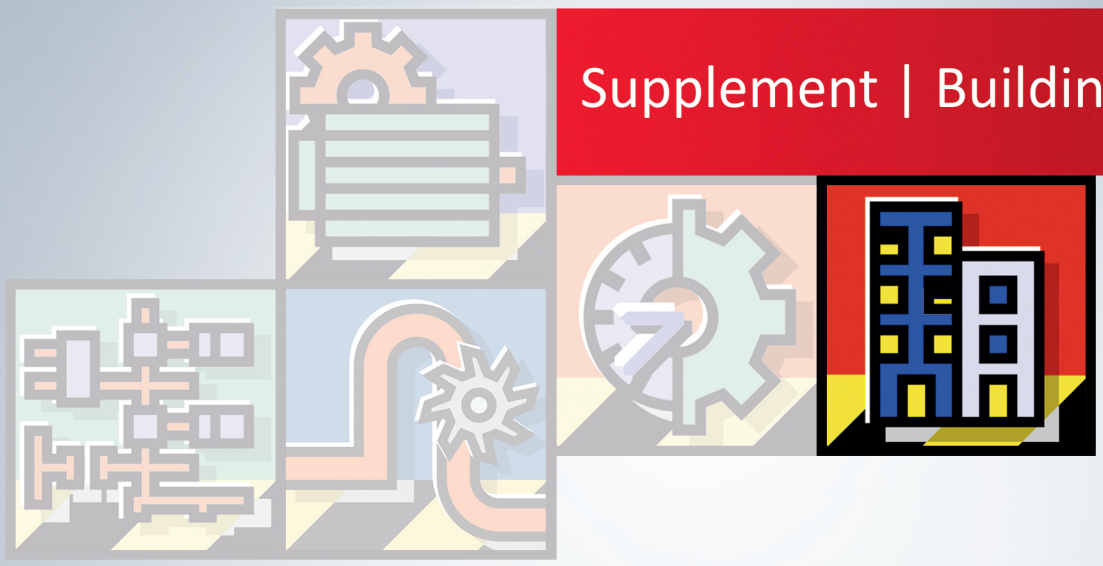
BECKHOFF New Automation Technology

Handbuch | DE

TS8010

TwinCAT 2 | PLC Building Automation Basic



Supplement | Building Automation

27.01.2022 | Version: 1.0

Inhaltsverzeichnis

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.
Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der Dokumentation und der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.
Das Fachpersonal ist verpflichtet, für jede Installation und Inbetriebnahme die zu dem betreffenden Zeitpunkt veröffentliche Dokumentation zu verwenden.

Das Fachpersonal hat sicherzustellen, dass die Anwendung bzw. der Einsatz der beschriebenen Produkte alle Sicherheitsanforderungen, einschließlich sämtlicher anwendbaren Gesetze, Vorschriften, Bestimmungen und Normen erfüllt.

Disclaimer

Diese Dokumentation wurde sorgfältig erstellt. Die beschriebenen Produkte werden jedoch ständig weiter entwickelt.
Wir behalten uns das Recht vor, die Dokumentation jederzeit und ohne Ankündigung zu überarbeiten und zu ändern.
Aus den Angaben, Abbildungen und Beschreibungen in dieser Dokumentation können keine Ansprüche auf Änderung bereits gelieferter Produkte geltend gemacht werden.

Marken

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® und XPlanar® sind eingetragene und lizenzierte Marken der Beckhoff Automation GmbH.
Die Verwendung anderer in dieser Dokumentation enthaltenen Marken oder Kennzeichen durch Dritte kann zu einer Verletzung von Rechten der Inhaber der entsprechenden Bezeichnungen führen.

Patente

Die EtherCAT-Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.

EtherCAT®

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

Copyright

1.2 Sicherheitshinweise

Sicherheitsbestimmungen

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

Haftungsausschluss

Die gesamten Komponenten werden je nach Anwendungsbestimmungen in bestimmten Hard- und Software-Konfigurationen ausgeliefert. Änderungen der Hard- oder Software-Konfiguration, die über die dokumentierten Möglichkeiten hinausgehen, sind unzulässig und bewirken den Haftungsausschluss der Beckhoff Automation GmbH & Co. KG.

Qualifikation des Personals

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs-, Automatisierungs- und Antriebstechnik, das mit den geltenden Normen vertraut ist.

Erklärung der Symbole

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

⚠ GEFAHR

Akute Verletzungsgefahr!

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

⚠ WARNUNG

Verletzungsgefahr!

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

⚠ VORSICHT

Schädigung von Personen!

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

HINWEIS

Schädigung von Umwelt oder Geräten

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

Tipp oder Fingerzeig

Dieses Symbol kennzeichnet Informationen, die zum besseren Verständnis beitragen.

1.3 Hinweise zur Informationssicherheit

Die Produkte der Beckhoff Automation GmbH & Co. KG (Beckhoff) sind, sofern sie online zu erreichen sind, mit Security-Funktionen ausgestattet, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen. Trotz der Security-Funktionen sind die Erstellung, Implementierung und ständige Aktualisierung eines ganzheitlichen Security-Konzepts für den Betrieb notwendig, um die jeweilige Anlage, das System, die Maschine und die Netzwerke gegen Cyber-Bedrohungen zu schützen. Die von Beckhoff verkauften Produkte bilden dabei nur einen Teil des gesamtheitlichen Security-Konzepts. Der Kunde ist dafür verantwortlich, dass unbefugte Zugriffe durch Dritte auf seine Anlagen, Systeme, Maschinen und Netzwerke verhindert werden. Letztere sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn entsprechende Schutzmaßnahmen eingerichtet wurden.

Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem https://www.beckhoff.de/secguide.

Die Produkte und Lösungen von Beckhoff werden ständig weiterentwickelt. Dies betrifft auch die Security-Funktionen. Aufgrund der stetigen Weiterentwicklung empfiehlt Beckhoff ausdrücklich, die Produkte ständig auf dem aktuellen Stand zu halten und nach Bereitstellung von Updates diese auf die Produkte aufzuspielen. Die Verwendung veralteter oder nicht mehr unterstützter Produktversionen kann das Risiko von Cyber-Bedrohungen erhöhen.

Um stets über Hinweise zur Informationssicherheit zu Produkten von Beckhoff informiert zu sein, abonnieren Sie den RSS Feed unter https://www.beckhoff.de/secinfo.

2 Allgemeine Informationen

Weitere erforderliche Bibliotheken

Für PC-Systeme (x86) und Embedded-PCs (CXxxxx):

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

Für Busklemmen-Controller der Serie BCxx00:

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

Für Busklemmen-Controller der Serie BCxx50, BCxx20 und BC9191:

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

Für Busklemmen-Controller der Serie BXxx00:

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

Speicherauslastung

Durch Einbinden der Bibliothek wird bereits SPS-Programmspeicher verbraucht. Abhängig vom Applikationsprogramm kann daher der verbleibende Speicher nicht ausreichend sein.

3 PLC-API

Die TwinCAT PLC Building Automation Bibliothek beinhaltet nützliche Funktionsbausteine zur Automatisierung von Gebäuden.

Beleuchtung

Name	Beschreibung
FB_Dimmer1Switch [▶ 11]	Lichtdimmer über einem Taster.
FB_Dimmer1SwitchEco [▶ 13]	Speicherplatzsparende Variante des FB_Dimmer1Switch() ohne Sonderfunktionen.
FB_Dimmer2Switch [▶ 15]	Lichtdimmer über zwei Taster.
FB_Dimmer2SwitchEco [▶ 17]	Speicherplatzsparende Variante des FB_Dimmer2Switch() ohne Sonderfunktionen.
FB_Dimmer3Switch [▶ 19]	Kombination aus den Bausteinen FB_Dimmer1Switch() und FB_Dimmer2Switch().
FB_Light [▶ 21]	Lichtsteuerung.
FB_LightControl [▶ 22]	tageslichtabhängige Lichtsteuerung.
FB_ConstantLightControlEco [▶ 24]	Konstantlichtregelung.
FB_Ramp [▶ 27]	Rampenbaustein.
FB_Sequencer [▶ 29]	Licht-Sequenz-Baustein.
FB_StairwellDimmer [▶ 32]	Treppenhausdimmer.
FB_StairwellLight [▶ 34]	Treppenhausbeleuchtung.

Fassade

Name	Beschreibung
FB_RoofWindow [▶ 34]	Dachfenster-Steuerung.
FB_VenetianBlind [▶ 36]	Jalousiesteuerung.
FB_VenetianBlindEx [▶ 38]	Jalousiesteuerung mit direkter Positionsvorgabe.
FB_VenetianBlindEx1Switch [▶ 41]	Jalousiesteuerung mit direkter Positionsvorgabe aber nur einem Schaltereingang.

Szenen Management

Name	Beschreibung
FB_RoomOperation [▶ 44]	Baustein zum Aufrufen und Verändern von Szenen über Taster. *)
FB_ScenesLighting [▶ 49]	Baustein zum Verwalten von Lichtszenen. *)
FB_ScenesVenetianBlind [▶ 52]	Baustein zum Verwalten von Jalousieszenen. *)

Signalverarbeitung

Name	Beschreibung
FB_ShortLongClick [▶ 54]	Unterscheidung zwischen kurzen und langen Tastendruck.
FB_SignallingContact [▶ 55]	Meldekontakt.
FB_SingleDoubleClick [▶ 56]	Unterscheidung zwischen einzelnen und doppelten Tastendruck.
FB_ThresholdSwitch [▶ 57]	Schwellwertschalter.

Filterfunktionen

Name	Beschreibung
FB_PT1 [▶ 58]	PT1-Glied zur Glättung von Eingangsgrößen.
FB_PT2 [▶ 60]	PT2-Glied zur Glättung von Eingangsgrößen.

Umrechnungsfunktionen

Name	Beschreibung
F_Scale [▶ 63]	Skalierung / Umwandlung von Rohwerte in Messwerte.
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]	Temperaturumrechnungsfunktionen zwischen Kelvin, Celsius, Reaumur und Fahrenheit.

Zeitschaltfunktionen

Name	Beschreibung
FB_WeeklyTimeSwitch [▶ 76]	Wochenzeitschaltuhr.
FB_CalcSunPosition [▶ 78]	Berechnung der Sonnenhöhe und das Sonnenazimut.
FB_CalcSunriseSunset [▶ 79]	Berechnung Sonnenauf- und Untergang. *)
FB_CalcPublicHolidaysDE [▶ 81]	Berechnung der deutschen Feiertage.
FB_CalcPublicHolidaysUS [▶ 83]	Berechnung der nordamerikanischen Feiertage.
FB_CalcFederalHolidaysUS [▶ 85]	Berechnung der nordamerikanischen Bundes Feiertage.
FB_DailyScheduler [▶ 66]	schaltet jeden n-ten Tag.
FB_WeeklyScheduler [▶ 68]	schaltet jede n-te Woche an bestimmten Wochentagen (Mehrfachauswahl möglich).
FB_MonthlyScheduler1 [▶ 69]	schaltet in bestimmten Monaten (Mehrfachauswahl möglich) an einem bestimmten Wochentag.
FB_MonthlyScheduler2 [▶ 70]	schaltet in bestimmten Monaten (Mehrfachauswahl möglich) an einem bestimmten Tag im Monat.
FB_YearlyScheduler [▶ 72]	schaltet an einem bestimmten Tag im Jahr.

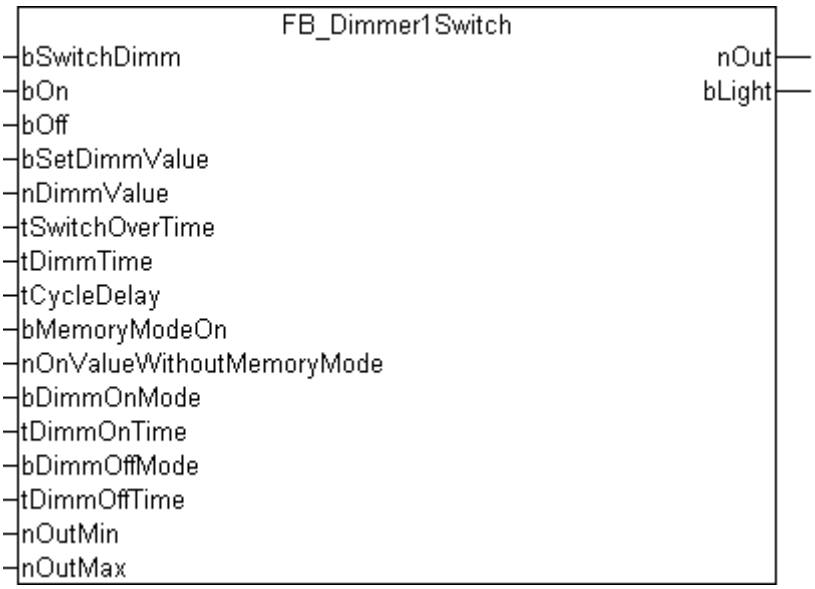
Energiemanagement

Name	Beschreibung
FB_MaximumDemandController [▶ 86]	Maximumwächter zur Reduzierung von Leistungsspitzen.

*) Dieser Baustein ist nur in der PC-Version der Library verfügbar.

3.1 Beleuchtung

3.1.1 FB_Dimmer1Switch



Beschreibung

Bedienung über den Eingang *bSwitchDimm*

Durch ein kurzes Signal am Eingang *bSwitchDimm*, wird das Licht ein- oder ausgeschaltet. Liegt das Signal länger als *tSwitchOverTime* (empfohlener Richtwert: 200ms) an, so wird in den Dimmermodus umgeschaltet. Das Ausgangssignal fährt zyklisch zwischen *nOutMin* und *nOutMax*. Um den maximalen oder minimalen Wert besser einstellen zu können, verweilt das Ausgangssignal für die Zeit *tCycleDelay* auf dem minimalen bzw. maximalen Wert. Wird das Signal wieder weggenommen, so bleibt das aktuelle Ausgangssignal anstehen. Durch einen erneuten Impuls auf den Eingang, wird der Ausgang auf 0 gesetzt.

Bedienung über die Eingänge *bOn* und *bOff*

Wird eine positive Flanken an den Eingängen *bOn* oder *bOff* angelegt, so wird das Licht direkt ein- oder ausgeschaltet. Z.B. für globale Ein-/Aus Funktionen. Beim Ausschalten wird der Ausgangswert auf 0 gesetzt. Das Verhalten beim Einschalten kann durch die Memoryfunktion beeinflußt werden (siehe unten).

Bedienung über die Eingänge *bSetDimmValue* und *nDimmValue*

Ändert sich der Wert *nDimmValue*, so wird das Signal direkt zum Ausgang durchgegeben. Wichtig ist hierbei das sich der Wert ändert. Durch eine Änderung auf den Wert 0, wird die Beleuchtung ausgeschaltet. Durch eine positive Flanke an den Eingang *bSetDimmValue* wird der Wert *nDimmValue* direkt an den Ausgang ausgegeben. Das direkte Ändern des Ausgangs kann durch ein statisches 1-Signal am Eingang *bSetDimmValue* unterdrückt werden. Hierdurch kann ein Wert am Eingang *nDimmValue* angelegt werden, der erst bei der nächsten positiven Flanke, von *bSetDimmValue* an den Ausgang übergeben wird.
Mit Hilfe der Eingänge *bSetDimmValue* und *nDimmValue* können z.B. verschiedene Beleuchtungszenarien realisiert werden. Das direkte Setzen des Ausgangs, mit Hilfe von *nDimmValue*, kann dazu benutzt werden, um bestimmte Helligkeiten anzufahren. Entweder direkt oder durch kontinuierliches Verändern des Wertes. *nDimmValue* muss einen Wert zwischen *nOutMin* und *nOutMax* haben. Ausnahme ist der Wert 0. Liegt der Wert außerhalb des Bereichs, so wird der Ausgangswert auf die obere, bzw. untere Grenze begrenzt.

Memoryfunktion

Beim Einschalten muss unterschieden werden, ob die Memoryfunktion (Eingang *bMemoryModeOn*) aktiv ist oder nicht. Ist die Memoryfunktion aktiv, so wird beim Einschalten der zuletzt eingestellte Wert an den Ausgang angelegt. Ist die Memoryfunktion nicht aktiv, so wird an den Ausgang der Wert angelegt, der über den Parameter *nOnValueWithoutMemoryMode* vorgegeben wird. Hierbei ist es gleichgültig, ob das Licht über den Eingang *bOn* oder über den Eingang *bSwitchDimm* geschaltet wird. Zu beachten ist, dass der Parameter *nOnValueWithoutMemoryMode* zwischen *nOutMin* und *nOutMax* liegen muß. Ist dieses nicht der Fall, so wird der Ausgangswert auf die untere bzw. obere Grenze angepaßt.

Schnelles Auf-/Abdimmen beim Ein-/Ausschalten

Besonderen Beleuchtungskomfort erreicht man dadurch, dass das Licht nicht plötzlich geschaltet, sondern langsam auf den gewünschten Wert gefahren wird. Durch die beiden Eingänge *bDimmOnMode* und *bDimmOffMode* kann sowohl für das Einschalten als auch für das Ausschalten der Modus aktiviert werden. Mit den Parametern *tDimmOnTime* und *tDimmOffTime* wird die Zeitdauer festgelegt, in der das Licht geschaltet wird. Der Wert bezieht sich immer auf den minimal- und den maximal möglichen Ausgangswert (*nOutMin* und *nOutMax*). Als mögliche Ein-/Ausschaltbefehle können die Eingänge *bOn* und *bOff* dienen, oder ein kurzer Impuls am Eingang *bSwitchDimm*. Wird der Eingang *nDimmValue* auf 0 gesetzt, so wird der Ausgang unverzögert gesetzt. Gleiches gilt auch, wenn durch eine positive Flanke an dem Eingang *bSetDimmValue* der Ausgang gesetzt wird.

Anmerkungen zu den Parametern *tSwitchOverTime* und *tDimmTime*

Wird für den Parameter *tSwitchOverTime* eine Dauer von 0 vorgegeben und für *tDimmTime* ein Wert größer 0, so kann mit dem Eingang *tSwitchDimm* das Licht nur gedimmt werden. Ein Ein-/Ausschalten ist nur mit den Eingängen *bOn* und *bOff* möglich.

Beträgt der Parameter *tDimmTime* 0, so kann mit dem Eingang *bSwitchDimm* das Licht nur ein-/ ausgeschaltet werden. Der Wert von *tSwitchOverTime* ist hierbei ohne Bedeutung.

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: Schaltet oder dimmt den Ausgang.

bOn: Schaltet den Ausgang auf den letzten Ausgangswert oder auf den Wert *nOnValueWithoutMemoryMode*.

bOff: Schaltet den Ausgang auf 0.

bSetDimmValue: Schaltet den Ausgang auf den Wert *nDimmValue*.

nDimmValue: Bei einer Änderung wird der Wert direkt an den Ausgang gelegt.

tSwitchOverTime: Umschaltzeit zwischen Licht ein/aus und Licht dimmen für den Eingang *bSwitchDimm.*

tDimmTime: Zeitdauer für das Dimmen vom minimalen Wert bis zum maximalen Wert.

tCycleDelay: Wartezeit, wenn der min- bzw. max-Wert erreicht ist.

bMemoryModeOn: Schaltet auf Memoryfunktion um, so das beim Einschalten der vorherige Wert an den Ausgang geschrieben wird.

nOnValueWithoutMemoryMode: Einschaltwert, wenn die Memoryfunktion nicht eingeschaltet ist.

bDimmOnMode: Beim Einschalten wird schrittweise der Ausgangswert erhöht.

tDimmOnTime: Zeitdauer, in der das Licht beim Einschalten hoch gefahren wird. *bDimmOnMode* muss aktiv sein.

bDimmOffMode: Beim Ausschalten wird schrittweise der Ausgangswert reduziert

tDimmOffTime: Zeitdauer, in der das Licht beim Ausschalten runter gefahren wird. *bDimmOffMode* muss aktiv sein.

nOutMin: minimaler Ausgabewert.

nOutMax: maximaler Ausgabewert. Ist der Parameter *nOutMin* nicht kleiner als *nOutMax*, so bleibt der Ausgang auf 0.

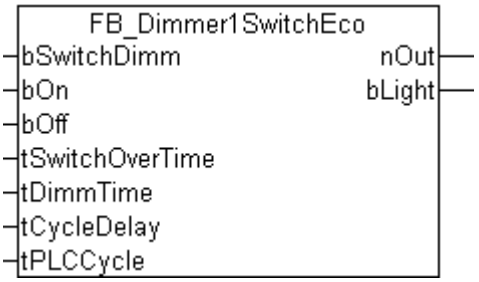
VAR_OUTPUT

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

nOut: analoger Ausgabewert.

bLight: digitaler Ausgabewert. Wird gesetzt, wenn *nOut* größer als 0 ist.

3.1.2 FB_Dimmer1SwitchEco



Beschreibung

Der Baustein FB_Dimmer1SwitchEco stellt die Speicherplatz-sparende Variante des FB_Dimmer1Switch() [▶ 11] dar. Er ist ohne die Sonderfunktionen "Helligkeitswert setzen" und "Memoryfunktion ausschalten" ausgestattet, welche bei vielen Anwendungen u.U. nicht nötig sind. Darüber hinaus sind die Werte *nOutMin* und *nOutMax* des FB_Dimmer1Switch() [▶ 11] hier intern fest auf 0 und 32767 gesetzt. Diese Ausgangsspanne entspricht dem Darstellungsbereich einer analogen Ausgangsklemme. Wichtig ist der Eingang *tPLCCycle.* Über diese Zeit wird intern errechnet, um welchen Betrag der Ausgang *nOut* pro Zyklus erhöht werden muß - das erspart zusätzliche Zeitberechnungen.

Bedienung über den Eingang *bSwitchDimm*

Durch ein kurzes Signal am Eingang *bSwitchDimm*, wird das Licht ein- oder ausgeschaltet. Liegt das Signal länger als *tSwitchOverTime* (empfohlener Richtwert: 200ms) an, so wird in den Dimmermodus umgeschaltet. Das Ausgangssignal fährt zyklisch zwischen 0 und 32767. Um den maximalen oder minimalen Wert besser einstellen zu können, verweilt das Ausgangssignal für die Zeit *tCycleDelay* auf dem minimalen bzw. maximalen Wert. Wird das Signal wieder weggenommen, so bleibt das aktuelle Ausgangssignal anstehen. Durch einen erneuten Impuls auf den Eingang, wird der Ausgang auf 0 gesetzt.

Bedienung über die Eingänge *bOn* und *bOff*

Wird eine positive Flanken an den Eingängen *bOn* oder *bOff* angelegt, so wird das Licht direkt ein- oder ausgeschaltet. Z.B. für globale Ein-/Aus Funktionen. Beim Ausschalten wird der Ausgangswert auf 0 gesetzt.

Memoryfunktion

Im Gegensatz zum FB_Dimmer1Switch() [▶ 11], bei der die Memoryfunktion über den Eingang *bMemoryModeOn* aktiviert oder ausgeschaltet werden kann, ist bei dieser Speicherplatz-sparenden Version die Memoryfunktion immer aktiv. Das bedeutet, dass beim Einschalten der zuletzt eingestellte Wert als Helligkeitswert übernommen wird. Hierbei ist es gleichgültig, ob das Licht über den Eingang *bOn* oder über den Eingang *bSwitchDimm* geschaltet wird.

Anmerkung zum Parameter *tSwitchOverTime*

Wird für den Parameter *tSwitchOverTime* eine Dauer von 0 vorgegeben, so kann mit dem Eingang *bSwitchDimm* das Licht nur gedimmt werden. Ein Ein-/Ausschalten ist nur mit den Eingängen *bOn* und *bOff* möglich.

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: Schaltet oder dimmt den Ausgang.

bOn: Schaltet den Ausgang auf den letzten Ausgangswert oder auf den Wert *nOnValueWithoutMemoryMode*.

bOff: Schaltet den Ausgang auf 0.

tSwitchOverTime: Umschaltzeit zwischen Licht ein/aus und Licht dimmen für den Eingang *bSwitchDimm.*

tDimmTime: Zeitdauer für das Dimmen vom minimalen Wert bis zum maximalen Wert.

tCycleDelay: Wartezeit, wenn der min- bzw. max-Wert erreicht ist.

tPLCCycle: Eingestellte PLC-Zykluszeit.

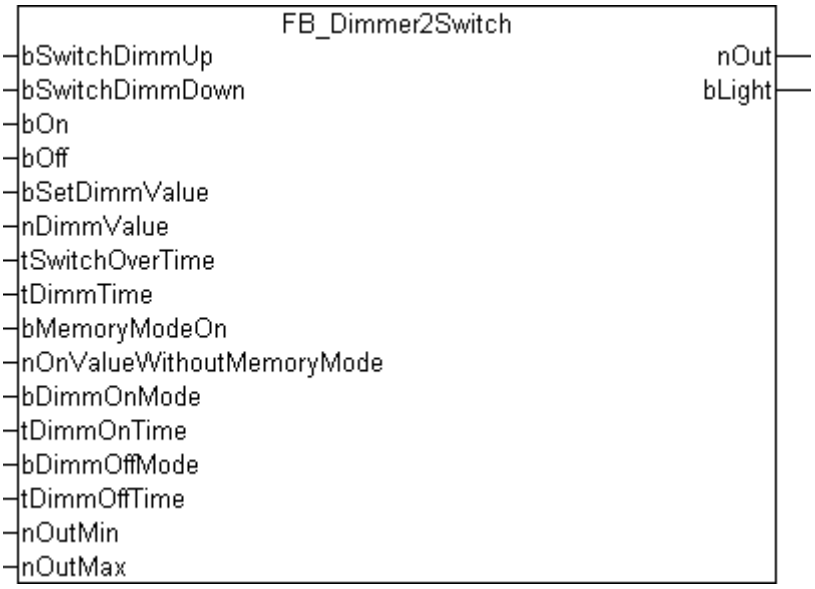
VAR_OUTPUT

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

nOut: analoger Ausgabewert.

bLight: digitaler Ausgabewert. Wird gesetzt, wenn *nOut* größer als 0 ist.

3.1.3 FB_Dimmer2Switch



Beschreibung

Der Funktionsbaustein FB_Dimmer2Switch entspricht im Funktionsumfang dem Funktionsbaustein FB_Dimmer1Switch() [▶ 11]. Der Unterschied besteht darin, dass bei dem Funktionsbaustein FB_Dimmer2Switch zwei Schalter angeschlossen werden. Hierdurch kann der Bediener gezielt Auf- oder Abdimmen.

Bedienung über die Eingänge *bSwitchDimmUp* und *bSwitchDimmDown*

Durch ein kurzes Signal am Eingang *bSwitchDimmUp* bzw. *bSwitchDimmDown*, wird das Licht ein- oder ausgeschaltet. Liegt das Signal länger als *tSwitchOverTime* (empfohlener Richtwert: 200ms) an, so wird in den Dimmermodus umgeschaltet. Das Ausgangssignal fährt auf *nOutMin* bzw. *nOutMax*. Wird das Signal wieder weggenommen, so bleibt das aktuelle Ausgangssignal anstehen. Durch einen erneuten Impuls auf einen der Eingänge, wird der Ausgang auf 0 gesetzt.

Bedienung über die Eingänge *bOn* und *bOff*

Wird eine positive Flanken an den Eingängen *bOn* oder *bOff* angelegt, so wird das Licht direkt ein- oder ausgeschaltet. Z.B. für globale Ein-/Aus Funktionen. Beim Ausschalten wird der Ausgangswert auf 0 gesetzt. Das Verhalten beim Einschalten kann durch die Memoryfunktion beeinflußt werden (siehe unten).

Bedienung über die Eingänge *bSetDimmValue* und *nDimmValue*

Ändert sich der Wert *nDimmValue*, so wird das Signal direkt zum Ausgang durchgegeben. Wichtig ist hierbei das sich der Wert ändert. Durch eine Änderung auf den Wert 0, wird die Beleuchtung ausgeschaltet. Durch eine positive Flanke an den Eingang *bSetDimmValue* wird der Wert *nDimmValue* direkt an den Ausgang ausgegeben. Das dirkete Ändern des Ausgangs kann durch ein statisches 1-Signal am Eingang *bSetDimmValue* unterdrückt werden. Hierdurch kann ein Wert am Eingang *nDimmValue* angelegt werden, der erst bei der nächsten positiven Flanke von *bSetDimmValue* an den Ausgang übergeben wird.
Mit Hilfe der Eingänge *bSetDimmValue* und *nDimmValue* können z.B. verschiedene Beleuchtungszenarien realisiert werden. Das direkte Setzen des Ausgangs, mit Hilfe von *nDimmValue*, kann dazu benutzt werden bestimmte Helligkeiten anzufahren. Entweder direkt oder durch kontinuierliches Verändern des Wertes. *nDimmValue* muss einen Wert zwischen *nOutMin* und *nOutMax* haben. Ausnahme ist der Wert 0. Liegt der Wert außerhalb des Bereichs, so wird der Ausgangswert auf die obere, bzw. untere Grenze begrenzt.

Memoryfunktion

Beim Einschalten muss unterschieden werden, ob die Memoryfunktion (Eingang *bMemoryModeOn*) aktiv ist oder nicht. Ist die Memoryfunktion aktiv, so wird beim Einschalten der zuletzt eingestellte Wert an den Ausgang angelegt. Ist die Memoryfunktion nicht aktiv, so wird an den Ausgang der Wert angelegt, der über den Parameter *nOnValueWithoutMemoryMode* vorgegeben wird. Hierbei ist es gleichgültig, ob das Licht über den Eingang *bOn* oder über den Eingang *bSwitchDimmUp* bzw. *bSwitchDimmDown* geschaltet wird. Zu beachten ist, dass der Parameter *nOnValueWithoutMemoryMode* zwischen *nOutMin* und *nOutMax* liegen muß. Ist dieses nicht der Fall, so wird der Ausgangswert auf die untere bzw. obere Grenze angepaßt.

Schnelles Auf-/Abdimmen beim Ein-/Ausschalten

Besonderen Beleuchtungskomfort erreicht man dadurch, dass das Licht nicht plötzlich geschaltet, sondern langsam auf den gewünschten Wert gefahren wird. Durch die beiden Eingänge *bDimmOnMode* und *bDimmOffMode* kann sowohl für das Einschalten, als auch für das Ausschalten der Modus aktiviert werden. Mit den Parametern *tDimmOnTime* und *tDimmOffTime* wird die Zeitdauer festgelegt, in der das Licht geschaltet wird. Der Wert bezieht sich immer auf den minimal- und den maximal möglichen Ausgangswert (*nOutMin* und *nOutMax*). Als mögliche Ein-/Ausschaltbefehle können die Eingänge *bOn* und *bOff* dienen, oder ein kurzer Impuls am Eingang *bSwitchDimmUp* bzw. *bSwitchDimmDown*. Wird der Eingang *nDimmValue* auf 0 gesetzt, so wird der Ausgang unverzögert gesetzt. Gleiches gilt auch, wenn durch eine positive Flanke an dem Eingang *bSetDimmValue* der Ausgang gesetzt wird.

Anmerkungen zu den Parametern *tSwitchOverTime* und *tDimmTime*

Wird für den Parameter *tSwitchOverTime* eine Dauer von 0 vorgegeben und für *tDimmTime* ein Wert größer 0, so kann mit dem Eingang *bSwitchDimmUp* bzw. *bSwitchDimmDown* das Licht nur gedimmt werden. Ein Ein-/Ausschalten ist nur mit den Eingängen *bOn* und *bOff* möglich.

Beträgt der Parameter *tDimmTime* 0, so kann mit dem Eingang *bSwitchDimmUp* bzw. *bSwitchDimmDown* das Licht nur ein-/ausgeschaltet werden. Der Wert von *tSwitchOverTime* ist hierbei ohne Bedeutung.

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: Schaltet oder dimmt den Ausgang Auf.

bSwitchDimmDown: Schaltet oder dimmt den Ausgang Ab.

bOn: Schaltet den Ausgang auf den letzten Ausgangswert oder auf den Wert *nOnValueWithoutMemoryMode*.

bOff: Schaltet den Ausgang auf 0.

bSetDimmValue: Schaltet den Ausgang auf den Wert *nDimmValue*.

nDimmValue: Bei einer Änderung wird der Wert direkt an den Ausgang gelegt.

tSwitchOverTime: Umschaltzeit zwischen Licht ein/aus und Licht dimmen für den Eingang *bSwitchDimmUp* und *bSwitchDimmDown.*

tDimmTime: Zeitdauer für das Dimmen vom minimalen Wert bis zum maximalen Wert.

bMemoryModeOn: Schaltet auf Memoryfunktion um, so das beim Einschalten der vorherige Wert an den Ausgang geschrieben wird.

nOnValueWithoutMemoryMode: Einschaltwert, wenn die Memoryfunktion nicht eingeschaltet ist.

bDimmOnMode: Beim Einschalten wird schrittweise der Ausgangswert erhöht.

tDimmOnTime: Zeitdauer, in der das Licht beim Einschalten hochgefahren wird. *bDimmOnMode* muss aktiv sein.

bDimmOffMode: Beim Ausschalten wird schrittweise der Ausgangswert reduziert

tDimmOffTime: Zeitdauer, in der das Licht beim Ausschalten runtergefahren wird. *bDimmOffMode* muss aktiv sein.

nOutMin: minimaler Ausgabewert.

nOutMax: maximaler Ausgabewert. Ist der Parameter *nOutMin* nicht kleiner als *nOutMax*, so bleibt der Ausgang auf 0.

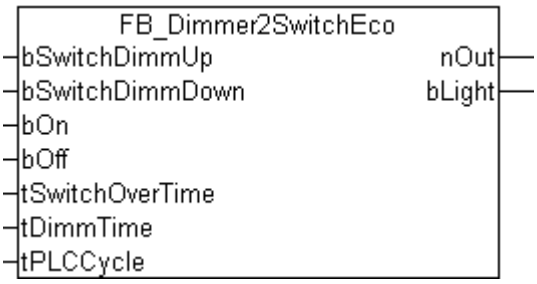
VAR_OUTPUT

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

nOut: analoger Ausgabewert.

bLight: digitaler Ausgabewert. Wird gesetzt, wenn *nOut* größer als 0 ist.

3.1.4 FB_Dimmer2SwitchEco



Beschreibung

Der Baustein FB_Dimmer2SwitchEco stellt die Speicherplatz-sparende Variante des FB_Dimmer2Switch() [▶ 15] dar. Er ist ohne die Sonderfunktionen "Helligkeitswert setzen" und "Memoryfunktion ausschalten" ausgestattet, welche bei vielen Anwendungen u.U. nicht nötig sind. Darüber hinaus sind die Werte *nOutMin* und *nOutMax* des FB_Dimmer2Switch() [▶ 15] hier intern fest auf 0 und 32767 gesetzt. Diese Ausgangsspanne entspricht dem Darstellungsbereich einer analogen Ausgangsklemme. Wichtig ist der Eingang *tPLCCycle.* Über diese Zeit wird intern errechnet, um welchen Betrag der Ausgang *nOut* pro Zyklus erhöht werden muß - das erspart zusätzliche Zeitberechnungen.

Bedienung über die Eingänge *bSwitchDimmUp* und *bSwitchDimmDown*

Durch ein kurzes Signal am Eingang *bSwitchDimmUp* bzw. *bSwitchDimmDown*, wird das Licht ein- oder ausgeschaltet. Liegt das Signal länger als *tSwitchOverTime* (empfohlener Richtwert: 200ms) an, so wird in den Dimmermodus umgeschaltet. Das Ausgangssignal fährt auf *nOutMin* bzw. *nOutMax*. Wird das Signal wieder weggenommen, so bleibt das aktuelle Ausgangssignal anstehen. Durch einen erneuten Impuls auf einen der Eingänge, wird der Ausgang auf 0 gesetzt.

Bedienung über die Eingänge *bOn* und *bOff*

Wird eine positive Flanken an den Eingängen *bOn* oder *bOff* angelegt, so wird das Licht direkt ein- oder ausgeschaltet. Z.B. für globale Ein-/Aus Funktionen. Beim Ausschalten wird der Ausgangswert auf 0 gesetzt.

Memoryfunktion

Im Gegensatz zum FB_Dimmer2Switch() [▶ 15], bei der die Memoryfunktion über den Eingang *bMemoryModeOn* aktiviert oder ausgeschaltet werden kann, ist bei dieser Speicherplatz-sparenden Version die Memoryfunktion immer aktiv. Das bedeutet, daß beim Einschalten der zuletzt eingestellte Wert als Helligkeitswert übernommen wird. Hierbei ist es gleichgültig, ob das Licht über den Eingang *bOn* oder über den Eingang *bSwitchDimmUp* bzw. *bSwitchDimmDown* geschaltet wird.

Anmerkung zum Parameter *tSwitchOverTime*

Wird für den Parameter *tSwitchOverTime* eine Dauer von 0 vorgegeben, so kann mit dem Eingang *bSwitchDimmUp* bzw. *bSwitchDimmDown*das Licht nur gedimmt werden. Ein Ein-/Ausschalten ist nur mit den Eingängen *bOn* und *bOff* möglich.

VAR_INPUT

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

bSwitchDimmUp: Schaltet oder dimmt den Ausgang Auf.

bSwitchDimmDown: Schaltet oder dimmt den Ausgang Ab.

bOn: Schaltet den Ausgang auf den letzten Ausgangswert.

bOff: Schaltet den Ausgang auf 0.

tSwitchOverTime: Umschaltzeit zwischen Licht ein/aus und Licht dimmen für den Eingang *bSwitchDimmUp* und *bSwitchDimmDown.*

tDimmTime: Zeitdauer für das Dimmen vom minimalen Wert bis zum maximalen Wert.

tCycleDelay: Wartezeit, wenn der min- bzw. max-Wert erreicht ist.

tPLCCycle: Eingestellte PLC-Zykluszeit.

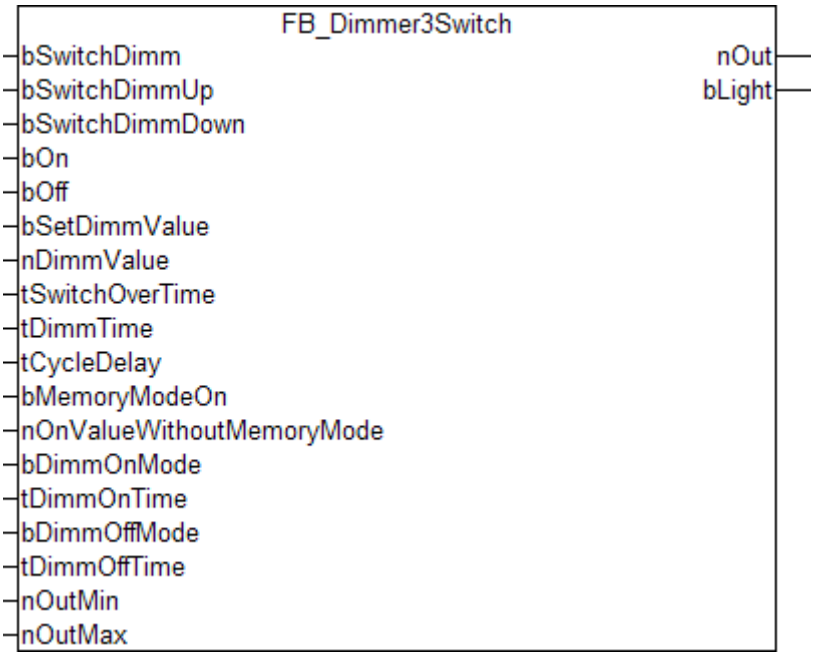
VAR_OUTPUT

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

nOut: analoger Ausgabewert.

bLight: digitaler Ausgabewert. Wird gesetzt, wenn *nOut* größer als 0 ist.

3.1.5 FB_Dimmer3Switch



Beschreibung

Der Funktionsbaustein FB_Dimmer3Switch entspricht im Funktionsumfang den Funktionsbausteinen FB_Dimmer1Switch() [▶ 11] und FB_Dimmer2Switch() [▶ 15].

Bedienung über den Eingang *bSwitchDimm*

Durch ein kurzes Signal am Eingang *bSwitchDimm*, wird das Licht ein- oder ausgeschaltet. Liegt das Signal länger als *tSwitchOverTime* (empfohlener Richtwert: 200ms) an, so wird in den Dimmermodus umgeschaltet. Das Ausgangssignal fährt zyklisch zwischen *nOutMin* und *nOutMax*. Um den maximalen oder minimalen Wert besser einstellen zu können, verweilt das Ausgangssignal für die Zeit *tCycleDelay* auf dem minimalen bzw. maximalen Wert. Wird das Signal wieder weggenommen, so bleibt das aktuelle Ausgangssignal anstehen. Durch einen erneuten Impuls auf den Eingang, wird der Ausgang auf 0 gesetzt.

Bedienung über die Eingänge *bSwitchDimmUp* und *bSwitchDimmDown*

Durch ein kurzes Signal am Eingang *bSwitchDimmUp* bzw. *bSwitchDimmDown*, wird das Licht ein- oder ausgeschaltet. Liegt das Signal länger als *tSwitchOverTime* (empfohlener Richtwert: 200ms) an, so wird in den Dimmermodus umgeschaltet. Das Ausgangssignal fährt auf *nOutMin* bzw. *nOutMax*. Wird das Signal wieder weggenommen, so bleibt das aktuelle Ausgangssignal anstehen. Durch einen erneuten Impuls auf einen der Eingänge, wird der Ausgang auf 0 gesetzt.

Bedienung über die Eingänge *bOn* und *bOff*

Wird eine positive Flanken an den Eingängen *bOn* oder *bOff* angelegt, so wird das Licht direkt ein- oder ausgeschaltet. Z.B. für globale Ein-/Aus Funktionen. Beim Ausschalten wird der Ausgangswert auf 0 gesetzt. Das Verhalten beim Einschalten kann durch die Memoryfunktion beeinflußt werden (siehe unten).

Bedienung über die Eingänge *bSetDimmValue* und *nDimmValue*

Ändert sich der Wert *nDimmValue*, so wird das Signal direkt zum Ausgang durchgegeben. Wichtig ist hierbei das sich der Wert ändert. Durch eine Änderung auf den Wert 0, wird die Beleuchtung ausgeschaltet. Durch eine positive Flanke an den Eingang *bSetDimmValue* wird der Wert *nDimmValue* direkt an den Ausgang ausgegeben. Das dirkete Ändern des Ausgangs kann durch ein statisches 1-Signal am Eingang *bSetDimmValue* unterdrückt werden. Hierdurch kann ein Wert am Eingang *nDimmValue* angelegt werden, der erst bei der nächsten positiven Flanke von *bSetDimmValue* an den Ausgang übergeben wird.
Mit Hilfe der Eingänge *bSetDimmValue* und *nDimmValue* können z.B. verschiedene Beleuchtungszenarien

realisiert werden. Das direkte Setzen des Ausgangs, mit Hilfe von *nDimmValue*, kann dazu benutzt werden bestimmte Helligkeiten anzufahren. Entweder direkt oder durch kontinuierliches Verändern des Wertes. *nDimmValue* muss einen Wert zwischen *nOutMin* und *nOutMax* haben. Ausnahme ist der Wert 0. Liegt der Wert außerhalb des Bereichs, so wird der Ausgangswert auf die obere, bzw. untere Grenze begrenzt.

Memoryfunktion

Beim Einschalten muss unterschieden werden, ob die Memoryfunktion (Eingang *bMemoryModeOn*) aktiv ist oder nicht. Ist die Memoryfunktion aktiv, so wird beim Einschalten der zuletzt eingestellte Wert an den Ausgang angelegt. Ist die Memoryfunktion nicht aktiv, so wird an den Ausgang der Wert angelegt, der über den Parameter *nOnValueWithoutMemoryMode* vorgegeben wird. Hierbei ist es gleichgültig, ob das Licht über den Eingang *bOn* oder über den Eingang *bSwitchDimmUp* bzw. *bSwitchDimmDown* geschaltet wird. Zu beachten ist, dass der Parameter *nOnValueWithoutMemoryMode* zwischen *nOutMin* und *nOutMax* liegen muß. Ist dieses nicht der Fall, so wird der Ausgangswert auf die untere bzw. obere Grenze angepaßt.

Schnelles Auf-/Abdimmen beim Ein-/Ausschalten

Besonderen Beleuchtungskomfort erreicht man dadurch, dass das Licht nicht plötzlich geschaltet, sondern langsam auf den gewünschten Wert gefahren wird. Durch die beiden Eingänge *bDimmOnMode* und *bDimmOffMode* kann sowohl für das Einschalten als auch für das Ausschalten der Modus aktiviert werden. Mit den Parametern *tDimmOnTime* und *tDimmOffTime* wird die Zeitdauer festgelegt, in der das Licht geschaltet wird. Der Wert bezieht sich immer auf den minimal- und den maximal möglichen Ausgangswert (*nOutMin* und *nOutMax*). Als mögliche Ein-/Ausschaltbefehle können die Eingänge *bOn* und *bOff* dienen, oder ein kurzer Impuls am Eingang *bSwitchDimmUp* bzw. *bSwitchDimmDown*. Wird der Eingang *nDimmValue* auf 0 gesetzt, so wird der Ausgang unverzögert gesetzt. Gleiches gilt auch, wenn durch eine positive Flanke an dem Eingang *bSetDimmValue* der Ausgang gesetzt wird.

Anmerkungen zu den Parametern *tSwitchOverTime* und *tDimmTime*

Wird für den Parameter *tSwitchOverTime* eine Dauer von 0 vorgegeben und für *tDimmTime* ein Wert größer 0, so kann mit dem Eingang *bSwitchDimmUp* bzw. *bSwitchDimmDown* das Licht nur gedimmt werden. Ein Ein-/Ausschalten ist nur mit den Eingängen *bOn* und *bOff* möglich.

Beträgt der Parameter *tDimmTime* 0, so kann mit dem Eingang *bSwitchDimmUp* bzw. *bSwitchDimmDown* das Licht nur ein-/ausgeschaltet werden. Der Wert von *tSwitchOverTime* ist hierbei ohne Bedeutung.

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: Schaltet oder dimmt den Ausgang.

bSwitchDimmUp: Schaltet oder dimmt den Ausgang Auf.

bSwitchDimmDown: Schaltet oder dimmt den Ausgang Ab.

bOn: Schaltet den Ausgang auf den letzten Ausgangswert oder auf den Wert *nOnValueWithoutMemoryMode*.

bOff: Schaltet den Ausgang auf 0.

bSetDimmValue: Schaltet den Ausgang auf den Wert *nDimmValue*.

nDimmValue: Bei einer Änderung wird der Wert direkt an den Ausgang gelegt.

tSwitchOverTime: Umschaltzeit zwischen Licht ein/aus und Licht dimmen für den Eingang *bSwitchDimmUp* und *bSwitchDimmDown.*

tDimmTime: Zeitdauer für das Dimmen vom minimalen Wert bis zum maximalen Wert.

tCycleDelay: Wartezeit, wenn der min- bzw. max-Wert erreicht ist.

bMemoryModeOn: Schaltet auf Memoryfunktion um, so das beim Einschalten der vorherige Wert an den Ausgang geschrieben wird.

nOnValueWithoutMemoryMode: Einschaltwert, wenn die Memoryfunktion nicht eingeschaltet ist.

bDimmOnMode: Beim Einschalten wird schrittweise der Ausgangswert erhöht.

tDimmOnTime: Zeitdauer, in der das Licht beim Einschalten hochgefahren wird. *bDimmOnMode* muss aktiv sein.

bDimmOffMode: Beim Ausschalten wird schrittweise der Ausgangswert reduziert

tDimmOffTime: Zeitdauer, in der das Licht beim Ausschalten runtergefahren wird. *bDimmOffMode* muss aktiv sein.

nOutMin: minimaler Ausgabewert.

nOutMax: maximaler Ausgabewert. Ist der Parameter *nOutMin* nicht kleiner als *nOutMax*, so bleibt der Ausgang auf 0.

VAR_OUTPUT

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

nOut: analoger Ausgabewert.

bLight: digitaler Ausgabewert. Wird gesetzt, wenn *nOut* größer als 0 ist.

3.1.6 FB_Light

Durch eine positive Flanke an dem Eingang *bOn* wird der Ausgang *bLight* gesetzt. Zurückgesetzt wird der Ausgang durch eine positive Flanke an dem Eingang *bOff*. Wird an *bToggle* eine positive Flanke angelegt, so wird der Ausgang negiert; also von An nach Aus, bzw. von Aus nach An umgeschaltet.

VAR_INPUT

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

bOn: Schaltet den Ausgang ein.

bOff: Schaltet den Ausgang aus.

bToggle: Negiert den Zustand des Ausgangs.

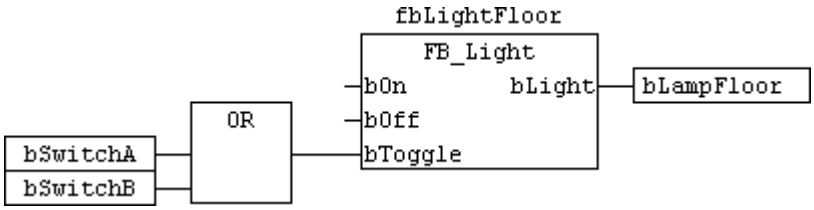
VAR_OUTPUT

```
bLight          : BOOL;
```

bLight: Bei einer positiven Flanke an *bOn* wird der Ausgang gesetzt.

Beispiel 1:

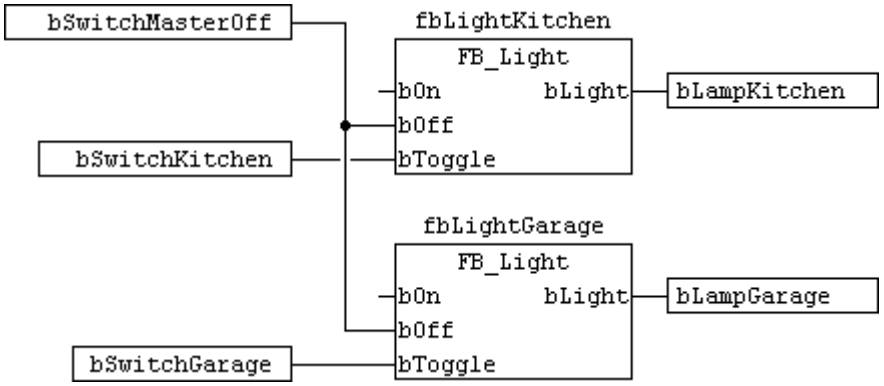
Bei dem folgenden Beispiel wird eine Lampe über zwei Taster geschaltet.



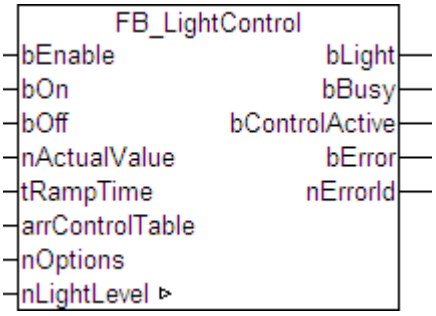
Wird einer der beiden Taster *bSwitchA* oder *bSwitchB* betätigt, so ändert sich an dem Ausgang *bLight* der Zustand der Lampe.

Beispiel 2:

Bei dem folgenden Beispiel werden mit dem Schalter *bSwitchMasterOff* die Lampen *bLampKitchen* und *bLampGarage* zusammen ausgeschaltet. Diese Funktion kann dazu genutzt werden, um z.B. ein Gebäudebereich zentral zu steuern.

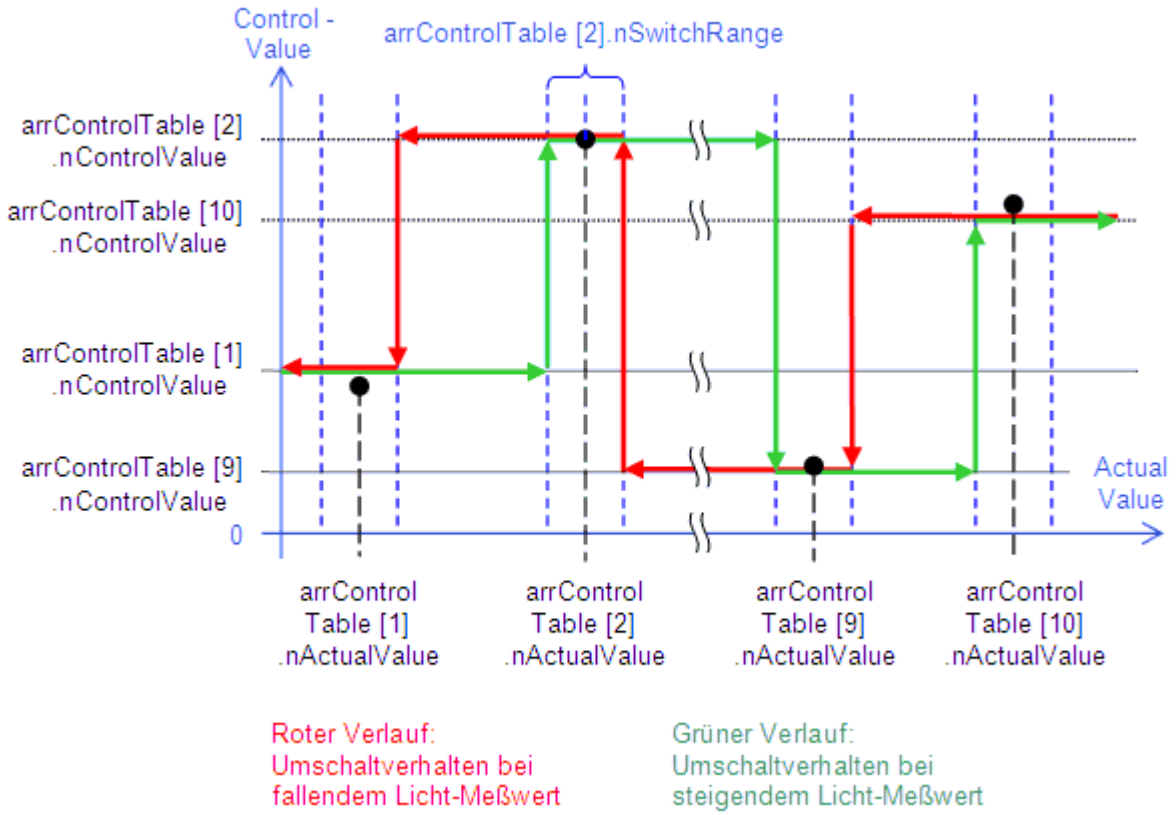


3.1.7 FB_LightControl



Funktionsblock zur tageslichtabhängigen Lichtsteuerung mit bis zu 30 Stützpunkten.

Kernstück dieses Funktionsblockes ist eine Eingangs-/Stellgrößentabelle von 30 Elementen mit einer Schwellwertumschaltung. Erreicht der Eingangswert den Bereich eines Stützpunktes (*arrControlTable[n].nActualValue-arrControlTable[n].nSwitchRange/2 ... arrControlTable[n].nActualValue+arrControlTable[n].nSwitchRange/2*) so springt die Stellgröße auf den entsprechenden Funktionswert *arrControlTable[n].nSetpoint* (siehe Diagramm). Daran angekoppelt ist ein Rampenbaustein, der die Stellgröße in der Zeit *tRampTime* anfährt.
Beim Einschalten mit einer positiven Flanke an *bOn* wird das Licht jedoch zunächst auf die nächstgelegene Stellgröße direkt geschaltet. Erst dann wird die Steuerung aktiviert. Während die Steuerung aktiv ist, kann jederzeit mit einer positiven Flanke an *bOn* "nachgestartet" und damit das Licht auf den nächstgelegene Stellgröße direkt gesteuert werden. Eine positive Flanke an *bOff* schaltet das Licht direkt aus.



Es muß nicht der ganze Bereich der Tabelle genutzt werden. Das erste Tabellenelement (*arrControlTable*, s.u.), bei dem der Parameter *nSwitchrange* eine 0 hat, wird als Beginn des unbenutzten Bereiches angesehen.

VAR_INPUT

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

bEnable: Solange dieser Eingang auf *TRUE* steht, sind die Eingänge *bOn* und *bOff* aktiv. Ein negativer Zustand deaktiviert die Eingänge.

bOn: Eine steigende Flanke schaltet *nLightLevel* direkt auf den nächst gelegene Stellgröße .

bOff: Eine steigende Flanke schaltet *nLightLevel* unmittelbar auf "*0*".

nActualValue: aktuelle Helligkeit.

tRampTime: Zeitdauer, in der der Helligkeitswert auf den nächste Stellgröße gesteuert wird. (Voreingestellter Wert: 30s).

arrControlTable: Eingagswert-/Stellgrößen-Tabelle. *arrControlTable[1]* bis *arrControlTable[30]* vom Typ ST_ControlTable.

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

nActualValue: aktuelle Helligkeit.

nControlValue: Zugehöriger Umschaltpunkt (Stellgröße).

nSwitchRange: Schwellwert um die Eingangswert-Stützstelle bei der umgeschaltet wird. Der Eintrag "0" kennzeichnet den Anfang des nicht genutzten Bereiches der Tabelle.

nOptions: Reserviert für zukünftige Entwicklungen.

VAR_OUTPUT

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

bLight: Dieser Ausgang ist so lange gesetzt, wie *nLightLevel* größer als "*0*" ist.

bBusy: Dieser Ausgang ist immer dann aktiv, solange eine Befehlsabarbeitung (*bOn*, *bOff*, *bToggle* oder Rampenfahrt) aktiv ist.

bControlActive: Dieser Ausgang ist so lange aktiv, wie auch die Regelung aktiv ist.

bError: Dieser Ausgang wird auf *TRUE* geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *nErrorId* enthalten. Wird durch das Ausführen eines Befehls an den Eingängen auf *FALSE* zurückgesetzt.

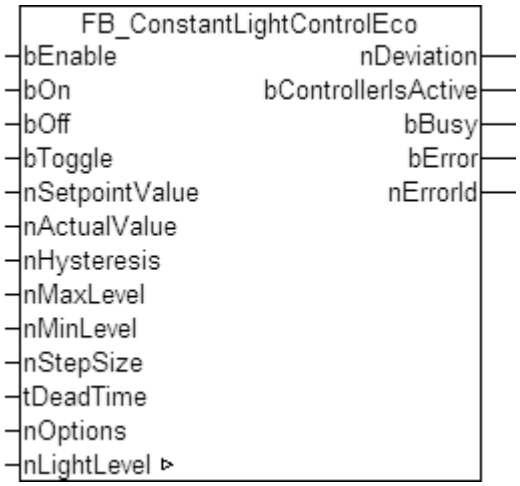
nErrorId: Enthält den spezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf "*0*" zurückgesetzt. Siehe Fehlercodes [▶ 90].

VAR_IN_OUT

```
nLightLevel           : UINT;
```

nLightLevel: Ausgabe-Stellgröße des Bausteines und Verweis auf den Lichtausgangswert.

3.1.8 FB_ConstantLightControlEco



Der Baustein FB_ConstantLightControlEco() dient zur Konstantlichtregelung.

Durch zyklisches Auf- und Abdimmen wird versucht, auf einen vorgegebenen Sollwert zu regeln. Die Dynamik der Regelung wird durch eine Totzeit (*tDeadTime*) sowie der Schrittweite (*nStepSize*) bestimmt. Die Totzeit legt die Wartezeit zwischen den einzelnen Auf- bzw. Abstufungen der Stellgröße um die eingestellte Schrittweite fest. Je kleiner die Totzeit, desto schneller wird nachgeregelt. Eine frei definierbare Hysterese (*nHysteresis*) verhindert ein ständiges Schwingen um den Sollwert. Liegt der Istwert im Bereich der Hysterese um den Sollwert, so wird die Helligkeit der Lampen nicht verändert.

Hinweis: Ist die eingestellte Schrittweite *nStepSize* zu groß bzw. die Hysterese *nHysteresis* zu klein gewählt, kann es passieren, daß der Hysteresebereich nicht "getroffen" wird. Dieses läßt sich nicht durch den Baustein verhindern, da die Lichleistung *nLightLevel* nur im physikalischen Zusammenhang mit aufgenommenen Licht-Istwert, *nActualLevel,* steht.

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: Schaltet den Baustein frei. Ist dieser Eingang auf FALSE, so sind die Eingänge *bOn*, *bOff* und *bToogle* gesperrt. Die Stellgröße bleibt unverändert.

bOn: Schaltet die angesprochenen Geräte auf *nMaxLevel* und aktiviert die Konstantlichtregelung.

bOff: Schaltet die angesprochenen Geräte aus und deaktiviert die Konstantlichtregelung.

bToggle: Je nach Zustand des Referenzgerätes wird die Beleuchtung ein- oder ausgeschaltet.

nSetpointValue: An diesem Eingang wird der Sollwert vorgegeben.

nActualValue: An diesem Eingang wird der Istwert angelegt.

nHysteresis: Regelhysterese um den Sollwert. Liegt der Istwert innerhalb dieses Bereiches, so werden die Stellgrößen der Lampen nicht verändert.

nMaxLevel: Maximalwert der Stellgröße an *nLightLevel*.

nMinLevel: Minmalwert der Stellgröße an *nLightLevel*. Soll dieser Wert durch einen Abdimmvorgang unterschritten werden, so wird *nLightLevel* direkt auf "0" gesetzt. Andersherum, wenn bei aktiver Regelung *nLightLevel* auf "0" steht beginnt der Aufdimmvorgang direkt bei diesem Wert.

nStepSize: Schrittweite mit der die Stellgröße *nLightLevel* verändert wird.

tDeadTime: Totzeit zwischen den einzelnen Auf- bzw. Abstufungen der Stellgröße.

nOptions: Derzeit ohne Verwendung. Reserviert für zukünftige Erweiterungen.

VAR_OUTPUT

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

nDeviation: Aktuelle Regelabweichung (Sollwert - Istwert).

bControllerIsActive: Dieser Ausgang wird gesetzt, sobald die Regelung aktiviert wurde.

bBusy: Bei aktiviertem Baustein ist dieser Ausgang immer dann aktiv, wenn Änderungen der Stellgröße erfolgen.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *nErrorId* enthalten. Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.

nErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 90].

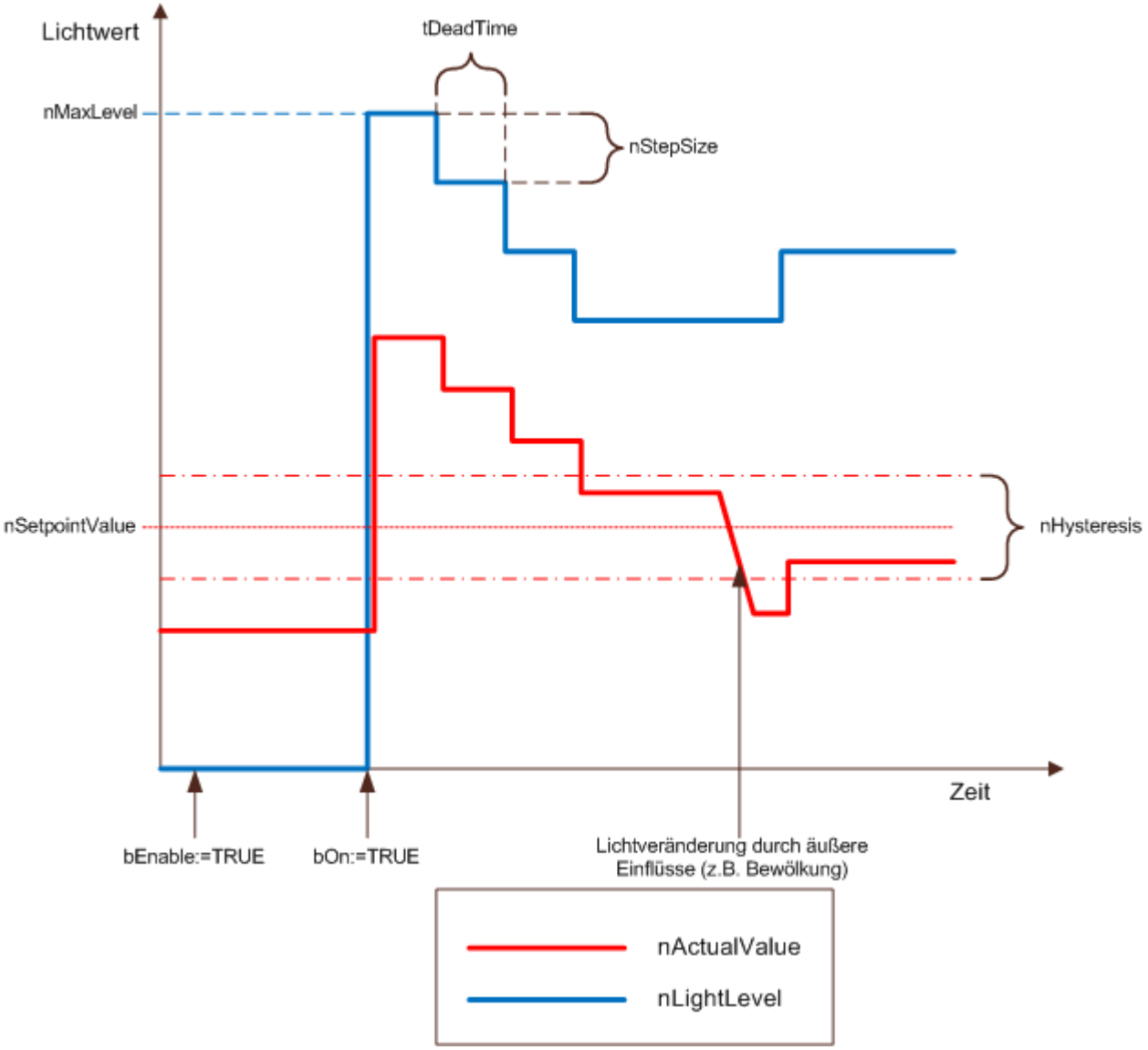
VAR_IN_OUT

```
nLightLevel      : UINT;
```

nLightLevel: Ausgabe-Stellgröße des Bausteines und Verweis auf den Lichtausgangswert. Dieser Wert ist deshalb als IN-Out-Variable definiert, weil der Funktionsbaustein den Lichtwert zum einen ausliest und zum anderen beschreibt.

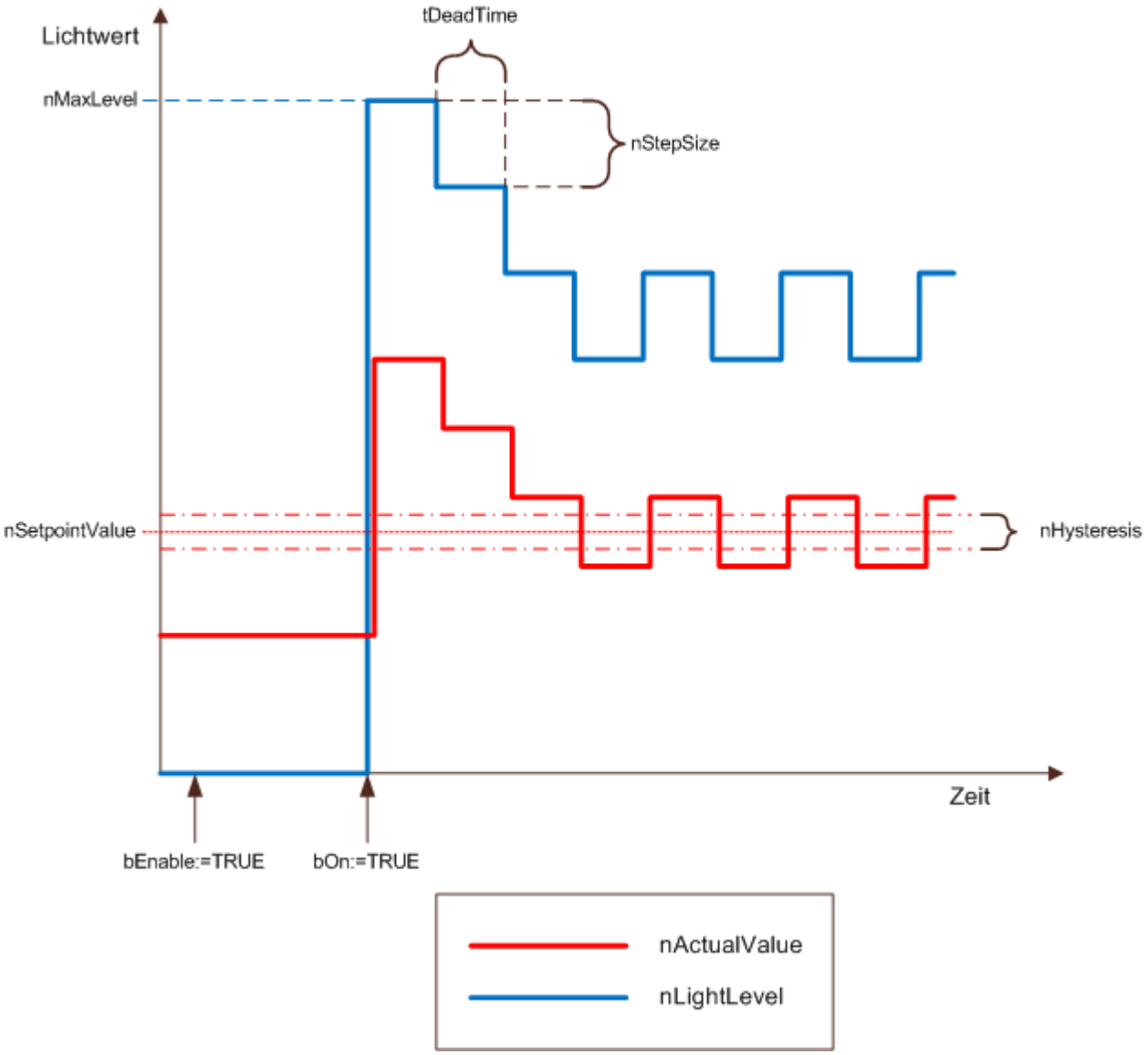
Ablaufdiagramm

Folgendes Diagramm soll verdeutlichen, wie die Regelung im normalen Ablauf verhält:

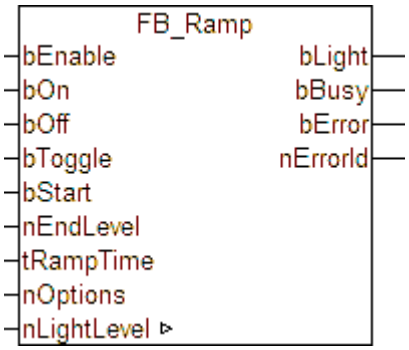


Die Regelung wird zunächst durch ein TRUE-Signal am Eingang *bEnable* frei geschaltet. Bei der darauffolgenden positiven Flanke an *bOn* wird *nLightLevel* erst auf den Maximalwert gesteuert. Dies beeinflußt auch den gemessenen Lichtwert *nActualValue*, welcher ansteigt, so dass ein Abdimmen erforderlich wird. *nLightLevel* wird nun schrittweise reduziert, bis sich der gemessene Lichtwert *nActualValue* im Hysteresebereich um den Sollwert befindet (*nSetpointValue* - 0.5**nHysteresis* < x < *nSetpointValue* + 0.5**nHysteresis*).
fällt dann beispielsweise der gemessene Lichtwert, etwa durch Bewölkung außen, so steuert die Regelung durch schrittweises Aufdimmen dagegen, bis der Lichtwert wieder im Hystereseband liegt.

Wird die Schrittweite *nStepSize* zu groß bzw. die Hysterese zu klein gewählt kann es zu einer Oszillation um den Sollwert kommen. Dadurch dass die Hysterese klein gegenüber den Sprüngen ist, wird der Hysteresebereich ständig verfehlt:



3.1.9 FB_Ramp



Funktionsblock zur Realisierung einer Lichtrampe.

Durch eine steigende Flanke am Eingang *bOn* wird das Licht auf den Maximalwert (*32767*) geschaltet - eine steigende Flanke am Eingang *bOff* schaltet das Licht wieder aus.Steigende Flanken am Eingang *bToggle* invertieren den jeweiligen Lichtzustand. Eine positive Flanke am Eingang *bStart* läßt den Baustein das Licht vom aktuellen Wert auf *nEndLevel* dimmen. Die dazu benötigte Zeit wird durch *tRampTime* festgelegt. Alle Eingänge sind nur aktiv, solange *bEnable* auf TRUE steht, ansonsten wird der Baustein intern zurückgesetzt.

VAR_INPUT

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

bEnable: Solange dieser Eingang auf TRUE steht, sind die Eingänge *bOn*, *bOff*, *bToggle* und *bStart* aktiv. Ein negativer Zustand deaktiviert die Eingänge und setzt den Baustein zurück.

bOn: Eine steigende Flanke schaltet *nLightLevel* direkt auf den den Maximalwert (*32767*).

bOff: Eine steigende Flanke schaltet *nLightLevel* unmittelbar auf "*0*".

bToggle: Schaltet den Lichtzustand zwischen An (*32767*) und Aus (*0*) jeweils hin und her.

bStart: Liegt an diesem Eingang eine steigende Flanke an, so wird das Licht vom derzeitigen Wert auf *nEndLevel* herauf bzw. herabgedimmt. Die dafür benötigte Zeit wird mit *tRampTime* festgelegt. Der Dimmvorgang kann jederzeit durch *bOn*, *bOff* oder *bToggle* unterbrochen werden.

nEndLevel: Zielwert des Dimmvorgangs.

tRampTime: Rampenzeit, siehe *bStart*. (Voreingestellter Wert: 10 Sekunden).

nOptions: Reserviert für zukünftige Entwicklungen.

VAR_OUTPUT

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

bLight: Dieser Ausgang ist gesetzt, solange *nLightLevel* größer als 0 ist.

bBusy: Dieser Ausgang ist immer dann aktiv, solange eine Befehlsabarbeitung (*bOn*, *bOff*, *bToggle* oder Rampenfahrt) aktiv ist.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *nErrorId* enthalten. Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.

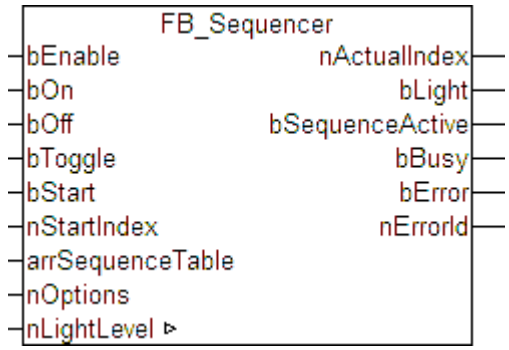
nErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 90].

VAR_IN_OUT

```
nLightLevel           : UINT;
```

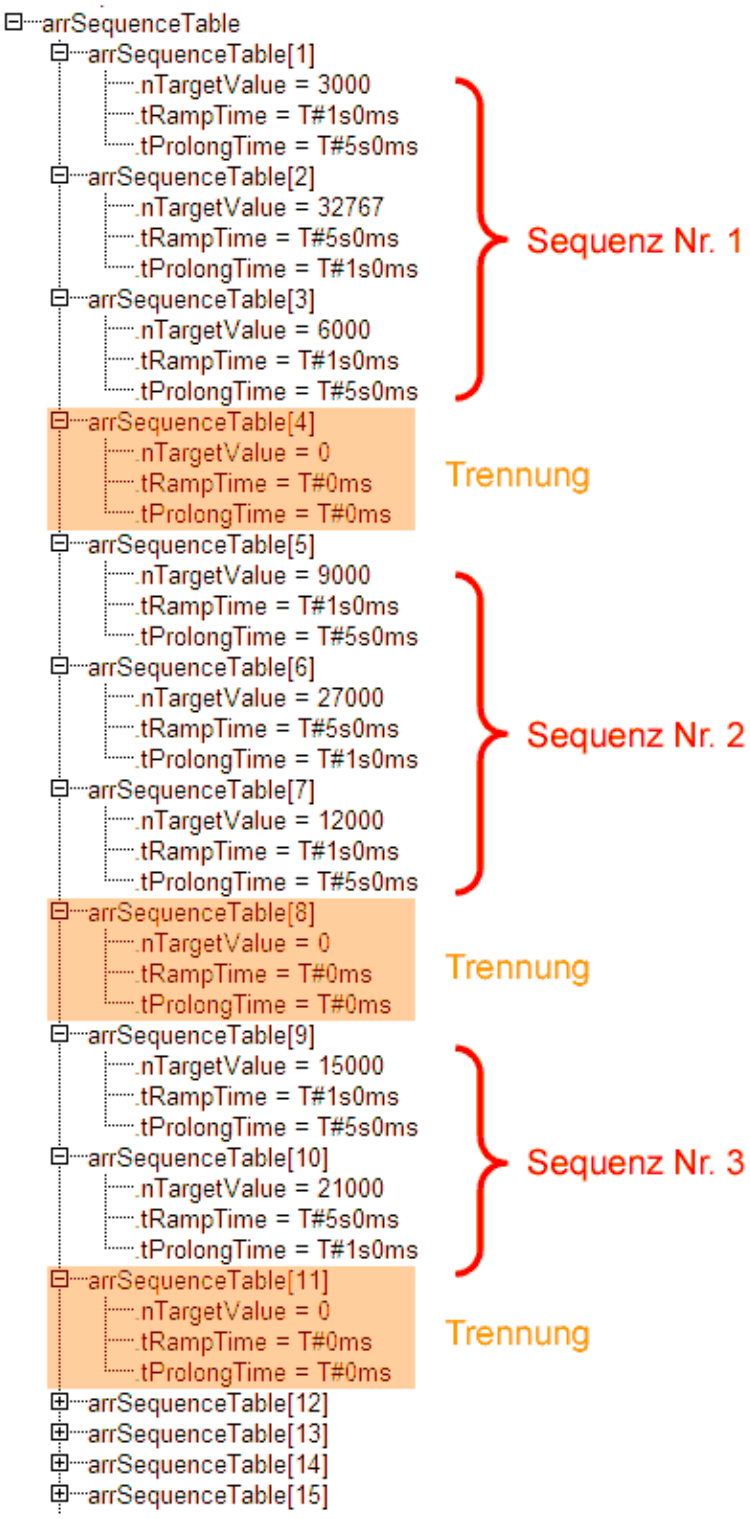
nLightLevel: Ausgabe-Stellgröße des Bausteines und Verweis auf den Lichtausgangswert. Dieser Wert ist deshalb als IN-Out-Variable definiert, weil der Funktionsbaustein den Lichtwert zum einen ausliest und zum anderen beschreibt.

3.1.10 FB_Sequencer

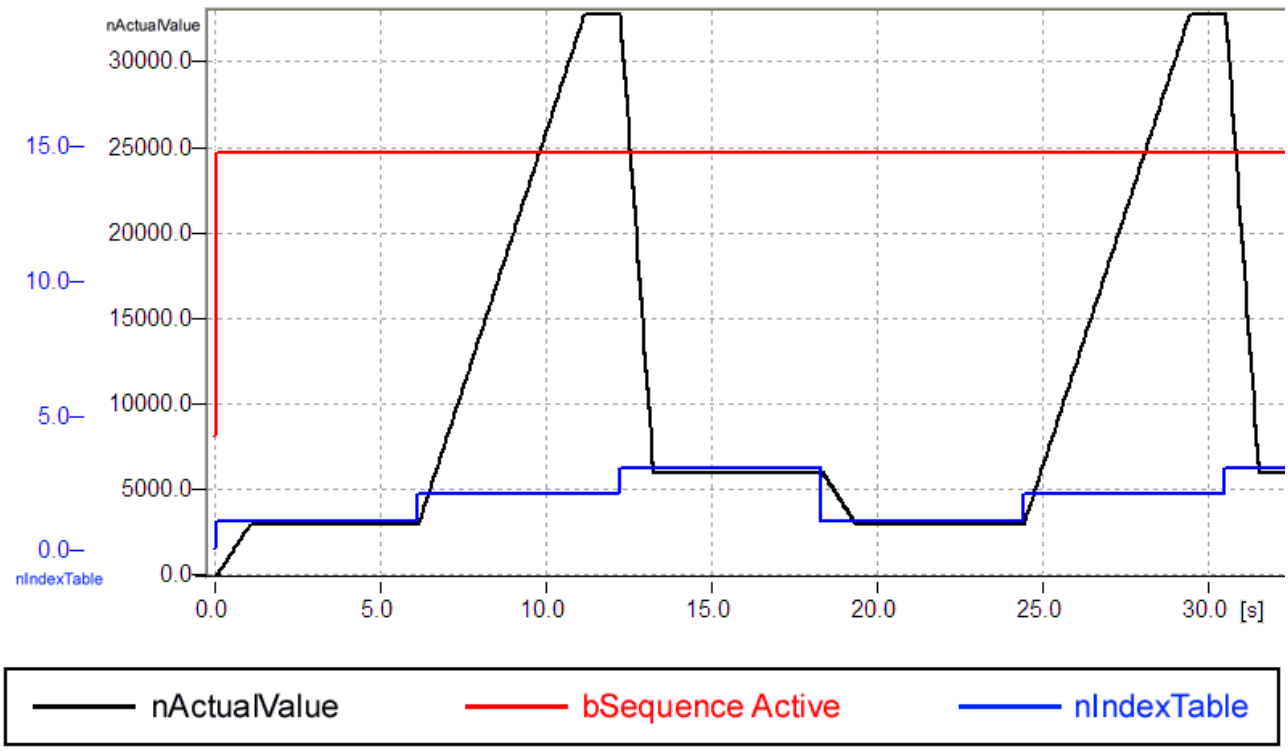


Funktionsblock Realisierung von Lichtsequenzen mit bis zu 50 Stützpunkten.

Kernstück dieses Funktionsblockes ist ein Rampenbaustein, der einzelne, in einer Tabelle definierte Helligkeitswerte in einer einstellbaren Zeit anfährt und auf diesem Helligkeitswert dann eine ebenfalls definierbare Zeit verweilt. Nach dem Verweilen wird dann der nächste Wert angefahren. Die Tabelle *arrSequenceTable* besteht, wie bereits erwähnt, aus 50 Einträgen mit den Werten für *nTargetValue* (Zielwert), *tRampTime* (Zeit zum Erreichen des Zielwertes) und *tProlongTime* (Verweilzeit auf dem Zielwert). Es ist nicht zwingend notwendig, alle 50 Werte zu nutzen. Ein 0-Eintrag aller 3 Werte markiert das Ende einer Sequenz. Darüber hinaus ist es mit dem Eingang *nStartIndex* möglich, eine Lichtsequenz an jeder beliebigen Stelle der Tabelle beginnen zu lassen. Dadurch lassen sich auch innerhalb der 50 Einträge mehrere verschiedene Lichtsequenzen programmieren, die untereinander jeweils durch 0-Eintrags-Elemente getrennt sind:



Sequenz 1 beispielsweise sieht im zeitlichen Verlauf folgendermaßen aus (*nStartIndex=1, nOptions.bit0=TRUE*, Erläuterung siehe unten):



Darüber hinaus läßt sich der Baustein "normal" ein- und ausschalten (Ein: *nLightLevel* = 32767, Aus: *nLightLevel* = 0) sowie über den Eingang *bToggle* zwischen "Ein" und "Aus" hin und herschalten. Alle Befehlseingänge sind jedoch nur dann aktiv, wenn der Eingang *bEnable* auf *TRUE* steht. Wird er zurück auf *FALSE* gesetzt, so werden keine Befehle mehr angenommen und der Lichtwert behält seinen aktuellen Zustand - auch aus einer Rampenfahrt heraus.

VAR_INPUT

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

bEnable: Solange dieser Eingang auf *TRUE* steht, sind die Eingänge *bOn*, *bOff*, *bToggle* und *bStart* aktiv. Ein negativer Zustand deaktiviert die Eingänge und setzt den Baustein zurück.

bOn: Eine steigende Flanke schaltet *nLightLevel* direkt auf den den Maximalwert (*32767*).

bOff: Eine steigende Flanke schaltet *nLightLevel* unmittelbar auf "*0*".

bToggle: Schaltet den Lichtzustand zwischen An (*32767*) und Aus (*0*) jeweils hin und her.

bStart: Eine positive Flanke startet eine Lichtsequenz ab dem unter *nStartIndex* definierten Anfang.

nStartIndex: Siehe *bStart.*

arrSequenceTable: Lichtwert-Tabelle mit den dazugehörigen Rampen- und Verweilzeiten.

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

nTargetValue: Zielwert.

tRampTime: Zeit zum Erreichen des Zielwertes.

tProlongTime: Verweilzeit auf dem Zielwert.

nOptions: Parametriereingang. Die Setzen (bzw. Nicht-Setzen) der einzelnen Bits dieser Variablen vom Typ *DWORD* hat folgende Wirkung:

Konstante	Beschreibung
OPTION_INFINITE_LOOP	Nach Ablauf einer Sequenz läuft der Baustein automatisch an der an *nStartIndex* definierten Stelle weiter. Ist diese Option nicht gesetzt, so stoppt die Sequenz nach dem Ablauf. Eine erneute positive Flanke an *bStart* wäre für einen Sequenz-Neustart nötig.

VAR_OUTPUT

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

nActualIndex: Verweis auf dem aktuellen Element in der Sequenz-Tabelle. Ist eine Sequenz beendet (*bSequenceActive* = *FALSE* , s.u.) wird dieser Ausgang zu "0".

bLight: Dieser Ausgang ist so lange gesetzt, wie *nLightLevel* größer als "*0*" ist.

bSequenceActive: Bei Abarbeitung einer Sequenz wird dieser Ausgang auf *TRUE* gesetzt.

bBusy: Dieser Ausgang ist immer dann aktiv, solange eine Befehlsabarbeitung (*bOn*, *bOff*, *bToggle* oder Rampenfahrt) aktiv ist.

bError: Dieser Ausgang wird auf *TRUE* geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *nErrorId* enthalten. Wird durch das Ausführen eines Befehls an den Eingängen auf *FALSE* zurückgesetzt.

nErrorId: Enthält den spezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf "*0*" zurückgesetzt. Siehe Fehlercodes [▶ 90].

VAR_IN_OUT

```
nLightLevel         : UINT;
```

nLightLevel: Ausgabe-Stellgröße des Bausteines und Verweis auf den Lichtausgangswert.

3.1.11 FB_StairwellDimmer

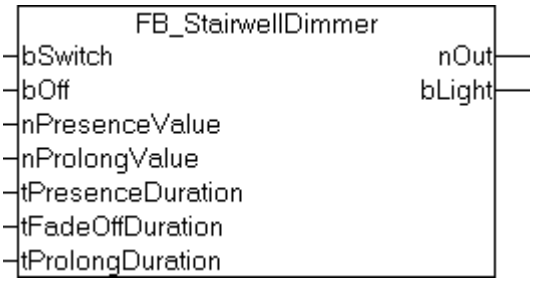


Abb. 1: FB_StairwellDimmer

Durch eine steigende Flanke am Eingang *bSwitch* wird der analoge Ausgang *nOut* auf den Wert *nPresenceValue* gesetzt. Durch eine fallende Flanke an *bSwitch* wird ein Timer mit der Laufzeit von *tPresenceDuration* gestartet bzw. erneut gestartet. Nach Ablauf dieses Timers wird *nOut* über einen Zeitraum von *tFadeOffDuration* auf den Wert *nProlongValue* gedimmt. Dieser Wert wird über den Zeitraum *tProlongDuration* beibehalten. Danach wird *nOut* auf 0 gesetzt. Eine positive Flanke am Eingang *bOff* schaltet den Ausgang *nOut* unmittelbar auf 0. Der digitale Ausgabewert *bLight* ist immer dann gesetzt, wenn *nOut* größer als 0 ist.

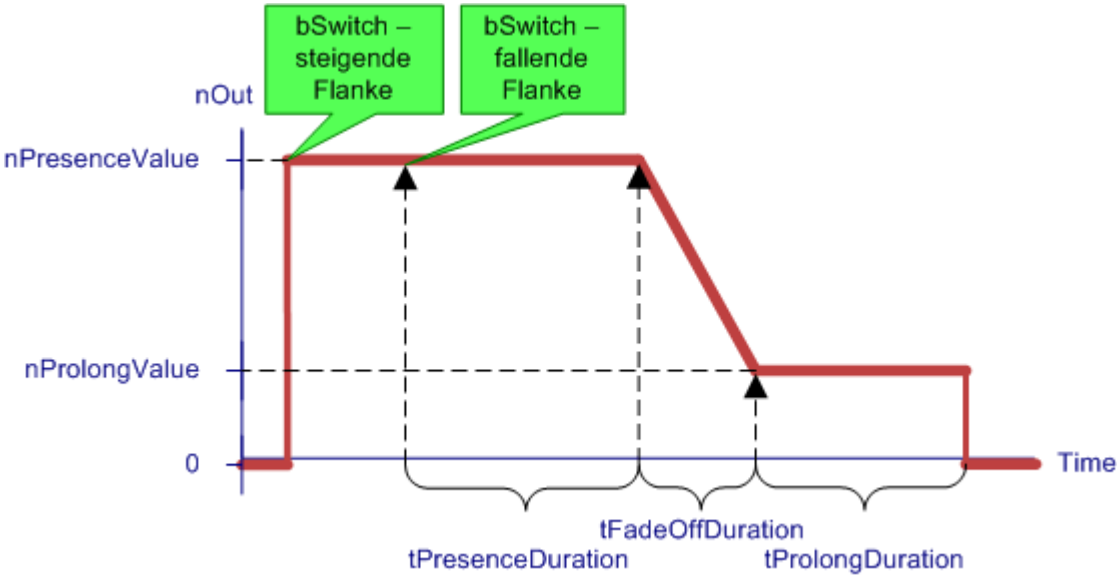


Abb. 2: FB_StairwellDimmer-Time-nOut

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: Bei steigender Flanke: *nOut* wird auf *nPresenceValue* gesetzt. Bei fallender Flanke: Starten der Präsenzzeit (siehe Grafik).

bOff: Schaltet *nOut* unmittelbar aus.

nPresenceValue: Wert, auf den *nOut* während der Präsenzzeit gesetzt werden soll. (Voreingestellter Wert: 32767).

nProlongValue: Wert, auf den *nOut* während der Verweilzeit gestzt werden soll. (Voreingestellter Wert: 10000).

tPresenceDuration: Dauer der Präsenzzeit, bei der *nOut* nach einer fallenden Flanke an *bSwitch* auf *nPresenceValue* gesetzt wird. (Voreingestellter Wert: 120 Sekunden).

tFadeOffDuration: Zeitdauer, in der *nOut* nach der Präsenzzeit auf die Verweilzeit heruntergedimmt wird. (Voreingestellter Wert: 10 Sekunden).

tProlongDuration: Dauer der Verweilzeit. (Voreingestellter Wert: 20 Sekunden).

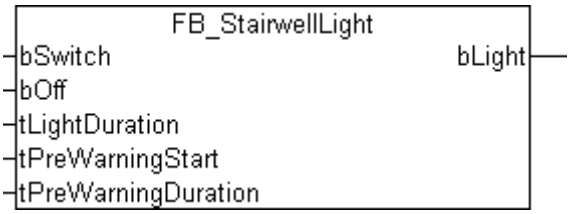
VAR_OUTPUT

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

nOut: Ausgang, der den momentanen Lichtwert ausgibt.

bLight: Dieser Ausgang ist solange gesetzt, wie *nOut* größer als 0 ist.

3.1.12 FB_StairwellLight



Durch eine positive Flanke am Eingang *bSwitch* wird der Ausgang *bLight* gesetzt. Nach Ablauflauf der Zeit *tLightDuration* wird der Ausgang wieder zurückgesetzt. Wird vor Ablauf der Zeit erneut ein Signal am Eingang *bSwitch* angelegt, so wird die Zeitdauer retriggert. Nach der Zeit *tPreWarningStart* wird für die Zeitdauer *tPreWarningDuration* das Licht ausgeschaltet (Vorwarnung). Soll keine Vorwarnung stattfinden, so muß der Parameter *tPreWarningStart* auf 0 gesetzt werden. Eine positive Flanke am Eingang *bOff* schaltet den Ausgang unmittelbar aus.

VAR_INPUT

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

bSwitch: Schaltet den Ausgang für die Zeit *tLightDuration* ein.

bOff: Schaltet den Ausgang aus.

tLightDuration: Zeitdauer, für die der Ausgang gesetzt wird.

tPreWarningStart: Vorwarnzeit.

tPreWarningDuration: Dauer der Vorwarnung.

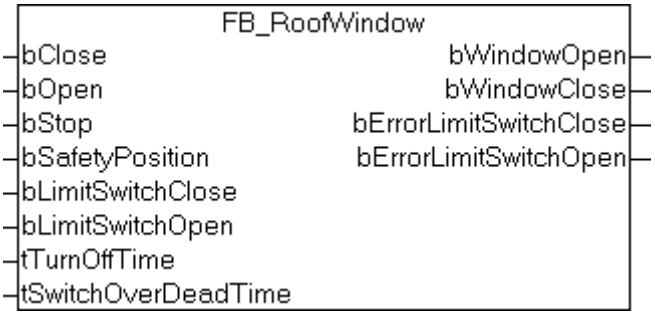
VAR_OUTPUT

```
bLight          : BOOL;
```

bLight: Bei einer positiven Flanke an *bSwitch* wird der Ausgang für die Dauer von *tLightDuration* gesetzt.

3.2 Fassade

3.2.1 FB_RoofWindow



Beschreibung

Durch eine positive Flanke an den Eingängen *bOpen* oder *bClose* werden die Ausgänge *bWindowOpen* oder *bWindowClose* gesetzt. Diese bleiben so lange anstehen, bis die Zeit *tTurnOffTime* abgelaufen ist, oder bis ein anderer Befehl an den Baustein gegeben wird. Durch eine positive Flanke am Eingang *bStop* werden beide Ausgänge unmittelbar zurückgesetzt.

Durch den Parameter *tSwitchOverDeadTime* kann verhindert werden, dass ein unmittelbarer Richtungswechsel den Antriebsmotor zerstört. In den meisten Fällen liegt dieser Wert zwischen 0,5sec und 1,0 sec. Den genauen Wert erfahren sie beim Antriebshersteller.

Sicherheitsposition

Das Fahren in die Sicherheitsposition (z.B. bei starkem Wind) kann durch Setzen des Eingangs *bSafetyPosition* erreicht werden. Für die Zeit *tTurnOffTime* wird der Ausgang *bWindowClose* gesetzt und der Ausgang *bWindowOpen* zurückgesetzt. Die Bedienung des Fensters ist solange gesperrt, wie der Eingang *bSafetyPosition* aktiv ist.

VAR_INPUT

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

bClose: Ausgang *bWindowClose* setzen und Ausgang *bWindowOpen* zurücksetzen. Der Ausgang *bWindowClose* bleibt in Selbsthaltung.

bOpen: Ausgang *bWindowOpen* setzen und Ausgang *bWindowClose* zurücksetzen. Der Ausgang *bWindowOpen* bleibt in Selbsthaltung.

bStop: Ausgang *bWindowClose* und *bWindowOpen* zurücksetzen

bSafetyPosition: Sicherheitsposition wird angefahren. Hierfür wird für die Zeit *tTurnOffTime* das Fenster geschlossen. Solange der Eingang ansteht, ist die Bedienung des Fensters gesperrt.

bLimitSwitchClose: Optionaler Endschalter. Ist *bClose* gesetzt und wird innerhalb von *tTurnOffTimebLimitSwitchClose* nicht gesetzt, so wird *bErrorLimitSwitchClose* gesetzt.

bLimitSwitchOpen: Optionaler Endschalter. Ist *bOpen* gesetzt und wird innerhalb von *tTurnOffTimebLimitSwitchOpen* nicht gesetzt, so wird *bErrorLimitSwitchOpen* gesetzt.

tTurnOffTime: Wird kein Eingang betätigt, so werden nach dieser Zeitdauer die Ausgänge zurückgesetzt. Wird eine Zeitdauer von 0 angegeben, so werden die Ausgänge nicht automatisch zurückgesetzt. Der hier angegebene Wert sollte ca. 10% größer sein, als die tatsächlich gemessene Fahrdauer.

tSwitchOverDeadTime: Verweildauer beim Richtungswechsel. Während dieser Zeit sind beide Ausgänge zurückgesetzt.

VAR_OUTPUT

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

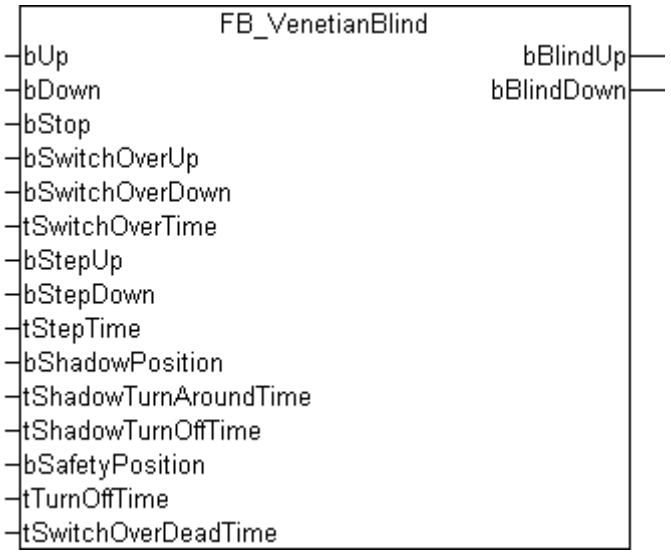
bWindowOpen: Das Fenster wird geöffnet.

bWindowClose: DasFenster wird geschlossen.

bErrorLimitSwitchClose: Fehler vom optionalen Endlagenschalter beim Schließen.

bErrorLimitSwitchOpen: Fehler vom optionalen Endlagenschalter beim Öffnen.

3.2.2 FB_VenetianBlind



Beschreibung

Es stehen drei verschiedene Arten zur Verfügung, wie die Jalousie angesteuert werden kann:

- Durch eine positive Flanke an den Eingängen *bUp* oder *bDown* werden die Ausgänge *bBlindUp* oder *bBlindDown* gesetzt. Diese bleiben so lange anstehen, bis die Zeit *tTurnOffTime* abgelaufen ist, oder bis ein anderer Befehl an den Baustein gegeben wird. Durch eine positive Flanke am Eingang *bStop* werden beide Ausgänge unmittelbar zurückgesetzt.
- An den Eingängen *bSwitchOverUp* oder *bSwitchOverDown* werden statische Signale angelegt (z.B. durch Taster). Dadurch werden die Ausgänge *bBlindUp* oder *bBlindDown* gesetzt. Steht dieses Signal länger als *tSwitchOverTime* an, so gehen die Ausgänge in Selbsthaltung. D.h. die Ausgänge bleiben weiterhin anstehen, auch dann, wenn die Signale an den Eingängen wieder weggenommen werden. In den meisten Fällen ist für den Parameter *tSwitchOverTime* ein Wert von 500ms ausreichend. Der Ausgang bleibt aber maximal für die Zeitdauer *tTurnOffTime* anstehen oder bis ein neuer Befehl an den Baustein gegeben wird.
- Die letzte Variante kann dann sinnvoll sein, wenn der Bediener die Stellung der Jalousie schrittweise ändern will. Bei jeder positiven Flanke am Eingang *bStepUp* oder *bStepDown* wird der entsprechende Ausgang für die Zeit *tStepTime* gesetzt. Für *tStepTime* hat sich ein Wert von 200ms bewährt.

Durch den Parameter *tSwitchOverDeadTime* kann verhindert werden, dass ein unmittelbarer Richtungswechsel den Antriebsmotor zerstört. In den meisten Fällen liegt dieser Wert zwischen 0,5sec und 1,0 sec. Den genauen Wert erfahren sie beim Antriebshersteller.

Sicherheitsposition

Das Fahren in die Sicherheitsposition (z.B. bei starkem Wind oder bei Wartungsarbeiten am Fenster) kann durch Setzen des Eingangs *bSafetyPosition* erreicht werden. Für die Zeit *tTurnOffTime* wird der Ausgang *bBlindUp* gesetzt und der Ausgang *bBlindDown* zurückgesetzt. Die Bedienung der Jalousie ist so lange gesperrt, wie der Eingang *bSafetyPosition* aktiv ist.

Beschattungsposition

Bei überdurchschnittlicher Sonneneinstrahlung kann die Jalousie in die Beschattungsposition gefahren werden. Hierzu wird nach dem Anlegen einer positiven Flanke an den Eingang *bShadowPosition* die Jalousie für die Zeitdauer *tShadowTurnOffTime* nach Unten gefahren. Anschließend wird für die Zeit *tShadowTurnAroundTime* die Jalousie wieder nach Oben gefahren. Üblicherweise wird hierfür eine Zeit von ca. 2sec eingestellt. Hierdurch wird eine komplette Verdunkelung des Raumes verhindert. Beim Richtungswechsel wird eine Pause von der Dauer *tSwitchOverDeadTime* eingehalten. Das Anfahren der Beschattungsposition kann jederzeit durch einen neuen Befehl unterbrochen werden.

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: Ausgang *bBlindUp* setzen und Ausgang *bBlindDown* zurücksetzen. Der Ausgang *bBlindUp* bleibt in Selbsthaltung.

bDown: Ausgang *bBlindDown* setzen und Ausgang *bBlindUp* zurücksetzen. Der Ausgang *bBlindDown* bleibt in Selbsthaltung.

bStop: Ausgang *bBlindUp* und *bBlindDown* zurücksetzen.

bSwitchOverUp: Ausgang *bBlindUp* setzen und Ausgang *bBlindDown* zurücksetzen. Bleibt das Signal länger als *tSwitchOverTime* anstehen, so bleibt der Ausgang *bBlindUp* in Selbsthaltung.

bSwitchOverDown: Ausgang *bBlindDown* setzen und Ausgang *bBlindUp* zurücksetzen. Bleibt das Signal länger als *tSwitchOverTime* anstehen, so bleibt der Ausgang *bBlindDown* in Selbsthaltung.

tSwitchOverTime: Gibt die Zeit an, die die Eingänge *bSwitchUp* und *bSwitchDown* anstehen müssen, bis die Ausgänge in Selbsthaltung gehen. Wird der Wert 0 angegeben, so gehen die Ausgänge sofort in Selbsthaltung.

bStepUp: Ausgang *bBlindDown* zurücksetzen und Ausgang *bBlindUp* für die Zeit *tStepTime setzen*.

bStepDown: Ausgang *bBlindUp* zurücksetzen und Ausgang *bBlindDown* für die Zeit *tStepTime setzen*.

tStepTime: Wird die Jalousie mit den Eingängen *bStepUp* oder *bStepDown* gesteuert, so bleiben die Ausgänge für diese Zeitdauer anstehen. Wird eine Zeitdauer von 0 angegeben, so werden die Ausgänge nicht gesetzt.

bShadowPosition: Die Beschattungsposition wird angefahren (siehe unten).

tShadowTurnAroundTime: Nach dem Erreichen der Beschattungsposition wird für die Zeitdauer *tShadowTurnAroundTime* ein Richtungswechsel durchgeführt.

tShadowTurnOffTime: Zeitdauer, für die der Ausgang *bBlindDown* gesetzt wird, um die Beschattungsposition anzufahren. Voraussetzung für das Anfahren in die Beschattungsposition ist eine Zeitdauer größer 0.

bSafetyPosition: Sicherheitsposition wird angefahren. Hierfür wird für die Zeit *tTurnOffTime* die Jalousie hochgefahren. Solange der Eingang ansteht, ist die Bedienung der Jalousie gesperrt.

tTurnOffTime: Wird kein Eingang betätigt, so werden nach dieser Zeitdauer die Ausgänge zurückgesetzt. Wird eine Zeitdauer von 0 angegeben, so werden die Ausgänge nicht automatisch zurückgesetzt. Der hier angegebene Wert sollte ca. 10% größer sein, als die tatsächlich gemessene Fahrdauer.

tSwitchOverDeadTime: Verweildauer beim Richtungswechsel. Während dieser Zeit sind beide Ausgänge zurückgesetzt.

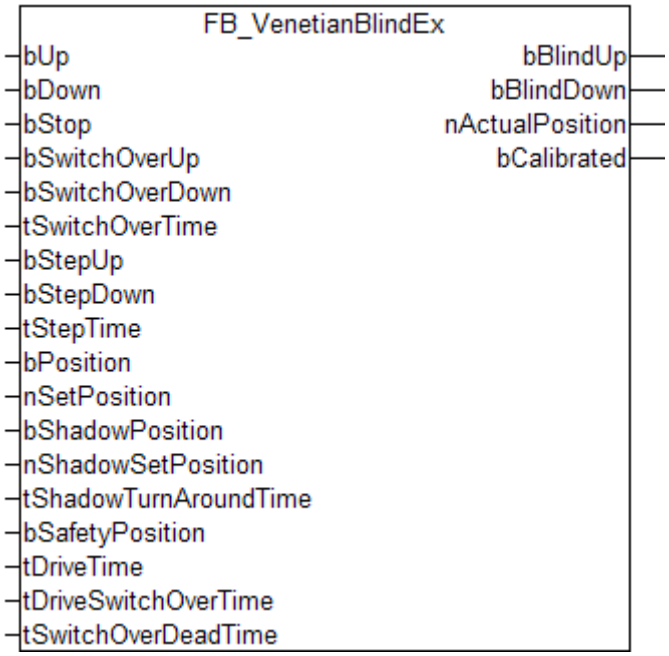
VAR_OUTPUT

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

bBlindUp: Die Jalousie fährt hoch.

bBlindDown: Die Jalousie fährt runter.

3.2.3 FB_VenetianBlindEx



Beschreibung

Es stehen vier verschiedene Arten zur Verfügung, wie die Jalousie angesteuert werden kann:

- Durch eine positive Flanke an den Eingängen *bUp* oder *bDown* werden die Ausgänge *bBlindUp* oder *bBlindDown* gesetzt. Diese bleiben so lange anstehen, bis die Zeit *tDriveTime* + 10% abgelaufen ist oder bis ein anderer Befehl an den Baustein gegeben wird. Durch eine positive Flanke am Eingang *bStop* werden beide Ausgänge unmittelbar zurückgesetzt.
- An den Eingängen *bSwitchOverUp* oder *bSwitchOverDown* werden statische Signale angelegt (z.B. durch Taster). Dadurch werden die Ausgänge *bBlindUp* oder *bBlindDown* gesetzt. Steht dieses Signal länger als *tSwitchOverTime* an, so gehen die Ausgänge in Selbsthaltung. D.h. die Ausgänge bleiben weiterhin anstehen, auch dann, wenn die Signale an den Eingängen wieder weggenommen werden. In den meisten Fällen ist für den Parameter *tSwitchOverTime* ein Wert von 500ms ausreichend. Der Ausgang bleibt aber maximal für die Zeitdauer *tDriveTime* + 10% anstehen oder bis ein neuer Befehl an den Baustein gegeben wird.
- In bestimmten Anwendungen kann es sinnvoll sein, wenn der Bediener die Stellung der Jalousie schrittweise ändern kann. Bei jeder positiven Flanke am Eingang *bStepUp* oder *bStepDown* wird der entsprechende Ausgang für die Zeit *tStepTime* gesetzt. Für *tStepTime* hat sich ein Wert von 200ms bewährt.
- Gegenüber dem Baustein FB_VenetianBlind() [▶ 36] kann mit diesem Baustein auch eine absolute Position angefahren werden. Hierzu wird ein Prozentwert an den Eingang *nSetPosition* angelegt und anschließend eine positive Flanke auf den Eingang *bPosition* gegeben.

Durch den Parameter *tSwitchOverDeadTime* kann verhindert werden, dass ein unmittelbarer Richtungswechsel den Antriebsmotor zerstört. In den meisten Fällen liegt dieser Wert zwischen 0,5sec und 1,0sec. Den genauen Wert erfahren sie beim Antriebshersteller.

Sicherheitsposition

Das Fahren in die Sicherheitsposition (z.B. bei starkem Wind oder bei Wartungsarbeiten am Fenster) kann durch Setzen des Eingangs *bSafetyPosition* erreicht werden. Für die Zeit *tDriveTime* + 10% wird der Ausgang *bBlindUp* gesetzt und der Ausgang *bBlindDown* zurückgesetzt. Die Bedienung der Jalousie ist so lange gesperrt, wie der Eingang *bSafetyPosition* aktiv ist.

Beschattungsposition

Bei überdurchschnittlicher Sonneneinstrahlung kann die Jalousie in die Beschattungsposition gefahren werden. Hierzu wird nach dem Anlegen einer positiven Flanke an den Eingang *bShadowPosition* die Jalousie auf die Position *nShadowSetPosition* gefahren. Anschließend wird für die Zeit *tShadowTurnAroundTime* die Jalousie wieder nach Oben gefahren. Hierdurch wird eine komplette Verdunkelung des Raumes verhindert. Wurde beim Anfahren der Beschattungsposition schon nach Oben gefahren, so wird die Jalousie für die Zeit *tDriveSwitchOverTime - tShadowTurnAroundTime* nach Unten gefahren. Es wird also der gleiche Winkel eingestellt, als wenn die Jalousie zum Verdunkeln nach Unten gefahren wäre.
Beim Richtungswechsel wird eine Pause von der Dauer *tSwitchOverDeadTime* eingehalten. Das Anfahren der Beschattungsposition kann jederzeit durch einen neuen Befehl unterbrochen werden.
Hinweis: Die eingetragene Beschattungszeit *tShadowTurnAroundTime* darf niemals größer als die Zeitdauer für den Richtungswechsel *tDriveSwitchOverTime* .

Anfahren einer absoluten Position

Anmerkungen

Da in den meisten Fällen eine Jalousie keine Rückmeldung über die aktuelle Position zurück gibt, kann diese nur anhand der Fahrdauer berechnet werden. Die Genauigkeit ist davon abhängig, wie konstant die Geschwindigkeit der Jalousie ist. Außerdem sollten die Geschwindigkeitsunterschiede zwischen Auf-und Zufahren möglichst gering sein.
Die Positionen werden immer in Prozent angegeben. Dabei entspricht 0% ganz Oben und 100% ganz Unten. Wird ein Wert größer 100 angegeben, so wird dieser im Baustein auf 100 begrenzt.

Ermitteln der Parameter

Als erstes müssen bestimmte Parameter der Jalousie ermittelt werden. Zum einen ist dieses die Fahrzeit. Also die Zeit, die benötigt wird, um die gesamte Strecke der Jalousie zu fahren. Zum Zweiten muss die Zeit ermittelt werden, die für ein Richtungswechsel benötigt wird. Während eines Richtungswechsels ändert sich der Winkel der einzelnen Lamellen. Die Fahrdauer wird an den Parameter *tDriveTime* übergeben. Die Dauer für einen Richtungswechsel an *tDriveSwitchOverTime*.

Baustein referenzieren

Da die aktuelle Position der Jalousie nur berechnet werden kann, summieren sich während des Betriebes die Ungenauigkeiten. Um die Abweichungen zu begrenzen, referenziert sich der Baustein möglichst häufig selbstständig. Dieses geschieht, wenn die Jalousie ganz nach Oben oder ganz nach Unten gefahren wird und der entsprechende Ausgang selbstständig zurückgesetzt wird. Also nach Ablauf der Zeit *tDriveTime* + 10%.

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: Ausgang *bBlindUp* setzen und Ausgang *bBlindDown* zurücksetzen. Der Ausgang *bBlindUp* bleibt in Selbsthaltung.

bDown: Ausgang *bBlindDown* setzen und Ausgang *bBlindUp* zurücksetzen. Der Ausgang *bBlindDown* bleibt in Selbsthaltung.

bStop: Ausgang *bBlindUp* und *bBlindDown* zurücksetzen.

bSwitchOverUp: Ausgang *bBlindUp* setzen und Ausgang *bBlindDown* zurücksetzen. Bleibt das Signal länger als *tSwitchOverTime* anstehen, so bleibt der Ausgang *bBlindUp* in Selbsthaltung.

bSwitchOverDown: Ausgang *bBlindDown* setzen und Ausgang *bBlindUp* zurücksetzen. Bleibt das Signal länger als *tSwitchOverTime* anstehen, so bleibt der Ausgang *bBlindDown* in Selbsthaltung.

tSwitchOverTime: Gibt die Zeit an, die die Eingänge *bSwitchUp* und *bSwitchDown* anstehen müssen, bis die Ausgänge in Selbsthaltung gehen. Wird der Wert 0 angegeben, so gehen die Ausgänge sofort in Selbsthaltung.

bStepUp: Ausgang *bBlindDown* zurücksetzen und Ausgang *bBlindUp* für die Zeit *tStepTime setzen*.

bStepDown: Ausgang *bBlindUp* zurücksetzen und Ausgang *bBlindDown* für die Zeit *tStepTime setzen*.

tStepTime: Wird die Jalousie mit den Eingängen *bStepUp* oder *bStepDown* gesteuert, so bleiben die Ausgänge für diese Zeitdauer anstehen. Wird eine Zeitdauer von 0 angegeben, so werden die Ausgänge nicht gesetzt.

bPosition: Jalousie auf eine vorgegebene Position fahren.

nSetPosition: Position (0%-100%) auf die die Jalousie gefahren werden soll, nachdem an dem Eingang *bPosition* eine positive Flanke angelegt wurde. 0% entspricht ganz Oben, 100% entspricht ganz Unten.

bShadowPosition: Die Beschattungsposition wird angefahren (siehe unten).

nShadowSetPosition: Beschattungsposition (0%-100%) auf die die Jalousie gefahren werden soll, nachdem an dem Eingang *bShadowPosition* eine positive Flanke angelegt wurde.

tShadowTurnAroundTime: Nach dem Erreichen der Beschattungsposition wird für die Zeitdauer *tShadowTurnAroundTime* die Jalousie nach Oben gefahren.

bSafetyPosition: Sicherheitsposition wird angefahren. Hierfür wird für die Zeit *tDriveTime* + 10% die Jalousie hochgefahren. Solange der Eingang ansteht, ist die Bedienung der Jalousie gesperrt.

tDriveTime: Fahrdauer der Jalousie von ganz Oben nach ganz Unten. Wird kein Eingang betätigt, so werden nach der Zeitdauer *tDriveTime* + 10% die Ausgänge zurückgesetzt. Wird eine Zeitdauer von 0 angegeben, so werden die Ausgänge nicht automatisch zurückgesetzt. Das Anfahren der Jalousie auf absolute Positionen ist dann nicht möglich.

tDriveSwitchOverTime: Zeitdauer, die die Jalousie für einen Richtungswechsel benötigt.

tSwitchOverDeadTime: Verweildauer beim Richtungswechsel. Während dieser Zeit sind beide Ausgänge zurückgesetzt.

VAR_OUTPUT

```
bBlindUp           : BOOL;
bBlindDown         : BOOL;
nActualPosition    : USINT;
bCalibrated        : BOOL;
```

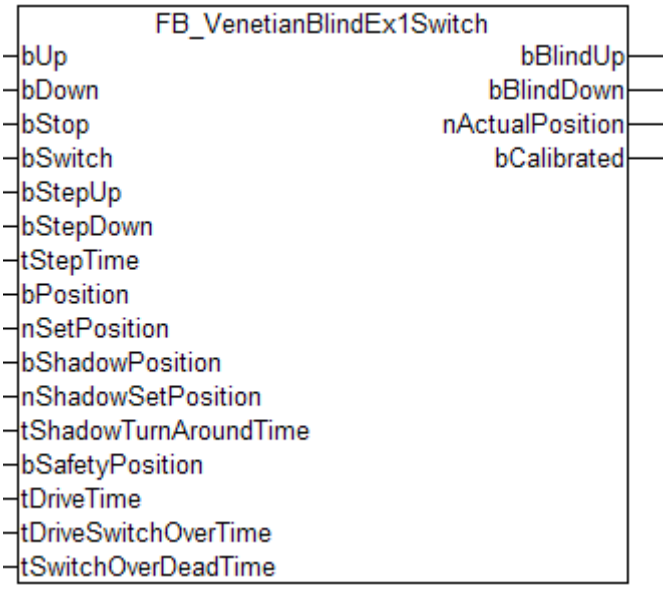
bBlindUp: Die Jalousie fährt hoch.

bBlindDown: Die Jalousie fährt runter.

nActualPosition: Aktuelle Position in Prozent.

bCalibrated: Gibt an, ob die Jalousie kalibriert ist.

3.2.4 FB_VenetianBlindEx1Switch



Beschreibung

Jalousiebaustein mit der gleiche Funktionalität wie der FB_VenetianBlindEx() [▶ 38] jedoch mit einen Eingang *bSwitch* zum Hoch- und Heruntertasten.
Es stehen vier verschiedene Arten zur Verfügung, wie die Jalousie angesteuert werden kann:

- Durch eine positive Flanke an den Eingängen *bUp* oder *bDown* werden die Ausgänge *bBlindUp* oder *bBlindDown* gesetzt. Diese bleiben so lange anstehen, bis die Zeit *tDriveTime* + 10% abgelaufen ist oder bis ein anderer Befehl an den Baustein gegeben wird. Durch eine positive Flanke am Eingang *bStop* werden beide Ausgänge unmittelbar zurückgesetzt.
- Mit den Eingang *bSwitch*, an dem üblicherweise ein Taster angeschlossen ist, kann die Jalousie hoch- bzw. heruntergetastet werden, wobei die Ausgänge im Gegensatz zum FB_VenetianBlindEx() [▶ 38]**sofort** in Selbsthaltung gehen. Die Funktionalität der Taste wechselt dabei immer von Fahren, Stoppen, Fahren in entgegen gesetzter Richtung, Stoppen und wieder Fahren. Die Ausgänge bleiben maximal für die Zeitdauer *tDriveTime* + 10% anstehen oder bis ein neuer Befehl an den Baustein gegeben wird. Für die Zeit tSwitchOverDeadTime ist ein Umschalten verriegelt, s.u. .
- In bestimmten Anwendungen kann es sinnvoll sein, wenn der Bediener die Stellung der Jalousie schrittweise ändern kann. Bei jeder positiven Flanke am Eingang *bStepUp* oder *bStepDown* wird der entsprechende Ausgang für die Zeit *tStepTime* gesetzt. Für *tStepTime* hat sich ein Wert von 200ms bewährt.
- Gegenüber dem Baustein FB_VenetianBlind() [▶ 36] kann mit diesem Baustein auch eine absolute Position angefahren werden. Hierzu wird ein Prozentwert an den Eingang *nSetPosition* angelegt und anschließend eine positive Flanke auf den Eingang *bPosition* gegeben.

Durch den Parameter *tSwitchOverDeadTime* kann verhindert werden, dass ein unmittelbarer Richtungswechsel den Antriebsmotor zerstört. In den meisten Fällen liegt dieser Wert zwischen 0,5sec und 1,0sec. Den genauen Wert erfahren sie beim Antriebshersteller.

Sicherheitsposition

Das Fahren in die Sicherheitsposition (z.B. bei starkem Wind oder bei Wartungsarbeiten am Fenster) kann durch Setzen des Eingangs *bSafetyPosition* erreicht werden. Für die Zeit *tDriveTime* + 10% wird der Ausgang *bBlindUp* gesetzt und der Ausgang *bBlindDown* zurückgesetzt. Die Bedienung der Jalousie ist so lange gesperrt, wie der Eingang *bSafetyPosition* aktiv ist.

Beschattungsposition

Bei überdurchschnittlicher Sonneneinstrahlung kann die Jalousie in die Beschattungsposition gefahren werden. Hierzu wird nach dem Anlegen einer positiven Flanke an den Eingang *bShadowPosition* die Jalousie auf die Position *nShadowSetPosition* gefahren. Anschließend wird für die Zeit *tShadowTurnAroundTime* die Jalousie wieder nach Oben gefahren. Hierdurch wird eine komplette Verdunkelung des Raumes verhindert. Wurde beim Anfahren der Beschattungsposition schon nach Oben gefahren, so wird die Jalousie für die Zeit *tDriveSwitchOverTime* - *tShadowTurnAroundTime* nach Unten gefahren. Es wird also der gleiche Winkel eingestellt, als wenn die Jalousie zum Verdunkeln nach Unten gefahren wäre.
Beim Richtungswechsel wird eine Pause von der Dauer *tSwitchOverDeadTime* eingehalten. Das Anfahren der Beschattungsposition kann jederzeit durch einen neuen Befehl unterbrochen werden.

Anfahren einer absoluten Position

Anmerkungen

Da in den meisten Fällen eine Jalousie keine Rückmeldung über die aktuelle Position zurück gibt, kann diese nur anhand der Fahrdauer berechnet werden. Die Genauigkeit ist davon abhängig, wie konstant die Geschwindigkeit der Jalousie ist. Außerdem sollten die Geschwindigkeitsunterschiede zwischen Auf-und Zufahren möglichst gering sein.
Die Positionen werden immer in Prozent angegeben. Dabei entspricht 0% ganz Oben und 100% ganz Unten. Wird ein Wert größer 100 angegeben, so wird dieser im Baustein auf 100 begrenzt.

Ermitteln der Parameter

Als erstes müssen bestimmte Parameter der Jalousie ermittelt werden. Zum Einen ist dieses die Fahrzeit. Also die Zeit, die benötigt wird, um die gesamte Strecke der Jalousie zu fahren. Zum Zweiten muß die Zeit ermittelt werden, die für ein Richtungswechsel benötigt wird. Während eines Richtungswechsel ändert sich der Winkel der einzelnen Lamellen. Die Fahrdauer wird an den Parameter *tDriveTime* übergeben. Die Dauer für einen Richtungswechsel an *tDriveSwitchOverTime*.

Baustein referenzieren

Da die aktuelle Position der Jalousie nur berechnet werden kann, summieren sich während des Betriebes die Ungenauigkeiten. Um die Abweichungen zu begrenzen, referenziert sich der Baustein möglichst häufig selbstständig. Dieses geschieht, wenn die Jalousie ganz nach Oben oder ganz nach Unten gefahren wird und der entsprechende Ausgang selbstständig zurückgesetzt wird. Also nach Ablauf der Zeit *tDriveTime* + 10%.

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: Ausgang *bBlindUp* setzen und Ausgang *bBlindDown* zurücksetzen. Der Ausgang *bBlindUp* bleibt in Selbsthaltung.

bDown: Ausgang *bBlindDown* setzen und Ausgang *bBlindUp* zurücksetzen. Der Ausgang *bBlindDown* bleibt in Selbsthaltung.

bStop: Ausgang *bBlindUp* und *bBlindDown* zurücksetzen.

bSwitch: Schateingang zum Tasten der Jalousie, siehe Beschreibung [▶ 38] oben. Wechselt in seiner Funktionalität von von Fahren, Stoppen, Fahren in entgegen gesetzter Richtung, Stoppen und wieder Fahren. Der jeweist angesteuerte Ausgang geht dabei **sofort** in Selbsthaltung.

bStepUp: Ausgang *bBlindDown* zurücksetzen und Ausgang *bBlindUp* für die Zeit *tStepTime setzen*.

bStepDown: Ausgang *bBlindUp* zurücksetzen und Ausgang *bBlindDown* für die Zeit *tStepTime setzen*.

tStepTime: Wird die Jalousie mit den Eingängen *bStepUp* oder *bStepDown* gesteuert, so bleiben die Ausgänge für diese Zeitdauer anstehen. Wird eine Zeitdauer von 0 angegeben, so werden die Ausgänge nicht gesetzt.

bPosition: Jalousie auf eine vorgegebene Position fahren.

nSetPosition: Position (0%-100%) auf die die Jalousie gefahren werden soll, nachdem an dem Eingang *bPosition* eine positive Flanke angelegt wurde. 0% entspricht ganz Oben, 100% entspricht ganz Unten.

bShadowPosition: Die Beschattungsposition wird angefahren (siehe unten).

nShadowSetPosition: Beschattungsposition (0%-100%) auf die die Jalousie gefahren werden soll, nachdem an dem Eingang *bShadowPosition* eine positive Flanke angelegt wurde.

tShadowTurnAroundTime: Nach dem Erreichen der Beschattungsposition wird für die Zeitdauer *tShadowTurnAroundTime* die Jalousie nach Oben gefahren.

bSafetyPosition: Sicherheitsposition wird angefahren. Hierfür wird für die Zeit *tDriveTime* + 10% die Jalousie hochgefahren. Solange der Eingang ansteht, ist die Bedienung der Jalousie gesperrt.

tDriveTime: Fahrdauer der Jalousie von ganz Oben nach ganz Unten. Wird kein Eingang betätigt, so werden nach der Zeitdauer *tDriveTime* + 10% die Ausgänge zurückgesetzt. Wird eine Zeitdauer von 0 angegeben, so werden die Ausgänge nicht automatisch zurückgesetzt. Das Anfahren der Jalousie auf absolute Positionen ist dann nicht möglich.

tDriveSwitchOverTime: Zeitdauer, die die Jalousie für einen Richtungswechsel benötigt.

tSwitchOverDeadTime: Verweildauer beim Richtungswechsel. Während dieser Zeit sind beide Ausgänge zurückgesetzt.

VAR_OUTPUT

```
bBlindUp            : BOOL;
bBlindDown          : BOOL;
nActualPosition     : USINT;
bCalibrated         : BOOL;
```

bBlindUp: Die Jalousie fährt hoch.

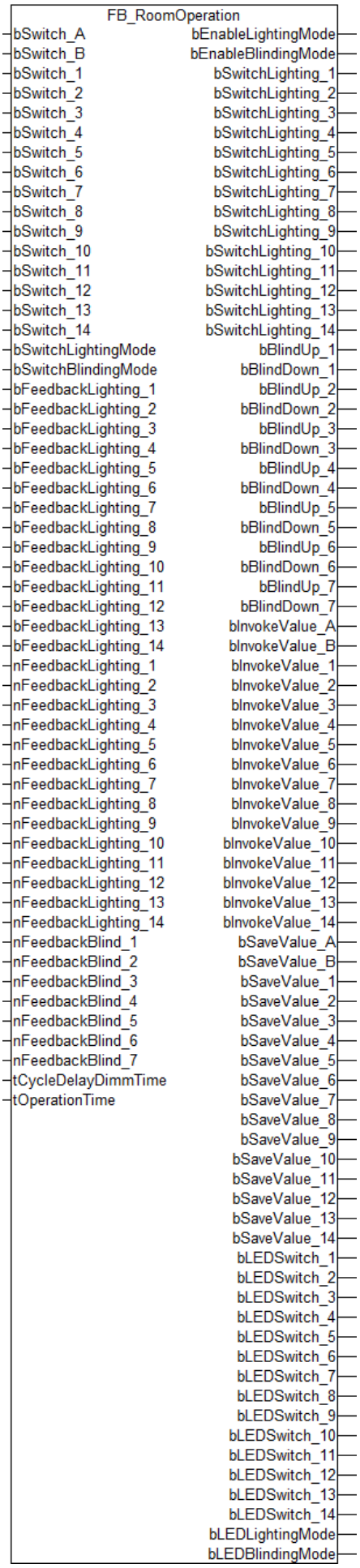
bBlindDown: Die Jalousie fährt runter.

nActualPosition: Aktuelle Position in Prozent.

bCalibrated: Gibt an, ob die Jalousie kalibriert ist.

3.3 Szenenverwaltung

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

Beschreibung:

Der Baustein FB_RoomOperation() ist für das Licht- und Jalousie- Management konzipiert. Im Ruhezustand werden Szenen aufgerufen und gedimmt. In dem entsprechenden Modus können Licht und Jalousien eingestellt und gespeichert werden. Dieser Baustein ist zur Benutzung mit den Bausteinen FB_ScenesLighting() [▶ 49], FB_ScenesVenetianBlind() [▶ 52], FB_Dimmer1Switch() [▶ 11] und FB_VenetianBlindEx() [▶ 38] vorgesehen.

Aufrufen von gespeicherten Szenen:

Durch einen steigende Flanke am Eingang *bSwitch_A, bSwitch_B oder bSwitch_1..14,* wird ein Impuls am Ausgang *bInvokeScene_A, bInvokeScene_B oder bInvokeScene_1..14* ausgegeben.

Dimmen von gespeicherten Szenen:

Durch ein Signal an Eingang *bSwitch_A, bSwitch_B oder bSwitch_1..14* das länger als die Zeit *tCycleDelayDimmTime* ansteht, wird eine Szenen aufgerufen und hoch gedimmt.

Einstellen Jalousie und Licht Werten:

Durch ein Signal am Eingang *bSwitchLightingMode* oder *bSwitchBlindingMode* wird in den jeweiligen Modus umgeschaltet. Hierbei werden durch die Eingänge, *bSwitch_1..14* die Stellgrössen über die Ausgänge *bSwitchLighting_1..14* oder *bSwitchBlindUp / bSwitchBlindDown_1..7* verändert.

Speichern der Einstellungen:

Durch das Setzen des Eingangs *bSwitchLightingMode* oder *bSwitchBlindingMode* und eines Signals am Eingang *bSwitch_A, bSwitch_B oder bSwitch_1..14* wird ein Impuls am Ausgang *bSaveScene_A, bSaveScene_B oder bSaveScene_1..14* ausgegeben. Die Werte werden im Baustein FB_ScenesLighting() [▶ 49], FB_ScenesVenetianBlind() [▶ 52] gespeichert.

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: Aufrufen der gespeicherten Szene A oder Szene B.

bSwitch_1..14: Einstellen und Aufrufen der gespeicherten Szenen.

bSwitchLightingMode: Umschalten in die Betriebsart Beleuchtung.

bSwitchBlindingMode: Umschalten in die Betriebsart Beschattung.

bFeedbackLighting_1..14: Aktueller Zustand der jeweiligen Lampe. Rückgabewert vom Dimmerbaustein FB_Dimmer1Switch() [▶ 11].

nFeedbackLighting_1..14: Aktuelle Stellgröße der jeweiligen Lampe. Rückgabewert vom Dimmerbaustein FB_Dimmer1Switch() [▶ 11].

nFeedbackBlind_1..7: Aktuelle Stellgröße der jeweiligen Jalousie. Rückgabewert vom Jalousie Baustein FB_VenetianBlindEx() [▶ 38].

t**CycleDelayDimmTime:** Umschaltzeit zwischen Dimmen und Aufrufen einer Szene.

tOperationTime: Ist die Betriebsart Beschattung oder Beleuchtung aktiv und es findet keine Bedienung satt, so wird nach Ablauf der Zeit selbstständig in den Szenenmodus zurückgeschaltet.

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: Freigabe des Speicherbausteins FB_ScenesLighting() [▶ 49].

bEnableBlindingMode: Freigabe des Speicherbausteins FB_ScenesVenetianBlind() [▶ 52].

bSwitchLighting_1..14: Ausgang zum Bedienen des Dimmer Bausteins FB_Dimmer1Switch() [▶ 11] über den Eingang *bSwitchDimm.*

bSwitchBlindUp_1..7: Ausgang zum Bedienen Jalousiebausteins FB_VenetianBlindEx() [▶ 38] über den Eingang *bSwitchOverUp.*

bSwitchBlindDown_1..7: Ausgang zum Bedienen des Jalousiebausteins FB_VenetianBlindEx() [▶ 38] über den Eingang *bSwitchOverDown*.

bInvokeScene_A, B, 1..14: Ausgangssignal zum Laden einer Szene. Wird an den Bausteinen FB_ScenesLighting() [▶ 49] und FB_ScenesVenetianBlind() [▶ 52] weitergegeben.

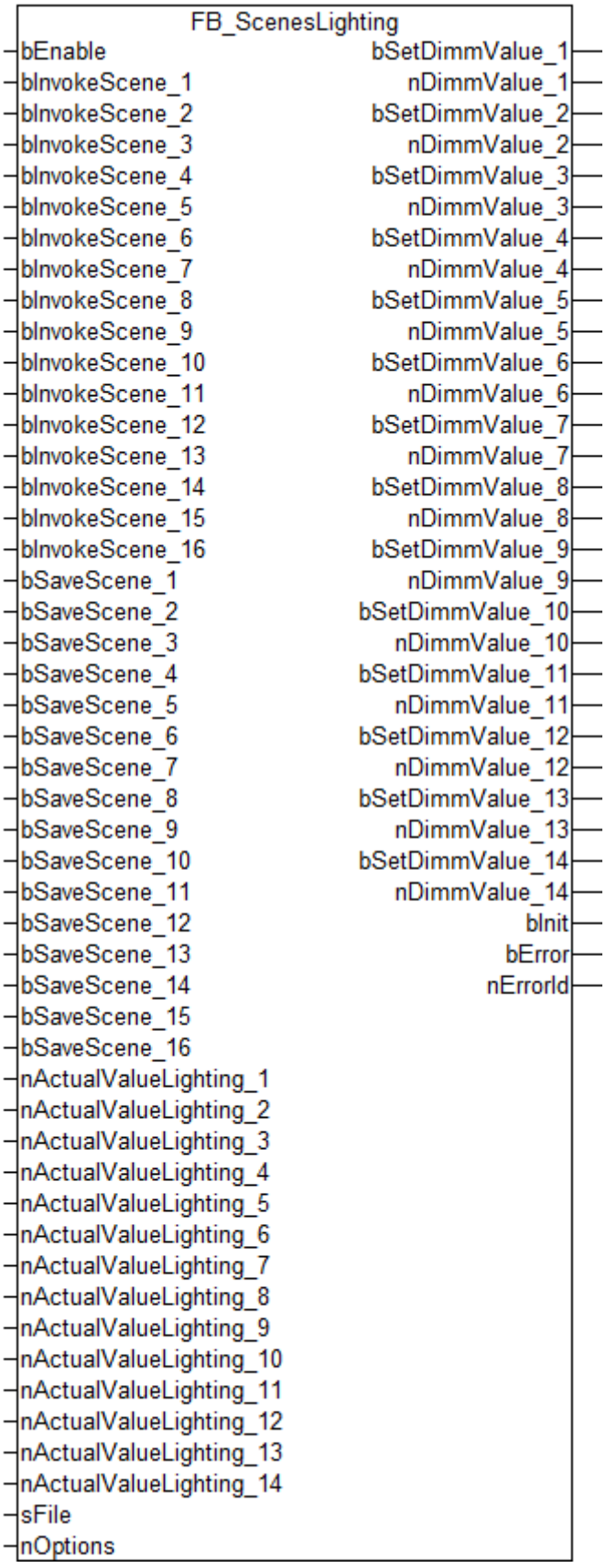
bSaveScene_A, B, 1..14: Ausgangssignal zum Speichern einer Szene. Wird an den Bausteinen FB_ScenesLighting() [▶ 49] und FB_ScenesVenetianBlind() [▶ 52] weitergegeben.

bLEDSwitch_1..14: Diese Ausgänge zeigen den Zustand der jeweiligen Beleuchtung (An / Aus) bzw. Beschattung (0% / 100%) an. Im Szenenmodus sind diese Ausgänge immer FALSE..

bLEDLightingMode: Dieser Ausgang ist TRUE, wenn die Betriebsart Beleuchtung aktiv ist.

bLEDBlindingMode: Dieser Ausgang ist TRUE, wenn die Betriebsart Beschattung aktiv ist.

3.3.2 FB_ScenesLighting



Beschreibung

Benutzung des Bausteins

Der Baustein ist zum Verwalten von Lichtszenen gedacht. Über den Eingang *bEnable* wird der Baustein freigeschaltet. Mit einer postiven Flanke an dem Eingang *bEnable* wird das Laden der gespeicherten Szenen gestartet. Der Eingang muss so lange TRUE sein, bis die Operation abgeschlossen ist. Die Werte der Szenen werden spannungsausfallsicher im TwinCAT Boot Verzeichnis als *.bin Datei gespeichert. Der letzte Datenstand wird als Sicherung in einer *.bak Datei gespeichert.

Scene speichern

Durch eine steigende Flanke am Eingang *bSaveScene_1...16* werden die Werte von den Eingängen *nActualValueLighting_1..14* in der jeweiligen Szene gespeichert.

Scenen laden

Durch eine steigende Flanke am Eingang *bInvokeScene_1..16* werden die gespeichteren Werte am Ausgang *nDimmValue_1..14* ausgegeben. Desweitern wird für ein SPS-Zyklus eine positive Flanke am Ausgang *bSetDimmValue_1..14* erzeugt.

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: Baustein freigeben.

bInvokeScene_1..16: Aufrufen der jeweiligen Szene.

bSaveScene_1..16: Speichern der aktuellen Analogwerte *nActualValueLighting_1..14* in der jeweiligen Szene.

nActualValueLighting_1..14: Aktuelle Stellgröße der jeweiligen Lampe. Rückgabewert vom Dimmerbaustein FB_Dimmer1Switch() [▶ 11].

sFile: Dateiname (ohne Pfad und Dateiendung) zur Sicherung der Szenen. Der Dateiname muß im gesamten Projekt eindeutig sein. Falls mehrere Instanzen von den Bausteinen FB_ScenesLighting() oder FB_ScenesVenetianBlind() [▶ 52] angelegt werden, so muß jede Instanze einen anderen Dateinamen benutzen. Die Datei wird immer in das TwinCAT Boot Verzeichnis abgelegt und erhält die Endung .bin. Beispiel: 'ControlPanelA'.

nOptions: Reserviert für zukünftige Entwicklungen.

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: Ausgang mit der Flanke für den Eingang b*SetDimmValue* vom Baustein FB_Dimmer1Switch() [▶ 11].

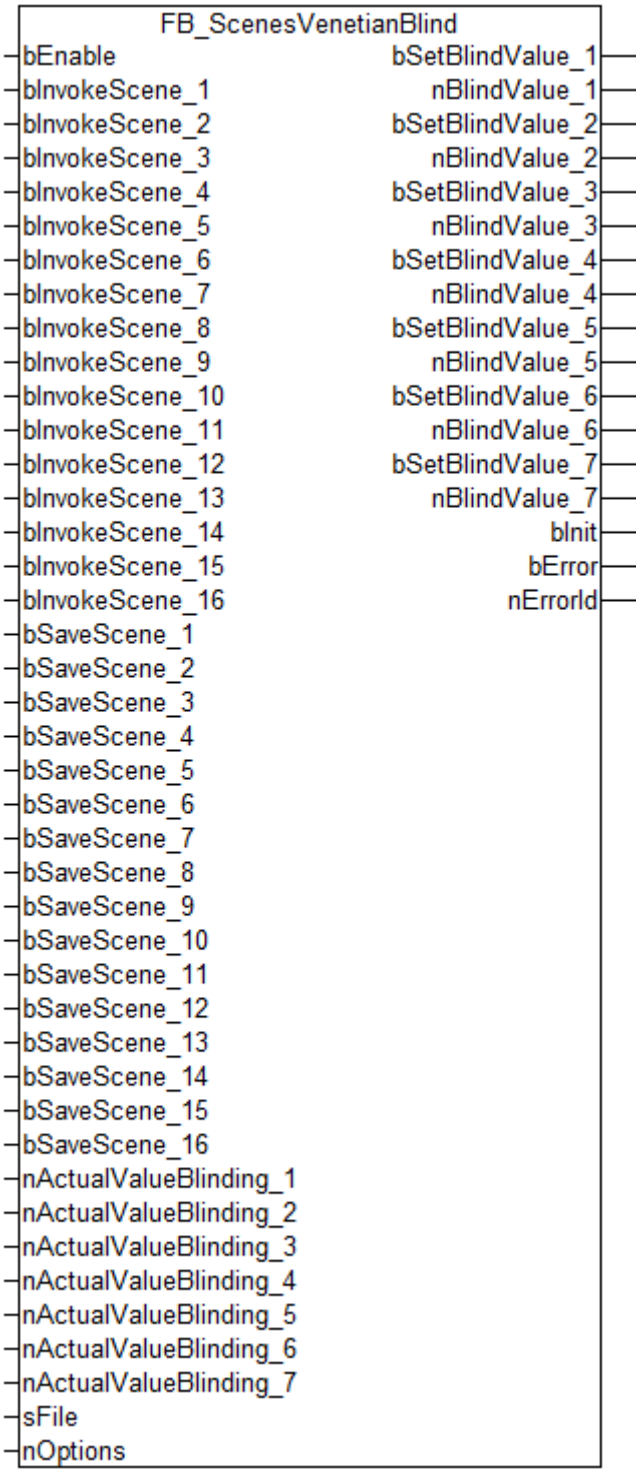
nDimmValue_1..14: Ausgang mit dem Wert für den Eingang *nDimmValue* vom Baustein FB_Dimmer1Switch() [▶ 11].

bInit: Sobald die Initialisierung des Bausteins abgeschlossen ist, ist der Ausgang TRUE.

bError: Dieser Ausgang wird auf TRUE gesetzt, sobald bei der Ausführung ein Fehler erkannt wurde. Der Fehlercode ist in *nErrorId* enthalten.

nErrorId: Enthält den Fehlerocde solbald *bError* auf TRUE ist. Siehe Fehlercodes [▶ 90].

3.3.3 FB_ScenesVenetianBlind



Beschreibung

Benutzung des Bausteins

Der Baustein ist zum Verwalten von Jalousieszenen gedacht. Über den Eingang *bEnable* wird der Baustein freigeschaltet. Mit einer postiven Flanke an dem Eingang *bEnable* wird das Laden der gespeicherten Szenen gestartet. Der Eingang muss so lange TRUE sein, bis die Operation abgeschlossen ist. Die Werte der Szenen werden spannungsausfallsicher im TwinCAT Boot Verzeichnis als *.bin Datei gespeichert. Der letzte Datenstand wird als Sicherung in einer *.bak Datei gespeichert.

Scene speichern

Durch eine steigende Flanke am Eingang *bSaveScene_1...16* werden die Werte von den Eingängen *nActualValueBlinding_1..7* in der jeweiligen Szene gespeichert.

Scenen laden

Durch eine steigende Flanke am Eingang *bInvokeScene_1..16* werden die gespeichteren Werte am Ausgang *nBlindValue_1..7* ausgegeben. Desweiteren wird für ein SPS-Zyklus eine positive Flanke am Ausgang *bSetBlindValue_1..7* erzeugt.

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: Baustein freigeben.

bInvokeScene_1..16: Aufrufen der jeweiligen Szene.

bSaveScene_1..16: Speichern der aktuellen Analogwerte *nActualValueBlinding_1..14* in der jeweiligen Szene.

nActualValueBlinding_1..7: Aktuelle Stellgröße der jeweiligen Jalousie. Rückgabewert vom Jalousiebaustein FB_VenetianBlindEx() [▶ 38].

sFile: Dateiname (ohne Pfad und Dateiendung) zur Sicherung der Szenen. Der Dateiname muß im gesamten Projekt eindeutig sein. Falls mehrere Instanzen von den Bausteinen FB_ScenesLighting() [▶ 49] oder FB_ScenesVenetianBlind() angelegt werden, so muß jede Instanze einen anderen Dateinamen benutzen. Die Datei wird immer in das TwinCAT Boot Verzeichnis abgelegt und erhält die Endung .bin. Beispiel: 'ControlPanelA'.

nOptions: Reserviert für zukünftige Entwicklungen.

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: Ausgang mit der Flanke für den Eingang *bPosition* vom Baustein FB_VenetianBlindEx() [▶ 38].

nBlindValue_1..7: Ausgang mit dem Wert für den Eingang *nSetPosition* vom Baustein FB_VenetianBlindEx() [▶ 38].

bInit: Sobald die Initialisierung des Bausteins abgeschlossen ist, ist der Ausgang TRUE.

bError: Dieser Ausgang wird auf TRUE gesetzt, sobald bei der Ausführung ein Fehler erkannt wurde. Der Fehlercode ist in *nErrorId* enthalten.

nErrorId: Enthält den Fehlerocde solbald *bError* auf TRUE ist. Siehe Fehlercodes [▶ 90].

3.4 Signalverarbeitung

3.4.1 FB_ShortLongClick

Liegt der Eingang *bSwitch* länger als *tSwitchTime* an, so wird für ein SPS-Zyklus der Ausgang *bLongClick* gesetzt. Ansonsten der Ausgang *bShortClick*.

VAR_INPUT

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

bSwitch: Eingangssignal.

tSwitchTime: Zeitdauer, ab der das Eingangssignal als langer Tastendruck interpretiert wird.

VAR_OUTPUT

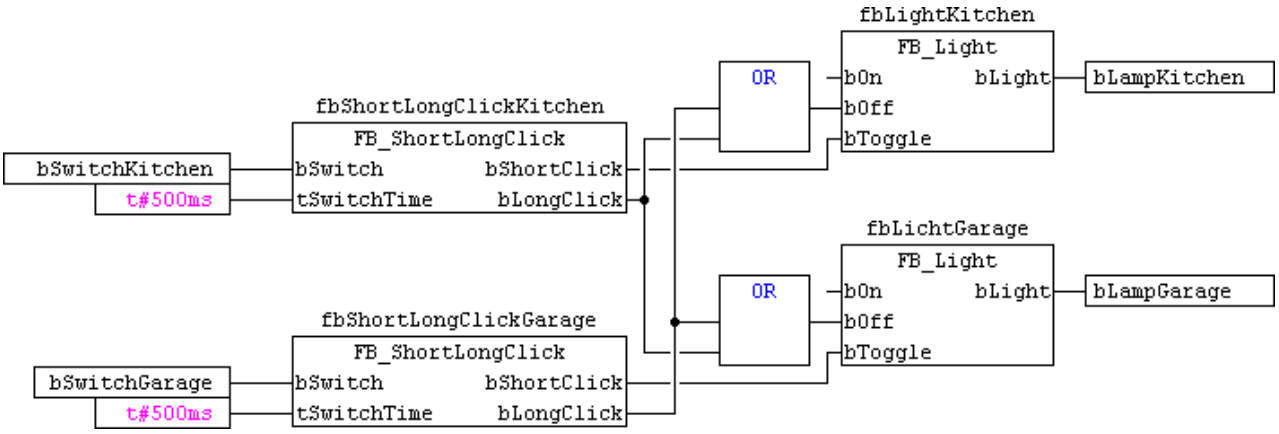
```
bShortClick      : BOOL;
bLongClick       : BOOL;
```

bShortClick: Signalisiert einen kurzen Tastendruck.

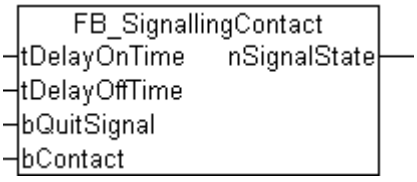
bLongClick: Signalisiert einen langen Tastendruck.

Beispiel

Bei dem folgenden Beispiel werden mit zwei Taster zwei verschiedene Lampen geschaltet. Jeder Lampe ist ein Schalter zugeordnet. Wird ein Schalter länger als 500ms betätigt, so werden beide Lampen ausgeschaltet.



3.4.2 FB_SignallingContact



Über die beiden Eingänge *tDelayOnTime* und *tDelayOffTime* kann eine Abfall- und Anzugsverzögerung eingestellt werden. Soll eine Meldung quittiert werden bevor diese beendet werden kann, so wird dieses über den Eingang *bQuitSignal* realisiert. Der Meldekontakt wird über den Eingang *bContact* an den Baustein übergeben.

Der Zustand der Meldung wird über den Ausgang *nSignalState* signalisiert. Insgesamt kann eine Meldung 6 verschiedene Zustände annehmen. In der Library sind entsprechende Konstanten definiert:

Konstante	Beschreibung
TCSIGNAL_INVALID	Die Meldung hat noch keinen definierten Zustand.
TCSIGNAL_SIGNALED	Die Meldung ist aktiv.
TCSIGNAL_RESET	Die Meldung wurde zurückgesetzt.
TCSIGNAL_CONFIRMED	Die Meldung ist bestätigt aber noch nicht zurückgesetzt.
TCSIGNAL_SIGNALCON	Die Meldung ist aktiv und bestätigt.
TCSIGNAL_RESETCON	Die Meldung ist bestätigt und zurückgesetzt.

VAR_INPUT

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

tDelayOnTime: Verzögerung für das Setzen der Meldung.

tDelayOffTime: Verzögerung für das Rücksetzen der Meldung.

bQuitSignal: Eingang um eine Meldung zu quittieren.

bContact: Eingang für den Meldekontakt.

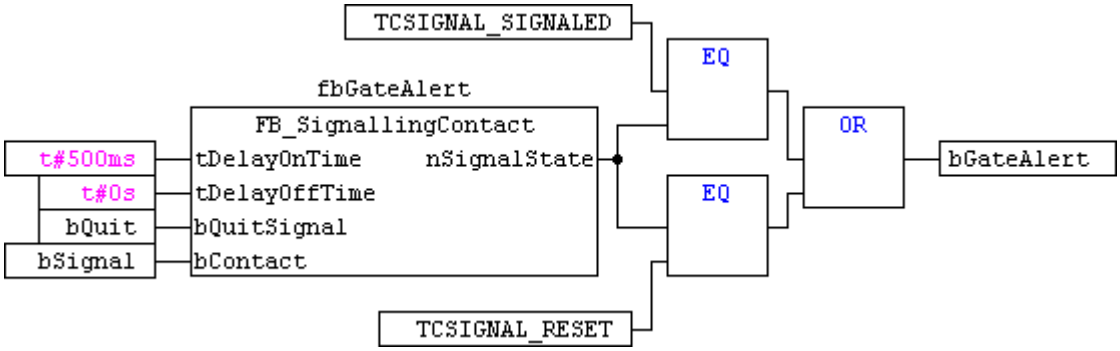
VAR_OUTPUT

```
nSignalState     : WORD;
```

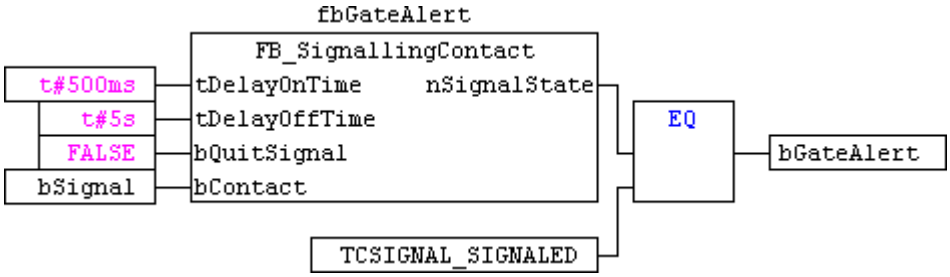
nSignalState: Zustand der Meldung.

Beispiele

Mit dem folgenden Beispiel wird eine quittierungspflichtige Meldung realisiert. Die Variable *bGateAlert* stellt den Zustand der Meldung dar. Hat der Ausgang *nSignalState* den Wert *TCSIGNAL_SIGNALED* oder *TCSIGNAL_RESET*, so ist die Meldung aktiv. Durch eine positive Flanke an dem Eingang *bQuitSignal*, wird die Meldung quittiert.

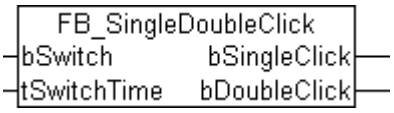


Das folgende Beispiel stellt den einfachsten Fall dar. Eine nicht quittierungspflichtige Meldung.



Durch die Abfallverzögerung wird erreicht, das die Meldung eine bestimmte Zeit aktiv bleibt. Mit Hilfe der Anzugsverzögerung kann z.B. das Prellen des Kontaktes unterdrückt werden.

3.4.3 FB_SingleDoubleClick



Wird innerhalb der Zeit *tSwitchTime* zweimal das Eingangssignal angelegt, so wird der Ausgang *bDoubleClick* für ein SPS-Zyklus gesetzt. Ansonsten der Ausgang *bSingleClick*.

VAR_INPUT

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

bSwitch: Eingangssignal.

tSwitchTime: Zeitdauer, ab der das Eingangssignal als doppelter Tastendruck interpretiert wird.

VAR_OUTPUT

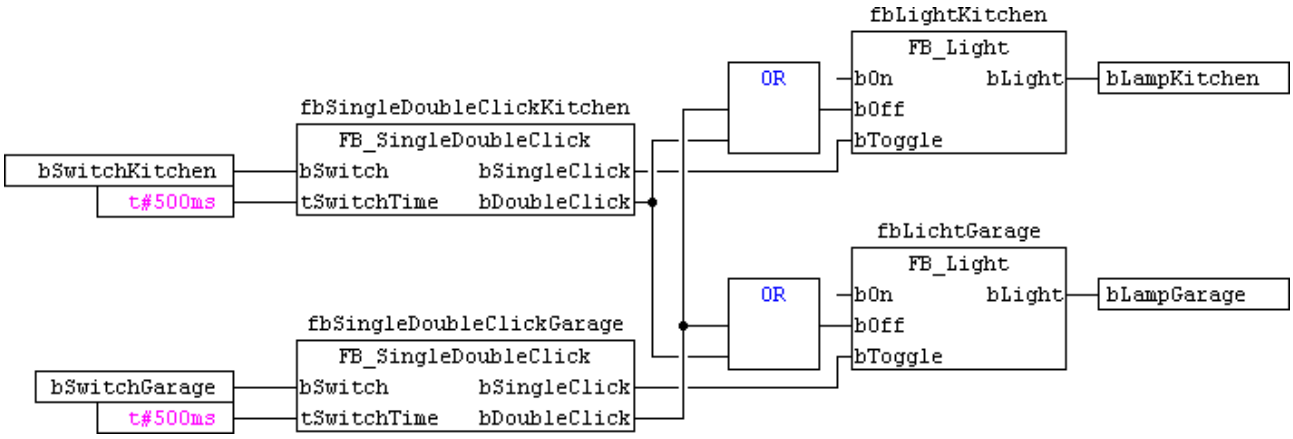
```
bSingleClick     : BOOL;
bDoublelick      : BOOL;
```

bSingleClick: Signalisiert einen einfachen Tastendruck.

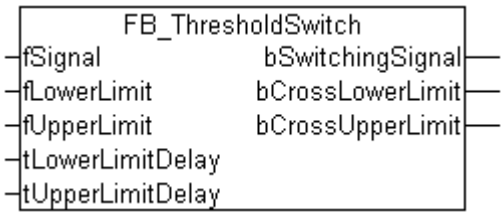
bDoublelick: Signalisiert einen doppelten Tastendruck.

Beispiel

Bei dem folgenden Beispiel können mit zwei Taster zwei verschiedene Lampen geschaltet werden. Jeder Lampe ist ein Schalter zugeordnet. Wird ein Schalter zweimal schnell hintereinander betätigt, so werden beide Lampen ausgeschaltet.



3.4.4 FB_ThresholdSwitch



Überschreitet das Eingangssignal den Grenzwert *fUpperLimit* für die Zeitdauer *tUpperLimitDelay*, so wird der Ausgang *bCrossUpperLimit* für ein SPS-Zyklus gesetzt. Der Ausgang *bSwitchingSignal* wird ebenfalls gesetzt. Dieser bleibt solange gesetzt bis das Eingangssignal den Grenzwert *fLowerLimit* für die Zeitdauer *tLowerLimitDelay* unterschreitet. Der Ausgang *fCrossLowerLimit* wird hierbei für ein SPS-Zyklus gesetzt.

VAR_INPUT

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

fSignal: Eingangssignal.

fLowerLimit: Unterer Grenzwert.

fUpperLimit: Oberer Grenzwert.

tLowerLimitDelay: Schaltverzögerung beim Unterschreiten des unteren Grenzwertes.

tUpperLimitDelay: Schaltverzögerung beim Überschreiten des oberen Grenzwertes.

VAR_OUTPUT

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

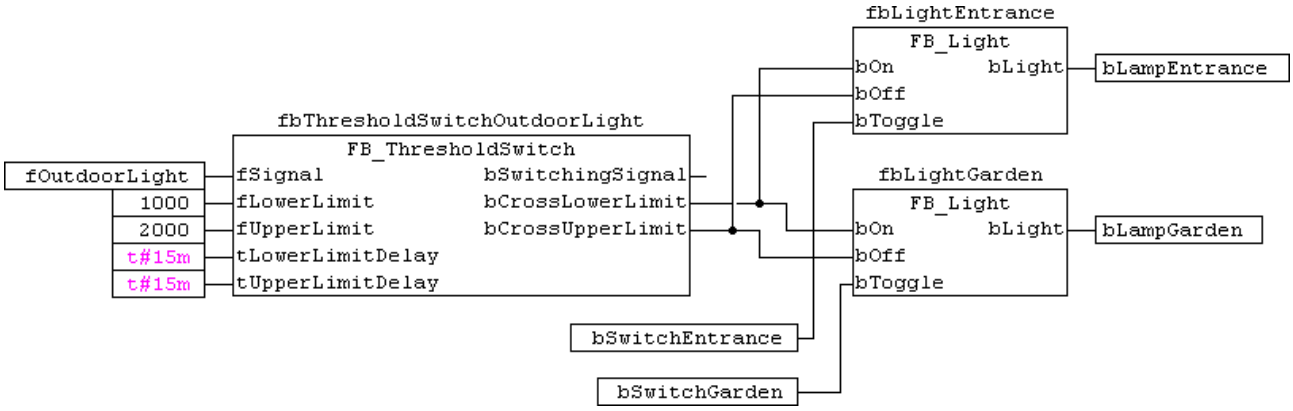
bSwitchingSignal: Zustand ist abhängig von *bCrossLowerLimit* und *bCrossUpperLimit*.

bCrossLowerLimit: Wird für 1 Zyklus TRUE, wenn *fLowerLimit* für die Zeit *tLowerLimitDelay* unterschritten wurde. Gleichzeitig wird *bSwitchingSignal* FALSE.

bCrossUpperLimit: Wird für 1 Zyklus TRUE, wenn *fUpperLimit* für die Zeit *tUpperLimitDelay* überschritten wurde. Gleichzeitig wird *bSwitchingSignal* TRUE.

Beispiel

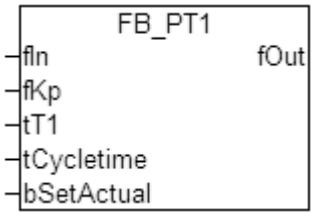
Bei dem folgenden Beispiel können die beiden Lampen mit je einem Schalter bedient werden. Mit Hilfe der Außenhelligkeit und des Schwellwertschalters werden die beiden Lampen automatisch geschaltet. Ist die Außenhelligkeit für 15min kleiner als 1000lux, so werden die Lampen eingeschaltet. Sobald die Helligkeit länger als 15min größer 2000lux ist, werden die Lampen ausgeschaltet.



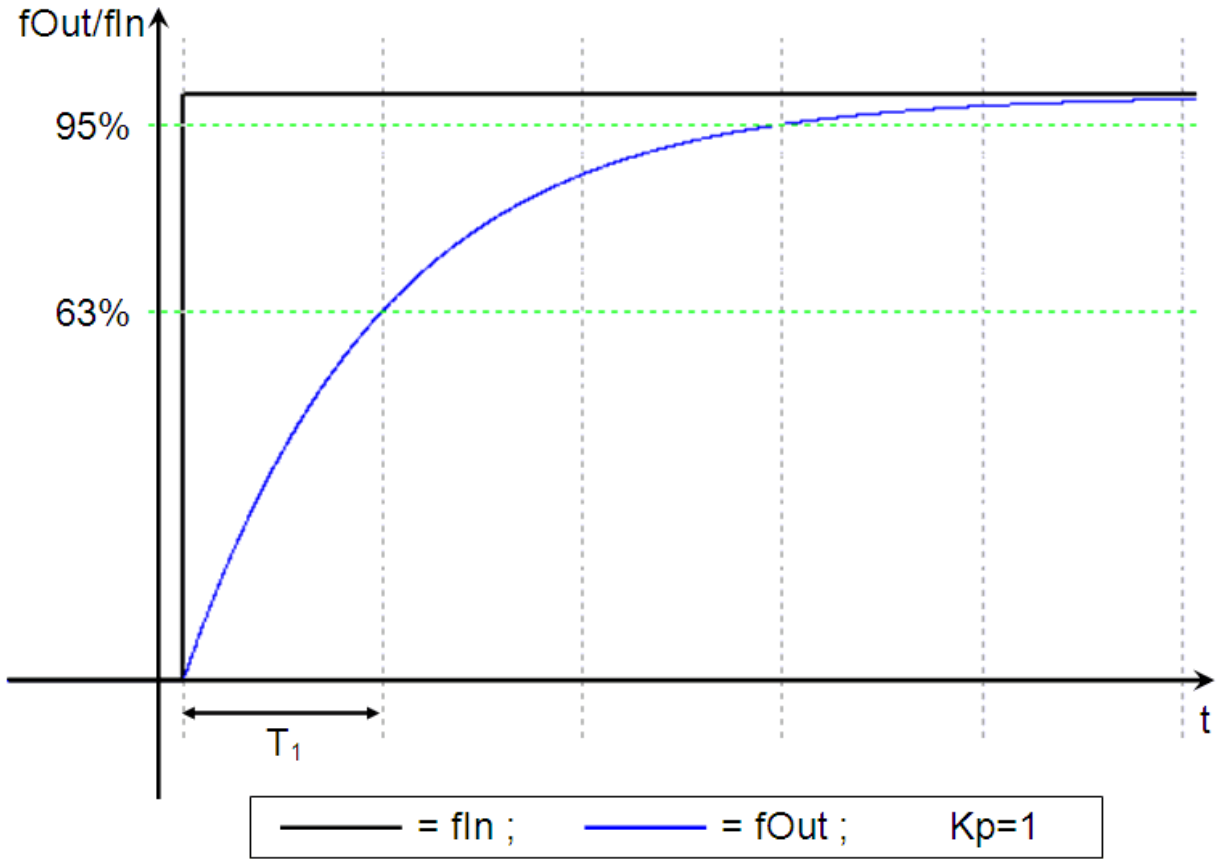
3.5 Filterfunktionen

3.5.1 FB_PT1

PT_1-Glied zur Glättung von Eingangsgrößen.



Dieser Baustein arbeitet kontinuierlich. Der Ausgang *fOut* folgt dabei immer dem Eingangswert *fIn* multipliziert mit *Kp* mit einer Exponentialkurve:



Am Beispiel mit *Kp*=1 würde der Ausgangswert direkt dem Eingangswert folgen. Dabei hat *fOut* nach Ablauf der Zeit *tT1* bereits 63% des Eingangswertes erreicht und nach Ablauf von 3 x *tT1* sind es 95%.

Die mathematische Formel lautet:

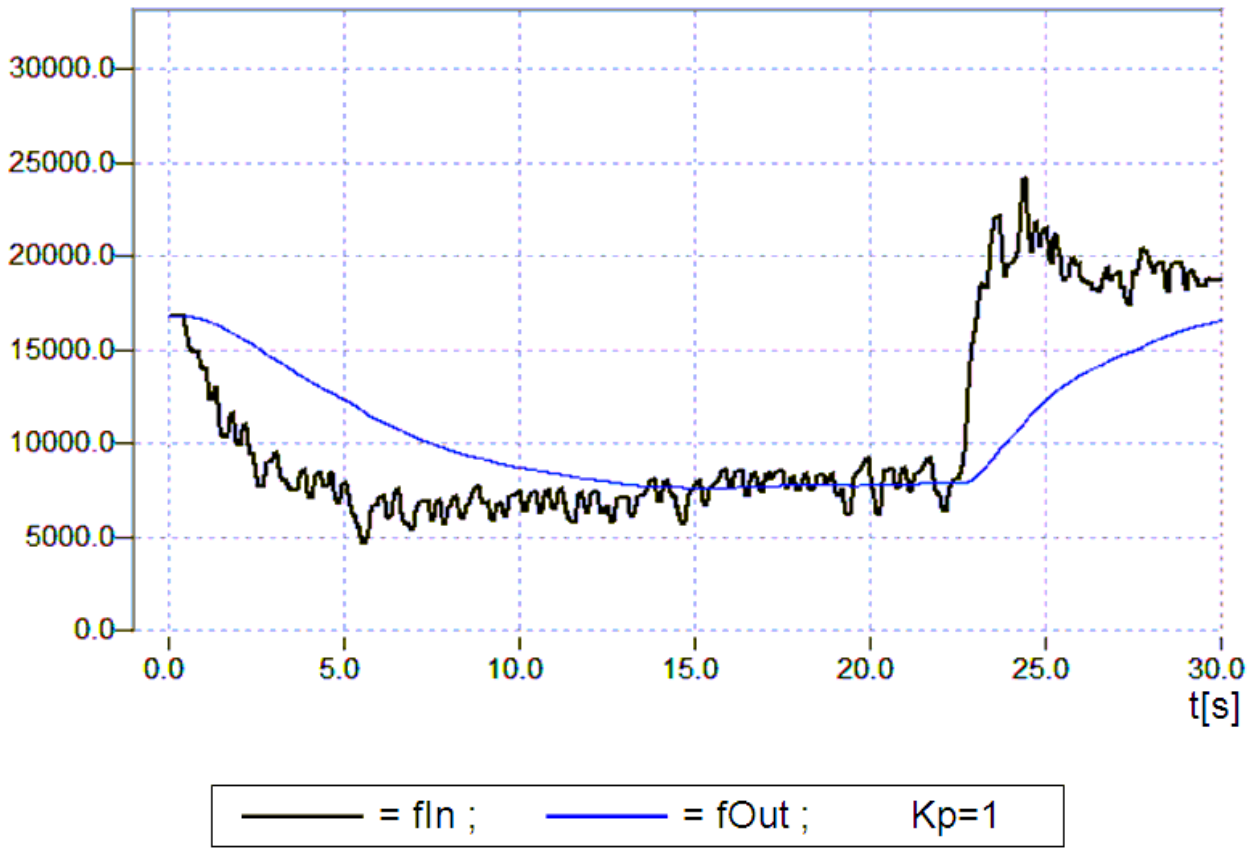
$$f(t) = K_p(1 - e^{-\frac{t}{T_1}})$$

Als zeitdiskrete Formel für die Berechnung in der PLC gilt:

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

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

Bei einem sich kontinuierlich veränderten Eingang *fIn* verhält sich *fOut* wie folgt (*fIn*= 0..33000, *Kp*= 1, *T1*= 5s):



Anmerkung zu den Dämpfungszeiten: Da es sich bei diesem Baustein um ein zeitdiskretes Modell eines PT_1-Gliedes handelt, arbeitet es nur korrekt, wenn die Dämpfungszeit sehr viel größer als die eingestellte Zykluszeit ist. Zur Sicherheit wird eine eingetragene Dämpfungszeit, die kleiner als das Doppelte der eingestellten Zykluszeit ist, intern zu Null gesetzt. Eine Dämpfungszeit von 0s bedeutet, daß die Ausgangsgröße der Eingangsgröße multipliziert mit Kp direkt folgt.

VAR_INPUT

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

fIn: Eingangswert.

fKp: Verstärkungsfaktor, voreingestellter Wert: 1.

tT1: Dämpfungszeit, voreingestellter Wert: 10s.

tCycleTime: PLC-Zykluszeit, voreingestellter Wert: 10ms.

bSetActual: Setzt den Ausgang *fOut* direkt auf den Eingangswert *fIn*.

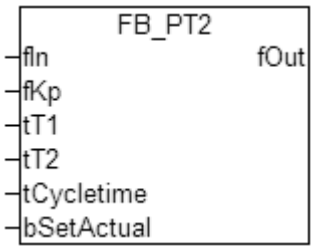
VAR_OUTPUT

```
fOut        : LREAL;
```

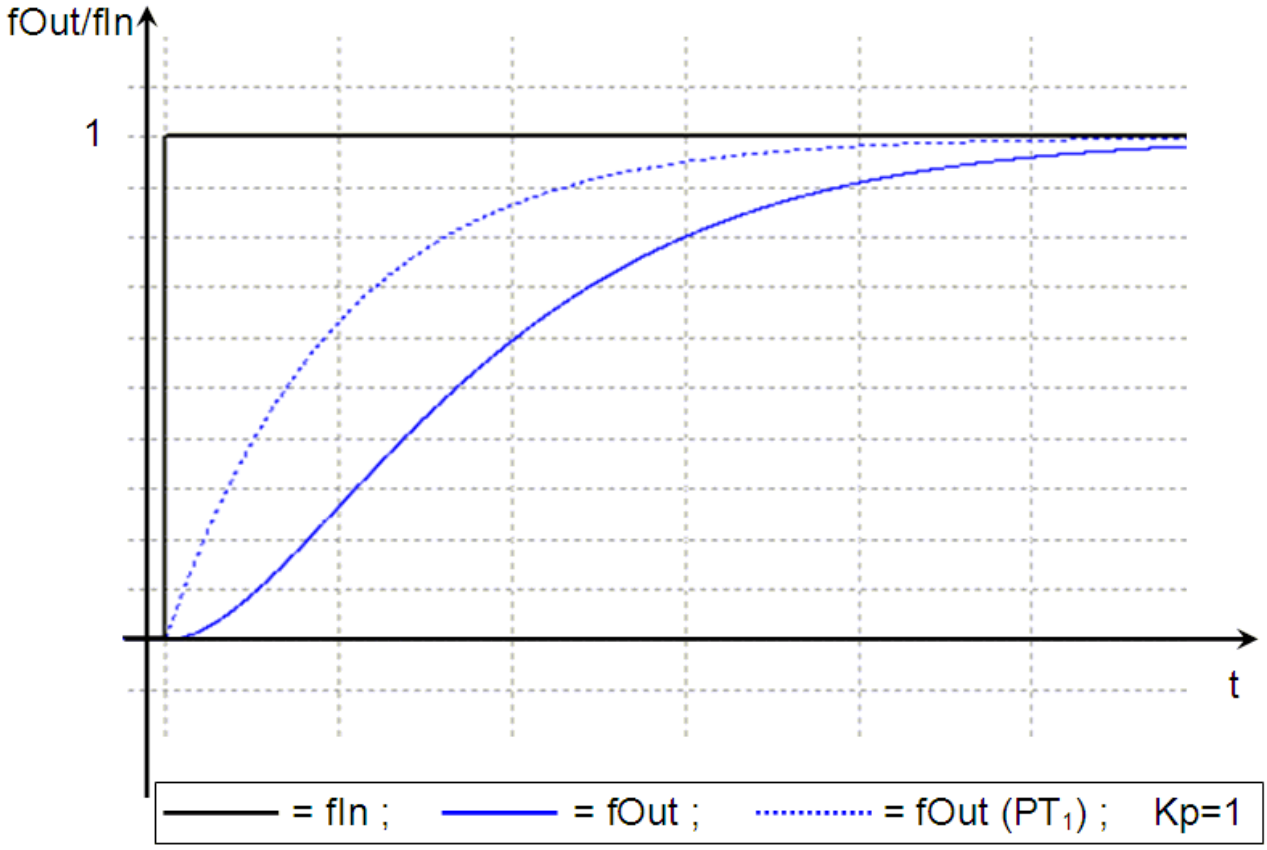
fOut: Ausgangswert.

3.5.2 FB_PT2

PT_2-Glied zur Glättung von Eingangsgrößen.

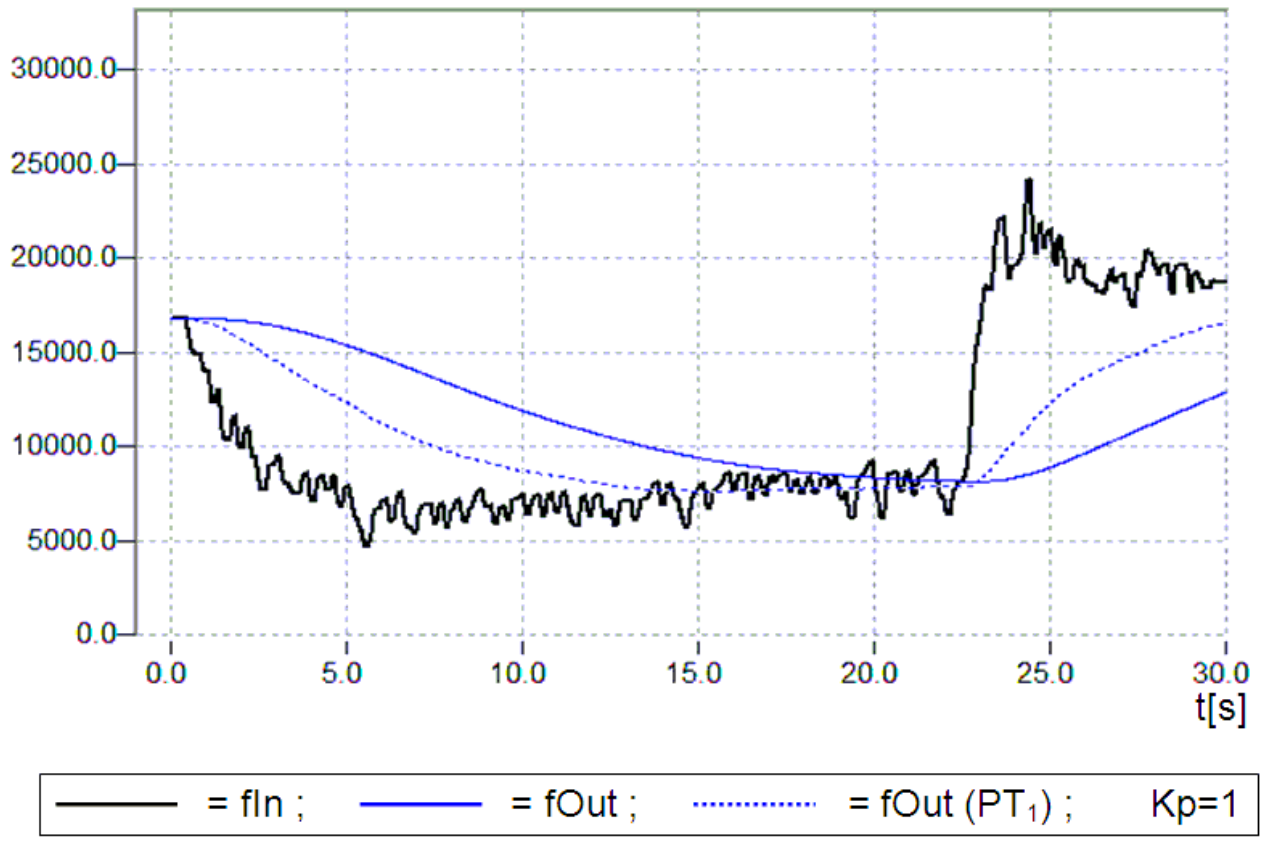


Dieser Baustein arbeitet kontinuierlich. Der Ausgang *fOut* folgt dabei immer dem Eingangswert *fIn* multipliziert mit *Kp*.



Dieses PT_2-Glied besteht aus der Aneinanderreihung zweier PT_1-Glieder, wobei die Zeitkonstanten, T1 und T2 unterschiedliche Werte innehaben können. Die Sprungantwort (s.o.) zeigt ein deutlich gedämpfteres Folgeverhalten im Vergleich zum PT_1-Glied (gestrichelt) vom ersten Moment an.

Bei einem sich kontinuierlich veränderten Eingang *fIn* verhält sich *fOut* wie folgt (*fIn*= 0..33000, *Kp*= 1, *T1,T2*= 5s):



Die gestrichelte Linie zeigt im Vergleich dazu das Verhalten eines PT1-Gliedes [▶ 58] mit *fIn*= 0..33000, *Kp*= 1, *T1*= 5s.

Anmerkung zu den Dämpfungszeiten: Da es sich bei diesem Baustein um ein zeitdiskretes Modell eines PT_2-Gliedes handelt, arbeitet es nur korrekt, wenn die Dämpfungszeiten sehr viel größer als die eingestellte Zykluszeit sind. Zur Sicherheit werden eingetragene Dämpfungszeit, die kleiner als das Doppelte der eingestellten Zykluszeit sind, intern zu Null gesetzt. Wie bereits erwähnt, besteht das PT_2-Glied aus zwei In Reihe geschalteten PT_1-Gliedern. Ist eine der beiden Dämpfungszeiten zu Null gesetzt, so wird das PT_2-Glied zu einem PT_1-Glied reduziert. Sind beide Dämpfungszeiten auf Null gesetzt, so folgt die Ausgangsgröße direkt der Eingangsgröße multipliziert mit Kp.

VAR_INPUT

```
fIn          : LREAL;
fKp          : LREAL := 1;
tT1          : TIME := t#10s;
tT2          : TIME := t#10s;
tCycletime   : TIME := t#10ms;
bSetActual   : BOOL;
```

fIn: Eingangswert.

fKp: Verstärkungsfaktor, voreingestellter Wert: 1.

tT1: Dämpfungszeit 1, voreingestellter Wert: 10s.

tT2: Dämpfungszeit 2, voreingestellter Wert: 10s.

tCycleTime: PLC-Zykluszeit, voreingestellter Wert: 10ms.

bSetActual: Setzt den Ausgang *fOut* direkt auf den Eingangswert *fIn*.

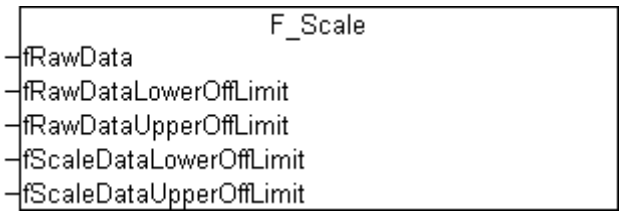
VAR_OUTPUT

```
fOut         : LREAL;
```

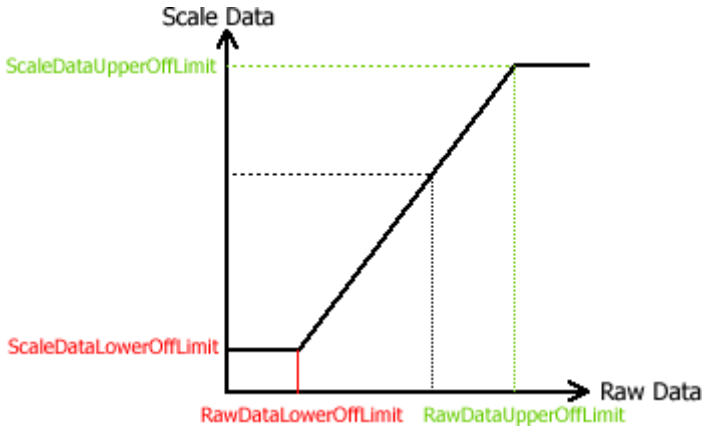
fOut: Ausgangswert.

3.6 Umrechnungsfunktionen

3.6.1 F_Scale



Ein analoger Rohwert wird auf den angegebenen Messbereich skaliert und als Funktionswert zurückgegeben. Überschreitet der Rohwert den oberen oder unteren Messbereich, so wird der entsprechende Grenzwert ausgegeben. Zwischen dem oberen und unteren Grenzwert der Rohdaten, muss mindestens ein Unterschied von 0.01 vorhanden sein. Ist dieses nicht der Fall, so wird der untere Grenzwert ausgegeben.



VAR_INPUT

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

fRawData: Rohwert.

fRawDataLowerOffLimit: unterer Grenzwert vom Rohwert.

fRawDataUpperOffLimit: oberer Grenzwert vom Rohwert.

fScaleDataLowerOffLimit: unterer Grenzwert vom skalierten Messwert.

fScaleDataUpperOffLimit: oberer Grenzwert vom skalierten Messwert.

3.6.2 Temperaturumrechnungsfunktionen

zwischen Kelvin, Celsius, Reaumur und 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**

Übersicht

	Kelvin (K)	Grad Celsius (°C)	Reaumur (°R)	Fahrenheit (°F)
absoluter Nullpunkt	0	-273,15	-218,52	-459,67
Schmelzpunkt	273,15	0	0	32
Siedepunkt	373,15	100	80	212

(Schmelz- und Siedepunkt beziehen sich auf reines Wasser)

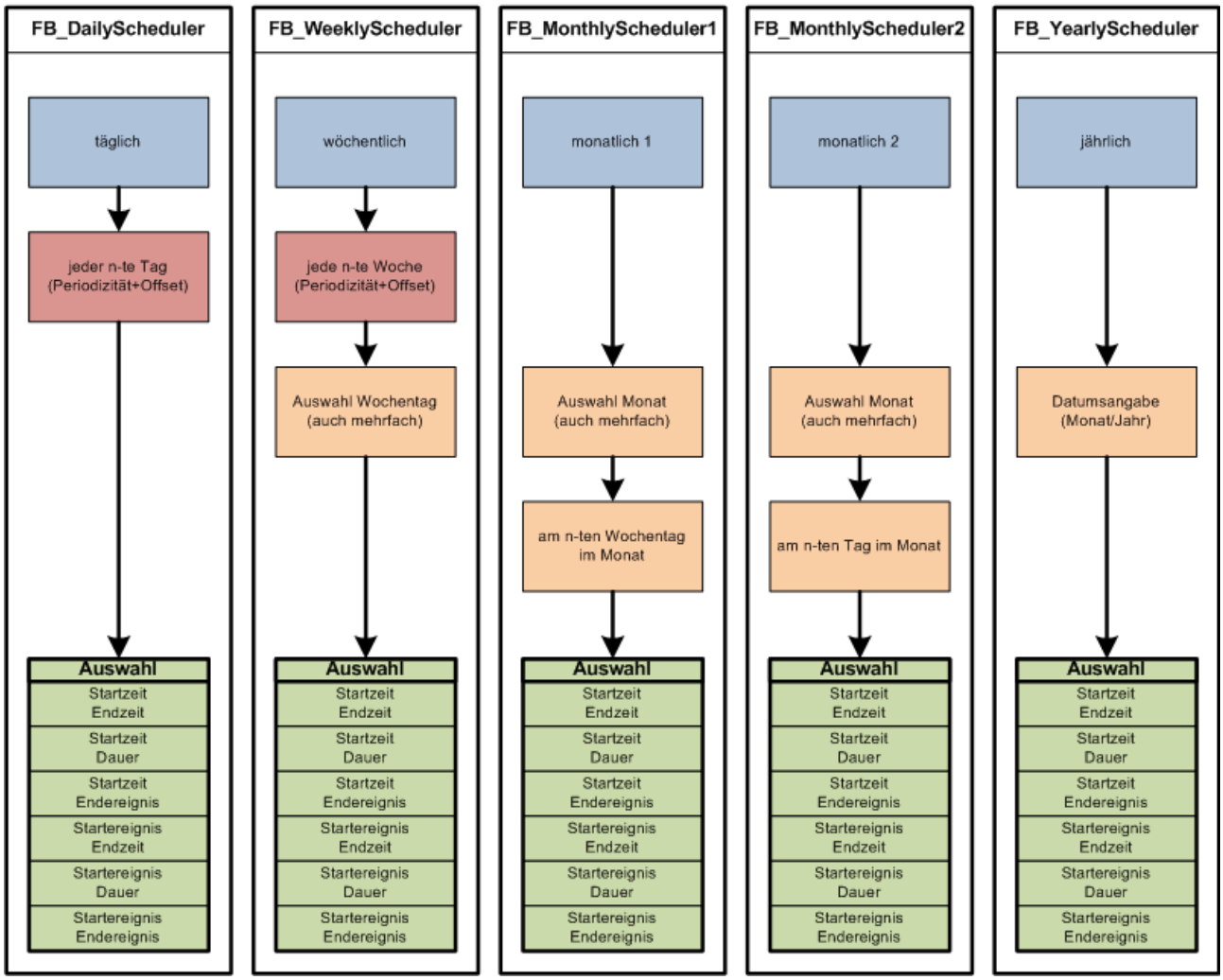
Umrechnungsregeln

	Kelvin (K)	Grad 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 = Grad 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 Zeitschaltfunktionen

3.7.1 Schaltuhr Übersicht

Die Zeitschaltuhr-Bausteine dienen dazu, an bestimmten Tagen im Jahr/ im Monat/ in der Woche jeweils eine Aktion auszuführen. Dabei kann die Aktion durch ein Startereignis oder einen Startzeitpunkt begonnen und durch ein Endereignis, einem Endzeitpunkt oder einer Dauer beendet werden. Es ergeben sich folgende Kombinationen:



Die graublauen Felder bezeichnen die Art der Zeitschaltuhr. Die Tagesauswahl wird bestimmt durch die Periodizität (rote Felder) und weitergehende Diskretisieren (orange). Alle Bausteine haben gemeinsam, dass sie die gleiche Auswahl an Start- und Endkriterien besitzen (grün). Das Startkriterium bezieht sich auf den zuvor ausgewählten Tag, das Endkriterium steht immer in Abhängigkeit zum Startpunkt. Pro Instanz eines Funktionsbausteines ist immer nur ein Start- und Endkriterium definierbar. Für mehrere Aktionen an ein und demselben Tag sind mehrere Instanzen des Funktionsbausteines nötig.

Zeitliche Überschneidungen

Zeitliche Überschneidungen Bei ein und derselben Instanz des Funktionsbausteines können zeitliche Überschneidungen von zwei aufeinanderfolgenden Ein- und Ausschaltkriterien dann auftreten, wenn die Schaltdauer nicht auf unter 1 Tag begrenzt ist. In diesem Fall kann es geschehen, dass einem Start ein weiterer folgt, ohne dass ein Ende der vorhergehenden Periode eingetreten ist . Bei den genannten Auswahlen ergeben sich folgende Überschneidungsmöglichkeiten:

Starttime / Endtime (Typ TOD, TOD)
Keine Überschneidung möglich da bei *Starttime<Endtime* Start- und Endpunkt am selben Tag sind und umgekehrt bei Starttime>=Endtime davon ausgegangen wird, dass der Endpunkt am nächsten Tag ist. Damit ist die Dauer auf unter 1 Tag begrenzt.

Starttime / Duration (Typ TOD, TIME)
Überschneidung möglich, da die Dauer frei wählbar ist und der TIME-Variablentyp bis knapp 50 Tage reicht. Somit wäre es beispielsweise möglich, täglich eine Aktion von 3 Tagen zu starten. Durch das fortwährende Nachstarten würde diese Aktion dann nie beendet werden.

Starttime / Endevent (Typ TOD, BOOL)
Überschneidung möglich, da das Endereignis (Endevent) in seinem Auftreten variabel ist und durchaus **nicht vor** dem nächsten Startzeitpunkt erfolgen kann.

Startevent / Endtime (Typ BOOL, TOD)
Überschneidung teilweise möglich. Beim Auftreten des Startevent wird der Endzeitpunkt errechnet. Ist *Starttime<Endtime* wird der Endzeitpunkt am selben Tag liegen. Dann gibt es keine Überschneidungsmöglichkeit. Umgekehrt, bei *Starttime>=Endtime* wird der Endpunkt auf den nächsten Tag gelegt. Ereignet sich an diesem Tag der Start bevor das Ende des vorherigen erreicht ist, so kommt es zur Überschneidung.

Startevent / Duration (Typ BOOL, TIME)
Überschneidung möglich, da die Dauer frei wählbar ist und der TIME-Variablentyp bis knapp 50 Tage reicht. Somit wäre es beispielsweise möglich, täglich eine Aktion von 3 Tagen zu starten. Durch das fortwährende Nachstarten würde diese Aktion dann nie beendet werden.

Startevent / Endevent (Typ BOOL, BOOL)
Überschneidung möglich, da das Endereignis (Endevent) in seinem Auftreten variabel ist und durchaus **nicht vor** dem nächsten Startereignis (Startevent) erfolgen kann.

Eine Überschneidung hat zur Folge, dass der Steuerausgang *bOut* des jeweiligen Funktionsbausteines nicht auf FALSE wechselt. Statt dessen wird das nächste Periodenende abgewartet. Da es jedoch auch beabsichtigt sein kann, dass die Bausteine eine Aktion lediglich "antriggern", ist auch der Flankenausgang *bTriggerOn*an denBausteinen mit herausgeführt.

Weitergehende Dokumentation

Die folgende Tabelle führt zu den genauen Dokumentationen der einzelnen Bausteine:

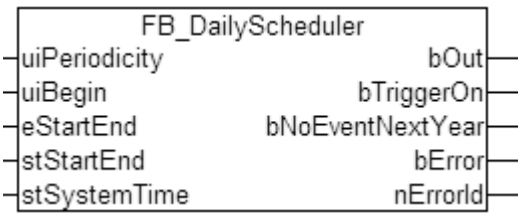
FB_DailyScheduler() [▶ 66]	FB_WeeklyScheduler() [▶ 68]	FB_MonthlyScheduler1() [▶ 69]	FB_MonthlyScheduler2() [▶ 70]	FB_YearlyScheduler() [▶ 72]
schaltet jeden n-ten Tag	schaltet jede n-te Woche an bestimmten Wochentagen (Mehrfachauswahl möglich)	schaltet in bestimmten Monaten (Mehrfachauswahl möglich) an einem bestimmten Wochentag	schaltet in bestimmten Monaten (Mehrfachauswahl möglich) an einem bestimmten Tag im Monat	schaltet an einem bestimmten Tag im Jahr

Beispielprogramm

In einem Beispielprogramm [▶ 73] wird anhand eines Tages-Schaltbausteines (*FB_DailyScheduler*) gezeigt, wie die Bausteine zu parametrieren sind.

3.7.1.1 FB_DailyScheduler

Funktionsbaustein zum Schalten von Aktionen an jedem n-ten Tag im Jahr.



Hinweis: Der Funktionsbaustein benötigt zum Schalten das Durchschreiten des Zeitschaltpunktes. Ein nachträgliches Ändern der Schaltereignisse bzw. der Zeit ist daher nicht zulässig.

VAR_INPUT

```
uiPeriodicity  : UINT;
uiBegin        : UINT;
eStartEnd      : ENUM;
stStartEnd     : TIMESTRUCT;
stSystemtime   : TIMESTRUCT;
```

uiPeriodicity: Periodizität bzw. Intervall. Darf im Bereich von 1..365 liegen.

uiBegin: Startwert für den Tageszähler. Darf im Bereich von 1..365 liegen. Beispiel: uiPeriodicity=5, uiBegin=2: Schaltereignisse am 2.Jan., 7.Jan. 12.Jan. etc. - uiPeriodicity=3, uiBegin=1: Schaltereignisse am 1.Jan., 4.Jan. 7.Jan. etc.

eStartEnd: Auswahl der 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: Startzeit/Endzeit-Auswahl. Ist die Start-Tageszeit größer oder gleich der End-Tageszeit, so wird das Ende auf den nächsten Tag gelegt.

eSTARTTIME_DURATION: Startzeit/Dauer-Auswahl.

eSTARTTIME_ENDEVENT: Startzeit/Endereignis-Auswahl.

eSTARTEVENT_ENDTIME: Startereignis/Endzeit-Auswahl. Ist die Start-Tageszeit größer oder gleich der End-Tageszeit, so wird das Ende auf den nächsten Tag gelegt.

eSTARTEVENT_DURATION: Startereignis/Dauer-Auswahl.

eSTARTEVENT_ENDEVENT: Startereignis/Endereignis-Auswahl.

stStartEnd: Struktur mit den Parametern, welche Start und Ende definieren. Nicht benötigte Variablen bleiben intern unberücksichtigt, wie beispielsweise die Dauer bei Startzeit/Endzeit-Auswahl.

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

todStartTime: Startzeit.

bStartEvent: Startereignis

tDuration: Schaltdauer.

todEndTime: Endzeit.

bEndEvent: Endereignis.

stSystemtime: aktuelle Uhrzeit im TIMESTRUCT-Format. Wichtig ist, dass jede Sekunde mitgezählt wird.

VAR_OUTPUT

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

bOut: Steuerausgang, welcher durch die Start- und Endereignisse ein- oder ausgeschaltet wird.

bTriggerOn: Triggerausgang für die Einschaltereignisse. Dieser Ausgang dient dazu, Einschaltereignisse zu erfassen. Sollten 2 Einschaltereignisse aufeinanderfolgen würde man dieses über den Steuerausgang *bOut* nicht bemerken, da dieser auf TRUE stehen bleiben würde, siehe auch zeitliche Überschneidungen [▶ 65] in der Übersicht.

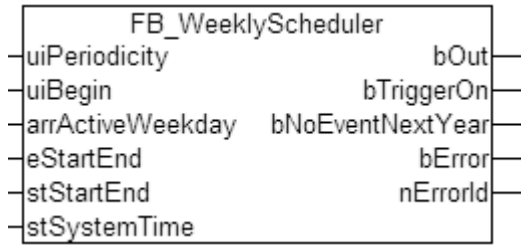
bNoEventNextYear: Hinweis: Innerhalb der nächsten 366 Tage kann kein Tag gefunden werden, auf den die Parametrierung zutrifft.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn die Parametrierung fehlerhaft ist. Der befehlsspezifische Fehlercode ist in *nErrorId* enthalten. Wird nach korrekter Parametrierung auf FALSE zurückgesetzt.

nErrorId: Enthält den befehlsspezifischen Fehlercode. Wird nach korrekter Parametrierung auf auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 90].

3.7.1.2 FB_WeeklyScheduler

Funktionsbaustein zum Schalten von Aktionen an bestimmten Wochentagen in jeder n-ten Woche im Jahr.



Hinweis: Der Funktionsbaustein benötigt zum Schalten das Durchschreiten des Zeitschaltpunktes. Ein nachträgliches Ändern der Schaltereignisse bzw. der Zeit ist daher nicht zulässig.

VAR_INPUT

```
uiPeriodicity      : UINT;
uiBegin            : UINT;
arrActiveWeekday   : ARRAY[0..6] OF BOOL;
eStartEnd          : ENUM;
stStartEnd         : TIMESTRUCT;
stSystemtime       : TIMESTRUCT;
```

uiPeriodicity: Periodizität bzw. Intervall. Darf im Bereich von 1..52 liegen.

uiBegin: Startwert für die Woche. Darf im Bereich von 1..52 liegen. Beispiel: *uiPeriodicity* = 5, *uiBegin* = 2: Schaltereignisse in KW2, KW7, KW12 etc. - *uiPeriodicity* = 3, *uiBegin* = 1: Schaltereignisse in KW1, KW4, KW7 etc.

arrActiveWeekday: Wochentag an dem geschaltet werden soll - *arrActiveWeekday[0]* => Sonntag .. *arrActiveWeekday[6]* => Samstag. Mehrfachauswahl ist möglich.

eStartEnd: Auswahl der 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: Startzeit/Endzeit-Auswahl. Ist die Start-Tageszeit größer oder gleich der End-Tageszeit, so wird das Ende auf den nächsten Tag gelegt.

eSTARTTIME_DURATION: Startzeit/Dauer-Auswahl.

eSTARTTIME_ENDEVENT: Startzeit/Endereignis-Auswahl.

eSTARTEVENT_ENDTIME: Startereignis/Endzeit-Auswahl. Ist die Start-Tageszeit größer oder gleich der End-Tageszeit, so wird das Ende auf den nächsten Tag gelegt.

eSTARTEVENT_DURATION: Startereignis/Dauer-Auswahl.

eSTARTEVENT_ENDEVENT: Startereignis/Endereignis-Auswahl.

stStartEnd: Struktur mit den Parametern, welche Start und Ende definieren. Nicht benötigte Variablen bleiben intern unberücksichtigt, wie beispielsweise die Dauer bei Startzeit/Endzeit-Auswahl.

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

todStartTime: Startzeit.

bStartEvent: Startereignis

tDuration: Schaltdauer.

todEndTime: Endzeit.

bEndEvent: Endereignis.

stSystemtime: aktuelle Uhrzeit im TIMESTRUCT-Format. Wichtig ist, dass jede Sekunde mitgezählt wird.

VAR_OUTPUT

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

bOut: Steuerausgang, welcher durch die Start- und Endereignisse ein- oder ausgeschaltet wird.

bTriggerOn: Triggerausgang für die Einschaltereignisse. Dieser Ausgang dient dazu, Einschaltereignisse zu erfassen. Sollten 2 Einschaltereignisse aufeinanderfolgen würde man dieses über den Steuerausgang *bOut* nicht bemerken, da dieser auf TRUE stehen bleiben würde, siehe auch zeitliche Überschneidungen [▶ 65] in der Übersicht.

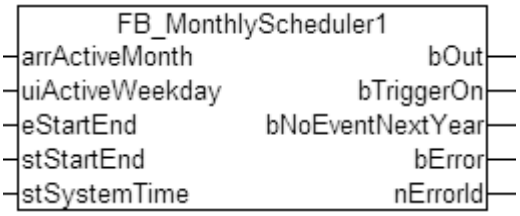
bNoEventNextYear: Hinweis: Innerhalb der nächsten 366 Tage kann kein Tag gefunden werden, auf den die Parametrierung zutrifft.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn die Parametrierung fehlerhaft ist. Der befehlsspezifische Fehlercode ist in *nErrorId* enthalten. Wird nach korrekter Parametrierung auf FALSE zurückgesetzt.

nErrorId: Enthält den befehlsspezifischen Fehlercode. Wird nach korrekter Parametrierung auf auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 90].

3.7.1.3 FB_MonthlyScheduler1

Funktionsbaustein zum Schalten von Aktionen an einem bestimmten Wochentag in bestimmten Monaten.



Hinweis: Der Funktionsbaustein benötigt zum Schalten das Durchschreiten des Zeitschaltpunktes. Ein nachträgliches Ändern der Schaltereignisse bzw. der Zeit ist daher nicht zulässig.

VAR_INPUT

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

arrActiveMonth: Monat in dem geschaltet werden soll - arrActiveMonth[1]=>Januar .. arrActiveMonth[12]=>Dezember. Mehrfachauswahl ist möglich.

uiActiveWeekday: Wochentag, an dem in den gewählten Monaten geschaltet werden soll. 0=Sonntag .. 6=Samstag. Mehrfachauswahl ist nicht möglich; der Wert darf 6 nicht überschreiten

eStartEnd: Auswahl der 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: Startzeit/Endzeit-Auswahl. Ist die Start-Tageszeit größer oder gleich der End-Tageszeit, so wird das Ende auf den nächsten Tag gelegt.

eSTARTTIME_DURATION: Startzeit/Dauer-Auswahl.

eSTARTTIME_ENDEVENT: Startzeit/Endereignis-Auswahl.

eSTARTEVENT_ENDTIME: Startereignis/Endzeit-Auswahl. Ist die Start-Tageszeit größer oder gleich der End-Tageszeit, so wird das Ende auf den nächsten Tag gelegt.

eSTARTEVENT_DURATION: Startereignis/Dauer-Auswahl.

eSTARTEVENT_ENDEVENT: Startereignis/Endereignis-Auswahl.

stStartEnd: Struktur mit den Parametern, welche Start und Ende definieren. Nicht benötigte Variablen bleiben intern unberücksichtigt, wie beispielsweise die Dauer bei Startzeit/Endzeit-Auswahl.

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

todStartTime: Startzeit.

bStartEvent: Startereignis

tDuration: Schaltdauer.

todEndTime: Endzeit.

bEndEvent: Endereignis.

stSystemtime: aktuelle Uhrzeit im TIMESTRUCT-Format. Wichtig ist, dass jede Sekunde mitgezählt wird.

VAR_OUTPUT

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

bOut: Steuerausgang, welcher durch die Start- und Endereignisse ein- oder ausgeschaltet wird.

bTriggerOn: Triggerausgang für die Einschaltereignisse. Dieser Ausgang dient dazu, Einschaltereignisse zu erfassen. Sollten 2 Einschaltereignisse aufeinanderfolgen würde man dieses über den Steuerausgang *bOut* nicht bemerken, da dieser auf TRUE stehen bleiben würde, siehe auch zeitliche Überschneidungen [▶ 65] in der Übersicht.

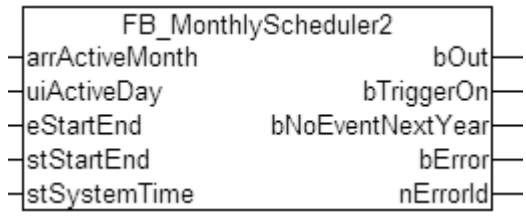
bNoEventNextYear: Hinweis: Innerhalb der nächsten 366 Tage kann kein Tag gefunden werden, auf den die Parametrierung zutrifft.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn die Parametrierung fehlerhaft ist. Der befehlsspezifische Fehlercode ist in *nErrorId* enthalten. Wird nach korrekter Parametrierung auf FALSE zurückgesetzt.

nErrorId: Enthält den befehlsspezifischen Fehlercode. Wird nach korrekter Parametrierung auf auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 90].

3.7.1.4 FB_MonthlyScheduler2

Funktionsbaustein zum Schalten von Aktionen an einem bestimmten Tag in bestimmten Monaten.



Hinweis: Der Funktionsbaustein benötigt zum Schalten das Durchschreiten des Zeitschaltpunktes. Ein nachträgliches Ändern der Schaltereignisse bzw. der Zeit ist daher nicht zulässig.

VAR_INPUT

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

arrActiveMonth: Monat in dem geschaltet werden soll - arrActiveMonth[1]=>Januar .. arrActiveMonth[12]=>Dezember. Mehrfachauswahl ist möglich.

uiActiveDay: Monatstag, an dem geschaltet werden soll. Mehrfachauswahl ist nicht möglich.

eStartEnd: Auswahl der 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: Startzeit/Endzeit-Auswahl. Ist die Start-Tageszeit größer oder gleich der End-Tageszeit, so wird das Ende auf den nächsten Tag gelegt.

eSTARTTIME_DURATION: Startzeit/Dauer-Auswahl.

eSTARTTIME_ENDEVENT: Startzeit/Endereignis-Auswahl.

eSTARTEVENT_ENDTIME: Startereignis/Endzeit-Auswahl. Ist die Start-Tageszeit größer oder gleich der End-Tageszeit, so wird das Ende auf den nächsten Tag gelegt.

eSTARTEVENT_DURATION: Startereignis/Dauer-Auswahl.

eSTARTEVENT_ENDEVENT: Startereignis/Endereignis-Auswahl.

stStartEnd: Struktur mit den Parametern, welche Start und Ende definieren. Nicht benötigte Variablen bleiben intern unberücksichtigt, wie beispielsweise die Dauer bei Startzeit/Endzeit-Auswahl.

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

todStartTime: Startzeit.

bStartEvent: Startereignis

tDuration: Schaltdauer.

todEndTime: Endzeit.

bEndEvent: Endereignis.

stSystemtime: aktuelle Uhrzeit im TIMESTRUCT-Format. Wichtig ist, dass jede Sekunde mitgezählt wird.

VAR_OUTPUT

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

bOut: Steuerausgang, welcher durch die Start- und Endereignisse ein- oder ausgeschaltet wird.

bTriggerOn: Triggerausgang für die Einschaltereignisse. Dieser Ausgang dient dazu, Einschaltereignisse zu erfassen. Sollten 2 Einschaltereignisse aufeinanderfolgen würde man dieses über den Steuerausgang *bOut* nicht bemerken, da dieser auf TRUE stehen bleiben würde, siehe auch zeitliche Überschneidungen [▶ 65] in der Übersicht.

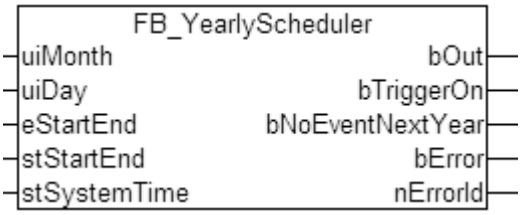
bNoEventNextYear: Hinweis: Innerhalb der nächsten 366 Tage kann kein Tag gefunden werden, auf den die Parametrierung zutrifft.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn die Parametrierung fehlerhaft ist. Der befehlsspezifische Fehlercode ist in *nErrorId* enthalten. Wird nach korrekter Parametrierung auf FALSE zurückgesetzt.

nErrorId: Enthält den befehlsspezifischen Fehlercode. Wird nach korrekter Parametrierung auf auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 90].

3.7.1.5 FB_YearlyScheduler

Funktionsbaustein zum Schalten von Aktionen an einem bestimmten Tag im Jahr.



Hinweis: Der Funktionsbaustein benötigt zum Schalten das Durchschreiten des Zeitschaltpunktes. Ein nachträgliches Ändern der Schaltereignisse bzw. der Zeit ist daher nicht zulässig.

VAR_INPUT

```
uiMonth      : UINT;
uiDay        : UINT;
eStartEnd    : ENUM;
stStartEnd   : TIMESTRUCT;
stSystemtime : TIMESTRUCT;
```

uiMonth: Monat, in dem geschaltet werden soll. Mehrfachauswahl ist nicht möglich.

uiDay: Tag, an dem geschaltet werden soll. Mehrfachauswahl ist nicht möglich.

eStartEnd: Auswahl der 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: Startzeit/Endzeit-Auswahl. Ist die Start-Tageszeit größer oder gleich der End-Tageszeit, so wird das Ende auf den nächsten Tag gelegt.

eSTARTTIME_DURATION: Startzeit/Dauer-Auswahl.

eSTARTTIME_ENDEVENT: Startzeit/Endereignis-Auswahl.

eSTARTEVENT_ENDTIME: Startereignis/Endzeit-Auswahl. Ist die Start-Tageszeit größer oder gleich der End-Tageszeit, so wird das Ende auf den nächsten Tag gelegt.

eSTARTEVENT_DURATION: Startereignis/Dauer-Auswahl.

eSTARTEVENT_ENDEVENT: Startereignis/Endereignis-Auswahl.

stStartEnd: Struktur mit den Parametern, welche Start und Ende definieren. Nicht benötigte Variablen bleiben intern unberücksichtigt, wie beispielsweise die Dauer bei Startzeit/Endzeit-Auswahl.

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

todStartTime: Startzeit.

bStartEvent: Startereignis

tDuration: Schaltdauer.

todEndTime: Endzeit.

bEndEvent: Endereignis.

stSystemtime: aktuelle Uhrzeit im TIMESTRUCT-Format. Wichtig ist, dass jede Sekunde mitgezählt wird.

VAR_OUTPUT

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

bOut: Steuerausgang, welcher durch die Start- und Endereignisse ein- oder ausgeschaltet wird.

bTriggerOn: Triggerausgang für die Einschaltereignisse. Dieser Ausgang dient dazu, Einschaltereignisse zu erfassen. Sollten 2 Einschaltereignisse aufeinanderfolgen würde man dieses über den Steuerausgang *bOut* nicht bemerken, da dieser auf TRUE stehen bleiben würde, siehe auch zeitliche Überschneidungen [▶ 65] in der Übersicht.

bNoEventNextYear: Hinweis: Innerhalb der nächsten 366 Tage kann kein Tag gefunden werden, auf den die Parametrierung zutrifft.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn die Parametrierung fehlerhaft ist. Der befehlsspezifische Fehlercode ist in *nErrorId* enthalten. Wird nach korrekter Parametrierung auf FALSE zurückgesetzt.

nErrorId: Enthält den befehlsspezifischen Fehlercode. Wird nach korrekter Parametrierung auf auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 90].

3.7.1.6 Schaltuhr Programmbeispiel

Das folgende Programmbeispiel soll anhand einer Tages-Schaltuhr verdeutlichen, wie die Bausteine insbesondere in Hinblick auf die Eingänge *eStartEnd*, *stStartEnd* und *stSystemTime* zu parametrieren sind.

Um die Systemzeit auf PC bzw. CX basierenden Systemen auszulesen empfiehlt sich die Verwendung des Bausteines NT_GetTime(), welcher in der Bibliothek *TcUtilities.lib* vorhanden ist. Ein Programm zum Auslesen könnte folgendermaßen aussehen:

```
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
```

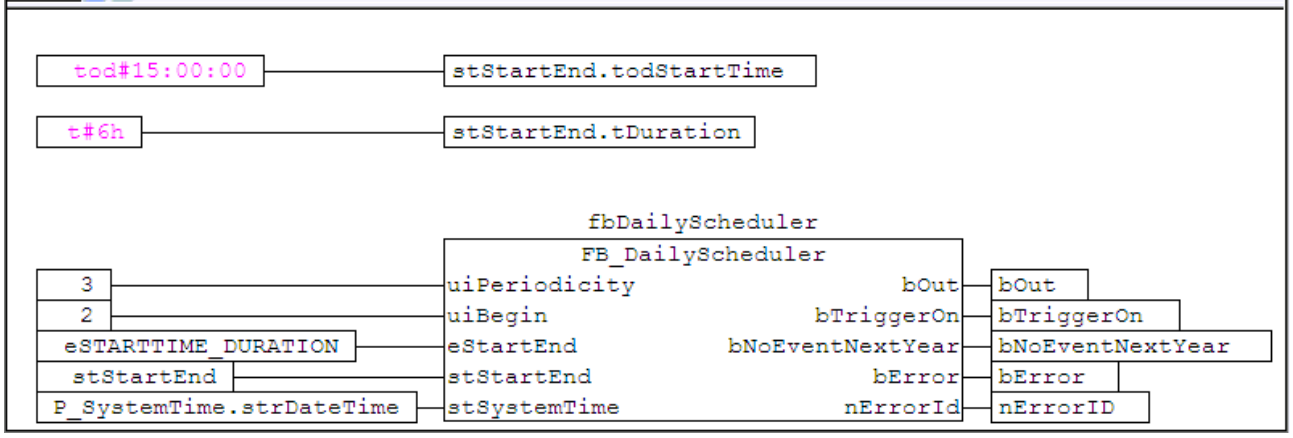
```
0001 (* Read The Time *)
0002 CASE nStep OF
0003 0:
0004     tonGetTime(IN := TRUE, PT := t#500ms);
0005     IF (tonGetTime.Q) THEN
0006         tonGetTime(IN := FALSE);
0007         nStep := 10;
0008     END_IF
0009 10:
0010     fbGetTime(  NETID := '',
0011                 START := TRUE,
0012                 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
```

Damit hat man eine Zeitbasis, mit der sich die Scheduler-Bausteine in Bezug auf den Zeiteingang *stSystemTime* parametrieren lassen. Am Eingang *eStartEnd* wird der Enumerator angelegt, der dem gewünschten Verhalten entspricht:

eSTARTTIME_ENDTIME	Startkriterium: Uhrzeit - Endkriterium: Uhrzeit
eSTARTTIME_DURATION	Startkriterium: Uhrzeit - Endkriterium: Dauer
eSTARTTIME_ENDEVENT	Startkriterium: Uhrzeit - Endkriterium: Ereignis (Boolscher Eingang)
eSTARTEVENT_ENDTIME	Startkriterium: Ereignis (Boolescher Eingang) - Endkriterium: Uhrzeit
eSTARTEVENT_DURATION	Startkriterium: Ereignis (Boolescher Eingang) - Endkriterium: Dauer
eSTARTEVENT_ENDEVENT	Startkriterium: Ereignis (Boolescher Eingang) - Endkriterium: Ereignis (Boolescher Eingang)

Für den Eingang *stStartEnd* muss eine Strukturvariable gleichen Typs deklariert werden, welche hier im Beispiel ebenfalls *stStartEnd* genannt wird. Im Programm werden dann die Untervariablen dieser Struktur beschrieben, die für die Funktionsart wichtig sind. Für das gezeigte Beispiel sind das *todStartTime* und *tDuration*. Alle anderen Variablen werden nicht gelesen und müssen daher auch nicht beschrieben werden.

```
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
```

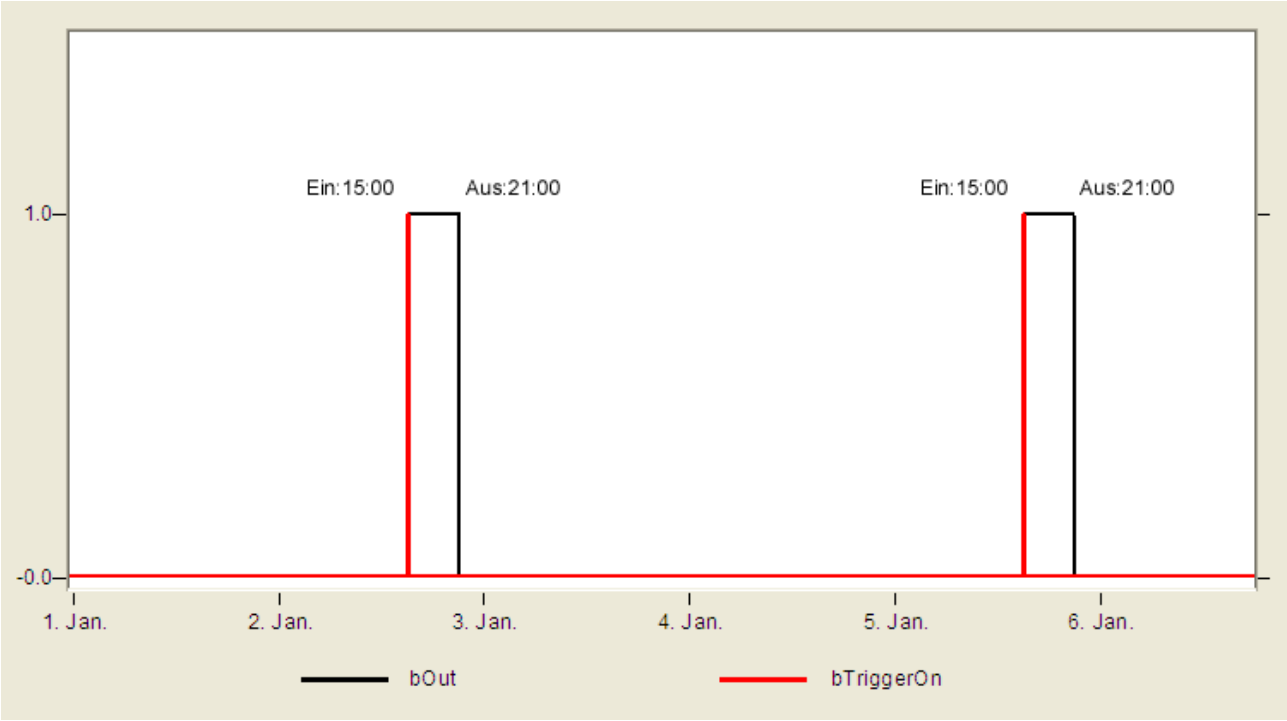


Beide Programme müssen im MAIN-Baustein aufgerufen werden, wobei der Programmteil *P_SchedulerExample* erst dann durchlaufen werden darf, wenn der Programmteil P_SystemTime gültige Daten liefert, also *P_SystemTimeValid* auf TRUE steht. Diese Verriegelung rührt daher, dass das Lesen der Uhrzeit einige Zyklen dauert und somit bei Programmstart eine ungültige Zeit vorliegt, mit der nicht gearbeitet werden darf.

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

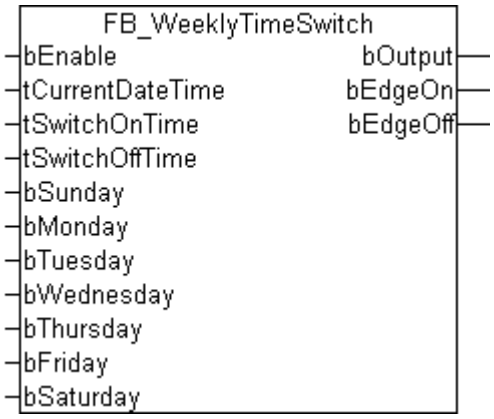
```
0001 P_SystemTime;
0002 IF NOT P_SystemTime.bSystemTimeValid THEN
0003 RETURN;
0004 END_IF
0005 P_SchedulerExample;
0006
```

Wird das Programm am 1.Januar gestartet, so ergibt sich folgender Verlauf:



Die Tage an denen geschaltet wird beginnen mit dem 2. des Jahres (*uiBegin:=2*). Das setzt sich dann alle 3 Tage fort (*uiPeriodicity:=3*). Der Einschaltzeitpunkt liegt bei 15 Uhr (*stStartEnd.todStartTime := tod#15:00:00*) und die Schaltdauer beträgt 6 Stunden (*stStartEnd.tDuration := t#6h*).

3.7.2 FB_WeeklyTimeSwitch



Die beiden Parameter *tSwitchOnTime* und *tSwitchOffTime* definieren einen Zeitraum, in dem der Ausgang *bOutput* aktiv gesetzt werden soll. Soll die Zeitschaltuhr nur an bestimmten Wochentagen gültig sein, so kann dieses durch die Eingänge *bSunday*, *bMonday*, ..., *bSaturday* festgelegt werden. Wollen sie mehrere Zeitkanäle schalten, so können sie dieses dadurch realisieren, in dem sie mehrere Instanzen des Funktionsbausteins anlegen. Jede Instanz ist für einen Zeitkanal zuständig.

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: Bausteinfreigabe.

tCurrentDateTime: aktuelles Datum und Uhrzeit.

tSwitchOnTime: Uhrzeit für das Einschalten.

tSwitchOffTime: Uhrzeit für das Ausschalten.

bSunday: Zeitschaltuhr Sonntags berücksichtigen.

bMonday: Zeitschaltuhr Montags berücksichtigen.

bTuesday: Zeitschaltuhr Dienstags berücksichtigen.

bWednesday: Zeitschaltuhr Mittwochs berücksichtigen.

bThursday: Zeitschaltuhr Donnerstags berücksichtigen.

bFriday: Zeitschaltuhr Freitags berücksichtigen.

bSaturday: Zeitschaltuhr Samstags berücksichtigen.

VAR_OUTPUT

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

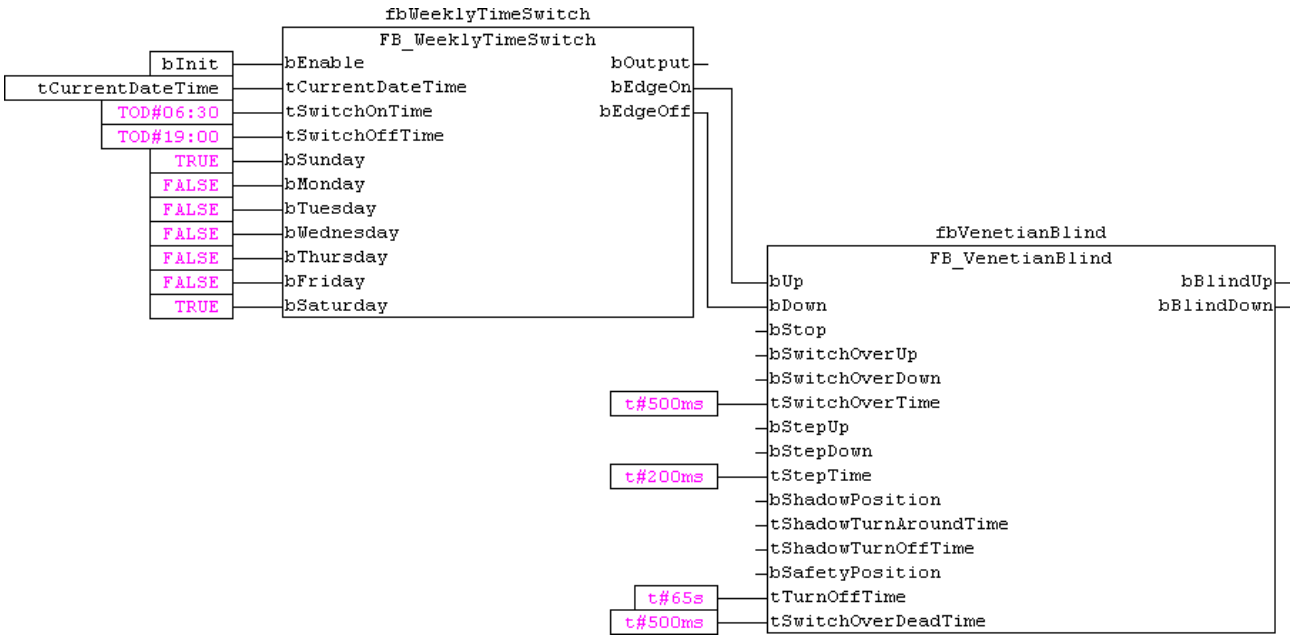
bOutput: Ausgang geht auf TRUE, wenn die aktuelle Uhrzeit zwischen der Einschaltzeit und der Ausschaltzeit liegt.

bEdgeOn: für einen SPS-Zyklus wird der Ausgang auf TRUE gesetzt, wenn die Zeitkanal aktiv wird.

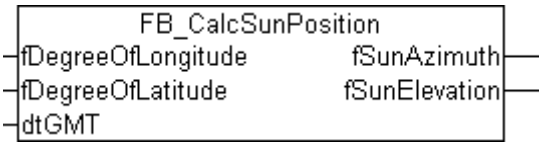
bEdgeOff: für einen SPS-Zyklus wird der Ausgang auf TRUE gesetzt, wenn die Zeitkanal deaktiv wird.

Beispiel

Bei dem folgenden Beispiel soll am Wochenende morgens um 6.30Uhr die Jalousie hoch und um 19.00Uhr wieder runtergefahren werden. Die beiden Ausgänge *bEdgeOn* und *bEdgeOff* der Zeitschaltuhr werden auf die Eingänge *bUp* und *bDown* des Jalousiebausteins gelegt. Durch die Impulse der Ausgänge wird dann die Jalousie zu den angegebenen Uhrzeiten hoch- bzw. runtergefahren.



3.7.3 FB_CalcSunPosition

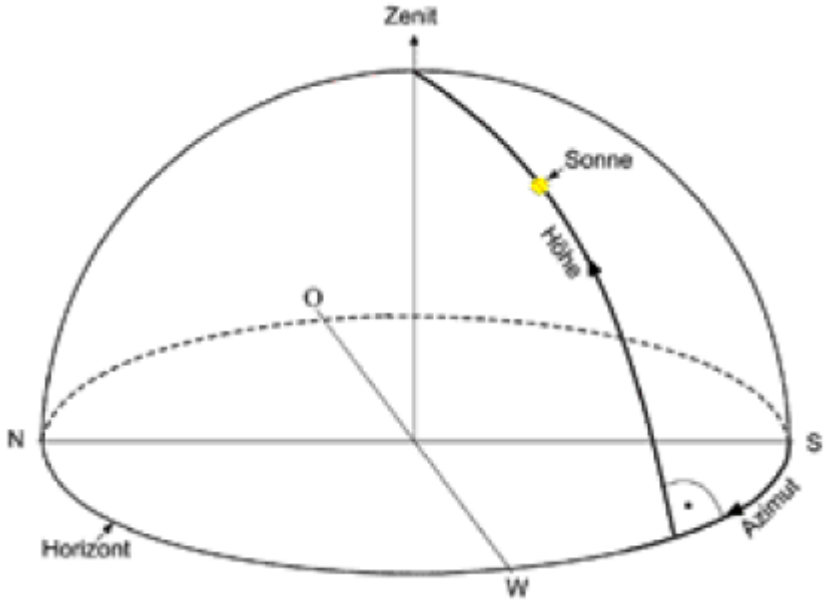


Berechnung des Sonnenstandes durch die Angabe von Datum, Uhrzeit, geographischer Länge und geographischer Breite.

Beschreibung

Der Sonnenstand für einen gegebenen Zeitpunkt lässt sich nach gängigen Methoden mit definierter Genauigkeit berechnen. Für Anwendungen mit mäßigen Anforderungen genügt der hier vorliegende Baustein. Als Grundlage wurde hierfür der SUNAE-Algorithmus verwendet, der einen günstigen Kompromiss zwischen Genauigkeit und Rechenaufwand darstellt.

Der Stand der Sonne an einem festen Beobachtungsort wird normalerweise mit zwei Winkelangaben bestimmt. Die eine gibt die Höhe über dem Horizont an, wobei 0° bedeutet, dass sich die Sonne in der Horizontalebene des Standortes befindet und ein Wert von 90°, dass sie sich senkrecht über dem Beobachter befindet. Die andere Winkelangabe gibt die Richtung an, in der die Sonne steht. Bei dem SUNAE-Algorithmus wird unterschieden, ob der Beobachter auf der nördlichen (Längengrad > 0) oder auf der südlichen (Längengrad < 0) Erdhalbkugel steht. Ist der Beobachtungspunkt auf der nördlichen Erdhalbkugel, so wird ein Wert von 0° für die nördliche Sonnenrichtung zugeordnet und läuft dann im Uhrzeigersinn um den Kompass, d.h. 90° ist Osten, 180° ist Süden, 270° Westen etc. Ist der Beobachtungspunkt auf der südlichen Erdhalbkugel, so entspricht 0° der südlichen Richtung und verläuft dann entgegen dem Uhrzeigersinn, d.h. 90° ist Osten, 180° ist Norden, 270° Westen etc.



Bei der Angabe der Uhrzeit muss die Zeit nach Greenwich Mean Time (GMT) angegeben werden.

Die geographische Breite ist die im Winkelmaß (also in Grad) angegebene nördliche oder südliche Entfernung eines Ortes der Erdoberfläche vom Äquator. Die Breite kann Werte von 0° (am Äquator) bis ±90° (an den Polen) annehmen. Dabei gibt ein positives Vorzeichen die nördliche Richtung und ein negatives Vorzeichen die südliche Richtung an. Die geographische Länge ist ein Winkel, der ausgehend vom Nullmeridian 0° (künstlich festgelegte Nord-Süd-Linie) Werte bis ±180° annehmen kann. Ein positives Vorzeichen gibt die Länge in östlicher Richtung und ein negatives Vorzeichen in westlicher Richtung an. Beispiele:

Ort	geographische Länge	geographische Breite
Syndey, Australien	151,2°	-33,9°
New York, USA	-74,0°	40,7°

Ort	geographische Länge	geographische Breite
London, England	-0,1°	51,5°
Moskau, Russland	37,6°	55,7°
Peking, China	116,3°	39,9°
Dubai, Vereinigte Arabische Emirate	55,3°	25,4°
Rio de Janeiro, Brasilien	-43,2°	-22,9°
Hawai, USA	-155,8°	20,2°
Verl, Deutschland	8,5°	51,9°

Gibt der Baustein *FB_CalcSunPosition()* für die Sonnenhöhe (*fSunElevation*) einen negativen Wert zurück, so ist die Sonne nicht sichtbar. Dieses kann zur Bestimmung von Sonnenaufgang und Sonnenuntergang genutzt werden.

VAR_INPUT

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

fDegreeOfLongitude: geographische Länge (Längengrad) in Grad.

fDegreeofLatitude: geographische Breite (Breitengrad) in Grad.

dtGMT: aktuelle Uhrzeit als Greenwich Mean Time (GMT).

VAR_OUTPUT

```
fSunAzimuth       : LREAL;
fSunElevation     : LREAL;
```

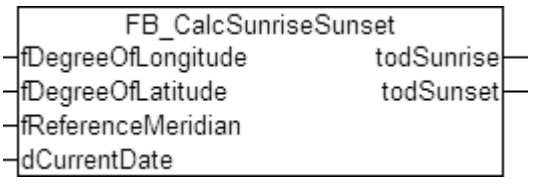
fSunAzimuth: Sonnenrichtung (nördliche Erdhalbkugel: 0° Norden ... 90° Osten ... 180° Süden ... 270° Westen ... / südliche Erdhalbkugel: 0° Süden ... 90° Osten ... 180° Norden ... 270° Westen ...).

fSunElevation: Sonnenhöhe (0° horizontal ... 90° senkrecht).

Beispiel

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

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

fbCalcSunPosition( fDegreeOfLongitude := 8.5,
           fDegreeOfLatitude := 51.9,
           dtGMT := FILETIME_TO_SYSTEMTIME(fileTime));
fSunAzimuth := fbCalcSunPosition.fSunAzimuth;
fSunElevation := fbCalcSunPosition.fSunElevation;
```

3.7.4 FB_CalcSunriseSunset



Funktionsbaustein zur Berechnung des Sonnenaufgangs und des Sonnenuntergangs durch die Angabe der geographischen Länge, der geographischen Breite, des Bezugsmeridian und der Uhrzeit.

Die Erde ist in mehrere Zeitzonen unterteilt. Jeder Zeitzone ist ein Bezugsmeridian zugeordnet. Bezugsmeridian einiger Zeitzonen:

Zeitzone	Bezugsmeridian
GMT (Greenwich Mean Time)	$\lambda_{GMT} = 0°$
MEZ (Mitteleuropäische Zeit)	$\lambda_{MEZ} = 15°$
MESZ (Mitteleuropäische Sommerzeit)	$\lambda_{MESZ} = 30°$

Bei der Angabe der Uhrzeit muss die Zeit nach Greenwich Mean Time (GMT) angegeben werden.

Hinweis: Dieser Baustein ist nur in der PC-Version der Library verfügbar.

VAR_INPUT

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

fDegreeOfLongitude: geographische Länge (Längengrad) in Grad.

fDegreeofLatitude: geographische Breite (Breitengrad) in Grad.

fReferenceMeridian: Bezugsmeridian von der Zeitzone.

dCurrentDate: aktuelles Datum.

VAR_OUTPUT

```
todSunrise        : TOD;
todSunset         : TOD;
```

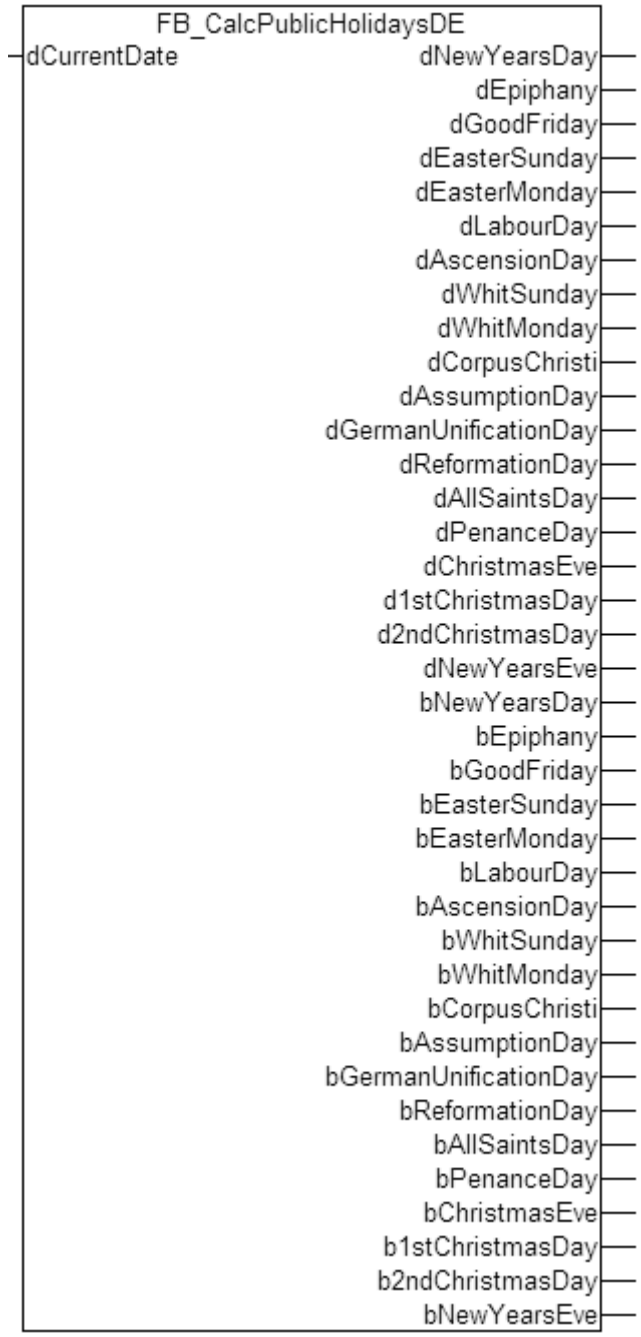
todSunrise: Sonnenaufgang. Ausgabe der Stunde und der Minute.

todSunset: Sonnenuntergang. Ausgabe der Stunde und der Minute.

Beispiel

```
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



Berechnung der deutschen Feiertage.

Beschreibung

Anhand der Eingabe des Datums werden die Feiertage für das gegebene Jahr errechnet. Des Weiteren erfolgt eine boolesche Ausgabe, ob das eingegebene Datum einem der errechneten Feiertage entspricht. Der Baustein wurde zur internationalen Lesbarkeit in die englische Sprache übersetzt. Es bedeuten:

englische Bezeichnung	deutsche Bezeichnung
NewYears Day	Neujahr
Epiphany	Heilige Drei Könige
Good Friday	Karfreitag
Easter Sunday	Ostersonntag
Easter Monday	Ostermontag

englische Bezeichnung	deutsche Bezeichnung
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: aktuelles Datum.

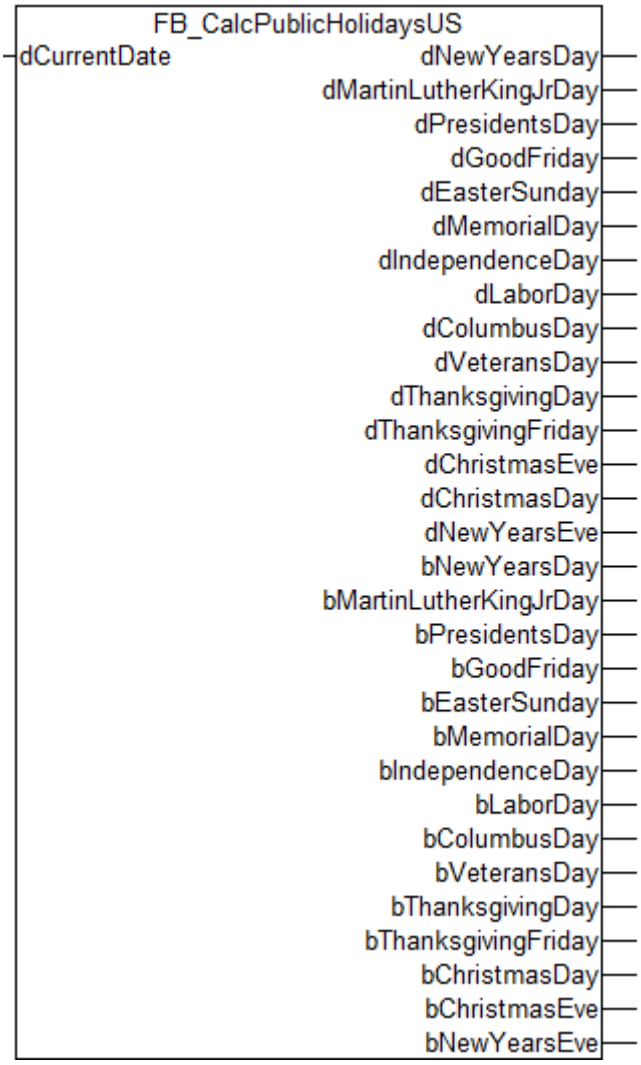
VAR_OUTPUT

```
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: Datumsausgabe des jeweiligen Feiertages.

bxxxxxx: Boolesche Aussage, ob der heutige Tag der jeweilige Feiertag ist.

3.7.6 FB_CalcPublicHolidaysUS



Berechnung der nordamerikanischen Feiertage.

Beschreibung

Anhand der Eingabe des Datums werden die Feiertage für das gegebene Jahr errechnet. Des Weiteren erfolgt eine boolesche Ausgabe, ob das eingegebene Datum einem der errechneten Feiertage entspricht. Der Baustein wurde zur internationalen Lesbarkeit in die Englische Sprache übersetzt. Es bedeuten:

englische Bezeichnung	deutsche Bezeichnung
NewYears Day	Neujahr
Epiphany	Heilige Drei Könige
Good Friday	Karfreitag
Easter Sunday	Ostersonntag
Easter Monday	Ostermontag
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

englische Bezeichnung	deutsche Bezeichnung
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: aktuelles Datum.

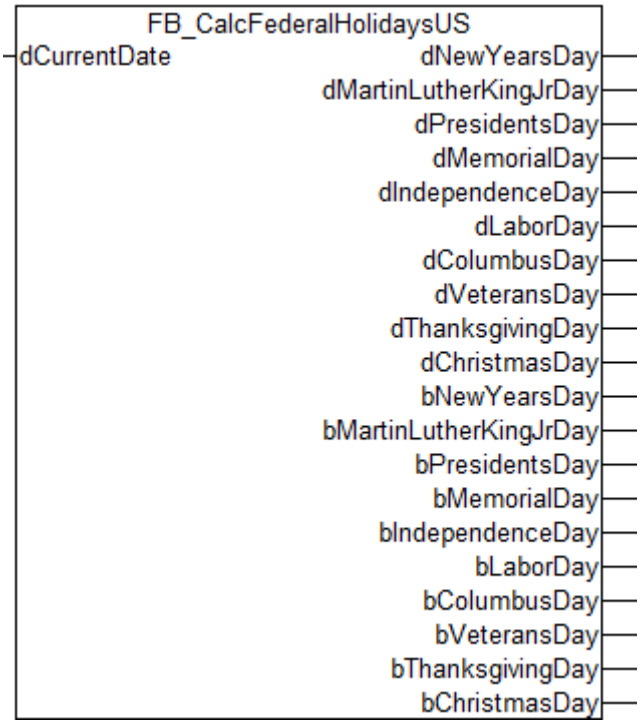
VAR_OUTPUT

```
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: Datumsausgabe des jeweiligen Feiertages.

bxxxxxx: Boolesche Aussage, ob der heutige Tag der jeweilige Feiertag ist.

3.7.7 FB_CalcFederalHolidaysUS



Berechnung der nordamerikanischen Bundes Feiertage.

Beschreibung

Anhand der Eingabe des Datums werden die Feiertage für das gegebene Jahr errechnet. Des Weiteren erfolgt eine boolesche Ausgabe, ob das eingegebene Datum einem der errechneten Feiertage entspricht. Der Baustein wurde zur internationalen Lesbarkeit in die englische Sprache übersetzt. Es bedeuten:

englische Bezeichnung	deutsche Bezeichnung
NewYears Day	Neujahr
Epiphany	Heilige Drei Könige
Good Friday	Karfreitag
Easter Sunday	Ostersonntag
Easter Monday	Ostermontag
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: aktuelles Datum.

VAR_OUTPUT

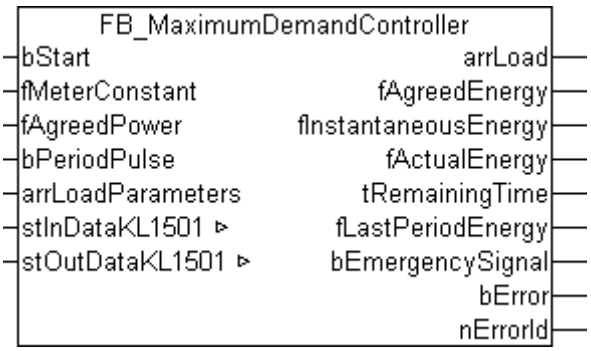
```
dNewYearsDay          : DATE;
dEpiphany             : DATE;
dGoodFriday           : DATE;
dEasterSunday         : DATE;
dEasterMonday         : DATE;
dLabourDay            : DATE;
dAscensionDay         : DATE;
dWhitSunday           : DATE;
dWhitMonday           : DATE;
dCorpusChristi        : DATE;
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: Datumsausgabe des jeweiligen Feiertages.

bxxxxxx: Boolesche Aussage, ob der heutige Tag der jeweilige Feiertag ist.

3.8 Energiemanagement

3.8.1 FB_MaximumDemandController



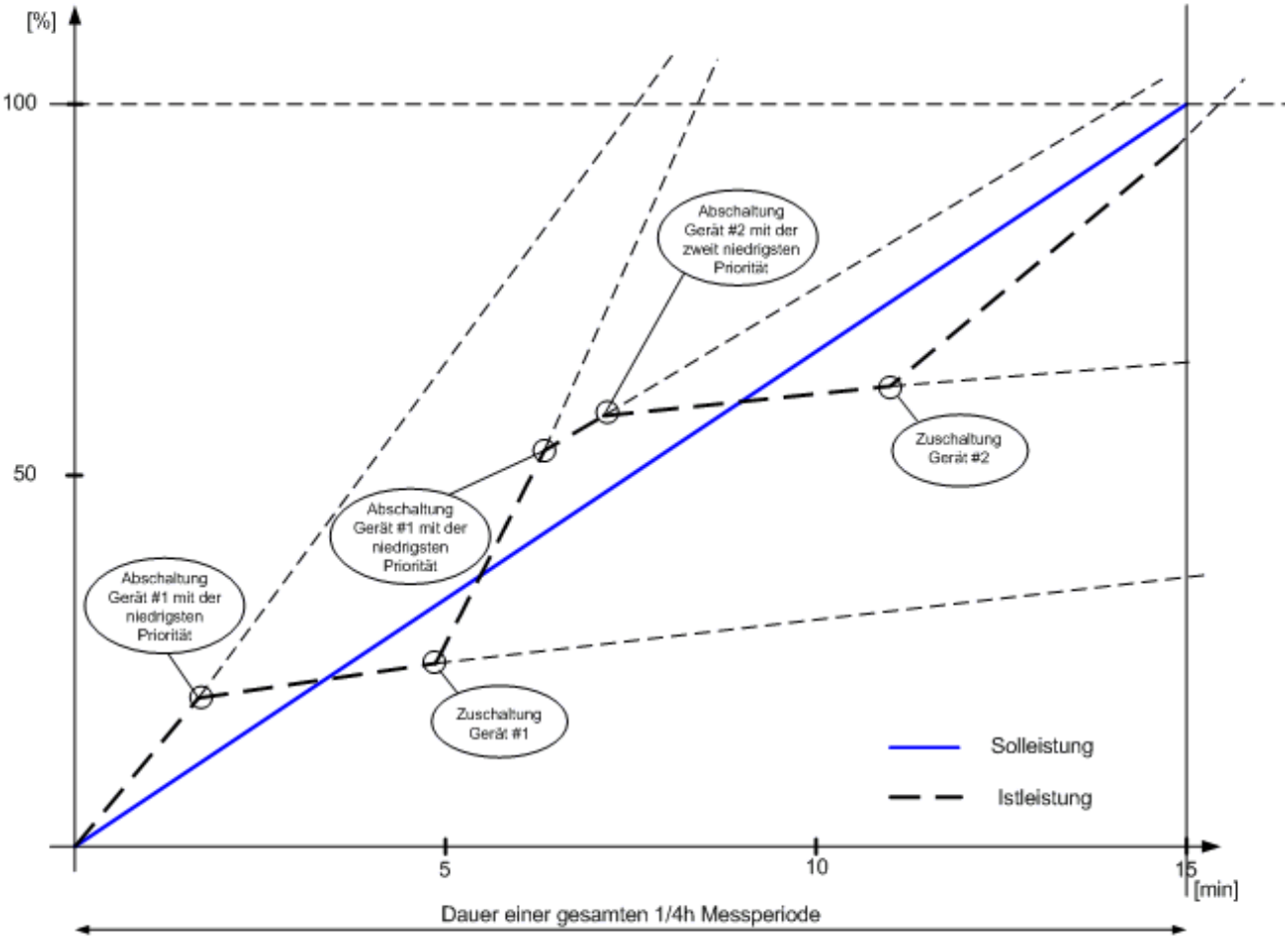
Funktionsbaustein zur Spitzenlastoptimierung, der durch Ab-/ Zuschaltung von bis zu acht Verbrauchern die Einhaltung der eingestellten Leistungsgrenze realisiert. Die Verbraucher können entsprechend ihrer Leistung und Priorität so abgeschaltet werden, dass keine Störungen des Produktionsablaufs entstehen.

Zur Erkennung der einzelnen Messzyklen wird von dem Energieversorgungsunternehmen (EVU) ein Synchronimpuls geliefert. Dieser gibt den Beginn eines neuen Messzyklus an und muss auf den Eingang *bPeriodPuls* aufgeschaltet werden. Die Erfassung der Ist-Leistung erfolgt über die Zählerklemme KL1501.

Der Baustein arbeitet mit einer festen Messperiodenzeit von 15 Minuten. Überschreitet der Synchronimpuls die 16-Minuten-Grenze, so wird der Ausgang *bEmergencySignal* gesetzt.

Zu Begin jeder Messperiode werden alle Verbraucher zugeschaltet. Innerhalb der Messperiode erfolgt bei drohender Überschreitung der Leistungsgrenze (fAgreedPower) nacheinander eine Abschaltung der Verbraucher. Ist die Gefahr der Lastüberschreitung nicht mehr gegeben, werden die Verbraucher wieder eingeschaltet.

Über eine Eingangsvariable können Besonderheiten, wie minimale Einschaltzeit, minimale Ausschaltzeit oder maximale Ausschaltzeit festgelegt werden. Ebenfalls kann die Priorität der einzelnen Verbraucher bestimmt werden. Verbraucher mit einer niedrigen Priorität werden vor Verbrauchern mit einer hohen Priorität abgeschaltet.



VAR_INPUT

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

bStart: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

fMeterConstant: Zählerkonstante [Impulse / kWh].

fAgreedPower: Dies ist die vereinbarte Leistungsgrenze, die im Betriebsfall nach Möglichkeit nicht überschritten werden darf [kW].

bPeriodPulse: Synchronimpuls, der vom Energieversorgungsunternehmen (EVU) gesendet wird. Mit diesem Impuls wird das Messintervall gestartet.

arrLoadParameter: Parameterstruktur des jeweiligen Verbrauchers. Diese besteht aus den folgenden Elementen:

```
TYPE ST_MDCLoadParameters: STRUCT bConnected : BOOL; nDegreeOfPriority : INT;
tMINPowerOnTime : TIME; tMINPowerOffTime : TIME; tMAXPowerOffTime : TIME; END_STRUCT
END_TYPE
```

bConnected: TRUE = Verbraucher angeschlossen; FALSE = Verbraucher nicht angeschlossen.

nDegreeOfPriority: Kennzeichnet die Abschaltpriorität, Verbraucher mit der niedrigen Priorität werden als erstes abgeschaltet. (1 => niedrige; ... 8 => hohe Priorität)

tMINPowerOnTime: Minimale Einschaltzeit (Mindest-Hochlaufzeit) in der, der Verbraucher nicht abgeschaltet werden darf.

tMINPowerOffTime: Minimale Ausschaltzeit (Erholzeit) in der, der Verbraucher nicht erneut eingeschaltet werden darf.

tMAXPowerOffTime: Maximale Ausschaltzeit nach der, der Verbraucher erneut eingeschaltet werden muss.

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: Ist ein Array des Datentyps BOOL; eingeschaltete Verbraucher sind TRUE.

fAgreedEnergy: Vereinbarter Energieverbrauch [kWh].

fInstantaneousEnergy: momentaner Energieverbrauch [kWh] bezogen auf einen Integrationszeitraum von 15s (interner Messintervall).

fActualEnergy: Zum "jetzigen" betrachteten Zeitpunkt der Messperiode verbrauchte Energie.

tRemainingTime: Restzeit bis zum nächsten Messintervall.

fLastPeriodEnergy: Sollleistung aus der vorhergegangenen Messperiode [kWh].

bEmergencySignal: Der Ausgang wird gesetzt so bald die vorgegebene Energie überschritten wird.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist.

nErrorId: Enthält den Fehlercode [▶ 90].

VAR_IN_OUT

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

stInDataKL1501: Verknüpft mit der KL1501.

```
TYPE ST_MDCInDataKL1501 :
STRUCT
  nStatus    : USINT;
  nDummy1    : USINT;
  nDummy2    : USINT;
  nDummy3    : USINT;
```

```
  nData      : DWORD;
END_STRUCT
END_TYPE
```

stOutDataKL1501: Verknüpft mit der KL1501.

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

Beispiel

```
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 Fehlercodes

Wert (hex)	Wert (dez)	Beschreibung
0x0000	0	kein Fehler.
0x0001	1	FB_MaximumDemandController() [▶ 86]: -- reservierter Fehlercode --
0x0002	2	FB_MaximumDemandController() [▶ 86]: Der Eingabeparameter *fMeterConstant* ist "0".
0x0003	3	FB_LightControl() [▶ 22] *:* Die Switchrange *(nSwitchRange)* im ersten oder zweiten Element der Wertetabelle *arrControlTable* ist 0. Damit wird davon ausgegangen, daß die Tabelle keinen oder nur einen Wertesatz hat.
0x0004	4	FB_LightControl() [▶ 22] *:* 2 benachbarte Eingangswerte *nActualValue* in der Wertetabelle *arrControlTable* liegen zu dicht beieinander d.h. im Schaltbereich des anderen.
0x0005	5	FB_Sequencer() [▶ 29] *:* Der Startindex *nStartIndex* ist außerhalb des gültigen Bereiches [1..50].
0x0006	6	FB_Sequencer() [▶ 29] *:* Der Startindex *nStartIndex* verweist auf eine Stelle, die ihrerseits ein Sequenzende (Nulleinträge) markiert.
0x0007	7	Scheduler-Bausteine: Ein Eingangsparameter ist nicht im gültigen Bereich.
0x0008	8	Scheduler-Bausteine: Bei den Auswahlparametern ist keiner gesetzt (Wochen-Zeituhr: Auswahl der Wochentage, Monats-Zeituhren: Auswahl der Monate).
0x0009	9	Scheduler-Bausteine: Es wurde ein Tag im Monat angewählt, der nicht gültig ist.
0x000A	10	FB_ConstantLightControlEco() [▶ 24] *:* Eingangsparameter *nMinLevel* ist größer oder gleich *nMaxLevel.*
0x000B	11	FB_ScenesLighting() [▶ 49], FB_ScenesVenetianBlind() [▶ 52]: Eingangsparameter *sFile* ist ungültig (leer).
0x000C	12	FB_ScenesLighting() [▶ 49], FB_ScenesVenetianBlind() [▶ 52]: Interner Fehler. Datei mit den Szenenwerten wurde nicht gefunden.
0x000D	13	FB_ScenesLighting() [▶ 49], FB_ScenesVenetianBlind() [▶ 52]: Interner Fehler: Keine weiteren freien File Handles vorhanden.

4 Anhang

4.1 Support und Service

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

Beckhoff Niederlassungen und Vertretungen

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

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

Dort finden Sie auch weitere Dokumentationen zu Beckhoff Komponenten.

Beckhoff Support

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

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

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

Beckhoff Service

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

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

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

Beckhoff Firmenzentrale

Beckhoff Automation GmbH & Co. KG

Hülshorstweg 20
33415 Verl
Deutschland

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

Mehr Informationen:
www.beckhoff.de/ts8010

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