BECKHOFF New Automation Technology

Manual | EN

TS8010

TwinCAT 2 | PLC Building Automation Basic

Supplement | Building Automation

2022-10-28 | Version: 1.1

Table of contents

1 Foreword

1.1 Notes on the documentation

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

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

Disclaimer

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

Trademarks

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

Patent Pending

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

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

Copyright

1.2 Safety instructions

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Description of symbols

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

⚠ DANGER

Serious risk of injury!

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

⚠ WARNING

Risk of injury!

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

⚠ CAUTION

Personal injuries!

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

NOTE

Damage to the environment or devices

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

Tip or pointer

This symbol indicates information that contributes to better understanding.

1.3 Notes on information security

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

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

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

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

2 General information

Further libraries are required

For PC systems (x86) and Embedded-PCs (CXxxxx):

- Standard.lib
- TcBase.lib
- TcSystem.lib
- TcUtilities.lib

For Bus Terminal Controller of BCxx00 series:

- Standard.lb6
- TcPlcUtilitiesBC.lb6
- PlcSystemBC.lb6

For Bus Terminal Controller of BCxx50, BCxx20 and BC9191 series:

- Standard.lbx
- TcBaseBCxx50.lbx
- TcSystemBCxx50.lbx

For Bus Terminal Controller of BXxx00 series:

- Standard.lbx
- TcBaseBX.lbx
- TcSystemBX.lbx

Memory usage

By linking the library PLC program memory is already consumed. Depending on the application program the remaining memory can not be sufficient.

3 PLC API

The TwinCAT PLC Building Automation Library contains a number of function blocks useful for the building automation.

Lightings

Name	Description
FB_Dimmer1Switch [▶ 11]	Light dimmer using a switch.
FB_Dimmer1SwitchEco [▶ 13]	Simplified version of the FB_Dimmer1Switch() without extra-functions. Needs less memory.
FB_Dimmer2Switch [▶ 15]	Light dimmer using two switches.
FB_Dimmer2SwitchEco [▶ 17]	Simplified version of the FB_Dimmer2Switch() without extra-functions. Needs less memory.
FB_Dimmer3Switch [▶ 18]	Combination of FB_Dimmer1Switch() und FB_Dimmer2Switch().
FB_Light [▶ 21]	Control of lighting.
FB_LightControl [▶ 22]	Daylight lamp control.
FB_ConstantLightControlEco [▶ 24]	Constant light control.
FB_Ramp [▶ 27]	Light-ramp.
FB_Sequencer [▶ 28]	Light-sequence.
FB_StairwellDimmer [▶ 32]	Stairwell light dimmer.
FB_StairwellLight [▶ 34]	Stairwell lighting.

Facade

Name	Description
FB_RoofWindow [▶ 34]	Control of roof-window.
FB_VenetianBlind [▶ 36]	Control of blinds.
FB_VenetianBlindEx [▶ 38]	Venetian blind control with direct position command.
FB_VenetianBlindEx1Switch [▶ 41]	Venetian blind control with direct position command with only one switch-input.

Scene Management

Name	Description
FB_RoomOperation [▶ 44]	Function block for calling and changing scenes via buttons. *)
FB_ScenesLighting [▶ 49]	Function block for managing lighting scenes. *)
FB_ScenesVenetianBlind [▶ 52]	Function block for managing blind scenes. *)

Signal Processing

Name	Description
FB_ShortLongClick [▶ 54]	Differentiation between short and long button presses.
FB_SignallingContact [▶ 55]	Signalling contact.
FB_SingleDoubleClick [▶ 56]	Differentiation between single and double button presses.
FB_ThresholdSwitch [▶ 57]	Threshold switch.

Filter Functions

Name	Description
FB_PT1 [▶ 58]	PT1-Filter for smoothing of input-values.
FB_PT2 [▶ 60]	PT2-Filter for smoothing of input-values.

Conversion Functions

Name	Description
F_Scale [▶ 63]	Scaling the / conversion from raw values to measured value
F_TO_C [▶ 63], F_TO_K [▶ 63], F_TO_R [▶ 63], K_TO_F [▶ 63], K_TO_C [▶ 63], K_TO_R [▶ 63], C_TO_F [▶ 63], C_TO_K [▶ 63], C_TO_R [▶ 63], R_TO_K [▶ 63], R_TO_C [▶ 63], R_TO_F [▶ 63]	Functions for converting temperatures between Kelvin, Celsius, Reaumur and Fahrenheit

Time Switches

Name	Description
FB_WeeklyTimeSwitch [▶ 75]	Weekly time switch.
FB_CalcSunPosition [▶ 77]	Calculating of sun height and sun sun azimuth.
FB_CalcSunriseSunset [▶ 78]	Calculation of sunrise and Sunset. *)
FB_CalcPublicHolidaysDE [▶ 80]	Calculation of german holidays.
FB_CalcPublicHolidaysUS [▶ 82]	Calculation of the United States public holidays.
FB_CalcFederalHolidaysUS [▶ 84]	Calculation of the United States federal holidays.
FB_DailyScheduler [▶ 66]	switches every n-th day.
FB_WeeklyScheduler [▶ 67]	switches every n-th week on specific days of the week.
FB_MonthlyScheduler1 [▶ 69]	switches in specific months on a specific weekday.
FB_MonthlyScheduler2 [▶ 70]	switches in specific months on a specific day of the month.
FB_YearlyScheduler [▶ 71]	switches on a specific day of the year.

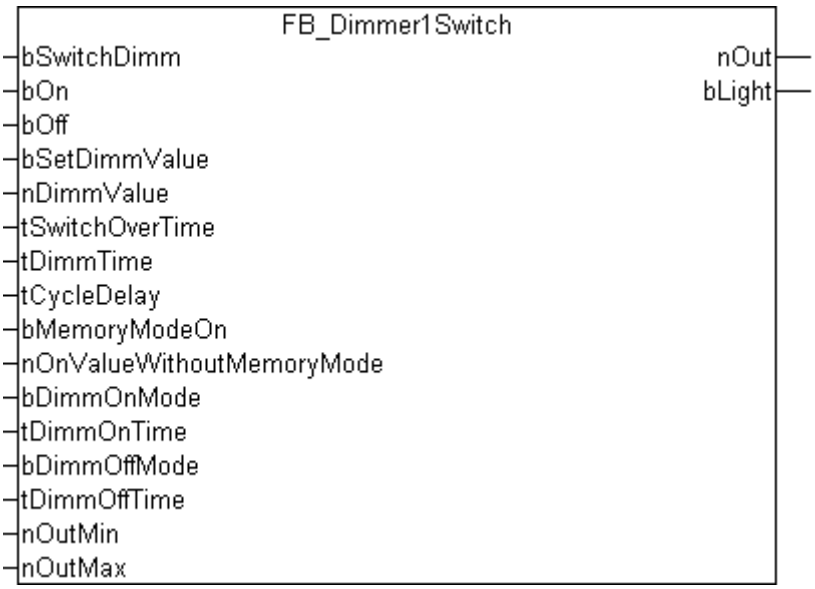
Energy Management

Name	Description
FB_MaximumDemandController [▶ 85]	Maximum Demand Controller in order to reduce power peaks.

*) **Note**: This functionblock is only available in the PC-based version of the library.

3.1 Lightings

3.1.1 FB_Dimmer1Switch



Description

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 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 rising edge is applied to the *bOn* or *bOff* inputs. This may, for instance, be used for global on/off functions. The output value is set to 0 when switching off. The switch-on behaviour 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 rising 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 rising edge of *bSetDimmValue*.
The *bSetDimmValue* and *nDimmValue* inputs can be used to implement a variety of lighting scenarios. Using *nDimmValue* to set the outputs directly 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 the *bSwitchDimm* input. 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 the *bSwitchDimm* input. 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 rising 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 tSwitchDimm input 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 bSwitchDimm input can only be used to switch the light on or off. In this case, the value of tSwitchOverTime is irrelevant.

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;
bDimmOffMode                : BOOL := FALSE;
tDimmOffTime                : TIME := t#0s;
nOutMin                     : UINT := 5000;
nOutMax                     : UINT := 32767;
```

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.

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 *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.

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: Value at switch on if the memory function is not active.

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

tDimmOnTime: Period over which the light is turned up when switching on. *bDimmOnMode* must be active.

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

tDimmOffTime: Period over which the light is turned down when switching 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.

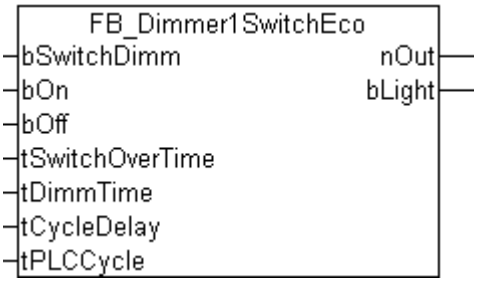
VAR_OUTPUT

```
nOut            : UINT;
bLight          : BOOL;
```

nOut: Analogue output-value.

bLight: Digital output-value. This bit is set if *nOut* is above 0.

3.1.2 FB_Dimmer1SwitchEco



Description

The function-block FB_Dimmer1SwitchEco is the memory-saving variation on the FB_Dimmer1Switch() [▶ 11]. It operates without the extra-functions "Set-Value" and "Memory-mode-off", which are not needed in many cases. In addition the values *nOutMin* and *nOutMax* of the FB_Dimmer1Switch() [▶ 11] are set to 0 and 32767 internally. The resulting output range is exactly the range of an analogue-output-terminal. The input *tPLCCycle* is very important for the calculation of increments per cycle for the output *nOut* . This method 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 then cycles 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 rising edge is applied to the *bOn* or *bOff* inputs. This may, for instance, be used for global on/off functions. The output value is set to 0 when switching off.

The memory function

Unlike the function-block FB_Dimmer1Switch() [▶ 11], which can operate with or without memory-function, this function is always activated in this variation. This means, that the light will, when turned on, always be set to the last on-level. How the light was turned on, either with the input *bSwitchDimm* or with the input *bOn* , doesn´t matter.

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

```
bSwitchDimm         : BOOL;
bOn                 : BOOL;
bOff                : BOOL;
tSwitchOverTime     : TIME := t#500ms;
tDimmTime           : TIME := t#5s;
tCycleDelay         : TIME := t#500ms;
tPLCCycle           : TIME := t#10ms;
```

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: PLC-Cycle time.

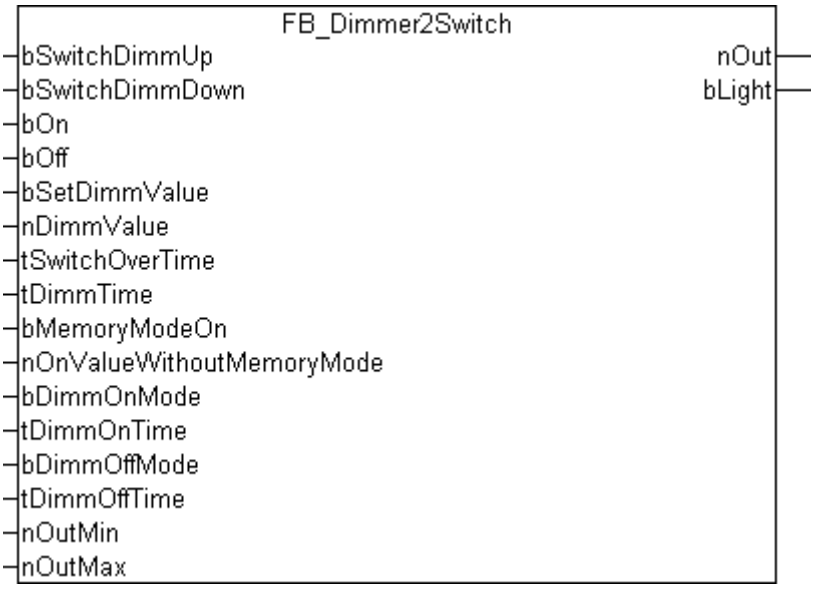
VAR_OUTPUT

```
nOut                : UINT;
bLight              : BOOL;
```

nOut: Analogue output-value.

bLight: Digital output-value. This bit is set if *nOut* is above 0.

3.1.3 FB_Dimmer2Switch



Description

The functions available in the Dimmer2Switch function block correspond closely to those in FB_Dimmer1Switch() [▶ 11]. The difference is simply that two switches are connected to the Dimmer2Switch function block. 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 rising edge is applied to the *bOn* or *bOff* inputs. This may, for instance, be used for global on/off functions. The output value is set to 0 when switching off. The switch-on behaviour 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 rising 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 rising edge of *bSetDimmValue*.
The *bSetDimmValue* and *nDimmValue* inputs can be used to implement a variety of lighting scenarios. Using *nDimmValue* to set the outputs directly can be used to achieve 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 rising 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

```
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;
```

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.

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: Value at switch on if the memory function is not active.

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

tDimmOnTime: Period over which the light is turned up when switching on. *bDimmOnMode* must be active.

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

tDimmOffTime: Period over which the light is turned down when switching 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.

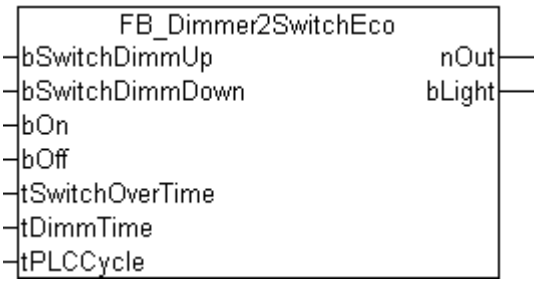
VAR_OUTPUT

```
nOut                : UINT;
bLight              : BOOL;
```

nOut: Analogue output-value.

bLight: Digital output-value. This bit is set if *nOut* is above 0.

3.1.4 FB_Dimmer2SwitchEco



Description

The function-block FB_Dimmer2SwitchEco is the memory-saving variation on the FB_Dimmer2Switch() [▶ 15]. It operates without the extra-functions "Set-Value" and "Memory-mode-off", which are not needed in many cases. In addition the values *nOutMin* and *nOutMax* of the FB_Dimmer2Switch() [▶ 15] are set to 0 and 32767 internally. The resulting output range is exactly the range of an analogue-output-terminal. The input *tPLCCycle* is very important for the calculation of increments per cycle for the output *nOut* . This method 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 rising edge is applied to the *bOn* or *bOff* inputs. This may, for instance, be used for global on/off functions. The output value is set to 0 when switching off.

The memory function

Unlike the function-block FB_Dimmer2Switch() [▶ 15], which can operate with or without memory-function, this function is always activated in this variation. This means, that the light will, when turned on, always be set to the last on-level. How the light was turned on, either with the input *bSwitchDimmUp* / *bSwitchDimmDown* or with the input *bOn* , doesn´t matter.

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: Sets the output to the last output value.

bOff: Sets 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.

tPLCCycle: PLC-Cycletime.

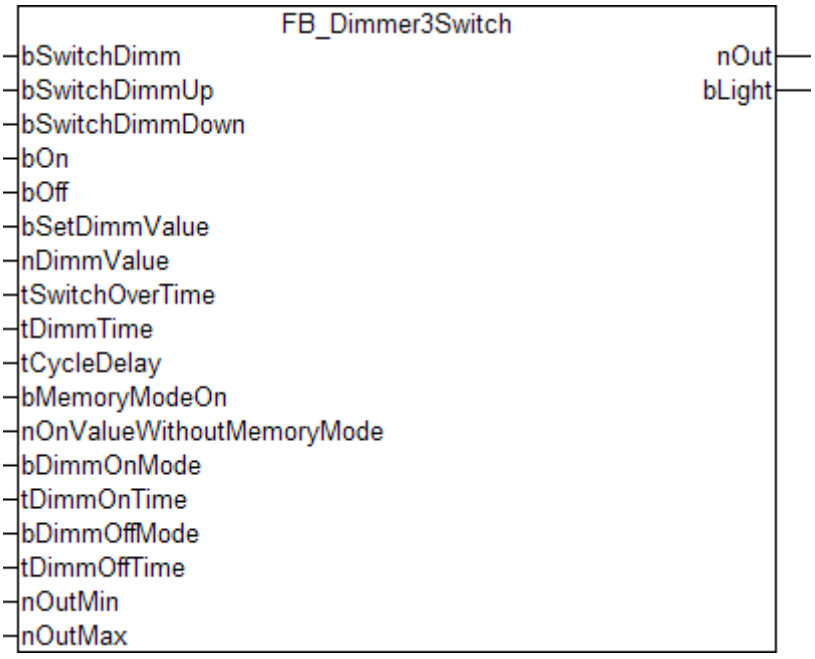
VAR_OUTPUT

```
nOut              : UINT;
bLight            : BOOL;
```

nOut: Analogue output-value.

bLight: Digital output-value. This bit is set if *nOut* is above 0.

3.1.5 FB_Dimmer3Switch



Description

The functions available in the FB_Dimmer3Switch function block correspond closely to those in FB_Dimmer1Switch() [▶ 11] and FB_Dimmer2Switch() [▶ 15].

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 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 rising edge is applied to the *bOn* or *bOff* inputs. This may, for instance, be used for global on/off functions. The output value is set to 0 when switching off. The switch-on behaviour 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 rising 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 rising edge of *bSetDimmValue*.
The *bSetDimmValue* and *nDimmValue* inputs can be used to implement a variety of lighting scenarios. Using *nDimmValue* to set the outputs directly 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 rising 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: Value at switch on if the memory function is not active.

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

tDimmOnTime: Period over which the light is turned up when switching on. *bDimmOnMode* must be active.

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

tDimmOffTime: Period over which the light is turned down when switching 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: Analogue output-value.

bLight: Digital output-value. This bit is set if *nOut* is above 0.

3.1.6 FB_Light

A rising edge at the *bOn* input sets the *bLight* output. The output is reset by a rising edge at the *bOff* input. If a rising edge is presented to *bToggle*, the output is negated; i.e., if On it goes Off, and if Off it goes On.

VAR_INPUT

```
bOn             : BOOL;
bOff            : BOOL;
bToggle         : BOOL;
```

bOn: Switches the output on.

bOff: Switches the output off.

bToggle: Negates the state of the output.

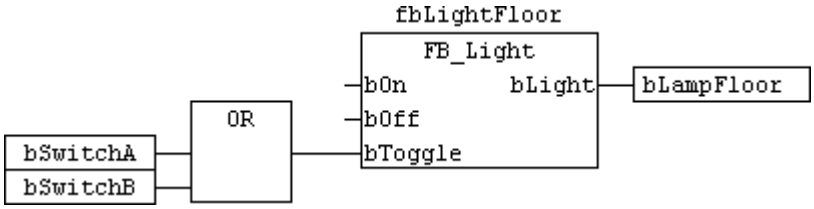
VAR_OUTPUT

```
bLight          : BOOL;
```

bLight: A rising edge at the *bOn* input sets the output.

Example 1

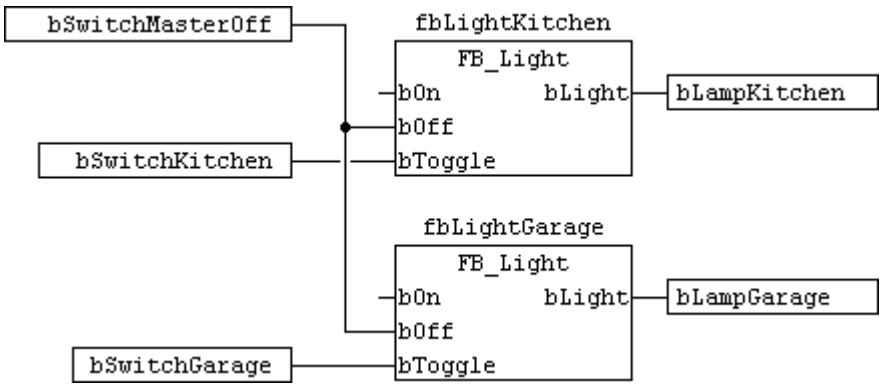
In the following example a light is operated by two switches.



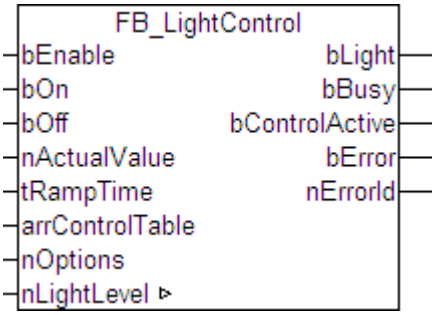
If either the *bSwitchA* or the *bSwitchB* button is pressed, then the state of the light, as represented by the *bLight* output, is changed.

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.

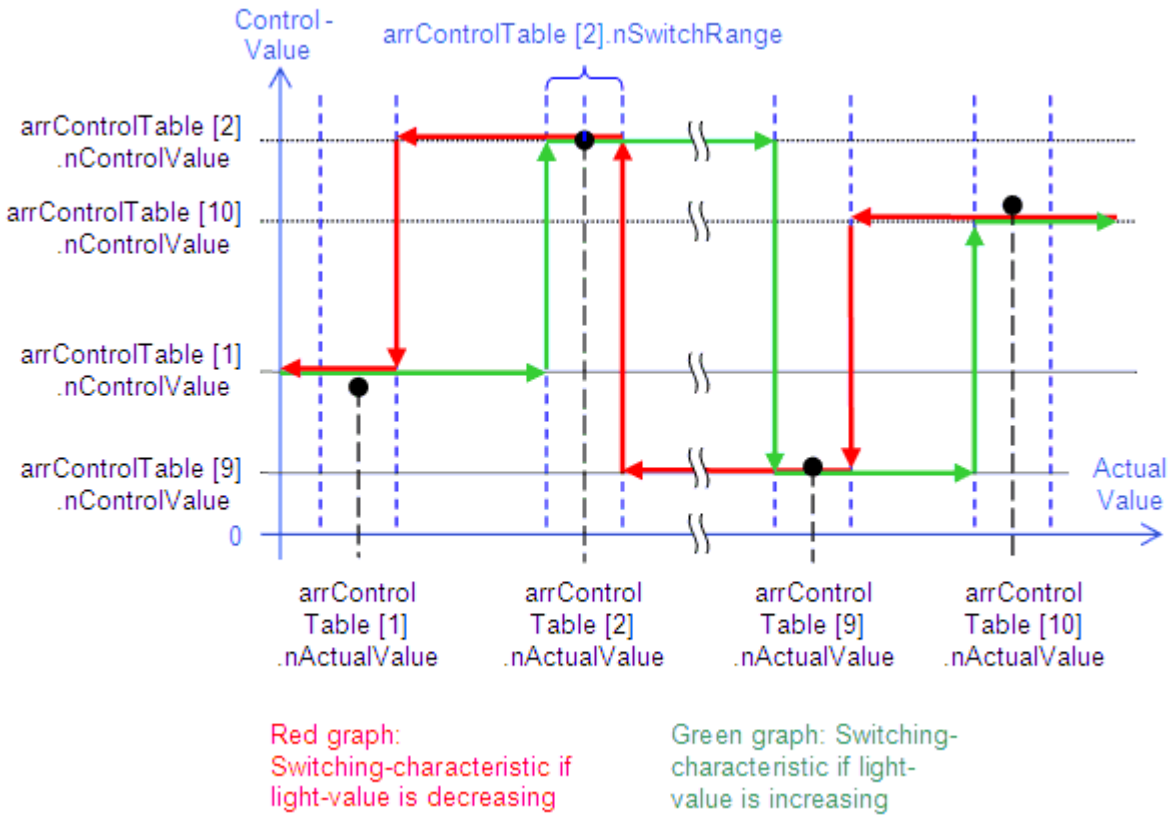


3.1.7 FB_LightControl



Daylight-Lamp-control.

This function-block is based upon a table of 30 nodes containing actual- and control-values for threshold-switching. If the actual value comes within the range of a new node (*arrControlTable[n].nActualValue - arrControlTable[n].nSwitchRange/2 ... arrControlTable[n].Input + arrControlTable[n].nSwitchRange/2*), the control-value will change (see diagram). The threshold-switch is followed by a ramp-function which ramps the light-level to the new control-value over the time *tRampTime.*With a rising-edge at *bOn* the light is switched immediatelyto the nearest control-value and similarily arising edge at *bOff* switches the light off,without the delay of a ramp. It is possible to trigger a positive edge on *bOn* or bOff *at* anytime.



It is not required to use all 30 entries in the node table. The first element with a switch-range of "*0*" will mark the beginning of the unused table-range.

VAR_INPUT

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

bEnable: A positive input enables the function block. A negative state deactivates the inputs and sets the function-block to the idle-mode.

bOn: A rising edge sets the output *nLightLevel* directly to the next control-value.

bOff: A rising edge sets the output *nLightLevel* immediately to "*0*".

nActualValue: measured light-value.

tRampTime: time to drive the lamp from the actual light-level to the new control-value. (Preset value: 30s).

arrControlTable: Actual-/control-value-table. *arrControlTable[1]* to *arrControlTable[30]* of ST_ControlTable

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

nActualValue: Measured light value.

nControlValue: Control value of a node.

nSwitchRange: Switching range around the node. *nSwitchRange* can only be "*0*" for the first node of the unused table-range.

nOptions: Reserved for future developments.

VAR_OUTPUT

```
bLight              : BOOL;
bBusy               : BOOL;
bControlActiv       : BOOL;
bError              : BOOL;
nErrorId            : UDINT;
```

bLight: If *nLightLevel* is greater than 0, this output is set to *TRUE*.

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bControlActive: If the control loop is active, this output is set to *TRUE*.

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

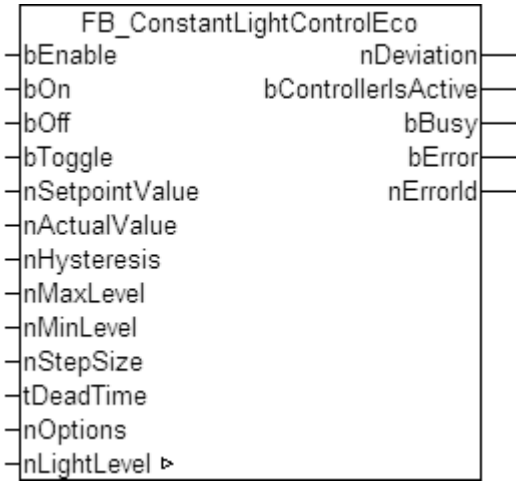
nErrorId: Contains the command-specific error code. *nErrorId* is reset to 0 by the execution of an instruction at the inputs. See Error codes [▶ 89].

VAR_IN_OUT

```
nLightLevel         : UINT;
```

nLightLevel: Reference to the actual light-level-output.

3.1.8 FB_ConstantLightControlEco



The function block FB_ConstantLightControlEco() 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 (*tDeadTime*) and the step size (*nStepSize*). 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 (*nHysteresis*) prevents continuous oscillation around the setpoint. If the actual value is within the hysteresis range around the setpoint, the lamps brightness remains unchanged.

If the set step size nStepSize is too large or the hysteresis nHysteresis is too small, the hysteresis range may be "missed". This cannot be prevented by the function block, because the light output nLightLevel is only physically linked to the recorded actual light value, nActualLevel.

VAR_INPUT

```
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;
```

bEnable: Enables the function-block. If this input is FALSE, the inputs *bOn*, *bOff* and *bToogle* are disabled. The control values *nLightLevel* remains unchanged.

bOn: Switches the addressed devices to *nMaxLevel* and activates constant light control. Note: an activated but disabled (*bEnable* = FALSE) function block will automatically resume its functionality, when it´s enabled again.

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 set value.

nActualValue: The actual value is applied at this input.

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

nMaxLevel: Maximum limit of the control value *nLightLevel*.

nMinLevel: Minimum limit of the control value *nLightLevel*. If the light-control requires to dim below this level, *nLightLevel* is set to "0". The other way around, if *nLightLevel* is "0", while the control is active, dimming up means setting the lamp to this value first.

nStepSize: Step-Size, by which the control-value *nLightLevel* is increased/decreased every active dimming-step.

tDeadTime: Dead time between the individual steps dimming up/down the light.

nOptions: Without functionality. Reserved for future developments.

VAR_OUTPUT

```
nDeviation           : INT;
bControllerIsActive  : BOOL;
bBusy                : BOOL;
bError               : BOOL;
nErrorId             : UDINT;
```

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

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

bBusy: When the control is activated, this output is always set, when the control-value *nLightLevel* changes.

bError: This output is switched to TRUE if 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 an instruction at the inputs.

nErrorId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of an instruction at the inputs. See Error codes [▶ 89].

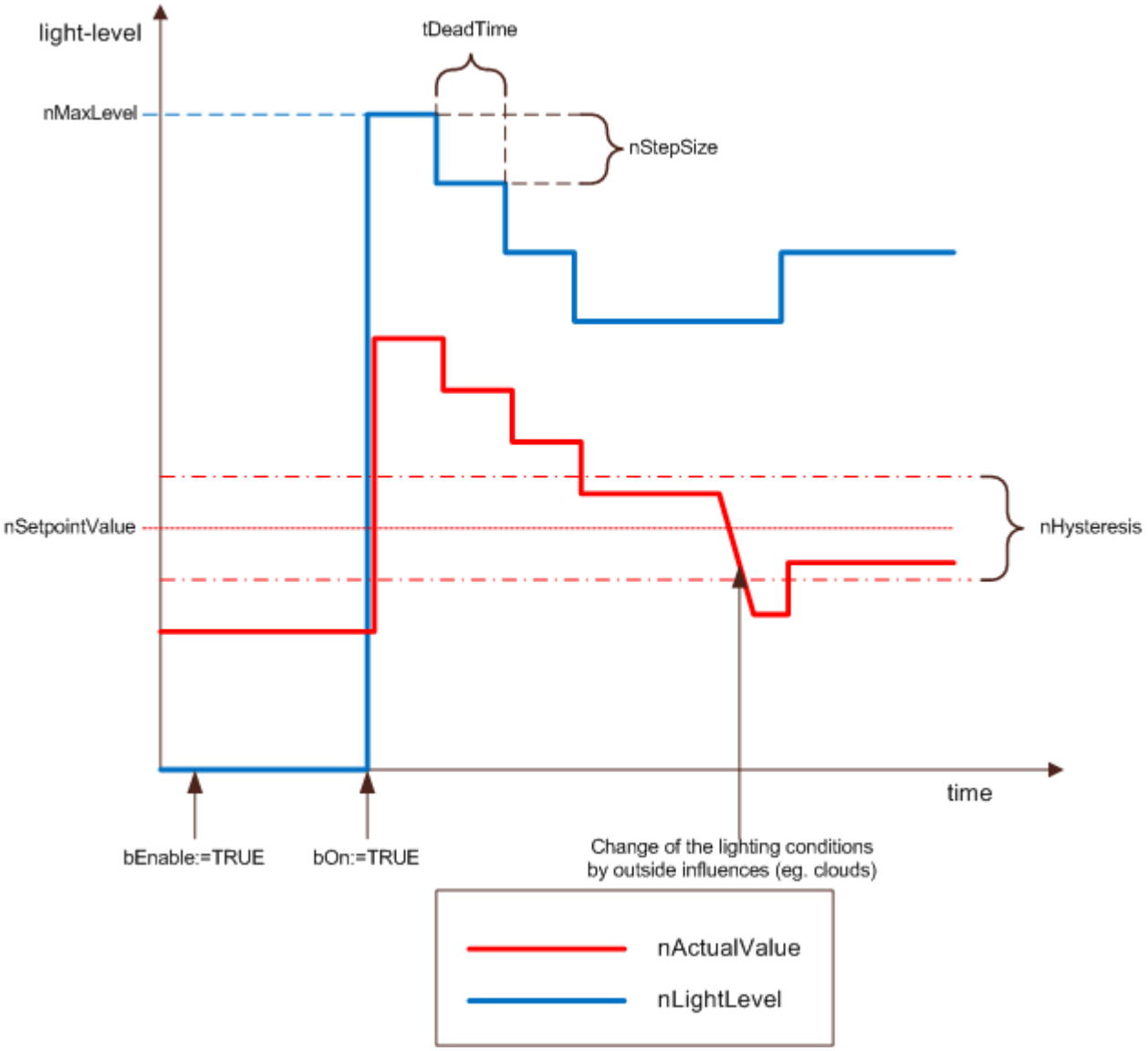
VAR_IN_OUT

```
nLightLevel        : UINT;
```

nLightLevel: Reference to the actual light-level-output. This output has to be an IN-OUT-Variable, because the function-block demands a read-/write-access.

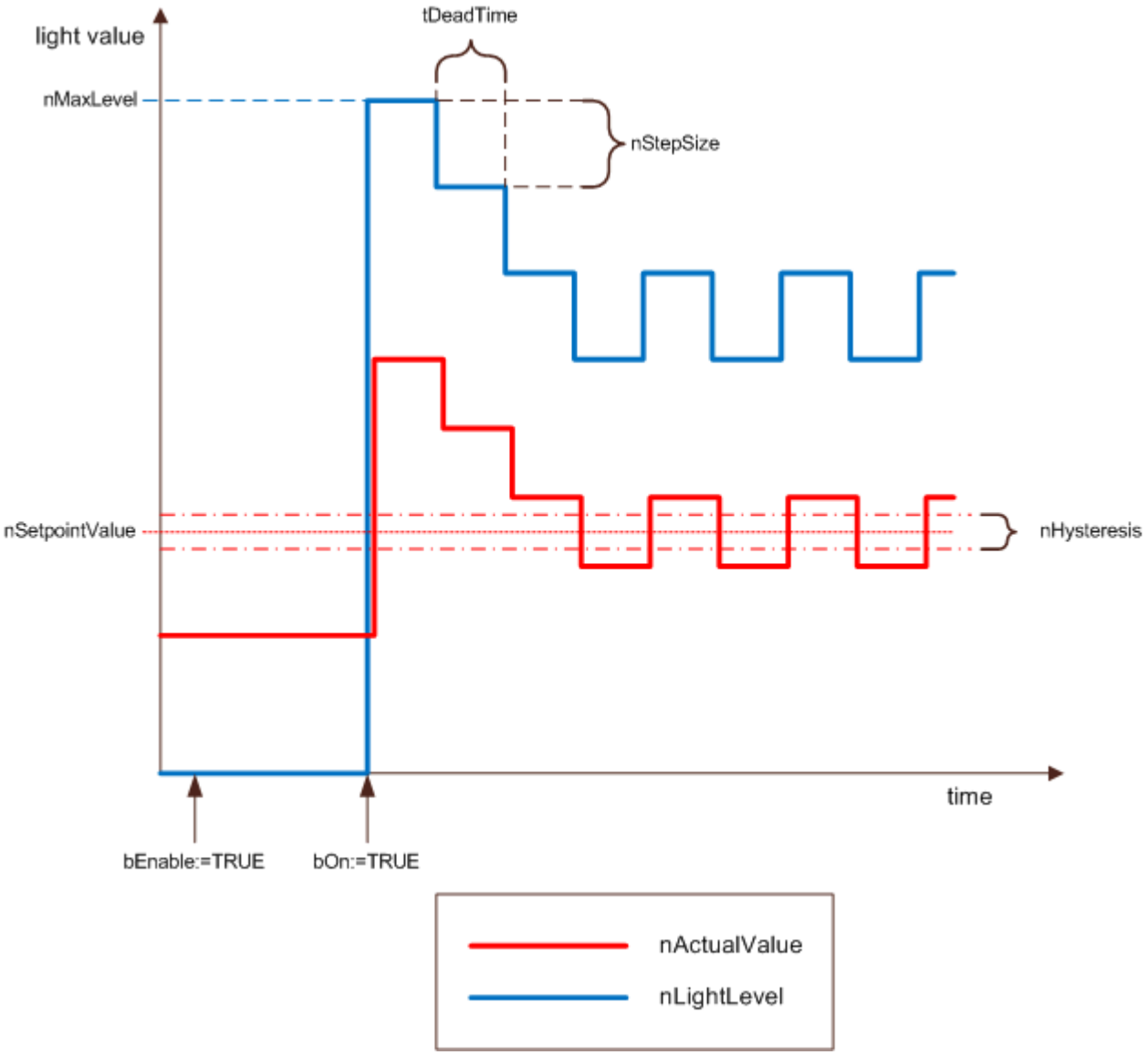
Operation diagram

The following diagram should make it clear, how the control works in normal operation:



First of all, the control will be enabled with a TRUE-signal at the input *bEnable*. Then, with a rising trigger at *bOn* the light-level (*nLightLevel*) will be set to its maximum-value. This has an influence on the surrounding light, measured by *nActualValue,* as well. With the actual light-level rising above the setpoint-value, the light-level at the output of the control has to be recuced; *nLightLevel* is now decreased step by step until the measured value *nActualValue* is within the hysteresis-range (*nSetpointValue* - 0.5**nHysteresis* < x < *nSetpointValue* + 0.5**nHysteresis*).
If the measured light-value decreases eg. by outside influences, the control will increase the light level (*nLightLevel*) until *nActualValue* is witlin the hysteresis-range again.

If the Step-size (nStepSize) is too big or the hysteresis too small, the control-value nLightLevel may oscillate around the setpoint-value. The following diagram shows that the actual-value, influenced by the control-value, will always miss the hysteresis-range.



3.1.9 FB_Ramp

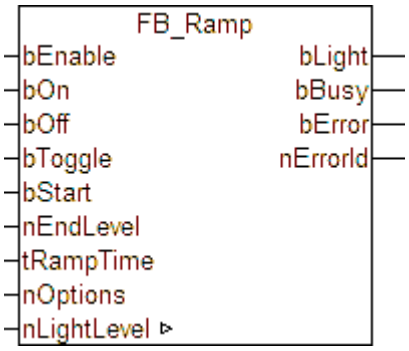


Fig. 1: FB_Ramp

Function-block creating a light-ramp.

With a rising-edge at *bOn* the light will be switched immediately to the maximum-level (*32767*) and a rising edge at *bOff* turns the light off. Triggering the input *bToggle* inverts the actual light-state. A rising-edge at *bStart* starts dimming the light from the actual to the end-level (*nEndLevel*) - the required time is defined by *tRampTime*. As long as *bEnable* is *TRUE* all inputs are active, otherwise the controlling inputs *bOn*, *bOff*, *bToggle* and *bStart* are deactivated and the function-block turns to its idle-mode.

VAR_INPUT

```
bEnable         : BOOL;
bOn             : BOOL;
bOff            : BOOL;
bToggle         : BOOL;
bStart          : BOOL;
nEndLevel       : BYTE;
tRampTime       : TIME := t#10s;
nOptions        : DWORD;
```

bEnable: A positive input enables the function block. A negative state deactivates the inputs and sets the function-block to the idle-mode.

bOn: A rising edge sets the output *nLightLevel* directly to the maximum-level (*32767*).

bOff: A rising edge sets the output *nLightLevel* immediately to "*0*".

bToggle: Rising edges at this input toggle the *nLightLevel* between "*0*" and "*32767*".

bStart: This input starts the dim-ramp from the actual value to *nEndLevel* within the time defined as *tRampTime*. This can be interrupted by *bOn*, *bOff* or *bToggle* at any time.

nEndLevel: Target-value of the dim-ramp.

tRampTime: Ramp-time, see *bStart*. (Initial value: 10s).

nOptions: Reserved for future developments.

VAR_OUTPUT

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

bLight: As long as *nLightLevel* is greater than "*0*", this output is set to *TRUE*.

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

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

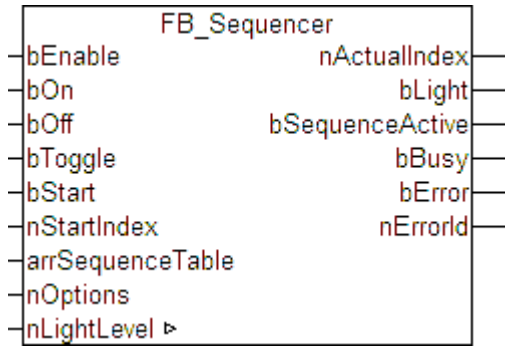
nErrorId: Contains the command-specific error code of the most recently executed command. *nErrorId* is reset to 0 by the execution of an instruction at the inputs. See Error codes [▶ 89].

VAR_IN_OUT

```
nLightLevel         : UINT;
```

nLightLevel: Reference to the actual light-level-output. This output has to be an IN-OUT-Variable because the function-block may need the actual level as a start-level for a ramp.

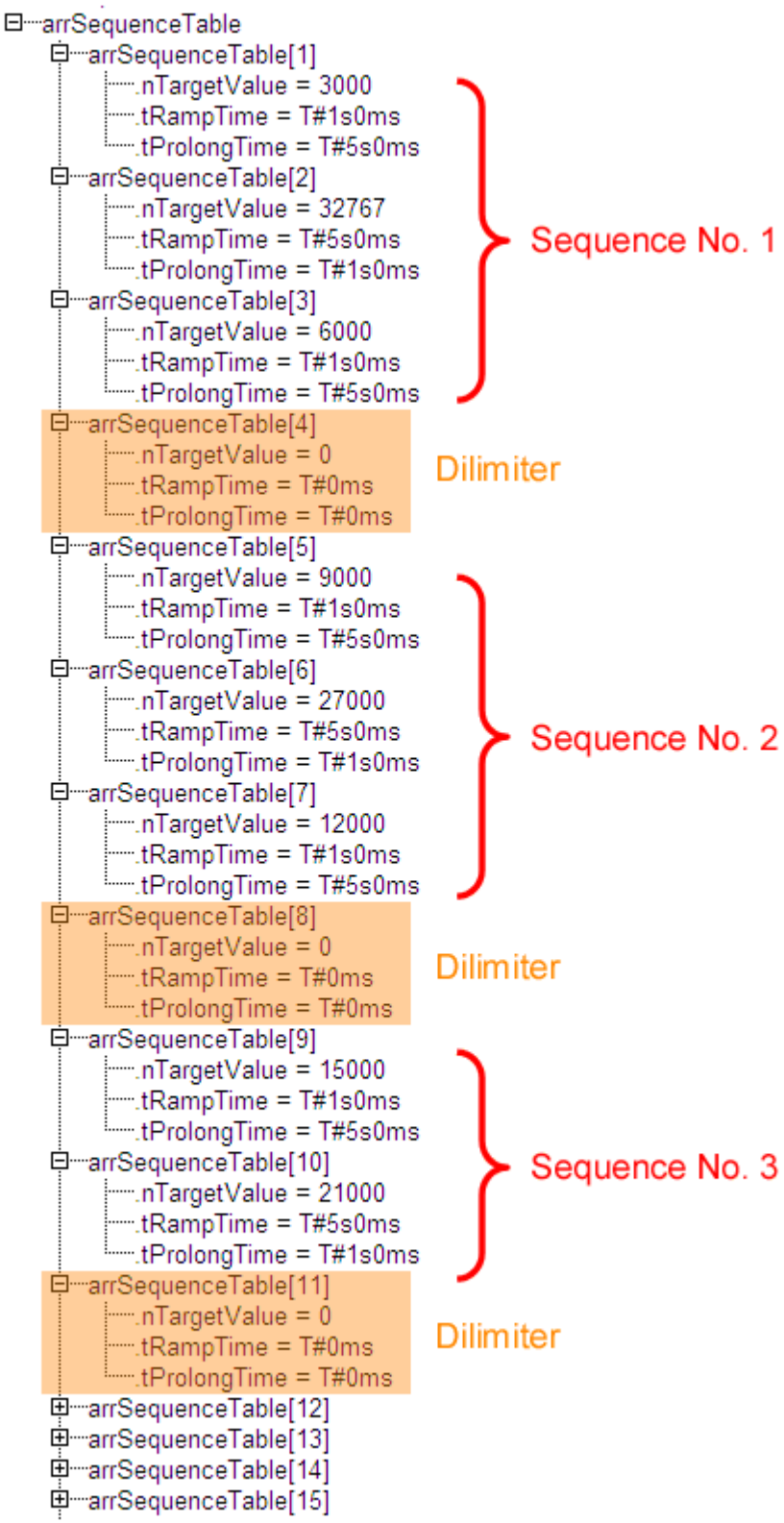
3.1.10 FB_Sequencer



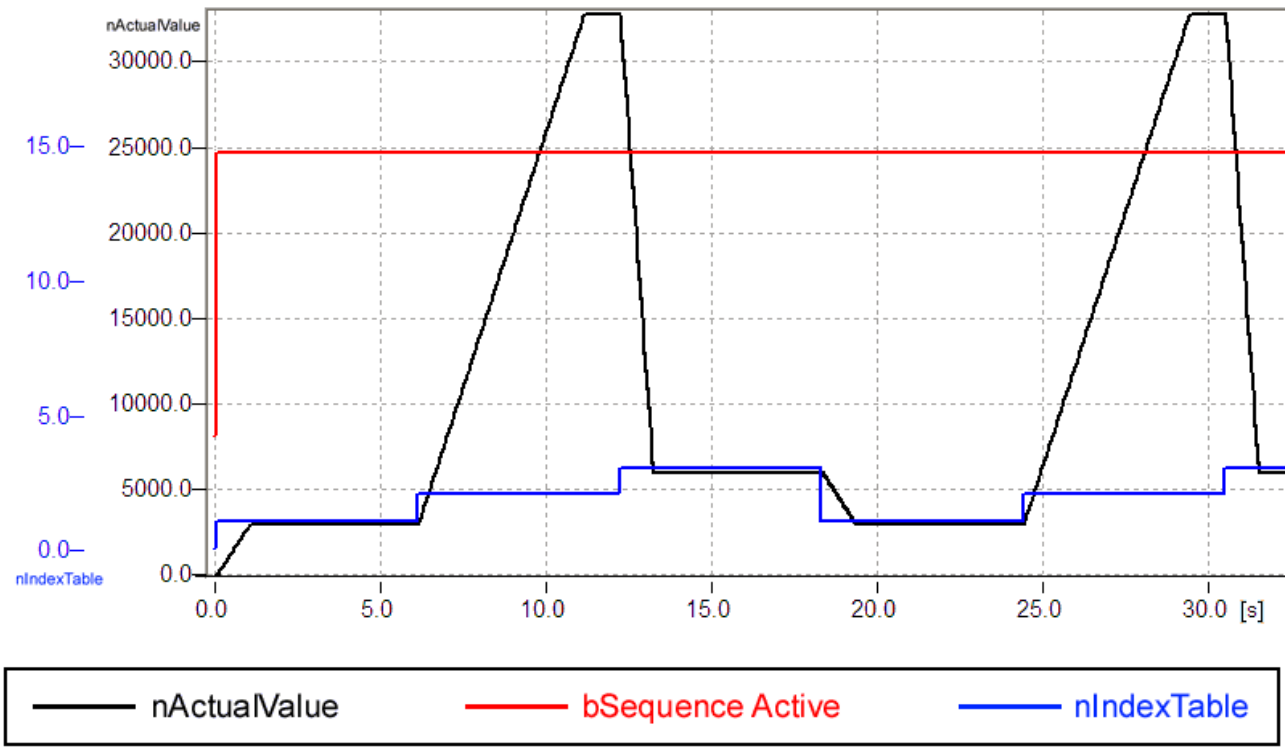
Function-block to program light-sequences with up to 50 different steps.

This function-block is based on a ramp-function, which drives the output to a target-value (*nTargetValue*)i n a specified time (*tRampTime*) . When the target-value is reached, the light will stay at this level for a specified time (*tProlongTime*). Once *tProlongTime* has elapsed, the light will ramp to the next target value.

The Sequence table consists of 50 elements containing the target-value (*nTargetValue*), the ramp-time (*tRampTime*) and the time to hold a reached light-level (*tProlongTime*). It is not required to use all 50 elements of the table. An element containing only zeros for the target-value, ramp-time and prolong-time will be recognized as the end of a sequence. Furthermore it is possible to let a sequence start at a specific position with the input-value, *nStartIndex*. With *nStartIndex* it is possible to program different light-sequences within the 50 elements of the table which are separated by simple zero-elements (delimiters).



The light-level programmed with sequence No.1, for example, will show the following behaviour (*nStartIndex=1, nOptions.bit0=TRUE*, explanations see below):



The function-block has inputs to switch the light on and off (on: *nLightLevel* = 32767, off: *nLightLevel* = 0) as well as an input *bToggle* to invert the actual light-state. All inputs are only read by the function-block, if *bEnable* is set to *TRUE.* If *bEnable* is reset to *FALSE,* all inputs are inactive and *nLightLevel* will remain on its actual value.

VAR_INPUT

```
bEnable             : BOOL;
bOn                 : BOOL;
bOff                : BOOL;
bToggle             : BOOL;
bStart              : BOOL;
nStartIndex         : USINT;
arrSequenceTable    : ARRAY[1..50] OF ST_SequenceTable;
nOptions            : DWORD;
```

bEnable: A positive input enables the function block. A negative state deactivates the inputs and sets the function-block to the idle-mode.

bOn: A rising edge sets the output *nLightLevel* directly to the maximum-level (*32767*).

bOff: A rising edge sets the output *nLightLevel* immediately to "*0*".

bToggle: Rising edges at this input toggle the *nLightLevel* between "*0*" and "*32767*".

bStart: This input lets the sequence begin with the element defined with *nStartIndex*.

nStartIndex: See *bStart.*

arrSequenceTable: Light-value-table with the information about the target-value, the ramp-time and the prolong-time.

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: Time to stay on the target-value.

nOptions: Parameter-input. Setting (resp. not-setting) of the single bits will affect the behaviour of the function-block as follows:

Constant	Description
OPTION_INFINITE_LOOP	After running through a sequence, the function-block will automatically restart at the element defined with *nStartIndex*. If this option is not set the function-block will stop after running through the sequence. In order to start again, a rising edge at *bStart* is neccessary.

VAR_OUTPUT

```
nActualIndex        : USINT;
bLight              : BOOL;
bSequenceActive     : BOOL;
bBusy               : BOOL;
bError              : BOOL;
nErrorId            : UDINT;
```

nActualIndex: This output shows the actual element of the light-sequence. If the sequence is finished or stopped (*bSequenceActive* = *FALSE* , see below) , the output will fall back to "0".

bLight: As long as *nLightLevel* is greater than "*0*", this output is set to *TRUE*.

bSequenceActive: If a light-sequence is running, this output is set to *TRUE*.

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

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

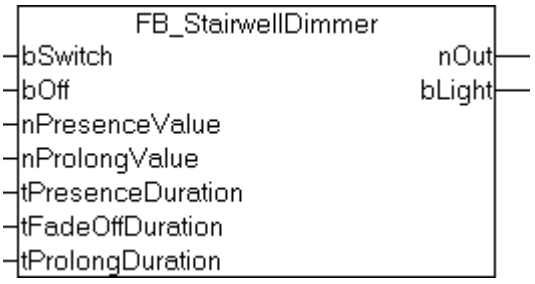
nErrorId: Contains the command-specific error code of the most recently executed command. *nErrorId* is reset to 0 by the execution of an instruction at the inputs. See Error codes [▶ 89].

VAR_IN_OUT

```
nLightLevel         : UINT;
```

nLightLevel: Reference to the actual light-level-output.

3.1.11 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 rising 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.

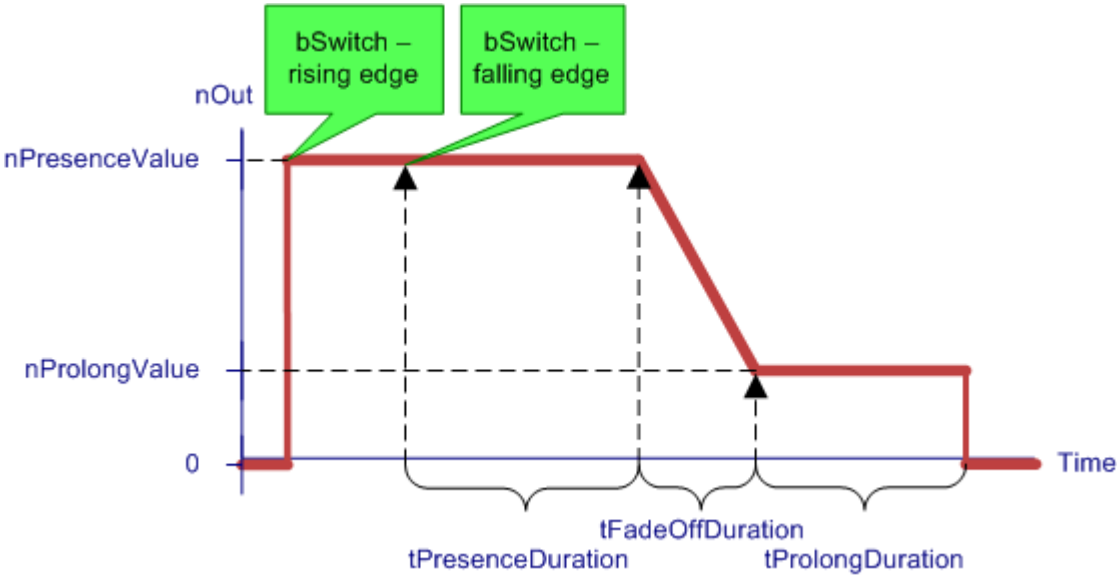


Fig. 2: FB_StairwellDimmer-Time-nOut

VAR_INPUT

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

bSwitch: Upon a rising edge: *nOut* is set to *nPresenceValue*. Upon a falling 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).

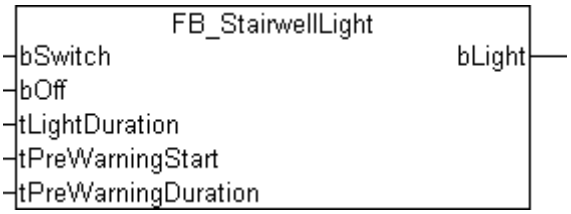
VAR_OUTPUT

```
nOut        : UINT;
bLight      : BOOL;
```

nOut: Output of the momentary light value.

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

3.1.12 FB_StairwellLight



A rising edge at the *bSwitch* input sets the *bLight* output. The output is reset again once *tLightDuration* 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 rising edge at the *bOff* input switches the output off immediately.

VAR_INPUT

```
bSwitch             : BOOL;
bOff                : BOOL;
tLightDuration      : TIME := t#120s;
tPreWarningStart    : TIME := t#110s;
tPreWarningDuration : TIME := t#500ms;
```

bSwitch: Switches the output on for the period of time given by *tLightDuration*.

bOff: Switches the output off.

tLightDuration: Period for which the output is set.

tPreWarningStart: Warning time.

tPreWarningDuration: Duration of the prewarning.

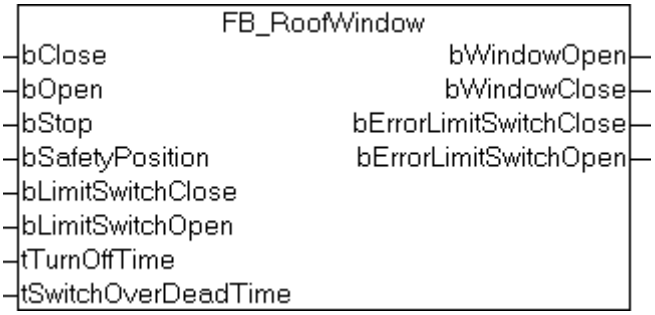
VAR_OUTPUT

```
bLight          : BOOL;
```

bLight: A rising edge at the *bSwitch* input sets the output for the duration of *tLightDuration*.

3.2 Facade

3.2.1 FB_RoofWindow



Description

A rising edge at the *bClose* or *bOpen* inputs set the *bWindowClose* or *bWindowOpen* outputs respectively. These remain asserted until the time *tTurnOffTime* has elapsed, or until the block receives some other command. Both outputs are immediately reset by a positive edge at the *bStop* input.

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 seconds and 1.0 seconds. The drive manufacturer can give you a precise value.

Safety position

Travel to the safety position (e.g. because there is a strong wind) can be achieved by setting the *bSafetyPosition* input. The output *bWindowClose* is set and the output *bWindowOpen* reset for the period specified by *tTurnOffTime*. Operation of the window is prevented for as long as the *bSafetyPosition* input is active.

VAR_INPUT

```
bClose               : BOOL;
bOpen                : BOOL;
bStop                : BOOL;
bSafetyPosition      : BOOL;
bLimitSwitchClose    : BOOL;
bLimitSwitchOpen     : BOOL;
tTurnOffTime         : TIME := t#60s;
tSwitchOverDeadTime  : TIME := t#400ms;
```

bClose: Set the *bWindowClose* output and reset the *bWindowOpen* output. The *bWindowClose* output remains latched.

bOpen: Set the *bWindowOpen* output and reset the *bWindowClose* output. The *bWindowOpen* output remains latched.

bStop: Reset the *bWindowClose* and *bWindowOpen* outputs.

bSafetyPosition: The safety position is approached. To do this, the window is closed for the period specified by *tTurnOffTime*. It is not possible to operate the window while this input is set.

bLimitSwitchClose: Optional limit switch. If *bClose* is set and *bLimitSwitchClose* is not set for the period specified by *tTurnOffTime*, *bErrorLimitSwitchClose* will be set.

bLimitSwitchOpen: Optional limit switch. If *bOpen* is set and *bLimitSwitchOpen* is not set for the period specified by *tTurnOffTime*, *bErrorLimitSwitchOpen* will be 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

```
bWindowOpen            : BOOL;
bWindowClose           : BOOL;
bErrorLimitSwitchClose : BOOL;
bErrorLimitSwitchOpen  : BOOL;
```

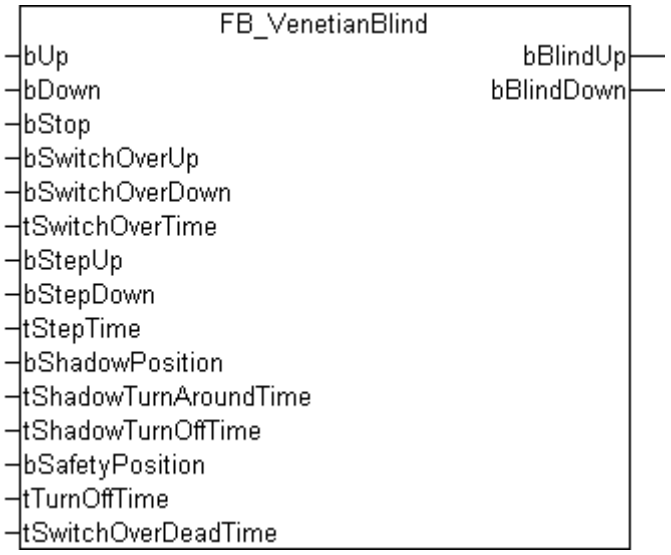
bWindowOpen: The window opens.

bWindowClose: The window closes.

bErrorLimitSwitchClose: Error of optional limit switch while closing.

bErrorLimitSwitchOpen: Error of optional limit switch while opening.

3.2.2 FB_VenetianBlind



Description

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

- A rising edge at the *bUp* or *bDown* inputs set the *bBlindUp* or *bBlindDown* outputs respectively. These remain asserted until the time *tTurnOffTime* has elapsed, or until the 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 rising 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 seconds and 1.0 seconds. 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 beat moved to the shading position. After presenting a rising edge to the *bShadowPosition* input, the blinds are lowered for the period of time specified by *tShadowTurnOffTime*. The blinds are then taken up 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. A pause of *tSwitchOverDeadTime* is maintained at the change of direction. Travel to the shading position can be interrupted at any time by a new command.

VAR_INPUT

```
bUp                   : BOOL;
bDown                 : BOOL;
bStop                 : BOOL;
bSwitchOverUp         : BOOL;
bSwitchOverDown       : BOOL;
tSwitchOverTime       : TIME := t#500ms;
bStepUp               : BOOL;
bStepDown             : BOOL;
tStepTime             : TIME:= t#200ms;
bShadowPosition       : BOOL;
tShadowTurnAroundTime : TIME := t#0s;
tShadowTurnOffTime    : TIME := t#20s;
bSafetyPosition       : BOOL;
tTurnOffTime          : TIME := t#60s;
tSwitchOverDeadTime   : TIME := t#400ms;
```

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. The outputs are not set if the specified duration is 0.

bShadowPosition: The shading position is approached (see below).

tShadowTurnAroundTime: The blind travels in the opposite direction for the period of time specified by *tShadowTurnAroundTime* after the shading position has been reached. A time of greater than 0 is necessary for the shading position to be approached.

tShadowTurnOffTime: The time for which the *bBlindDown* output is set in order to reach the shading position.

bSafetyPosition: The safety position is approached. To do this, the blind is raised for the period 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.

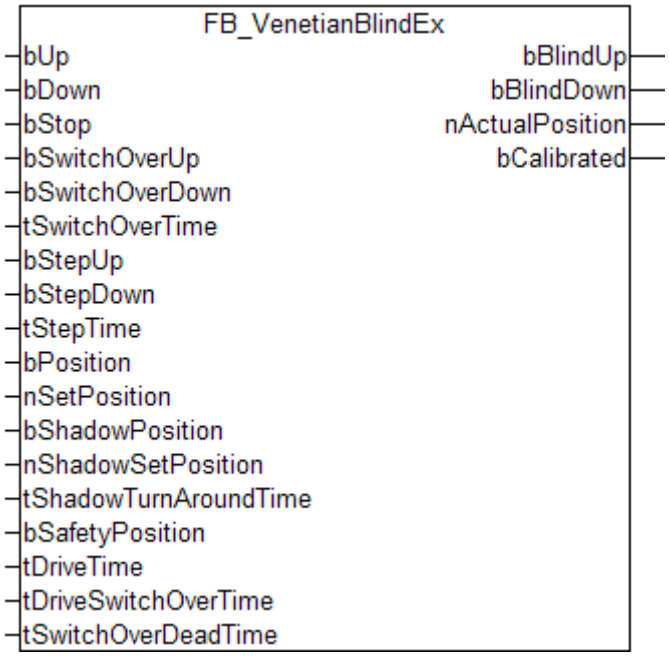
VAR_OUTPUT

```
bBlindUp          : BOOL;
bBlindDown        : BOOL;
```

bBlindUp: The blind drives up.

bBlindDown: The blind drives down.

3.2.3 FB_VenetianBlindEx



Description

Four different methods are available for controlling the blind:

- A rising edge at the *bUp* or *bDown* inputs set the *bBlindUp* or *bBlindDown* outputs respectively. These remain asserted until the time *tDriveTime* + 10% has elapsed, or until the 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 rising 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() [▶ 36] block, this block also enables movement to an absolute position. A percentage value is applied to input *nSetPosition*, and subsequently a rising 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 seconds and 1.0 seconds. 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 is therefore set as if the blind had moved down to darken.
When changing direction, a pause of the duration *tSwitchOverDeadTime* is observed. 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

In most cases a blind will not provide feedback about its current position. Therefore, this can only be calculated via the travel time. The accuracy 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, certain blind parameters have to be determined. The first one is the travel time, i.e. the time required for the blind to travel the complete distance. 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 duration is transferred to the parameter *tDriveTime*, the duration of a change of direction to *tDriveSwitchOverTime*.

Referencing a block

Since the current position of the blind has to be calculated, inaccuracies during operation will accumulate. In order to limit deviations, the block will automatically reference itself as often as possible. This occurs when the blind is moved either fully up or fully down, and the appropriate output is reset automatically, i.e. after the time *tDriveTime* + 10% has passed.

VAR_INPUT

```
bUp                   : BOOL;
bDown                 : BOOL;
bStop                 : BOOL;
bSwitchOverUp         : BOOL;
bSwitchOverDown       : BOOL;
tSwitchOverTime       : TIME := t#500ms;
bStepUp               : BOOL;
bStepDown             : BOOL;
tStepTime             : TIME := t#200ms;
bPosition             : BOOL;
nSetPosition          : USINT;
bShadowPosition       : BOOL;
nShadowSetPosition    : USINT := 80;
tShadowTurnAroundTime : TIME := t#0s;
bSafetyPosition       : BOOL;
tDriveTime            : TIME := t#60s;
tDriveSwitchOverTime  : TIME := t#200ms;
tSwitchOverDeadTime   : TIME := t#400ms;
```

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. 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 rising edge has been applied to input *bPosition*. 0% corresponds to fully up, 100% corresponds to fully down.

bShadowPosition: The shading position is approached (see below).

nShadowSetPosition: Shading position (0%-100%) to which the blind is to be moved, after a rising 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 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;
```

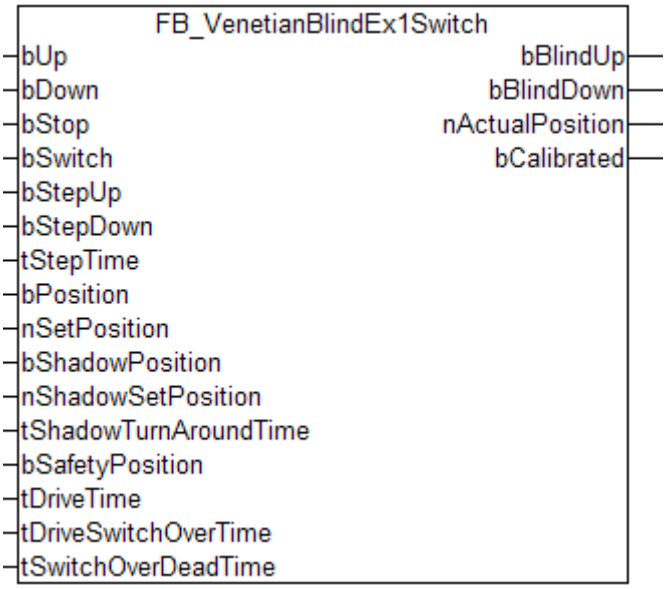
bBlindUp: The blind moves up.

bBlindDown: The blind moves down.

nActualPosition: Current position in percent.

bCalibrated: Specifies whether the blind is calibrated.

3.2.4 FB_VenetianBlindEx1Switch



Description

Function-block with the same functionality as FB_VenetianBlindEx() [▶ 38] but with only one input *bSwitch* to operate the blind.
Four different methods are available for controlling the blind:

- A rising edge at the *bUp* or *bDown* inputs set the *bBlindUp* or *bBlindDown* outputs respectively. These remain asserted until the time *tDriveTime* + 10% has elapsed, or until the block receives some other command. Both outputs are immediately reset by a positive edge at the *bStop* input.
- With the input *bSwitch,* which is normally connected to a pushbutton, it is possible to drive the blind up and down. In contrast to FB_VenetianBlindEx() [▶ 38] the specific output will be latched **immediately**. If operated in short succession, the functionality of *bSwich* will change from driving to stopping to driving in the opposite direction to stopping and to driving again. 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 rising 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() [▶ 36] block, this block also enables movement to an absolute position. A percentage value is applied to input *nSetPosition*, and subsequently a rising 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 seconds and 1.0 seconds. 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 rising edge to the input *bShadowPosition*, the blind is moved to the position *nShadowSetPosition*. The blinds are then taken up again for the period 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 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

In most cases a blind will not provide feedback about its current position. Therefore, this can only be calculated via the travel time. The accuracy 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, certain blind parameters have to be determined. The first one is the travel time, i.e., the time required for the blind to travel the complete distance. 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 duration is transferred to the parameter *tDriveTime*, the duration of a change of direction to *tDriveSwitchOverTime*.

Referencing a block

Since the current position of the blind has to be calculated, inaccuracies during operation will accumulate. In order to limit deviations, the block will automatically reference itself as often as possible. This occurs when the blind is moved either fully up or fully down, and the appropriate output is reset automatically, i.e. after the time *tDriveTime* + 10% has passed.

VAR_INPUT

```
bUp                   : BOOL;
bDown                 : BOOL;
bStop                 : BOOL;
bSwitch               : BOOL;
bStepUp               : BOOL;
bStepDown             : BOOL;
tStepTime             : TIME := t#200ms;
bPosition             : BOOL;
nSetPosition          : USINT;
bShadowPosition       : BOOL;
nShadowSetPosition    : USINT := 80;
tShadowTurnAroundTime : TIME := t#0s;
bSafetyPosition       : BOOL;
tDriveTime            : TIME := t#60s;
tDriveSwitchOverTime  : TIME := t#200ms;
tSwitchOverDeadTime   : TIME := t#400ms;
```

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: Control input to drive the blind up and down. In contrast to FB_VenetianBlindEx() [▶ 38] the specific output will be latched **immediately**. If operated in short succession, the functionality of *bSwich* will change from driving to stopping to driving in the opposite direction to stopping and to driving again.

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. 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 rising edge has been applied to input *bPosition*. 0% corresponds to fully up, 100% corresponds to fully down.

bShadowPosition: The shading position is approached (see below).

nShadowSetPosition: Shading position (0%-100%) to which the blind is to be moved, after a rising 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 moves up.

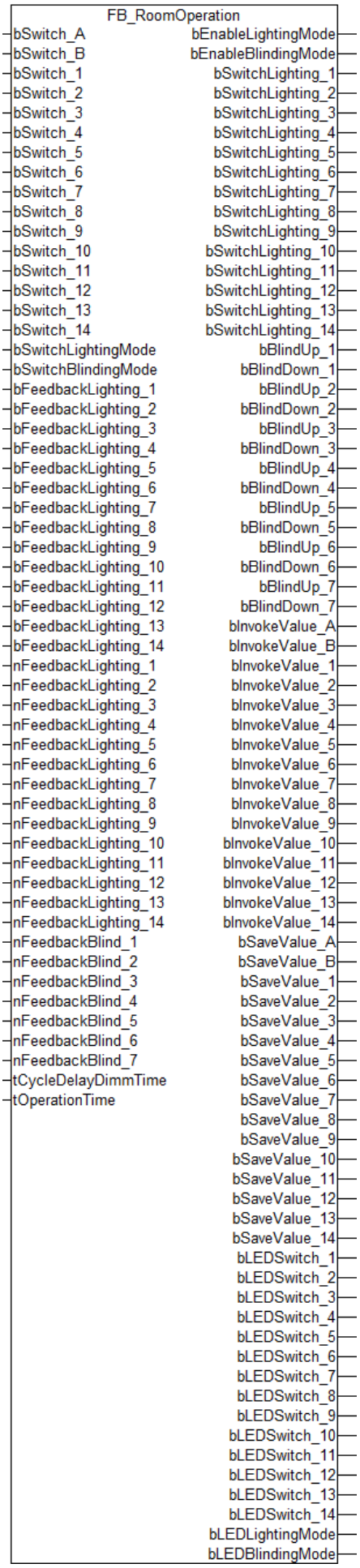
bBlindDown: The blind moves down.

nActualPosition: Current position in percent.

bCalibrated: Specifies whether the blind is calibrated.

3.3 Scene Management

3.3.1 FB_RoomOperation


FB_RoomOperation
bSwitch_A
bSwitch_B
bSwitch_1
bSwitch_2
bSwitch_3
bSwitch_4
bSwitch_5
bSwitch_6
bSwitch_7
bSwitch_8
bSwitch_9
bSwitch_10
bSwitch_11
bSwitch_12
bSwitch_13
bSwitch_14
bSwitchLightingMode
bSwitchBlindingMode
bFeedbackLighting_1
bFeedbackLighting_2
bFeedbackLighting_3
bFeedbackLighting_4
bFeedbackLighting_5
bFeedbackLighting_6
bFeedbackLighting_7
bFeedbackLighting_8
bFeedbackLighting_9
bFeedbackLighting_10
bFeedbackLighting_11
bFeedbackLighting_12
bFeedbackLighting_13
bFeedbackLighting_14
nFeedbackLighting_1
nFeedbackLighting_2
nFeedbackLighting_3
nFeedbackLighting_4
nFeedbackLighting_5
nFeedbackLighting_6
nFeedbackLighting_7
nFeedbackLighting_8
nFeedbackLighting_9
nFeedbackLighting_10
nFeedbackLighting_11
nFeedbackLighting_12
nFeedbackLighting_13
nFeedbackLighting_14
nFeedbackBlind_1
nFeedbackBlind_2
nFeedbackBlind_3
nFeedbackBlind_4
nFeedbackBlind_5
nFeedbackBlind_6
nFeedbackBlind_7
tCycleDelayDimmTime
tOperationTime
bEnableLightingMode
bEnableBlindingMode
bSwitchLighting_1
bSwitchLighting_2
bSwitchLighting_3
bSwitchLighting_4
bSwitchLighting_5
bSwitchLighting_6
bSwitchLighting_7
bSwitchLighting_8
bSwitchLighting_9
bSwitchLighting_10
bSwitchLighting_11
bSwitchLighting_12
bSwitchLighting_13
bSwitchLighting_14
bBlindUp_1
bBlindDown_1
bBlindUp_2
bBlindDown_2
bBlindUp_3
bBlindDown_3
bBlindUp_4
bBlindDown_4
bBlindUp_5
bBlindDown_5
bBlindUp_6
bBlindDown_6
bBlindUp_7
bBlindDown_7
bInvokeValue_A
bInvokeValue_B
bInvokeValue_1
bInvokeValue_2
bInvokeValue_3
bInvokeValue_4
bInvokeValue_5
bInvokeValue_6
bInvokeValue_7
bInvokeValue_8
bInvokeValue_9
bInvokeValue_10
bInvokeValue_11
bInvokeValue_12
bInvokeValue_13
bInvokeValue_14
bSaveValue_A
bSaveValue_B
bSaveValue_1
bSaveValue_2
bSaveValue_3
bSaveValue_4
bSaveValue_5
bSaveValue_6
bSaveValue_7
bSaveValue_8
bSaveValue_9
bSaveValue_10
bSaveValue_11
bSaveValue_12
bSaveValue_13
bSaveValue_14
bLEDSwitch_1
bLEDSwitch_2
bLEDSwitch_3
bLEDSwitch_4
bLEDSwitch_5
bLEDSwitch_6
bLEDSwitch_7
bLEDSwitch_8
bLEDSwitch_9
bLEDSwitch_10
bLEDSwitch_11
bLEDSwitch_12
bLEDSwitch_13
bLEDSwitch_14
bLEDLightingMode
bLEDBlindingMode

Description

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() [▶ 49], FB_ScenesVenetianBlind() [▶ 52], FB_Dimmer1Switch() [▶ 11] and FB_VenetianBlindEx() [▶ 52].

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() [▶ 49], FB_ScenesVenetianBlind() [▶ 52].

Note: This functionblock is only available in the PC-based version of the library.

VAR_INPUT

```
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;
bSwitchBlindingMode  : 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     : USINT;
nFeedbackBlind_2     : USINT;
nFeedbackBlind_3     : USINT;
nFeedbackBlind_4     : USINT;
nFeedbackBlind_5     : USINT;
nFeedbackBlind_6     : USINT;
nFeedbackBlind_7     : USINT;
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() [▶ 11].

nFeedbackLighting_1..14: current control value of the respective lamp. Return value from the dimmer function block FB_Dimmer1Switch() [▶ 11].

nFeedbackBlind_1..7: current control value of the respective blind. Return value from the blind function block FB_VenetianBlindEx() [▶ 38].

t**CycleDelayDimmTime:** 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;
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;
bSaveScene_A         : BOOL;
bSaveScene_B         : 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;
bLEDSwitch_1         : BOOL;
bLEDSwitch_2         : BOOL;
bLEDSwitch_3         : BOOL;
bLEDSwitch_4         : BOOL;
bLEDSwitch_5         : BOOL;
bLEDSwitch_6         : BOOL;
bLEDSwitch_7         : BOOL;
bLEDSwitch_8         : BOOL;
bLEDSwitch_9         : BOOL;
bLEDSwitch_10        : BOOL;
bLEDSwitch_11        : BOOL;
bLEDSwitch_12        : BOOL;
bLEDSwitch_13        : BOOL;
bLEDSwitch_14        : BOOL;
bLEDLightingMode     : BOOL;
bLEDBlindingMode     : BOOL;
```

bEnableLightingMode: enables the memory function block FB_ScenesLighting() [▶ 49].

bEnableBlindingMode: enables the memory function block FB_ScenesVenetianBlind() [▶ 52].

bSwitchLighting_1..14: output for operating the dimmer function block FB_Dimmer1Switch() [▶ 11] via the input *bSwitchDimm.*.

bSwitchBlindUp_1..7: output for operating the blind function block FB_VenetianBlindEx() [▶ 38] via the input *bSwitchOverUp*.

bSwitchBlindDown_1..7: output for operating the blind function block FB_VenetianBlindEx() [▶ 38] via the input *bSwitchOverDown*.

bInvokeScene_A, B, 1..14: output signal for loading a scene. Is passed on to the function blocks FB_ScenesLighting() [▶ 49] und FB_ScenesVenetianBlind() [▶ 52].

bSaveScene_A, B, 1..14: output signal for saving a scene. Is passed on to the function blocks FB_ScenesLighting() [▶ 49] und FB_ScenesVenetianBlind() [▶ 52].

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.

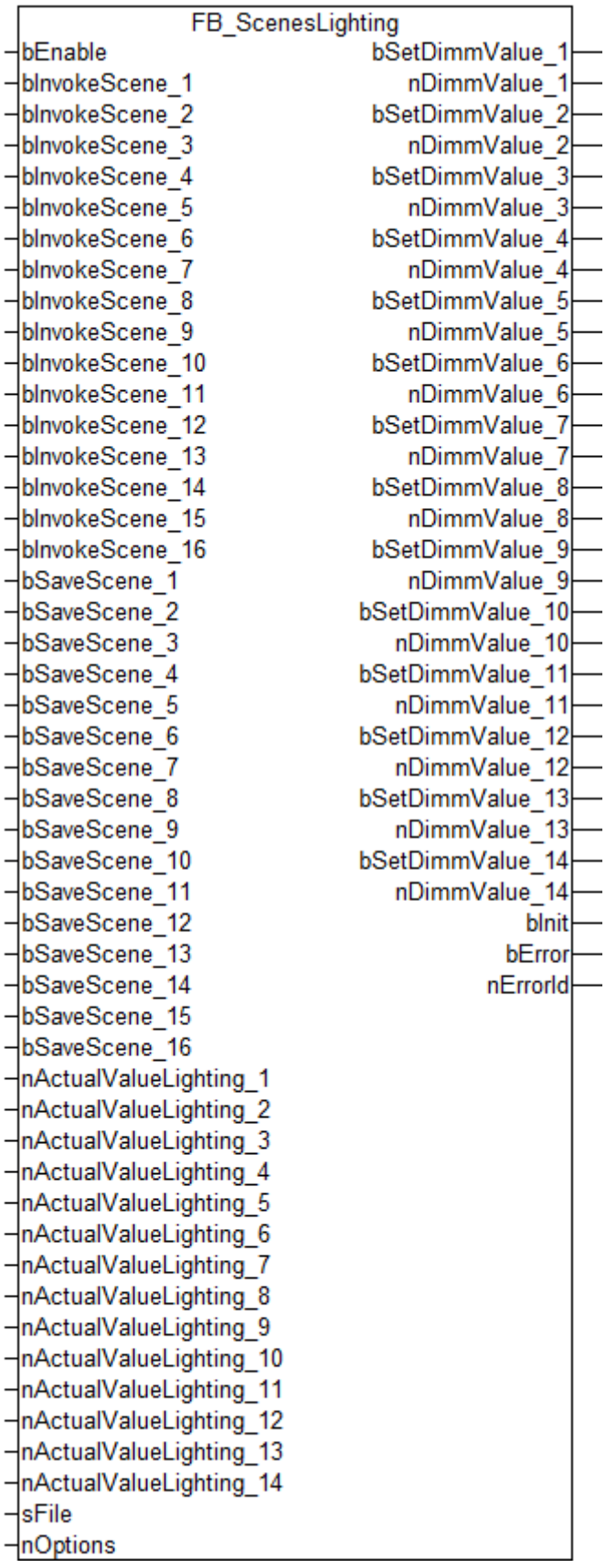
bLEDLightingMode: this output is TRUE if lighting mode is active.

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

Also see about this

FB_VenetianBlindEx [▶ 38]

3.3.2 FB_ScenesLighting



Description

Use of the function block

This function block is intended for the management of lighting scenes. The function block is enabled via the *bEnable* input. The loading of the saved scenes is started by a rising 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.

Saving a scene

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

Loading scenes

The saved values are output at the output *nDimmValue_1..14* by a rising edge at the input *bInvokeScene_1..16*. Furthermore, a rising edge is generated at the output *bSetDimmValue_1..14* for one PLC cycle.

Note: This functionblock is only available in the PC-based version of the library.

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;
```

bEnable: enables the function block.

bInvokeScene_1..16: calls the respective scene.

bSaveScene_1..16: saves the current analog value *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() [▶ 11].

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() [▶ 52] 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

```
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;
bInit               : BOOL;
bError              : BOOL;
nErrorId            : UDINT;
```

bSetDimmValue_1..14: output with the edge for the input b*SetDimmValue* of the function block FB_Dimmer1Switch() [▶ 11].

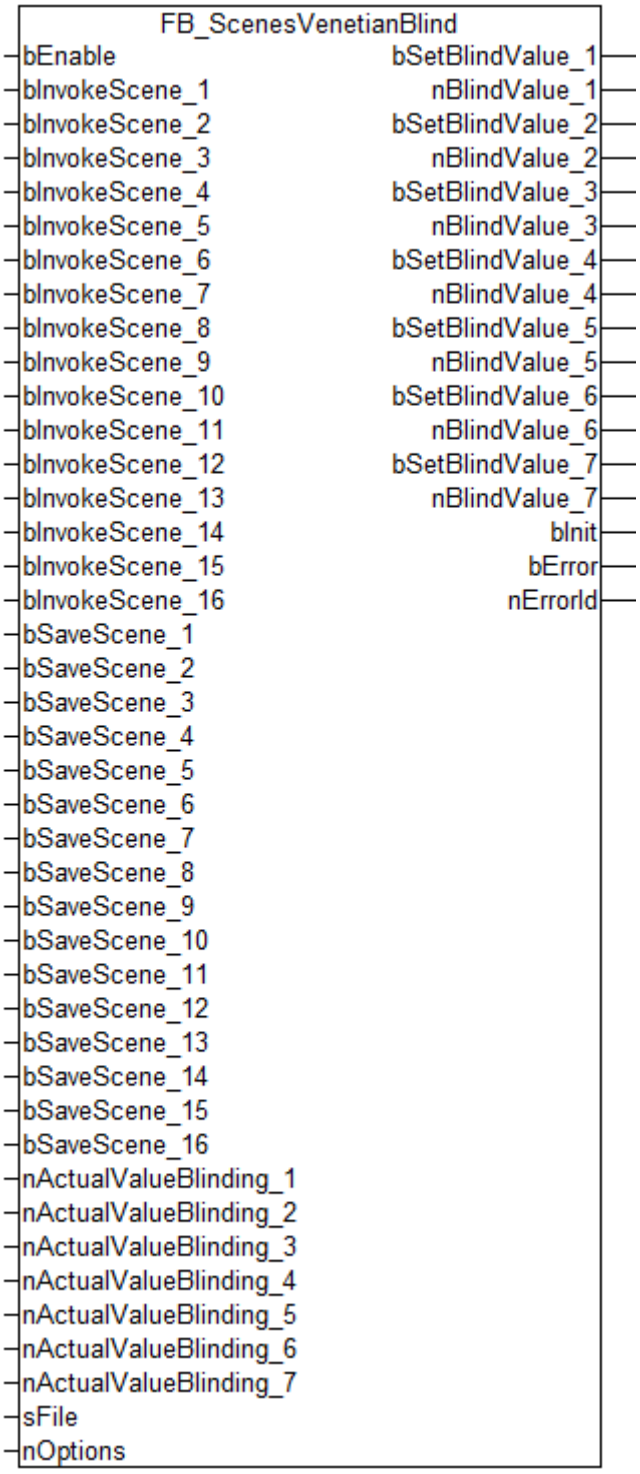
nDimmValue_1..14: Output with the value for the input *nDimmValue* of the function block FB_Dimmer1Switch() [▶ 11].

bInit: this output goes TRUE as soon as the initialisation 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 [▶ 89].

3.3.3 FB_ScenesVenetianBlind



Description

Use of the function block

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 rising 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.

Saving a 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*.

Loading scenes

The saved values are output at the output *nBlindValue_1..7* by a rising edge at the input *bInvokeScene_1..16*. Furthermore, a rising edge is generated at the output *bSetBlindValue_1..7* for one PLC cycle.

Note: This functionblock is only available in the PC-based version of the library.

VAR_INPUT

```
bEnable                 : BOOL;
bInvokeScene_1          : BOOL;
bInvokeScene_2          : 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;
nActualValueBlinding_1  : UINT;
nActualValueBlinding_2  : USINT;
nActualValueBlinding_3  : USINT;
nActualValueBlinding_4  : USINT;
nActualValueBlinding_5  : USINT;
nActualValueBlinding_6  : USINT;
nActualValueBlinding_7  : USINT;
sFile                   : STRING;
nOptions                : DWORD;
```

bEnable: enables the function block.

bInvokeScene_1..16: calls the respective scene.

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 FB_VenetianBlindEx() [▶ 38].

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() [▶ 49] or 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'.

nOptions: reserved for future developments.

VAR_OUTPUT

```
bSetBlindValue_1       : BOOL;
nBlindValue_1          : USINT;
bSetBlindValue_2       : BOOL;
nBlindValue_2          : USINT;
bSetBlindValue_3       : BOOL;
nBlindValue_3          : USINT;
bSetBlindValue_4       : BOOL;
nBlindValue_4          : USINT;
bSetBlindValue_5       : BOOL;
nBlindValue_5          : USINT;
bSetBlindValue_6       : BOOL;
nBlindValue_6          : USINT;
bSetBlindValue_7       : BOOL;
nBlindValue_7          : USINT;
bInit                  : BOOL;
bError                 : BOOL;
nErrorId               : UDINT;
```

bSetBlindValue_1..7: output with the edge for the input *bPosition* of the function block FB_VenetianBlindEx() [▶ 38].

nBlindValue_1..7: output with the value for the input *nSetPosition* of the function block FB_VenetianBlindEx() [▶ 38].

bInit: this output goes TRUE as soon as the initialisation 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 [▶ 89].

3.4 Signal Processing

3.4.1 FB_ShortLongClick

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

VAR_INPUT

```
bSwitch        : BOOL;
tSwitchTime    : TIME := t#50ms;
```

bSwitch: Input signal.

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

VAR_OUTPUT

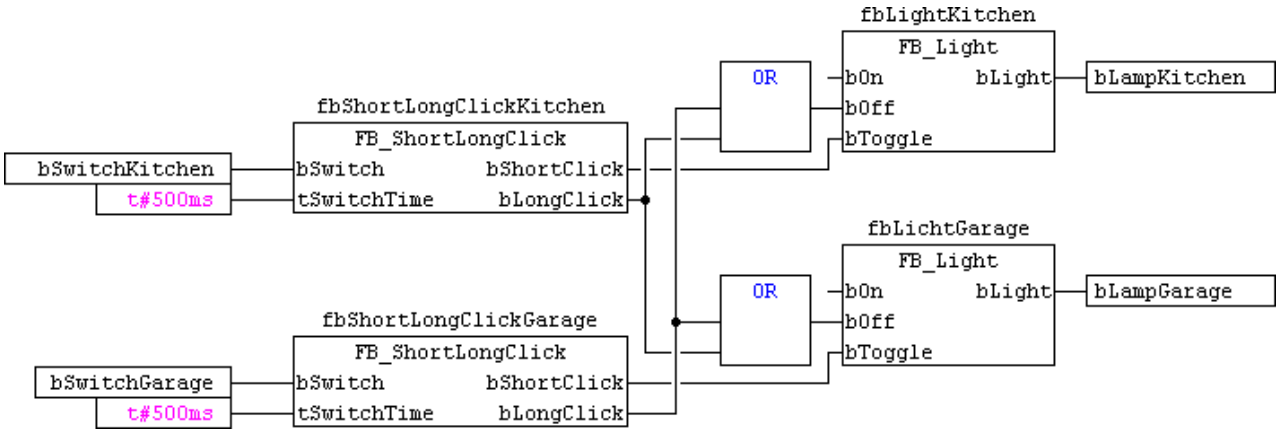
```
bShortClick    : BOOL;
bLongClick     : BOOL;
```

bShortClick: Indicates a short button press.

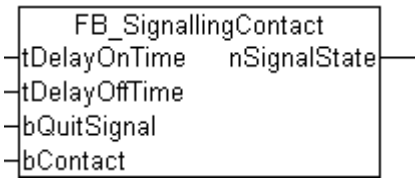
bLongClick: Indicates a long button press.

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 for longer than 500 ms, both lamps are switched off.



3.4.2 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 message contact is communicated to the 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:

Constant	Description
TCSIGNAL_INVALID	The message signal still does not have a defined state.
TCSIGNAL_SIGNALED	The message signal is active.
TCSIGNAL_RESET	The message signal has been reset.
TCSIGNAL_CONFIRMED	The message signal is confirmed, but has not yet been reset.
TCSIGNAL_SIGNALCON	The message signal is active and confirmed.
TCSIGNAL_RESETCON	The message signal is confirmed and reset.

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 message signal contact.

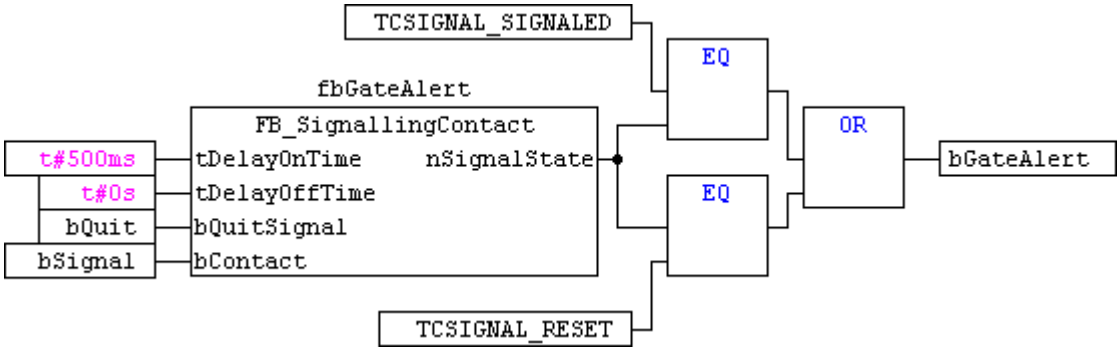
VAR_OUTPUT

```
nSignalState    : WORD;
```

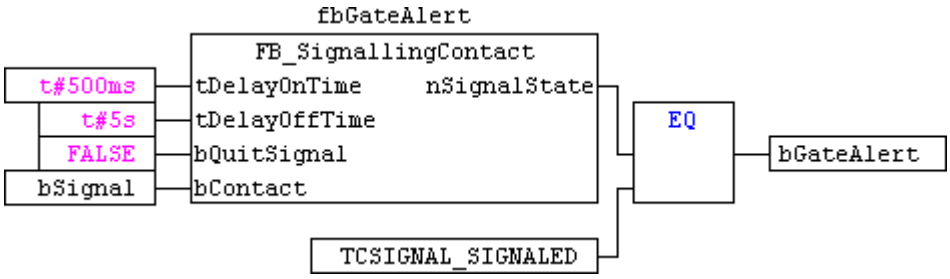
nSignalState: State of the message.

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_SIGNALED* or *TCSIGNAL_RESET*, the message is active. A rising edge at the *bQuitSignal* input acknowledges the message signal.

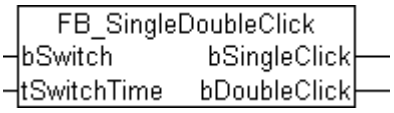


The following example illustrates the simplest case. A message signal not requiring acknowledgement.



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.

3.4.3 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

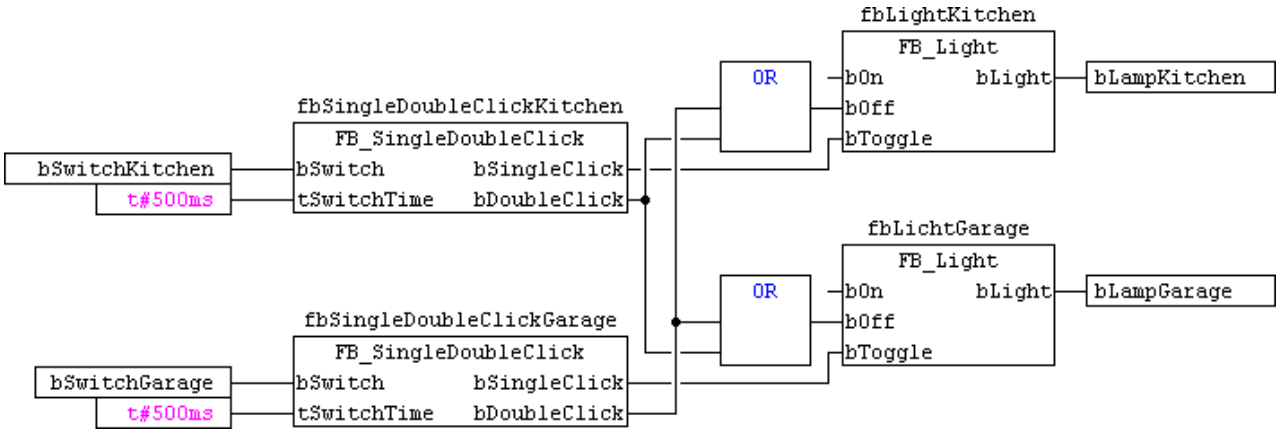
```
bSingleClick    : BOOL;
bDoublelick     : BOOL;
```

bSingleClick: Indicates a simple button press.

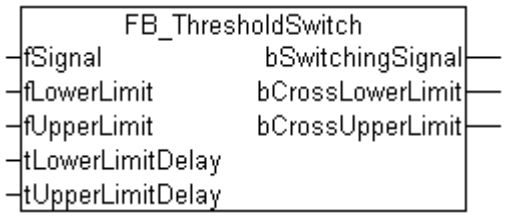
bDoublelick: Indicates a double button press.

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.



3.4.4 FB_ThresholdSwitch



If the input signal exceeds the limit value *fUpperLimit* for the duration specified by *tUpperLimitDelay*, the output *bCrossUpperLimit* is set for one PLC cycle. The *bSwitchingSignal* output is also set. This remains set until the input signal passes below the value of *fLowerLimit* for the duration specified by tLowerLimitDelay. In this case, the output *fCrossLowerLimit* is set for one PLC cycle.

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;
```

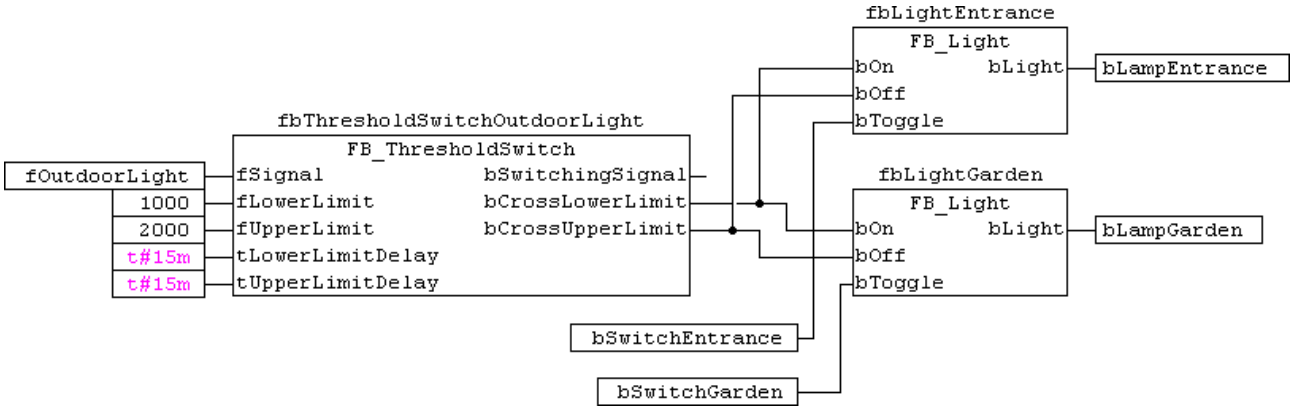
bSwitchingSignal: State depends on *bCrossLowerLimit* and *bCrossUpperLimit*.

bCrossLowerLimit: Is TRUE for one cycle, once *fLowerLimit* has fallen short of for the time *tLowerLimitDelay*. Simultaneously *bSwitchingSignal* is FALSE.

bCrossUpperLimit: Is TRUE for one cycle, once *fUpperLimit* was exceeded for the time *tUpperLimitDelay*. Simultaneously *bSwitchingSignal* is TRUE.

Example

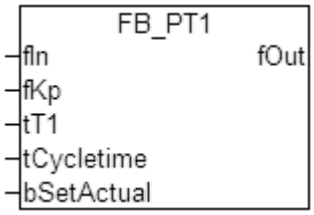
In the following example, the two lamps can each be controlled with one switch. The two lamps are automatically switched in response to the outside brightness and the threshold switch. The lamps are switched on if the outside brightness is less than 1000 lux for 15 minutes. The lamps are switched off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



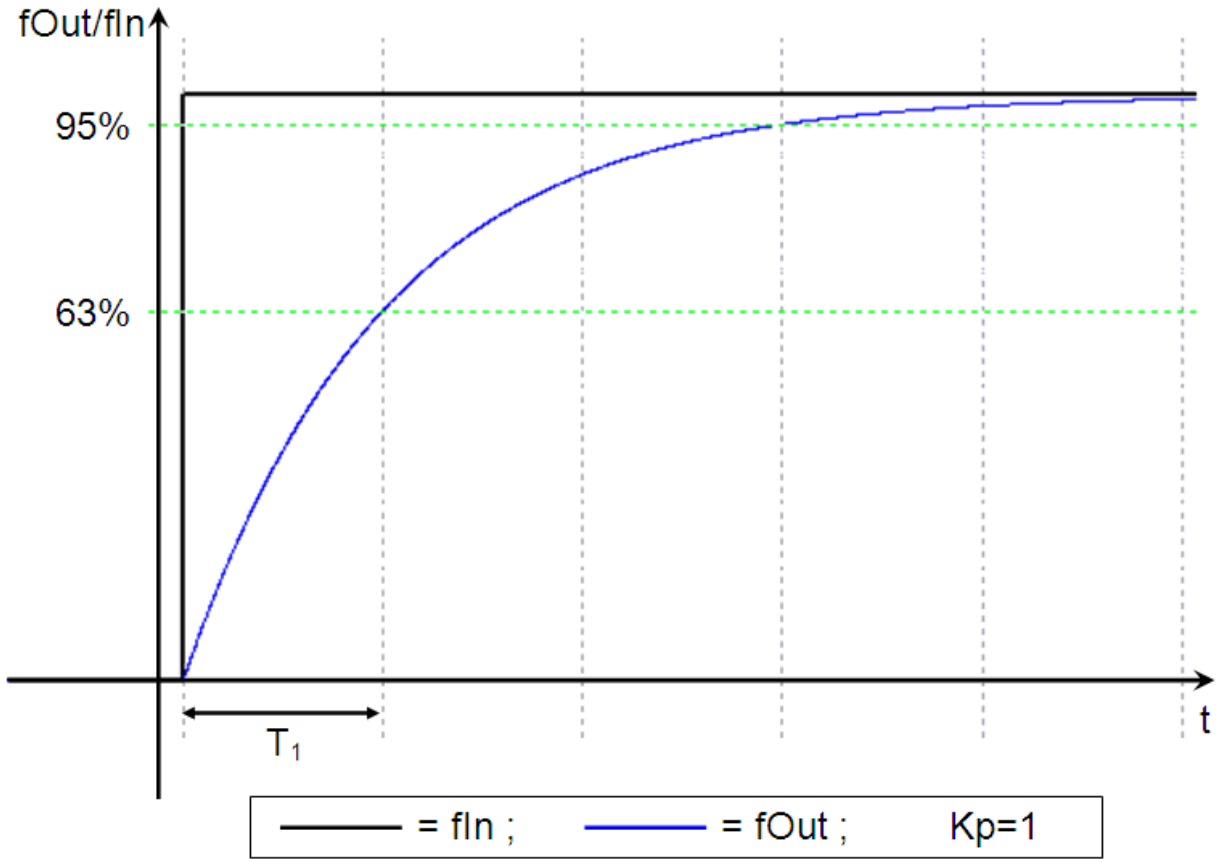
3.5 Filter Functions

3.5.1 FB_PT1

PT_1 element for smoothing of input values.



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



If *Kp* is 1 the output value directly follows the input value. *fOut* has already reached 63 % of the input value after the time *tT1* has elapsed, after 3 x *tT1* the value is 95 %.

The mathematical formula is:

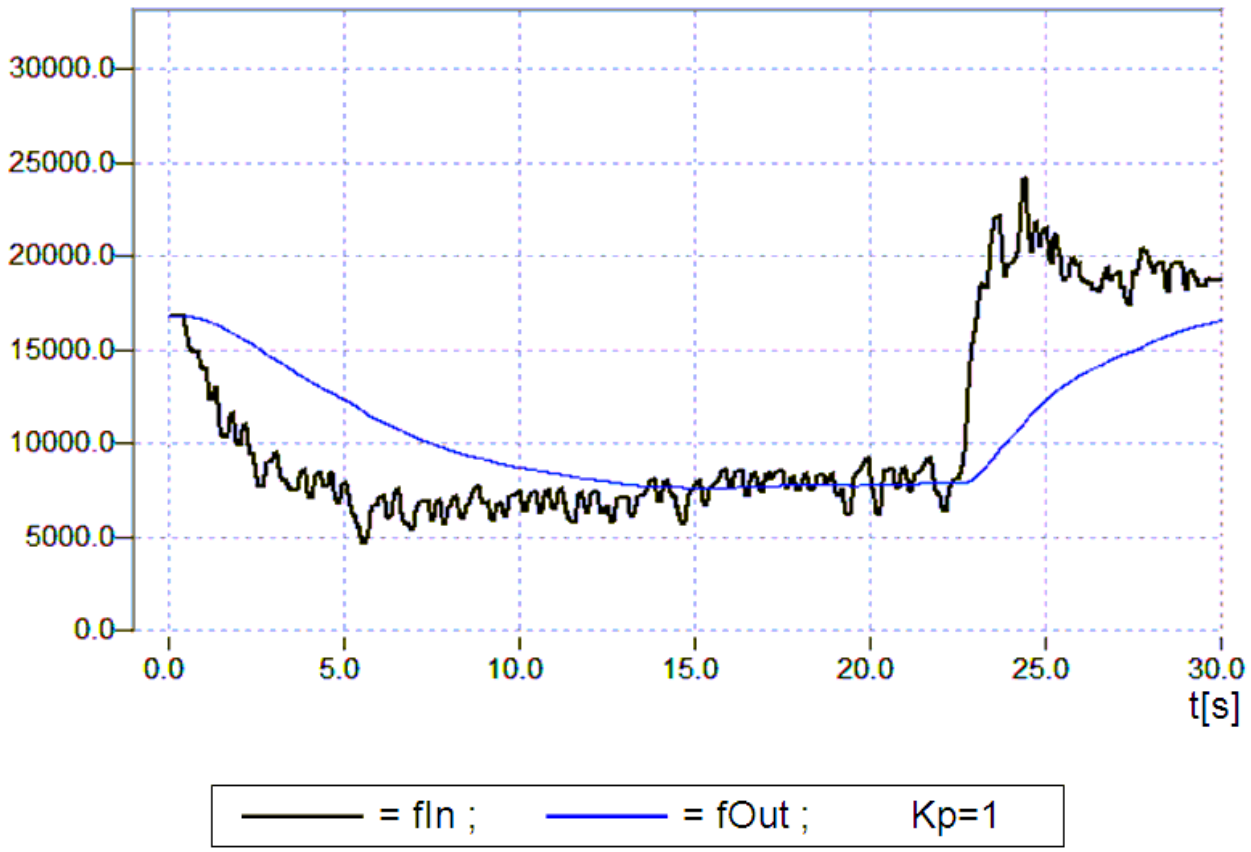
$$f(t) = K_p(1 - e^{-\frac{t}{T_1}})$$

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

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

$$\underline{T_1 = 0:} \qquad y_{n+1} = Kp \cdot x$$

With a continuously changing input *fIn*, *fOut* behaves as follows (*fIn*= 0..33000, *Kp*= 1, *T1*= 5 s):



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 0 s means that the output value directly follows the input value multiplied by Kp.

VAR_INPUT

```
fIn           : LREAL;
fKp           : LREAL := 1;
tT1           : TIME := t#10s;
tCycletime    : TIME := t#10ms;
bSetActual    : BOOL;
```

fIn: Input Value.

fKp: Amplifying-factor, preset value: 1.

tT1: Damping-time, preset value: 10s.

tCycleTime: PLC-cycle-time, preset value: 10ms.

bSetActual: Sets the output *fOut* directly to the input-value *fIn*.

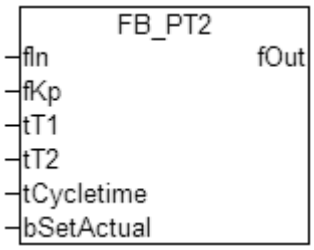
VAR_OUTPUT

```
fOut      : LREAL;
```

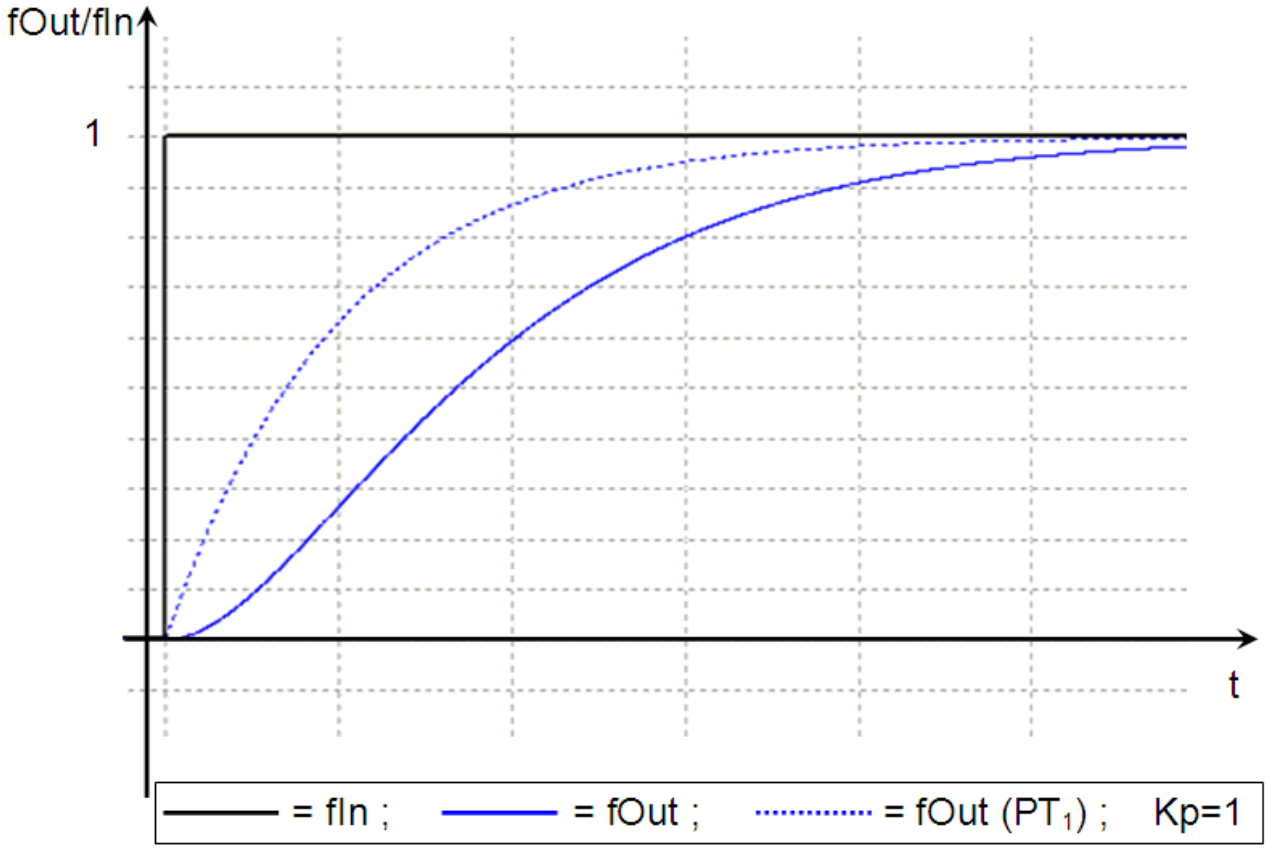
fOut: Output-Value.

3.5.2 FB_PT2

PT_2 element for smoothing of input values.

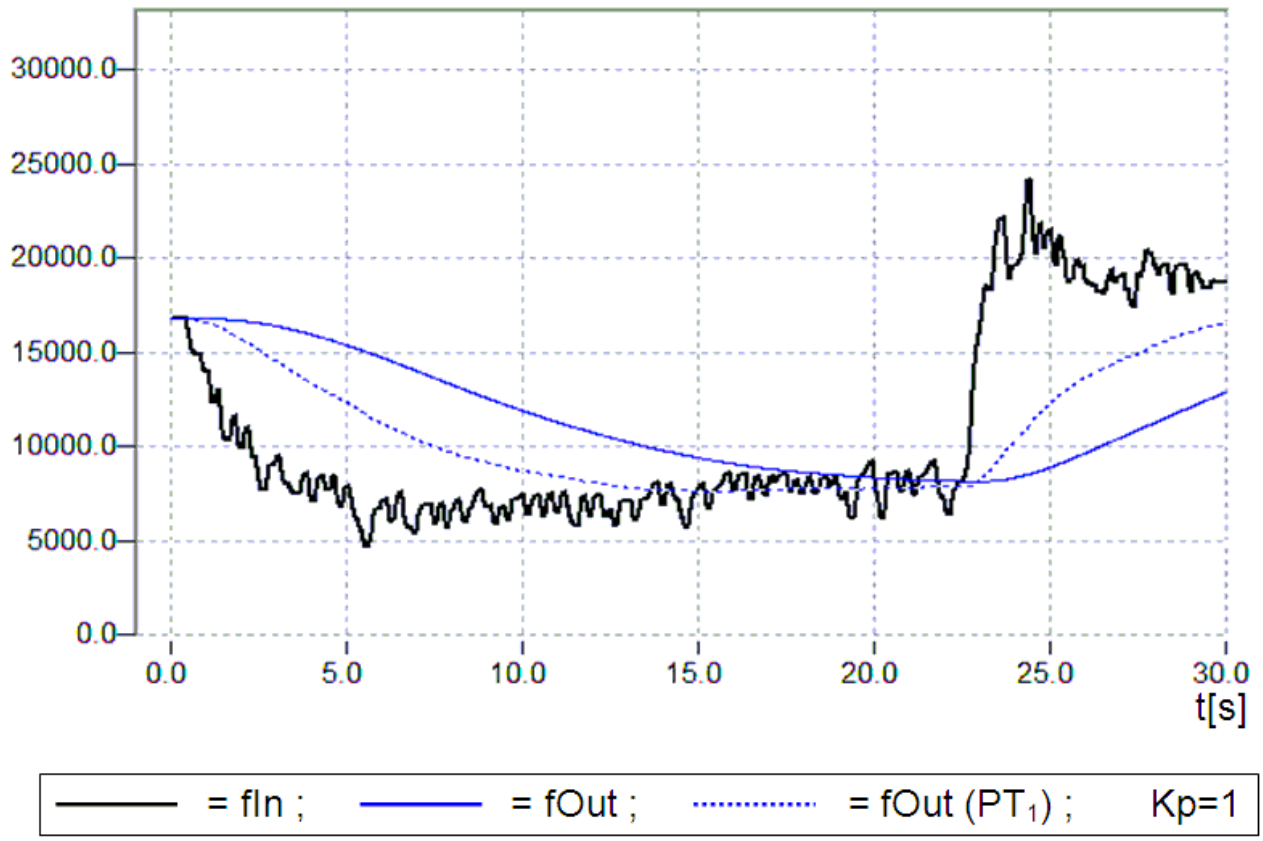


This function block is active continuously. The output *fOut* always follows the input value *fIn* multiplied by *Kp*.



This PT_2 element consists of a series of two PT_1 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 PT_1 element (dashed) right from the start.

With a continuously changing input *fIn*, *fOut* behaves as follows (*fIn*= 0..33000, *Kp*= 1, *T1,T2*= 5 s):



In comparison, the dotted line shows the behavior of a PT1 element [▶ 58] with *fIn*= 0..33000, *Kp*= 1, *T1*= 5 s.

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 value directly follows the input value 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: Amplifying-factor, preset value: 1.

tT1: Damping-time 1, preset value: 10s.

tT2: Damping-time 2, preset value: 10s.

tCycleTime: PLC-cycle-time, preset value: 10ms.

bSetActual: Sets the output *fOut* directly to the input-value *fIn*.

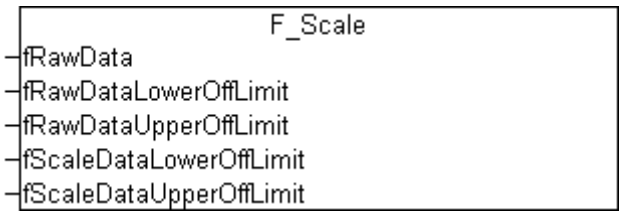
VAR_OUTPUT

```
fOut      : LREAL;
```

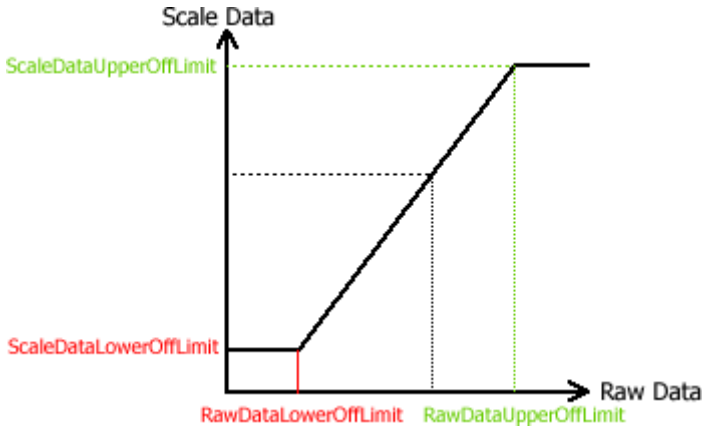
fOut: Output-Value.

3.6 Conversion Functions

3.6.1 F_Scale



A raw analog signal value is scaled to the specified range of measurements and returned as the function value. If the value of the raw signal extends beyond the upper or lower measurement 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.



VAR_INPUT

```
fRawData                    : LREAL;
fRawDataLowerOffLimit       : LREAL;
fRawDataUpperOffLimit       : LREAL;
fScaleDataLowerOffLimit     : LREAL;
fScaleDataUpperOffLimit     : LREAL;
```

fRawData: Raw data.

fRawDataLowerOffLimit: Lower limit for raw data.

fRawDataUpperOffLimit: Upper limit for raw data.

fScaleDataLowerOffLimit: Lower limit of scaled measurement.

fScaleDataUpperOffLimit: Upper limit of scaled measurement.

3.6.2 Temperature conversion functions

Functions for converting temperatures between Kelvin, Celsius, Reaumur and Fahrenheit.

F_TO_C	**K_TO_F**	**C_TO_F**	**R_TO_K**
F_TO_K	**K_TO_C**	**C_TO_K**	**R_TO_C**
F_TO_R	**K_TO_R**	**C_TO_R**	**R_TO_F**

Overview

	Kelvin (K)	Degrees Celsius (°C)	Reaumur (°R)	Fahrenheit (°F)
Absolute zero	0	-273,15	-218,52	-459,67
Melting point	273,15	0	0	32
Boiling point	373,15	100	80	212

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

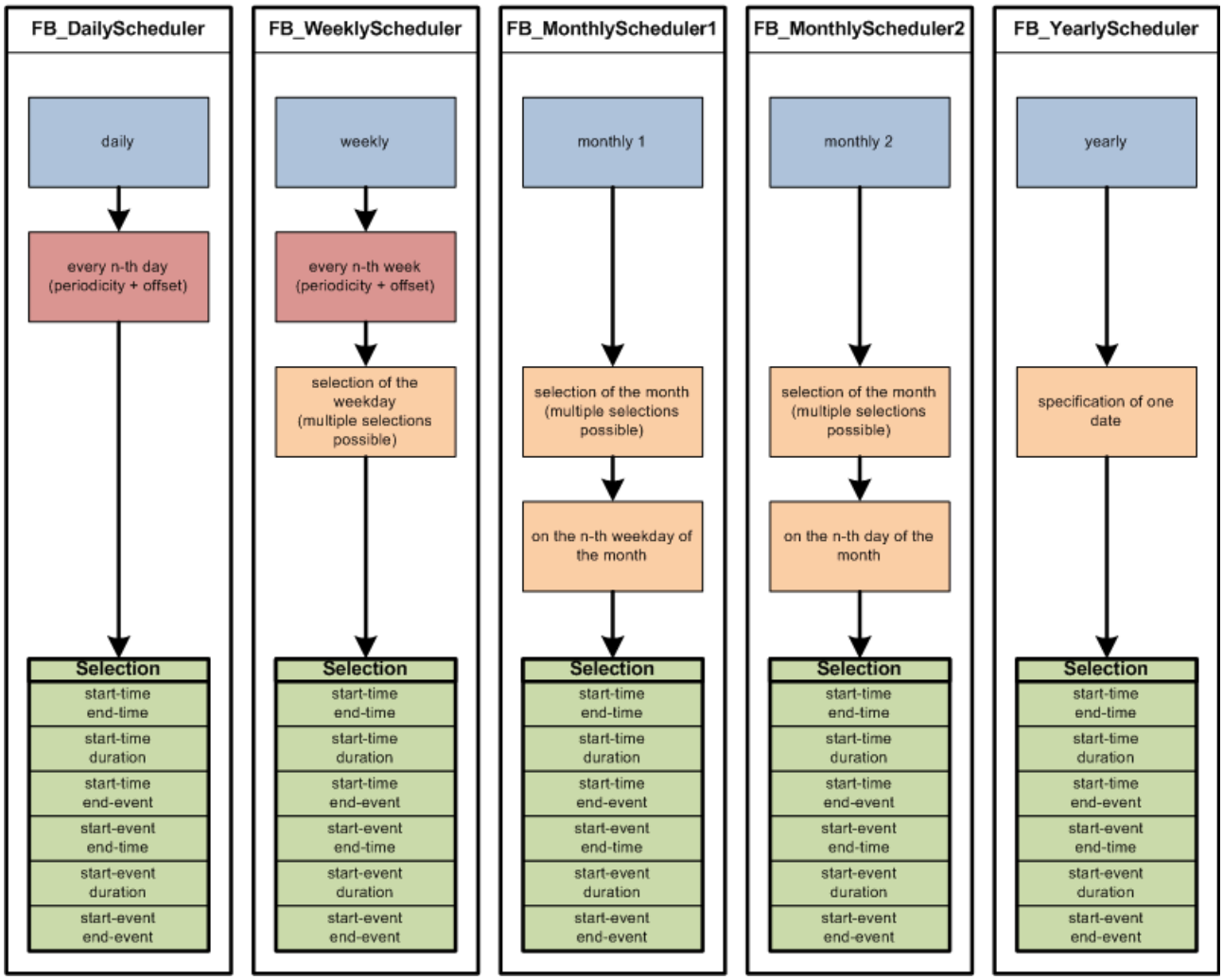
Conversion rules

	Kelvin (K)	Degrees Celsius (°C)	Reaumur (°R)	Fahrenheit (°F)
x = Kelvin (K)	-	$= x - 273{,}15°C$	$= \frac{4}{5}(x - 273{,}15)°R$	$= \frac{9}{5}(x - 273{,}15) + 32°F$
x = degrees Celsius (°C)	$= x + 273{,}15K$	-	$= \frac{4}{5}x°R$	$= \frac{9}{5}x + 32°F$
x = Reaumur (°R)	$= \frac{5}{4}x + 273{,}15K$	$= \frac{5}{4}x°C$	-	$= \frac{9}{4}x + 32°F$
x = Fahrenheit (°F)	$= \frac{5}{9}(x - 32) + 273{,}15K$	$= \frac{5}{9}(x - 32)°C$	$= \frac{4}{9}(x - 32)°R$	-

3.7 Time Switches

3.7.1 Scheduler Overview

The timer 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 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 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 daily. 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 end of the next period.

Further documentation

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

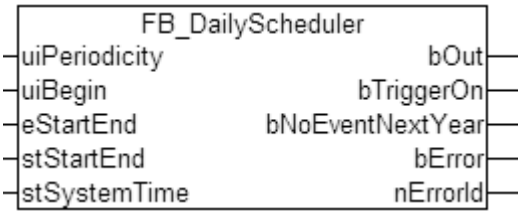
FB_DailyScheduler() [▶ 66]	FB_WeeklyScheduler() [▶ 67]	FB_MonthlyScheduler1() [▶ 69]	FB_MonthlyScheduler2() [▶ 70]	FB_YearlyScheduler() [▶ 71]
switches every n-th day	switches every n-th week on certain weekdays (multiple selection possible)	switches in certain months (multiple selection possible) on a certain day of the week	switches in certain months (multiple selection possible) on a certain day of the month	switches on a certain day of the year

Example Program

A sample program [▶ 73] uses a daily switching block (*FB_DailyScheduler*) to illustrate how the blocks must be parameterized.

3.7.1.1 FB_DailyScheduler

Function block for triggering actions every nth 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        : ENUM;
stStartEnd       : TIMESTRUCT;
stSystemtime     : TIMESTRUCT;
```

uiPeriodicity: Periodicity or interval. May be within the range 1 to 365.

uiBegin: Start value for the day counter. May be within the range 1 to 365. Example: *uiPeriodicity* = 5, *uiBegin* = 2: Switching events on 2 Jan., 7 Jan. 12 Jan. etc. - *uiPeriodicity* = 3, *uiBegin* = 1: Switching events on 1 Jan., 4 Jan. 7 Jan. etc.

eStartEnd: Selection of start/end definition.

TYPE E_StartEnd : (eSTARTTIME_ENDTIME := 1, eSTARTTIME_DURATION := 2, eSTARTTIME_ENDEVENT := 3, eSTARTEVENT_ENDTIME := 4, eSTARTEVENT_DURATION := 5, eSTARTEVENT_ENDEVENT := 6); 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.

stStartEnd: Structure with the parameters defining the start and end. Unused variables are ignored internally, e.g. the duration for the selection of start/end time.

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

todStartTime: Start time.

bStartEvent: Start event

tDuration: Switching duration.

todEndTime: End time.

bEndEvent: End event.

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 used to detect 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 [▶ 65] in the overview.

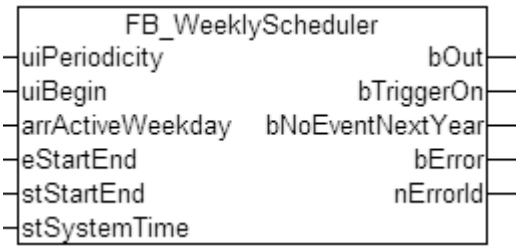
bNoEventNextYear: no day matching the parameterization was found within the next 366 days.

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 [▶ 89].

3.7.1.2 FB_WeeklyScheduler

Function block for triggering actions on certain weekdays in every nth 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    : UINT;
uiBegin          : UINT;
arrActiveWeekday : ARRAY[0..6] OF BOOL;
eStartEnd        : ENUM;
stStartEnd       : TIMESTRUCT;
stSystemtime     : TIMESTRUCT;
```

uiPeriodicity: Periodicity or interval. May be within the range 1 to 52.

uiBegin: Start value for the week. May be within the range 1 to 52. Example: *uiPeriodicity* = 5, *uiBegin* = 2: Switching events in week 2, week 7, week 12 etc. - *uiPeriodicity* = 3, *uiBegin* = 1: Switching events in week 2, week 7, week 12 etc.

arrActiveWeekday: Day of the week on which an action is to be triggered - *arrActiveWeekday[0]* => Sunday .. *arrActiveWeekday[6]* => Saturday. Multiple selections are possible.

eStartEnd: Selection of start/end definition.

TYPE E_StartEnd : (eSTARTTIME_ENDTIME := 1, eSTARTTIME_DURATION := 2, eSTARTTIME_ENDEVENT := 3, eSTARTEVENT_ENDTIME := 4, eSTARTEVENT_DURATION := 5, eSTARTEVENT_ENDEVENT := 6); 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.

stStartEnd: Structure with the parameters defining the start and end. Unused variables are ignored internally, e.g. the duration for the selection of start/end time.

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

todStartTime: Start time.

bStartEvent: Start event

tDuration: Switching duration.

todEndTime: End time.

bEndEvent: End event.

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 used to detect 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 [▶ 65] in the overview.

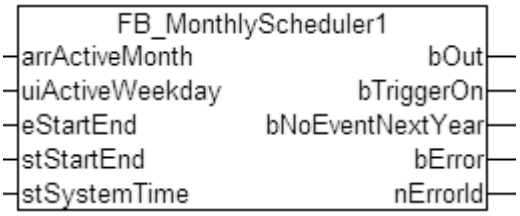
bNoEventNextYear: no day matching the parameterization was found within the next 366 days.

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 [▶ 89].

3.7.1.3 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

```
arrActiveMonth    : ARRAY[1..12] OF BOOL;
uiActiveWeekday   : UINT;
eStartEnd         : ENUM;
stStartEnd        : TIMESTRUCT;
stSystemtime      : TIMESTRUCT;
```

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

uiActiveWeekday: 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

eStartEnd: Selection of start/end definition.

TYPE E_StartEnd : (eSTARTTIME_ENDTIME := 1, eSTARTTIME_DURATION := 2, eSTARTTIME_ENDEVENT := 3, eSTARTEVENT_ENDTIME := 4, eSTARTEVENT_DURATION := 5, eSTARTEVENT_ENDEVENT := 6); 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.

stStartEnd: Structure with the parameters defining the start and end. Unused variables are ignored internally, e.g. the duration for the selection of start/end time.

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

todStartTime: Start time.

bStartEvent: Start event

tDuration: Switching duration.

todEndTime: End time.

bEndEvent: End event.

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 used to detect 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 [▶ 65] in the overview.

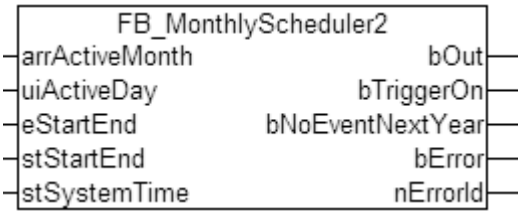
bNoEventNextYear: no day matching the parameterization was found within the next 366 days.

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 [▶ 89].

3.7.1.4 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   : ARRAY[1..12] OF BOOL;
uiActiveWeekday  : UINT;
eStartEnd        : ENUM;
stStartEnd       : TIMESTRUCT;
stSystemtime     : TIMESTRUCT;
```

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 start/end definition.

TYPE E_StartEnd : (eSTARTTIME_ENDTIME := 1, eSTARTTIME_DURATION := 2, eSTARTTIME_ENDEVENT := 3, eSTARTEVENT_ENDTIME := 4, eSTARTEVENT_DURATION := 5, eSTARTEVENT_ENDEVENT := 6); 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.

stStartEnd: Structure with the parameters defining the start and end. Unused variables are ignored internally, e.g. the duration for the selection of start/end time.

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

todStartTime: Start time.

bStartEvent: Start event

tDuration: Switching duration.

todEndTime: End time.

bEndEvent: End event.

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 used to detect 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 [▶ 65] in the overview.

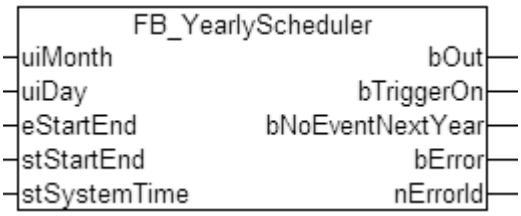
bNoEventNextYear: no day matching the parameterization was found within the next 366 days.

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 [▶ 89].

3.7.1.5 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        : ENUM;
stStartEnd       : TIMESTRUCT;
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 start/end definition.

TYPE E_StartEnd : (eSTARTTIME_ENDTIME := 1, eSTARTTIME_DURATION := 2, eSTARTTIME_ENDEVENT := 3, eSTARTEVENT_ENDTIME := 4, eSTARTEVENT_DURATION := 5, eSTARTEVENT_ENDEVENT := 6); 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.

stStartEnd: Structure with the parameters defining the start and end. Unused variables are ignored internally, e.g. the duration for the selection of start/end time.

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

todStartTime: Start time.

bStartEvent: Start event

tDuration: Switching duration.

todEndTime: End time.

bEndEvent: End event.

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 used to detect 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 [▶ 65] 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 [▶ 89].

3.7.1.6 Scheduler Example

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

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

```
0001 PROGRAM P_SystemTime
0002 VAR
0003     fbGetTime            :    NT_GetTime;
0004     dtSystemTime         :    DT;
0005     strDateTime          :    TIMESTRUCT;
0006     tonGetTime           :    TON;
0007     nStep                :    INT;
0008     bSystemTimeValid     :    BOOL := FALSE;
0009 END_VAR
```

```
0001 (* Read The Time *)
0002 CASE nStep OF
0003 0:
0004     tonGetTime(IN := TRUE, PT := t#500ms);
0005     IF (tonGetTime.Q) THEN
0006         tonGetTime(IN := FALSE);
0007         nStep := 10;
0008     END_IF
0009 10:
0010     fbGetTime(  NETID := '',
0011                 START := TRUE,
0012                 TMOUT := t#2s);
0013     IF (NOT fbGetTime.BUSY) THEN
0014         strDateTime := fbGetTime.TIMESTR;
0015         dtSystemTime := SYSTEMTIME_TO_DT(fbGetTime.TIMESTR);
0016         fbGetTime(START := FALSE);
0017         bSystemTimeValid := TRUE;
0018         nStep := 0;
0019     END_IF
0020
0021 END_CASE
```

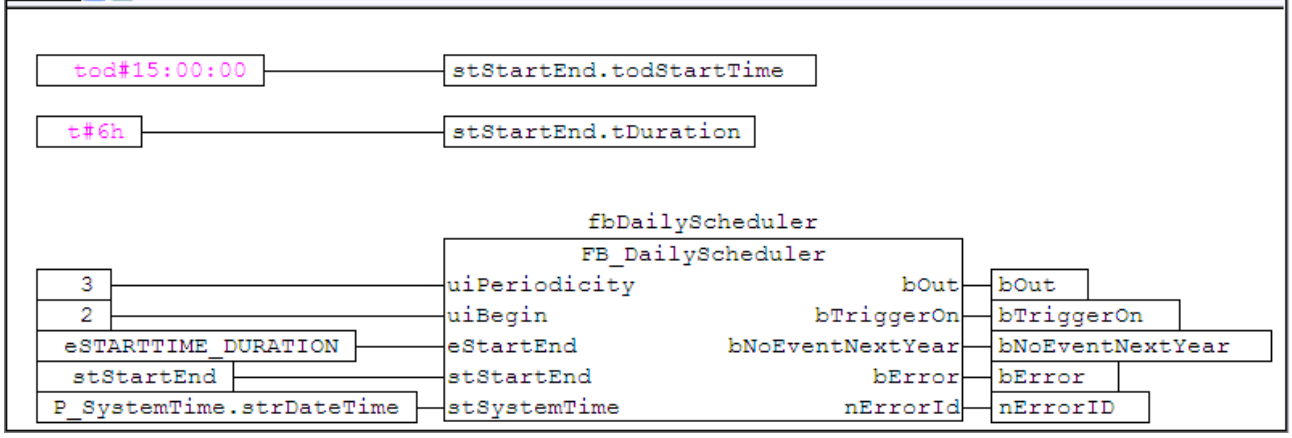
It provides a time base for parameterizing the scheduler blocks with regard to the time input *stSystemTime*. The enumerator matching the required behavior is created at input *eStartEnd*:

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*. In program the subvariables for this structure that are relevant for the function type are described. For the example shown here these are *todStartTime* and *tDuration*. All other variables are not read and therefore do not have to be described.

```
0001 PROGRAM P_SchedulerExample
0002 VAR
0003     fbDailyScheduler     : FB_DailyScheduler;
0004     stStartEnd           : ST_StartEnd;
0005
0006     bOut                 : BOOL;
0007     bTriggerOn           : BOOL;
0008     bNoEventNextYear     : BOOL;
0009     bError               : BOOL;
0010     nErrorID             : UDINT;
0011 END_VAR
0012
```

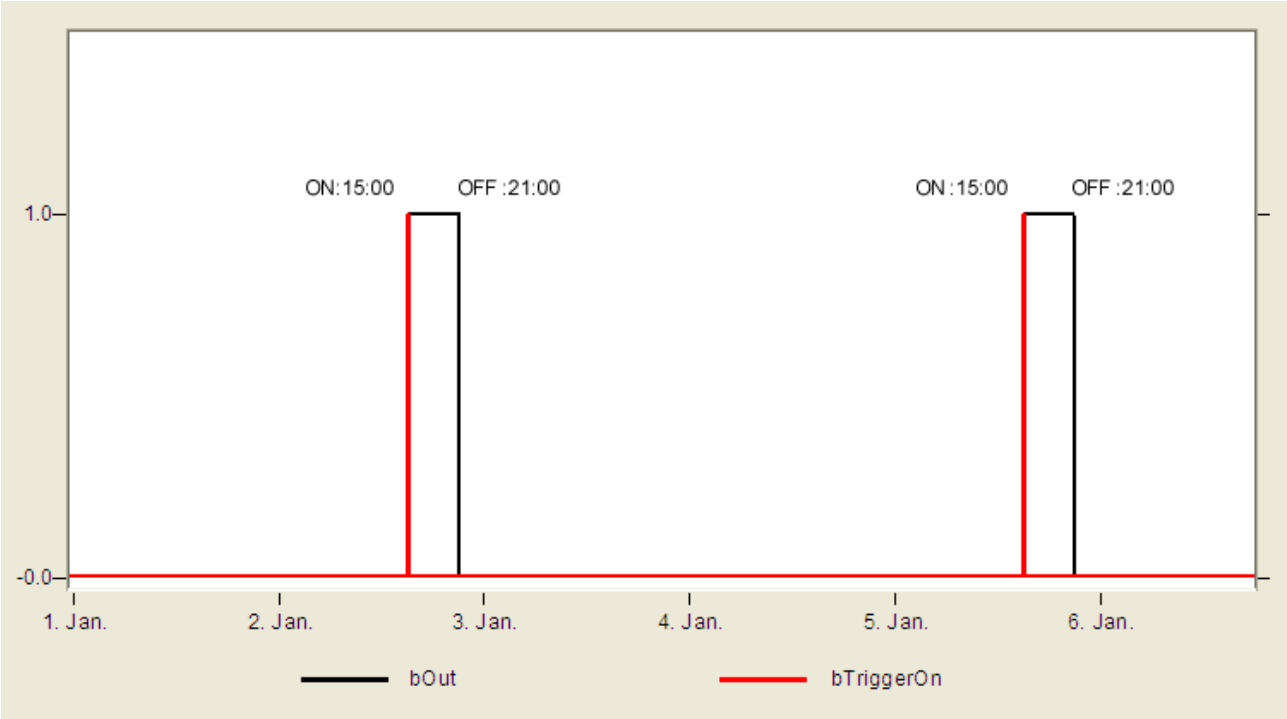


Both programs have to be called in the MAIN block. The program part *P_SchedulerExample* may only be called once the program part P_SystemTime supplies valid data, i.e. once *P_SystemTimeValid* is TRUE. The reason for this protective logic is that reading the time takes several cycles which means that the time when the program starts is invalid and must not be used.

```
0001 PROGRAM MAIN
0002 VAR
0003 END_VAR
```

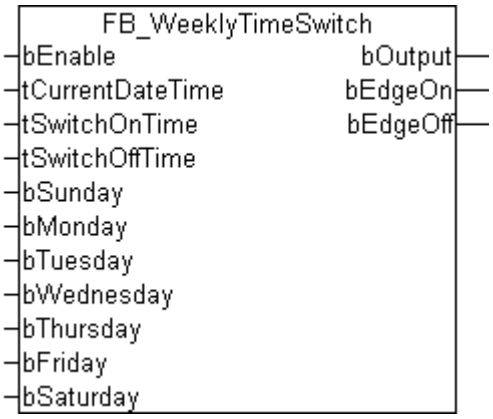
```
0001 P_SystemTime;
0002 IF NOT P_SystemTime.bSystemTimeValid THEN
0003 RETURN;
0004 END_IF
0005 P_SchedulerExample;
0006
```

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



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 := tod#15:00:00*) and the switching duration is 6 hours (*stStartEnd.tDuration := t#6h*).

3.7.2 FB_WeeklyTimeSwitch



The parameters *tSwitchOnTime* and *tSwitchOffTime* define a period of time, in which *bOutput* will be set to TRUE. The timer is only active on the selected days of the week. This selection is done by setting the inputs *bSunday*, *bMonday*, ..., *bSaturday*. It is only possible to define one switching-period per timer. Each further switching-period requires a new timer.

VAR_INPUT

```
bEnable           : BOOL;
tCurrentDateTime  : DATE_AND_TIME;
tSwitchOnTime     : TOD;
tSwitchOffTime    : TOD;
bSunday           : BOOL;
bMonday           : BOOL;
bTuesday          : BOOL;
bWednesday        : BOOL;
bThursday         : BOOL;
bFriday           : BOOL;
bSaturday         : BOOL;
```

bEnable: Timer-release.

tCurrentDateTime: Actual time and date

tSwitchOnTime: Time, when *bOutput* will be set to TRUE.

tSwitchOffTime: Time, when *bOutput* will be set to FALSE.

bSunday: Timer is active on Sunday.

bMonday: Timer is active on Monday.

bTuesday: Timer is active on Tuesday.

bWednesday: Timer is active on Wednesday.

bThursday: Timer is active on Thursday.

bFriday: Timer is active on Friday.

bSaturday: Timer is active on Saturday.

VAR_OUTPUT

```
bOutput           : BOOL;
bEdgeOn           : BOOL;
bEdgeOff          : BOOL;
```

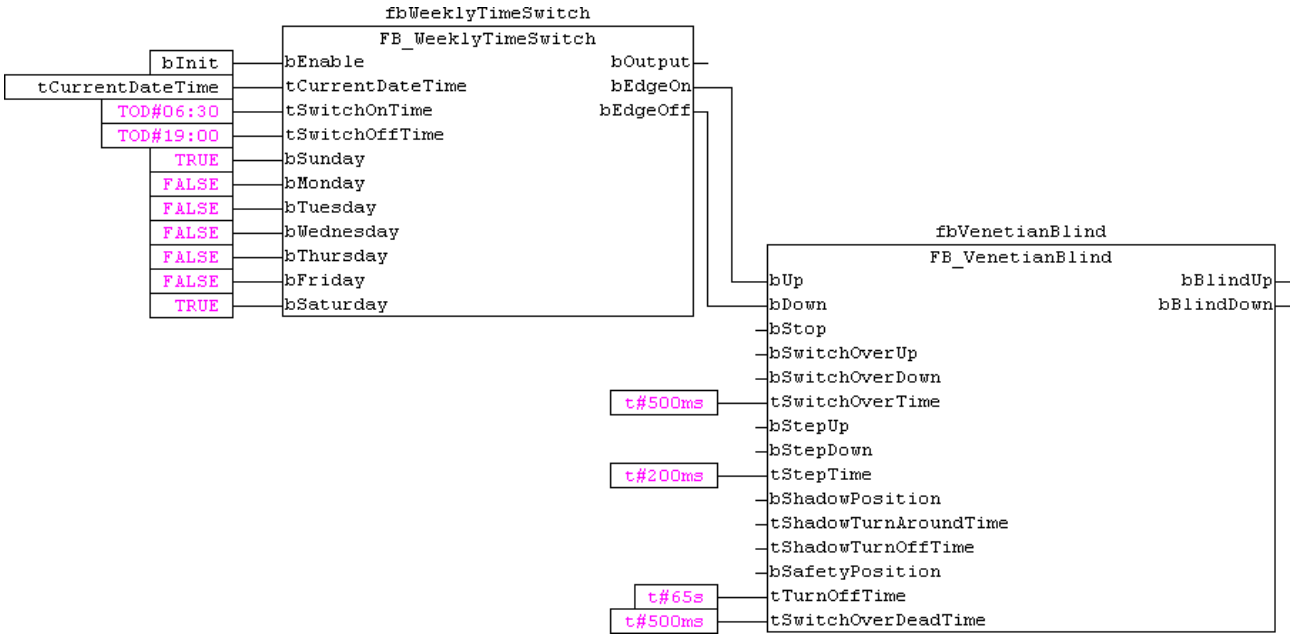
bOutput: As long as the actual time lies between the on- and the off-time, this output will be set to TRUE.

bEdgeOn: When *bOutput* turns to its TRUE-state, this output will be set to TRUE for one PLC-cycle.

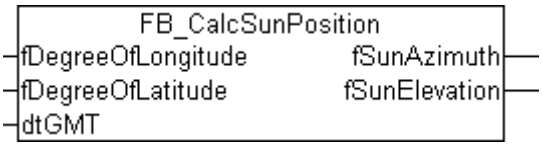
bEdgeOff: When *bOutput* turns to its FALSE-state, this output will be set to TRUE for one PLC-cycle.

Example

The following example shows a blind, which is programmed to go up at 6.30am and to go down at 7.00pm at the weekend. The timer outputs *bEdgeOn* and *bEdgeOff* are used to control the blind-function, which needs pulses at the inputs *bUp* und *bDown* to move the blind.



3.7.3 FB_CalcSunPosition

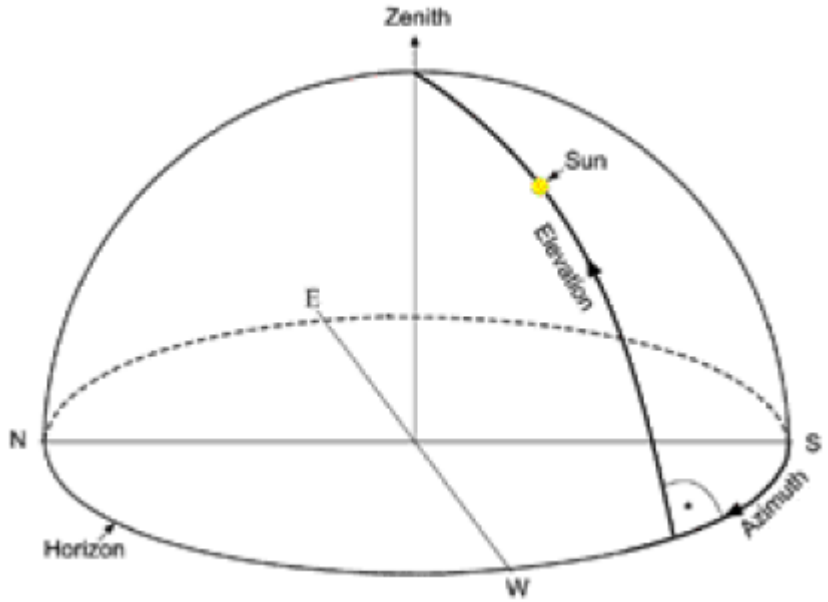


Calculation of the position of the sun by means of specifying the date, time, longitude and latitude.

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 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 is, then a value of 0° is assigned for the northern sun direction and it then runs in the clockwise direction around the compass, i.e., 90° is east, 180° is south, 270° is west etc. If the point of observation is in the southern hemisphere, then 0° corresponds to the southern direction and it then runs in the counterclockwise direction, 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:

Place	Longitude	Latitude
Sydney, Australia	151,2°	-33,9°
New York, USA	-74,0°	40,7°
London, England	-0,1°	51,5°
Moscow, Russia	37,6°	55,7°

Place	Longitude	Latitude
Peking, China	116,3°	39,9°
Dubai, United Arab Emirates	55,3°	25,4°
Rio de Janeiro, Brazil	-43,2°	-22,9°
Hawai, USA	-155,8°	20,2°
Verl, Germany	8,5°	51,9°

If the 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;
```

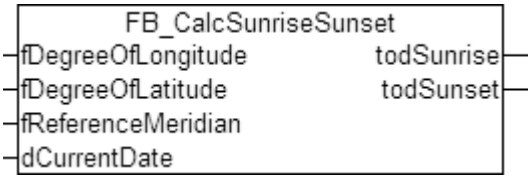
fSunAzimuth: Direction of the sun (northern hemisphere: 0° north… 90° east… 180° south… 270° west…/ southern hemisphere: 0° south… 90° east… 180° north… 270° west…).

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

Sample

```
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));
ffSunAzimuth := fbCalcSunPosition.fSunAzimuth;
fSunElevation := fbCalcSunPosition.fSunElevation;
```

3.7.4 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:

Time zone	Reference meridian
GMT (Greenwich Mean Time)	$\lambda_{GMT} = 0°$
CET (Central European Time)	$\lambda_{MEZ} = 15°$
CEST (Central European Summer Time)	$\lambda_{CEST} = 30°$

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;
```

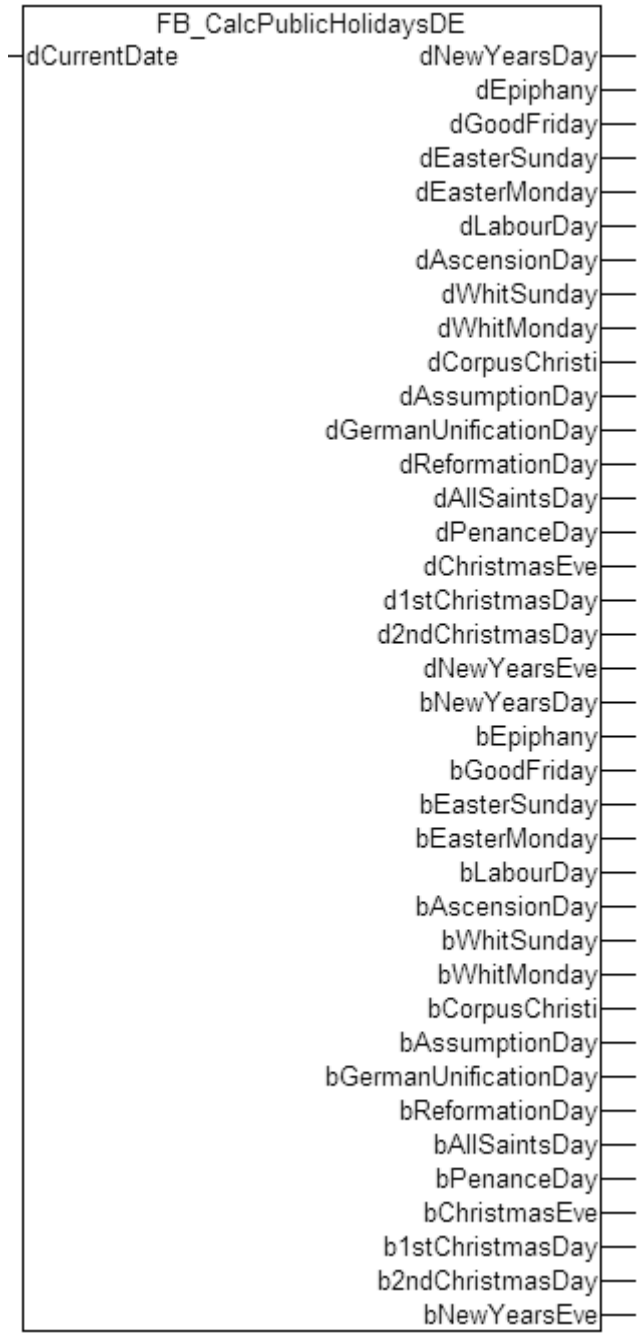
todSunrise: Sunrise. Output of hour and minute.

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);
```

3.7.5 FB_CalcPublicHolidaysDE



Calculation of German public holidays.

Description

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 block was translated into English. The parameters have the following meaning:

English name	German name
NewYears Day	Neujahr
Epiphany	Heilige Drei Könige
Good Friday	Karfreitag
Easter Sunday	Ostersonntag
Easter Monday	Ostermontag

English name	German name
Labour Day	Maifeiertag
Ascension Day	Christi Himmelfahrt
Whit Sunday	Pfingstsonntag
Whit Monday	Pfingstmontag
Corpus Christi	Fronleichnam
Assumption Day	Mariä Himmelfahrt
German Unification Day	Tag Der Deutschen Einheit
Reformation Day	Reformationstag
All Saints Day	Allerheiligen
Penance Day	Buß- und Bettag
Christmas Eve	Heiligabend
1st ChristmasDay	1. Weihnachtstag
2nd ChristmasDay	2. Weihnachtstag
New Years Eve	Silvester

VAR_INPUT

```
dCurrentDate      : DATE;
```

dCurrentDate: current date.

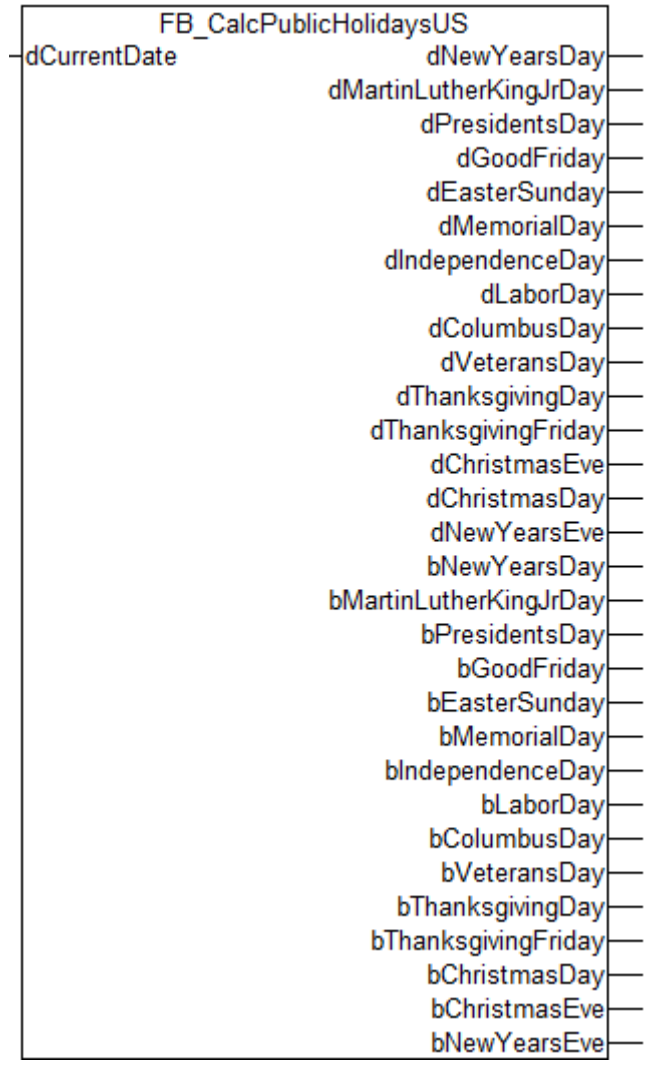
VAR_OUTPUT

```
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.

3.7.6 FB_CalcPublicHolidaysUS



Calculation of US public holidays.

Description

Public holidays recognized by the US Federal government, as well as most popular holiday 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 block was translated into English. The parameters have the following meaning:

English name	German name
New Years Day	Neujahr
Martin Luther King, Jr. Day	MLKtag
Presidents Day	Presidentstag
Good Friday	Karfreitag
Easter Sunday	Ostersonntag
Memorial Day	Ostermontag
Independence Day	Independencetag
Labor Day	Maifeiertag
Columbus Day	Columbustag
Veterans Day	Veteranstag
Thanksgiving Day	Thanksgivingtag

English name	German name
Thanksgiving Friday	Thanksgivingfriestag
Christmas Eve	Heiligabend
Christmas Day	Weihnachtstag
New Years Eve	Silvester

VAR_INPUT

```
dCurrentDate       : DATE;
```

dCurrentDate: Current date.

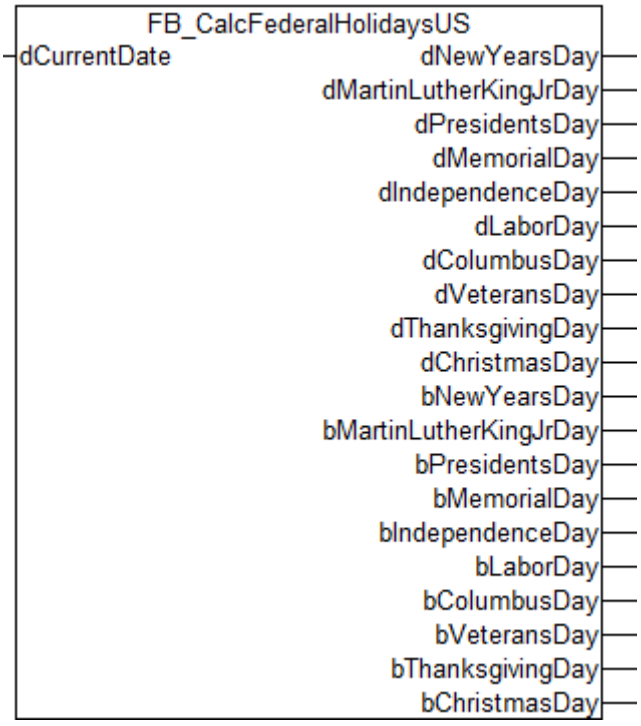
VAR_OUTPUT

```
dNewYearsDay           : DATE;
dMartinLutherKingJrDay : DATE;
dPresidentsDay         : DATE;
dGoodFriday            : DATE;
dEasterSunday          : DATE;
dMemorialDay           : DATE;
dIndependenceDay       : DATE;
dLaborDay              : DATE;
dColumbusDay           : DATE;
dVeteransDay           : DATE;
dThanksgivingDay       : DATE;
dThanksgivingFriday    : DATE;
dChristmasEve          : DATE;
dChristmasDay          : DATE;
dNewYearsEve           : DATE;

bNewYearsDay           : BOOL;
bMartinLutherKingJrDay : BOOL;
bPresidentsDay         : BOOL;
bGoodFriday            : BOOL;
bEasterSunday          : BOOL;
bMemorialDay           : BOOL;
bLaborDay              : BOOL;
bColumbusDay           : BOOL;
bVeteransDay           : BOOL;
bThanksgivingDay       : BOOL;
bThanksgivingFriday    : BOOL;
bChristmasEve          : BOOL;
bChristmasDay          : BOOL;
bNewYearsEve           : BOOL;
```

dxxxxxx: Date of the respective holiday.

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

3.7.7 FB_CalcFederalHolidaysUS



Calculation of US federal holidays.

Description

Public holidays recognized by the US Federal government 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 block was translated into English. The parameters have the following meaning:

English name	German name
New Years Day	Neujahr
Martin Luther King, Jr. Day	MLKtag
Presidents Day	Presidentstag
Memorial Day	Memorialtag
Independence Day	Independencetag
Labor Day	Maifeiertag
Columbus Day	Columbustag
Veterans Day	Veteranstag
Thanksgiving Day	Thanksgivingtag
Christmas Day	Weihnachtstag

VAR_INPUT

```
dCurrentDate       : DATE;
```

dCurrentDate: Current date.

VAR_OUTPUT

```
dNewYearsDay           : DATE;
dMartinLutherKingJrDay : DATE;
dPresidentsDay         : DATE;
dMemorialDay           : DATE;
dIndependenceDay       : DATE;
dLaborDay              : DATE;
```

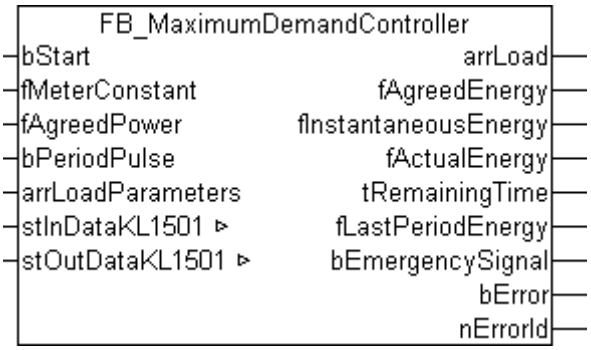
```
dColumbusDay            : DATE;
dThanksgivingDay        : DATE;
dChristmasDay           : DATE;

bNewYearsDay            : BOOL;
bMartinLutherKingJrDay  : BOOL;
bPresidentsDay          : BOOL;
bMemorialDay            : BOOL;
bIndependenceDay        : BOOL;
bLaborDay               : BOOL;
bColumbusDay            : BOOL;
bThanksgivingDay        : BOOL;
bChristmasDay           : BOOL;
```

dxxxxxx: Date of the respective holiday.

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

3.8 Energy Management

3.8.1 FB_MaximumDemandController



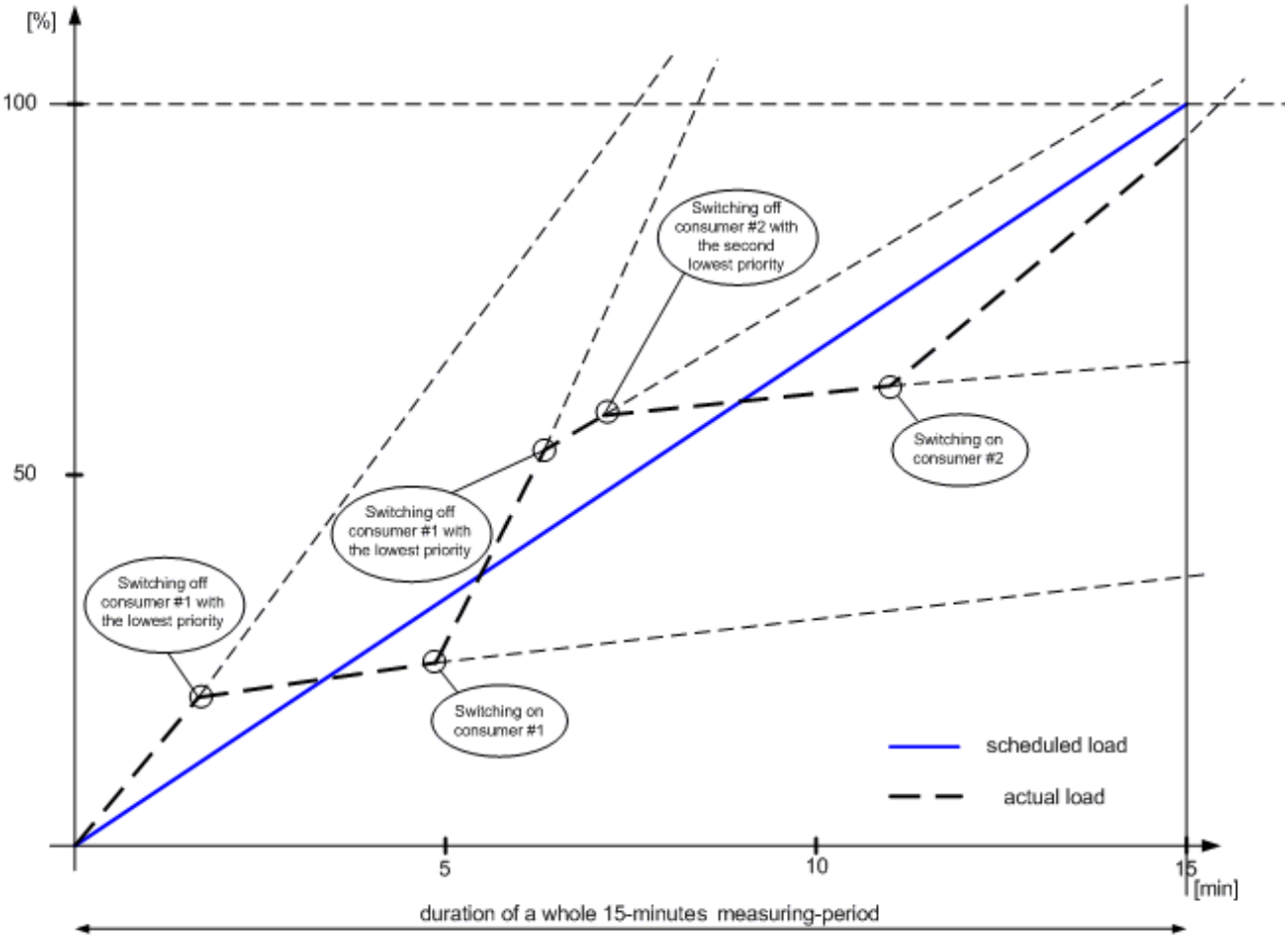
Function block for peak load optimization, which ensures compliance with the set power limit by means of switching on or off up to eight consumers. The consumers can be switched off according to their rated power and priority in such a manner that the production sequence is not disturbed.

To distinguish the individual measurement cycles, a synchronization pulse is supplied by the electricity supply company (ESC). This indicates the start of a new measurement cycle and must be connected to the input *bPeriodPuls*. The actual power is recorded via the counter terminal KL1501.

The block works with a fixed measurement period of 15 minutes. If the synchronization pulse exceeds the 16-minute limit, the output *bEmergencySignal* is set.

All consumers are switched on at the start of each measurement period. If the power limit (fAgreedPower) threatens to be exceeded within the measurement 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

```
bStart              : BOOL;
fMeterConstant      : LREAL;
fAgreedPower        : LREAL;
bPeriodPulse        : BOOL;
arrLoadParameter    : ARRAY[1..8] OF ST_MDCLoadParameters;
```

bStart: The block is activated by a rising edge at this input.

fMeterConstant: Meter constant [pulses / kWh].

fAgreedPower: This is the agreed power limit which, as far as possible, should not be exceeded in the operational case [kW].

bPeriodPulse: Synchronisation pulse sent by the electricity supply company (ESC). This pulse starts the measurement interval.

arrLoadParameter: Parameter structure of the respective consumer. This consists of the following elements:

TYPE ST_MDCLoadParameters: STRUCT bConnected : BOOL; nDegreeOfPriority : INT; tMINPowerOnTime : TIME; tMINPowerOffTime : 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 => lox; ... 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.

VAR_OUTPUT

```
arrLoad                 : ARRAY[1..8] OF BOOL;
fAgreedEnergy           : LREAL;
fInstantaneousEnergy    : LREAL;
fActualEnergy           : LREAL;
tRemainingTime          : TIME;
fLastPeriodEnergy       : LREAL;
bEmergencySignal        : BOOL;
bError                  : BOOL;
nErrorId                : UDINT;
```

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

fAgreedEnergy: Agreed energy consumption [kWh].

fInstantaneousEnergy: Momentary energy consumption [kWh] in relation of the integration period with 15s (internal measurement interval).

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

tRemainingTime: Time remaining until the next measurement interval.

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

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

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

nErrorId: Contains the Errorcode [▶ 89].

VAR_IN_OUT

```
stInDataKL1501      : ST_MDCInDataKL1501;
stOutDataKL1501     : ST_MDCOutDataKL1501;
```

stInDataKL1501: Linked to the KL1501.

```
TYPE ST_MDCInDataKL1501 :
STRUCT
  nStatus    : USINT;
  nDummy1    : USINT;
  nDummy2    : USINT;
  nDummy3    : USINT;
  nData      : DWORD;
END_STRUCT
END_TYPE
```

stOutDataKL1501: Linked to the KL1501.

```
TYPE ST_MDCOutDataKL1501 :
STRUCT
  nCtrl      : USINT;
  nDummy1    : USINT;
  nDummy2    : USINT;
  nDummy3    : USINT;
  nData      : DWORD;
END_STRUCT
END_TYPE
```

Sample

```
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;
```

3.9 Error codes

Value (hex)	Value (dec)	Description
0x0000	0	No error.
0x0001	1	FB_MaximumDemandController(): [▶ 85] -- reserved errorcode --
0x0002	2	FB_MaximumDemandController(): [▶ 85] The input-parameter *fMeterConstant* is "0".
0x0003	3	FB_LightControl(): [▶ 22] The switch-range *nSwitchRange* of the 1st or 2nd element of the table *arrControlTable* is “0”. Thus it´s assumed, that the table has no or only one element.
0x0004	4	FB_LightControl(): [▶ 22] At least one input-value *nActualValue* the table *arrControlTable* lies in the switch range of its neighbour.
0x0005	5	FB_Sequencer(): [▶ 28] The startindex *nStartIndex* is not within the valid range [1..50].
0x0006	6	FB_Sequencer(): [▶ 28] The startindex *nStartIndex* has a reference to a dilimiting-element (zero-entries).
0x0007	7	Scheduler-Modules: At least one input-parameter is not in its valid range.
0x0008	8	Scheduler-Modules: No selection-parameter is set (Weekly-scheduler: selection of weekdays, Monthly-scheduler: selection of the month).
0x0009	9	Scheduler-Modules: A non-existant day of the month was selected.
0x000A	10	FB_ConstantLightControlEco(): [▶ 24] The input-parameter *nMinLevel* is greater or equal then *nMaxLevel.*
0x000B	11	FB_ScenesLighting() [▶ 49], FB_ScenesVenetianBlind(): [▶ 52] The input-parameter *sFile* is invalid (empty).
0x000C	12	FB_ScenesLighting() [▶ 49], FB_ScenesVenetianBlind(): [▶ 52] Internal error: File with the scenes values not found.
0x000D	13	FB_ScenesLighting() [▶ 49], FB_ScenesVenetianBlind(): [▶ 52] Internal error: No more free file handles.

4 Appendix

4.1 Support and Service

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

Beckhoff's branch offices and representatives

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

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

You will also find further documentation for Beckhoff components there.

Beckhoff Support

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

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

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

Beckhoff Service

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

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

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

Beckhoff Headquarters

Beckhoff Automation GmbH & Co. KG

Huelshorstweg 20
33415 Verl
Germany

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

More Information:
www.beckhoff.com/ts8010

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