Table of contents

1 Foreword ........................................................................................................................................... 5
  1.1 Notes on the documentation ................................................................................................. 5
  1.2 Safety instructions ............................................................................................................... 6

2 Introduction ................................................................................................................................. 7

3 Integration in TwinCAT ............................................................................................................... 8
  3.1 System requirements ........................................................................................................... 8
  3.2 Installation .......................................................................................................................... 8
  3.3 Licensing ............................................................................................................................ 11

4 Programming ............................................................................................................................... 14
  4.1 POUs ......................................................................................................................................... 14
    4.1.1 Conversion functions ...................................................................................................... 14
    4.1.2 Energy management ....................................................................................................... 15
    4.1.3 Facade ............................................................................................................................ 19
    4.1.4 Filter functions ................................................................................................................ 27
    4.1.5 Lighting .......................................................................................................................... 32
    4.1.6 Scene management ......................................................................................................... 57
    4.1.7 Signal processing ............................................................................................................. 67
    4.1.8 Timer functions ............................................................................................................... 71
    4.1.9 Error Codes ..................................................................................................................... 92
  4.2 DUTs .......................................................................................................................................... 92
    4.2.1 Enumerations .................................................................................................................... 92
    4.2.2 Structures ......................................................................................................................... 93
  4.3 GVLs .......................................................................................................................................... 95
    4.3.1 Constants ........................................................................................................................ 95

5 Appendix ....................................................................................................................................... 96
  5.1 Support and Service ............................................................................................................... 96
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®, 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>
<tr>
<td><img src="image" alt="Tip or pointer" /></td>
<td>This symbol indicates information that contributes to better understanding.</td>
</tr>
</tbody>
</table>
2 Introduction

The library offers users basic functions for room automation and building control.

The user of this library requires basic knowledge of the following:

- TwinCAT XAE
- Design and characteristics of Beckhoff IPCs and their Bus Terminal system
- Relevant safety regulations for building technical equipment

This software library 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.

The Tc3 Building Automation Basic library can be used on all hardware platforms that support TwinCAT 3.1 or higher.
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 CE 7</td>
</tr>
<tr>
<td>Target platform</td>
<td>PC architecture (x86, x64 or ARM)</td>
</tr>
<tr>
<td>TwinCAT version</td>
<td>TwinCAT 3.1 build 4020.32 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>TF8010_TC3 Building Automation Basic</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.

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

![Image of TwinCAT installation dialog box]

- A dialog box informs you that the TwinCAT system must be stopped to proceed with the installation.

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

![Beckhoff Setup Completed](image)

- The TwinCAT 3 Function has been successfully installed and can be licensed (see [Licensing](#11))

### 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. "TF6420: TC3 Database Server").

![Manage Licenses](image)

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

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

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

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 Programming

The TwinCAT PLC Building Automation library contains useful function blocks for building automation.

4.1 POUs

4.1.1 Conversion functions

4.1.1.1 F_Scale

A raw analog value is scaled to the specified measuring range and returned as the function value. If the raw value extends beyond the upper or lower measuring range, the corresponding limit value is output. There must be a difference of at least 0.01 between the upper and lower limit values for the raw data. If this is not the case, the lower limit value is output.

FUNCTION F_Scale: LREAL

VAR_INPUT
fRawData: raw value.
fRawDataLowerOffLimit: Lower limit for raw value.
fRawDataUpperOffLimit: Upper limit for raw value.
fScaleDataLowerOffLimit: Lower limit of scaled measured value.
fScaleDataUpperOffLimit: Upper limit of scaled measured value.
4.1.1.2 Temperature conversion functions

between Kelvin, Celsius, Reaumur and Fahrenheit.

<table>
<thead>
<tr>
<th>Function</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_TO_C</td>
<td>°F → °C</td>
</tr>
<tr>
<td>F_TO_K</td>
<td>°F → °K</td>
</tr>
<tr>
<td>F_TO_R</td>
<td>°F → °R</td>
</tr>
<tr>
<td>K_TO_C</td>
<td>°K → °C</td>
</tr>
<tr>
<td>K_TO_F</td>
<td>°K → °F</td>
</tr>
<tr>
<td>K_TO_R</td>
<td>°K → °R</td>
</tr>
<tr>
<td>C_TO_F</td>
<td>°C → °F</td>
</tr>
<tr>
<td>C_TO_K</td>
<td>°C → °K</td>
</tr>
<tr>
<td>C_TO_R</td>
<td>°C → °R</td>
</tr>
<tr>
<td>R_TO_K</td>
<td>°R → °K</td>
</tr>
<tr>
<td>R_TO_C</td>
<td>°R → °C</td>
</tr>
<tr>
<td>R_TO_F</td>
<td>°R → °F</td>
</tr>
</tbody>
</table>

Overview

<table>
<thead>
<tr>
<th>Kelvin (K)</th>
<th>Degrees Celsius (°C)</th>
<th>Reaumur (°R)</th>
<th>Fahrenheit (°F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute zero</td>
<td>0</td>
<td>-273.15</td>
<td>-218.52</td>
</tr>
<tr>
<td>Melting point</td>
<td>273.15</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Boiling point</td>
<td>373.15</td>
<td>100</td>
<td>80</td>
</tr>
</tbody>
</table>

(The melting and boiling points refer to pure water.)

Conversion rules

<table>
<thead>
<tr>
<th>x = Kelvin (K)</th>
<th>Kelvin (K)</th>
<th>Degrees Celsius (°C)</th>
<th>Reaumur (°R)</th>
<th>Fahrenheit (°F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = degrees Celsius (°C)</td>
<td>= x + 273.15</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>x = Reaumur (°R)</td>
<td>-</td>
<td>= 5/4 x + 273.15°K</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>x = Fahrenheit (°F)</td>
<td>-</td>
<td>-</td>
<td>= 9/5 x + 32°F</td>
<td>-</td>
</tr>
</tbody>
</table>

4.1.2 Energy management

4.1.2.1 FB_MaximumDemandController
Function block for peak load optimization, which ensures that the set power limit is maintained by switching up to eight consumers on or off. The consumers can be switched off according to their power and priority in such a manner that the production process is not disturbed.

In order to distinguish the individual measurement cycles, a synchronization pulse is supplied by the electricity supply company. It indicates the start of a new measuring cycle and must be linked to the input $b_{\text{PeriodPulse}}$. The actual power is recorded via the KL1501 counter terminal.

The function block works with a fixed measuring period of 15 minutes. If the synchronization pulse exceeds the 16 minute limit, the output $b_{\text{EmergencySignal}}$ is set.

All consumers are switched on at the start of each measuring period. In the event that the power limit ($f_{\text{AgreedPower}}$) threatens to be exceeded within the measuring period, the consumers are switched off one after the other. If the danger of an excess load no longer exists, the consumers are switched on again.

Special items, such as minimum power-on time, minimum power-off time or maximum power-off time can be specified via an input variable. The priority of the individual consumers can similarly be determined. Consumers with a low priority will be switched off before consumers with a high priority.

---

**VAR_INPUT**

- $b_{\text{Start}}$: The function block is activated by a positive edge at this input.
- $f_{\text{MeterConstant}}$: Meter constant [pulses / kWh].
- $f_{\text{AgreedPower}}$: This is the agreed power limit which, as far as possible, should not be exceeded in the operational case [kW].

---

**Diagram Description**

- Abschaltung Gerät #1 mit der niedrigsten Priorität
- Zuschaltung Gerät #1
- Abstimmung Gerät #1 mit der niedrigsten Priorität
- Zuschaltung Gerät #2
- Abschaltung Gerät #2 mit der zweit-niedrigsten Priorität

---

**Sollleistung**

**Istleistung**

**Dauer einer gesamten 1/4h Messperiode**

---

**Version:** 1.0

TF8010 Building Automation Basic
bPeriodPulse: Synchronization pulse sent by the electricity supply company (ESC). This pulse starts the measurement interval.

arrLoadParameter: Parameter structure of the respective consumer (see ST_MDCLoadParameters [93]).

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrLoad</td>
<td>ARRAY[1..8] OF BOOL;</td>
</tr>
<tr>
<td>fAgreedEnergy</td>
<td>LREAL;</td>
</tr>
<tr>
<td>fInstantaneousEnergy</td>
<td>LREAL;</td>
</tr>
<tr>
<td>fActualEnergy</td>
<td>LREAL;</td>
</tr>
<tr>
<td>tRemainingTime</td>
<td>TIME;</td>
</tr>
<tr>
<td>fLastPeriodEnergy</td>
<td>LREAL;</td>
</tr>
<tr>
<td>bEmergencySignal</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL;</td>
</tr>
<tr>
<td>nErrorId</td>
<td>UDINT;</td>
</tr>
</tbody>
</table>

arrLoad: This is an array of data type BOOL; consumers that are switched on are TRUE.

fAgreedEnergy: Agreed energy consumption [kWh].

fInstantaneousEnergy: Current energy consumption [kWh] over an integration period of 15s (internal measuring interval).

fActualEnergy: Energy consumed at the "presently" observed point in time of the measuring period.

tRemainingTime: Time remaining until the next measurement interval.

fLastPeriodEnergy: Rated power from the preceding measuring period [kWh].

bEmergencySignal: This output is set as soon as the specified energy is exceeded.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command.

nErrorId: Contains the error code [92].

**VAR_IN_OUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>stInDataKL1501</td>
<td>ST_MDCInDataKL1501;</td>
</tr>
<tr>
<td>stOutDataKL1501</td>
<td>ST_MDCOutDataKL1501;</td>
</tr>
</tbody>
</table>

stInDataKL1501: Linked to KL1501 (see ST_MDCInDataKL1501 [93]).

stOutDataKL1501: Linked to KL1501 (see ST_MDCOutDataKL1502 [94]).

**Example**

```
VAR_GLOBAL
  arrLoadParameters AT %MB100 : ARRAY[1..8] OF ST_MDCLoadParameters;
  (* KL1002 *)
    bPeriodPulse AT %IX6.0 : BOOL;
  (* KL1501 *)
    stInDataKL1501 AT %IB0 : ST_MDCInDataKL1501;
    stOutDataKL1501 AT %QB0 : ST_MDCOutDataKL1501;
  (* KL2404 *)
    bLoadOut1 AT %QX6.0 : BOOL;
    bLoadOut2 AT %QX6.1 : BOOL;
    bLoadOut3 AT %QX6.2 : BOOL;
    bLoadOut4 AT %QX6.3 : BOOL;
  (* KL2404 *)
    bLoadOut5 AT %QX6.4 : BOOL;
    bLoadOut6 AT %QX6.5 : BOOL;
    bLoadOut7 AT %QX6.6 : BOOL;
    bEmergencySignal AT %QX6.7 : BOOL;
END_VAR
```

PROGRAM MAIN
VAR
  fbMaximumDemandController : FB_MaximumDemandController;
END_VAR
arrLoadParameters[1].bConnected := TRUE;
arrLoadParameters[1].nDegreeOfPriority := 1;
arrLoadParameters[1].tMINPowerOnTime := t#60s;
arrLoadParameters[1].tMINPowerOffTime := t#120s;
arrLoadParameters[1].tMAXPowerOffTime := t#600s;

arrLoadParameters[2].bConnected := TRUE;
arrLoadParameters[2].nDegreeOfPriority := 2;
arrLoadParameters[2].tMINPowerOnTime := t#60s;
arrLoadParameters[2].tMINPowerOffTime := t#120s;
arrLoadParameters[2].tMAXPowerOffTime := t#600s;

arrLoadParameters[3].bConnected := TRUE;
arrLoadParameters[3].nDegreeOfPriority := 3;
arrLoadParameters[3].tMINPowerOnTime := t#60s;
arrLoadParameters[3].tMINPowerOffTime := t#120s;
arrLoadParameters[3].tMAXPowerOffTime := t#300s;

arrLoadParameters[4].bConnected := TRUE;
arrLoadParameters[4].nDegreeOfPriority := 4;
arrLoadParameters[4].tMINPowerOnTime := t#20s;
arrLoadParameters[4].tMINPowerOffTime := t#30s;
arrLoadParameters[4].tMAXPowerOffTime := t#8m;

arrLoadParameters[5].bConnected := TRUE;
arrLoadParameters[5].nDegreeOfPriority := 5;
arrLoadParameters[5].tMINPowerOnTime := t#20s;
arrLoadParameters[5].tMINPowerOffTime := t#50s;
arrLoadParameters[5].tMAXPowerOffTime := t#20m;

arrLoadParameters[6].bConnected := TRUE;
arrLoadParameters[6].nDegreeOfPriority := 6;
arrLoadParameters[6].tMINPowerOnTime := t#30s;
arrLoadParameters[6].tMINPowerOffTime := t#1m;
arrLoadParameters[6].tMAXPowerOffTime := t#1m;

arrLoadParameters[7].bConnected := TRUE;
arrLoadParameters[7].nDegreeOfPriority := 7;
arrLoadParameters[7].tMINPowerOnTime := t#0s;
arrLoadParameters[7].tMINPowerOffTime := t#0s;
arrLoadParameters[7].tMAXPowerOffTime := t#1m;

arrLoadParameters[8].bConnected := FALSE;

fbMaximumDemandController(bStart := TRUE,
fMeterConstant := 20000,
fAgreedPower := 600,
bPeriodPulse := bPeriodPulse,
arrLoadParameters := arrLoadParameters,
stInDataKL1501 := stInDataKL1501,
stOutDataKL1501 := stOutDataKL1501);

bLoadOut1 := fbMaximumDemandController.arrLoad[1];
bLoadOut2 := fbMaximumDemandController.arrLoad[2];
bLoadOut3 := fbMaximumDemandController.arrLoad[3];
bLoadOut4 := fbMaximumDemandController.arrLoad[4];
bLoadOut5 := fbMaximumDemandController.arrLoad[5];
bLoadOut6 := fbMaximumDemandController.arrLoad[6];
bLoadOut7 := fbMaximumDemandController.arrLoad[7];
bEmergencySignal := fbMaximumDemandController.bEmergencySignal;

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>
4.1.3 Facade

4.1.3.1 FB_RoofWindow

The outputs \( b\text{WindowOpen} \) or \( b\text{WindowClose} \) are set through a positive edge at the inputs \( b\text{Open} \) or \( b\text{Close} \). These remain asserted until the time \( t\text{TurnOffTime} \) has elapsed, or until the function block receives some other command. Both outputs are immediately reset by a positive edge at the \( b\text{Stop} \) input.

The \( t\text{SwitchOverDeadTime} \) can be used to prevent damage to the drive motor caused by immediate changes in direction. In most cases, this value is between 0.5 s and 1.0 s. The drive manufacturer can give you a precise value.

Safety position

Moving to the safety position (e.g. in strong wind) can be achieved by setting the input \( b\text{SafetyPosition} \). The output \( b\text{WindowClose} \) is set for the time \( t\text{TurnOffTime} \), and the output \( b\text{WindowOpen} \) is reset. Window operation is disabled as long as the \( b\text{SafetyPosition} \) input is active.

\[
\text{VAR_INPUT}
\begin{align*}
b\text{Close} & : \text{BOOL}; \\
b\text{Open} & : \text{BOOL}; \\
b\text{Stop} & : \text{BOOL}; \\
b\text{SafetyPosition} & : \text{BOOL}; \\
b\text{LimitSwitchClose} & : \text{BOOL}; \\
b\text{LimitSwitchOpen} & : \text{BOOL}; \\
t\text{TurnOffTime} & : \text{TIME} := t\#60s; \\
t\text{SwitchOverDeadTime} & : \text{TIME} := t\#400ms;
\end{align*}
\]

\( b\text{Close} \): Set output \( b\text{WindowClose} \) and reset output \( b\text{WindowOpen} \). The output \( b\text{WindowClose} \) remains latched.

\( b\text{Open} \): Set output \( b\text{WindowOpen} \) and reset output \( b\text{WindowClose} \). The output \( b\text{WindowOpen} \) remains latched.

\( b\text{Stop} \): Reset outputs \( b\text{WindowClose} \) and \( b\text{WindowOpen} \)

\( b\text{SafetyPosition} \): The safety position is approached. The window is closed for the time \( t\text{TurnOffTime} \). Window operation is blocked as long as the input is active.

\( b\text{LimitSwitchClose} \): Optional limit switch. If \( b\text{Close} \) is set and is not set within \( t\text{TurnOffTime} \), \( b\text{ErrorLimitSwitchClose} \) is set.

\( b\text{LimitSwitchOpen} \): Optional limit switch. If \( b\text{Open} \) is set and is not set within \( t\text{TurnOffTime} \), \( b\text{ErrorLimitSwitchOpen} \) is set.

\( t\text{TurnOffTime} \): If no input is activated, then the outputs are reset after this period of time. The outputs are not automatically reset if the specified duration is 0. The value given here should be about 10% larger than the travel time that is actually measured.

\( t\text{SwitchOverDeadTime} \): Dwell time at a change of direction. Both outputs are reset during this period.
VAR_OUTPUT

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bWindowOpen</td>
<td>The window opens.</td>
</tr>
<tr>
<td>bWindowClose</td>
<td>The window closes.</td>
</tr>
<tr>
<td>bErrorLimitSwitchClose</td>
<td>Error relating to the optional limit switch during closing.</td>
</tr>
<tr>
<td>bErrorLimitSwitchOpen</td>
<td>Error relating to the optional limit switch during opening.</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

4.1.3.2 FB_VenetianBlind

There are three different ways in which the blinds may be controlled:

- A positive edge at the bUp or bDown inputs sets the bBlindUp or bBlindDown outputs respectively. These remain asserted until the time tTurnOffTime has elapsed, or until the function block receives some other command. Both outputs are immediately reset by a positive edge at the bStop input.

- Static signals are provided to the bSwitchOverUp or bSwitchOverDown inputs (e.g. by buttons). These set the bBlindUp and bBlindDown outputs. If this signal is asserted for longer than tSwitchOverTime, the outputs are latched. This means that the outputs will continue to be asserted, even if the signals at the inputs are removed again. In most cases, a value of 500 ms is sufficient for the tSwitchOverTime parameter. However, the output only remains asserted for the time tTurnOffTime, or until a new command is given to the function block.

- This last variation can be useful if the user wants to alter the setting of the blind step by step. Each positive edge at the bStepUp or bStepDown inputs sets the corresponding output for the time tStepTime. A value of 200 ms has been found effective for tStepTime.

The tSwitchOverDeadTime can be used to prevent damage to the drive motor caused by immediate changes in direction. In most cases, this value is between 0.5 s and 1.0 s. The drive manufacturer can give you a precise value.
Safety position

Travel to the safety position (e.g. because there is a strong wind or because maintenance is being carried out at the window) can be achieved by setting the bSafetyPosition input. The output bBlindUp is set and the output bBlindDown reset for the period specified by tTurnOffTime. Operation of the blinds is prevented for as long as the bSafetyPosition input is active.

Shading position

Under conditions of above-average sunshine, the blinds can be moved to the shading position. After presenting a positive edge to the bShadowPosition input, the blinds are lowered for the period of time specified by tShadowTurnOffTime. The blind is then moved upwards again for the period of time specified by tShadowTurnAroundTime. A time of about 2 seconds is usually set for this. This prevents the room from being completely darkened. During a change of direction, a pause of duration tSwitchOverDeadTime is maintained. Travel to the shading position can be interrupted at any time by a new command.

VAR_INPUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bUp</td>
<td>BOOL</td>
</tr>
<tr>
<td>bDown</td>
<td>BOOL</td>
</tr>
<tr>
<td>bStop</td>
<td>BOOL</td>
</tr>
<tr>
<td>bSwitchOverUp</td>
<td>BOOL</td>
</tr>
<tr>
<td>bSwitchOverDown</td>
<td>BOOL</td>
</tr>
<tr>
<td>tSwitchOverTime</td>
<td>TIME := t#500ms</td>
</tr>
<tr>
<td>bStepUp</td>
<td>BOOL</td>
</tr>
<tr>
<td>bStepDown</td>
<td>BOOL</td>
</tr>
<tr>
<td>tStepTime</td>
<td>TIME := t#200ms</td>
</tr>
<tr>
<td>bShadowPosition</td>
<td>BOOL</td>
</tr>
<tr>
<td>tShadowTurnAroundTime</td>
<td>TIME := t#0s</td>
</tr>
<tr>
<td>tShadowTurnOffTime</td>
<td>TIME := t#20s</td>
</tr>
<tr>
<td>bSafetyPosition</td>
<td>BOOL</td>
</tr>
<tr>
<td>tTurnOffTime</td>
<td>TIME := t#60s</td>
</tr>
<tr>
<td>tSwitchOverDeadTime</td>
<td>TIME := t#400ms</td>
</tr>
</tbody>
</table>

bUp: Set the bBlindUp output and reset the bBlindDown output. The bBlindUp output remains latched.

bDown: Set the bBlindDown output and reset the bBlindUp output. The bBlindDown output remains latched.

bStop: Reset the bBlindUp and bBlindDown outputs.

bSwitchOverUp: Set the bBlindUp output and reset the bBlindDown output. If the signal remains present for longer than tSwitchOverTime, the output bBlindUp remains latched.

bSwitchOverDown: Set the bBlindDown output and reset the bBlindUp output. If the signal remains present for longer than tSwitchOverTime, the output bBlindDown remains latched.

tSwitchOverTime: Gives the time for which the bSwitchUp and bSwitchDown inputs must remain asserted before the outputs are latched. If the value is 0, the outputs are latched immediately.

bStepUp: Reset the bBlindDown output and set the bBlindUp output for the time tStepTime.

bStepDown: Reset the bBlindUp output and set the bBlindDown output for the time tStepTime.

tStepTime: If the blind is controlled through the bStepUp or bStepDown inputs, the outputs remain asserted for this period of time. The outputs are not set if the specified duration is 0.

bShadowPosition: The shading position is approached.

tShadowTurnAroundTime: The blind travels in the opposite direction for the period of time specified by tShadowTurnAroundTime after the shading position has been reached.

tShadowTurnOffTime: The time for which the bBlindDown output is set in order to reach the shading position. A time of greater than 0 is necessary for the shading position to be approached.

bSafetyPosition: The safety position is approached. To do this, the blind is raised for the period of time specified by tTurnOffTime. It is not possible to operate the blinds while this input is set.

tTurnOffTime: If no input is activated, then the outputs are reset after this period of time. The outputs are not automatically reset if the specified duration is 0. The value given here should be about 10% larger than the travel time that is actually measured.
tSwitchOverDeadTime: Dwell time at a change of direction. Both outputs are reset during this period.

VAR OUTPUT

| bBlindUp        : BOOL; |
| bBlindDown      : BOOL; |

bBlindUp: The blind opens.
bBlindDown: The blind closes.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

4.1.3.3 FB_VenetianBlindEx

Four different methods are available for controlling the blind:

- A positive edge at the bUp or bDown inputs sets the bBlindUp or bBlindDown outputs respectively. These remain asserted until the time tDriveTime + 10% has elapsed, or until the function block receives some other command. Both outputs are immediately reset by a positive edge at the bStop input.

- Static signals are provided to the bSwitchOverUp or bSwitchOverDown inputs (e.g. by buttons). These set the bBlindUp and bBlindDown outputs. If this signal is asserted for longer than tSwitchOverTime, the outputs are latched. This means that the outputs will continue to be asserted, even if the signals at the inputs are removed again. In most cases, a value of 500 ms is sufficient for the tSwitchOverTime parameter. However, the output only remains asserted for the time tDriveTime + 10%, or until a new command is given to the function block.

- In certain applications it may be useful for the operator to be able to alter the blind position step by step. Each positive edge at the bStepUp or bStepDown inputs sets the corresponding output for the time tStepTime. A value of 200 ms has been found effective for tStepTime.

- Unlike the FB_VenetianBlind() [20] function block, this function block also enables movement to an absolute position. A percentage value is applied to input nSetPosition, and subsequently a positive edge is applied to input bPosition.
The `tSwitchOverDeadTime` can be used to prevent damage to the drive motor caused by immediate changes in direction. In most cases, this value is between 0.5 s and 1.0 s. The drive manufacturer can give you a precise value.

**Safety position**

Travel to the safety position (e.g. because there is a strong wind or because maintenance is being carried out at the window) can be achieved by setting the `bSafetyPosition` input. The output `bBlindUp` is set and the output `bBlindDown` reset for the period specified by `tDriveTime + 10%`. Operation of the blinds is prevented for as long as the `bSafetyPosition` input is active.

**Shading position**

Under conditions of above-average sunshine, the blinds can be moved to the shading position. After applying a positive edge to the input `bShadowPosition`, the blind is moved to the position `nShadowSetPosition`. The blind is then moved upwards again for the period of time specified by `tShadowTurnAroundTime`. This prevents the room from being completely darkened. If the blind had moved upwards during the approach of the shading position, it is moved downwards for the time period `tDriveSwitchOverTime - tShadowTurnAroundTime`. The same angle will therefore be set as if the blind had been moved downwards for darkening purposes.

During a change of direction, a pause of duration `tSwitchOverDeadTime` is maintained. Travel to the shading position can be interrupted at any time by a new command.

The set shading time `tShadowTurnAroundTime` must never be longer than the time for the change of direction `tDriveSwitchOverTime`.

### Moving to an absolute position

**Comments**

In most cases a blind will not provide feedback about its current position. Therefore, this can only be calculated via the travel time. The precision depends on the uniformity of the blind speed. Furthermore, the speed differences between opening and closing should be as small as possible.

The positions are always specified in percent. 0% corresponds to fully up, 100% to fully down. If a value greater than 100 is specified, it will be limited to 100 within the function block.

**Determining the parameters**

First of all, certain blind parameters have to be determined. One parameter is the travel time. This is the time it takes for the blind to open or close fully. The second parameter is the time required for a change of direction. During a change of direction, the angle between the individual blades will change. The travel time is transferred to the parameter `tDriveTime`. The duration for a direction change is transferred to `tDriveSwitchOverTime`.

**Referencing a function block**

Since the current position of the blind has to be calculated, inaccuracies during operation will accumulate. In order to limit deviations, the function block will automatically reference itself as often as possible. This happens when the blind is moved fully up or down and the corresponding output is reset automatically. The corresponding time is `tDriveTime + 10%`.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bUp</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bDown</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bStop</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bSwitchOverUp</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bSwitchOverDown</td>
<td>BOOL;</td>
</tr>
<tr>
<td>tSwitchOverTime</td>
<td>TIME := t#500ms;</td>
</tr>
<tr>
<td>bStepUp</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bStepDown</td>
<td>BOOL;</td>
</tr>
<tr>
<td>tStepTime</td>
<td>TIME := t#200ms;</td>
</tr>
<tr>
<td>bPosition</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>
bUp: Set the bBlindUp output and reset the bBlindDown output. The bBlindUp output remains latched.

bDown: Set the bBlindDown output and reset the bBlindUp output. The bBlindDown output remains latched.

bStop: Reset the bBlindUp and bBlindDown outputs.

bSwitchOverUp: Set the bBlindUp output and reset the bBlindDown output. If the signal remains present for longer than tSwitchOverTime, the output bBlindUp remains latched.

bSwitchOverDown: Set the bBlindDown output and reset the bBlindUp output. If the signal remains present for longer than tSwitchOverTime, the output bBlindDown remains latched.

tSwitchOverTime: Gives the time for which the bSwitchUp and bSwitchDown inputs must remain asserted before the outputs are latched. If the value is 0, the outputs are latched immediately.

bStepUp: Reset the bBlindDown output and set the bBlindUp output for the time tStepTime.

bStepDown: Reset the bBlindUp output and set the bBlindDown output for the time tStepTime.

tStepTime: If the blind is controlled through the bStepUp or bStepDown inputs, the outputs remain asserted for this period of time. The outputs are not set if the specified duration is 0.

bPosition: Move blind to specified position.

nSetPosition: Position (0%-100%) to which the blind is to be moved, after a positive edge has been applied to input bPosition. 0% corresponds to fully up, 100% corresponds to fully down.

bShadowPosition: The shading position is approached.

nShadowSetPosition: Shading position (0%-100%) to which the blind is to be moved, after a positive edge has been applied to input bShadowPosition.

tShadowTurnAroundTime: Once the shading position has been reached, the blind is moved upwards for the period tShadowTurnAroundTime.

bSafetyPosition: The safety position is approached. To do this, the blind is raised for the period tDriveTime + 10%. It is not possible to operate the blinds while this input is set.

tDriveTime: Travel time of the blind from fully up to fully down. If no input is activated, the outputs are reset after the period tDriveTime + 10%. The outputs are not automatically reset if the specified duration is 0. In this case, the blind cannot be moved to absolute positions.

tDriveSwitchOverTime: Period of time required for a change of direction of the blind.

tSwitchOverDeadTime: Dwell time at a change of direction. Both outputs are reset during this period.

VAR OUTPUT

bBlindUp : BOOL;

bBlindDown : BOOL;
nActualPosition : USINT;
bCalibrated : BOOL;

bBlindUp: The blind opens.

bBlindDown: The blind closes.

nActualPosition: Current position in percent.

bCalibrated: Indicates whether the blind is calibrated.
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

**4.1.3.4 FB_VenetianBlindEx1Switch**

Blind function block with the same functionality as FB_VenetianBlindEx() [22] but with an input bSwitch for jogging up and down.

Four different methods are available for controlling the blind:

- A positive edge at the bUp or bDown inputs sets the bBlindUp or bBlindDown outputs respectively. These remain asserted until the time tDriveTime + 10% has elapsed, or until the function block receives some other command. Both outputs are immediately reset by a positive edge at the bStop input.

- The input bSwitch, which is usually linked to a push button, can be used to switch the blind movement between up or down. In contrast to FB_VenetianBlindEx() [22], the outputs immediately switch to latching. Pressing the button switches between moving, stopping, moving in the opposite direction, stopping and moving again. The outputs remain active for a maximum period of tDriveTime + 10% or until a new command is issued for the function block. Switching is locked for the time tSwitchOverDeadTime.

- In certain applications it may be useful for the operator to be able to alter the blind position step by step. Each positive edge at the bStepUp or bStepDown inputs sets the corresponding output for the time tStepTime. A value of 200 ms has been found effective for tStepTime.

- Unlike the FB_VenetianBlind() [20] function block, this function block also enables movement to an absolute position. A percentage value is applied to input nSetPosition, and subsequently a positive edge is applied to input bPosition.

The tSwitchOverDeadTime can be used to prevent damage to the drive motor caused by immediate changes in direction. In most cases, this value is between 0.5 s and 1.0 s. The drive manufacturer can give you a precise value.

**Safety position**

Travel to the safety position (e.g. because there is a strong wind or because maintenance is being carried out at the window) can be achieved by setting the bSafetyPosition input. The output bBlindUp is set and the output bBlindDown reset for the period specified by tDriveTime + 10%. Operation of the blinds is prevented for as long as the bSafetyPosition input is active.
Shading position

Under conditions of above-average sunshine, the blinds can be moved to the shading position. After applying a positive edge to the input `bShadowPosition`, the blind is moved to the position `nShadowSetPosition`. The blind is then moved upwards again for the period of time specified by `tShadowTurnAroundTime`. This prevents the room from being completely darkened. If the blind had moved upwards during the approach of the shading position, it is moved downwards for the time period `tDriveSwitchOverTime - tShadowTurnAroundTime`. The same angle will therefore be set as if the blind had been moved downwards for darkening purposes.

During a change of direction, a pause of duration `tSwitchOverDeadTime` is maintained. Travel to the shading position can be interrupted at any time by a new command.

Moving to an absolute position

Comments

In most cases a blind will not provide feedback about its current position. Therefore, this can only be calculated via the travel time. The precision depends on the uniformity of the blind speed. Furthermore, the speed differences between opening and closing should be as small as possible.

The positions are always specified in percent. 0% corresponds to fully up, 100% to fully down. If a value greater than 100 is specified, it will be limited to 100 within the function block.

Determining the parameters

First of all, certain blind parameters have to be determined. One parameter is the travel time. This is the time it takes for the blind to open or close fully. The second parameter is the time required for a change of direction. During a change of direction, the angle between the individual blades will change. The travel time is transferred to the parameter `tDriveTime`. The duration for a direction change is transferred to `tDriveSwitchOverTime`.

Referencing a function block

Since the current position of the blind has to be calculated, inaccuracies during operation will accumulate. In order to limit deviations, the function block will automatically reference itself as often as possible. This happens when the blind is moved fully up or down and the corresponding output is reset automatically. The corresponding time is `tDriveTime + 10%`.

VAR_INPUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bUp</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bDown</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bStop</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bSwitch</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bStepUp</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bStepDown</td>
<td>BOOL;</td>
</tr>
<tr>
<td>tStepTime</td>
<td>TIME := t#200ms;</td>
</tr>
<tr>
<td>bPosition</td>
<td>BOOL;</td>
</tr>
<tr>
<td>nSetPosition</td>
<td>USINT;</td>
</tr>
<tr>
<td>bShadowPosition</td>
<td>BOOL;</td>
</tr>
<tr>
<td>nShadowSetPosition</td>
<td>USINT := 80;</td>
</tr>
<tr>
<td>tShadowTurnAroundTime</td>
<td>TIME := t#0s;</td>
</tr>
<tr>
<td>bSafetyPosition</td>
<td>BOOL;</td>
</tr>
<tr>
<td>tDriveTime</td>
<td>TIME := t#60s;</td>
</tr>
<tr>
<td>tDriveSwitchOverTime</td>
<td>TIME := t#200ms;</td>
</tr>
<tr>
<td>tSwitchOverDeadTime</td>
<td>TIME := t#400ms;</td>
</tr>
</tbody>
</table>

bUp: Set the `bBlindUp` output and reset the `bBlindDown` output. The `bBlindUp` output remains latched.

bDown: Set the `bBlindDown` output and reset the `bBlindUp` output. The `bBlindDown` output remains latched.

bStop: Reset the `bBlindUp` and `bBlindDown` outputs.

bSwitch: Switching input for jogging the blind, see description above. Switches between moving, stopping, moving in the opposite direction, stopping and moving again. In each case the activated output immediately switches to latching.
bStepUp: Reset the bBlindDown output and set the bBlindUp output for the time tStepTime.

bStepDown: Reset the bBlindUp output and set the bBlindDown output for the time tStepTime.

tStepTime: If the blind is controlled through the bStepUp or bStepDown inputs, the outputs remain asserted for this period of time. The outputs are not set if the specified duration is 0.

bPosition: Move blind to specified position.

nSetPosition: Position (0%-100%) to which the blind is to be moved after a positive edge has been applied to input bPosition. 0% corresponds to fully up, 100% corresponds to fully down.

bShadowPosition: The shading position is approached.

nShadowSetPosition: Shading position (0%-100%) to which the blind is to be moved, after a positive edge has been applied to input bShadowPosition.

tShadowTurnAroundTime: Once the shading position has been reached, the blind is moved upwards for the period tShadowTurnAroundTime.

bSafetyPosition: The safety position is approached. To do this, the blind is moved upwards for the period tDriveTime + 10%. It is not possible to operate the blinds while this input is set.

tDriveTime: Travel time of the blind from fully up to fully down. If no input is activated, the outputs are reset after the period tDriveTime + 10%. The outputs are not automatically reset if the specified duration is 0. In this case, the blind cannot be moved to absolute positions.

tDriveSwitchOverTime: Period of time required for a change of direction of the blind.

tSwitchOverDeadTime: Dwell time at a change of direction. Both outputs are reset during this period.

VAR_OUTPUT

<table>
<thead>
<tr>
<th>bBlindUp</th>
<th>BOOL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBlindDown</td>
<td>BOOL;</td>
</tr>
<tr>
<td>nActualPosition</td>
<td>USINT;</td>
</tr>
<tr>
<td>bCalibrated</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>

bBlindUp: The blind opens.

bBlindDown: The blind closes.

nActualPosition: Current position in percent.

bCalibrated: Indicates whether the blind is calibrated.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

4.1.4 Filter functions

4.1.4.1 FB_PT1
PT, element for smoothing of input variables.

This function block is active continuously. The output \( fOut \) always follows the input value \( fIn \) multiplied by \( Kp \) with an exponential curve:

\[
f(t) = K_p \left( 1 - e^{-\frac{t}{T_1}} \right)
\]

The following time-discrete formula is used for the calculation in the PLC:

\[
y_{n+1} = \left( K_p \cdot x \cdot \frac{t_{cycle}}{T_1} \right) + \left( 1 - \frac{t_{cycle}}{T_1} \right) \cdot y_n
\]

\[
T_1 = 0: \quad y_{n+1} = K_p \cdot x
\]

With a continuously changing input \( fIn \), \( fOut \) behaves as follows \( fIn = 0..33000, Kp = 1, T1 = 5s \):
Note on damping times: Since this function block is a time-discrete model of a PT1 element, it only works correctly if the damping time is significantly longer than the set cycle time. To be on the safe side, if a damping time is entered that is less than twice the set cycle time it is internally set to zero. A damping time of 0s means that the output variable directly follows the input variable multiplied by Kp.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fIn</td>
<td>LREAL</td>
<td>Input value.</td>
</tr>
<tr>
<td>fKp</td>
<td>LREAL</td>
<td>Gain factor, preset value: 1.</td>
</tr>
<tr>
<td>tT1</td>
<td>TIME</td>
<td>Damping time, preset value: 10 s.</td>
</tr>
<tr>
<td>tCycleTime</td>
<td>TIME</td>
<td>PLC cycle time, preset value: 10 ms.</td>
</tr>
<tr>
<td>bSetActual</td>
<td>BOOL</td>
<td>Sets the output fOut directly to the input value fIn.</td>
</tr>
</tbody>
</table>

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fOut</td>
<td>LREAL</td>
<td>Output value.</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>
4.1.4.2 FB_PT2

PT₂ element for smoothing of input variables.

This function block is active continuously. The output fOut always follows the input value fln multiplied by Kp.

This PT₂ element consists of a series of two PT₁ elements; the time constants T₁ and T₂ can have different values. The step response (see above) shows a significantly more attenuated subsequent behavior compared to the PT₁ element (dashed) right from the start.

With a continuously changing input fln, fOut behaves as follows (fln= 0..33000, Kp= 1, T₁, T₂= 5s):
In comparison, the dotted line shows the behavior of a PT1 element \[\text{PT1} \] with \( fIn = 0..33000, \ Kp = 1, \ T1 = 5\text{s} \).

Note on damping times: Since this function block is a time-discrete model of a PT2 element, it only works correctly if the damping time is significantly longer than the set cycle time. To be on the safe side, if damping times are entered that are less than twice the set cycle time they are internally set to zero. As already mentioned, the PT2 element consists of two PT1 elements connected in series. If one of the two damping times is set to zero, the PT2 element is reduced to a PT1 element. If both damping times are set to zero, the output variable directly follows the input variable multiplied by Kp.

**VAR_INPUT**

- \( fIn \): LREAL;
- \( fKp \): LREAL := 1;
- \( tT1 \): TIME := t\#10s;
- \( tT2 \): TIME := t\#10s;
- \( tCycleTime \): TIME := t\#10ms;
- \( bSetActual \): BOOL;

- \( fIn \): Input value.
- \( fKp \): Gain factor, preset value: 1.
- \( tT1 \): Damping time 1, preset value: 10 s.
- \( tT2 \): Damping time 2, preset value: 10 s.
- \( tCycleTime \): PLC cycle time, preset value: 10 ms.
- \( bSetActual \): Sets the output \( fOut \) directly to the input value \( fIn \).

**VAR_OUTPUT**

- \( fOut \): LREAL;

- \( fOut \): Output value.
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

Also see about this

FB_PT1 \[27\]

### 4.1.5 Lighting

#### 4.1.5.1 FB_ConstantLightControlEco

The function block is used for constant light control.

The system tries to match a specified setpoint through cyclic dimming. The control dynamics are determined by a dead time \((t_{\text{DeadTime}})\) and the step size \((n_{\text{StepSize}})\). The dead time specifies the waiting time between the individual steps or increments of the control value, which are determined by the set step size. The smaller the dead time, the faster the control. A freely definable hysteresis \((n_{\text{Hysteresis}})\) prevents continuous oscillation around the setpoint. If the actual value is within the hysteresis range around the setpoint, the lamps brightness remains unchanged.

**NOTE**

If the set step size \(n_{\text{StepSize}}\) is too large or the hysteresis \(n_{\text{Hysteresis}}\) is too small, the hysteresis range may be "missed". This cannot be prevented by the function block, because the light output \(n_{\text{LightLevel}}\) is only physically linked to the recorded actual light value, \(n_{\text{ActualLevel}}\).

**VAR_INPUT**

```plaintext
bEnable : BOOL;
bOn : BOOL;
bOff : BOOL;
bToggle : BOOL;
nSetpointValue : UINT := 16000;
nActualValue : UINT;
nHysteresis : UINT := 100;
nMaxLevel : UINT := 32767;
nMinLevel : UINT := 3276;
nStepSize : UINT := 10;
tDeadTime : TIME := t\#50ms;
nOptions : DWORD;
```

\(b_{\text{Enable}}\): Enables the function block. If this input is FALSE, the inputs \(b_{\text{On}}, b_{\text{Off}}\) and \(b_{\text{Toggle}}\) are disabled. The control value remains unchanged.
bOn: Switches the controlled devices to \( n_{\text{MaxLevel}} \) and activates constant light control.

bOff: Switches the addressed devices off and disables constant light control.

bToggle: The lighting is switched on or off, depending on the state of the reference device.

nSetpointValue: This input is used for specifying the setpoint.

nActualValue: The actual value is applied at this input.

nHysteresis: Control hysteresis around the setpoint. If the actual value is within this range, the control values for the lamps remain unchanged.

nMaxLevel: Maximum value of the control value at \( n_{\text{LightLevel}} \).

nMinLevel: Minimum value of the control value at \( n_{\text{LightLevel}} \). If this value is to be undercut by a dimming process, \( n_{\text{LightLevel}} \) is set directly to "0". On the other hand, if \( n_{\text{LightLevel}} \) is set to "0" when control is active, the dimming process starts directly at this value.

nStepSize: Step size with which the control value \( n_{\text{LightLevel}} \) is changed.

tDeadTime: Dead time between the individual steps (up or down) of the control value.

nOptions: Currently not used. Reserved for future extensions.

### VAR_OUTPUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>nDeviation</td>
<td>INT;</td>
</tr>
<tr>
<td>bControllerIsActive</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bBusy</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL;</td>
</tr>
<tr>
<td>nErrorId</td>
<td>UDINT;</td>
</tr>
</tbody>
</table>

nDeviation: Current control deviation (setpoint/actual value).

bControllerIsActive: This output is set once the control is activated.

bBusy: When the function block is activated, this output is always active when changes are made to the control value.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in nErrorId. Is reset to FALSE by the execution of a command at the inputs.

nErrorId: Contains the command-specific error code of the most recently executed command. Is reset to "0" by the execution of a command at the inputs.

See Error codes [1, 92].

### VAR_IN_OUT

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

nLightLevel: Output control value of the function block and reference to the light output value. This value is defined as In-Out variable because the function block reads and writes the light value.

**Flow diagram**

The following diagram illustrates the regular control behavior:
The control is enabled by a TRUE signal at input $bEnable$. When a positive edge is encountered at $bOn$, $nLightLevel$ is set to the maximum value. This also influences the measured light value $nActualValue$, which increases, so that dimming becomes necessary. $nLightLevel$ is now reduced step-by-step until the measured light value $nActualValue$ is in the hysteresis range around the setpoint ($nSetpointValue - 0.5*nHysteresis < x < nSetpointValue + 0.5*nHysteresis$).

If, for example, the measured light value then falls, e.g. due to cloudiness, the control system counteracts this by gradually increasing the lighting level until the light value is back in the hysteresis band.

If the step size $nStepSize$ is too large or the hysteresis is too small, an oscillation around the setpoint can occur. Because the hysteresis is small compared to the step changes, the hysteresis range is constantly missed:
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>
4.1.5.2 FB_Dimmer1Switch

The function block is used to dim lights with a switch.

**Operating by means of the bSwitchDimm input**

The light is switched on or off by a short signal at the `bSwitchDimm` input. Dimmer mode will be activated if the signal remains for longer than `tSwitchOverTime` (typical recommended value: 200 ms). The output signal then cycles between `nOutMin` and `nOutMax`. In order to be able to set the maximum or minimum value more easily, the output signal pauses at the level of the maximum and minimum values for the time given by `tCycleDelay`. When the signal is once more removed, the output signal being generated at that time is retained. Another pulse at the input will set the output to 0.

**Operation by means of the bOn and bOff inputs**

The light is immediately switched on or off if a positive edge is applied to the `bOn` or `bOff` inputs. For example, for global on/off functions. The output value is set to 0 when switching off. The switch-on behavior can be affected by the memory function (see below).

**Operation by means of the bSetDimmValue and nDimmValue inputs**

If the value of `nDimmValue` changes, the signal will be passed through directly to the output. The significant point here is that the value changes. The lighting is switched off by changing the value to 0. If there is a positive edge at the `bSetDimmValue` input, the value of `nDimmValue` immediately appears at the output. Immediate modification of the output can be suppressed by a static 1- signal at the `bSetDimmValue` input. This makes it possible to apply a value to the `nDimmValue` input, but for this value only to be passed to the output at the next positive edge of `bSetDimmValue`.

The `bSetDimmValue` and `nDimmValue` inputs can be used to implement a variety of lighting scenarios. Direct setting of the output, by means of `nDimmValue`, can be used to achieve particular brightness levels. Either directly or by continuously changing the value. `nDimmValue` must have a value between `nOutMin` and `nOutMax`. The value 0 is an exception. If the value is outside this range, the output value is limited to the upper or lower limit, as appropriate.

**The memory function**

It is necessary to determine whether the memory function (`bMemoryModeOn` input) is active or not at switch-on. If the memory function is active, then the last set value is placed at the output as soon as the lamp is switched on. If the memory function is not active, then the value specified by the `nOnValueWithoutMemoryMode` parameter is output. It is irrelevant, in this case, whether the light it has been
switched on by means of the \texttt{bOn} input or the \texttt{bSwitchDimm} input. It should be noted that the \texttt{nOnValueWithoutMemoryMode} parameter must lie between \texttt{nOutMin} and \texttt{nOutMax}. If this is not the case, the output value is adjusted to the upper or lower limit, as appropriate.

**Fast dimming up/down when switching on and off**

Lighting is particularly pleasant if sudden changes are replaced by a slow change to the desired value. This mode can be activated both for switching on and for switching off by means of the two inputs, \texttt{bDimmOnMode} and \texttt{bDimmOffMode}. The \texttt{tDimmOnTime} and \texttt{tDimmOffTime} parameters specify the time that will be taken by the switching processes. This value is always related to the minimum and maximum possible output values (\texttt{nOutMin} and \texttt{nOutMax}). The \texttt{bOn} and \texttt{bOff} inputs are one way in which the switch on/off commands may be given. Alternatively, a short pulse can be provided to the \texttt{bSwitchDimm} input. If the \texttt{nDimmValue} input is set to 0, the output is modified without delay. The same is true if the output is set by a positive edge at the \texttt{bSetDimmValue} input.

**Comments on the \texttt{tSwitchOverTime} and \texttt{tDimmTime} parameters**

If a duration of 0 is specified for the \texttt{tSwitchOverTime} parameter, while a value of greater than 0 is specified for \texttt{tDimmTime}, then the \texttt{bSwitchDimm} input can only be used to dim the light. Switching on and off is only possible with the \texttt{bOn} and \texttt{bOff} inputs.

If the \texttt{tDimmTime} parameter is 0, the \texttt{bSwitchDimm} input can only be used to switch the light on or off. In this case, the value of \texttt{tSwitchOverTime} is irrelevant.

**VAR_INPUT**

\begin{verbatim}
VAR_INPUT
bSwitchDimm : BOOL;
bOn : BOOL;
bOff : BOOL;
bSetDimmValue : BOOL;
nDimmValue : UINT;
tSwitchOverTime : TIME := t#500ms;
tDimmTime : TIME := t#5s;
tCycleDelay : TIME := t#10ms;
bMemoryModeOn : BOOL := FALSE;
nOnValueWithoutMemoryMode : UINT := 20000;
bDimmOnMode : BOOL := FALSE;
tDimmOnTime : TIME := t#0s;
bDimmOffNode : BOOL := FALSE;
tDimmOffTime : TIME := t#0s;
nOutMin : UINT := 5000;
nOutMax : UINT := 32767;
\end{verbatim}

- \texttt{bSwitchDimm}: Switches or dims the output.
- \texttt{bOn}: Switches the output to the last output value, or to the value specified by \texttt{nOnValueWithoutMemoryMode}.
- \texttt{bOff}: Switches the output to 0.
- \texttt{bSetDimmValue}: Switches the output to the value \texttt{nDimmValue}.
- \texttt{nDimmValue}: The value is immediately applied to the output when there is a change.
- \texttt{tSwitchOverTime}: Time for switching between the light on/off and dimming functions for the \texttt{bSwitchDimm} input.
- \texttt{tDimmTime}: Time required for dimming to go from its minimum value to its maximum value.
- \texttt{tCycleDelay}: Delay time, if either the minimum or maximum value is reached.
- \texttt{bMemoryModeOn}: Switches over to use the memory function, so that the previous value is written to the output as soon as it is switched on.
- \texttt{nOnValueWithoutMemoryMode}: Switch-on value if the memory function is not active.
- \texttt{bDimmOnMode}: The output value is increased in steps when switching on.
- \texttt{tDimmOnTime}: Time period during which the light level is increased when the light is switched on. \texttt{bDimmOnMode} must be active.
bDimmOffMode: The output value is reduced in steps when switching off

tDimmOffTime: Time period during which the light is reduced when the light is switched off. bDimmOffMode must be active.

nOutMin: Minimum output value.

nOutMax: Maximum output value. If the parameter nOutMin is not smaller than nOutMax, the output will remain at 0.

VAR_OUTPUT

<table>
<thead>
<tr>
<th>nOut</th>
<th>UINT;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bLight</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>

nOut: analog output value.

bLight: digital output value. This is set if nOut is greater than 0.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.4020.32</td>
<td>Tc2_BABasic v3.1.0.0</td>
</tr>
</tbody>
</table>

4.1.5.3 FB_Dimmer1SwitchEco

This function block is a memory-saving variant of FB_Dimmer1Switch(). It does not have the special functions "Set brightness value" and "Switch off memory function", which may not be required for many applications. Moreover, the values nOutMin and nOutMax of the FB_Dimmer1Switch() are set internally here to 0 and 32767 respectively. This output span corresponds to the display range of an analog output terminal. The tPLCCycle input is important. This time is used to calculate internally the amount by which the nOut output must be increased per cycle - that saves additional time calculations.

Operating by means of the bSwitchDimm input

The light is switched on or off by a short signal at the bSwitchDimm input. Dimmer mode will be activated if the signal remains for longer than tSwitchOverTime (typical recommended value: 200ms). The output signal moves cyclically between 0 and 32767. In order to be able to set the maximum or minimum value more easily, the output signal pauses at the level of the maximum and minimum values for the time given by tCycleDelay. When the signal is once more removed, the output signal being generated at that time is retained. Another pulse at the input will set the output to 0.

Operation by means of the bOn and bOff inputs

The light is immediately switched on or off if a positive edge is applied to the bOn or bOff inputs. For example, for global on/off functions. The output value is set to 0 when switching off.
The memory function

In contrast to FB_Dimmer1Switch() [36], where the memory function can be activated or deactivated via the input bMemoryModeOn, the memory function is always active in this memory-saving version. This means that the last-set value is adopted as the brightness value when switching on. It is irrelevant, in this case, whether the light it has been switched on by means of the bOn input or the bSwitchDimm input.

Comment on the tSwitchOverTime parameter

If a duration of 0 is specified for the parameter tSwitchOverTime, the bSwitchDimm input can only be used to dim the light. Switching on and off is only possible with the bOn and bOff inputs.

VAR_INPUT

<table>
<thead>
<tr>
<th>bSwitchDimm</th>
<th>BOOL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bOn</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bOff</td>
<td>BOOL;</td>
</tr>
<tr>
<td>tSwitchOverTime</td>
<td>TIME := t#500ms;</td>
</tr>
<tr>
<td>tDimmTime</td>
<td>TIME := t#5s;</td>
</tr>
<tr>
<td>tCycleDelay</td>
<td>TIME := t#500ms;</td>
</tr>
<tr>
<td>tPLCCycle</td>
<td>TIME := t#10ms;</td>
</tr>
</tbody>
</table>

bSwitchDimm: Switches or dims the output.

bOn: Switches the output to the last output value, or to the value specified by nOnValueWithoutMemoryMode.

bOff: Switches the output to 0.

tSwitchOverTime: Time for switching between the light on/off and dimming functions for the bSwitchDimm input.

tDimmTime: Time required for dimming to go from its minimum value to its maximum value.

tCycleDelay: Delay time, if either the minimum or maximum value is reached.

tPLCCycle: the set PLC cycle time.

VAR_OUTPUT

<table>
<thead>
<tr>
<th>nOut</th>
<th>UINT;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bLight</td>
<td>BOOL;</td>
</tr>
</tbody>
</table>

nOut: analog output value.

bLight: digital output value. This is set if nOut is greater than 0.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>
4.1.5.4 FB_Dimmer2Switch

The function range of the function block corresponds to that of the function block FB_Dimmer1Switch() [36]. The difference is that in the function block FB_Dimmer2Switch two switches are connected. This allows the user to choose specifically between dimming up or dimming down.

**Operation by means of the bSwitchDimmUp and bSwitchDimmDown inputs**

The light is switched on or off by a short signal at the bSwitchDimmUp or bSwitchDimmDown inputs. Dimmer mode will be activated if the signal remains for longer than tSwitchOverTime (typical recommended value: 200ms). The output signal goes to nOutMin or nOutMax. When the signal is once more removed, the output signal being generated at that time is retained. Another pulse at one of the inputs will set the output to 0.

**Operation by means of the bOn and bOff inputs**

The light is immediately switched on or off if a positive edge is applied to the bOn or bOff inputs. For example, for global on/off functions. The output value is set to 0 when switching off. The switch-on behavior can be affected by the memory function (see below).

**Operation by means of the bSetDimmValue and nDimmValue inputs**

If the value of nDimmValue changes, the signal will be passed through directly to the output. The significant point here is that the value changes. The lighting is switched off by changing the value to 0. If there is a positive edge at the bSetDimmValue input, the value of nDimmValue immediately appears at the output. Immediate modification of the output can be suppressed by a static 1- signal at the bSetDimmValue input. This makes it possible to apply a value to the nDimmValue input, but for this value only to be passed to the output at the next positive edge of bSetDimmValue.

The bSetDimmValue and nDimmValue inputs can be used to implement a variety of lighting scenarios. Direct setting of the output, by means of nDimmValue, can be used to achieve particular brightness levels. Either directly or by continuously changing the value. nDimmValue must have a value between nOutMin and nOutMax. The value 0 is an exception. If the value is outside this range, the output value is limited to the upper or lower limit, as appropriate.

**The memory function**

It is necessary to determine whether the memory function (bMemoryModeOn input) is active or not at switch-on. If the memory function is active, then the last set value is placed at the output as soon as the lamp is switched on. If the memory function is not active, then the value specified by the nOnValueWithoutMemoryMode parameter is output. It is irrelevant, in this case, whether the light it has been
switched on by means of the \textit{bOn} input or one of the \textit{bSwitchDimmUp} or \textit{bSwitchDimmDown} inputs. It should be noted that the \textit{nOnValueWithoutMemoryMode} parameter must lie between \textit{nOutMin} and \textit{nOutMax}. If this is not the case, the output value is adjusted to the upper or lower limit, as appropriate.

**Fast dimming up/down when switching on and off**

Lighting is particularly pleasant if sudden changes are replaced by a slow change to the desired value. This mode can be activated both for switching on and for switching off by means of the two inputs, \textit{bDimmOnMode} and \textit{bDimmOffMode}. The \textit{tDimmOnTime} and \textit{tDimmOffTime} parameters specify the time that will be taken by the switching processes. This value is always related to the minimum and maximum possible output values (\textit{nOutMin} and \textit{nOutMax}). The \textit{bOn} and \textit{bOff} inputs are one way in which the switch on/off commands may be given. Alternatively, a short pulse can be provided to either of the inputs \textit{bSwitchDimmUp} or \textit{bSwitchDimmDown}. If the \textit{nDimmValue} input is set to 0, the output is modified without delay. The same is true if the output is set by a positive edge at the \textit{bSetDimmValue} input.

**Comments on the \textit{tSwitchOverTime} and \textit{tDimmTime} parameters**

If a duration of 0 is specified for the \textit{tSwitchOverTime} parameter, while a value of greater than 0 is specified for \textit{tDimmTime}, then the \textit{bSwitchDimmUp} or \textit{bSwitchDimmDown} inputs can only be used to dim the light. Switching on and off is only possible with the \textit{bOn} and \textit{bOff} inputs.

If the \textit{tDimmTime} parameter is 0, the \textit{bSwitchDimmUp} or \textit{bSwitchDimmDown} inputs can only be used to switch the light on or off. In this case, the value of \textit{tSwitchOverTime} is irrelevant.

**VAR_INPUT**

\begin{verbatim}
VAR_INPUT
bSwitchDimmUp : BOOL;
bSwitchDimmDown : BOOL;
bOn : BOOL;
bOff : BOOL;
bSetDimmValue : BOOL;
nDimmValue : UINT;
tSwitchOverTime : TIME := t#500ms;
tDimmTime : TIME := t#5s;
bMemoryModeOn : BOOL := FALSE;
nOnValueWithoutMemoryMode : UINT := 20000;
bDimmOnMode : BOOL := FALSE;
tDimmOnTime : TIME := t#0s;
bDimmOffMode : BOOL := FALSE;
tDimmOffTime : TIME := t#0s;
nOutMin : UINT := 5000;
nOutMax : UINT := 32767;
\end{verbatim}

- \textbf{bSwitchDimmUp}: Switches or dims the output Up.
- \textbf{bSwitchDimmDown}: Switches or dims the output Down.
- \textbf{bOn}: Switches the output to the last output value, or to the value specified by \textit{nOnValueWithoutMemoryMode}.
- \textbf{bOff}: Switches the output to 0.
- \textbf{bSetDimmValue}: Switches the output to the value \textit{nDimmValue}.
- \textbf{nDimmValue}: The value is immediately applied to the output when there is a change.
- \textbf{tSwitchOverTime}: Time for switching between the light on/off and dimming functions for the \textit{bSwitchDimmUp} and \textit{bSwitchDimmDown} inputs.
- \textbf{tDimmTime}: Time required for dimming to go from its minimum value to its maximum value.
- \textbf{bMemoryModeOn}: Switches over to use the memory function, so that the previous value is written to the output as soon as it is switched on.
- \textbf{nOnValueWithoutMemoryMode}: Switch-on value if the memory function is not active.
- \textbf{bDimmOnMode}: The output value is increased in steps when switching on.
- \textbf{tDimmOnTime}: Time period during which the light level is increased when the light is switched on. \textit{bDimmOnMode} must be active.
bDimmOffMode: The output value is reduced in steps when switching off

tDimmOffTime: Time period during which the light is reduced when the light is switched off. bDimmOffMode must be active.

nOutMin: Minimum output value.

nOutMax: Maximum output value. If the parameter nOutMin is not smaller than nOutMax, the output will remain at 0.

VAR_OUTPUT

nOut : UINT;
bLight : BOOL;

nOut: analog output value.

bLight: digital output value. This is set if nOut is greater than 0.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

4.1.5.5 FB_Dimmer2SwitchEco

This function block is a memory-saving variant of FB_Dimmer2Switch(). It does not have the special functions "Set brightness value" and "Switch off memory function", which may not be required for many applications. Moreover, the values nOutMin and nOutMax of the FB_Dimmer2Switch() are set internally here to 0 and 32767 respectively. This output span corresponds to the display range of an analog output terminal. The tPLCCycle input is important. This time is used to calculate internally the amount by which the nOut output must be increased per cycle - that saves additional time calculations.

Operation by means of the bSwitchDimmUp and bSwitchDimmDown inputs

The light is switched on or off by a short signal at the bSwitchDimmUp or bSwitchDimmDown inputs. Dimmer mode will be activated if the signal remains for longer than tSwitchOverTime (typical recommended value: 200ms). The output signal goes to nOutMin or nOutMax. When the signal is once more removed, the output signal being generated at that time is retained. Another pulse at one of the inputs will set the output to 0.

Operation by means of the bOn and bOff inputs

The light is immediately switched on or off if a positive edge is applied to the bOn or bOff inputs. For example, for global on/off functions. The output value is set to 0 when switching off.
The memory function

In contrast to FB_Dimmer2Switch([40]), where the memory function can be activated or deactivated via the input bMemoryModeOn, the memory function is always active in this memory-saving version. This means that the last-set value is adopted as the brightness value when switching on. It is irrelevant, in this case, whether the light it has been switched on by means of the bOn input or one of the bSwitchDimmUp or bSwitchDimmDown inputs.

Comment on the tSwitchOverTime parameter

If a duration of 0 is specified for the parameter tSwitchOverTime, the bSwitchDimmUp and bSwitchDimmDown inputs can only be used to dim the light. Switching on and off is only possible with the bOn and bOff inputs.

VAR_INPUT

bSwitchDimmUp : BOOL;
bSwitchDimmDown : BOOL;
bOn : BOOL;
bOff : BOOL;
tSwitchOverTime : TIME := t#500ms;
tDimmTime : TIME := t#5s;
tPLCCycle : TIME := t#10ms;

bSwitchDimmUp: Switches or dims the output Up.
bSwitchDimmDown: Switches or dims the output Down.
bOn: Switches the output to the last output value.
bOff: Switches the output to 0.
tSwitchOverTime: Time for switching between the light on/off and dimming functions for the bSwitchDimmUp and bSwitchDimmDown inputs.
tDimmTime: Time required for dimming to go from its minimum value to its maximum value.
tCycleDelay: Delay time, if either the minimum or maximum value is reached.
tPLCCycle: the set PLC cycle time.

VAR_OUTPUT

nOut : UINT;
bLight : BOOL;
nOut: analog output value.
bLight: digital output value. This is set if nOut is greater than 0.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>
The functionality of the function block corresponds to the function blocks FB_Dimmer1Switch() \[36\] and FB_Dimmer2Switch() \[40\].

**Operating by means of the bSwitchDimm input**

The light is switched on or off by a short signal at the bSwitchDimm input. Dimmer mode will be activated if the signal remains for longer than tSwitchOverTime (typical recommended value: 200 ms). The output signal then cycles between nOutMin and nOutMax. In order to be able to set the maximum or minimum value more easily, the output signal pauses at the level of the maximum and minimum values for the time given by tCycleDelay. When the signal is once more removed, the output signal being generated at that time is retained. Another pulse at the input will set the output to 0.

**Operation by means of the bSwitchDimmUp and bSwitchDimmDown inputs**

The light is switched on or off by a short signal at the bSwitchDimmUp or bSwitchDimmDown inputs. Dimmer mode will be activated if the signal remains for longer than tSwitchOverTime (typical recommended value: 200ms). The output signal goes to nOutMin or nOutMax. When the signal is once more removed, the output signal being generated at that time is retained. Another pulse at one of the inputs will set the output to 0.

**Operation by means of the bOn and bOff inputs**

The light is immediately switched on or off if a positive edge is applied to the bOn or bOff inputs. For example, for global on/off functions. The output value is set to 0 when switching off. The switch-on behavior can be affected by the memory function (see below).

**Operation by means of the bSetDimmValue and nDimmValue inputs**

If the value of nDimmValue changes, the signal will be passed through directly to the output. The significant point here is that the value changes. The lighting is switched off by changing the value to 0. If there is a positive edge at the bSetDimmValue input, the value of nDimmValue immediately appears at the output. Immediate modification of the output can be suppressed by a static 1-signal at the bSetDimmValue input. This makes it possible to apply a value to the nDimmValue input, but for this value only to be passed to the output at the next positive edge of bSetDimmValue.

The bSetDimmValue and nDimmValue inputs can be used to implement a variety of lighting scenarios. Direct setting of the output, by means of nDimmValue, can be used to achieve particular brightness levels.
Either directly or by continuously changing the value. \( nDimmValue \) must have a value between \( nOutMin \) and \( nOutMax \). The value 0 is an exception. If the value is outside this range, the output value is limited to the upper or lower limit, as appropriate.

**The memory function**

It is necessary to determine whether the memory function (\( bMemoryModeOn \) input) is active or not at switch-on. If the memory function is active, then the last set value is placed at the output as soon as the lamp is switched on. If the memory function is not active, then the value specified by the \( nOnValueWithoutMemoryMode \) parameter is output. It is irrelevant, in this case, whether the light it has been switched on by means of the \( bOn \) input or one of the \( bSwitchDimmUp \) or \( bSwitchDimmDown \) inputs. It should be noted that the \( nOnValueWithoutMemoryMode \) parameter must lie between \( nOutMin \) and \( nOutMax \). If this is not the case, the output value is adjusted to the upper or lower limit, as appropriate.

**Fast dimming up/down when switching on and off**

Lighting is particularly pleasant if sudden changes are replaced by a slow change to the desired value. This mode can be activated both for switching on and for switching off by means of the two inputs, \( bDimmOnMode \) and \( bDimmOffMode \). The \( tDimmOnTime \) and \( tDimmOffTime \) parameters specify the time that will be taken by the switching processes. This value is always related to the minimum and maximum possible output values (\( nOutMin \) and \( nOutMax \)). The \( bOn \) and \( bOff \) inputs are one way in which the switch on/off commands may be given. Alternatively, a short pulse can be provided to either of the inputs \( bSwitchDimmUp \) or \( bSwitchDimmDown \). If the \( nDimmValue \) input is set to 0, the output is modified without delay. The same is true if the output is set by a positive edge at the \( bSetDimmValue \) input.

**Comments on the \( tSwitchOverTime \) and \( tDimmTime \) parameters**

If a duration of 0 is specified for the \( tSwitchOverTime \) parameter, while a value of greater than 0 is specified for \( tDimmTime \), then the \( bSwitchDimmUp \) or \( bSwitchDimmDown \) inputs can only be used to dim the light. Switching on and off is only possible with the \( bOn \) and \( bOff \) inputs.

If the \( tDimmTime \) parameter is 0, the \( bSwitchDimmUp \) or \( bSwitchDimmDown \) inputs can only be used to switch the light on or off. In this case, the value of \( tSwitchOverTime \) is irrelevant.

**VAR_INPUT**

```
bSwitchDimm            : BOOL;
bSwitchDimmUp         : BOOL;
bSwitchDimmDown       : BOOL;
bOn                    : BOOL;
bOff                   : BOOL;
bSetDimmValue         : BOOL;
nDimmValue            : UINT;
tSwitchOverTime       : TIME := t#500ms;
tDimmTime              : TIME := t#5s;
tCycleDelay           : TIME := t#10ms;
bMemoryModeOn         : BOOL := FALSE;
nOnValueWithoutMemoryMode : UINT := 20000;
bDimmOnMode           : BOOL := FALSE;
tDimmOnTime           : TIME := t#0s;
bDimmOffMode          : BOOL := FALSE;
tDimmOffTime          : TIME := t#0s;
nOutMin               : UINT := 5000;
nOutMax               : UINT := 32767;
```

- \( bSwitchDimm \): Switches or dims the output.
- \( bSwitchDimmUp \): Switches or dims the output Up.
- \( bSwitchDimmDown \): Switches or dims the output Down.
- \( bOn \): Switches the output to the last output value, or to the value specified by \( nOnValueWithoutMemoryMode \).
- \( bOff \): Switches the output to 0.
- \( bSetDimmValue \): Switches the output to the value \( nDimmValue \).
- \( nDimmValue \): The value is immediately applied to the output when there is a change.
**tSwitchOverTime**: Time for switching between the light on/off and dimming functions for the 
*bSwitchDimmUp* and *bSwitchDimmDown* inputs.

**tDimmTime**: Time required for dimming to go from its minimum value to its maximum value.

**tCycleDelay**: Delay time, if either the minimum or maximum value is reached.

**bMemoryModeOn**: Switches over to use the memory function, so that the previous value is written to the 
output as soon as it is switched on.

**nOnValueWithoutMemoryMode**: Switch-on value if the memory function is not active.

**bDimmOnMode**: The output value is increased in steps when switching on.

**tDimmOnTime**: Time period during which the light level is increased when the light is switched on. 
*bDimmOnMode* must be active.

**bDimmOffMode**: The output value is reduced in steps when switching off.

**tDimmOffTime**: Time period during which the light is reduced when the light is switched off. 
*bDimmOffMode* must be active.

**nOutMin**: Minimum output value.

**nOutMax**: Maximum output value. If the parameter *nOutMin* is not smaller than *nOutMax*, the output will 
remain at 0.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>nOut</th>
<th>UINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bLight</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

*nOut*: analog output value.

*bLight*: digital output value. This is set if *nOut* is greater than 0.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

4.1.5.7 **FB_Light**

A positive edge at the *bOn* input sets the *bLight* output. 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**

<table>
<thead>
<tr>
<th>bOn</th>
<th>BOOL</th>
</tr>
</thead>
<tbody>
<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 state of the output.
VAR_OUTPUT

bLight : BOOL;

bLight: The output is set with a positive edge at bOn.

Example 1:
In the following example a light is operated by two switches.

Example 2:
In the following example the bSwitchMasterOff switch is used to switch the bLampKitchen and bLampGarage lights off together. This function can be used, for instance, for central control of an area of a building.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>
4.1.5.8 FB_LightControl

Function block for daylight-dependent lighting control with up to 30 interpolation points.

At the core of this function block is an input/control value table consisting of 30 elements with threshold switching. If the input value reaches the range of an interpolation point (\(arrControlTable[n].nActualValue - arrControlTable[n].nSwitchRange/2\) ... \(arrControlTable[n].nActualValue + arrControlTable[n].nSwitchRange/2\)), the control value jumps to the corresponding function value \(arrControlTable[n].nSetpoint\) (see diagram).

Coupled to this is a ramp block that runs up the control value over the time \(tRampTime\).

When switching on with a positive edge at \(bOn\), however, the light is initially switched directly to the nearest control value. Only then is the controller activated. While the control is active, "post-starting" can take place at any time with a positive edge at \(bOn\), thus directly controlling the light to the nearest control value. A positive edge at \(bOff\) switches the light off directly.

The whole range of the table does not have to be used. The first table element (\(arrControlTable\), see below) for which the parameter \(nSwitchrange\) is 0 is considered the start of the unused area.
VAR_INPUT

bEnable : BOOL;
bOn : BOOL;
bOff : BOOL;
nActualValue : UINT;
tRampTime : TIME := t#30s;
arrControlTable : ARRAY[1..30] OF ST_ControlTable;
nOptions : DWORD;

bEnable: The bOn and bOff inputs are active as long as this input is TRUE. A negative state deactivates the inputs.

bOn: A rising edge directly switches nLightLevel to the nearest control value.

bOff: A rising edge directly switches nLightLevel to "0".

nActualValue: Current brightness.

tRampTime: Time period during which the brightness value is controlled to the next control value. (Preset value: 30 s).

arrControlTable: Input value/control value table. arrControlTable[1] to arrControlTable[30] of type ST_ControlTable [1].

nOptions: Reserved for future developments.

VAR_OUTPUT

bLight : BOOL;
bBusy : BOOL;
bControlActive : BOOL;
bError : BOOL;
nErrorId : UDINT;

bLight: This output is set as long as nLightLevel is greater than "0".

bBusy: This output is always active as long as the processing of a command (bOn, bOff, bToggle or ramp) is active.

bControlActive: This output is active as long as the control is active.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in nErrorId. Is reset to FALSE by the execution of a command at the inputs.

nErrorId: Contains the specific error code of the most recently executed command. Is reset to "0" by the execution of a command at the inputs. See Error codes [1].

VAR_IN_OUT

nLightLevel : UINT;

nLightLevel: Output control value of the function block and reference to the light output value.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>
Function block for realizing a light-ramp.

A rising edge at input bOn switches the light to the maximum value (32767). A rising edge at input bOff switches the light off again. Rising edges at the bToggle input invert the respective light state. A positive edge at the bStart input allows the function block to dim the light from the current level to nEndLevel. The time required for this is defined by tRampTime. All inputs are only active as long as bEnable is TRUE, otherwise the function block is reset internally.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEnable</td>
<td>BOOL</td>
</tr>
<tr>
<td>bOn</td>
<td>BOOL</td>
</tr>
<tr>
<td>bOff</td>
<td>BOOL</td>
</tr>
<tr>
<td>bToggle</td>
<td>BOOL</td>
</tr>
<tr>
<td>bStart</td>
<td>BOOL</td>
</tr>
<tr>
<td>nEndLevel</td>
<td>BYTE</td>
</tr>
<tr>
<td>tRampTime</td>
<td>TIME</td>
</tr>
<tr>
<td>nOptions</td>
<td>DWORD</td>
</tr>
</tbody>
</table>

**bEnable**: The bOn, bOff, bToggle and bStart inputs are active as long as this input is TRUE. A negative state deactivates the inputs and resets the function block.

**bOn**: A rising edge directly switches nLightLevel to the maximum value (32767).

**bOff**: A rising edge directly switches nLightLevel to "0".

**bToggle**: Switches the light state between On (32767) and Off (0).

**bStart**: If a rising edge is applied to this input, the light is dimmed up or down from the current value to nEndLevel. The time required for this is defined by tRampTime. The dimming procedure can be interrupted at any time by bOn, bOff or bToggle.

**nEndLevel**: Target value of the dimming procedure.

**tRampTime**: Ramp time, see bStart. (Preset value: 10 seconds).

**nOptions**: Reserved for future developments.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bLight</td>
<td>BOOL</td>
</tr>
<tr>
<td>bBusy</td>
<td>BOOL</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
</tr>
<tr>
<td>nErrorId</td>
<td>UDINT</td>
</tr>
</tbody>
</table>

**bLight**: This output is set as long as nLightLevel is greater than 0.

**bBusy**: This output is always active as long as the processing of a command (bOn, bOff, bToggle or ramp) is active.

**bError**: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in nErrorId. Is reset to FALSE by the execution of a command at the inputs.
**nErrorId**: Contains the command-specific error code of the most recently executed command. Is reset to "0" by the execution of a command at the inputs. See [Error codes](#).

**VAR_IN_OUT**

```plaintext
nLightLevel : UINT;
```

**nLightLevel**: Output control value of the function block and reference to the light output value. This value is defined as In-Out variable because the function block reads and writes the light value.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

### 4.1.5.10 FB_Sequencer

Function block for realizing light sequences with up to 50 interpolation points.

At the core of this function block is a ramp that approaches individual brightness values defined in a table within an adjustable time and then remains at this brightness value for a time that can also be defined. After the dwell time the next value is then driven to. As already mentioned, the table `arrSequenceTable` consists of 50 entries with the values for `nTargetValue` (target value), `tRampTime` (time taken to reach the target value) and `tProlongTime` (dwell time at the target value). It is not absolutely necessary to use all 50 values. A 0 entry of all 3 values marks the end of a sequence. Beyond that it is possible using the `nStartIndex` input to have a light sequence begin at any desired place in the table. This allows several different light sequences to be programmed even within the 50 entries, the sequences being separated from one another by 0 entry elements:
Over the course of time sequence 1, for example, looks like the following (\(n\text{StartIndex}=1, n\text{Options.bit0}=TRUE\), see below for explanation):
In addition, the function block can be switched on and off "normally" (On: nLightLevel = 32767, Off: nLightLevel = 0) or toggled between "On" and "Off" via the input bToggle. However, none of the command inputs is active unless the bEnable input is TRUE. If it is reset to FALSE, no more commands are accepted and the light value retains its current state – even from a ramp.

\[
\begin{align*}
\text{VAR_INPUT} & \\
\text{bEnable} & : \text{BOOL}; \\
\text{bOn} & : \text{BOOL}; \\
\text{bOff} & : \text{BOOL}; \\
\text{bToggle} & : \text{BOOL}; \\
\text{bStart} & : \text{BOOL}; \\
\text{nStartIndex} & : \text{USINT}; \\
\text{arrSequenceTable} & : \text{ARRAY}[1..50] \text{OF} \text{ST_SequenceTable}; \\
\text{nOptions} & : \text{DWORD}; \\
\end{align*}
\]

\textit{bEnable}: The \textit{bOn}, \textit{bOff}, \textit{bToggle} and \textit{bStart} inputs are active as long as this input is \textit{TRUE}. A negative state deactivates the inputs and resets the function block.

\textit{bOn}: A rising edge directly switches \textit{nLightLevel} to the maximum value (32767).

\textit{bOff}: A rising edge directly switches \textit{nLightLevel} to "0".

\textit{bToggle}: Switches the light state between On (32767) and Off (0).

\textit{bStart}: A positive edge starts a light sequence from the beginning defined under \textit{nStartIndex}.

\textit{nStartIndex}: See \textit{bStart}.

\textit{arrSequenceTable}: Light value table with the corresponding ramp and dwell times (see \textit{ST_SequenceTable} [94]).

\textit{nOptions}: Parameterization input. The setting (or non-setting) of the individual bits of this variable of the type \textit{DWORD} has the following effect:
### Constant

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTION_INFINITE_LOOP</td>
<td>Following the expiry of a sequence, the function block automatically</td>
</tr>
<tr>
<td></td>
<td>continues at the point defined at nStartIndex. If this option is not set,</td>
</tr>
<tr>
<td></td>
<td>the sequence stops after it has elapsed. A new positive edge at bStart</td>
</tr>
<tr>
<td></td>
<td>would be necessary to restart a sequence.</td>
</tr>
</tbody>
</table>

### VAR_OUTPUT

- **nActualIndex**: Reference to the current element in the sequence table. If a sequence has been completed (bSequenceActive = FALSE, see below), this output goes to "0".
- **bLight**: This output is set as long as nLightLevel is greater than "0".
- **bSequenceActive**: On processing a sequence this output is set to TRUE.
- **bBusy**: This output is always active as long as the processing of a command (bOn, bOff, bToggle or ramp) is active.
- **bError**: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in nErrorId. Is reset to FALSE by the execution of a command at the inputs.
- **nErrorId**: Contains the specific error code of the most recently executed command. Is reset to "0" by the execution of a command at the inputs. See Error codes [92].

### VAR_IN_OUT

- **nLightLevel**: Output control value of the function block and reference to the light output value.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

### 4.1.5.11 FB_StairwellDimmer

A rising edge at the input bSwitch sets the analog output nOut to the value nPresenceValue. A falling edge on bSwitch starts or restarts a timer with the runtime tPresenceDuration. Following the expiry of this timer, nOut is dimmed to the value nProlongValue over the time period tFadeOffDuration. This value is maintained for the time period tProlongDuration. After that, nOut is set to 0. A positive edge at the input bOff switches the output nOut to 0 immediately. The digital output value bLight is always set when nOut is greater than 0.
VAR_INPUT

bSwitch : BOOL;
bOff : BOOL;
nPresenceValue : UINT := 32767;
nProlongValue : UINT := 10000;
tPresenceDuration : TIME := t#120s;
tFadeOffDuration : TIME := t#10s;
tProlongDuration : TIME := t#20s;

VAR_OUTPUT

nOut : UINT;
bLight : BOOL;

bSwitch: Upon a positive edge: \( nOut \) is set to \( nPresenceValue \). Upon a negative edge: start of the presence time (see diagram).

bOff: Switches \( nOut \) off immediately.

nPresenceValue: Value to which \( nOut \) should be set during the presence time. (Preset value: 32767).

nProlongValue: Value to which \( nOut \) should be set during the dwell time. (Preset value: 10000).

tPresenceDuration: Duration of the presence time in which \( nOut \) is set to \( nPresenceValue \) following a falling edge on \( bSwitch \). (Preset value: 120 seconds).

tFadeOffDuration: Duration over which \( nOut \) is faded down to the dwell time following the presence time. (Preset value: 10 seconds).

tProlongDuration: Duration of the dwell time. (Preset value: 20 seconds).

VAR_OUTPUT

nOut : UINT;
bLight : BOOL;

nOut: Output of the momentary light value.

bLight: This output is set as long as \( nOut \) is greater than 0.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>
4.1.5.12 FB_StairwellLight

A positive edge at the `bSwitch` input sets the `bLight` output. The output is reset once the `tLightDuration` time has elapsed. If a signal is presented again to the `bSwitch` input before this time has elapsed, the timer is restarted. When `tPreWarningStart` has elapsed, the light is switched off (as a prewarning) for the period `tPreWarningDuration`. If this prewarning is not to be given, the parameter `tPreWarningStart` must be set to 0. A positive edge at the `bOff` input switches the output off immediately.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bSwitch</code></td>
<td>Switches the output on for the period of time given by <code>tLightDuration</code>.</td>
</tr>
<tr>
<td><code>bOff</code></td>
<td>Switches the output off.</td>
</tr>
<tr>
<td><code>tLightDuration</code></td>
<td>Period of time for which the output is set.</td>
</tr>
<tr>
<td><code>tPreWarningStart</code></td>
<td>Warning time.</td>
</tr>
<tr>
<td><code>tPreWarningDuration</code></td>
<td>Duration of the prewarning.</td>
</tr>
</tbody>
</table>

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bLight</code></td>
<td>The output is set for the duration of <code>tLightDuration</code> with a positive edge at <code>bSwitch</code>.</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>
4.1.6 Scene management

4.1.6.1 FB_RoomOperation
The function block FB_RoomOperation() is conceived for the management of lighting and blinds. Scenes are called and dimmed in a state of rest. Lighting and blinds can be set and saved in the appropriate mode. This function block is intended for use with the function blocks FB_ScenesLighting() [62], FB_ScenesVenetianBlind() [65], FB_Dimmer1Switch() [36] and FB_VenetianBlindEx() [22].

**Calling saved scenes:**

A rising edge at the input bSwitch_A, bSwitch_B or bSwitch_1..14 causes a pulse to be output at the output bInvokeScene_A, bInvokeScene_B or bInvokeScene_1..14.

**Dimming saved scenes:**

A scene is called and dimmed up by a signal that is applied to the input bSwitch_A, bSwitch_B or bSwitch_1..14 for a time exceeding tCycleDelayDimmTime.

**Setting blind and lighting values:**

A signal at the input bSwitchLightingMode or bSwitchBlindingMode switches to the respective mode. The control values are changed by the inputs bSwitch_1..14 via the outputs bSwitchLighting_1..14 or bSwitchBlindUp / bSwitchBlindDown_1..7.

**Saving the settings:**

By means of setting the input bSwitchLightingMode or bSwitchBlindingMode and a signal at the input bSwitch_A, bSwitch_B or bSwitch_1..14, a pulse is output at the output bSaveScene_A, bSaveScene_B or bSaveScene_1..14. The values are saved in the function block FB_ScenesLighting() [62], FB_ScenesVenetianBlind() [65].

**VAR_INPUT**

```plaintext
bSwitch_A : BOOL;
bSwitch_B : BOOL;
bSwitch_1 : BOOL;
bSwitch_2 : BOOL;
bSwitch_3 : BOOL;
bSwitch_4 : BOOL;
bSwitch_5 : BOOL;
bSwitch_6 : BOOL;
bSwitch_7 : BOOL;
bSwitch_8 : BOOL;
bSwitch_9 : BOOL;
bSwitch_10 : BOOL;
bSwitch_11 : BOOL;
bSwitch_12 : BOOL;
bSwitch_13 : BOOL;
bSwitch_14 : BOOL;
bSwitchLightingMode : BOOL;
bSwitchBlindUpMode : BOOL;
bFeedbackLighting_1 : BOOL;
bFeedbackLighting_2 : BOOL;
bFeedbackLighting_3 : BOOL;
bFeedbackLighting_4 : BOOL;
bFeedbackLighting_5 : BOOL;
bFeedbackLighting_6 : BOOL;
bFeedbackLighting_7 : BOOL;
bFeedbackLighting_8 : BOOL;
bFeedbackLighting_9 : BOOL;
bFeedbackLighting_10 : BOOL;
bFeedbackLighting_11 : BOOL;
bFeedbackLighting_12 : BOOL;
bFeedbackLighting_13 : BOOL;
bFeedbackLighting_14 : BOOL;
nFeedbackLighting_1 : UINT;
nFeedbackLighting_2 : UINT;
nFeedbackLighting_3 : UINT;
nFeedbackLighting_4 : UINT;
nFeedbackLighting_5 : UINT;
nFeedbackLighting_6 : UINT;
nFeedbackLighting_7 : UINT;
nFeedbackLighting_8 : UINT;
nFeedbackLighting_9 : UINT;
```
nFeedbackLighting_10 : UINT;
nFeedbackLighting_11 : UINT;
nFeedbackLighting_12 : UINT;
nFeedbackLighting_13 : UINT;
nFeedbackLighting_14 : UINT;
nFeedbackBlind_1    : UINT;
nFeedbackBlind_2    : UINT;
nFeedbackBlind_3    : UINT;
nFeedbackBlind_4    : UINT;
nFeedbackBlind_5    : UINT;
nFeedbackBlind_6    : UINT;
nFeedbackBlind_7    : UINT;
tCycleDelayDimmTime : TIME := t#500ms;
tOperationTime      : TIME := t#60s;

bSwitch_A, B: calls the saved Scene A or Scene B.

bSwitch_1..14: sets and calls the saved scenes.

bSwitchLightingMode: switches to the lighting mode.

bSwitchBlindingMode: switches to the blinding mode.

bFeedbackLighting_1..14: current status of the respective lamp. Return value from the dimmer function block FB_Dimmer1Switch().

nFeedbackLighting_1..14: current control value of the respective lamp. Return value from the dimmer function block FB_Dimmer1Switch().

nFeedbackBlind_1..7: current control value of the respective blind. Return value from the blind function block FB_VenetianBlindEx().

tCycleDelayDimmTime: switching time between dimming and calling a scene.

tOperationTime: if the blinding or lighting mode is active and no operation takes place, the mode is automatically switched back to scene mode after the expiry of this time.

VAR_OUTPUT
bEnableLightingMode  : BOOL;
bEnableBlindingMode  : BOOL;
bSwitchLighting_1   : BOOL;
bSwitchLighting_2   : BOOL;
bSwitchLighting_3   : BOOL;
bSwitchLighting_4   : BOOL;
bSwitchLighting_5   : BOOL;
bSwitchLighting_6   : BOOL;
bSwitchLighting_7   : BOOL;
bSwitchLighting_8   : BOOL;
bSwitchLighting_9   : BOOL;
bSwitchLighting_10  : BOOL;
bSwitchLighting_11  : BOOL;
bSwitchLighting_12  : BOOL;
bSwitchLighting_13  : BOOL;
bSwitchLighting_14  : BOOL;
bSwitchBlindUp_1    : BOOL;
bSwitchBlindDown_1  : BOOL;
bSwitchBlindUp_2    : BOOL;
bSwitchBlindDown_2  : BOOL;
bSwitchBlindUp_3    : BOOL;
bSwitchBlindDown_3  : BOOL;
bSwitchBlindUp_4    : BOOL;
bSwitchBlindDown_4  : BOOL;
bSwitchBlindUp_5    : BOOL;
bSwitchBlindDown_5  : BOOL;
bSwitchBlindUp_6    : BOOL;
bSwitchBlindDown_6  : BOOL;
bSwitchBlindUp_7    : BOOL;
bSwitchBlindDown_7  : BOOL;
bInvokeScene_A      : BOOL;
bInvokeScene_B      : BOOL;
bInvokeScene_1      : BOOL;
bInvokeScene_2      : BOOL;
bInvokeScene_3      : BOOL;
bInvokeScene_4      : BOOL;
bInvokeScene_5      : BOOL;
bEnableLightingMode: enables the memory function block FB_ScenesLighting().

bEnableBlindingMode: enables the memory function block FB_ScenesVenetianBlind().

bSwitchLighting_1..14: output for operating the dimmer function block FB_Dimmer1Switch() via the input bSwitchDimm.

bSwitchBlindUp_1..7: output for operating the blind function block FB_VenetianBlindEx() via the input bSwitchOverUp.

bSwitchBlindDown_1..7: output for operating the blind function block FB_VenetianBlindEx() via the input bSwitchOverDown.

bInvokeScene_A, B, 1..14: output signal for loading a scene. Is passed on to the function blocks FB_ScenesLighting() und FB_ScenesVenetianBlind().

bSaveScene_A, B, 1..14: output signal for saving a scene. Is passed on to the function blocks FB_ScenesLighting() und FB_ScenesVenetianBlind().

bLEDSwitch_1..14: these outputs indicate the status of the respective lighting (on/off) or shading (0%/100%). These outputs are always FALSE in scene mode.

bLEDLightingMode: this output is TRUE if lighting mode is active.

bLEDBlindingMode: this output is TRUE if blinding mode is active.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>
### 4.1.6.2 FB_ScenesLighting

```plaintext
FB_ScenesLighting

- bEnable BOOL
- bInvokeScene_1 BOOL
- bInvokeScene_2 BOOL
- bInvokeScene_3 BOOL
- bInvokeScene_4 BOOL
- bInvokeScene_5 BOOL
- bInvokeScene_6 BOOL
- bInvokeScene_7 BOOL
- bInvokeScene_8 BOOL
- bInvokeScene_9 BOOL
- bInvokeScene_10 BOOL
- bInvokeScene_11 BOOL
- bInvokeScene_12 BOOL
- bInvokeScene_13 BOOL
- bInvokeScene_14 BOOL
- bInvokeScene_15 BOOL
- bInvokeScene_16 BOOL
- bSaveScene_1 BOOL
- bSaveScene_2 BOOL
- bSaveScene_3 BOOL
- bSaveScene_4 BOOL
- bSaveScene_5 BOOL
- bSaveScene_6 BOOL
- bSaveScene_7 BOOL
- bSaveScene_8 BOOL
- bSaveScene_9 BOOL
- bSaveScene_10 BOOL
- bSaveScene_11 BOOL
- bSaveScene_12 BOOL
- bSaveScene_13 BOOL
- bSaveScene_14 BOOL
- bSaveScene_15 BOOL
- bSaveScene_16 BOOL
- nActualValueLighting_1 UINT
- nActualValueLighting_2 UINT
- nActualValueLighting_3 UINT
- nActualValueLighting_4 UINT
- nActualValueLighting_5 UINT
- nActualValueLighting_6 UINT
- nActualValueLighting_7 UINT
- nActualValueLighting_8 UINT
- nActualValueLighting_9 UINT
- nActualValueLighting_10 UINT
- nActualValueLighting_11 UINT
- nActualValueLighting_12 UINT
- nActualValueLighting_13 UINT
- nActualValueLighting_14 UINT
- sFile STRING
- nOptions UDINT
- bSetDimmValue_1 BOOL
- nDimmValue_1 UINT
- bSetDimmValue_2 BOOL
- nDimmValue_2 UINT
- bSetDimmValue_3 BOOL
- nDimmValue_3 UINT
- bSetDimmValue_4 BOOL
- nDimmValue_4 UINT
- bSetDimmValue_5 BOOL
- nDimmValue_5 UINT
- bSetDimmValue_6 BOOL
- nDimmValue_6 UINT
- bSetDimmValue_7 BOOL
- nDimmValue_7 UINT
- bSetDimmValue_8 BOOL
- nDimmValue_8 UINT
- bSetDimmValue_9 BOOL
- nDimmValue_9 UINT
- bSetDimmValue_10 BOOL
- nDimmValue_10 UINT
- bSetDimmValue_11 BOOL
- nDimmValue_11 UINT
- bSetDimmValue_12 BOOL
- nDimmValue_12 UINT
- bSetDimmValue_13 BOOL
- nDimmValue_13 UINT
- bSetDimmValue_14 BOOL
- nDimmValue_14 UINT
- bint BOOL
- bError BOOL
- nErrorId UDINT
```

---

TF8010 Building Automation Basic

Version: 1.0
This function block is intended for the management of lighting scenes. The function block is enabled via the \texttt{bEnable} input. The loading of the saved scenes is started by a positive edge at the \texttt{bEnable} input. The input must remain \texttt{TRUE} until the operation is completed. The values of the scenes are saved non-volatile in the TwinCAT Boot directory as a *.bin file. The last data status is saved in a *.bak file as a backup.

**Save scene**

The values of the inputs \texttt{nActualValueLighting_1..14} are saved in the respective scene by a rising edge at the input \texttt{bSaveScene_1..16}.

**Load scenes**

The saved values are output at the output \texttt{nDimmValue_1..14} by a rising edge at the input \texttt{bInvokeScene_1..16}. Furthermore, a positive edge is generated at output \texttt{bSetDimmValue_1..14} for a PLC cycle.

**VAR_INPUT**

\begin{verbatim}
VAR_INPUT
bEnable               : BOOL;
bInvokeScene_1        : BOOL;
bInvokeScene_2        : BOOL;
bInvokeScene_3        : BOOL;
bInvokeScene_4        : BOOL;
bInvokeScene_5        : BOOL;
bInvokeScene_6        : BOOL;
bInvokeScene_7        : BOOL;
bInvokeScene_8        : BOOL;
bInvokeScene_9        : BOOL;
bInvokeScene_10       : BOOL;
bInvokeScene_11       : BOOL;
bInvokeScene_12       : BOOL;
bInvokeScene_13       : BOOL;
bInvokeScene_14       : BOOL;
bInvokeScene_15       : BOOL;
bInvokeScene_16       : BOOL;
bSaveScene_1          : BOOL;
bSaveScene_2          : BOOL;
bSaveScene_3          : BOOL;
bSaveScene_4          : BOOL;
bSaveScene_5          : BOOL;
bSaveScene_6          : BOOL;
bSaveScene_7          : BOOL;
bSaveScene_8          : BOOL;
bSaveScene_9          : BOOL;
bSaveScene_10         : BOOL;
bSaveScene_11         : BOOL;
bSaveScene_12         : BOOL;
bSaveScene_13         : BOOL;
bSaveScene_14         : BOOL;
bSaveScene_15         : BOOL;
bSaveScene_16         : BOOL;
nActualValueLighting_1: UINT;
nActualValueLighting_2: UINT;
nActualValueLighting_3: UINT;
nActualValueLighting_4: UINT;
nActualValueLighting_5: UINT;
nActualValueLighting_6: UINT;
nActualValueLighting_7: UINT;
nActualValueLighting_8: UINT;
nActualValueLighting_9: UINT;
nActualValueLighting_10: UINT;
nActualValueLighting_11: UINT;
nActualValueLighting_12: UINT;
nActualValueLighting_13: UINT;
nActualValueLighting_14: UINT;
sFile                 : STRING;
nOptions              : UDINT;
\end{verbatim}

**bEnable:** enables the function block.

**bInvokeScene_1..16:** calls the respective scene.

**bSaveScene_1..16:** saves the current analog value \texttt{nActualValueLighting_1..14} in the respective scene.
nActualValueLighting_1..14: current control value of the respective lamp. Return value from the dimmer function block FB_Dimmer1Switch() [36].

sFile: file name (without path and file extension) for saving the scenes. The file name must be unique in the entire project. If several instances of the function blocks FB_ScenesLighting() or FB_ScenesVenetianBlind() [65] are created, then each instance must use a different file name. The file is always saved to the TwinCAT Boot directory and is given the extension .bin. Example: 'ControlPanelA'.

nOptions: Reserved for future developments.

VAR_OUTPUT

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bSetDimmValue_1</td>
<td>BOOL</td>
</tr>
<tr>
<td>nDimmValue_1</td>
<td>UINT</td>
</tr>
<tr>
<td>bSetDimmValue_2</td>
<td>BOOL</td>
</tr>
<tr>
<td>nDimmValue_2</td>
<td>UINT</td>
</tr>
<tr>
<td>bSetDimmValue_3</td>
<td>BOOL</td>
</tr>
<tr>
<td>nDimmValue_3</td>
<td>UINT</td>
</tr>
<tr>
<td>bSetDimmValue_4</td>
<td>BOOL</td>
</tr>
<tr>
<td>nDimmValue_4</td>
<td>UINT</td>
</tr>
<tr>
<td>bSetDimmValue_5</td>
<td>BOOL</td>
</tr>
<tr>
<td>nDimmValue_5</td>
<td>UINT</td>
</tr>
<tr>
<td>bSetDimmValue_6</td>
<td>BOOL</td>
</tr>
<tr>
<td>nDimmValue_6</td>
<td>UINT</td>
</tr>
<tr>
<td>bSetDimmValue_7</td>
<td>BOOL</td>
</tr>
<tr>
<td>nDimmValue_7</td>
<td>UINT</td>
</tr>
<tr>
<td>bSetDimmValue_8</td>
<td>BOOL</td>
</tr>
<tr>
<td>nDimmValue_8</td>
<td>UINT</td>
</tr>
<tr>
<td>bSetDimmValue_9</td>
<td>BOOL</td>
</tr>
<tr>
<td>nDimmValue_9</td>
<td>UINT</td>
</tr>
<tr>
<td>bSetDimmValue_10</td>
<td>BOOL</td>
</tr>
<tr>
<td>nDimmValue_10</td>
<td>UINT</td>
</tr>
<tr>
<td>bSetDimmValue_11</td>
<td>BOOL</td>
</tr>
<tr>
<td>nDimmValue_11</td>
<td>UINT</td>
</tr>
<tr>
<td>bSetDimmValue_12</td>
<td>BOOL</td>
</tr>
<tr>
<td>nDimmValue_12</td>
<td>UINT</td>
</tr>
<tr>
<td>bSetDimmValue_13</td>
<td>BOOL</td>
</tr>
<tr>
<td>nDimmValue_13</td>
<td>UINT</td>
</tr>
<tr>
<td>bSetDimmValue_14</td>
<td>BOOL</td>
</tr>
<tr>
<td>nDimmValue_14</td>
<td>UINT</td>
</tr>
<tr>
<td>bInit</td>
<td>BOOL</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
</tr>
<tr>
<td>nErrorId</td>
<td>UDINT</td>
</tr>
</tbody>
</table>

bSetDimmValue_1..14: output with the edge for the input bSetDimmValue of the function block FB_Dimmer1Switch() [36].

nDimmValue_1..14: Output with the value for the input nDimmValue of the function block FB_Dimmer1Switch() [36].

bInit: this output goes TRUE as soon as the initialization of the function block is complete.

bError: this output is set to TRUE as soon as an error is detected during execution. The error code is contained in nErrorId.

nErrorId: contains the error code as soon as bError goes TRUE. See Error codes [92].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>
**4.1.6.3 FB_ScenesVenetianBlind**

This function block is intended for the management of blind scenes. The function block is enabled via the `bEnable` input. The loading of the saved scenes is started by a positive edge at the `bEnable` input. The input must remain TRUE until the operation is completed. The values of the scenes are saved non-volatile in the TwinCAT Boot directory as a *.bin file. The last data status is saved in a *.bak file as a backup.
Save scene

The values of the inputs \( nActualValueBlinding_{1..7} \) are saved in the respective scene by a rising edge at the input \( bSaveScene_{1..16} \).

Load scenes

The saved values are output at the output \( nBlindValue_{1..7} \) by a rising edge at the input \( bInvokeScene_{1..16} \). Furthermore, a positive edge is generated at the output \( bSetBlindValue_{1..7} \) for one PLC cycle.

**VAR_INPUT**

\[
\begin{align*}
\textit{bEnable} : & \text{ BOOL;} \\
\textit{bInvokeScene}_{1..16} : & \text{ BOOL;} \\
\textit{bSaveScene}_{1..16} : & \text{ BOOL;} \\
n\textit{ActualValueBlinding}_{1..7} : & \text{ UINT;} \\
n\textit{ActualValueBlinding}_{1..7} : & \text{ USINT;} \\
n\textit{ActualValueBlinding}_{1..7} : & \text{ USINT;} \\
n\textit{ActualValueBlinding}_{1..7} : & \text{ USINT;} \\
n\textit{ActualValueBlinding}_{1..7} : & \text{ USINT;} \\
s\textit{File} : & \text{ STRING;} \\
n\textit{Options} : & \text{ DWORD;} \\
\end{align*}
\]

\( \textit{bEnable} \): enables the function block.

\( \textit{bInvokeScene}_{1..16} \): calls the respective scene.

\( \textit{bSaveScene}_{1..16} \): saves the current analog value \( nActualValueBlinding_{1..14} \) in the respective scene.

\( nActualValueBlinding_{1..7} \): current control value of the respective blind. Return value from the blind function block \( \text{FB_VenetianBlindEx()} \).

\( s\text{File} \): file name (without path and file extension) for saving the scenes. The file name must be unique in the entire project. If several instances of the function blocks \( \text{FB_ScenesLighting()} \) or \( \text{FB_ScenesVenetianBlind()} \) are created, then each instance must use a different file name. The file is always saved to the TwinCAT Boot directory and is given the extension .bin. Example: ‘ControlPanelA’.

\( n\text{Options} \): Reserved for future developments.
VAR_OUTPUT

VAR_INPUT

VAR_OUTPUT

Requirements

4.1.7 Signal processing

4.1.7.1 FB_ShortLongClick

VAR_INPUT

VAR_OUTPUT

bSetupBlindValue_1..7: output with the edge for the input bPosition of the function block FB_VenetianBlindEx() [22].

nBlindValue_1..7: output with the value for the input nSetPosition of the function block FB_VenetianBlindEx() [22].

bInit: this output goes TRUE as soon as the initialization of the function block is complete.

bError: this output is set to TRUE as soon as an error is detected during execution. The error code is contained in nErrorId.

nErrorId: contains the error code as soon as bError goes TRUE. See Error codes [92].

Development environment | Required PLC library
--------------------------|----------------------
TwinCAT from v3.1.4020.32 | Tc2_BABasic from v3.1.0.0

If the bSwitch input is longer than the tSwitchTime, the bLongClick output is set for one PLC cycle. Otherwise, the bShortClick output is set.

bSwitch: Input signal.

tSwitchTime: Duration above which the input signal is to be interpreted as a long button press.

bShortClick: Indicates a short push of the button.
bLongClick: Indicates a long push of the button.

Example

In the following example, two push buttons are used to control two different lamps. A switch is assigned to each lamp. If a button is pressed for longer than 500 ms, both lamps are switched off.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

4.1.7.2 FB_SignallingContact

The two inputs tDelayOnTime and tDelayOffTime allow slow operation and slow release delays to be set. If a message signal is to be acknowledged before this time can be ended, this is done by means of the bQuitSignal input. The state of the signaling contact is communicated to the function block via the bContact input.

The state of the message signal is indicated by the nSignalState output. A message signal can adopt one of altogether 6 different states. Corresponding constants are defined in the library:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCSIGNAL_INVALID</td>
<td>The message signal still does not have a defined state.</td>
</tr>
<tr>
<td>TCSIGNAL_SIGNALED</td>
<td>The message signal is active.</td>
</tr>
<tr>
<td>TCSIGNAL_RESET</td>
<td>The message signal has been reset.</td>
</tr>
<tr>
<td>TCSIGNAL_CONFIRMED</td>
<td>The message signal is confirmed, but has not yet been reset.</td>
</tr>
<tr>
<td>TCSIGNAL_SIGNALCON</td>
<td>The message signal is active and confirmed.</td>
</tr>
<tr>
<td>TCSIGNAL_RESETCON</td>
<td>The message signal is confirmed and reset.</td>
</tr>
</tbody>
</table>

VAR_INPUT

tDelayOnTime : TIME := t#100ms;
tDelayOffTime : TIME := t#100ms;
bQuitSignal : BOOL;
bContact : BOOL;

4.1.7.2 FB_SignallingContact

VAR_INPUT

tDelayOnTime : TIME := t#100ms;
tDelayOffTime : TIME := t#100ms;
bQuitSignal : BOOL;
bContact : BOOL;

tDelayOnTime: Delay before setting the message signal.
tDelayOffTime: Delay before resetting the message signal.

bQuitSignal: Input to acknowledge message signal.

bContact: Input for the signaling contact.

VAR_OUTPUT

nSignalState : WORD;

nSignalState: Message status.

Examples

A message signal requiring acknowledgement is implemented in the following example. The variable bGateAlert represents the state of the message signal. If the output nSignalState has the value TCSIGNAL_SIGNALALED or TCSIGNAL_RESET, the message is active. A positive edge at the bQuitSignal input acknowledges the message signal.

The following example illustrates the simplest case. A message signal not requiring acknowledgement.

The slow release delay allows the message signal to remain active for a certain time. The slow operation delay can be used, for example, to suppress contact bounce.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

4.1.7.3 FB_SingleDoubleClick

If the input signal is presented twice within the time tSwitchTime, the bDoubleClick output is set for one PLC cycle. Otherwise, the bSingleClick output is set.
VAR_INPUT

bSwitch : BOOL;
tSwitchTime : TIME := t#500ms;

bSwitch: Input signal.
tSwitchTime: Duration above which the input signal is to be interpreted as a double button press.

VAR_OUTPUT

bSingleClick : BOOL;
bDoubleclick : BOOL;

bSingleClick: Indicates a single push of the button.
bDoubleclick: Indicates a double push of the button.

Example

In the following example, two switches are used to control two different lamps. A switch is assigned to each lamp. If a switch is pressed twice in rapid succession, both lamps are switched off.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

4.1.7.4  FB_ThresholdSwitch

VAR_INPUT

fSignal : LREAL;
fLowerLimit : LREAL := 16000;
fUpperLimit : LREAL := 17000;
tLowerLimitDelay : TIME := t#100ms;
tUpperLimitDelay : TIME := t#100ms;

fSignal: Input signal.
fLowerLimit: Lower limit value.
fUpperLimit: Upper limit value.
tLowerLimitDelay: Switching delay when passing beyond the lower limit.
tUpperLimitDelay: Switching delay when passing beyond the upper limit.

VAR_OUTPUT

bSwitchingSignal : BOOL;
bCrossLowerLimit : BOOL;
bCrossUpperLimit : BOOL;

bSwitchingSignal: State depends on bCrossLowerLimit and bCrossUpperLimit.
bCrossLowerLimit: Becomes TRUE for a cycle if fLowerLimit has fallen below the time tLowerLimitDelay. At the same time bSwitchingSignal becomes FALSE.
bCrossUpperLimit: Becomes TRUE for a cycle if fUpperLimit is exceeded for the time tUpperLimitDelay. At the same time bSwitchingSignal becomes TRUE.

Example

In the following example, the two lamps can each be controlled with one switch. The two lamps are automatically switched in response to the outside brightness and the threshold switch. The lamps are switched on if the outside brightness is less than 1000 lux for 15 minutes. The lamps are switched off as soon as the brightness is greater than 2000 lux for more than 15 minutes.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

4.1.8 Timer functions

The timer function blocks are intended to trigger actions on certain days in the year/ month/ week. The action can be triggered via a start event or a start time and terminated via an end event, end time or duration. The following combinations are possible:
The grey-blue fields indicate the timer type. The day is determined by the periodicity (red fields) and further discretization (orange). A common feature of all function blocks is that they have the same start and end criteria (green). The start criterion relates to the selected day, the end criterion depends on the starting point. For each instance of a function block only one start and end criterion can be defined. To trigger several actions on the same day several instances of the function block are required.

Time overlaps

Time overlaps of two consecutive switch-on and switch-off criteria may occur in the same instance of the function block if the switching duration is not limited to less than 1 day. In this case a start event may be followed by another start event before the end of the preceding period. The following overlap scenarios are possible in the situation described above:

**Starttime / Endtime (type TOD, TOD)**

No overlap possible since for Starttime<Endtime the start and end point are on same day, and for Starttime>=Endtime the end point is assumed to be on the next day. This means that the duration is this limited to less than 1 day.

**Starttime / Duration (type TOD, TIME)**

Overlap is possible, since the duration is freely selectable and the TIME variable type can be up to 50 days. It would therefore be possible to trigger an action with a duration of 3 days on a daily basis. The action would never be completed since it would be constantly restarted.

**Starttime / End event (type TOD, BOOL)**

Overlap possible, since the end event is variable and cannot occur before the next start time.

**Start event / Endtime (type BOOL, TOD)**
Overlap may be possible. The end time is calculated when the start event occurs. If Starttime<Endtime the end time is on the same day. In this case no overlap is possible. On the other hand, if Starttime>=Endtime the end point is on the next day. An overlap occurs if the start is triggered on this day before the end of the previous action has been reached.

**Start event / Duration (type BOOL, TIME)**

Overlap is possible, since the duration is freely selectable and the TIME variable type can be up to 50 days. It would therefore be possible to trigger an action with a duration of 3 days on a daily basis. The action would never be completed since it would be constantly restarted.

**Start event / End event (type BOOL, BOOL)**

Overlap possible, since the end event is variable and cannot occur before the next start event.

An overlap means that the control output bOut for the respective function block does not change to FALSE. Instead, the system waits for the next period end. However, since the intention may also be that the function blocks merely trigger an action, the edge output bTriggerOn is also linked to the function blocks.

**Further documentation**

The following table contains an overview of the documentation for the individual function blocks:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>switches every n-th day</td>
<td>switches every n-th week on certain weekdays (multiple selection possible)</td>
<td>switches in certain months (multiple selection possible) on a certain day of the week</td>
<td>switches in certain months (multiple selection possible) on a certain day of the month</td>
<td>switches on a certain day of the year</td>
</tr>
</tbody>
</table>

**Sample program**

A sample program uses a daily switching function block (FB_DailyScheduler) to illustrate how the function blocks have to be parameterized.

### 4.1.8.1 FB_DailyScheduler

Function block for triggering actions every n-th day of the year.

- The function block triggers switching when the switching time is passed. Subsequent modification of the switching events or the time is therefore not permitted.

**VAR_INPUT**

| uiPeriodicity : UINT; |
| uiBegin : UINT; |
| eStartEnd : E_StartEnd; |
| stStartEnd : ST_StartEnd; |
| stSystemtime : TIMESTRUCT; |

**uiPeriodicity:** Periodicity or interval. May be within the range 1...365.
uiBegin: Starting value for the day counter. May be within the range 1...365.

Example1:
uiPeriodicity = 5,
uiBegin = 2: Switching events on 2 Jan., 7 Jan. 12 Jan. etc.

Example2:
uiPeriodicity = 3,
uiBegin = 1: Switching events on 1 Jan., 4 Jan. 7 Jan. etc.

eStartEnd: Selection of the start/end definition (see E_StartEnd [92]).

stStartEnd: Structure with the parameters defining the start and end. Unused variables, for example the
duration for the selection of start/end time, are ignored internally (see ST_StartEnd [94]).

stSystemtime: current time in TIMESTRUCT format. It is important to count every second.

VAR_OUTPUT

bOut : BOOL;
bTriggerOn : BOOL;
bNoEventNextYear : BOOL;
bError : BOOL;
nErrorId : UDINT;

bOut: Control output that is switched on or off by the start and end event.

bTriggerOn: Trigger output for switch-on events. This output is intended for logging switch-on events. If two
switch-on events occur consecutively they would not be detected via the control output bOut, since this
output would remain TRUE. See also time overlaps in the overview.

bNoEventNextYear: No day matching the parameterization was found within the next 366 days.

bError: This output is set to TRUE if the parameterization is faulty. The command-specific error code is
contained in nErrorId. Reset to FALSE once the parameterization is correct.

nErrorId: Contains the command-specific error code. Reset to 0 once the parameterization is correct. See
Error codes [92].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

4.1.8.2 FB_WeeklyScheduler

Function block for triggering actions on certain weekdays in each n-th week of the year.

The function block triggers switching when the switching time is passed. Subsequent modification of
the switching events or the time is therefore not permitted.
### VAR_INPUT

- **uiPeriodicity**: Periodicity or interval. May be within the range 1...52.
- **uiBegin**: Starting value for the week. May be within the range 1...52.

**Example 1:**
- **uiPeriodicity** = 5,
- **uiBegin** = 2: Switching events in week 2, week 7, week 12 etc.

**Example 2:**
- **uiPeriodicity** = 3,
- **uiBegin** = 1: Switching events in week 1, week 4, week 7 etc.

- **arrActiveWeekday**: Day of the week on which switching is to take place - **arrActiveWeekday[0]** => Sunday .. **arrActiveWeekday[6]** => Saturday. Multiple selections are possible.
- **eStartEnd**: Selection of the start/end definition (see **E_StartEnd** [92]).
- **stStartEnd**: Structure with the parameters defining the start and end. Unused variables, for example the duration for the selection of start/end time, are ignored internally (see **ST_StartEnd** [94]).
- **stSystemtime**: Current time in TIMESTRUCT format. It is important to count every second.

### VAR_OUTPUT

- **bOut**: Control output that is switched on or off by the start and end event.
- **bTriggerOn**: Trigger output for switch-on events. This output is intended for logging switch-on events. If two switch-on events occur consecutively they would not be detected via the control output **bOut**, since this output would remain TRUE. See also time overlaps in the overview.
- **bNoEventNextYear**: No day matching the parameterization was found within the next 366 days.
- **bError**: This output is set to TRUE if the parameterization is faulty. The command-specific error code is contained in **nErrorId**. Reset to FALSE once the parameterization is correct.
- **nErrorId**: Contains the command-specific error code. Reset to 0 once the parameterization is correct. See **Error codes** [92].

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>
4.1.8.3 FB_MonthlyScheduler1

Function block for triggering actions on a certain day of the week in certain months.

The function block triggers switching when the switching time is passed. Subsequent modification of the switching events or the time is therefore not permitted.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrActiveMonth</td>
<td>Month in which an action is to be triggered - arrActiveMonth[1] =&gt; January .. arrActiveMonth[12] =&gt; December. Multiple selections are possible.</td>
</tr>
<tr>
<td>uiActiveWeekday</td>
<td>Day of the week on which an action is to be triggered in the selected months. 0=Sunday ... 6=Saturday. Multiple selections are not possible; the maximum value is 6</td>
</tr>
<tr>
<td>eStartEnd</td>
<td>Selection of the start/end definition (see E_StartEnd [92]).</td>
</tr>
<tr>
<td>stStartEnd</td>
<td>Structure with the parameters defining the start and end. Unused variables, for example the duration for the selection of start/end time, are ignored internally (see ST_StartEnd [94]).</td>
</tr>
<tr>
<td>stSystemtime</td>
<td>Current time in TIMESTRUCT format. It is important to count every second.</td>
</tr>
</tbody>
</table>

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bOut</td>
<td>Control output that is switched on or off by the start and end event.</td>
</tr>
<tr>
<td>bTriggerOn</td>
<td>Trigger output for switch-on events. This output is intended for logging switch-on events. If two switch-on events occur consecutively they would not be detected via the control output bOut, since this output would remain TRUE. See also time overlaps in the overview.</td>
</tr>
<tr>
<td>bNoEventNextYear</td>
<td>No day matching the parameterization was found within the next 366 days.</td>
</tr>
<tr>
<td>bError</td>
<td>This output is set to TRUE if the parameterization is faulty. The command-specific error code is contained in nErrorId. Reset to FALSE once the parameterization is correct.</td>
</tr>
<tr>
<td>nErrorId</td>
<td>Contains the command-specific error code. Reset to 0 once the parameterization is correct. See Error codes [92].</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>
4.1.8.4 FB_MonthlyScheduler2

Function block for triggering actions on a certain day in certain months.

- The function block triggers switching when the switching time is passed. Subsequent modification of the switching events or the time is therefore not permitted.

VAR_INPUT

- arrActiveMonth: Month in which an action is to be triggered - arrActiveMonth[1] => January ...
  arrActiveMonth[12] => December. Multiple selections are possible.

- uiActiveDay: Day of the month on which an action is to be triggered. Multiple selections are not possible.

- eStartEnd: Selection of the start/end definition (see E_StartEnd [92]).

- stStartEnd: Structure with the parameters defining the start and end. Unused variables, for example the duration for the selection of start/end time, are ignored internally (see ST_StartEnd [94]).

- stSystemtime: Current time in TIMESTRUCT format. It is important to count every second.

VAR_OUTPUT

- bOut: Control output that is switched on or off by the start and end event.

- bTriggerOn: Trigger output for switch-on events. This output is intended for logging switch-on events. If two switch-on events occur consecutively they would not be detected via the control output bOut, since this output would remain TRUE. See also time overlaps in the overview.

- bNoEventNextYear: No day matching the parameterization was found within the next 366 days.

- bError: This output is set to TRUE if the parameterization is faulty. The command-specific error code is contained in nErrorId. Reset to FALSE once the parameterization is correct.

- nErrorId: Contains the command-specific error code. Reset to 0 once the parameterization is correct. See Error codes [92].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>
4.1.8.5  FB_YearlyScheduler

Function block for triggering actions on a certain day of the year.

The function block triggers switching when the switching time is passed. Subsequent modification of the switching events or the time is therefore not permitted.

**VAR_INPUT**

- uiMonth : UINT;  
- uiDay : UINT;  
- eStartEnd : E_StartEnd;  
- stStartEnd : ST_StartEnd;  
- stSystemtime : TIMESTRUCT;

uiMonth: Month in which an action is to be triggered. Multiple selections are not possible.

uiDay: Day on which an action is to be triggered. Multiple selections are not possible.

eStartEnd: Selection of the start/end definition (see E_StartEnd [p. 92]).

stStartEnd: Structure with the parameters defining the start and end. Unused variables, for example the duration for the selection of start/end time, are ignored internally (see ST_StartEnd [p. 94]).

stSystemtime: Current time in TIMESTRUCT format. It is important to count every second.

**VAR_OUTPUT**

- bOut : BOOL;  
- bTriggerOn : BOOL;  
- bNoEventNextYear : BOOL;  
- bError : BOOL;  
- nErrorId : UDINT;

bOut: Control output that is switched on or off by the start and end event.

bTriggerOn: Trigger output for switch-on events. This output is intended for logging switch-on events. If two switch-on events occur consecutively they would not be detected via the control output bOut, since this output would remain TRUE. See also time overlaps in the overview.

bNoEventNextYear: No day matching the parameterization was found within the next 366 days.

bError: This output is set to TRUE if the parameterization is faulty. The command-specific error code is contained in nErrorId. Reset to FALSE once the parameterization is correct.

nErrorId: Contains the command-specific error code. Reset to 0 once the parameterization is correct. See Error codes [p. 92].

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>
4.1.8.6 Timer program example

The following program example uses a day timer to illustrate how the function blocks should be parameterized, particularly with regard to the inputs eStartEnd, stStartEnd and stSystemTime.

We recommend using the function block NT_GetTime, which is available in the library Tc2_Utilities, for reading the system time in PC- and CX-based systems. A program for reading might look as follows:

```plaintext
PROGRAM P_SystemTime
VAR
    fbGetTime : NT_GetTime;
    dtSystemTime : DT;
    stDateTime : TIMESTRUCT;
    tonGetTime : TCN;
    nStep : INT;
    bSystemTimeValid : BOOL := FALSE;
END_VAR

(*Read the Time*)
CASE nStep OF
    0:
        tonGetTime(IN := TRUE, PT := T#500MS);
        IF (tonGetTime.Q) THEN
            tonGetTime(IN := FALSE);
            nStep := 10;
        END_IF
    10:
        fbGetTime(NETID := '',
            START := TRUE,
            TMOUT := T#2S);
        IF (NOT fbGetTime.BUSY) THEN
            stDateTime := fbGetTime.TIMESTR;
            dtSystemtime := SYSTEMTIME_TO_DT(fbGetTime.TIMESTR);
            fbGetTime(Start := FALSE);
            bSystemTimeValid := TRUE;
            nStep := 0;
        END_IF
END_CASE
```

It provides a timebase for parameterizing the scheduler function blocks with regard to the time input stSystemTime. The enumerator matching the required behavior is created at input eStartEnd:

- eSTARTTIME_ENDTIME Start criterion: Time - End criterion: Time
- eSTARTTIME_DURATION Start criterion: Time - End criterion: Duration
- eSTARTTIME_ENDEVENT Start criterion: Time - End criterion: Event (boolean input)
- eSTARTEVENT_ENDTIME Start criterion: Event (boolean input) - End criterion: Time
- eSTARTEVENT_DURATION Start criterion: Event (boolean input) - End criterion: Duration
- eSTARTEVENT_ENDEVENT Start criterion: Event (boolean input) - End criterion: Event (boolean input)

For the input stStartEnd a structure variable of the same type has to be declared that is referred to in the example as stStartEnd. The program then describes the subvariables of this structure, which are important for the function type. For the example shown, these are todStartTime and tDuration. All other variables are not read and therefore do not have to be described.
Both programs have to be called in the MAIN function block. The program part \( P_{\text{SchedulerExample}} \) may only be called once the program part \( P_{\text{SystemTime}} \) supplies valid data, i.e. once \( P_{\text{SystemTimeValid}} \) is TRUE. The reason for this protective logic is that reading the time takes several cycles which means that the time when the program starts is invalid and must not be used.

If the program starts on 1 January, the sequences is as follows:

```plaintext
PROGRAM MAIN
VAR
END_VAR

P_SystemTime();

IF NOT (P_SystemTime.bSystemTimeValid) THEN
    RETURN;
END_IF
P_SchedulerExample();
```

The days on which actions are triggered start with the 2nd of the year \( (uiBegin:=2) \). The process is repeated every three days \( (uiPeriodicity:=3) \). The switch-on time is 15:00 (\( stStartEnd.todStartTime:=\text{tod}#15:00:00 \)) and the switching duration is 6 hours \( (stStartEnd.tDuration:=\text{t#6h}) \).
4.1.8.7 FB_WeeklyTimeSwitch

The two parameters $t\text{SwitchOnTime}$ and $t\text{SwitchOffTime}$ define a time period during which the output $b\text{Output}$ is to be active. If the timer is only to apply on certain days of the week, this can be set via the inputs $b\text{Sunday}$, $b\text{Monday}$, ..., $b\text{Saturday}$. Several time channels can be switched by creating several instances of the function block. Each instance is responsible for one time channel.

**VAR_INPUT**

- $b\text{Enable}$: Function block enable.
- $t\text{CurrentDateTime}$: current date and time.
- $t\text{SwitchOnTime}$: switch-on time.
- $t\text{SwitchOffTime}$: switch-off time.
- $b\text{Sunday}$: consider timer on Sundays.
- $b\text{Monday}$: consider timer on Mondays.
- $b\text{Tuesday}$: consider timer on Tuesdays.
- $b\text{Wednesday}$: consider timer on Wednesdays.
- $b\text{Thursday}$: consider timer on Thursdays.
- $b\text{Friday}$: consider timer on Fridays.
- $b\text{Saturday}$: consider timer on Saturdays.

**VAR_OUTPUT**

- $b\text{Output}$: output becomes TRUE if the current time is between the switch-on time and the switch-off time.
- $b\text{EdgeOn}$: the output is set to TRUE for one PLC cycle when the time channel becomes active.
bEdgeOff: the output is set to TRUE for one PLC cycle when the time channel becomes deactivated.

Example

In the following example, the blinds are to be raised at 6.30 a.m. at weekends and lowered at 7.00 p.m. The two timer outputs bEdgeOn and bEdgeOff are linked to the inputs bUp and bDown of the blind function block. The pulses from the outputs then trigger raising or lowering of the blinds at the specified time.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

4.1.8.8 FB_CalcSunPosition

Calculation of sun position based on the date, time, longitude and latitude.

Description

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, where 0° means that the sun is in the horizontal plane of the observation site and 90° means that the sun is directly over the observer's head. The other angle indicates the direction in which the sun is standing. 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.

In specifying the time, the time according to Greenwich Mean Time (GMT) must be given.

The latitude is specified as the distance of a place on the surface of the earth from the equator to the north or to the south 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>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_CalcSunPosition()` returns a negative value for the height of the sun (`fSunElevation`), then the sun is not visible. This can be used to determine sunrise and sunset.

**VAR_INPUT**

- `fDegreeOfLongitude` : LREAL := 8.5;
- `fDegreeOfLatitude` : LREAL := 51.9;
- `dtGMT` : TIMESTRUCT;

- `fDegreeOfLongitude`: Longitude in degrees.
- `fDegreeOfLatitude`: Latitude in degrees.
- `dtGMT`: Current time as Greenwich Mean Time (GMT).
VAR_OUTPUT

fSunAzimuth : LREAL;
fSunElevation : LREAL;

fSunAzimuth: Direction of the sun (northern hemisphere: 0° north ... 90° east ... 180° south ... 270° west ... / southern hemisphere: 0° south ... 90° east ... 180° north ... 270° west ...).

fSunElevation: Height of the sun (0° horizontal ... 90° vertical).

Example

PROGRAM MAIN
VAR
  fbCalcSunPosition : FB_CalcSunPosition;
  fSunAzimuth : LREAL;
  fSunElevation : LREAL;
  fbGetSystemTime : GETSYSTEMTIME;
  fileTime : T_FILETIME;
END_VAR

fbGetSystemTime(timeLoDW=>fileTime.dwLowDateTime,
                 timeHiDW=>fileTime.dwHighDateTime);

fbCalcSunPosition( fDegreeOfLongitude := 8.5,
                   fDegreeOfLatitude := 51.9,
                   dtGMT := FILETIME_TO_SYSTEMTIME(fileTime));

fSunAzimuth := fbCalcSunPosition.fSunAzimuth;
fSunElevation := fbCalcSunPosition.fSunElevation;

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

4.1.8.9 FB_CalcSunriseSunset

Function block for calculating sunrise and sunset based on the longitude, latitude, reference meridian and time.

The earth is divided into several time zones. Each time zone is associated with a reference meridian. Reference meridian for some of the time zones:

<table>
<thead>
<tr>
<th>Time zone</th>
<th>Reference meridian</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMT (Greenwich Mean Time)</td>
<td>( \lambda_{\text{GMT}} = 0° )</td>
</tr>
<tr>
<td>CET (Central European Time)</td>
<td>( \lambda_{\text{CET}} = 15° )</td>
</tr>
<tr>
<td>CEST (Central European Summer Time)</td>
<td>( \lambda_{\text{CEST}} = 30° )</td>
</tr>
</tbody>
</table>

In specifying the time, the time according to Greenwich Mean Time (GMT) must be given.

This function block is only available in the PC version of the library.
VAR_INPUT

fDegreeOfLongitude   : LREAL := 8.5;
fDegreeOfLatitude    : LREAL := 51.9;
fReferenceMeridian   : LREAL;
dCurrentDate         : DATE;

fDegreeOfLongitude: Longitude in degrees.
fDegreeofLatitude: Latitude in degrees.
fReferenceMeridian: Reference meridian of the time zone.
dCurrentDate: current date.

VAR_OUTPUT

todSunrise         : TOD;
todSunset          : TOD;

todSunrise: Sunrise. Output of hour and minute.
todSunset: Sunset. Output of hour and minute.

Example

PROGRAM MAIN
VAR
    fbCalcSunriseSunset : FB_CalcSunriseSunset;
todSunrise           : TOD;
todSunset            : TOD;
fbGetSystemTime      : GETSYSTEMTIME;
fileTime             : T_FILETIME;
dtCurrentDate        : DT; END_VAR

fbGetSystemTime(timeLoDW => fileTime.dwLowDateTime,
    timeHiDW => fileTime.dwHighDateTime);
dtCurrentDate := FILETIME_TO_DT(fileTime);

fbCalcSunriseSunset{ fDegreeOfLongitude := 8.5, (* Longitude of Verl *)
    fDegreeOfLatitude := 51.9, (* Latitude of Verl *)
    fReferenceMeridian := 30.0, (* Central European Summer Time *)
    dCurrentDate := DT_TO_DATE(dtCurrentDate),
    todSunrise => todSunrise,
    todSunset => todSunset};

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>
German holidays for the current year are calculated based on the date entered. A boolean output indicates whether the entered date matches one of the calculated holidays. To ensure international readability the function block was translated into English. The parameters have the following meaning:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>dCurrentDate</td>
<td>DATE</td>
</tr>
<tr>
<td>dNewYearsDay</td>
<td>DATE</td>
</tr>
<tr>
<td>dEpiphany</td>
<td>DATE</td>
</tr>
<tr>
<td>dGoodFriday</td>
<td>DATE</td>
</tr>
<tr>
<td>dEasterSunday</td>
<td>DATE</td>
</tr>
<tr>
<td>dEasterMonday</td>
<td>DATE</td>
</tr>
<tr>
<td>dLabourDay</td>
<td>DATE</td>
</tr>
<tr>
<td>dAscensionDay</td>
<td>DATE</td>
</tr>
<tr>
<td>dWhitSunday</td>
<td>DATE</td>
</tr>
<tr>
<td>dWhitMonday</td>
<td>DATE</td>
</tr>
<tr>
<td>dCorpusChristi</td>
<td>DATE</td>
</tr>
<tr>
<td>dAssumptionDay</td>
<td>DATE</td>
</tr>
<tr>
<td>dGermanUnificationDay</td>
<td>DATE</td>
</tr>
<tr>
<td>dReformationDay</td>
<td>DATE</td>
</tr>
<tr>
<td>dAllSaintsDay</td>
<td>DATE</td>
</tr>
<tr>
<td>dPenanceDay</td>
<td>DATE</td>
</tr>
<tr>
<td>dChristmasEve</td>
<td>DATE</td>
</tr>
<tr>
<td>d1stChristmasDay</td>
<td>DATE</td>
</tr>
<tr>
<td>d2ndChristmasDay</td>
<td>DATE</td>
</tr>
<tr>
<td>dNewYearsEve</td>
<td>DATE</td>
</tr>
<tr>
<td>bNewYearsDay</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEpiphany</td>
<td>BOOL</td>
</tr>
<tr>
<td>bGoodFriday</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEasterSunday</td>
<td>BOOL</td>
</tr>
<tr>
<td>bEasterMonday</td>
<td>BOOL</td>
</tr>
<tr>
<td>bLabourDay</td>
<td>BOOL</td>
</tr>
<tr>
<td>bAscensionDay</td>
<td>BOOL</td>
</tr>
<tr>
<td>bWhitSunday</td>
<td>BOOL</td>
</tr>
<tr>
<td>bWhitMonday</td>
<td>BOOL</td>
</tr>
<tr>
<td>bCorpusChristi</td>
<td>BOOL</td>
</tr>
<tr>
<td>bAssumptionDay</td>
<td>BOOL</td>
</tr>
<tr>
<td>bGermanUnificationDay</td>
<td>BOOL</td>
</tr>
<tr>
<td>bReformationDay</td>
<td>BOOL</td>
</tr>
<tr>
<td>bAllSaintsDay</td>
<td>BOOL</td>
</tr>
<tr>
<td>bPenanceDay</td>
<td>BOOL</td>
</tr>
<tr>
<td>bChristmasEve</td>
<td>BOOL</td>
</tr>
<tr>
<td>b1stChristmasDay</td>
<td>BOOL</td>
</tr>
<tr>
<td>b2ndChristmasDay</td>
<td>BOOL</td>
</tr>
<tr>
<td>bNewYearsEve</td>
<td>BOOL</td>
</tr>
<tr>
<td>English name</td>
<td>German name</td>
</tr>
<tr>
<td>------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>New Years Day</td>
<td>Neujahr</td>
</tr>
<tr>
<td>Epiphany</td>
<td>Heilige Drei Könige</td>
</tr>
<tr>
<td>Good Friday</td>
<td>Karfreitag</td>
</tr>
<tr>
<td>Easter Sunday</td>
<td>Ostersonntag</td>
</tr>
<tr>
<td>Easter Monday</td>
<td>Ostermontag</td>
</tr>
<tr>
<td>Labor Day</td>
<td>Maifeiertag</td>
</tr>
<tr>
<td>Ascension Day</td>
<td>Christi Himmelfahrt</td>
</tr>
<tr>
<td>Whit Sunday</td>
<td>Pfingstsonntag</td>
</tr>
<tr>
<td>Whit Monday</td>
<td>Pfingstmontag</td>
</tr>
<tr>
<td>Corpus Christi</td>
<td>Fronleichnam</td>
</tr>
<tr>
<td>Assumption Day</td>
<td>Mariä Himmelfahrt</td>
</tr>
<tr>
<td>German Unification Day</td>
<td>Tag Der Deutschen Einheit</td>
</tr>
<tr>
<td>Reformation Day</td>
<td>Reformationstag</td>
</tr>
<tr>
<td>All Saints Day</td>
<td>Allerheiligen</td>
</tr>
<tr>
<td>Penance Day</td>
<td>Buß- und Bettag</td>
</tr>
<tr>
<td>Christmas Eve</td>
<td>Heiligabend</td>
</tr>
<tr>
<td>1st Christmas Day</td>
<td>1. Weihnachtstag</td>
</tr>
<tr>
<td>2nd Christmas Day</td>
<td>2. Weihnachtstag</td>
</tr>
<tr>
<td>New Years Eve</td>
<td>Silvester</td>
</tr>
</tbody>
</table>

**VAR_INPUT**

```plaintext
dCurrentDate    : DATE;
```

dCurrentDate: current date.

**VAR OUTPUT**

```plaintext
dNewYearsDay     : DATE;
dEpiphany        : DATE;
dGoodFriday      : DATE;
dEasterSunday    : DATE;
dEasterMonday    : DATE;
LabourDay        : DATE;
dAscensionDay    : DATE;
dWhitSunday      : DATE;
dWhitMonday      : DATE;
dCorpusChristi   : DATE;
dAssumptionDay   : DATE;
dGermanUnificationDay : DATE;
dReformationDay  : DATE;
dAllSaintsDay    : DATE;
PenanceDay       : DATE;
dChristmasEve    : DATE;
d1stChristmasDay : DATE;
d2ndChristmasDay : DATE;
NewYearsEve      : DATE;

dNewYearsDay      : BOOL;
bEpiphany         : BOOL;
bGoodFriday       : BOOL;
bEasterSunday     : BOOL;
bEasterMonday     : BOOL;
bLabourDay        : BOOL;
bAscensionDay     : BOOL;
bWhitSunday       : BOOL;
bWhitMonday       : BOOL;
bCorpusChristi    : BOOL;
bAssumptionDay    : BOOL;
bGermanUnificationDay : BOOL;
bReformationDay   : BOOL;
bAllSaintsDay     : BOOL;
bPenanceDay       : BOOL;
bChristmasEve     : BOOL;
```
b1stChristmasDay : BOOL;
b2ndChristmasDay : BOOL;
bNewYearsEve : BOOL;

dxxxxxx: Date of the respective holiday.
bxxxxxx: Boolean statement indicating whether today is the respective holiday.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

4.1.8.11 FB_CalcPublicHolidaysUS

Calculation of the United States public holidays.

US public holidays for the current year are calculated based on the date entered. A boolean output indicates whether the entered date matches one of the calculated holidays. To ensure international readability the function block was translated into English. The parameters have the following meaning:
<table>
<thead>
<tr>
<th>English name</th>
<th>German</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Year’s Day</td>
<td>Neujahr</td>
</tr>
<tr>
<td>Martin Luther King &quot;JR&quot; Day</td>
<td>Martin Luther King Tag</td>
</tr>
<tr>
<td>Presidents Day</td>
<td>Tag der Präsidenten</td>
</tr>
<tr>
<td>Good Friday</td>
<td>Karfreitag</td>
</tr>
<tr>
<td>Easter Sunday</td>
<td>Ostersonntag</td>
</tr>
<tr>
<td>Memorial Day</td>
<td>Gedenktag</td>
</tr>
<tr>
<td>Independence Day</td>
<td>Unabhängigkeitstag</td>
</tr>
<tr>
<td>Labor Day</td>
<td>Maifeiertag</td>
</tr>
<tr>
<td>Columbus Day</td>
<td>Kolumbus-Tag</td>
</tr>
<tr>
<td>Veterans Day</td>
<td>Veteranentag</td>
</tr>
<tr>
<td>Thanksgiving Day</td>
<td>Erntedank</td>
</tr>
<tr>
<td>Thanksgiving Friday</td>
<td>Schwarzver Friday (Freitag nach Erntedank)</td>
</tr>
<tr>
<td>Christmas Eve</td>
<td>Heiligabend</td>
</tr>
<tr>
<td>Christmas Day</td>
<td>Weihnachtstag</td>
</tr>
<tr>
<td>New Years Eve</td>
<td>Silvester</td>
</tr>
</tbody>
</table>

**VAR_INPUT**

dCurrentDate : DATE;

**VAR_OUTPUT**

dNewYearsDay : DATE;
dEpiphany : DATE;
dGoodFriday : DATE;
dEasterSunday : DATE;
dEasterMonday : DATE;
dLabourDay : DATE;
dAscensionDay : DATE;
dWhitSunday : DATE;
dWhitMonday : DATE;
dCorpusChristi : DATE;
dAssumptionDay : DATE;
dGermanUnificationDay : DATE;
dReformationDay : DATE;
dAllSaintsDay : DATE;
dPenanceDay : DATE;
dChristmasEve : DATE;
d1stChristmasDay : DATE;
d2ndChristmasDay : DATE;
dNewYearsEve : DATE;
bNewYearsDay : BOOL;
bEpiphany : BOOL;
bGoodFriday : BOOL;
bEasterSunday : BOOL;
bEasterMonday : BOOL;
bLabourDay : BOOL;
bAscensionDay : BOOL;
bWhitSunday : BOOL;
bWhitMonday : BOOL;
bCorpusChristi : BOOL;
bAssumptionDay : BOOL;
bGermanUnificationDay : BOOL;
bReformationDay : BOOL;
bAllSaintsDay : BOOL;
bPenanceDay : BOOL;
bChristmasEve : BOOL;
b1stChristmasDay : BOOL;
b2ndChristmasDay : BOOL;
bNewYearsEve : BOOL;

dxxxxxx: Date of the respective holiday.

bxxxxxx: Boolean statement indicating whether today is the respective holiday.
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

4.1.8.12 FB_CalcFederalHolidaysUS

US federal holidays for the current year are calculated based on the date entered. A boolean output indicates whether the entered date matches one of the calculated holidays. To ensure international readability the function block was translated into English. The parameters have the following meaning:

<table>
<thead>
<tr>
<th>English name</th>
<th>German name</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Years Day</td>
<td>Neujahr</td>
</tr>
<tr>
<td>Martin Luther King &quot;JR&quot; Day</td>
<td>Martin Luther King Tag</td>
</tr>
<tr>
<td>Presidents Day</td>
<td>Tag der Präsidenten</td>
</tr>
<tr>
<td>Memorial Day</td>
<td>Gedenktag</td>
</tr>
<tr>
<td>Independence Day</td>
<td>Unabhängigkeitstag</td>
</tr>
<tr>
<td>Labor Day</td>
<td>Labor Day</td>
</tr>
<tr>
<td>Columbus Day</td>
<td>Kolumbus-Tag</td>
</tr>
<tr>
<td>Veterans Day</td>
<td>Veteranentag</td>
</tr>
<tr>
<td>Thanksgiving Day</td>
<td>Erntedank</td>
</tr>
<tr>
<td>Christmas Day</td>
<td>Weihnachtstag</td>
</tr>
</tbody>
</table>

VAR INPUT

```
VAR_INPUT

dCurrentDate : DATE;
```

dCurrentDate: current date.
VAR_OUTPUT

dNewYearsDay : DATE;
dMartinLutherKingJrDay : DATE;
dPresidentsDay : DATE;
dMemorialDay : DATE;
dIndependenceDay : DATE;
DLaborDay : DATE;
DColumbusDay : DATE;
DVeteransDay : DATE;
dThanksgivingDay : DATE;
dChristmasDay : DATE;
dNewYearsDay : BOOL;
dMartinLutherKingJrDay : BOOL;
dPresidentsDay : BOOL;
dMemorialDay : BOOL;
dIndependenceDay : BOOL;
DLaborDay : BOOL;
DColumbusDay : BOOL;
DVeteransDay : BOOL;
dThanksgivingDay : BOOL;
dChristmasDay : BOOL;

dxxxxxx: Date of the respective holiday.

bxxxxxx: Boolean statement indicating whether today is the respective holiday.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>
4.1.9 Error Codes

<table>
<thead>
<tr>
<th>Value (hex)</th>
<th>Value (dec)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>0</td>
<td>No error.</td>
</tr>
<tr>
<td>0x0001</td>
<td>1</td>
<td>FB_MaximumDemandController() [15]: -- reserved error code --</td>
</tr>
<tr>
<td>0x0002</td>
<td>2</td>
<td>FB_MaximumDemandController() [15]: The input parameter fMeterConstant is &quot;0&quot;.</td>
</tr>
<tr>
<td>0x0003</td>
<td>3</td>
<td>FB_LightControl([48] : The switch range (nSwitchRange) in the first or second element of the value table arrControlTable is 0. It is thus assumed that the table has only one value set or none at all.</td>
</tr>
<tr>
<td>0x0004</td>
<td>4</td>
<td>FB_LightControl([48] : 2 neighboring input values nActualValue in the value table arrControlTable lie too close together i.e. each in the switching range of the other.</td>
</tr>
<tr>
<td>0x0005</td>
<td>5</td>
<td>FB_Sequencer([51] : The start index nStartIndex is outside of the valid range [1..50].</td>
</tr>
<tr>
<td>0x0006</td>
<td>6</td>
<td>FB_Sequencer([51] : The start index nStartIndex refers to a point that, for its part, marks the end of a sequence (zero entries).</td>
</tr>
<tr>
<td>0x0007</td>
<td>7</td>
<td>Scheduler function blocks: An input parameter is not in the valid range.</td>
</tr>
<tr>
<td>0x0008</td>
<td>8</td>
<td>Scheduler function blocks: None is set for the selection parameters (weekly time clock: selection of weekdays, monthly time clocks: selection of months).</td>
</tr>
<tr>
<td>0x0009</td>
<td>9</td>
<td>Scheduler function blocks: A day in the month was selected that is not valid.</td>
</tr>
<tr>
<td>0x000A</td>
<td>10</td>
<td>FB_ConstantLightControlEco([32] : Input parameter nMinLevel is greater than or equal to nMaxLevel.</td>
</tr>
<tr>
<td>0x000B</td>
<td>11</td>
<td>FB_ScenesLighting([62], FB_ScenesVenetianBlind([65]: Input parameter sFile is invalid (empty).</td>
</tr>
<tr>
<td>0x000C</td>
<td>12</td>
<td>FB_ScenesLighting([62], FB_ScenesVenetianBlind([65]: Internal error. File with the scene values was not found.</td>
</tr>
<tr>
<td>0x000D</td>
<td>13</td>
<td>FB_ScenesLighting([62], FB_ScenesVenetianBlind([65]: Internal error: No further free file handles available.</td>
</tr>
</tbody>
</table>

4.2 DUTs

4.2.1 Enumerations

4.2.1.1 E_StartEnd

```
TYPE E_StartEnd :
{
  eSTARTTIME_ENDTIME := 1,
  eSTARTTIME_DURATION := 2,
  eSTARTTIME_ENDEVENT := 3,
  eSTARTEVENT_ENDTIME := 4,
  eSTARTEVENT_DURATION := 5,
  eSTARTEVENT_ENDEVENT := 6
} INT := Undefined;
END_TYPE
```

eSTARTTIME_ENDTIME: Selection of start/end time. If the start time is equal or greater the end time, the end is allocated to the next day.
eSTARTTIME_DURATION: Selection of start time/duration.
eSTARTTIME_ENDEVENT: Selection of start time/end event.
eSTARTEVENT_ENDTIME: Selection of start event/end time. If the start time is equal or greater the end time, the end is allocated to the next day.

eSTARTEVENT_DURATION: Selection of start event/duration.

eSTARTEVENT_ENDEVENT: Selection of start event/end event.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

4.2.2 Structures

4.2.2.1 ST_ControlTable

TYPE ST_ControlTable :

STRUCT
  nActualValue : UINT;
  nControlValue : UINT;
  nSwitchRange : UINT;
END_STRUCT
END_TYPE

nActualValue: Current brightness.
nControlValue: Corresponding switch point (control value).
nSwitchRange: Threshold value around the input value interpolation point at which switching takes place. The entry "0" marks the beginning of the unused area of the table.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

4.2.2.2 ST_MDCInDataKL1501

TYPE ST_MDCInDataKL1501 :

STRUCT
  nStatus : USINT;
  nDummy1 : USINT;
  nDummy2 : USINT;
  nDummy3 : USINT;
  nData : DWORD;
END_STRUCT
END_TYPE

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

4.2.2.3 ST_MDCLoadParameters

TYPE ST_MDCLoadParameters :

STRUCT
  bConnected : BOOL;
  nDegreeOfPriority : INT;
  tMINPowerOnTime : TIME;
  tMAXPowerOffTime : TIME;
END_STRUCT
END_TYPE
bConnected: TRUE = consumer connected; FALSE = consumer not connected.

nDegreeOfPriority: Indicates the switch-off priority; consumers with a low priority will be switched off first. (1 => low; ... 8 => high priority)

tMINPowerOnTime: The minimum power-on time (minimum ramp-up time) during which the consumer may not be switched off.

tMINPowerOffTime: The minimum power-off time (recovery time) during which the consumer may not be switched on again.

tMAXPowerOffTime: The maximum power-off time after which the consumer must be switched on again.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

4.2.2.4  ST_MDCOutDataKL1502

TYPE ST_MDCOutDataKL1502 :

  STRUCT
    nCtrl             : USINT;
    nDummy1           : USINT;
    nDummy2           : USINT;
    nDummy3           : USINT;
    nData             : DWORD;
  END_STRUCT
END_TYPE

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

4.2.2.5  ST_SequenceTable

TYPE ST_SequenceTable :

  STRUCT
    nTargetValue    : UINT;
    tRampTime       : TIME;
    tProlongTime    : TIME;
  END_STRUCT
END_TYPE

nTargetValue: Target value.

tRampTime: Time to reach the target value.

tProlongTime: Dwell time at the target value.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

4.2.2.6  ST_StartEnd

TYPE ST_StartEnd :

  STRUCT
    todStartTime   : TOD;
    bStartEvent    : BOOL;
    tDuration      : TIME;
    todEndTime     : TOD;
  END_STRUCT
END_TYPE
bEndEvent : BOOL;
END_STRUCT
END_TYPE

todStartTime: Start time.
bStartEvent: Start event
tDuration: Switching duration.
todEndTime: End time.
bEndEvent: End event.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Required PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.32</td>
<td>Tc2_BABasic from v3.1.0.0</td>
</tr>
</tbody>
</table>

4.3 GVLs

4.3.1 Constants

VAR_GLOBAL CONSTANT
TCSIGNAL_INVALID : WORD := 16#0000);
TCSIGNAL_SIGNALED : WORD := 16#0001;
TCSIGNAL_RESET : WORD := 16#0002;
TCSIGNAL_CONFIRMED : WORD := 16#0010;
TCSIGNAL_SIGNALCON : WORD := 16#0011;
TCSIGNAL_RESETCON : WORD := 16#0012;
OPTION_INIFINITE_LOOP : DWORD := 16#0000_0001;
END_VAR
5 Appendix

5.1 Support and Service

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

Beckhoff's branch offices and representatives

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

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

You will also find further documentation for Beckhoff components there.

Beckhoff Headquarters

Beckhoff Automation GmbH & Co. KG
Huelshorstweg 20
33415 Verl
Germany

Phone: +49(0)5246/963-0
Fax: +49(0)5246/963-198
e-mail: info@beckhoff.com

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(0)5246/963-157
Fax: +49(0)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(0)5246/963-460
Fax: +49(0)5246/963-479
e-mail: service@beckhoff.com