconVerticalAlignment
tchmi:general#/definitions/VerticalAlignment
Definition of the vertical alignment of the icon within the button (see VerticalAlignment).

IconRotation
tchmi:general#/definitions/Number
Determines by how many degrees the icon should be rotated (see Number).
IconRotationSpeed

tchmi:general#/definitions/Number

Sets the speed at which the icon should rotate (see Number).

IconRotationDirection

tchmi:general#/definitions/TcHmi.BuildingAutomation.Controls.Direction

Sets the direction in which the icon rotates if the attribute IconRotationSpeed has been defined. The default value is clockwise. (see Missing).

Calendar

The control Calendar is used to display and manage exceptions to a schedule and select a date.

Use

Use on any page where a date is to be selected.

If a Schedule object is passed to the BaObject attribute in the EventCalendar display type, the exceptions of a time schedule can also be displayed or edited.

Features

Provides two different display modes, the ability to manage exceptions and a color highlighting of related exceptions when the mouse pointer is over them.

DatePicker

A space-saving view of a calendar. Returns the selected date via an event.

EventCalendar

An event-oriented view of a calendar. It returns the selected date via an event and offers the option to manage an exception by clicking on it.
Menu

In the upper right-hand corner of the calendar there is a button that provides a menu with further actions.

- Today: jumps to the current date.
- Show/Hide: shows or hides exceptions.
- Add: adds local exceptions.
- Reset: discards all unconfirmed changes.

Local exceptions

Entries in the aException collection of a Schedule object (e.g. FB_BA_SchedM) are regarded as local exceptions.
In the upper area, you can set the date or the repetition type of the local exception. Below this, the time periods can be defined with the applicable values.

The automatic numbering of the local exceptions starts at 1.

**Global exceptions (calendar reference)**

Entries in the `aCalendar` collection of a Schedule object (e.g. FB_BA_SchedM) are regarded as global exceptions.

For global exceptions, only the time periods and values to be applied can be defined. The date or the repetition type must be configured in the referenced Calendar object (e.g. FB_BA_Cal).

The automatic numbering of the global exceptions starts at 100.

**Attributes**

The control inherits from `BaseControl` and thus has the same attributes. In addition, there are the following attributes.

**Common**

**DisplayMode**

Defines the display type of the calendar.

**Events**

**onDateChanged**

Returns the selected date.
Checkbox

Description
The checkbox shows or edits binary values.

Use
Can be used on any page where binary values are to be displayed or edited.

Special features
The active and inactive text can be set (e.g. "On" / "Off").

The appearance can be adjusted via the attribute Appearance (see image above).

Ability to link a BaObject so that only a single binding has to be created. All the required attributes are then linked via this binding and changes to the value are automatically written back to the PLC.

Attributes
The control inherits from TextControl and thus has the same attributes. In addition, there are the following attributes.

Feedback concept
The control can use the feedback concept (see 15).

Common

State
tchmi:general#/definitions/Boolean
Status of the checkbox (see Boolean).

UserStateFeedback
tchmi:general#/definitions/Boolean
Determines whether or not the attribute StateFeedback is observed (see Boolean).

StateFeedback
tchmi:general#/definitions/Boolean
Feedback for the state of the checkbox (see Boolean).

ActiveText
tchmi:general#/definitions/String
Specifies the text that is displayed when State is TRUE (see String).
InactiveText
tchmi:general#/definitions/String

Specifies the text that is displayed when State is FALSE (see String).

Appearance

Determines how the checkbox appears at the top (see Appearance [453]).

Colors

CheckBackgroundColor
tchmi:general#/definitions/SolidColor

Background color of the checkbox if State is TRUE (see SolidColor).

This attribute has no effect if Appearance [453] is set to ToggleSlider.

CheckmarkColor
tchmi:general#/definitions/SolidColor

Color of the checkmark or toggle (see SolidColor).

Events

onStateChanged

Triggered when the value of State has changed.

Enumerations

Appearance

Scheme for the Appearance [453] attribute of the checkbox.

```
"$schema": "http://json-schema.org/draft-04/schema",
"type": "string",
"enum": [ 
  0,
  1,
  2
 ],
"options": [
  { 
    "label": "checkbox",
    "value": 0
  },
  { 
    "label": "toggleSwitch",
    "value": 1
  }
];
```
ColorPicker

Description
The ColorPicker can be used to select a color from various color palettes.

Use
Can be used on any page where it is necessary to select a color.

Special features
Color selection can be done using different color spaces.

HSL Ring
All colors are selected without shading.

HSV Circle
Selection of all colors up to white.
HSL Rect All color
Selection of all colors up to black.

HSL Rect single
Selection of all shades of a color. To change the color, change the attribute BackgroundColor.

Attributes
The control inherits from TcHmiControl and thus has the same attributes. In addition, there are the following attributes.

**ColorPlateType**
tchmi:framework#/definitions/TcHmi.BuildingAutomation.Controls.ColorPlateType
Defines the color palette used (see ColorPlateType [456]).

**BackgroundColor**
tchmi:framework#/definitions/SolidColor
Sets the background color when the HslRect1 color palette is selected (see SolidColor).

**SelectedSolidColor** (read-only)
tchmi:framework#/definitions/SolidColor
Selected color (see SolidColor).

**SelectedRgbaColor** (read-only)
tchmi:framework#/definitions/TcHmi.BuildingAutomation.RGBAColor
Selected color in RGBA format (see RGBAColor [457]).

Events

**onSelectedColorChanged**
Triggered when the selected color has changed.
Enumerations

ColorPlateType

Scheme for the attribute ColorPlateType [455] of the ColorPicker.

```
"TcHmi.BuildingAutomation.Controls.ColorPicker.ColorPlateType": {
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "string",
  "enum": [ 0, 1, 2, 3 ],
  "options": [
    { "label": "hslRect1", "value": 0 },
    { "label": "hslRectAll", "value": 1 },
    { "label": "hsvCircle", "value": 2 },
    { "label": "hslRing", "value": 3 }
  ]
}
```

Types

HSLColor

Scheme for a color in HSL format.

```
"TcHmi.BuildingAutomation.HSLColor": {
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "object",
  "title": "HSLColor",
  "propertiesMeta": [
    { "name": "h", "category": "Common", "displayName": "Hue", "displayPriority": 10, "description": "Hue 0-1", "defaultValue": null, "defaultValueInternal": null },
    { "name": "s", "category": "Common", "displayName": "Saturation", "displayPriority": 10, "description": "Saturation 0-1", "defaultValue": null, "defaultValueInternal": null },
    { "name": "l", "category": "Common", "displayName": "Lightness", "displayPriority": 10, "description": "Lightness 0-1", "defaultValue": null, "defaultValueInternal": null }
  ],
  "properties": {
    "h": {
      "type": "number"
    }
  }
}
```
RGBAColor

Scheme for a color in RGBA format.

"TcHmi.BuildingAutomation.RGBAColor": {
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "object",
  "title": "RGBAColor",
  "propertiesMeta": [
    {
      "name": "r",
      "category": "Common",
      "displayName": "Red",
      "displayPriority": 10,
      "description": "Red 0-255",
      "defaultValue": null,
      "defaultValueInternal": null
    },
    {
      "name": "g",
      "category": "Common",
      "displayName": "Green",
      "displayPriority": 10,
      "description": "Green 0-255",
      "defaultValue": null,
      "defaultValueInternal": null
    },
    {
      "name": "b",
      "category": "Common",
      "displayName": "Blue",
      "displayPriority": 10,
      "description": "Blue 0-255",
      "defaultValue": null,
      "defaultValueInternal": null
    },
    {
      "name": "a",
      "category": "Common",
      "displayName": "Alpha",
      "displayPriority": 10,
      "description": "Alpha 0-1",
      "defaultValue": null,
      "defaultValueInternal": null
    }
  ],
  "properties": {
    "r": {
      "type": "number"
    },
    "g": {
      "type": "number"
    },
    "b": {
      "type": "number"
    },
    "a": {
      "type": "number"
    }
  }
}

Combo box

The combo box displays or edits multistate values.
Use

Can be used on any page where multistate values are to be displayed or edited.

Special features

Ability to link a BaObject [433] so that only a single binding has to be created. All the required attributes are then linked via this binding and changes to the value are automatically written back to the PLC.

Attributes

The control inherits from TextControl [442] and thus has the same attributes. In addition, there are the following attributes.

Feedback concept

The control can use the feedback concept [15].

Common

Data

Data for the combo box (see ComboboxItems [459]).

SelectedData (read-only)

Currently selected data (see ComboboxItem [459]).

SelectedValue

Value of the currently selected data (see number).

UseSelectedValueFeedback

Determines whether or not the attribute SelectedValueFeedback is observed (see boolean).

SelectedValueFeedback

Feedback for the selected value (see number).

Colors

ButtonColor

Color of the button that opens the dropdown list (see SolidColor).
**ButtonArrowColor**

tchmi:framework#/definitions/SolidColor

Color of the arrow in the button (see SolidColor).

**Events**

**onChanged**

Triggered when the selected value has changed. This happens when the user selects a new entry.

**Types**

**ComboboxItems**

Create a collection of [ComboboxItems](#459):

```
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "object",
  "title": "ComboboxItems",
  "items": { 
  } 
}
```

**ComboboxItem**

Scheme to create a [Combobox](#457) entry.

```
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "object",
  "title": "ComboboxItem",
  "description": "Defines a combobox item",
  "engineeringColumns": [ 
    "text",
    "value"
  ],
  "propertiesMeta": [ 
    { 
      "name": "text",
      "category": "Common",
      "displayName": "Text",
      "description": "",
      "defaultValue": null,
      "defaultValueInternal": "Entry"
    },
    { 
      "name": "value",
      "category": "Common",
      "displayName": "Value",
      "description": "",
      "defaultValue": null,
      "defaultValueInternal": null
    }
  ],
  "properties": { 
    "text": { 
      "type": "string"
    },
    "value": { 
      "type": "number"
    }
  }
}
```
DateTimeField

The *DateTimeField* can be used to select or display a date and time.

**Use**

Use on any page where date values are to be displayed or edited.

**Features**

If the attribute `ReadOnly` is FALSE, the *DateTimePicker* can be opened via the button to select a new date or time.

**Attributes**

The control inherits from `TextControl` and thus has the same attributes. In addition, there are the following attributes.

**Common**

**DateTime**

`tchmi:framework#/definitions/TcHmi.BuildingAutomation.Controls.BaDateTime`

Current value of time and date.
Events

**onChanged**
Triggered when the value of the date or time has changed.

**EventList**
The *EventList* displays events in list form.

To use the control, the generic functionalities [61] of TcHmiBa must be used.

Use
Can be dragged to any page where events are to be listed.

Features
Displays the events from a specific *BaObject* or *BaView*. It can also display the events of all connected controllers.

Using the buttons in the upper area, events can be filtered according to different event types.

Allows you to acknowledge one or all events. The event currently selected in the list is acknowledged.

The *History* button shows or hides the event history.

Attributes
The control inherits from *BaseControl* [432] and thus has the same attributes. In addition, there are the following attributes.

Common

*IsGlobalEventList*

`tchmi:general#/definitions/Boolean`

Determines whether the events of all connected controllers should be displayed (see *Boolean*).
ActiveEventsCount (read-only)

tchmi:general#/definitions/Number

Number of active events that the user can acknowledge (see Number).

Events

donEventsChanged

This is triggered when the events collection has changed.

InputBox

The InputBox is used to display and edit numerical or textual values.

Use

Use on any page where numerical or textual values are to be displayed or edited.

Features

Numerical input

If the DataType is equal to number, the user input is checked for the following criteria:

- purely numerical input (letters and special characters are not allowed)
- minimum value (if MinValue is set)
- maximum value (if MaxValue is set)

The unit and number of decimal places for a numerical value can also be specified with the attributes Unit and Digits.

Attributes

The control inherits from TextControl [442] and thus has the same attributes. In addition, there are the following attributes.

Feedback concept

The control can use the feedback concept [15].

Common

Value

tchmi:framework#/definitions/TcHmi.BuildingAutomation.StringOrNumber

Current Value. Depending on the selected DataType the value is numerical or textual (see StringOrNumber [464]).

UseValueFeedback

tchmi:general#/definitions/Boolean
If TRUE, then attention is paid to the ValueFeedback attribute (see Boolean).

**ValueFeedback**

tchmi:framework#/definitions/TcHmi.BuildingAutomation.StringOrNumber

Feedback for the value (see StringOrNumber [464]).

**DataType**

tchmi:framework#/definitions/TcHmi.BuildingAutomation.Controls.InputBox.InputDataType

Data type of the InputBox. If auto is selected, the default value or the first input is analyzed and the data type is set accordingly (see InputDataType [463]). If number is selected Value does not contain the unit.

**Number**

**MinValue**

tchmi:general#/definitions/Number

Lowest permissible input value (if DataType is equal to number) (see Number).

**MaxValue**

tchmi:general#/definitions/Number

Largest allowed input value (if DataType is equal to number) (see Number).

**Unit**

tchmi:framework#/definitions/TcHmi.BuildingAutomation.StringOrNumber

Sets the unit after Value (if DataType is equal to number) (see StringOrNumber [464]). Possible values:

- textual (e.g. “°C”)
- numerical (enumeration value of E_BA_Unit)

**Digits**

tchmi:general#/definitions/Number

Number of decimal places (if DataType is equal to number) (see Number).

**Events**

**onUserInteractionFinished**

Triggered when the user exits the input. This means:

- Enter key is pressed
- InputBox loses focus

**Enumerations**

**InputDataType**

Scheme for the DataType attribute of the checkbox.

```
"TcHmi.BuildingAutomation.Controls.InputBox.InputDataType": {
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "string",
  "enum": [0, 1,
```
Types

StringOrNumber

Scheme for an attribute, which can be of data type string or number.

"TcHmi.BuildingAutomation.StringOrNumber": {
  "$schema": "http://json-schema.org/draft-04/schema",
  "oneOf": [
    {
      "title": "String",
      "type": "string"
    },
    {
      "title": "Number",
      "type": "number"
    }
  ]
}

ProjectNavigationTextual

The ProjectNavigationTextual is one of the generic controls and can be used to navigate through the entire project structure of a device. It shows the type, description, value and events of the objects.

More detailed information about the generic capabilities of TcHmiBa and how they can be used can be found here (see Generic HMI [61]).
Use

To be able to navigate through the complete project structure of a runtime, it must be linked with the attribute BaObject [433]. Furthermore, the control allows you to navigate through the children of any view. If only a single object (that is, no view) is linked, then only one entry is displayed with this object.

Features

Generic navigation

The navigation is configured generically based on the structure of the linked object / view. This makes it possible to reach all objects with just one binding. For each object, the type (e.g. analog input, structured view), the description, a value (if available) and the events are displayed.

Header menu

The button in the header opens a menu in which:

- the label to be displayed for the entries can be selected.
- the trend configurator [16] can be opened and the generated trend configurations can be displayed.

The search field allows you to filter the list by a specific term.
Parameter window

The object type is identified by the icon of the button at the start of an entry. Pressing this button opens the parameter window of this object.

The content of this window differs, depending on the selected object. First, all parameters of the respective object are displayed. In the image above, a view is selected that has fewer parameters than an analog input, for example. Which parameters are displayed and which are writable or read-only depends on the rights of the logged-in user.

The changed parameters are written to the PLC when the dialog is closed via the Confirm button. If you click on the Close button, the settings you have made are lost.

Depending on the selected object, the header shows one or two additional buttons.

The button with the event bell is available for every event-enabled object and replaces the contents of the window with the event list of the object when clicked.
The second button is intended for various trend [476] functionalities and is only available for objects that support them or for views that contain objects with trend functions.

**Attributes**

The control inherits from BaseControl [432] and thus has the same attributes. In addition, there are the following attributes.

**Common**

**BAUsedTitle**


Determines which parameter is used for the description in an entry (see BaUsedTitle [467]). The setting can be customized in the client.

**Show header**

tchmi:general#/definitions/Boolean

Determines whether the header is displayed or not (see Boolean).

**Enumerations**

**BaUsedTitle**

Enumeration over the descriptive parameters of a BaObject.

  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "string",
  "enum": [
    "InstDescription",
    "Description",
    "SymbolPath",
    "ObjectName",
    ""
Scale

The Schedule control can be used to display and operate schedules and calendar entries. The current schedule is created on the basis of the weekly schedule and exceptions.
Use

Use on any page where a schedule is to be managed.

If a Schedule object is passed to the `BaObject [433]` attribute, the `generic functions [61]` can be used.

Features

Resulting schedule

The first tab, Current schedule, displays the combination of the weekly schedule and the exceptions. The following hierarchy applies:

1. Local exceptions
2. Global exceptions
3. Weekly schedule

Editing the weekly schedule

On the Weekly schedule tab it is possible to edit the weekly schedule without taking into account exceptions that have already been defined.

In this view there is a schedule with different entries for each day. An entry can be edited or deleted via its menu. The start and end time or position can also be changed with the mouse or finger.

Each daily schedule also has a menu that can be used to add entries and reset changes.
Managing exceptions

The exceptions are managed on the Calendar tab. For more information on how to use it, see the Calendar [449] control.

Menu

Using the menu in the upper right-hand area of the schedule, you can either transfer all changes made to the PLC or discard them.

Attributes

The control inherits from BaseControl [432] and thus has the same attributes. In addition, there are the following attributes.

Common

Orientation


Sets the orientation of the weekly schedule (see Orientation [436]).

SnapPeriod

tchmi:general#/definitions/Number

Determines how precisely schedule entries can be set (see Number). If SnapPeriod is set to 15, for example, entries can be set to the nearest quarter of an hour.

Slider

The slider can be used to display and edit numerical values.

Use

Can be dragged to any page where numerical values are to be edited.

Features

The value can be set by drag 'n' drop or by clicking on the slider. You can also click on the display that shows the current value and then enter the desired value.
You can set whether the min. and max. value or the current value is to be displayed.

Different areas can be colored for the slider. Here you have the possibility to set color gradients or exact color areas.

If the feedback concept [471] is used, the value of the feedback is displayed with a slight shadow. Thus, for example, both values can be visualized at the same time for an object that has a target value and an actual value.

**Attributes**

The control inherits from TextControl [442] and thus has the same attributes. In addition, there are the following attributes.

**Feedback concept**

The control can use the feedback concept [15].

**Common**

**Value**

tchmi:general#/definitions/Number

The current value of the slider (see Number).

**UseValueFeedback**

tchmi:general#/definitions/Boolean

Determines whether or not the attribute ValueFeedback [471] is observed (see Boolean).

**ValueFeedback**

tchmi:general#/definitions/Number

The feedback for the value of the slider (see Number).

**Unit**

tchmi:general#/definitions/String

Defines the unit that is displayed after Value [471] (see String).

**MinValue**

tchmi:general#/definitions/Number

The minimum value of the slider (see Number).
MaxValue
tchmi:general#/definitions/Number

The maximum value of the slider (see Number).

Step
tchmi:general#/definitions/Number

Defines the accuracy with which the value can be set with the slider (e.g. 0.01) (see Number).

Appearance

ShowValue
tchmi:general#/definitions/Boolean

Determines whether the current value is displayed (see Boolean).

ShowScale
tchmi:general#/definitions/Boolean

Determines whether the MinValue [471] and MaxValue [472] are displayed (see Boolean).

Orientation
tchmi:framework#/definitions/TcHmi.BuildingAutomation.Orientation

Sets the orientation of the slider (horizontal or vertical) (see Orientation [436]).

SwitchMinMax
tchmi:general#/definitions/Boolean

If active, the positions of MinValue and MaxValue are swapped (see Boolean).

Ranges

Specifies different color areas or color gradients to be displayed in the slider (see SliderRanges [475]).
KnobAppearance


Sets the appearance of the slider's knob (see KnobAppearance [474]).

Events

onUserInteractionFinished

The event is triggered when the value change has been completed by the user. This happens with drag 'n' drop, when the user releases the slider again or after clicking on an area of the slider.

OnValueChanged

The event is triggered every time the value of the slider changes. Thus, for example, also when the slider is moved.

Enumerations

KnobAppearance

Scheme for the attribute KnobAppearance [470] of the slider.
  "type": "string",
  "enum": [
    0,
    1
  ],
  "options": [
    { "label": "rectangle", "value": 0 },
    { "label": "round", "value": 1 }
  ]
}

**Types**

**SliderRanges**

Scheme for the **Ranges** attribute of the slider.

  "type": "array",
  "title": "SliderRanges",
  "items": {
    "type": "object",
    "title": "SliderRange",
    "description": "Defines a slider range",
    "engineeringColumns": [
      "start",
      "end"
    ],
    "propertiesMeta": [
      { "name": "color", "category": "Color", "displayName": "Color" },
      { "name": "start", "category": "Common", "displayName": "Start " },
      { "name": "end", "category": "Common", "displayName": "End" },
      { "name": "defaultValue", "category": "Common", "displayName": "DefaultValue" },
      { "name": "defaultValueInternal", "category": "Common", "displayName": "DefaultValueInternal" }
    ],
    "properties": {
      "color": { "$ref": "tchmi:framework#/definitions/SolidColor" },
      "start": { "type": "number" },
      "end": { "type": "number" }
    },
    "required": [ "start", "color" ]
  }
}"
Trend

The Trend control can display several trend curves. It allows you to select different trend curves and change the settings of all axes.

Use

Can be used on any page where a trend is to be displayed. Allows linking to a BaObject of type trend object or view.

For more information, see the documentation on Trending.

Features

Multiple trend curves

If the BaObject is a trend object, then only the associated trend curve is displayed. It is not possible to select from different trend curves.

If the BaObject is a view, it is searched for trend objects, and existing trend curves are displayed accordingly. It is possible to select from different trend curves if more than two trend objects are found.

In the listing, the trend curves to be displayed in the chart can be selected via the checkboxes. The adjacent button opens the parameter window of the respective trend object.

Axis parameterization

The settings of a y-axis can be opened by selecting the respective scale values.
Menu

The menu allows further settings for the trend.

Cursor

If Cursor is enabled, a cursor is displayed under the x-axis. By default, this function is disabled.

Data zoom

The checkbox can be used to show or hide the zoom. By default, the zoom is shown.
Redo
Restores the default settings.

Update
The trend curves can be updated once.

Auto Update
If the checkbox is checked, the trend curves are automatically updated as soon as new trend entries are available.

Displayed objects
Determines the objects to be displayed in the listing.

- RefObject: Displays recorded values.
- Trend: Displays all trend objects that record a value.

Displayed label
Selection of the label to be used in the listing.

Attributes
The control inherits from `BaseControl` and thus has the same attributes.

UiIcon
The `UiIcon` can be used to display events and values. It looks like a normal `button` and can be filled with different icons.
Use

Suitable for creating P&I diagrams to represent various plant components (e.g. pump). The attribute Connections can be used to create suitable connections to connect the Uilcon with a main line, for example.

Features

Value displays

Various displays can be added to the Uilcon via the attribute DisplaysData.

![Value display example](image)

Event displays

The attribute EventsData can be used to display various events around the Uilcon.

![Event display example](image)

If the generic approach [61] of TcHmiBa is used and a BaObject / BaView is linked to the control, active events are displayed automatically. When the Uilcon is actuated, the project navigation [464] of the linked object opens and, in the case of an event, the parameter window [465] with the event view opens accordingly.

![Project Structure PLC1](image)
Attributes

The control inherits from the button \[447\] and thus has the same attributes. In addition, there are the following attributes.

Common

DisplaysData

The attribute makes it possible to create different displays via an editor (see DisplaysData \[482\]).

The following properties can be set for each display:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextColor</td>
<td>Font color of the display.</td>
</tr>
<tr>
<td>Value</td>
<td>Display value in the display. If a binding exists and ReadOnly is disabled, the value is written to this binding when the user ends the input.</td>
</tr>
<tr>
<td>Position</td>
<td>Position of the display. Several displays created at the same position are arranged on top of each other.</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>Determines whether the display is editable or read-only.</td>
</tr>
<tr>
<td>Unit</td>
<td>Unit to be appended to the value (if it is a number).</td>
</tr>
<tr>
<td>Digits</td>
<td>Number of decimal places.</td>
</tr>
<tr>
<td>FontWeight</td>
<td>Font weight of the text.</td>
</tr>
</tbody>
</table>

EventsData

The attribute makes it possible to create various events via an editor (see EventsData \[484\]).
The following properties can be set for each event:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Determines whether the event is active or not.</td>
</tr>
<tr>
<td>Event condition</td>
<td>Sets the type (priority) of the event. The icons are arranged according to their priority in a clockwise direction. Top right is the highest priority.</td>
</tr>
<tr>
<td>Event state</td>
<td>Current state of the event.</td>
</tr>
<tr>
<td>Event count</td>
<td>Determines how many events of this type and state are active.</td>
</tr>
<tr>
<td>Acknowledge symbol</td>
<td>Writes TRUE to the symbol when the event is pressed.</td>
</tr>
<tr>
<td>Is Acknowledgable</td>
<td>Determines whether the event can be pressed.</td>
</tr>
<tr>
<td>Icon</td>
<td>Icon to use if no Event condition has been selected, in order to allow user-specific icons.</td>
</tr>
<tr>
<td>Enable pulse</td>
<td>Evaluation only takes place if Event condition and Event state are not used. When enabled, a red pulse is displayed around the UlIcon.</td>
</tr>
</tbody>
</table>

**Connections**

Connections can be used to represent connections to other lines in an P&I diagram.

Connections can be created here that extend vertically or horizontally away from the UlIcon (see Padding).
The length of the connection must be specified in each case.

The unit pixel is always used. Percent is **not** supported at this point.

**ConnectionExtensions**

tchmi:framework#/definitions/Padding

Here extensions can be created for the connections created above (see **Padding**).

The length of the extension must be specified in each case.

The unit pixel is always used. Percent is **not** supported at this point.

**ConnectionsWidth**

tchmi:framework#/definitions/PositiveNumber

Specification of the width in pixels for the connections (see **PositiveNumber**).

**ConnectionsColor**

tchmi:framework#/definitions/SolidColor

Specification of the color for the connections (see **SolidColor**).

**ConnectionsColorPerSide**

tchmi:framework#/definitions/TcHmi.BuildingAutomation.FourSidedColor

Defines the color for different connectors (see **FourSidedColor [436]**). The attribute **ConnectionsColor** must be set to **NULL** or **NONE**.

**Types**

**DisplaysData**

Scheme for an editor that makes it possible to create different displays for a **UiIcon [478]**.
"TcHmi.BuildingAutomation.Controls.UiIcon.DisplaysData": {  
  "$schema": "http://json-schema.org/draft-04/schema",  
  "type": "array",  
  "items": {  
    "type": "object",  
    "title": "DisplayData",  
    "description": "Defines a display for the object.",  
    "propertiesMeta": [  
      {  
        "name": "value",  
        "category": "Common",  
        "displayName": "Value",  
        "description": "Value that is displayed in the display.",  
        "displayPriority": 10,  
        "defaultValue": null,  
        "defaultValueInternal": null  
      },  
      {  
        "name": "position",  
        "category": "Common",  
        "displayName": "Position",  
        "description": "Position of the display.",  
        "displayPriority": 10,  
        "defaultValue": 1,  
        "defaultValueInternal": 1  
      },  
      {  
        "name": "readOnly",  
        "category": "Common",  
        "displayName": "ReadOnly",  
        "description": "Defines if the display is editable or read only.",  
        "displayPriority": 10,  
        "defaultValue": true,  
        "defaultValueInternal": true  
      },  
      {  
        "name": "unit",  
        "category": "Common",  
        "displayName": "Unit",  
        "description": "The unit will be added to the displayed value if the type of Value is 'number'.",  
        "displayPriority": 10,  
        "defaultValue": null,  
        "defaultValueInternal": null  
      },  
      {  
        "name": "digits",  
        "category": "Common",  
        "displayName": "Digits",  
        "description": "Number of displayed digits.",  
        "displayPriority": 10,  
        "defaultValue": 1,  
        "defaultValueInternal": 1  
      },  
      {  
        "name": "textColor",  
        "category": "Colors",  
        "displayName": "TextColor",  
        "description": "Color of the displayed text.",  
        "displayPriority": 10,  
        "defaultValue": null,  
        "defaultValueInternal": null  
      },  
      {  
        "name": "fontWeight",  
        "category": "Text",  
        "displayName": "FontWeight",  
        "description": "Font weight of the displayed text.",  
        "displayPriority": 10,  
        "defaultValue": "Bold",  
        "defaultValueInternal": "Bold"  
      }  
    ],  
    "properties": {  
      "value": {  
        "type": "string"  
      },  
      "position": {  
        "$ref": "tchmi:framework#/definitions/TcHmi.BuildingAutomation.PositionEx"  
      }  
    }  
  }  
}
EventsData

Scheme for an editor that makes it possible to create different events for a UlIcon [478].

"TcHmi.BuildingAutomation.Controls.UiIcon.EventsData": {
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "array",
  "items": {
    "type": "object",
    "title": "EventData",
    "description": "Defines an event for the object.",
    "properties": {
      "name": "activate",
      "category": "Common",
      "displayName": "Activate",
      "description": "If true the event will be displayed on the UlIcon.",
      "defaultValue": false,
      "defaultValueInternal": false
    },
    "name": "eventCondition",
    "category": "Common",
    "displayName": "Event condition",
    "description": "BA event condition. If this is used the icon of the event will be selected automatically.",
    "defaultValue": null,
    "defaultValueInternal": null
  },
  "name": "eventIconState",
  "category": "Common",
  "displayName": "Event state",
  "description": "BA event icon state. If this is used the icon state of the event will be selected automatically.",
  "defaultValue": null,
  "defaultValueInternal": null
},
  "name": "eventCount",
  "category": "Common",
  "displayName": "Event count",
  "description": "If larger than 0 a small number will be displayed on the event. This indicates the number of active events with the same icon.",
  "defaultValue": null,
  "defaultValueInternal": null
},
  "name": "isAcknowledgeable",
  "category": "Other",
  "displayName": "Acknowledge symbol",
  "description": "Boolean symbol that should be set to true when event icon was pressed.",
  "displayPriority": 200,
  "defaultValue": null,
  "defaultValueInternal": null
}
7.2.1.3.3 Templates

RoomAutomation

RoomControl

The RoomControl can combine the various controls of the room automation. Available components are:

- AirConditioning [495]
- Light [495]
- Sunblind [499]
- Window [504]
Use

The *RoomControl* can be used to automate a room or area with only one control. It is possible, for example, to combine only one area with lamps so that the overview in the visualization is maintained.

Features

All settings that are possible with the individual controls can be made.

Room status

The general room information is displayed in the header of the control:

- Light on or off
- Mode or message with the highest priority
- Occupancy
- Overrun time

Side menu for control

Clicking on the control opens the side menu, where the controls for the individual areas can be found.
Distributed control

Controls are located above each area, e.g. to control all lamps simultaneously. This makes it possible to set all lamps to the same brightness value.

Attributes

The control inherits from BaseControl [432] and thus has the same attributes. In addition, there are the following attributes.

Feedback concept

The control can use the feedback concept [15].

Layout

HideRoomStatus
tchmi:general#/definitions/Boolean

If TRUE, then the room information (header) is not visible (see Boolean).

ShowRoomName
tchmi:general#/definitions/Boolean
If TRUE, the room name [488] is displayed instead of the room information (see Boolean).

**Common**

**ControlUnits**


Specifies which components (AirConditioning [495], Light [495], Sunblind [499], Window [504]) should be added (see ControlList [489]).

**Name**

tchmi:general#/definitions/String

The room name (see String).

**Presence**

tchmi:general#/definitions/Boolean

If TRUE, then presence was detected in the room (see Boolean).

- Presence active

- Presence inactive

**DelayActive**

tchmi:general#/definitions/Boolean

If TRUE, then the overrun time for automatic control is active (see Boolean).

- Overrun time active

- Overrun time inactive

**AirConditionings**

**ShowAirConditionings**

tchmi:general#/definitions/Boolean

If TRUE, the AirConditionings are displayed in the control (see Boolean). The components are always visible in the side menu.

Further information on the attributes can be found in the documentation for the AirConditioning [495] control.

**Lights**

**ShowLights**

tchmi:general#/definitions/Boolean
If TRUE, the lights are displayed in the control. The components are always visible in the side menu (see Boolean).

Further information about the attributes can be found in the documentation for the Light control.

Sunblinds

ShowSunblinds
tchmi:general#/definitions/Boolean

If TRUE, the Sunblinds are displayed in the control (see Boolean). The components are always visible in the side menu.

Further information about the attributes can be found in the documentation for the Sunblind control.

Windows

ShowWindows
tchmi:general#/definitions/Boolean

If TRUE, then the Windows are displayed in the control (see Boolean). The components are always visible in the side menu.

Further information about the attributes can be found in the documentation for the Window control.

Types

ControlList

Scheme for the attribute ControlUnits of the RoomControl.

```
"TcHmi.BuildingAutomation.Controls.RoomControl.ControlList": {  
  "title": "ControlList",  
  "type": "array",  
  "items": {  
    "oneOf": [  
      {  
        "title": "Light",  
        "type": "object",  
        "additionalProperties": false,  
        "engineeringColumns": [  
          "Name"  
        ],  
        "propertiesMeta": [  
          {  
            "name": "baObject",  
            "displayName": "BaObject",  
            "category": "BA",  
            "displayPriority": 1,  
            "description": "BaView of the light.",  
            "defaultValue": null,  
            "defaultValueInternal": null  
          },  
          {  
            "name": "name",  
            "displayName": "Name",  
            "category": "Common",  
          }  
        ]  
      }  
    ]  
  }  
}  
```
"displayPriority": 10,
"description": "Name of the light."
},

"name": "brightness",
"displayName": "Brightness",
"category": "Common",
"displayPriority": 10,
"description": "Brightness of the light. When used as data binding the brightness will be written when changed by the user."
},

"defaultValue": 0,
"defaultValueInternal": 0
},

"name": "brightnessFeedback",
"displayName": "BrightnessFeedback",
"category": "Common",
"displayPriority": 10,
"description": "Feedback for the brightness of the light."
},

"defaultValue": null,
"defaultValueInternal": null
},

"name": "mode",
"displayName": "Mode",
"category": "Common",
"displayPriority": 10,
"description": "Mode of the light."
},

"defaultValue": null,
"defaultValueInternal": null
},

"name": "resetHandMode",
"displayName": "ResetHandMode",
"category": "Common",
"displayPriority": 10,
"description": "Symbol which is set to true if the hand mode was reseted."
},

"defaultValue": null,
"defaultValueInternal": null
}

"properties": {
  "baObject": [ "string", "null" ],
  "name": [ "string", "null" ],
  "brightness": [ "number", "boolean", "string", "null" ],
  "brightnessFeedback": [ "number", "boolean", "string", "null" ],
  "mode": [ "oneOf": [ "$ref": "tchmi:framework#/definitions/TcHmi.BuildingAutomation.Controls.Light.Mode" ],
            "type": [ "string", "null" ] ]
},

"resetHandMode": [ "string", "null" ],

"title": "Sunblind",
"type": "object",
"additionalProperties": false,
"engineeringColumns": [ "Name" ],

"propertiesMeta": [ "baObject": "string", "displayName": "BaObject", "category": "BA" ]
"displayPriority": 1,
"description": "BaView of the sunblind."
},
{ "name": "name",
"displayName": "Name",
"category": "Common",
"displayPriority": 10,
"description": "Name of the sunblind.",
"defaultValue": null,
"defaultValueInternal": null
},
{ "name": "facadeName",
"displayName": "FacadeName",
"category": "Common",
"displayPriority": 10,
"description": "Defines the facade where the sunblind is mounted. (Equal to the name in BuildingInformation control)",
"defaultValue": null,
"defaultValueInternal": null
},
{ "name": "position",
"displayName": "Position",
"category": "Common",
"displayPriority": 10,
"description": "Position of the sunblind. When used as data binding the position will be written when changed by the user.",
"defaultValue": null,
"defaultValueInternal": 0
},
{ "name": "positionFeedback",
"displayName": "PositionFeedback",
"category": "Common",
"displayPriority": 10,
"description": "Feedback for the position of the sunblind.",
"defaultValue": null,
"defaultValueInternal": null
},
{ "name": "angle",
"displayName": "Angle",
"category": "Common",
"displayPriority": 10,
"description": "Angle of the sunblind. When used as data binding the angle will be written when changed by the user.",
"defaultValue": null,
"defaultValueInternal": 0
},
{ "name": "angleFeedback",
"displayName": "AngleFeedback",
"category": "Common",
"displayPriority": 10,
"description": "Feedback for the angle of the sunblind.",
"defaultValue": null,
"defaultValueInternal": null
},
{ "name": "mode",
"displayName": "Mode",
"category": "Common",
"displayPriority": 10,
"description": "Mode of the sunblind.",
"defaultValue": null,
"defaultValueInternal": null
},
{ "name": "resetHandMode",
"displayName": "ResetHandMode",
"category": "Common",
"displayPriority": 10,
"description": "Symbol which is set to true if the hand mode was reseted.",
"defaultValue": null,
"defaultValueInternal": null
}
Programming

{
  "properties": {
    "baObject": {
      "type": ["string", "null"]
    },
    "name": {
      "type": ["string", "null"]
    },
    "facadeName": {
      "type": ["string", "null"]
    },
    "position": {
      "type": ["number", "null", "string"]
    },
    "positionFeedback": {
      "type": ["number", "null", "string"]
    },
    "angle": {
      "type": ["number", "null", "string"]
    },
    "angleFeedback": {
      "type": ["number", "null", "string"]
    },
    "mode": {
      "oneOf": [null]
    },
    "resetHandMode": {
      "type": ["string", "null"]
    }
  },
  "title": "Air conditioning",
  "type": "object",
  "additionalProperties": false,
  "engineeringColumns": [
    "Name"
  ],
  "propertiesMeta": {
    "name": "baObject",
    "displayName": "BaObject",
    "category": "BA",
    "displayPriority": 1,
    "description": "BaView of the air conditioning.",
    "defaultValue": null,
    "defaultValueInternal": null
  },
  "name": "name",
  "displayName": "Name",
  "category": "Common",
  "displayPriority": 10,
  "description": "Name of the light.",
  "defaultValue": null,
  "defaultValueInternal": null
},
"name": "currentTemp",
"displayName": "CurrentTemperature",
"category": "Common",
"displayPriority": 10,
"description": "The current temperature.",
"defaultValue": 0,
"defaultValueInternal": 0
},
"name": "tempAdjust",
"displayName": "TemperatureAdjustment",
"category": "Common",
"displayPriority": 10,
"description": "The adjustment for the current temperature setpoint.",
"defaultValue": null,
"defaultValueInternal": null
}
{ "name": "tempAdjustFeedback", "displayName": "TempAdjustFeedback", "category": "Common", "displayPriority": 10, "description": "Feedback for the temp adjust.", "defaultValue": null, "defaultValueInternal": null },
{ "name": "heatingSetpoint", "displayName": "HeatingSetpoint", "category": "Common", "displayPriority": 10, "description": "Setpoint of the heating controller. If no cooling setpoint is used it is possible to use the CoolingActive flag to switch between heating and cooling mode.", "defaultValue": null, "defaultValueInternal": null },
{ "name": "coolingSetpoint", "displayName": "CoolingSetpoint", "category": "Common", "displayPriority": 10, "description": "Setpoint of the cooling controller.", "defaultValue": null, "defaultValueInternal": null },
{ "name": "heatingActive", "displayName": "HeatingActive", "category": "Common", "displayPriority": 10, "description": "Flag that defines if heating mode is active or not.", "defaultValue": null, "defaultValueInternal": null },
{ "name": "coolingActive", "displayName": "CoolingActive", "category": "Common", "displayPriority": 10, "description": "Flag that defines if cooling mode is active or not.", "defaultValue": null, "defaultValueInternal": null }
],
"properties": {
  "beObject": { "type": [ "string", "null"] },
  "name": { "type": [ "string", "null"] },
  "currentTemp": { "type": [ "number", "null", "string"] },
  "tempAdjust": { "type": [ "number", "null", "string"] },
  "tempAdjustFeedback": { "type": [ "number", "string", "null"] },
  "heatingSetpoint": { "type": [ "number", "null", "string"] },
  "coolingSetpoint": { "type": [ "number", "null", "string"] },
  "heatingActive": { "type": [ "boolean", "number", "null", "string"] },
  "coolingActive": { "type": [ "boolean", "number", "null", "string"] }
} },
"engineeringColumns": [ "Name" ],
"propertiesMeta": [
{"name": "baObject", "displayName": "BaObject", "category": "BA", "displayPriority": 1, "description": "BaView of the light.", "defaultValue": null, "defaultValueInternal": null },
{"name": "name", "displayName": "Name", "category": "Common", "displayPriority": 10, "description": "Name of the window.", "defaultValue": null, "defaultValueInternal": null },
{"name": "facadeName", "displayName": "FacadeName", "category": "Common", "displayPriority": 10, "description": "Defines the facade where the sunblind is mounted. (Equal to the name in BuildingInformation control)", "defaultValue": null, "defaultValueInternal": null },
{"name": "windowPosition", "displayName": "Position", "category": "Common", "displayPriority": 10, "description": "Position of the light. When used as data binding the position will be written when changed by the user.", "defaultValue": null, "defaultValueInternal": 0 },
{"name": "windowPositionFeedback", "displayName": "PositionFeedback", "category": "Common", "displayPriority": 10, "description": "Feedback for the position of the window.", "defaultValue": null, "defaultValueInternal": null },
{"name": "mode", "displayName": "Mode", "category": "Common", "displayPriority": 10, "description": "Mode of the window.", "defaultValue": null, "defaultValueInternal": null },
{"name": "resetHandMode", "displayName": "ResetHandMode", "category": "Common", "displayPriority": 10, "description": "Symbol which is set to true if the hand mode was reseted.", "defaultValue": null, "defaultValueInternal": null },
{"name": "displayMode", "displayName": "DisplayMode", "category": "Common", "displayPriority": 10, "description": "Defines the display mode of the window.", "defaultValue": 0, "defaultValueInternal": 0 },
{"name": "iconRotation",}]}
"displayName": "IconRotation",
"category": "Common",
"displayPriority": 10,
"description": "The rotation of the icon in degree (0°-360°).",
"defaultValue": null,
"defaultValueInternal": null
],
"properties": {
  "baObject": {
    "type": ["string", "null"]
  },
  "name": {
    "type": ["string", "null"]
  },
  "facadeName": {
    "type": ["string", "null"]
  },
  "windowPosition": {
    "type": ["number", "boolean", "string", "null"]
  },
  "windowPositionFeedback": {
    "type": ["number", "boolean", "string", "null"]
  },
  "mode": {
    "oneOf": [
      { "type": ["string", "null"] }
    ]
  },
  "resetHandMode": {
    "type": ["string", "null"]
  },
  "displayMode": {
    "oneOf": [
      { "type": ["string", "null"] }
    ]
  },
  "iconRotation": {
    "type": ["number", "null"]
  }
}]

AirConditioning

Light

The Light control is used to display and control the brightness of a light source.
Use

Use on any page where controls are needed to control light sources.

Features

Brightness specification

The brightness can be adjusted using an analog (dimmable) or binary (switchable) value.

Buttons with predefined brightness values are available for dimmable lamps.

Switchable lamps have only one switch.

Operation modes

Display of different modes via the attribute *Mode*.

Sizes

Depending on the size of the control, various information and operating elements are visible.

Clicking on the Light control changes the visibility of the menu for adjusting the brightness.
Attributes

The control inherits from `BaseControl`[^432] and thus has the same attributes. In addition, there are the following attributes.

Feedback concept

The control can use the `feedback concept`[^15].

Common

DisplayMode


Sets the display mode of the lamp (see `Mode`[^499]).

<table>
<thead>
<tr>
<th>Name</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>lightBulb</td>
<td><img src="image" alt="lightBulb" /></td>
</tr>
<tr>
<td>lightBulbFilled</td>
<td><img src="image" alt="lightBulbFilled" /></td>
</tr>
<tr>
<td>files</td>
<td><img src="image" alt="files" /></td>
</tr>
</tbody>
</table>

Mode


Sets the display mode for the lamp (see `Mode`[^499]).
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoActive</td>
<td>Automatic for the lighting is active.</td>
<td></td>
</tr>
<tr>
<td>autoInactive</td>
<td>Automatic for the lighting is switched off or not available.</td>
<td></td>
</tr>
<tr>
<td>hand</td>
<td>Automatic was overwritten by a manual intervention. Display of a button for resetting the &quot;hand&quot; mode. When pressed, the onHandModeReset event is triggered.</td>
<td></td>
</tr>
</tbody>
</table>

**ShowValue**

tchmi:general#/definitions/Boolean

If TRUE, the brightness value is displayed (see Boolean).

**Brightness**

tchmi:framework#/definitions/TC.Hmi.BuildingAutomation.NumberOrBoolean

Current brightness value (see NumberOrBoolean [437]).

**UserBrightnessFeedback**

tchmi:general#/definitions/Boolean

If TRUE, then the BrightnessFeedback attribute is observed (see Boolean).

**BrightnessFeedback**

tchmi:framework#/definitions/TC.Hmi.BuildingAutomation.NumberOrBoolean

Feedback for the brightness value (see NumberOrBoolean [437]).

**MinBrightness**

tchmi:general#/definitions/Number

Lowest permissible brightness value (see Number).

**MaxBrightness**

tchmi:general#/definitions/Number

Maximum permissible brightness value (see Number).

**Events**

**onBrightnessChanged**

Triggered when the user changes the brightness.
onHandModeReset
Triggered when the button for resetting from "hand" to automatic mode is pressed.

Enumerations

Mode
Scheme for the attribute **Mode** of the Light control.

```
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "integer",
  "enum": [1, 2, 3],
  "options": [
    { "label": "autoActive", "value": 1 }
  ]
}
```

DisplayMode
Scheme for the attribute **DisplayMode** of the Light control.

```
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "string",
  "enum": [0, 1, 2],
  "options": [
    { "label": "lightBulb", "value": 0 }
  ]
}
```

Sunblind
The **Sunblind** control can display and control the position and angle of sunblinds.
Use
Use on any page where controls are needed to control sunblinds.

Features

Operation modes
Display of different modes via the attribute Mode.

Angle setting
The angle setting is optional and can only be used if the sunblind supports it. This function is activated or deactivated via the attribute UseAngle.

Size
Depending on the size of the control, various information and operating elements are visible.

Clicking on the Sunblind control changes the visibility of the menu for setting the position or angle.
Attributes

The control inherits from BaseControl [432] and thus has the same attributes. In addition, there are the following attributes.

Feedback concept

The control can use the feedback concept [15].

Common

FacadeName

tchmi:general#/definitions/String

Name of the facade to which the sunblind is assigned (see String). It can be set in BuildingInformation control [445].

Controls


Specifies the type of controls to be used for position and angle settings (see Controls [504]).

- sliderHorizontal

![Image of sliderHorizontal]

- buttons

![Image of buttons]

ShowValue

tchmi:general#/definitions/Boolean

If TRUE, the values for angle and position are displayed (see Boolean).
### Mode


Sets the displayed mode for the sunblind (see Mode [504]).

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoActive</td>
<td>Automatic for the sunblind is active.</td>
<td></td>
</tr>
<tr>
<td>autoInactive</td>
<td>Automatic for the sunblind is switched off or not available.</td>
<td></td>
</tr>
<tr>
<td>hand</td>
<td>Automatic was overwritten by a manual intervention. Display of a button for resetting the “hand” mode. When pressed, the onHandModeReset event is triggered.</td>
<td></td>
</tr>
<tr>
<td>maintenance</td>
<td>Sunblind is in the maintenance position.</td>
<td></td>
</tr>
<tr>
<td>safetyPosition</td>
<td>Sunblind is in the safety position.</td>
<td></td>
</tr>
</tbody>
</table>

### Position

tchmi:general#/definitions/Number

Current position (see Number).

### UsePositionFeedback

tchmi:general#/definitions/Boolean

If TRUE, then the attribute PositionFeedback is observed (see Boolean).

### PositionFeedback

tchmi:general#/definitions/Number

Feedback for the position (see Number).

### MinPosition

tchmi:general#/definitions/Number

Position value for the lower end position (see Number).
### MaxPosition
`tchmi:general#/definitions/Number`

Position value for the upper end position (see `Number`).

### Angle

#### UseAngle
`tchmi:general#/definitions/Boolean`

If TRUE, then the controls for controlling the angle are displayed (see `Boolean`).

#### Angle
`tchmi:general#/definitions/Number`

Current angle (see `Number`).

#### UseAngleFeedback
`tchmi:general#/definitions/Boolean`

If TRUE, then the attribute `AngleFeedback` is observed (see `Boolean`).

#### AngleFeedback
`tchmi:general#/definitions/Number`

Feedback for the angle (see `Number`).

#### MinAngle
`tchmi:general#/definitions/Number`

Lowest permissible angle (see `Number`).

#### MaxAngle
`tchmi:general#/definitions/Number`

Maximum permissible angle (see `Number`).

#### AngleStep
`tchmi:general#/definitions/Number`

Defines the step size with which the angle is adjusted via the angle buttons (see `Number`).

### Events

#### onPositionChanged

Triggered when the user changes the position of the sunblind.

#### onAngleChanged

Triggered when the user changes the angle of the sunblind.

#### onHandModeReset

Triggered when the button for resetting from "hand" to automatic mode is pressed.
Enumerations

Mode
Scheme for the attribute Mode of the Sunblind control.

  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "integer",
  "enum": [1, 2, 3, 4, 5],
  "options": [
    {
      "label": "autoActive",
      "value": 1
    },
    {
      "label": "autoInactive",
      "value": 2
    },
    {
      "label": "hand",
      "value": 3
    },
    {
      "label": "maintenance",
      "value": 4
    },
    {
      "label": "safetyPosition",
      "value": 5
    }
  ]
}

Controls
Scheme for the Controls attribute of the Sunblind control.

  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "string",
  "enum": [
    "buttons",
    "sliderHorizontal"
  ]
}

Window
The Window control can display and control the position of windows or roof domes with drives.
Use

Use on any page where controls are needed to control windows.

Features

Position specification

The position of the window can be specified via an analog or binary value if the drive supports it.

For analog values, the setting is possible via a slider and buttons with predefined positions.

For binary values only the settings "open" or "closed" are possible.

Sizes

Depending on the size of the control, various information and operating elements are visible.

Clicking on the Window control changes the visibility of the menu for setting the position.
Attributes

The control inherits from BaseControl [432] and thus has the same attributes. In addition, there are the following attributes.

Feedback concept

The control can use the feedback concept [15].

Common

FacadeName
tchmi:general#/definitions/String

Name of the facade to which the window is assigned (see String). It can be set in Building Information Control [445].

ShowValue
tchmi:general#/definitions/Boolean

If TRUE, the value for the position is displayed (see Boolean).

DisplayMode

Defines the display mode of the window (see DisplayMode [508]).

<table>
<thead>
<tr>
<th>Name</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>roofDome</td>
<td><img src="image" alt="roofDome" /></td>
</tr>
<tr>
<td>window</td>
<td><img src="image" alt="window" /></td>
</tr>
</tbody>
</table>

IconRotation
tchmi:general#/definitions/Number

Rotates the icon. Angle specification in degrees (see Number).

Position
tchmi:framework#/definitions/TcHmi.BuildingAutomation.NumberOrBoolean

Current position (see NumberOrBoolean [437]).

UsePositionFeedback
tchmi:general#/definitions/Boolean
If TRUE, then the attribute *PositionFeedback* is observed (see *Boolean*).

**PositionFeedback**  
tchmi:framework#/definitions/TcHmi.BuildingAutomation.NumberOrBoolean

Feedback for the position (see *NumberOrBoolean* [437]).

**MinPosition**  
tchmi:general#/definitions/Number

Lowest permissible position value (see *Number*).

**MaxPosition**  
tchmi:general#/definitions/Number

Largest permissible position value (see *Number*).

**Events**

**onPositionChanged**
Triggered when the user changes the position of the window or a roof dome.

**onHandModeReset**
Triggered when the button for resetting from "hand" to automatic mode is pressed.

**Enumerations**

**Mode**

Scheme for the attribute *Mode* [504] of the Window control.

```
  "$schema": "http://json-schema.org/draft-04/schema",  
  "type": "integer",  
  "enum": [  
    1,  
    2,  
    3,  
    4,  
    5  
  ],  
  "options": [  
    {  
      "label": "autoActive",  
      "value": 1  
    },  
    {  
      "label": "autoInactive",  
      "value": 2  
    },  
    {  
      "label": "hand",  
      "value": 3  
    },  
    {  
      "label": "maintenance",  
      "value": 4  
    },  
    {  
      "label": "safetyPosition",  
      "value": 5  
    }  
  ]
```
DisplayMode

Scheme for the attribute DisplayMode [504] of the window control.

```
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "string",
  "enum": [0, 1],
  "options": [
    { "label": "standard", "value": 0 },
    { "label": "roofDome", "value": 1 }
  ]
}
```

7.2.1.4 Server extension

The TcHmiBaServerExtension serves as an interface between a TF8040 PLC (TF8040 Getting Started [49]) and a TcHmi client. The extension makes it possible to offer generic functions that significantly simplify and accelerate the engineering of the HMI.

Links between the PLC and the HMI server are still possible via the ADS extension, even without the extension. In this case the benefits referred to above no longer apply.

System requirements

- TF8040 (current version)
- TE2000 for the development system
- TF2000 for the target system
- Windows 10 or higher for development and target system

Use

In order to use the generic functionalities in a project, the extension must be installed (see Generic HMI [61]).

Configuration

The extension can be configured in TcHmi engineering or via the configuration page of the HMI server.

Configuration from TcHmi engineering:
Interval

This time determines how often variables are read (updated) in the interface.

Devices

The extension must be notified of the devices to which the HMI is to connect. The devices can be added or removed via the configuration mask.

**TcHmiBaServerExtension**

<table>
<thead>
<tr>
<th>Interval for reading PLC variables.</th>
<th>Can be set for specific remote configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td></td>
</tr>
</tbody>
</table>

**Runtimes**

**PLC1**

<table>
<thead>
<tr>
<th>Enabled</th>
<th>AmsNetId</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>127.0.0.1.1.1</td>
<td>851</td>
</tr>
</tbody>
</table>

+ Add

Accept:

Settings required for each device:

- Determination whether enabled/disabled
- AmsNetId
- PLC port

The configuration is enabled by clicking Accept.

Checking the configuration

All devices should now be listed under All symbols in the TwinCAT HMI Configuration window.
A mapping for the device was also created.

**Required mappings**

The mapping automatically created for each device with the same name as the device itself is mandatory and must not be renamed or deleted.

Activating the configuration automatically adds a device of the same name and with the same settings in the ADS extension. This device in the ADS extension is also mandatory and must not be renamed or deleted.
Symbols

Device symbols

A dynamic symbol with the same name as the device is created for each device. The structure of the first level of a device symbol is always the same.

![Device symbol structure]

The first level contains the `ProjectStructure` of the device, which then contains the `Children`. These `Children` are thus the first actual elements in the project structure.

TerminalClientEvents

When this symbol is read, the extension returns all events captured on all devices, including the history. This makes the symbol suitable for a cross-device event list.

TerminalClientActiveEventsCount

The symbol indicates the number of active events on the device.

> Active events are events that require interaction from the user (acknowledge).

7.2.1.5 Project templates

Project templates are intended to make it easier to get started with a `TcHmiBa` project by pre-installing required dependencies and providing additional elements (e.g. navigation).

Installation

The installation of the project templates into the available development environments is **not** performed by the TF8040 but by the batch file `InstallProjectTemplates.bat`.

> After installing the TwinCAT 3 HMI and TF8040, the batch file is located in the following directory:

```
C:\TwinCAT\Functions\TF8040 TC3 Building Automation\HMI\ProjectTemplates
```

A double click on the `InstallProjectTemplates.bat` executes it and a console window opens.
Notes
The project templates currently have some limitations. The following section describes these in more detail and how they can be remedied.

User management
In order to use the integrated user management of TF8040, it is necessary that certain user groups and matching users exist in the TcHmi project. The project template does not include these users.

The program CreateDefaultUserManagement.exe can be used to create the required settings for a project.

After installing the TwinCAT 3 HMI and TF8040, the program is located in the following directory:

C:\TwinCAT\Functions\TF8040 TC3 Building Automation\HMI\Tools\CreateDefaultUserManagement

Double-clicking CreateDefaultUserManagement.exe will execute it.

Steps:
1. First, the HMI project must be selected.
2. Close the project (if not already closed).
3. Check the output

```
Please select TcHmi project file:
Selected project file: 'C:\temp\TcHmiBaProject1\TcHmiBaProject1.hmiproj'
Start editing TcHmiUserManagement.Config.default.json
'Advanced User' added to Users.
'Default User' added to Users.
'Expert User' added to Users.
'Internal User' added to Users.
Finished editing TcHmiUserManagement.Config.default.json

Writing to TcHmiUserManagement.Config.default.json ...
Finished writing to TcHmiUserManagement.Config.default.json.

Start editing TcHmiSrv.Config.default.json
'Advanced User' added to UserGroupUsers.
'Default User' added to UserGroupUsers.
'Expert User' added to UserGroupUsers.
'Internal User' added to UserGroupUsers.
'Advanced' added to UserGroups.
'Default' added to UserGroups.
'Expert' added to UserGroups.
'Internal' added to UserGroups.
Finished editing TcHmiSrv.Config.default.json

Writing to TcHmiSrv.Config.default.json ...
Finished writing to TcHmiSrv.Config.default.json.
```
4. Open the project and create passwords for the created users.

No passwords

No passwords have been set for created users! Go to the TwinCAT HMI Configuration window and set a password for each user!

TcHmiBaServerExtension

The server extension will start inactive because the configuration files are not loaded with the project template.

Therefore a manual start is necessary.
7.2.1.5.1 TcHmiBaProject

The project template is used to create a TcHmi project quickly and easily.

The instructions for project templates must be observed.

Contents

Overview of the contents of the project template.
The project template contains the following NuGet packages:

- Beckhoff.TwinCAT.HMI.BA.Controls
- Beckhoff.TwinCAT.HMI.BA.Framework
- Beckhoff.TwinCAT.HMI.BA.Icons
- Beckhoff.TwinCAT.HMI.BA.ServerExtension
- Beckhoff.TwinCAT.HMI.Controls
- Beckhoff.TwinCAT.HMI.Framework
- Beckhoff.TwinCAT.HMI.Functions
- Beckhoff.TwinCAT.HMI.ResponsiveNavigation

To make it easier to get started, the project template already contains some content pages, such as a navigation in the header.

The following pages are required for the header to function correctly:

- EventList.content
• ServerLog.content
• StartPage.content

The navigation in the header refers to the following pages:
• PlantView.content
• ProjectNavigation.content

These pages are customizable, and further pages can be added.

If pages are added, renamed or removed, the attribute MenuData [517] of the header must be updated.

Header

The header is a UserControl and serves as an entry point for users. It offers an easy way to configure a navigation for the HMI and several other features.

Features

The features at a glance (from left to right).
• Logo (1)
• Responsive navigation (2)
• User settings and further information (3)
• Event list (4)
• Building information (5)
• Outdoor temperature (6)
• Date and time (7)

User settings and further information

In this menu, users can make various settings that affect them. In addition, diagnostic data and the server log can be displayed.

Event list

The event list can be called up via the button with the bell symbol. The button also displays the number of active events, if the number was linked via the EventCount attribute.
Building information

The button with the info symbol is the BuildingInformation control.

It can be used to open a window with information about the building and the facades.

Attributes

The control inherits from TcHmiControl and thus has the same attributes. In addition, there are the following attributes.

Logo

tchmi:framework#/definitions/ContentPath

Path specification to the image with the logo that is displayed at the start of the header.

MenuData


Defines the structure and hierarchy of the navigation. Entries in the header can be linked directly to content pages, or submenus can be created that expand when selected.

SwitchBreakpoint

tchmi:general#/definitions/Number

Determines at which pixel width the navigation changes to the Hamburger symbol.
EventCount
tchmi:general#/definitions/Number
Here you can link a symbol that contains the number of active events in the event list. This number is then displayed in a badge on the button for the event list.

CurrentTemperature
tchmi:general#/definitions/Number
Current temperature to be displayed in the header. Typically, the outdoor temperature is displayed here.

CurrentTemperatureUnit
tchmi:general#/definitions/String
Unit of the current temperature.

CloseMenu
tchmi:general#/definitions/Boolean
If the attribute has the value TRUE, the header menus can be closed (this can be changed during runtime).

NavContent
Here a control of type TcHmi.Controls.ResponsiveNavigation.TcHmiNavigationContent must be linked, in which the content of the responsive navigation is displayed.

UserContent
Here a control of type TcHmi.Controls.ResponsiveNavigation.TcHmiNavigationContent must be linked, in which the content of the user menu is displayed.

TargetRegion
tchmi:framework#/definitions/TcHmiRegion
Here the TcHmiRegion must be linked, which is used to display the content pages selected from the menu.

EventContent
tchmi:framework#/definitions/ContentPath
Content on which the event list is located.

StartPage
tchmi:framework#/definitions/ContentPath
Start page of the HMI. This page is loaded into the TargetRegion when you click the logo.
8 Tools

8.1 Symbol Explorer

8.1.1 Introduction
With the Symbol Explorer you can access controllers online via the ADS communication interface:

• You can read them as persistently declared variables.
• The Symbol Explorer provides functions with which variable backups can be created. These can be duplicated for follow-up projects and uploaded to other controllers.
• Variables can be checked for differences and merged using a compare function.

8.1.2 Definitions

8.1.2.1 Symbol
A symbol is a type for describing variables of a Beckhoff controller. If variables are read from a controller using the Symbol Explorer, information such as name, type, size, subvariables and many other parameters are merged into one variable. A symbol is then formed from this quantity of parameters.

Symbol types
A symbol can describe different types of variables.

Complex symbols describe variables that can consist of function blocks, arrays and structures.

Primitive symbols describe variables that can consist of basic types, INT, REAL, BOOL, etc.

8.1.2.2 Snapshot
A snapshot describes a snapshot of a symbol or symbol structure.

The function to create a snapshot is provided for several use cases. You can create a snapshot of all the symbols and thus have a complete backup of the symbols on a controller. However, using the various representations and filter functions in the symbol list, you can also create just "sections of a symbol/symbols" and save them in a snapshot. You have thus created a template for the simplified duplication of symbol values.

8.1.3 Description of the interface
The interface is divided into various windows with different tasks for using the Symbol Explorer.

8.1.3.1 Main window
After opening the start page, the following options can be selected:

• Recent snapshots
• Recent routes
• Connect
• Snapshot
8.1.3.1.1 Menu

The menu is divided into the following tabs.

**Open**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect route</td>
<td>Connect to a route</td>
</tr>
<tr>
<td>Open snapshot</td>
<td>Open a snapshot</td>
</tr>
<tr>
<td>Recent snapshots</td>
<td>List of frequently used snapshots</td>
</tr>
<tr>
<td>Recent routes</td>
<td>List of frequently used routes</td>
</tr>
</tbody>
</table>

**Tools**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare</td>
<td>Comparison of two symbol lists</td>
</tr>
</tbody>
</table>

**Help**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>About TwinCAT Symbol Explorer</td>
<td>Version specification</td>
</tr>
</tbody>
</table>

8.1.3.1.2 Start Page button

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Connect route</td>
<td>Connect to the controller</td>
</tr>
<tr>
<td></td>
<td>Open snapshot</td>
<td>Open a snapshot</td>
</tr>
<tr>
<td></td>
<td>Recent snapshots</td>
<td>List of frequently used snapshots</td>
</tr>
<tr>
<td></td>
<td>Recent routes</td>
<td>List of frequently used routes</td>
</tr>
</tbody>
</table>

8.1.3.2 Output window

The Output window displays information, errors and warnings at runtime.

The functions of the toolbar are described in the following table.
## Toolbar

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Expand / Collapse</td>
<td>Expands and collapses the Output window</td>
</tr>
<tr>
<td></td>
<td>Clear messages</td>
<td>Clears the messages</td>
</tr>
<tr>
<td></td>
<td>Show / Hide errors</td>
<td>Shows and hides the error messages</td>
</tr>
<tr>
<td></td>
<td>Show / Hide warnings</td>
<td>Shows and hides the warnings</td>
</tr>
<tr>
<td></td>
<td>Show / Hide messages</td>
<td>Shows and hides the messages</td>
</tr>
</tbody>
</table>

### 8.1.3.3 Online window

The Online window displays the online symbols of a controller. Functions for creating backups, monitoring and editing symbols are available via the toolbar and the context menus.

## Toolbar

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Expand all</td>
<td>Expands the list of the symbols and sub-symbols</td>
</tr>
<tr>
<td></td>
<td>Collapse all</td>
<td>Collapses the list of the symbols and sub-symbols</td>
</tr>
<tr>
<td></td>
<td>Reset filter</td>
<td>Sets the symbol filter to the standard setting</td>
</tr>
<tr>
<td></td>
<td>Changed symbols</td>
<td>Displays changed symbols in a list</td>
</tr>
<tr>
<td></td>
<td>Upload symbols</td>
<td>Writes changed symbols to the controller</td>
</tr>
<tr>
<td></td>
<td>Take snapshot</td>
<td>Takes a snapshot of the symbols</td>
</tr>
<tr>
<td></td>
<td>Refresh</td>
<td>Refreshes symbol value once</td>
</tr>
<tr>
<td></td>
<td>Auto refresh</td>
<td>Refreshes symbol value every 2 seconds</td>
</tr>
<tr>
<td></td>
<td>Export</td>
<td>Exports list of symbols</td>
</tr>
<tr>
<td></td>
<td>Close window</td>
<td>Close window</td>
</tr>
</tbody>
</table>

### NOTE

**Exclusions when refreshing**

Symbols that have been edited via the Symbol Explorer and symbols that are currently being edited are excluded from refreshing.
Notes on ADS communication

The Symbol Explorer uses the ADS protocol for the online communication. Note that ADS is only a transport layer; however, side effects can occur. Read these requirements and observe the restrictions:
ADS itself is only the transport layer, the requested ADS device must support the ADS command. If the PLC processes an ADS request (reading/writing of the symbol values), it will work completely on this single ADS request before the start of a new PLC cycle.
To keep the load on the controller low, the number of symbols to be read or written is for this reason limited to a maximum of 250 per ADS request.

8.1.3.4 Snapshot window

The Snapshot window displays the offline symbols from a snapshot. Functions for editing, uploading and creating copy templates are available via the toolbar and the context menus.

Toolbar

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Expand all</td>
<td>Expands the list of the symbols and sub-symbols</td>
</tr>
<tr>
<td></td>
<td>Collapse all</td>
<td>Collapses the list of the symbols and sub-symbols</td>
</tr>
<tr>
<td></td>
<td>Reset filter</td>
<td>Sets the symbol filter to the standard setting</td>
</tr>
<tr>
<td></td>
<td>Changed symbols</td>
<td>Displays changed symbols in a list</td>
</tr>
<tr>
<td></td>
<td>Copy symbols</td>
<td>Copies symbol values to another symbol list</td>
</tr>
<tr>
<td></td>
<td>Save</td>
<td>Saves a snapshot</td>
</tr>
<tr>
<td></td>
<td>Export</td>
<td>Exports list of symbols</td>
</tr>
<tr>
<td></td>
<td>Close window</td>
<td>Close window</td>
</tr>
</tbody>
</table>

8.1.3.5 Comparison window

The Symbol Compare window displays the offline symbols from a snapshot. Functions for editing, uploading and creating copy templates of the symbols are available via the toolbar and the context menus.
The Symbol Compare window displays the following differences:

- Differences from symbol to symbol
- Removed symbols
- Added symbols
- Missing symbols

### Toolbar

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="previous_change_icon" alt="Prev" /></td>
<td>Previous change</td>
<td>Go to previous change</td>
</tr>
<tr>
<td><img src="next_change_icon" alt="Next" /></td>
<td>Next change</td>
<td>Go to next change</td>
</tr>
<tr>
<td><img src="first_change_icon" alt="First" /></td>
<td>First change</td>
<td>Go to first change</td>
</tr>
<tr>
<td><img src="undo_icon" alt="Undo" /></td>
<td>Undo all</td>
<td>Reset changes</td>
</tr>
<tr>
<td><img src="last_change_icon" alt="Last" /></td>
<td>Last change</td>
<td>Go to last change</td>
</tr>
<tr>
<td><img src="copy_left_icon" alt="CopyLeft" /></td>
<td>Copy all changes to left</td>
<td>Copies all changes into the left-hand list</td>
</tr>
<tr>
<td><img src="copy_right_icon" alt="CopyRight" /></td>
<td>Copy all changes to right</td>
<td>Copies all changes into the right-hand list</td>
</tr>
<tr>
<td><img src="copy_selection_icon" alt="CopySel" /></td>
<td>Copy change to left</td>
<td>Copies the selected row into the left-hand list</td>
</tr>
<tr>
<td><img src="copy_selection_icon" alt="CopySel" /></td>
<td>Copy change to right</td>
<td>Copies the selected row into the right-hand list</td>
</tr>
<tr>
<td><img src="close_icon" alt="Close" /></td>
<td>Close window</td>
<td>Close window</td>
</tr>
</tbody>
</table>
8.1.3.6 Symbol list

By means of the hotkeys and context menus, the symbol list provides functions such as Edit values or Copy/Paste. Furthermore, the symbol list offers a filter function, with which symbols to be edited can be purposefully filtered and highlighted.

### Hotkeys

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl + left-click</td>
<td>Expands or collapses the symbol and its sub-symbols.</td>
</tr>
<tr>
<td>Ctrl + left-click</td>
<td>Selective multi-selection on a row</td>
</tr>
<tr>
<td>Ctrl + C</td>
<td>Copies the selected symbol (and all sub-symbols)</td>
</tr>
<tr>
<td>Ctrl + V</td>
<td>Pastes the copied symbol values into the selected symbol (and the sub-symbols)</td>
</tr>
<tr>
<td>Ctrl + Shift + C</td>
<td>Copies the contents of the selected cell</td>
</tr>
<tr>
<td>Ctrl + R</td>
<td>Resets the symbol value to the previous value</td>
</tr>
<tr>
<td>Shift + left-click</td>
<td>Multi-selection</td>
</tr>
<tr>
<td>Double left-click</td>
<td>Starts the editing of a symbol</td>
</tr>
<tr>
<td>F2</td>
<td>If a symbol had been selected beforehand, editing starts</td>
</tr>
</tbody>
</table>

### Context menu

<table>
<thead>
<tr>
<th>Selection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Reads the current value from the PLC on the selected symbol</td>
</tr>
<tr>
<td>Copy Symbol Content</td>
<td>Copies the selected symbol (and all subsystems)</td>
</tr>
<tr>
<td>Paste Symbol Content</td>
<td>Pastes the copied symbol values into the selected symbol (and the sub-symbols)</td>
</tr>
<tr>
<td>Copy Cell Value</td>
<td>Copies the contents of the selected table cell</td>
</tr>
<tr>
<td>Reset</td>
<td>Resets the symbol value to the previous value</td>
</tr>
</tbody>
</table>

Copying and pasting behaves differently in the Symbol Explorer than in other software tools. If a symbol is copied in the Symbol Explorer and pasted to a different symbol, the symbol is not copied like a file, e.g. in Windows Explorer; instead, only the symbol values from the copying source are transferred to the target symbol.

8.1.4 Find and replace

By means of the individual filtering of the symbols, it is possible to find the desired symbols faster. You can filter for the name, type or comment of a symbol.
A function is a subordinate quantity of functions, parameters, operators and value pairs. A function begins and ends with brackets (). Each function, parameter, operator and value pair is followed by a semicolon.

### 8.1.4.1 AND function

The AND function is used to filter for symbols whose conditions are satisfied by a TRUE. Two examples of the use of the AND function and in conjunction with the OR function are shown below.

```plaintext
AND ( Type = BOOL;
     Name = 'bEnabled';
     Value = True )
```

Filters for symbols with the type BOOL whose name is `bEnabled` and whose value is TRUE.

```plaintext
AND ( Name = nCounter;
      OR ( Value = 10;
           Value = 50 ))
```

Filters for symbols with the name `nCounter` whose value is '10' or '50'.
8.1.4.1.2 OR function

The OR function is used to filter for symbols where one condition is satisfied by a TRUE. Two examples of the use of the OR function and in conjunction with the AND function are shown below.

| OR ( Value = 150;  
| Value = -150 ) |

The function filters for symbols whose value is '150' or '-150'.

| AND ( Name = 'nCounter';  
| OR ( Value = 10;  
| Value = 50 )) |

The function filters for symbols with the name 'nCounter' whose value is '10' or '50'.

8.1.4.1.3 PARENT function

The PARENT function is used to filter for symbols whose direct parent symbol (higher-level symbol) is subject to an identical condition. The conditions specified in the PARENT function are logically ANDed.

The PARENT function can only be used in an AND function or in an OR function.

Here is a general example of the use of the PARENT function:

| AND ( Name = 'sObjectName';  
| PARENT ( Name = 'Plant';  
| Value = TRUE )) |

The function filters for symbols with the name 'sObjectName' and their parent symbol with the name 'Plant'. Furthermore, the value of this symbol must be TRUE.

8.1.4.1.4 ANCESTOR function

The ANCESTOR function is used to filter symbols whose ancestors in the parent chain of symbols are subject to a condition, e.g. a special name or type. The condition specified in the ANCESTOR function is logically ANDed.

The ANCESTOR function can only be used in an AND function or in an OR function.

Here are some general examples of the use of the ANCESTOR function:

| AND ( Name = 'sObjectName';  
| ANCESTOR ( Type = 'FB_BAC_AI')) |

The function filters for symbols with the name 'sObjectName'. The ancestors of these symbols come from the parent chain of symbols with the type 'FB_BAC_AI'.

| AND ( Name = 'sObjectName';  
| ANCESTOR ( Name = 'Plant01')) |

The function filters for symbols with the name 'sObjectName'. Within the parent chain of these symbols, one of the ancestors bears the name 'Plant01'.

8.1.4.1.5 CHILD function

The CHILD function is used to filter symbols whose child symbols are subject to a condition. The conditions specified in the CHILD function are logically ANDed.
The CHILD function can only be used in an AND function or in an OR function.

Here is a general example of the use of the CHILD function:

```
AND (   Name = 'Plant';
Child { Name = 'sObjectName';
        Value = 'P001' }
```

### 8.1.4.1.6 Value Parameter

The function searches for symbols whose value satisfies a condition with a certain numerical value.

**Examples:**

- `Value = '100'`  
  Search for symbols whose value is '100'.
- `Value > '100'`  
  Search for symbols whose value is greater than '100'.
- `Value < '100'`  
  Search for symbols whose value is smaller than '100'.
- `Value != '100'`  
  Search for symbols whose value is not '100'.
- `Value = ''`  
  Search for symbols whose value is empty ''. 

### 8.1.4.1.7 Comment Parameter

The function searches for symbols whose value satisfies a condition with a certain comment.

**Examples:**

- `Comment = 'Value'`  
  Search for symbols whose comment is 'Value'.
- `Comment > 'Value'`  
  Search for symbols whose comment is greater than 'Value'.
- `Comment < 'Value'`  
  Search for symbols whose comment is smaller than 'Value'.
- `Comment != 'Value'`  
  Search for symbols whose comment is not 'Value'.
- `Comment = ''`  
  Search for symbols whose comment is empty ''. 

Search for symbols whose comment is 'Signal to detect'.

Search for symbols whose comment is not 'Signal to detect'.

Search for symbols whose comment begins with 'Signal'.

Search for symbols whose comment ends with 'detect'.

**8.1.4.1.8 Name Parameter**

The function searches for symbols whose value satisfies a condition with a certain name.

Examples:

Name = 'bEnable'

Search for symbols whose name is 'bEnable'.

Name != 'bEnable'

Search for symbols whose name is not 'bEnable'.

Name > 'bEn'

Search for symbols whose name begins with 'bEn'.

Name < 'le'

Search for symbols whose name ends with 'le'.

**8.1.4.1.9 Type Parameter**

The function searches for symbols whose value corresponds to a condition of a certain type.

Examples:
8.1.5 **Principle of operation**

This chapter describes step by step how to work with the Symbol Explorer and is intended to provide an overview of its functions.

8.1.5.1 **Connecting to a controller**

The Symbol Explorer communicates with a controller via the ADS communication interface. So that communication with a controller can be successfully established, the following conditions must be satisfied:

- The controller can be reached via the network.
- TwinCAT is in Run Mode.
- An AMS route to the controller has been set up.
- A PLC runtime has been activated and started.

If the criteria are fulfilled, a connection can be established and the symbols read out. It is only possible to read out symbols that have been declared as persistent and symbols that contain persistently declared symbols.

**Starting the connection to a controller**

Start the Symbol Explorer and click the **Connect** button on the start page. In the **Choose Route** dialog, select the **Ams Route** belonging to the controller to which you wish to connect.

If you were able to successfully connect to the controller, the online window opens, showing you the read-out symbols in a list.

8.1.5.2 **Taking a snapshot**

Start the **Symbol Explorer** and connect to a controller. Click the **Take Snapshot** button in the online window in order to take the snapshot.

You will then be requested to select a location to save the snapshot file. If you have selected the folder and confirmed with **Ok**, all values of the symbols in the list are synchronized with the PLC and written to the file.

8.1.5.3 **Loading a snapshot**

Open the **Symbol Explorer** and click the **Snapshot** button on the start page. You will then be requested to select a snapshot file. Select a snapshot file to be loaded and confirm with **Ok**.

The snapshot window then opens, showing you the loaded symbols in a list.
8.1.5.4 Viewing and editing symbol lists in parallel

The Symbol Explorer can display two symbol lists in parallel. It is thus possible, for example, to place the current symbols (online) and symbols from a snapshot next to one another and to edit symbol values between these two lists. Simply connect to a controller and then open a snapshot. A different order of opening/connecting is also possible.

8.1.5.5 Copying a snapshot to a controller

Open and connect the Symbol Explorer to a controller. Then open a snapshot. You can now see the Online and Snapshot windows side by side. Press the Copy Symbols button in the snapshot window. The Symbol Transfer Protocol dialog then opens:
The dialog informs you in detail about the copying procedure. If the symbol values are correct, confirm the copying procedure by clicking **Ok**, otherwise select **Cancel**. If you have confirmed the copying procedure, the symbol values from the snapshot are copied into the online symbol list.

Once the copy procedure is completed, the symbols with changed values are highlighted by a red square in front of the symbol name. You can now load the changes from the Online window into the controller. To do this, press the **Upload Symbol** button in the toolbar.

### 8.1.5.6 Comparing symbols and synchronizing differences

With the Symbol Explorer it is possible to compare two lists of symbols. You can thus compare online symbols with symbols from a snapshot and check for differences.

#### 8.1.5.6.1 Starting a comparison

Connect to a controller or load a snapshot. Repeat the procedure so that two symbol lists stand side by side. Then press the hotkey **[F8]** or select **Compare Symbols** in the Tools menu. The comparison window then opens.

#### 8.1.5.6.2 Synchronizing differences

During synchronization, differences are copied from one symbol to another, either from left to right or from right to left. This procedure differs from the direct editing of a symbol via the Online or Snapshot window.

#### 8.1.5.6.3 Synchronization functions

The synchronization functions can be applied implicitly to differences on the basis of a row selection. For example, if you click a row with a difference and then the right arrow button on the central toolbar, the symbol value is copied from the left into the symbol on the right. This allows the easy merging of many small differences.
8.1.6 Appendix

8.1.6.1 Examples of regular expressions

For an understanding of the expressions used for filtering and sorting, the following link opens a collection of frequently used regular expressions:

Overview of regular expressions

8.2 Template Repository

The Template Repository is an app for managing templates.

Starting the app in the development environment

The TF8040 setup automatically installs the Template Repository in the TwinCAT development environment (XAE Shell or Visual Studio). The following steps describe how to start the app:

1. Right-click on a free space in the toolbar.
2. In the context menu, select the Template Repository app.
3. Start the app by clicking the icon TwinCAT Template Repository. The main window opens.

Main Window

The main window is divided into 6 different areas:
1 Menu bar

The following functions are available in the menu bar:

- **Refresh Template Repositories** updates all repositories

- **Show LogFile** opens the log file dialog

Detailed log information about the software version used and errors are displayed here.

2 Search bar
In the search bar, the integrated repositories are searched for keywords (*tags*) or names within the templates. The search results are displayed in the respective areas.

3 TwinCAT repository area

The TwinCAT repository area contains all templates that are supplied with the TF8040 installation. Templates can be taken from this area but not added.

---

**Nature and source of the danger**

With the TF8040 installation, you receive a repository with templates. In order to use them for your project-specific applications, you must temporarily load the repository into the global area.

---

4 Global Repository area

The Global Repository area serves as a storage location for all customer-specific templates that can be used by several developers across projects. Here templates can be taken and added across projects. The storage location can be a network drive in the company network or any location on the local hard disk, for example.

5 Local Repository area

All templates that are used in the local solution are stored in the Local Repository area.
Templates can be exchanged between the local and global area.

6 Notification area

The notification area displays detailed information about the currently selected templates.

Adding templates to a PLC project

Templates can be integrated into the current solution in two ways:

- Double-click the desired template
- Right click the desired template -> Add to -> PLC Project
The dialog box Add Template to PLC Project opens.

**Add Template dialog**

Select the template in the following dialog and confirm with OK.

The desired template including all dependent subtemplates now appears in the PLC project.
Call and instantiation of a template

After the desired template has been added to the project from the repository, it can be instantiated and called in the project applications.

**Instantiation in the declaration part:**

```
Pu : FB_BA_Pulst;
```

**Call in the program part:**
9 Appendix

9.1 Third-party components

This software contains third-party components. Please refer to the license file provided in the following folder for further information: \TwinCAT\Functions\SymbolExplorer\Licenses.

9.2 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
More Information:
www.beckhoff.com/tf8040