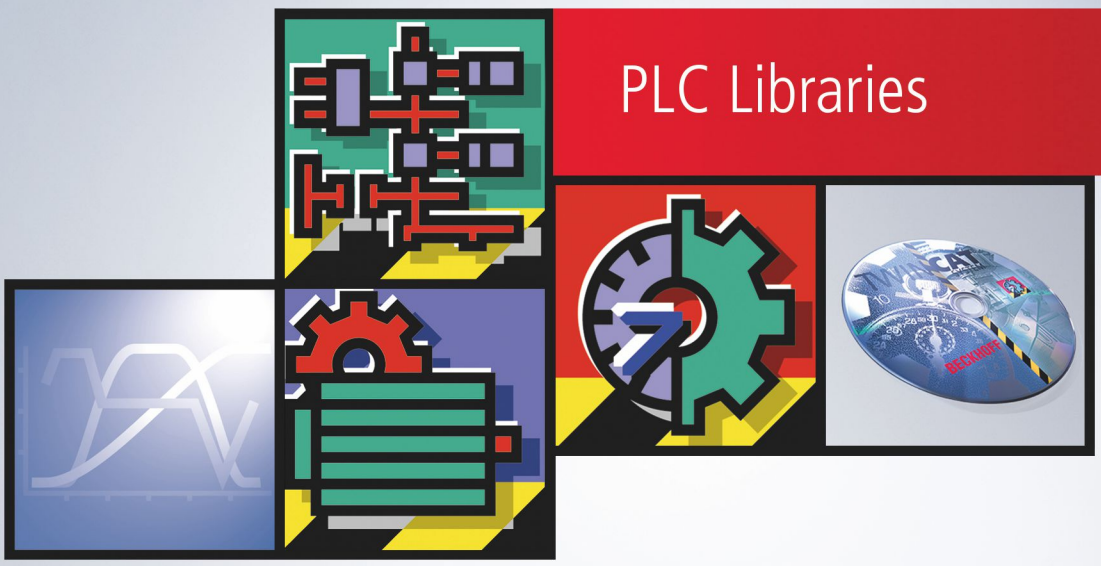
BECKHOFF New Automation Technology

Handbuch | DE

TX1200

TwinCAT 2 | PLC-Bibliothek: TcSMI



15.11.2023 | Version: 1.1

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, stets die aktuell gültige 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 weiterentwickelt.
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® ist eine eingetragene Marke und patentierte Technologie lizenziert durch die Beckhoff Automation GmbH, Deutschland

Copyright

1.2 Zu Ihrer Sicherheit

Sicherheitsbestimmungen

Lesen Sie die folgenden Erklärungen zu Ihrer Sicherheit.
Beachten und befolgen Sie stets produktspezifische Sicherheitshinweise, die Sie gegebenenfalls an den entsprechenden Stellen in diesem Dokument vorfinden.

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.

Signalwörter

Im Folgenden werden die Signalwörter eingeordnet, die in der Dokumentation verwendet werden. Um Personen- und Sachschäden zu vermeiden, lesen und befolgen Sie die Sicherheits- und Warnhinweise.

Warnungen vor Personenschäden

⚠ GEFAHR

Es besteht eine Gefährdung mit hohem Risikograd, die den Tod oder eine schwere Verletzung zur Folge hat.

⚠ WARNUNG

Es besteht eine Gefährdung mit mittlerem Risikograd, die den Tod oder eine schwere Verletzung zur Folge haben kann.

⚠ VORSICHT

Es besteht eine Gefährdung mit geringem Risikograd, die eine mittelschwere oder leichte Verletzung zur Folge haben kann.

Warnung vor Umwelt- oder Sachschäden

HINWEIS

Es besteht eine mögliche Schädigung für Umwelt, Geräte oder Daten.

Information zum Umgang mit dem Produkt

Diese Information beinhaltet z. B.:
Handlungsempfehlungen, Hilfestellungen oder weiterführende Informationen zum Produkt.

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 Zielgruppen

Für den Nutzer dieser Bibliothek werden folgende Grundkenntnisse vorausgesetzt:

- TwinCAT PLC Control
- TwinCAT System Manager
- PC und Netzwerkkenntnisse
- Aufbau und Eigenschaften der Beckhoff Embedded-PC und deren Busklemmensystem.
- Technologie von SMI Geräten
- Einschlägige Sicherheitsvorschriften der technischen Gebäudeausrüstung

Diese Softwarebibliothek ist für Gebäudeautomation-Systempartner der Beckhoff Automation GmbH & Co. KG. Die Systempartner sind tätig in dem Bereich Gebäudeautomation und beschäftigen sich mit Errichtung, Inbetriebsetzung, Erweiterung, Wartung und Service von mess-, steuer- und regelungstechnischen Anlagen der technischen Gebäudeausrüstung.

3 SMI

Mit dem SMI-Bussystem (Standard Motor Interface) wird in der Gebäudeautomatisierung die Rollladen- und Sonnenschutzautomation vereinfacht. Mit SMI-Antrieben können exakte Positionen bei Rollladen und gradgenaue Winkelpositionen bei Jalousieantrieben angefahren werden. Die SMI-Antriebe können Ist Positionen, Fehlermeldungen und Serviceinformationen an die SMI-Masterklemme zurücksenden.

Bedeutende europäische Hersteller haben sich zum SMI-Arbeitskreis zusammengeschlossen und das digitale Interface entwickelt. Über diese einheitliche Schnittstelle werden Antriebe mittels Telegrammen angesteuert. Mit Standard-Befehlen lassen sich Funktionen realisieren, die bei konventionellen Antrieben nicht so leicht möglich sind. Beispiele sind das präzise Anfahren von Positionen, die Rückmeldung der aktuellen Position so wie die Diagnose. Für die Verstellung von Lamellen der Verschattungsanlage können z. B. Winkelauflösungen von 2° erreicht werden. Damit ist eine sonnenstandsabhängige Nachführung der Lamellen zur Konstantlichtregelung möglich. In der TwinCAT-HVAC-Bibliothek stehen leistungsfähige SPS-Bausteine für die Raumautomation nach VDI 3813 zur Verfügung.

Es sind Distanzen bis 350 Meter zwischen Steuerung und Antrieb möglich. Zur Verkabelung kann ein normales 5-adriges Stromkabel verwendet werden (mit PE, N, L sowie dem SMI spezifischen I+ und I-), wobei I+ und I- verpolungssicher sind. Bis zu 16 Antriebe können parallelgeschaltet und einzeln adressiert werden. SMI-Antriebe gibt es für Netzspannung (230 V AC) und für Kleinspannung (24 V DC).

Um die Kompatibilität der SMI-Produkte untereinander zu gewährleisten, müssen alle Produkte, die mit dem SMI-Logo gekennzeichnet werden sollen, zertifiziert werden. Eine positive Zertifizierung lässt sich auf der SMI-Group-Homepage (https://standard-motor-interface.com) nachlesen. Dort finden Sie auch weitere Informationen über den SMI-Bus und SMI-Antriebe.

3.1 Geräte Adressierung

SMI definiert verschiedene Arten der Teilnehmeradressierung. Grundsätzlich kann unterschieden werden zwischen der Einzeladressierung, Gruppenadressierung und dem Sammelruf (Broadcast). Die meisten SPS-Bausteine besitzen hierfür den Eingang *dwAddr* und *eAddrType*. Während der Eingang *dwAddr* die notwendige Angabe der Adresse enthält, definiert *eAddrType* die Art der Adressierung. Die einzelnen Arten der Adressierung werden im Folgenden beschrieben. Beachten sie, dass nicht jeder Befehl alle Adressierungsarten unterstützt. Details hierzu finden sie in der Beschreibung des jeweiligen SPS-Bausteins.

per Adresse eines Teilnehmers (eAddrType := eSMIAddrTypeAddress)

Jedem SMI-Teilnehmer kann eine Adresse von 0 bis 15 zugewiesen werden. Die Adresse wird im SMI-Teilnehmer abgespeichert und muss bei einem Austausch des Antriebes wieder korrekt eingestellt werden. Da jede Adresse nur einmal zugewiesen werden sollte, lässt sich jeder SMI-Teilnehmer einzeln ansprechen. Der Eingang *dwAddr* enthält bei dieser Adressierungsart die Adresse im Bereich von 0 bis 15. Wird ein Wert außerhalb des gültigen Bereichs angegeben, so gibt der jeweilige Baustein einen Fehler [▶ 58] aus.

Gelegentlich wird diese Adresse auch Slave-Adresse genannt. Die Slave-Adresse darf nicht mit der Slave-Id verwechselt werden (siehe unten).

per Slave-Id (eAddrType := eSMIAddrTypeSlaveId)

Die einzelnen Gerätehersteller hinterlegen in jedem SMI-Gerät eine eindeutige 32-Bit große Nummer. Diese Slave-Id, auch Key-Id genannt, kann ebenfalls für die Adressierung eines Teilnehmers genutzt werden. Der Eingang *dwAddr* enthält bei dieser Adressierungsart die 32-Bit große Slave-Id und der Eingang *dwAddrOption* den Herstellercode (siehe unten). Hierdurch wird sichergestellt, dass SMI-Geräte weltweit eindeutig adressiert werden können ohne das diese eine Adresse benötigen.

Bei einigen SMI-Geräten ist der Slave-Id auf dem Typenschild aufgedruckt oder durch eine Beschriftung am Kabel sichtbar.

Die Adressierung per Slave-Id wird von den meisten Lese-Befehlen nicht unterstützt.

per Herstellercode (eAddrType := eSMIAddrTypeManufacturer)

SMI definiert für jeden Hersteller neben der Slave-Id eine weitere eindeutige Id, den sogenannten Herstellercode [▶ 67]. Der Herstellercode ist fest im SMI-Gerät hinterlegt und kann nicht verändert werden. Der mögliche Wertebereich ist von 0 bis 15, wobei der Wert 0 und 14 eine Sonderbedeutung haben. Der Herstellercode 0 adressiert alle Teilnehmer, unabhängig vom Hersteller. Somit kann diese Adressierungsart auch zum Versenden von Broadcast-Befehlen verwendet werden. Der Wert 15 ist reserviert für zukünftige Erweiterungen und darf nicht verwendet werden. Sehr häufig findet man hier die englische Bezeichnung *Manufacturer Code* oder auch die Abkürzung *M-ID*. Durch diese Adressierung werden immer alle Geräte eines Herstellers angesprochen. Der Eingang *dwAddr* enthält bei dieser Adressierungsart den Herstellercode im Bereich von 0 bis 14. Wird ein Wert außerhalb des gültigen Bereichs angegeben, so gibt der jeweilige Baustein einen Fehler [▶ 58] aus.

Bei einigen SMI-Geräten ist der Herstellercode auf dem Typenschild aufgedruckt oder durch eine Beschriftung am Kabel sichtbar.

per Gruppenadressierung (eAddrType := eSMIAddrTypeGroup)

Jeder Teilnehmer der über die Gruppenadressierung angesteuert werden soll, muß eine Adresse von 0 bis 15 besitzen. Jedes Bit des Eingangs *dwAddr* entspricht bei der Gruppenadressierung eine Adresse. Wird Bit 0 von *dwAddr* gesetzt, so wird der Teilnehmer mit der Adresse 0 angesteuert. Wird Bit 1 gesetzt, so wird Teilnehmer 1 angesprochen usw. Somit belegt die Gruppenadressierung die Bits 0 bis 15, was den Wertebereich von 0 bis 65535 entspricht. Wird ein Wert außerhalb des gültigen Bereichs angegeben, so gibt der jeweilige Baustein einen Fehler [▶ 58] aus.

Beispiel: Es sollen die Antriebe mit der Adresse 1, 4, 7 und 12 angesprochen werden. Somit muss an *dwAddr* der Wert 2#00010000_10010010 oder 16#1092 oder 4242 übergeben werden.

per Broadcast (eAddrType := eSMIAddrTypeBroadcast)

Bei der Adressierung per Broadcast werden immer alle Teilnehmer angesprochen, unabhängig der eingestellten Adresse am Gerät. Der Eingang *dwAddr* wird bei dieser Adressierung nicht benötigt und auch nicht ausgewertet. Intern verwenden die SPS-Bausteine die Adressierung per Herstellercode, wobei als Herstellercode 0 verwendet wird.

4 Integration in TwinCAT

4.1 Integration in TwinCAT (CX9020)

Dieses Beispiel beschreibt, wie ein einfaches SPS-Programm für SMI in TwinCAT geschrieben werden kann und wie es mit der Hardware verknüpft wird. Es soll ein SMI-Motor angesteuert und per Taster die Laufrichtung verändert werden.

https://infosys.beckhoff.com/content/1031/tcplclibsmi/Resources/12074354059.zip https://infosys.beckhoff.com/content/1031/tcplclibsmi/Resources/12074354059.zip

Hardware

Einrichtung der Komponenten

Es wird folgende Hardware benötigt:

- 1x Embedded-PC CX9020
- 1x Digitale 2-Kanal-Eingangsklemme KL1002 (für die Schritt hoch und runter Funktion)
- 1x SMI-Klemme KL6831
- 1x Endklemme KL9010

Richten Sie die Hardware sowie die SMI-Komponenten wie in den entsprechenden Dokumentationen beschrieben ein.

Dieses Beispiel geht davon aus, dass ein Hoch-Taster auf den ersten und ein Runter-Taster auf den zweiten Eingang der KL1002 gelegt wurde und sich an der Adresse 1 ein SMI-Motor befindet.

Software

Erstellung des SPS-Programms

Erstellen Sie ein neues SPS-Projekt für PC-basierte Systeme (ARM) und fügen die Bibliothek *TcSMI.lib* hinzu.

Erzeugen Sie als Nächstes die folgenden globalen Variablen:

```
VAR_GLOBAL
    bUp           AT %I* : BOOL;
    bDown         AT %I* : BOOL;
    stInData      AT %I* : ST_KL6831KL6841InData;
    stOutData     AT %Q* : ST_KL6831KL6841OutData;
    stCommandBuffer      : ST_SMICommandBuffer;
END_VAR
```

bUp: Eingangsvariable für den Hoch-Taster.

bDown: Eingangsvariable für den Runter-Taster.

stInData: Eingangsvariable [▶ 57] für die SMI-Klemme.

stOutData: Ausgangsvariable [▶ 57] für die SMI-Klemme.

stCommandBuffer: Wird für die Kommunikation [▶ 57] mit SMI benötigt.

Legen Sie anschließend ein Programm (CFC) für die Hintergrundkommunikation mit SMI an. In diesem wird der Baustein FB_KL6831KL6841Communication [▶ 19] aufgerufen. Achten Sie beim Kommunikationsbaustein darauf, mit **stInData**, **stOutData** und **stCommandBuffer** zu verknüpfen.

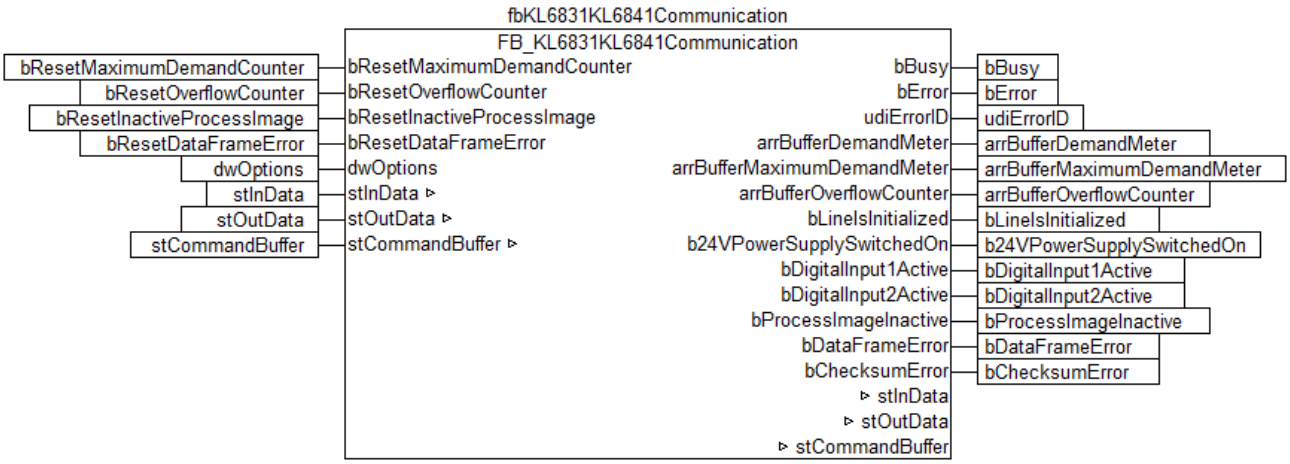
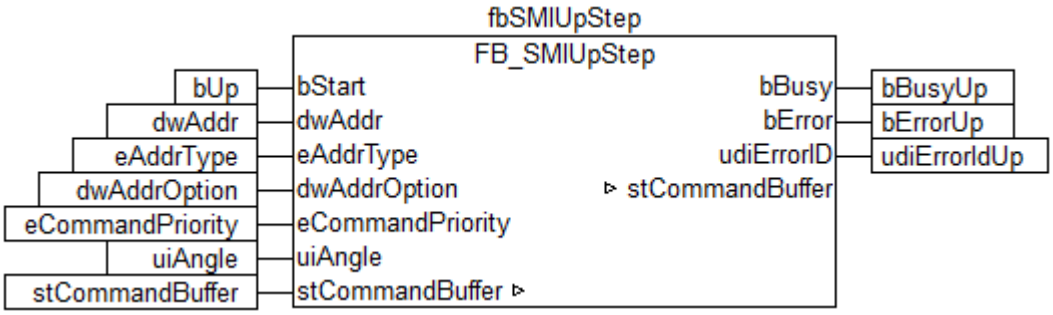
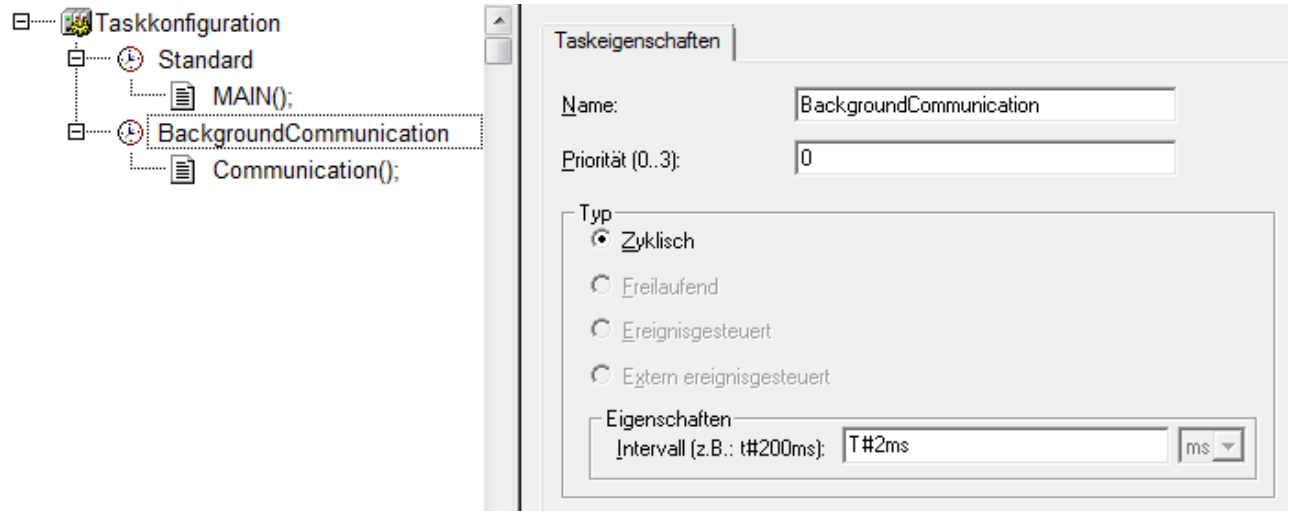


Abb. 1: Sample_FB_Com

Legen Sie ein MAIN-Programm (CFC) an in dem die Bausteine FB_SMIUpStep [▶ 42] und FB_SMIDownStep [▶ 29] aufgerufen werden. Der Eingang **bStart** des StepUp-Bausteins wird mit der globalen Variable **bUp** verknüpft und **stCommandBuffer** mit der globalen Variable **stCommandBuffer**. Verfahren Sie entsprechend für den StepDown-Baustein.



Gehen Sie in die Taskkonfiguration und geben Sie der Task eine niedrigere Intervall-Zeit. Genauere Informationen dazu finden Sie in der Beschreibung des Bausteins FB_KL6831KL6841Communication [▶ 19].

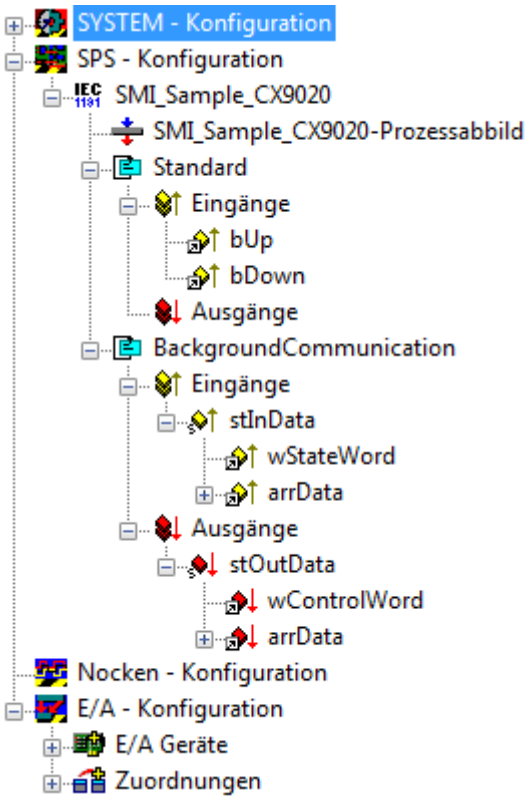


Laden Sie das Projekt als Bootprojekt auf den CX und speichern Sie es ab.

Konfiguration im System Manager

Legen Sie ein neues System-Manager-Projekt an, wählen Sie als Zielsystem den CX und lassen Sie nach dessen Hardware suchen.

Fügen Sie das oben angelegte SPS-Programm unter SPS-Konfiguration hinzu. Beim Aufklappen des SPS-Projekts in der Baumansicht werden die beiden Tasks aufgelistet. Beim Erweitern der Tasks ist sicherzustellen, dass alle globalen Ein- und Ausgangsvariablen der Standard-Task zugeordnet sind. Da die Variablen **stInData** und **stOutData** jedoch schneller abgearbeitet werden sollen, müssen diese mit Drag&Drop auf die Hintergrund-Kommunikations-Task verschoben werden.



Verknüpfen Sie die globalen Variablen des SPS-Programms nun mit den Ein- und Ausgängen der Busklemmen, erzeugen Sie die Zuordnungen und aktivieren Sie die Konfiguration. Starten Sie dann das Gerät im Run-Modus.
Ihr CX ist jetzt einsatzbereit.

Durch Betätigen von einem der Richtungstaster steuert der SMI-Motor die gewählte Richtung für den unter *uiAngle* angegebenen Winkelgrad an.

4.2 Integration in TwinCAT (BC9191)

Dieses Beispiel beschreibt, wie ein einfaches SPS-Programm für SMI in TwinCAT geschrieben werden kann und wie es mit der Hardware verknüpft wird. Es soll ein SMI-Motor angesteuert und per Taster die Laufrichtung verändert werden.

https://infosys.beckhoff.com/content/1031/tcplclibsmi/Resources/12074355467.zip https://infosys.beckhoff.com/content/1031/tcplclibsmi/Resources/12074355467.zip

Hardware

Einrichtung der Komponenten

Es wird folgende Hardware benötigt:

- 1x Busklemmen Controller BC9191
- 1x Potenzialeinspeiseklemme 24V DC
- 1x Digitale 2-Kanal-Eingangsklemme KL1002 (für die Schritt hoch und runter Funktion)
- 1x SMI-Klemme KL6831
- 1x Endklemme KL9010

Richten Sie die Hardware sowie die SMI-Komponenten wie in den entsprechenden Dokumentationen beschrieben ein.

Dieses Beispiel geht davon aus, dass ein Hoch-Taster auf den ersten und ein Runter-Taster auf den zweiten Eingang der KL1002 gelegt wurde und sich an der Adresse 1 ein SMI-Motor befindet.

Software

Erstellung des SPS-Programms

Erstellen Sie ein neues SPS-Projekt für BC-basierte Systeme (BCxx50 über AMS) und fügen die Bibliotheken *TcSMI.lbx* und *TcSystemBCxx50.lbx* hinzu. Gehen Sie danach im Menü auf *Projekt → Optionen... → Übersetzungsoptionen* und wählen *LREAL als REAL übersetzen* an.

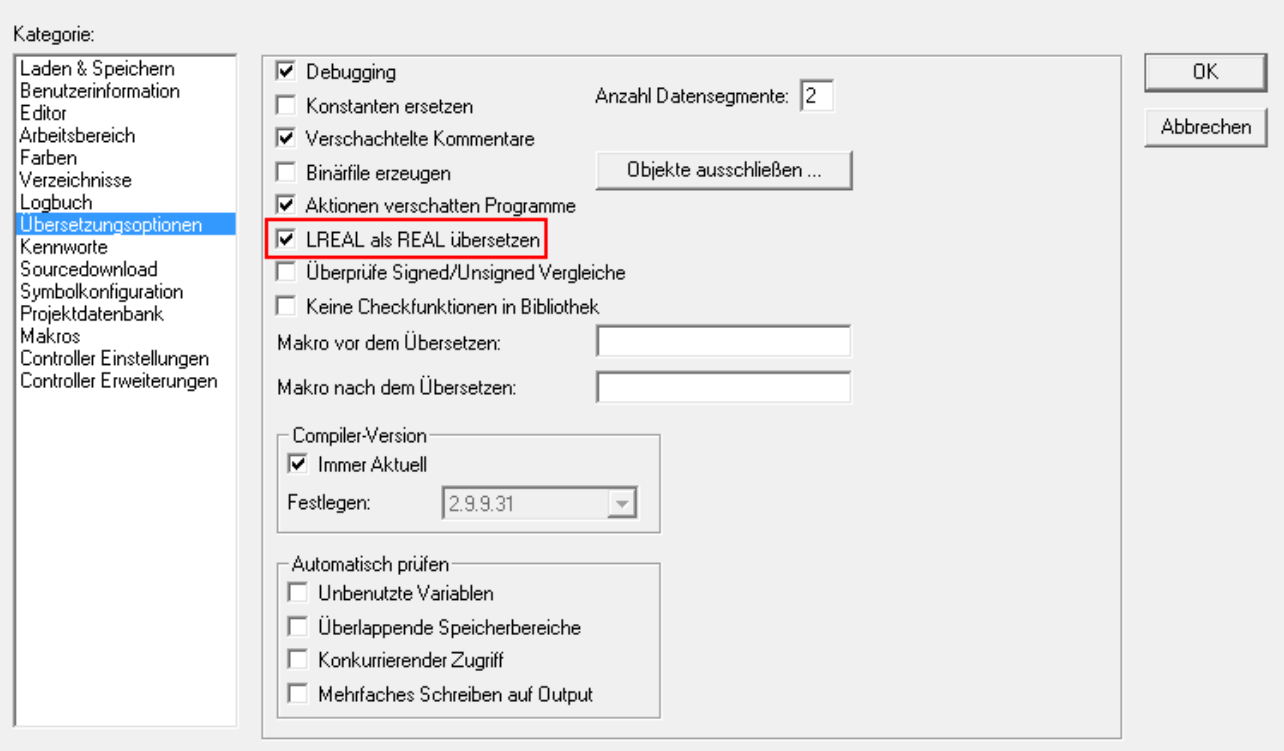


Abb. 2: Sample-BC-Options

Erzeugen Sie als Nächstes die folgenden globalen Variablen:

```
VAR_GLOBAL
    bUp           AT %I* : BOOL;
    bDown         AT %I* : BOOL;
    stInData      AT %I* : ST_KL6831KL6841InData;
    stOutData     AT %Q* : ST_KL6831KL6841OutData;
    stCommandBuffer      : ST_SMICommandBuffer;
END_VAR
```

bUp: Eingangsvariable für den Hoch-Taster.

bDown: Eingangsvariable für den Runter-Taster.

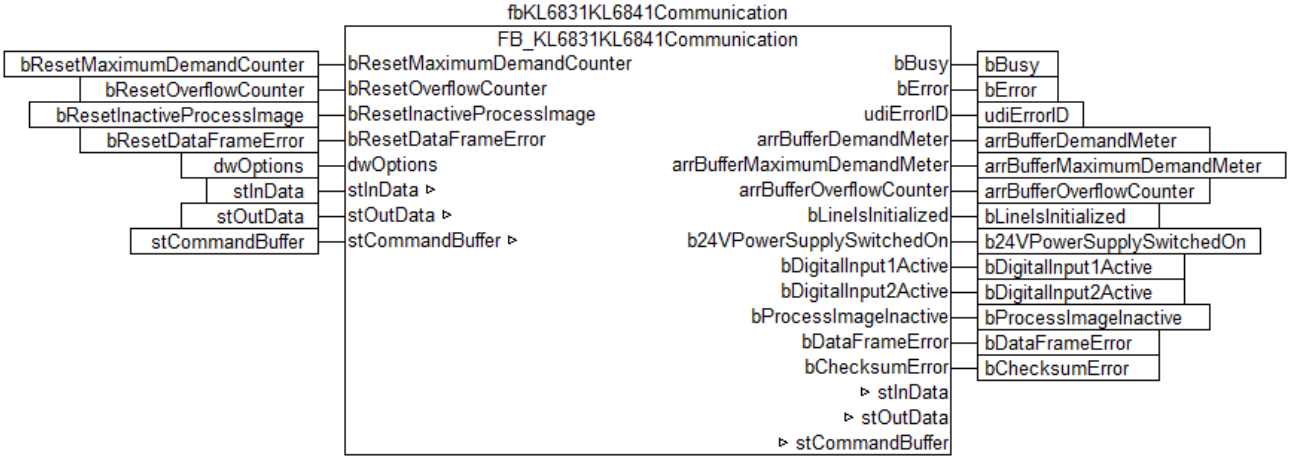
stInData: Eingangsvariable [▶ 57] für die SMI-Klemme.

stOutData: Ausgangsvariable [▶ 57] für die SMI-Klemme.

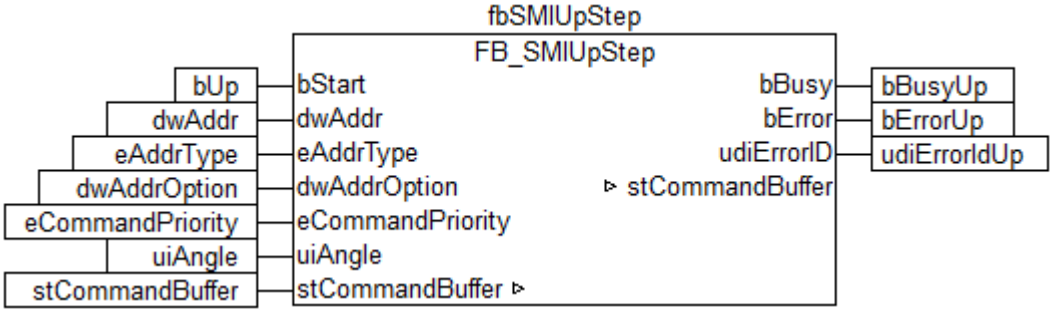
stCommandBuffer: Wird für die Kommunikation [▶ 57] mit SMI benötigt.

Da BC-Geräte nur über eine Task verfügen, kann die Kommunikation mit SMI nicht separat ausgeführt werden.

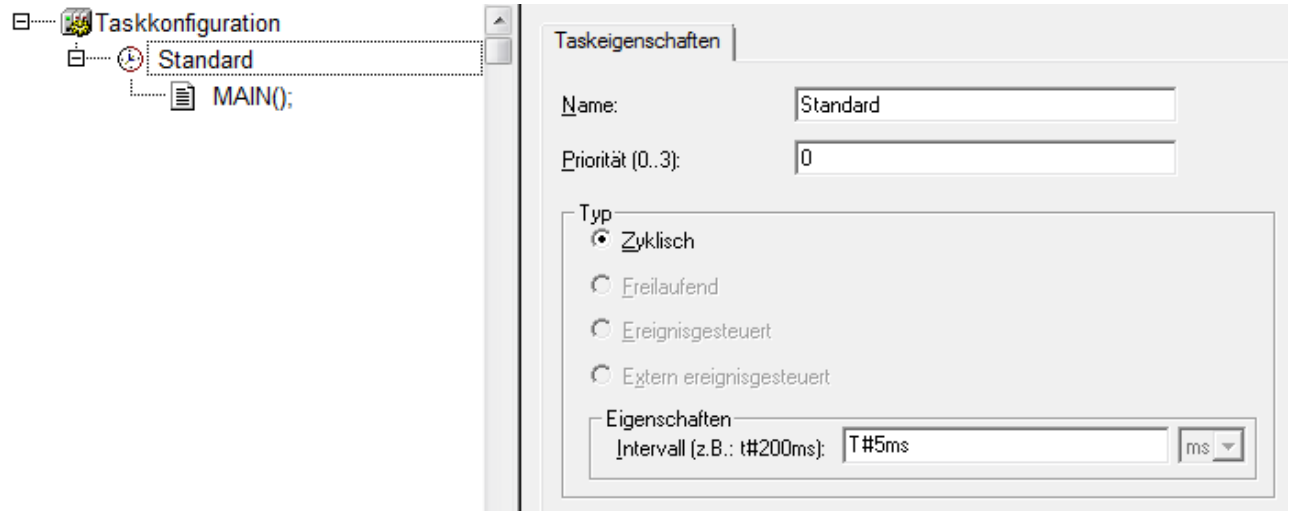
Legen Sie daher ein MAIN-Programm (CFC) an in dem die Bausteine FB_KL6831KL6841Communication [▶ 19], FB_SMIUpStep [▶ 42] und FB_SMIDownStep [▶ 29] aufgerufen werden. Achten Sie beim Kommunikationsbaustein darauf, mit **stInData**, **stOutData** und **stCommandBuffer** zu verknüpfen.



Der Eingang **bStart** des StepUp-Bausteins wird mit der globalen Variable **bUp** verknüpft und **stCommandBuffer** mit der globalen Variable **stCommandBuffer**. Verfahren Sie entsprechend für den StepDown-Baustein.



Gehen Sie in die Taskkonfiguration und geben Sie der Task eine niedrigere Intervall-Zeit. Genauere Informationen dazu finden Sie in der Beschreibung des Bausteins FB_KL6831KL6841Communication [▶ 19].

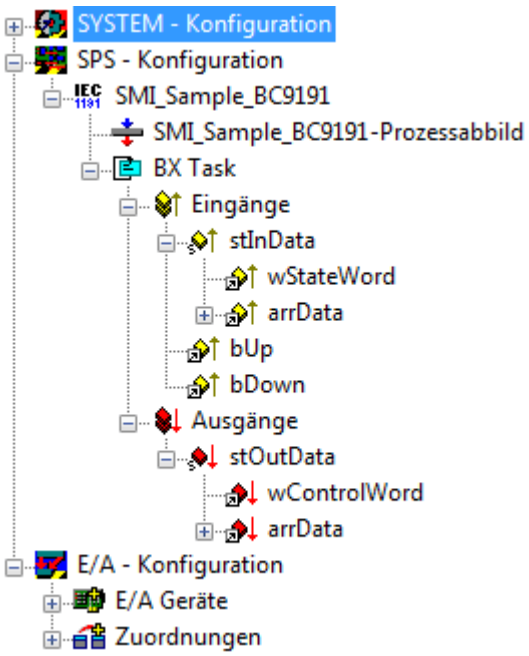


Laden Sie das Projekt als Bootprojekt auf den BC und speichern Sie es ab.

Konfiguration im System Manager

Legen Sie ein neues System-Manager-Projekt an, wählen Sie als Zielsystem den BC und lassen Sie nach dessen Hardware suchen.

Fügen Sie das oben angelegte SPS-Programm unter SPS-Konfiguration hinzu.



Verknüpfen Sie die globalen Variablen des SPS-Programms nun mit den Ein- und Ausgängen der Busklemmen, erzeugen Sie die Zuordnungen und aktivieren Sie die Konfiguration. Starten Sie dann das Gerät im Run-Modus.
Ihr BC ist jetzt einsatzbereit.

Durch Betätigen von einem der Richtungstaster steuert der SMI-Motor die gewählte Richtung für den unter *uiAngle* angegebenen Winkelgrad an.

5 Programmierung

Installation

Ab TwinCAT 2.11 Build 2240 (R3 und x64 Engineering) werden die Bibliotheken "TcSMI.lib/.lb6/.lbx" standardmäßig mitinstalliert.

Hardware Dokumentation im Beckhoff Information System: KL6831/KL6841 - SMI-Masterklemme

Weitere erforderliche Bibliotheken

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

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

Für Busklemmen-Controller der Serie BCxx00:

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

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

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

Für Busklemmen-Controller der Serie BXxx00:

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

Speicherauslastung

Durch das Einbinden der Bibliothek wird bereits SPS-Programmspeicher verbraucht. Abhängig vom Applikationsprogramm ist es daher möglich, dass der verbleibende Speicher nicht ausreichend ist.

5.1 Funktionsbausteine

Aktoren

Name	Beschreibung
FB_SMIVenetianBlind [▶ 65]	Jalousiesteuerung.

Basisbefehle

Name	Beschreibung
FB_KL6831KL6841Communication [▶ 19]	Liest sequentiell die SMI-Befehle aus den internen Puffern der SPS-Bibliothek aus und gibt diese zu der KL6831/KL6841
FB_KL6831KL6841Config [▶ 21]	Mit diesem Baustein kann die KL6831/KL6841 konfiguriert werden.
FB_SMISendSMICommand [▶ 24]	Dieser Baustein dient zum allgemeinen Senden eines SMI-Kommandos.

Grundbefehle

Name	Beschreibung
FB_SMIDiagAll [▶ 25]	Diagnosetelegramm wird versendet.
FB_SMIDown [▶ 27]	Motorlauf bis zur unteren Endlage.
FB_SMIDownStep [▶ 29]	Motorlauf nach unten um einen vorgegebenen Winkelgrad.
FB_SMIPos1 [▶ 30]	Fahrt zur motorseitig konfigurierten Fixposition *Pos1*.
FB_SMIPos1Read [▶ 31]	Lesen der motorseitig konfigurierten Fixposition *Pos1*.
FB_SMIPos1Write [▶ 32]	Schreiben der motorseitig konfigurierten Fixposition *Pos1*.
FB_SMIPos2 [▶ 33]	Fahrt zur motorseitig konfigurierten Fixposition *Pos2*.
FB_SMIPos2Read [▶ 34]	Lesen der motorseitig konfigurierten Fixposition *Pos2*.
FB_SMIPos2Write [▶ 35]	Schreiben der motorseitig konfigurierten Fixposition *Pos2*.
FB_SMIPosRead [▶ 36]	Aktuelle Position lesen.
FB_SMIPosWrite [▶ 37]	Anfahren einer Position.
FB_SMIStop [▶ 38]	Stopp des Motorlaufs.
FB_SMISyn [▶ 39]	Abfragen Herstellercode und Antriebstyp.
FB_SMIUp [▶ 41]	Motorlauf bis zur oberen Endlage.
FB_SMIUpStep [▶ 42]	Motorlauf nach oben um einen vorgegebenen Winkelgrad.

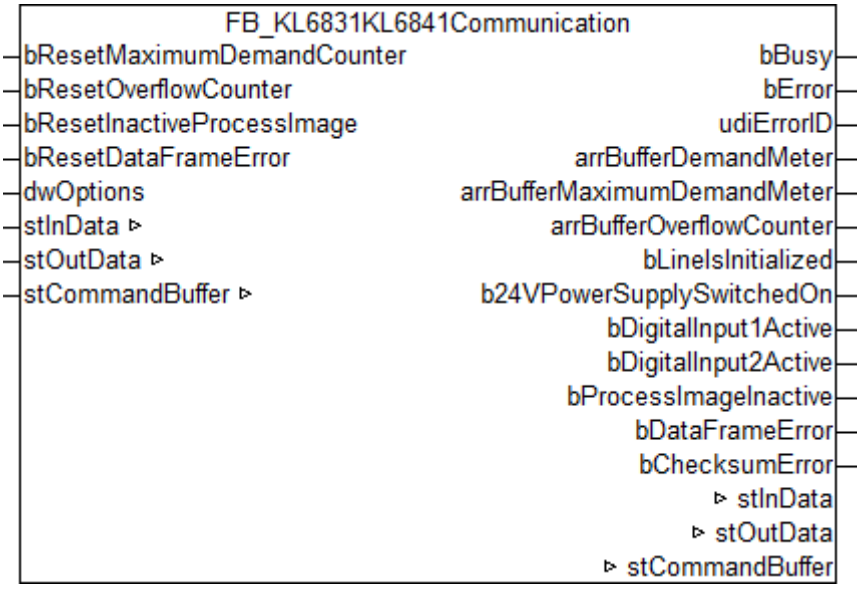
Adressierungsbefehle

Name	Beschreibung
FB_SMIAddressing [▶ 43]	SMI-Geräte adressieren.
FB_SMIDiscoverySlaveId [▶ 44]	SMI-Geräte nach Herstellercode suchen.
FB_SMISlaveAddrRead [▶ 45]	Adresse (0-15) eines Antriebes auslesen.
FB_SMISlaveAddrWrite [▶ 47]	Adresse (0-15) in einen oder mehrere Antriebe schreiben.
FB_SMISlaveIdCompare [▶ 48]	Vergleichen einer vorgegebenen Slave-Id (32 Bit Key-Id) mit der motorseitig definierten Slave-Id (32 Bit Key-Id) eines oder mehrerer Antriebe.
FB_SMISlaveIdRead [▶ 50]	Lesen der Slave-Id (32 Bit Key-Id).

Systembefehle

Name	Beschreibung
FB_SMIParValueReadByte [▶ 52]	Lesen eines motorseitig gespeicherten Byte-Parameters (1-Byte).
FB_SMIParValueReadWord [▶ 53]	Lesen eines motorseitig gespeicherten Word-Parameters (2-Bytes).
FB_SMIParValueReadDWord [▶ 54]	Lesen eines motorseitig gespeicherten DWord-Parameters (4-Bytes).

5.1.1 FB_KL6831KL6841Communication



Die Bausteine für die SMI-Befehle greifen nicht direkt auf das Prozessabbild der KL6831/KL6841 zu, sondern legen die einzelnen SMI-Befehle in drei verschiedene Puffer ab. Der Baustein FB_KL6831KL6841Communication() liest sequentiell die SMI-Befehle aus diesen drei Puffern aus und gibt die SMI-Befehle zu der KL6831/KL6841 weiter. Hierdurch wird sichergestellt, dass nicht mehrere Bausteine gleichzeitig auf das Prozessabbild der KL6831/KL6841 zugreifen. Jeder dieser drei Puffer wird mit einer anderen Priorität (hoch, mittel oder niedrig) abgearbeitet. Der Anwender der SPS-Library kann durch den Parameter *eCommandPriority*, den es bei den meisten Bausteinen gibt, beeinflussen mit welcher Priorität der jeweilige SMI-Befehl von dem Baustein FB_KL6831KL6841Communication() bearbeitet werden soll.

Die Puffer, in denen die SMI-Befehle abgelegt werden, sind alle in einer Variablen vom Typ ST_SMICommandBuffer [▶ 57] enthalten. Pro KL6831/KL6841 gibt es eine Instanz vom Baustein FB_KL6831KL6841Communication() und eine Variable vom Typ ST_SMICommandBuffer. Der Baustein FB_KL6831KL6841Communication() sollte, wenn möglich, in einer separaten, schnelleren Task aufgerufen werden.

Über die Ausgänge des Bausteins kann ermittelt werden, wie stark die Puffer ausgelastet sind. Hierzu werden drei Arrays ausgegeben, bei dem jedes Element (0, 1 oder 2) für einen der drei Puffer (hoch, mittel oder niedrig) steht. Sollten Sie feststellen, dass einer der drei Puffer regelmäßig überläuft, so sollten Sie folgende Maßnahmen in Betracht ziehen:

- Wie stark sind die einzelnen SPS-Task ausgelastet? Der TwinCAT System Manager bietet zur Analyse entsprechende Hilfsmittel an.

- Versuchen Sie die Zykluszeit der Task, in der der Baustein FB_KL6831KL6841Communication() aufgerufen wird, zu verringern. Der Wert sollte nicht größer als 6 ms sein, optimal sind 2 ms.

- Überprüfen Sie die Zykluszeit der SPS-Task, in der die Bausteine für die einzelnen SMI-Befehle aufgerufen werden. Dieser Wert sollte zwischen 10 ms und 60 ms liegen.

- Vermeiden Sie möglichst das Pollen (regelmäßiges Auslesen) von Werten. Lesen Sie nur dann Werte aus, wenn diese auch benötigt werden.

- Verteilen Sie die einzelnen Antriebe gleichmäßig auf mehrere SMI-Linien. Da pro SPS-Zyklus mehrere SMI-Linien gleichzeitig bearbeitet werden, erhöht sich hierdurch der Datendurchsatz insgesamt.

VAR_INPUT

```
bResetMaximumDemandCounter    : BOOL;
bResetOverflowCounter         : BOOL;
bResetInactiveProcessImage    : BOOL;
bResetDataFrameError          : BOOL;
dwOptions                     : DWORD := 0;
```

bResetMaximumDemandCounter: Eine positive Flanke setzt den gespeicherten Wert der maximalen Befehlspuffer-Auslastung *(arrBufferMaximumDemandMeter)* zurück.

bResetOverflowCounter: Eine positive Flanke setzt den gespeicherten Wert der Anzahl der Befehlspuffer-Überläufe *(arrBufferOverflowCounter)* zurück.

bResetInactiveProcessImage: Eine positive Flanke hebt die Sperrung des Prozeßabbildes der Klemme wieder auf. Die Ausgänge *bProcessImageInactive*, *bDigitalInput1Active* und *bDigitalInput2Active* werden wieder auf FALSE gesetzt. Die Sperrung wird aktiviert, sobald einer der beiden digitalen Eingänge an der Klemme betätigt wurde.

bResetDataFrameError: Eine positive Flanke setzt die Meldung für einen Telegrammfehler wieder zurück. Der Ausgang *bDataFrameError* wird wieder auf FALSE gesetzt. Die Sperrung wird aktiviert, sobald von der Klemme ein Telegrammfehler erkannt wurde.

dwOptions: reserviert für zukünftige Erweiterungen.

VAR_OUTPUT

```
bBusy                       : BOOL;
bError                      : BOOL;
udiErrorId                  : UDINT;
arrBufferDemandMeter        : ARRAY [0..2] OF BYTE;
arrBufferMaximumDemandMeter : ARRAY [0..2] OF BYTE;
arrBufferOverflowCounter    : ARRAY [0..2] OF UINT;
bLineIsInitialized          : BOOL;
b24VPowerSupplySwitchedOn   : BOOL;
bDigitalInput1Active        : BOOL;
bDigitalInput2Active        : BOOL;
bProcessImageInactive       : BOOL;
bDataFrameError             : BOOL;
bChecksumError              : BOOL;
```

bBusy: Der Ausgang wird gesetzt, sobald der Baustein einen Befehl verarbeitet und bleibt so lange aktiv, bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

arrBufferDemandMeter: Belegung des jeweiligen Puffers (0 - 100 %).

arrBufferMaximumDemandMeter: Bisherige maximale Auslastung des jeweiligen Puffers (0 - 100 %).

arrBufferOverflowCounter: Bisherige Anzahl der Pufferüberläufe.

bLineIsInitialized: Wird der Baustein das erste Mal aufgerufen (z.B. beim Starten der Steuerung), so wird eine Initialisierung durchgeführt. Während dieser Zeit können keine SMI-Befehle bearbeitet werden. Ist die Initialisierung abgeschlossen, wird dieser Ausgang auf TRUE gesetzt.

b24VPowerSupplySwitchedOn: Die KL6831/KL6841 muß über zwei Anschlüsse mit 24 V DC versorgt werden. Der Ausgang wird gesetzt, sobald die 24 V DC erkannt wurden. Fehlen die 24 V DC geht der Ausgang auf FALSE und es können keine SMI-Befehle über die Steuerung bearbeitet werden, solang die 24 V DC nicht vorhanden sind.

bDigitalInput1Active: Der digitale Eingang 1 an der Klemme wurde betätigt oder ist betätigt (siehe auch Klemmendokumentation). Der Ausgang *bProcessImageInactive* wird gesetzt und es können keine weiteren SMI-Befehle über die Steuerung bearbeitet werden.

bDigitalInput2Active: Der digitale Eingang 2 an der Klemme wurde betätigt oder ist betätigt (siehe auch Klemmendokumentation). Der Ausgang *bProcessImageInactive* wird gesetzt und es können keine weiteren SMI-Befehle über die Steuerung bearbeitet werden.

bProcessImageInactive: Einer der beiden digitalen Eingänge wurde an der Klemme betätigt. Es können keine weiteren SMI-Befehle über die Steuerung bearbeitet werden. Über den Eingang *bResetInactiveProcessImage* muss die Sperrung wieder freigeschaltet werden.

bDataFrameError: Die Klemme hat einen Telegrammfehler auf dem SMI-Bus erkannt. Über den Eingang *bResetDataFrameError* muss der Fehler wieder zurückgesetzt werden.

bChecksumError: Die Klemme hat einen Checksummenfehler auf dem SMI-Bus erkannt. Die Meldung wird automatisch zurückgesetzt, sobald eine Telegramm wieder fehlerfrei übertragen wurde.

VAR_IN_OUT

```
stInData              : ST_KL6831KL6841InData;
stOutData             : ST_KL6831KL6841OutData;
stCommandBuffer       : ST_SMICommandBuffer;
```

stInData: Verweis auf die Struktur [▶ 57] zur Kommunikation mit der KL6831/KL6841.

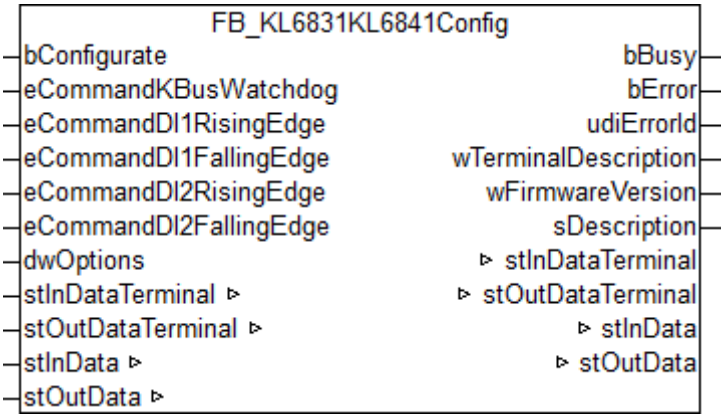
stOutData: Verweis auf die Struktur [▶ 57] zur Kommunikation mit der KL6831/KL6841.

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit den SMI-Bausteinen.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

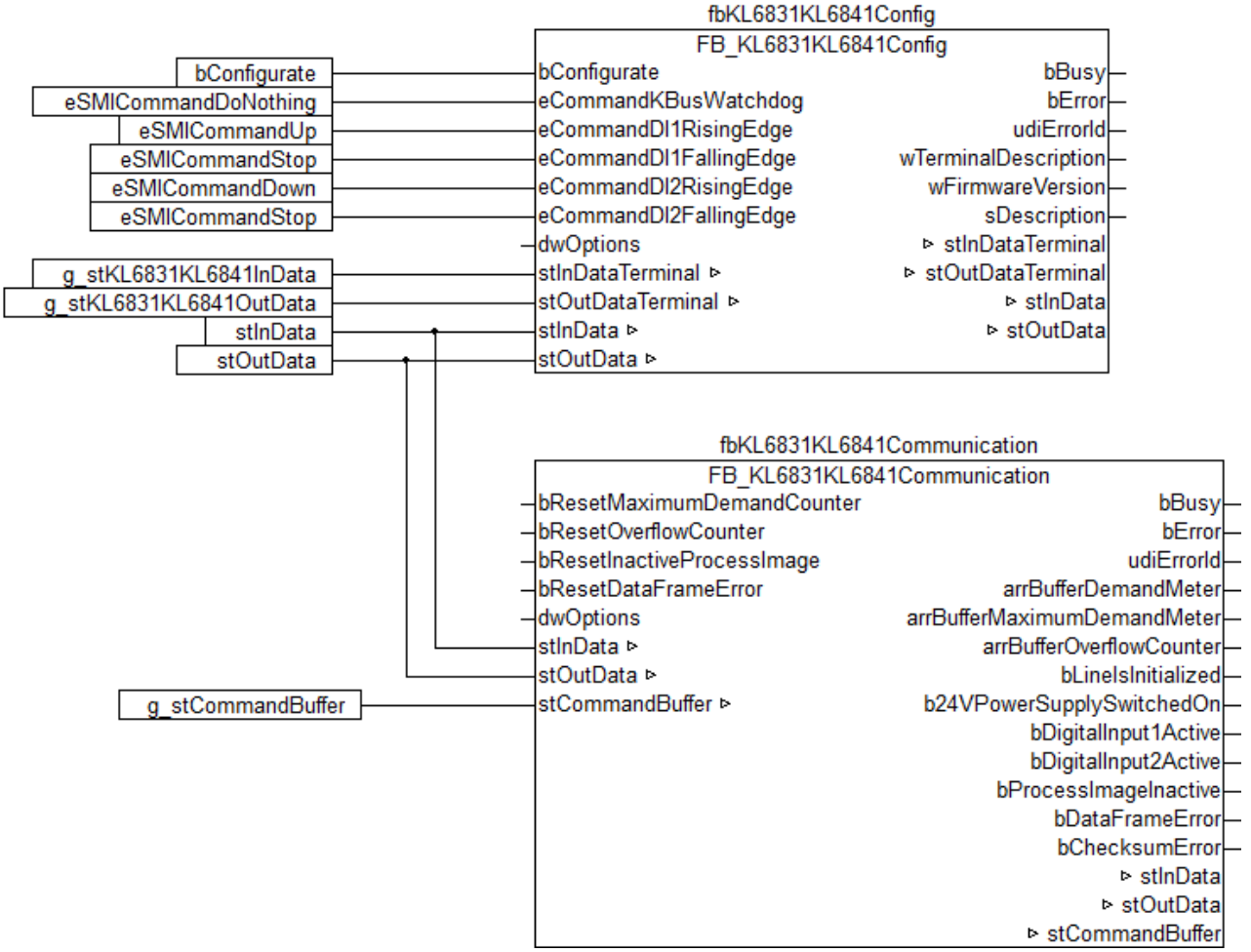
5.1.2 FB_KL6831KL6841Config



Dieser Baustein dient zum Konfigurieren der KL6831/KL6841. Das Konfigurieren wird beim Aufstarten des SPS-Programms ausgeführt oder durch eine positive Flanke am Eingang *bConfigurate*. Die Parameter werden in den jeweiligen Registern der KL6831/KL6841 spannungsausfallsicher abgespeichert. Des Weiteren werden aus der KL6831/KL6841 einige allgemeine Informationen, wie die Version der Firmware, ausgelesen.

Beispiel

Der Baustein wird in der gleichen Task, wie der Baustein FB_KL6831KL6841Communication() [▶ 19] aufgerufen.



Der Baustein FB_KL6831KL6841Config() ist mit dem Prozessabbild der KL6831/KL6841 verbunden. Nach Abschluss der Konfiguration erhält der Baustein FB_KL6831KL6841Communication() die Prozesswerte der KL6831/KL6841. Während des Konfigurieren können keine SMI-Befehle versendet werden.

Beispieldateien entpacken https://infosys.beckhoff.com/content/1031/tcplclibsmi/Resources/

12074356875.zip

VAR_INPUT

```
bConfigurate              : BOOL := FALSE;
eCommandKBusWatchdog      : E_SMIConfigurationCommands := eSMICommandDoNothing;
eCommandDI1RisingEdge     : E_SMIConfigurationCommands := eSMICommandUp;
eCommandDI1FallingEdge    : E_SMIConfigurationCommands := eSMICommandStop;
eCommandDI2RisingEdge     : E_SMIConfigurationCommands := eSMICommandDown;
eCommandDI2FallingEdge    : E_SMIConfigurationCommands := eSMICommandStop;
dwOptions                 : DWORD := 0;
```

bConfigurate: Durch eine positive Flanke an diesem Eingang wird das Konfigurieren der Busklemme gestartet.

eCommandKBusWatchdog: Definiert den SMI-Befehl [▶ 56], der versendet wird, sobald die Busklemme über den K-Bus nicht mehr angesprochen wird. Entspricht Register 33 bis 35 der Busklemme.

eCommandDI1RisingEdge: Definiert den SMI-Befehl, der versendet wird, sobald am Eingang 1 der Busklemme eine steigende Flanke erkannt wird. Entspricht Register 36 bis 38 der Busklemme.

eCommandDI1FallingEdge: Definiert den SMI-Befehl, der versendet wird, sobald am Eingang 1 der Busklemme eine fallende Flanke erkannt wird. Entspricht Register 39 bis 41 der Busklemme.

eCommandDI2RisingEdge: Definiert den SMI-Befehl, der versendet wird, sobald am Eingang 2 der Busklemme eine steigende Flanke erkannt wird. Entspricht Register 42 bis 44 der Busklemme.

eCommandDI2FallingEdge: Definiert den SMI-Befehl, der versendet wird, sobald am Eingang 2 der Busklemme eine fallende Flanke erkannt wird. Entspricht Register 45 bis 47 der Busklemme.

dwOptions: Reserviert für zukünftige Erweiterungen.

VAR_OUTPUT

```
bBusy                 : BOOL;
bError                : BOOL;
udiErrorId            : UDINT;
wTerminalDescription  : WORD;
wFirmwareVersion      : WORD;
sDescription          : STRING;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bConfigurate* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bConfigurate* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

wTerminalDescription: Enthält die Klemmenbezeichnung (z.B. 6831). Entspricht Register 8 der Busklemme.

wFirmwareVersion: Enthält die Version der Firmware. Entspricht Register 9 der Busklemme.

sDescription: Klemmenbezeichnung und die Version der Firmware als String (z.B. 'Terminal KL6831 / Firmware 1D').

VAR_IN_OUT

```
stInDataTerminal          : ST_KL6831KL6841InData;
stOutDataTerminal         : ST_KL6831KL6841OutData;
stInData                  : ST_KL6831KL6841InData;
stOutData                 : ST_KL6831KL6841OutData;
```

stInDataTerminal: Verweis auf die Struktur [▶ 57] zur Kommunikation mit der KL6831/KL6841.

stOutDataTerminal: Verweis auf die Struktur [▶ 57] zur Kommunikation mit der KL6831/KL6841.

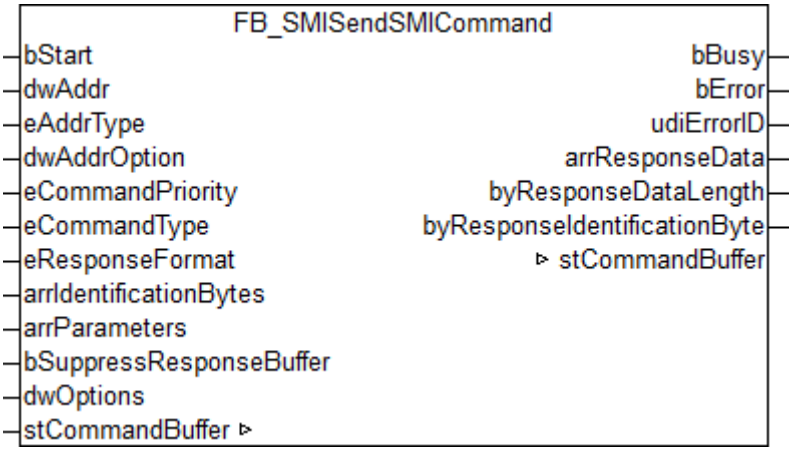
stInData: Verweis auf die Struktur zur Kommunikation mit dem Baustein FB_KL6831KL6841Communication() [▶ 19].

stOutData: Verweis auf die Struktur zur Kommunikation mit dem Baustein FB_KL6831KL6841Communication() [▶ 19].

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2253	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.1.0

5.1.3 FB_SMISendSMICommand



Dieser Baustein dient zum allgemeinen Senden eines SMI-Kommandos. Hierzu muss der genaue Aufbau eines SMI-Befehls und die Funktionsweise der KL6831/KL6841 bekannt sein. Der Einsatz dieses Bausteins ist nur notwendig, wenn ein SMI-Befehl versendet werden soll, der nicht durch die anderen SPS-Bausteine abgedeckt wird.

VAR_INPUT

```
bStart                  : BOOL;
dwAddr                  : DWORD := 0;
eAddrType               : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption            : DWORD := 0;
eCommandPriority        : E_SMICommandPriority := eSMICommandPriorityMiddle;
eCommandType            : E_SMICommandType := eSMICommandTypeWrite;
eResponseFormat         : E_SMIResponseFormat := eSMIResponseFormatDiagnosis;
arrIdentificationBytes  : ARRAY [0..2] OF BYTE;
arrParameters           : ARRAY [0..2] OF DWORD;
bSuppressResponseBuffer : BOOL := FALSE;
dwOptions               : DWORD := 0;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers, zur Gruppenadressierung oder als Slave-Id ausgewertet werden soll.

dwAddrOption: Wird das SMI-Gerät per Slave-Id adressiert (*eAddrType* = *eSMIAddrTypeSlaveId*), so muß über diesen Eingang der Herstellercode [▶ 67] angegeben werden.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

eCommandType: Kommandoart [▶ 55]: Schreiben/Lesen. Dieser Parameter beeinflusst das Bit 5 vom Startbyte des SMI-Telegramms.

eResponseFormat: Antwortformat [▶ 57]: Diagnose-Sonderformat/Standard. Dieser Parameter beeinflusst das Bit 6 vom Startbyte des SMI-Telegramms.

arrIdentificationBytes: Ein SMI-Telegramm kann aus bis zu 3 Blöcken bestehen. Jeder Block besitzt ein Kennungsbyte. Über dieses Array werden die drei Kennungsbytes definiert.

arrParameters: Ein SMI-Telegramm kann aus bis zu 3 Blöcken bestehen. Jeder Block besitzt bis zu vier Wertebytes. Über dieses Array werden die Wertebytes der einzelnen Blöcke definiert.

bSuppressResponseBuffer: Wird dieser Eingang auf TRUE gesetzt, so wird der interne Software-Puffer nicht mit den Antworten des Bausteins FB_KL6831KL6841Communication() [▶ 19] gefüllt.

dwOptions: reserviert für zukünftige Erweiterungen.

VAR_OUTPUT

```
bBusy                       : BOOL;
bError                      : BOOL;
udiErrorId                  : UDINT;
arrResponseData             : ARRAY [0..7] OF BYTE;
byResponseDataLength        : BYTE;
byResponseIdentificationByte : BYTE;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

arrResponseData: Die empfangenen Daten von den SMI-Geräten.

byResponseDataLength: Die Länge der empfangenen Daten, in Bytes.

byResponseIdentificationByte: Das empfangene Kennungsbyte.

VAR_IN_OUT

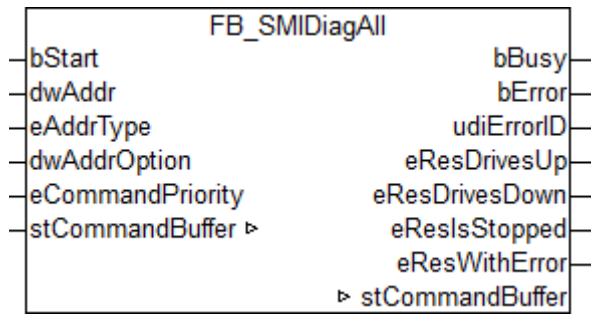
```
stCommandBuffer          : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.4 FB_SMIDiagAll



Mit diesem Befehl kann ermittelt werden, in welche Richtung die Antriebe fahren, ob diese gestoppt sind oder ob ein Motorfehler vorliegt. Der Befehl kann auch an mehrere SMI-Slaves gesendet werden. Dadurch lassen sich die Zustände aller SMI-Slaves mit einem Befehl abfragen.

Das Ergebnis der Abfrage wird durch vier Ausgänge weitergegeben. Jeder dieser Ausgänge kann drei Zustände annehmen:

- Die Bedingung trifft auf mindestens einen Antrieb zu.
- Die Bedingung trifft auf keinen Antrieb zu.
- Die Bedingung konnte nicht ermittel werden.

Weiter unten werden hierzu einige Beispiele erläutert.

VAR_INPUT

```
bStart            : BOOL;
dwAddr            : DWORD := 0;
eAddrType         : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption      : DWORD := 0;
eCommandPriority  : E_SMICommandPriority := eSMICommandPriorityMiddle;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers oder zur Gruppenadressierung ausgewertet werden soll. Eine Adressierung per Slave-Id (*eAddrType* = *eSMIAddrTypeSlaveId*) ist nicht zulässig.

dwAddrOption: reserviert für zukünftige Erweiterungen.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

VAR_OUTPUT

```
bBusy             : BOOL;
bError            : BOOL;
udiErrorId        : UDINT;
eResDrivesUp      : E_SMIDiagResDrivesUp;
eResDrivesDown    : E_SMIDiagResDrivesDown;
eResIsStopped     : E_SMIDiagResIsStopped;
eResWithError     : E_SMIDiagResWithError;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

eResDrivesUp: Mindestens ein Motor fährt hoch [▶ 56] / Kein Motor fährt hoch / Der Wert ist undefiniert.

eResDrivesDown: Mindestens ein Motor fährt runter [▶ 56] / Kein Motor fährt runter / Der Wert ist undefiniert.

eResIsStopped: Mindestens ein Motor ist gestopped [▶ 56] / Kein Motor ist gestopped / Der Wert ist undefiniert.

eResWithError: Mindestens ein Motor ist in Störung [▶ 57] / Kein Motor ist in Störung / Der Wert ist undefiniert.

VAR_IN_OUT

```
stCommandBuffer   : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Beispiele

Alle Antriebe sind gestoppt:

Ausgänge	Bedeutung
eResDrivesUp = eSMIDiagResNoMotorDrivesUp	Kein Antrieb fährt hoch.
eResDrivesDown = eSMIDiagResNoMotorDrivesDown	Kein Antrieb fährt runter.
eResIsStopped = eSMIDiagResAtLeastOneMotorIsStopped	Mindestens ein Antrieb ist gestoppt.
eResWithError = eSMIDiagResNoMotorWithError	Kein Antrieb mit Motorfehler.

Alle Antriebe fahren hoch:

Ausgänge	Bedeutung
eResDrivesUp = eSMIDiagResAtLeastOneMotorDrivesUp	Mindestens ein Antrieb fährt hoch.
eResDrivesDown = eSMIDiagResNoMotorDrivesDown	Kein Antrieb fährt runter.
eResIsStopped = eSMIDiagResNoMotorIsStopped	Kein Antrieb ist gestoppt.
eResWithError = eSMIDiagResNoMotorWithError	Kein Antrieb mit Motorfehler.

Ein Antrieb ist gestoppt und ein Antrieb fährt hoch:

Ausgänge	Bedeutung
eResDrivesUp = eSMIDiagResAtLeastOneMotorDrivesUp	Mindestens ein Antrieb fährt hoch.
eResDrivesDown = eSMIDiagResNoMotorDrivesDown	Kein Antrieb fährt runter.
eResIsStopped = eSMIDiagResAtLeastOneMotorIsStopped	Mindestens ein Antrieb ist gestoppt.
eResWithError = eSMIDiagResNoMotorWithError	Kein Antrieb mit Motorfehler.

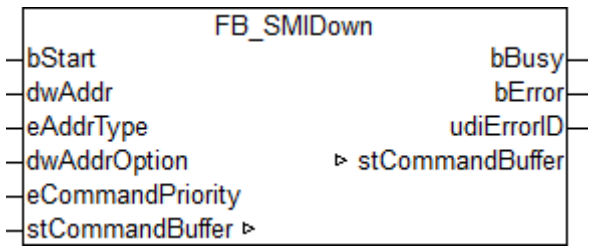
Ein Antrieb ist gestoppt, ein Antrieb fährt hoch und ein Antrieb fährt runter:

Ausgänge	Bedeutung
eResDrivesUp = eSMIDiagResAtLeastOneMotorDrivesUp	Mindestens ein Antrieb fährt hoch.
eResDrivesDown = eSMIDiagResAtLeastOneMotorDrivesDown	Mindestens ein Antrieb fährt runter.
eResIsStopped = eSMIDiagResAtLeastOneMotorIsStopped	Mindestens ein Antrieb ist gestoppt.
eResWithError = eSMIDiagResNoMotorWithError	Kein Antrieb mit Motorfehler.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.5 FB_SMIDown



Motorlauf bis zur unteren Endlage.

VAR_INPUT

```
bStart            : BOOL;
dwAddr            : DWORD := 0;
eAddrType         : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption      : DWORD := 0;
eCommandPriority  : E_SMICommandPriority := eSMICommandPriorityMiddle;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers, zur Gruppenadressierung oder als Slave-Id ausgewertet werden soll.

dwAddrOption: Wird das SMI-Gerät per Slave-Id adressiert (*eAddrType* = *eSMIAddrTypeSlaveId*), so muß über diesen Eingang der Herstellercode [▶ 67] angegeben werden.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

VAR_OUTPUT

```
bBusy         : BOOL;
bError        : BOOL;
udiErrorId    : UDINT;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

VAR_IN_OUT

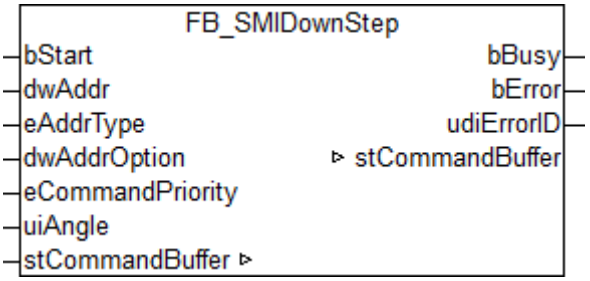
```
stCommandBuffer   : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.6 FB_SMIDownStep



Motorlauf nach unten um einen vorgegebenen Winkelgrad (0-510 Grad).

VAR_INPUT

```
bStart            : BOOL;
dwAddr            : DWORD := 0;
eAddrType         : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption      : DWORD := 0;
eCommandPriority  : E_SMICommandPriority := eSMICommandPriorityMiddle;
uiAngle           : UINT := 0;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers, zur Gruppenadressierung oder als Slave-Id ausgewertet werden soll.

dwAddrOption: Wird das SMI-Gerät per Slave-Id adressiert (*eAddrType* = *eSMIAddrTypeSlaveId*), so muß über diesen Eingang der Herstellercode [▶ 67] angegeben werden.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

uiAngle: der vorgegebene Winkelgrad. Der Wertebereich ist 0...510 Grad. Der SMI-Standard reduziert die Genauigkeit auf eine Auflösung von 2 Grad.

VAR_OUTPUT

```
bBusy         : BOOL;
bError        : BOOL;
udiErrorId    : UDINT;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

VAR_IN_OUT

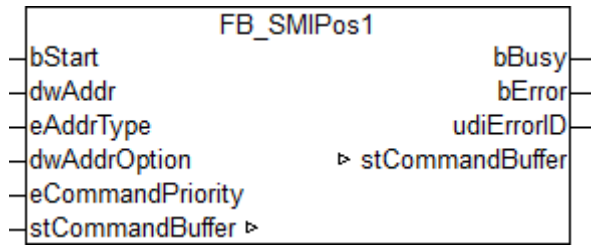
```
stCommandBuffer   : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.7 FB_SMIPos1



Fahrt zur motorseitig konfigurierten Fixposition *Pos1*. Auslesen und verändern von *Pos1* ist mit den Bausteinen FB_SMIPos1Read() [▶ 31] und FB_SMIPos1Write() [▶ 32] möglich.

VAR_INPUT

```
bStart            : BOOL;
dwAddr            : DWORD := 0;
eAddrType         : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption      : DWORD := 0;
eCommandPriority  : E_SMICommandPriority := eSMICommandPriorityMiddle;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers, zur Gruppenadressierung oder als Slave-Id ausgewertet werden soll.

dwAddrOption: Wird das SMI-Gerät per Slave-Id adressiert (*eAddrType* = *eSMIAddrTypeSlaveId*), so muß über diesen Eingang der Herstellercode [▶ 67] angegeben werden.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

VAR_OUTPUT

```
bBusy         : BOOL;
bError        : BOOL;
udiErrorId    : UDINT;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

VAR_IN_OUT

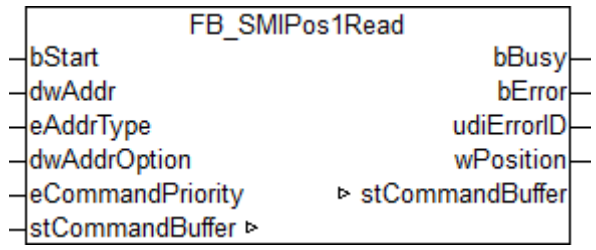
```
stCommandBuffer    : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.8 FB_SMIPos1Read



Lesen der motorseitig konfigurierten Fixposition *Pos1*. Mit dem Baustein FB_SMIPos1Write() [▶ 32] kann *Pos1* verändert werden.

VAR_INPUT

```
bStart            : BOOL;
dwAddr            : DWORD := 0;
eAddrType         : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption      : DWORD := 0;
eCommandPriority  : E_SMICommandPriority := eSMICommandPriorityMiddle;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers oder zur Gruppenadressierung ausgewertet werden soll. Eine Adressierung per Slave-Id (*eAddrType* = *eSMIAddrTypeSlaveId*) ist nicht zulässig.

dwAddrOption: reserviert für zukünftige Erweiterungen.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

VAR_OUTPUT

```
bBusy         : BOOL;
bError        : BOOL;
udiErrorId    : UDINT;
wPosition     : WORD;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

wPosition: Die ausgelesene Fixposition *Pos1*. Hierbei entspricht der Wert 0 der oberen Endlage und der Wert 65535 (0xFFFF) der unteren Endlage.

VAR_IN_OUT

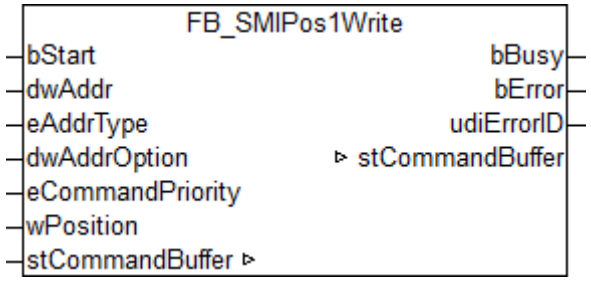
```
stCommandBuffer    : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.9 FB_SMIPos1Write



Schreiben der motorseitig konfigurierbaren Fixposition *Pos1*. Mit dem Baustein FB_SMIPos1Read() [▶ 31] kann *Pos1* ausgelesen werden.

VAR_INPUT

```
bStart            : BOOL;
dwAddr            : DWORD := 0;
eAddrType         : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption      : DWORD := 0;
eCommandPriority  : E_SMICommandPriority := eSMICommandPriorityMiddle;
wPosition         : WORD := 0;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers, zur Gruppenadressierung oder als Slave-Id ausgewertet werden soll.

dwAddrOption: Wird das SMI-Gerät per Slave-Id adressiert (*eAddrType* = *eSMIAddrTypeSlaveId*), so muß über diesen Eingang der Herstellercode [▶ 67] angegeben werden.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

wPosition: Die neue Fixposition *Pos1*. Hierbei entspricht der Wert 0 der oberen Endlage und der Wert 65535 (0xFFFF) der unteren Endlage.

VAR_OUTPUT

```
bBusy         : BOOL;
bError        : BOOL;
udiErrorId    : UDINT;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

VAR_IN_OUT

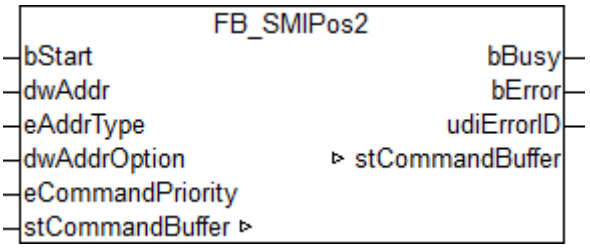
```
stCommandBuffer    : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.10 FB_SMIPos2



Fahrt zur motorseitig konfigurierten Fixposition *Pos2*. Auslesen und verändern von *Pos2* ist mit den Bausteinen FB_SMIPos2Read() [▶ 34] und FB_SMIPos2Write() [▶ 35] möglich.

VAR_INPUT

```
bStart            : BOOL;
dwAddr            : DWORD := 0;
eAddrType         : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption      : DWORD := 0;
eCommandPriority  : E_SMICommandPriority := eSMICommandPriorityMiddle;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers, zur Gruppenadressierung oder als Slave-Id ausgewertet werden soll.

dwAddrOption: Wird das SMI-Gerät per Slave-Id adressiert (*eAddrType* = *eSMIAddrTypeSlaveId*), so muß über diesen Eingang der Herstellercode [▶ 67] angegeben werden.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

VAR_OUTPUT

```
bBusy         : BOOL;
bError        : BOOL;
udiErrorId    : UDINT;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

VAR_IN_OUT

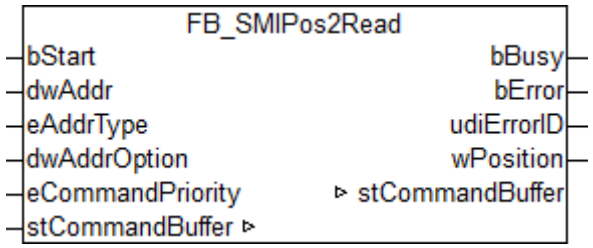
```
stCommandBuffer    : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.11 FB_SMIPos2Read



Lesen der motorseitig konfigurierten Fixposition *Pos2*. Mit dem Baustein FB_SMIPos2Write() [▶ 35] kann *Pos2* verändert werden.

VAR_INPUT

```
bStart             : BOOL;
dwAddr             : DWORD := 0;
eAddrType          : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption       : DWORD := 0;
eCommandPriority   : E_SMICommandPriority := eSMICommandPriorityMiddle;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers oder zur Gruppenadressierung ausgewertet werden soll. Eine Adressierung per Slave-Id (*eAddrType* = *eSMIAddrTypeSlaveId*) ist nicht zulässig.

dwAddrOption: reserviert für zukünftige Erweiterungen.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

VAR_OUTPUT

```
bBusy         : BOOL;
bError        : BOOL;
udiErrorId    : UDINT;
wPosition     : WORD;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

wPosition: Die ausgelesene Fixposition *Pos2*. Hierbei entspricht der Wert 0 der oberen Endlage und der Wert 65535 (0xFFFF) der unteren Endlage.

VAR_IN_OUT

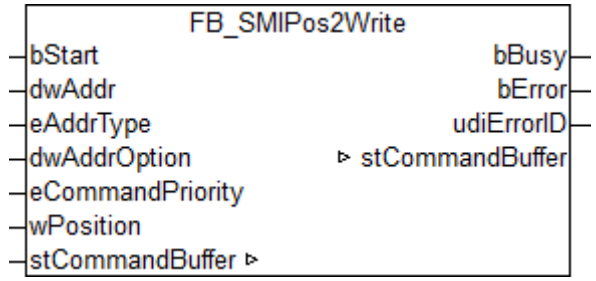
```
stCommandBuffer    : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.12 FB_SMIPos2Write



Schreiben der motorseitig konfigurierbaren Fixposition *Pos2*. Mit dem Baustein FB_SMIPos2Read() [▶ 34] kann *Pos2* ausgelesen werden.

VAR_INPUT

```
bStart             : BOOL;
dwAddr             : DWORD := 0;
eAddrType          : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption       : DWORD := 0;
eCommandPriority   : E_SMICommandPriority := eSMICommandPriorityMiddle;
wPosition          : WORD := 0;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers, zur Gruppenadressierung oder als Slave-Id ausgewertet werden soll.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

dwAddrOption: Wird das SMI-Gerät per Slave-Id adressiert (*eAddrType* = *eSMIAddrTypeSlaveId*), so muß über diesen Eingang der Herstellercode [▶ 67] angegeben werden.

wPosition: Die neue Fixposition *Pos2*. Hierbei entspricht der Wert 0 der oberen Endlage und der Wert 65535 (0xFFFF) der unteren Endlage.

VAR_OUTPUT

```
bBusy          : BOOL;
bError         : BOOL;
udiErrorId     : UDINT;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

VAR_IN_OUT

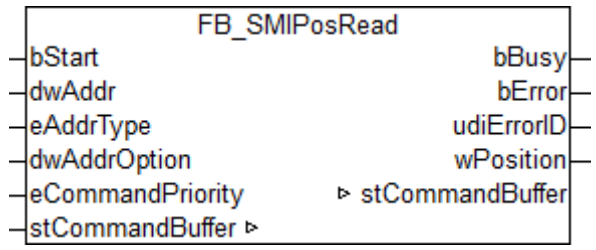
```
stCommandBuffer    : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.13 FB_SMIPosRead



Die aktuelle Position wird aus dem Antrieb ausgelesen.

VAR_INPUT

```
bStart             : BOOL;
dwAddr             : DWORD := 0;
eAddrType          : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption       : DWORD := 0;
eCommandPriority   : E_SMICommandPriority := eSMICommandPriorityMiddle;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers oder zur Gruppenadressierung ausgewertet werden soll. Eine Adressierung per Slave-Id (*eAddrType* = *eSMIAddrTypeSlaveId*) ist nicht zulässig.

dwAddrOption: reserviert für zukünftige Erweiterungen.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

VAR_OUTPUT

```
bBusy          : BOOL;
bError         : BOOL;
udiErrorId     : UDINT;
wPosition      : WORD;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

wPosition: Die ausgelesene Position. Hierbei entspricht der Wert 0 der oberen Endlage und der Wert 65535 (0xFFFF) der unteren Endlage.

VAR_IN_OUT

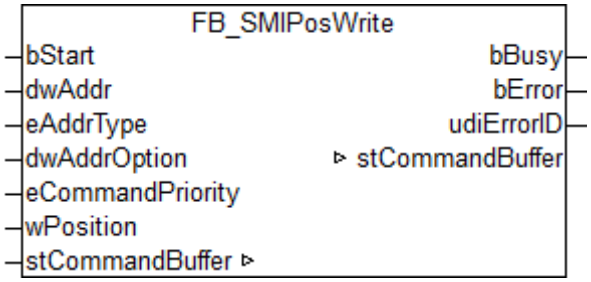
```
stCommandBuffer    : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.14 FB_SMIPosWrite



Der Antrieb wird auf die angegebene Position gefahren.

VAR_INPUT

```
bStart             : BOOL;
dwAddr             : DWORD := 0;
eAddrType          : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption       : DWORD := 0;
eCommandPriority   : E_SMICommandPriority := eSMICommandPriorityMiddle;
wPosition          : WORD := 0;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers, zur Gruppenadressierung oder als Slave-Id ausgewertet werden soll.

dwAddrOption: Wird das SMI-Gerät per Slave-Id adressiert (*eAddrType* = *eSMIAddrTypeSlaveId*), so muß über diesen Eingang der Herstellercode [▶ 67] angegeben werden.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

wPosition: Die neue Position. Hierbei entspricht der Wert 0 der oberen Endlage und der Wert 65535 (0xFFFF) der unteren Endlage.

VAR_OUTPUT

```
bBusy         : BOOL;
bError        : BOOL;
udiErrorId    : UDINT;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

VAR_IN_OUT

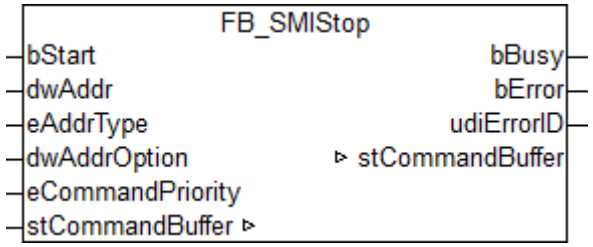
```
stCommandBuffer   : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.15 FB_SMIStop



Der Motorlauf wird gestoppt.

VAR_INPUT

```
bStart            : BOOL;
dwAddr            : DWORD := 0;
eAddrType         : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption      : DWORD := 0;
eCommandPriority  : E_SMICommandPriority := eSMICommandPriorityMiddle;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers, zur Gruppenadressierung oder als Slave-Id ausgewertet werden soll.

dwAddrOption: Wird das SMI-Gerät per Slave-Id adressiert (*eAddrType* = *eSMIAddrTypeSlaveId*), so muß über diesen Eingang der Herstellercode [▶ 67] angegeben werden.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

VAR_OUTPUT

```
bBusy         : BOOL;
bError        : BOOL;
udiErrorId    : UDINT;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

VAR_IN_OUT

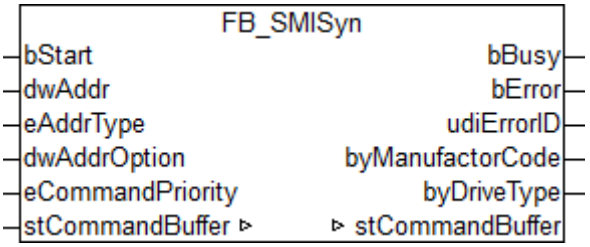
```
stCommandBuffer   : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.16 FB_SMISyn



Der Herstellercode und der Antriebstyp werden abgefragt.

VAR_INPUT

```
bStart            : BOOL;
dwAddr            : DWORD := 0;
eAddrType         : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption      : DWORD := 0;
eCommandPriority  : E_SMICommandPriority := eSMICommandPriorityMiddle;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers oder zur Gruppenadressierung ausgewertet werden soll. Eine Adressierung per Slave-Id (*eAddrType* = *eSMIAddrTypeSlaveId*) ist nicht zulässig.

dwAddrOption: reserviert für zukünftige Erweiterungen.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

VAR_OUTPUT

```
bBusy             : BOOL;
bError            : BOOL;
udiErrorId        : UDINT;
byManufactorCode  : BYTE;
byDriveType       : BYTE;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

byManufactorCode: Der Herstellercode [▶ 67] (1-14).

byDriveType: Der Typ des Antriebs (0-15). Die Bedeutung ist herstellerspezifisch.

VAR_IN_OUT

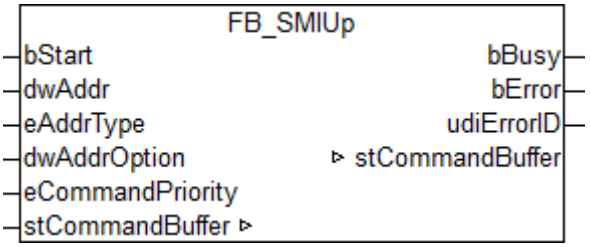
```
stCommandBuffer   : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.17 FB_SMIUp



Motorlauf bis zur oberen Endlage.

VAR_INPUT

```
bStart            : BOOL;
dwAddr            : DWORD := 0;
eAddrType         : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption      : DWORD := 0;
eCommandPriority  : E_SMICommandPriority := eSMICommandPriorityMiddle;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers, zur Gruppenadressierung oder als Slave-Id ausgewertet werden soll.

dwAddrOption: Wird das SMI-Gerät per Slave-Id adressiert (*eAddrType* = *eSMIAddrTypeSlaveId*), so muß über diesen Eingang der Herstellercode [▶ 67] angegeben werden.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

VAR_OUTPUT

```
bBusy         : BOOL;
bError        : BOOL;
udiErrorId    : UDINT;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

VAR_IN_OUT

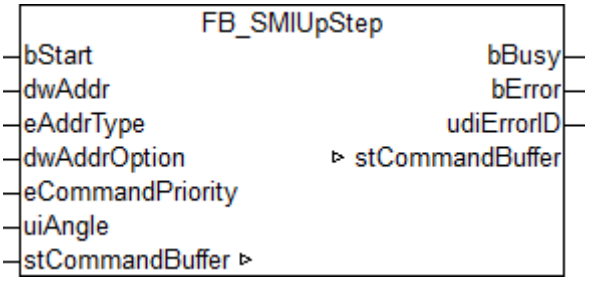
```
stCommandBuffer   : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.18 FB_SMIUpStep



Motorlauf nach oben um einen vorgegebenen Winkelgrad (0-510 Grad).

VAR_INPUT

```
bStart            : BOOL;
dwAddr            : DWORD := 0;
eAddrType         : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption      : DWORD := 0;
eCommandPriority  : E_SMICommandPriority := eSMICommandPriorityMiddle;
uiAngle           : UINT := 0;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers, zur Gruppenadressierung oder als Slave-Id ausgewertet werden soll.

dwAddrOption: Wird das SMI-Gerät per Slave-Id adressiert (*eAddrType* = *eSMIAddrTypeSlaveId*), so muß über diesen Eingang der Herstellercode [▶ 67] angegeben werden.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

uiAngle: der vorgegebene Winkelgrad. Der Wertebereich ist 0...510 Grad. Der SMI-Standard reduziert die Genauigkeit auf eine Auflösung von 2 Grad.

VAR_OUTPUT

```
bBusy         : BOOL;
bError        : BOOL;
udiErrorId    : UDINT;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

VAR_IN_OUT

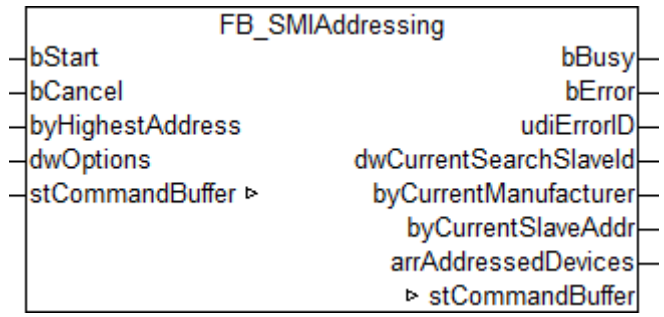
```
stCommandBuffer   : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.19 FB_SMIAddressing



Dieser Funktionsbaustein adressiert die angeschlossenden SMI-Geräte nach dem Zufallsprinzip. Der Anwender hat keinen Einfluss darauf, welches SMI-Gerät welche Adresse zugewiesen bekommt. Die Vergabe der Adressen erfolgt absteigend, beginnend bei der Adresse, die durch den Parameter *byHighestAddress* vorgegeben wird.

Durch eine positive Flanke an dem Eingang *bStart* wird der Baustein gestartet und der Ausgang *bBusy* geht auf TRUE. Der Baustein adressiert jetzt selbständig alle SMI-Geräte. Die Ausgangsvariable *arrAddressedDevices* gibt Auskunft darüber, welche SMI-Gerätes schon eine Adresse erhalten haben. Sind alle SMI-Geräte adressiert, so geht der Ausgang *bBusy* wieder auf FALSE. Die Adressierung kann vorzeitig durch eine positive Flanke am Eingang *bCancel* abgebrochen werden. Abhängig davon, wie viele SMI-Geräte angeschlossen sind, kann die Abarbeitung dieses Bausteines mehrere Minuten dauern.

VAR_INPUT

```
bStart               : BOOL;
bCancel              : BOOL;
byHighestAddress     : BYTE := 15;
dwOptions            : DWORD := 0;
```

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

bCancel: Über eine positive Flanke an diesem Eingang wird der Baustein deaktiviert und die Suche abgebrochen.

byHighestAddress: Adresse, ab der absteigend die SMI-Geräte adressiert werden (0-15).

dwOptions: Reserviert für zukünftige Erweiterungen.

VAR_OUTPUT

```
bBusy                   : BOOL;
bError                  : BOOL;
udiErrorId              : UDINT;
dwCurrentSearchSlaveId  : DWORD;
byCurrentManufacturer   : BYTE;
byCurrentSlaveAddr      : BYTE;
arrAddressedDevices     : ARRAY [0..15] OF BOOL;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

dwCurrentSearchSlaveId: Aktuelle Slave-Id, die im Such-Algorithmus verwendet wird.

byCurrentManufacturer: Aktueller Herstellercode [▶ 67], der im Such-Algorithmus verwendet wird.

byCurrentSlaveAddr: Aktuelle Adresse, die im Such-Algorithmus verwendet wird. Solange der Wert größer *byHighestAddress* ist, wurde noch kein SMI-Gerät adressiert.

arrAddressedDevices: Wird einem SMI-Gerät eine Adresse zugewiesen, so wird in dem Array das entsprechende Element auf TRUE gesetzt.

VAR_IN_OUT

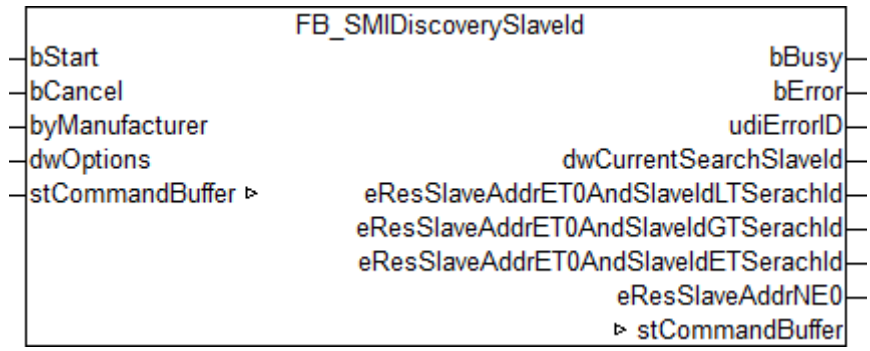
```
stCommandBuffer      : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.20 FB_SMIDiscoverySlaveId



Es wird der erste Antrieb gesucht, der dem vorgegebenen Herstellercode entspricht und bei dem die Adresse 0 ist. Dieser Baustein findet Verwendung bei der Adressierung von SMI-Geräten und wird im Baustein FB_SMIAddressing() [▶ 43] benutzt.

VAR_INPUT

```
bStart               : BOOL;
bCancel              : BOOL;
byManufacturer       : BYTE := 0;
dwOptions            : DWORD := 0;
```

bStart: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert und die Suche gestartet.

bCancel: Über eine positive Flanke an diesem Eingang wird der Baustein deaktiviert und die Suche abgebrochen.

byManufacturer: Der vorgegebene Herstellercode [▶ 67] für die Suche nach dem SMI-Gerät. Einige SMI-Geräte erlauben nicht den Herstellercode 0.

dwOptions: reserviert für zukünftige Erweiterungen.

VAR_OUTPUT

```
bBusy                                : BOOL;
bError                               : BOOL;
udiErrorId                           : UDINT;
dwCurrentSearchSlaveId               : DWORD;
eResSlaveAddrET0AndSlaveIdLTSerachId  : E_SMICompResSlaveAddrET0AndSlaveIdLTSearchId;
eResSlaveAddrET0AndSlaveIdGTSerachId  : E_SMICompResSlaveAddrET0AndSlaveIdGTSearchId;
eResSlaveAddrET0AndSlaveIdETSerachId  : E_SMICompResSlaveAddrET0AndSlaveIdETSearchId;
eResSlaveAddrNE0                     : E_SMICompResSlaveAddrNE0;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

dwCurrentSearchSlaveId: Sobald der Bausteine seine Ausführung beendet hat (*bBusy* wechselt von TRUE auf FALSE) zeigt dieser Ausgang die Slave-Id des gefundenen SMI-Gerätes an.

eResSlaveAddrET0AndSlaveIdLTSerachId: Bei mindestens einem Motor / Bei keinem Motor ist die Adresse gleich 0 und die Slave-Id ist kleiner als die gesuchte Slave-Id [▶ 56] (*dwSlave-Id*) / Der Wert ist undefiniert.

eResSlaveAddrET0AndSlaveIdGTSerachId: Bei mindestens einem Motor / Bei keinem Motor ist die Adresse gleich 0 und die Slave-Id ist größer als die gesuchte Slave-Id [▶ 56] (*dwSlave-Id*) / Der Wert ist undefiniert.

eResSlaveAddrET0AndSlaveIdETSerachId: Bei mindestens einem Motor / Bei keinem Motor ist die Adresse gleich 0 und die Slave-Id ist ebenfalls gleich der gesuchten Slave-Id [▶ 55] (*dwSlave-Id*) / Der Wert ist undefiniert.

eResSlaveAddrNE0: Bei mindestens einem Motor / Bei keinem Motor ist die Adresse ungleich 0 [▶ 56] / Der Wert ist undefiniert.

VAR_IN_OUT

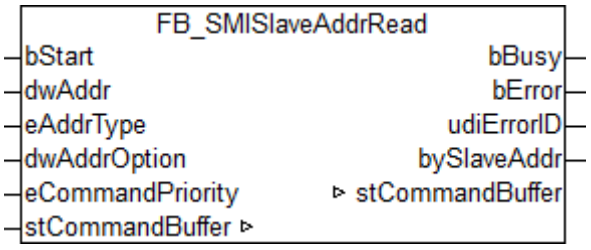
```
stCommandBuffer              : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.21 FB_SMISlaveAddrRead



Die Adresse (0-15) eines Antriebes wird ausgelesen.

VAR_INPUT

```
bStart            : BOOL;
dwAddr            : DWORD := 0;
eAddrType         : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption      : DWORD := 0;
eCommandPriority  : E_SMICommandPriority := eSMICommandPriorityMiddle;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers, zur Gruppenadressierung oder als Slave-Id ausgewertet werden soll.

dwAddrOption: Wird das SMI-Gerät per Slave-Id adressiert (*eAddrType* = *eSMIAddrTypeSlaveId*), so muß über diesen Eingang der Herstellercode [▶ 67] angegeben werden.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

VAR_OUTPUT

```
bBusy          : BOOL;
bError         : BOOL;
udiErrorId     : UDINT;
bySlaveAddr    : BYTE;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

bySlaveAddr: Die ausgelesende Slave-Adresse (0-15).

VAR_IN_OUT

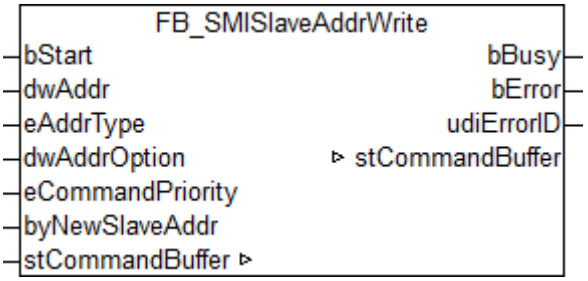
```
stCommandBuffer    : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.22 FB_SMISlaveAddrWrite



Die Adresse (0-15) einer oder mehrerer Antriebe schreiben.

VAR_INPUT

```
bStart            : BOOL;
dwAddr            : DWORD := 0;
eAddrType         : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption      : DWORD := 0;
eCommandPriority  : E_SMICommandPriority := eSMICommandPriorityMiddle;
byNewSlaveAddr    : BYTE := 0;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers, zur Gruppenadressierung oder als Slave-Id ausgewertet werden soll.

dwAddrOption: Wird das SMI-Gerät per Slave-Id adressiert (*eAddrType* = *eSMIAddrTypeSlaveId*), so muß über diesen Eingang der Herstellercode [▶ 67] angegeben werden.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

byNewSlaveAddr: Die neue Slave-Adresse (0-15).

VAR_OUTPUT

```
bBusy         : BOOL;
bError        : BOOL;
udiErrorId    : UDINT;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

VAR_IN_OUT

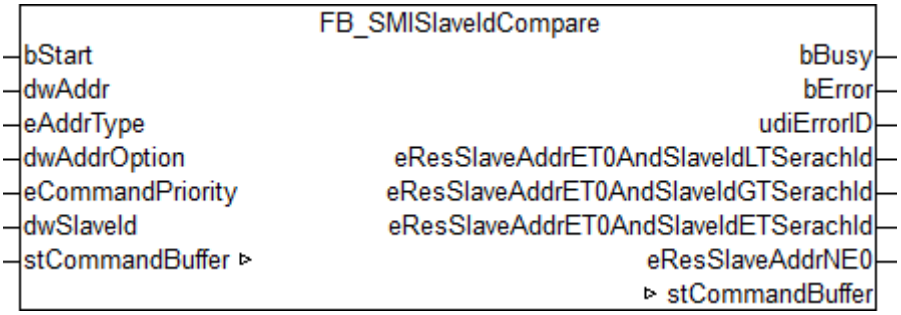
```
stCommandBuffer   : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.23 FB_SMISlaveIdCompare



Eine vorgegebene Slave-Id (32 Bit Key-Id) wird mit der motorseitig definierten Slave-Id (32 Bit Key-Id) eines oder mehrerer Antriebe verglichen. Der Befehl kann auch an mehrere SMI-Slaves gesendet werden.

Das Ergebnis der Abfrage wird durch vier Ausgänge weitergegeben. Jeder dieser Ausgänge kann drei Zustände annehmen:

- Die Bedingung trifft auf mindestens einen Antrieb zu.
- Die Bedingung trifft auf keinen Antrieb zu.
- Die Bedingung konnte nicht ermittel werden.

Weiter unten werden hierzu einige Beispiele erläutert.

VAR_INPUT

```
bStart            : BOOL;
dwAddr            : DWORD := 0;
eAddrType         : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption      : DWORD := 0;
eCommandPriority  : E_SMICommandPriority := eSMICommandPriorityMiddle;
dwSlaveId         : DWORD := 0;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers, zur Gruppenadressierung oder als Slave-Id ausgewertet werden soll.

dwAddrOption: Wird das SMI-Gerät per Slave-Id adressiert (*eAddrType* = *eSMIAddrTypeSlaveId*), so muß über diesen Eingang der Herstellercode [▶ 67] angegeben werden.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

dwSlaveId: Die Slave-Id, mit der die motorseitige Slave-Id verglichen wird.

VAR_OUTPUT

```
bBusy                                : BOOL;
bError                               : BOOL;
udiErrorId                           : UDINT;
eResSlaveAddrET0AndSlaveIdLTSerachId  : E_SMICompResSlaveAddrET0AndSlaveIdLTSearchId;
eResSlaveAddrET0AndSlaveIdGTSerachId  : E_SMICompResSlaveAddrET0AndSlaveIdGTSearchId;
eResSlaveAddrET0AndSlaveIdETSerachId  : E_SMICompResSlaveAddrET0AndSlaveIdETSearchId;
eResSlaveAddrNE0                     : E_SMICompResSlaveAddrNE0;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

eResSlaveAddrET0AndSlaveIdLTSerachId: Bei mindestens einem Motor / Bei keinem Motor ist die Adresse gleich 0 und die Slave-Id ist kleiner als die gesuchte Slave-Id [▶ 56] (*dwSlaveId*) / Der Wert ist undefiniert.

eResSlaveAddrET0AndSlaveIdGTSerachId: Bei mindestens einem Motor / Bei keinem Motor ist die Adresse gleich 0 und die Slave-Id ist größer als die gesuchte Slave-Id [▶ 56] (*dwSlaveId*) / Der Wert ist undefiniert.

eResSlaveAddrET0AndSlaveIdETSerachId: Bei mindestens einem Motor / Bei keinem Motor ist die Adresse gleich 0 und die Slave-Id ist ebenfalls gleich der gesuchten Slave-Id [▶ 55] (*dwSlaveId*) / Der Wert ist undefiniert.

eResSlaveAddrNE0: Bei mindestens einem Motor / Bei keinem Motor ist die Adresse ungleich 0 [▶ 56] / Der Wert ist undefiniert.

VAR_IN_OUT

```
stCommandBuffer             : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Beispiele

Die folgenden Tabellen zeigen die Ergenisse des Bauusteins bei unterschiedlichen Ausgangssituationen. In allen Fällen sind zwei SMI-Geräte an einer SMI-Klemme angeschlossen und beide Adressen sind größer 0.

Die gesuchte Slave-Id () liegt zwischen den Slave-Ids der beiden Antriebe: *dwSlaveId*

Ausgänge	Bedeutung
eResSlaveAddrET0AndSlaveIdLTSerachId = eSMIDiagResAtLeastOneSlaveAddrET0AndSlaveIdLTSearchId	Bei mindestens einem Motor ist die Slave-Adresse gleich 0 und die Slave-Id ist kleiner der gesuchten Slave-Id.
eResSlaveAddrET0AndSlaveIdGTSerachId = eSMIDiagResAtLeastOneSlaveAddrET0AndSlaveIdGTSearchId	Bei mindestens einem Motor ist die Slave-Adresse gleich 0 und die Slave-Id ist größer der gesuchten Slave-Id.
eResSlaveAddrET0AndSlaveIdETSerachId = eSMIDiagResNoSlaveAddrET0AndSlaveIdETSearchId	Bei keinem Motor ist die Slave-Adresse gleich 0 und die Slave-Id gleich der gesuchten Slave-Id.
eResSlaveAddrNE0 = eSMIDiagResNoSlaveAddrNE0	Bei keinem Motor ist die Slave-Adresse ungleich 0.

Die gesuchte Slave-Id () ist größer als die Slave-Ids der beiden Antriebe: *dwSlaveId*

Ausgänge	Bedeutung
eResSlaveAddrET0AndSlaveIdLTSerachId = eSMIDiagResAtLeastOneSlaveAddrET0AndSlaveIdLTSearchId	Bei mindestens einem Motor ist die Slave-Adresse gleich 0 und die Slave-Id ist kleiner der gesuchten Slave-Id.
eResSlaveAddrET0AndSlaveIdGTSerachId = eSMIDiagResNoSlaveAddrET0AndSlaveIdGTSearchId	Bei keinem Motor ist die Slave-Adresse gleich 0 und die Slave-Id größer der gesuchten Slave-Id.

Ausgänge	Bedeutung
eResSlaveAddrET0AndSlaveIdETSerachId = eSMIDiagResNoSlaveAddrET0AndSlaveIdLTSearchId	Bei keinem Motor ist die Slave-Adresse gleich 0 und die Slave-Id kleiner der gesuchten Slave-Id.
eResSlaveAddrNE0 = eSMIDiagResNoSlaveAddrNE0	Bei keinem Motor ist die Slave-Adresse ungleich 0.

Die gesuchte Slave-Id () ist kleiner als die Slave-Ids der beiden Antriebe: *dwSlaveId*

Ausgänge	Bedeutung
eResSlaveAddrET0AndSlaveIdLTSerachId = eSMIDiagResNoSlaveAddrET0AndSlaveIdLTSearchId	Bei keinem Motor ist die Slave-Adresse gleich 0 und die Slave-Id kleiner der gesuchten Slave-Id.
eResSlaveAddrET0AndSlaveIdGTSerachId = eSMIDiagResAtLeastOneSlaveAddrET0AndSlaveIdGTSearchId	Bei mindestens einem Motor ist die Slave-Adresse gleich 0 und die Slave-Id größer der gesuchten Slave-Id.
eResSlaveAddrET0AndSlaveIdETSerachId = eSMIDiagResNoSlaveAddrET0AndSlaveIdETSearchId	Bei keinem Motor ist die Slave-Adresse gleich 0 und die Slave-Id gleich der gesuchten Slave-Id.
eResSlaveAddrNE0 = eSMIDiagResNoSlaveAddrNE0	Bei keinem Motor ist die Slave-Adresse ungleich 0.

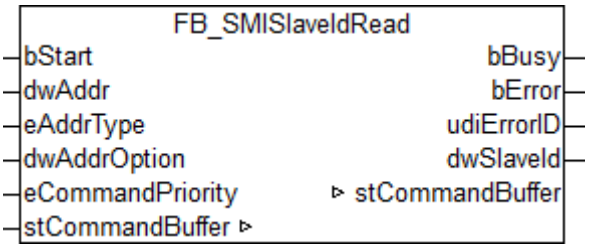
Die gesuchte Slave-Id () ist gleich der Slave-Id eines Antriebes: *dwSlaveId*

Ausgänge	Bedeutung
eResSlaveAddrET0AndSlaveIdLTSerachId = eSMIDiagResNoSlaveAddrET0AndSlaveIdLTSearchId	Bei keinem Motor ist die Slave-Adresse gleich 0 und die Slave-Id kleiner der gesuchten Slave-Id.
eResSlaveAddrET0AndSlaveIdGTSerachId = eSMIDiagResAtLeastOneSlaveAddrET0AndSlaveIdGTSearchId	Bei mindestens einem Motor ist die Slave-Adresse gleich 0 und die Slave-Id größer der gesuchten Slave-Id.
eResSlaveAddrET0AndSlaveIdETSerachId = eSMIDiagResAtLeastOneSlaveAddrET0AndSlaveIdETSearchId	Bei mindestens einem Motor ist die Slave-Adresse gleich 0 und die Slave-Id gleich der gesuchten Slave-Id.
eResSlaveAddrNE0 = eSMIDiagResNoSlaveAddrNE0	Bei keinem Motor ist die Slave-Adresse ungleich 0.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.24 FB_SMISlaveIdRead



Aus einem Antrieb wird die Slave-Id (32 Bit Key-Id) ausgelesen.

VAR_INPUT

```
bStart            : BOOL;
dwAddr            : DWORD := 0;
eAddrType         : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption      : DWORD := 0;
eCommandPriority  : E_SMICommandPriority := eSMICommandPriorityMiddle;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers, zur Gruppenadressierung oder als Slave-Id ausgewertet werden soll.

dwAddrOption: Wird das SMI-Gerät per Slave-Id adressiert (*eAddrType* = *eSMIAddrTypeSlaveId*), so muß über diesen Eingang der Herstellercode [▶ 67] angegeben werden.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

VAR_OUTPUT

```
bBusy         : BOOL;
bError        : BOOL;
udiErrorId    : UDINT;
dwSlaveId     : DWORD;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

dwSlaveId: Die ausgelesene Slave-Id.

VAR_IN_OUT

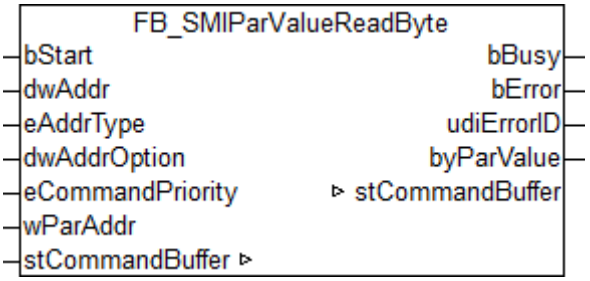
```
stCommandBuffer   : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.25 FB_SMIParValueReadByte



Lesen eines motorseitig gespeicherten Byte-Parameters (1-Byte). Die Bedeutung der einzelnen Parameter ist herstellerabhängig.

VAR_INPUT

```
bStart            : BOOL;
dwAddr            : DWORD := 0;
eAddrType         : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption      : DWORD := 0;
eCommandPriority  : E_SMICommandPriority := eSMICommandPriorityMiddle;
wParAddr          : WORD := 0;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers oder zur Gruppenadressierung ausgewertet werden soll. Eine Adressierung per Slave-Id (*eAddrType* = *eSMIAddrTypeSlaveId*) ist nicht zulässig.

dwAddrOption: reserviert für zukünftige Erweiterungen.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

wParAddr: Adresse des Parameters (0-4095) der gelesen werden soll.

VAR_OUTPUT

```
bBusy         : BOOL;
bError        : BOOL;
udiErrorId    : UDINT;
byParValue    : BYTE;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

byParValue: Der ausgelesene Byte-Parameter.

VAR_IN_OUT

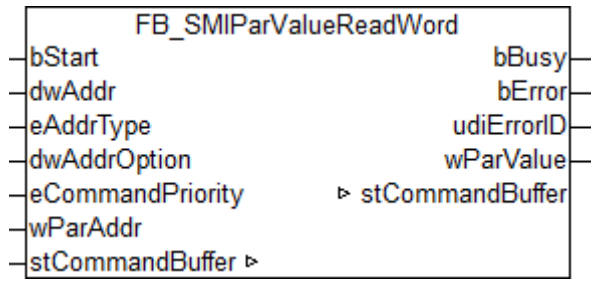
```
stCommandBuffer   : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.26 FB_SMIParValueReadWord



Lesen eines motorseitig gespeicherten Word-Parameters (2-Bytes). Die Bedeutung der einzelnen Parameter ist herstellerabhängig.

VAR_INPUT

```
bStart            : BOOL;
dwAddr            : DWORD := 0;
eAddrType         : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption      : DWORD := 0;
eCommandPriority  : E_SMICommandPriority := eSMICommandPriorityMiddle;
wParAddr          : WORD := 0;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers oder zur Gruppenadressierung ausgewertet werden soll. Eine Adressierung per Slave-Id (*eAddrType* = *eSMIAddrTypeSlaveId*) ist nicht zulässig.

dwAddrOption: reserviert für zukünftige Erweiterungen.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

wParAddr: Adresse des Parameters (0-4095) der gelesen werden soll.

VAR_OUTPUT

```
bBusy         : BOOL;
bError        : BOOL;
udiErrorId    : UDINT;
wParValue     : WORD;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

wParValue: Der ausgelesene Word-Parameter.

VAR_IN_OUT

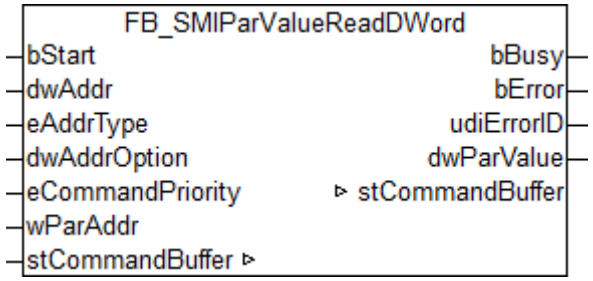
```
stCommandBuffer    : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.1.27 FB_SMIParValueReadDWord



Lesen eines motorseitig gespeicherten DWord-Parameters (4-Bytes). Die Bedeutung der einzelnen Parameter ist herstellerabhängig.

VAR_INPUT

```
bStart             : BOOL;
dwAddr             : DWORD := 0;
eAddrType          : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption       : DWORD := 0;
eCommandPriority   : E_SMICommandPriority := eSMICommandPriorityMiddle;
wParAddr           : WORD := 0;
```

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

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers oder zur Gruppenadressierung ausgewertet werden soll. Eine Adressierung per Slave-Id (*eAddrType* = *eSMIAddrTypeSlaveId*) ist nicht zulässig.

dwAddrOption: reserviert für zukünftige Erweiterungen.

eCommandPriority: Priorität [▶ 55] (hoch, mittel oder niedrig), mit der der Befehl von der SPS-Bibliothek abgearbeitet wird.

wParAddr: Adresse des Parameters (0-4095) der gelesen werden soll.

VAR_OUTPUT

```
bBusy          : BOOL;
bError         : BOOL;
udiErrorId     : UDINT;
dwParValue     : DWORD;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv bis der Befehl abgearbeitet wurde.

bError: Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in *udiErrorId* enthalten. Durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wird der Ausgang wieder auf FALSE zurückgesetzt.

udiErrorId: Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das erneute Aktivieren des Bausteins über den Eingang *bStart* wieder auf 0 zurückgesetzt. Siehe Fehlercodes [▶ 58].

dwParValue: Der ausgelesene DWord-Parameter.

VAR_IN_OUT

```
stCommandBuffer    : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

5.2 Datentypen

5.2.1 E_SMIAddrType

```
TYPE E_SMIAddrType :
(
  eSMIAddrTypeManufacturer  := 0,
  eSMIAddrTypeAddress       := 1,
  eSMIAddrTypeGroup         := 2,
  eSMIAddrTypeSlaveId       := 3,
  eSMIAddrTypeBroadcast     := 4
);
END_TYPE
```

5.2.2 E_SMICommandPriority

```
TYPE E_SMICommandPriority :
(
  eSMICommandPriorityHigh     := 0,
  eSMICommandPriorityMiddle   := 1,
  eSMICommandPriorityLow      := 2
);
END_TYPE
```

5.2.3 E_SMICommandType

```
TYPE E_SMICommandType :
(
  eSMICommandTypeWrite    := 0,
  eSMICommandTypeRead     := 1
);
END_TYPE
```

5.2.4 E_SMICompResSlaveAddrET0AndSlaveIdETSearchId

```
TYPE E_SMICompResSlaveAddrET0AndSlaveIdETSearchId :
(
  eSMIDiagResSlaveAddrET0AndSlaveIdETSearchIdUndefined    := 0,
  eSMIDiagResNoSlaveAddrET0AndSlaveIdETSearchId           := 1,
  eSMIDiagResAtLeastOneSlaveAddrET0AndSlaveIdETSearchId   := 2
);
END_TYPE
```

5.2.5 E_SMICompResSlaveAddrET0AndSlaveIdGTSearchId

```
TYPE E_SMICompResSlaveAddrET0AndSlaveIdGTSearchId :
(
  eSMIDiagResSlaveAddrET0AndSlaveIdGTSearchIdUndefined    := 0,
  eSMIDiagResNoSlaveAddrET0AndSlaveIdGTSearchId           := 1,
  eSMIDiagResAtLeastOneSlaveAddrET0AndSlaveIdGTSearchId   := 2
);
END_TYPE
```

5.2.6 E_SMICompResSlaveAddrET0AndSlaveIdLTSearchId

```
TYPE E_SMICompResSlaveAddrET0AndSlaveIdLTSearchId :
(
  eSMIDiagResSlaveAddrET0AndSlaveIdLTSearchIdUndefined    := 0,
  eSMIDiagResNoSlaveAddrET0AndSlaveIdLTSearchId           := 1,
  eSMIDiagResAtLeastOneSlaveAddrET0AndSlaveIdLTSearchId   := 2
);
END_TYPE
```

5.2.7 E_SMICompResSlaveAddrNE0

```
TYPE E_SMICompResSlaveAddrNE0 :
(
  eSMIDiagResSlaveAddrNE0Undefined    := 0,
  eSMIDiagResNoSlaveAddrNE0           := 1,
  eSMIDiagResAtLeastOneSlaveAddrNE0   := 2
);
END_TYPE
```

5.2.8 E_SMIConfigurationCommands

```
TYPE E_SMIConfigurationCommands :
(
  eSMICommandDoNothing   := 0,
  eSMICommandUp          := 1,
  eSMICommandDown        := 2,
  eSMICommandStop        := 3,
  eSMICommandPos1        := 4,
  eSMICommandPos2        := 5
);
END_TYPE
```

5.2.9 E_SMIDiagResDrivesDown

```
TYPE E_SMIDiagResDrivesDown :
(
  eSMIDiagResDrivesDownUndefined        := 0,
  eSMIDiagResNoMotorDrivesDown          := 1,
  eSMIDiagResAtLeastOneMotorDrivesDown  := 2
);
END_TYPE
```

5.2.10 E_SMIDiagResDrivesUp

```
TYPE E_SMIDiagResDrivesUp :
(
  eSMIDiagResDrivesUpUndefined        := 0,
  eSMIDiagResNoMotorDrivesUp          := 1,
  eSMIDiagResAtLeastOneMotorDrivesUp  := 2
);
END_TYPE
```

5.2.11 E_SMIDiagResIsStopped

```
TYPE E_SMIDiagResIsStopped :
(
  eSMIDiagResIsStoppedUndefined       := 0,
  eSMIDiagResNoMotorIsStopped         := 1,
  eSMIDiagResAtLeastOneMotorIsStopped := 2
);
END_TYPE
```

5.2.12 E_SMIDiagResponse

```
TYPE E_SMIDiagResponse :
(
  eSMIDiagResponseDiscard    := 0,
  eSMIDiagResponseAck        := 1,
  eSMIDiagResponseNack       := 2
);
END_TYPE
```

5.2.13 E_SMIDiagResWithError

```
TYPE E_SMIDiagResWithError :
(
  eSMIDiagResWithErrorUndefined       := 0,
  eSMIDiagResNoMotorWithError         := 1,
  eSMIDiagResAtLeastOneMotorWithError := 2
);
END_TYPE
```

5.2.14 E_SMIResponseFormat

```
TYPE E_SMIResponseFormat :
(
  eSMIResponseFormatDiagnosis    := 0,
  eSMIResponseFormatStandard     := 1
);
END_TYPE
```

5.2.15 ST_KL6831KL6841InData

```
TYPE ST_KL6831KL6841InData :
STRUCT
  wStateWord   : WORD;
  arrData      : ARRAY [0..21] OF BYTE;
END_STRUCT
END_TYPE
```

5.2.16 ST_KL6831KL6841OutData

```
TYPE ST_KL6831KL6841OutData :
STRUCT
  wControlWord    : WORD;
  arrData     : ARRAY [0..21] OF BYTE;
END_STRUCT
END_TYPE
```

5.2.17 ST_SMICommandBuffer

```
TYPE ST_SMICommandBuffer :
STRUCT
  arrMessageQueue      : ARRAY [0..2] OF ST_SMIMessageQueue;
  stResponseTable      : ST_SMIResponseTable;
  udiMessageHandle     : UDINT;
END_STRUCT
END_TYPE
```

Sehen Sie dazu auch

ST_SMIMessageQueue [▶ 57]

5.2.18 ST_SMIMessageQueue

```
TYPE ST_SMIMessageQueue :
STRUCT
  arrBuffer                    : ARRAY [1..SMI_COMMAND_BUFFER_ENTRIES] OF ST_SMIMessageQueueItem;
  byBufferReadPointer          : BYTE;
  byBufferWritePointer         : BYTE;
  byBufferDemandCounter        : BYTE;
  byBufferMaximumDemandCounter : BYTE;
  uiBufferOverflowCounter      : UINT;
```

```
  bLockSemaphore               : BOOL;
END_STRUCT
END_TYPE
```

Sehen Sie dazu auch

- ST_SMIMessageQueueItem [▶ 58]

5.2.19 ST_SMIMessageQueueItem

```
TYPE ST_SMIMessageQueueItem :
STRUCT
  dwAddr                    : DWORD;
  eAddrType                 : E_SMIAddrType;
  eCommandType              : E_SMICommandType;
  eResponseFormat           : E_SMIResponseFormat;
  arrIdentificationBytes    : ARRAY [0..2] OF BYTE;
  arrParameters             : ARRAY [0..2] OF DWORD;
  udiMessageHandle          : UDINT;
  bSuppressResponseBuffer   : BOOL;
END_STRUCT
END_TYPE
```

Sehen Sie dazu auch

- E_SMIAddrType [▶ 55]
- E_SMICommandType [▶ 55]
- E_SMIResponseFormat [▶ 57]

5.2.20 ST_SMIResponseTable

```
TYPE ST_SMIResponseTable :
STRUCT
  arrResponseTable                : ARRAY [1..SMI_COMMAND_BUFFER_ENTRIES] OF ST_SMIResponseTableItem
;
  byResponseTableCounter          : BYTE;
  byResponseTableMaxCounter       : BYTE;
  uiResponseTableOverflowCounter  : UINT;
  bLockSemaphore                  : BOOL;
END_STRUCT
END_TYPE
```

Sehen Sie dazu auch

- ST_SMIResponseTableItem [▶ 58]

5.2.21 ST_SMIResponseTableItem

```
TYPE ST_SMIResponseTableItem :
STRUCT
  arrResponseData       : ARRAY [0..7] OF BYTE;
  byDataLength          : BYTE;
  byIdentificationByte  : BYTE;
  udiMessageHandle      : UDINT;
  udiErrorId            : UDINT;
END_STRUCT
END_TYPE
```

5.3 Fehlercodes

Wert (hex)	Wert (dez)	Beschreibung
0x0000	0	Kein Fehler.
0x8001	32769	Keine Rückantwort vom SMI-Antrieb.
0x8002	32770	Keine Klemmenrückmeldung für die Sendedaten von der SMI-Klemme.

Wert (hex)	Wert (dez)	Beschreibung
0x8003	32771	Klemme hat ein Telegrammfehler erkannt (StatusWord.6 = true). Diese Meldung muß durch den Eingang *bResetDataFrameError* vom FB_KL6831KL6841Communication() [▶ 19] quittiert werden.
0x8004	32772	NACK vom Antrieb empfangen.
0x8005	32773	Ungültige Rückmeldung vom Antrieb empfangen.
0x8006	32774	Überlauf Kommunikationspuffer.
0x8007	32775	Keine Antwort vom Kommunikationsbaustein.
0x8008	32776	Die Konstante *SMI_COMMAND_BUFFER_ENTRIES* liegt außerhalb des gültigen Bereichs (2-250).
0x8009	32777	Das empfangene Id Byte ist nicht korrekt.
0x800A	32778	Die empfangene Datenlänge ist nicht korrekt.
0x800B	32779	24 V DC Versorgungsspannung an der KL6831/KL6841 fehlt (StatusWord.2 = false).
0x800C	32780	Prozeßabbild wurde durch die Eingänge Switch1 oder Switch2 der Klemme deaktiviert (StatusWord.5 = true). Diese Meldung muß durch den Eingang *bResetInactiveProcessImage* vom FB_KL6831KL6841Communication() [▶ 19] quittiert werden.
0x800D	32781	Klemme hat einen Checksummenfehler erkannt (StatusWord.8 = true). Sobald ein Telegramm erfolgreich übertragen wurde, wird diese Meldung wieder zurückgesetzt.
0x800E	32782	Der SMI-Befehl unterstützt nicht die Adressierung per Slave-Id *(eAddrType = eSMIAddrTypeSlaveId).*
0x800F	32783	Parameter *dwAddr* (Bitfeld für Gruppenadressierung) ist außerhalb des gültigen Bereichs (0-65535).
0x8010	32784	Parameter *dwAddr* (Adresse) ist außerhalb des gültigen Bereichs (0-15).
0x8011	32785	Parameter *eCommandPriority* ist ungültig.
0x8012	32786	Parameter *eCommandType* ist ungültig.
0x8013	32787	Parameter *uiAngle* ist außerhalb des gültigen Bereichs (0-510).
0x8014	32788	Parameter *wParAddr* ist außerhalb des gültigen Bereichs (0-4095).
0x8015	32789	Parameter *eAddrType* ist ungültig.
0x8016	32790	Parameter *eResponseFormat* ist ungültig.
0x8017	32791	Parameter *dwAddr* (Herstellercode) ist außerhalb des gültigen Bereichs (0-15).
0x8018	32792	Das Kommando unterstützt nur Einzeladressierung.
0x8019	32793	Parameter-Option *dwAddrOption* (Herstellercode) ist außerhalb des gültigen Bereichs (0-15).
0x801A	32794	Interner Fehler im Baustein FB_SMIDiscoverySlaveId() [▶ 44] aufgetreten.
0x801B	32795	Es wurden keine Geräte gefunden.
0x801C	32796	Alle 16 Adressen wurden schon vergeben. Evtl sind mehr als 16 Geräte am SMI-Bus angeschlossen.
0x801D	32797	Ungültige Diagnoseantwort erhalten (weder NACK noch ACK).
0x801E	32798	Parameter *byHighestAddress* (höchste Adresse) ist außerhalb des gültigen Bereichs (0-15).
0x801F	32799	FB_KL6831KL6841Config() [▶ 21]: Während der Konfigurierung der Klemme ist ein Fehler aufgetreten.
0x8020	32800	FB_KL6831KL6841Config() [▶ 21]: Parameter *eCommandKBusWatchdog* liegt außerhalb des gültigen Bereichs.
0x8021	32801	FB_KL6831KL6841Config() [▶ 21]: Parameter *eCommandDI1RisingEdge* liegt außerhalb des gültigen Bereichs.
0x8022	32802	FB_KL6831KL6841Config() [▶ 21]: Parameter *eCommandDI1FallingEdge* liegt außerhalb des gültigen Bereichs.
0x8023	32803	FB_KL6831KL6841Config() [▶ 21]: Parameter *eCommandDI2RisingEdge* liegt außerhalb des gültigen Bereichs.

Wert (hex)	Wert (dez)	Beschreibung
0x8024	32804	FB_KL6831KL6841Config() [▶ 21]: Parameter *eCommandDI2FallingEdge* liegt außerhalb des gültigen Bereichs.

6 Anhang

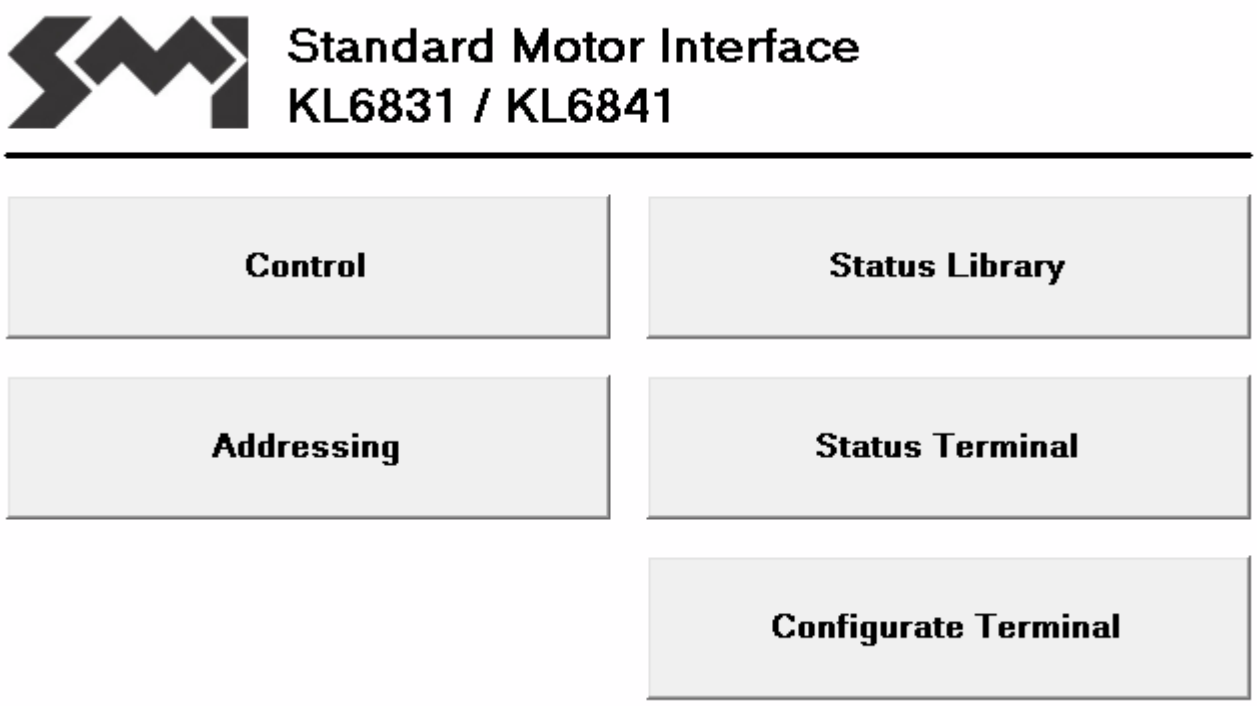
6.1 Beispiel: Konfigurieren von SMI-Geräten

Voraussetzungen

Zielsystem mit TwinCAT Projektdateien
https://infosys.beckhoff.com/content/1031/tcplclibsmi/Resources/12074358283.zip
https://infosys.beckhoff.com/content/1031/tcplclibsmi/Resources/12074359691.zip
https://infosys.beckhoff.com/content/1031/tcplclibsmi/Resources/12074361099.zip
https://infosys.beckhoff.com/content/1031/tcplclibsmi/Resources/12074362507.zip
https://infosys.beckhoff.com/content/1031/tcplclibsmi/Resources/12074363915.zip

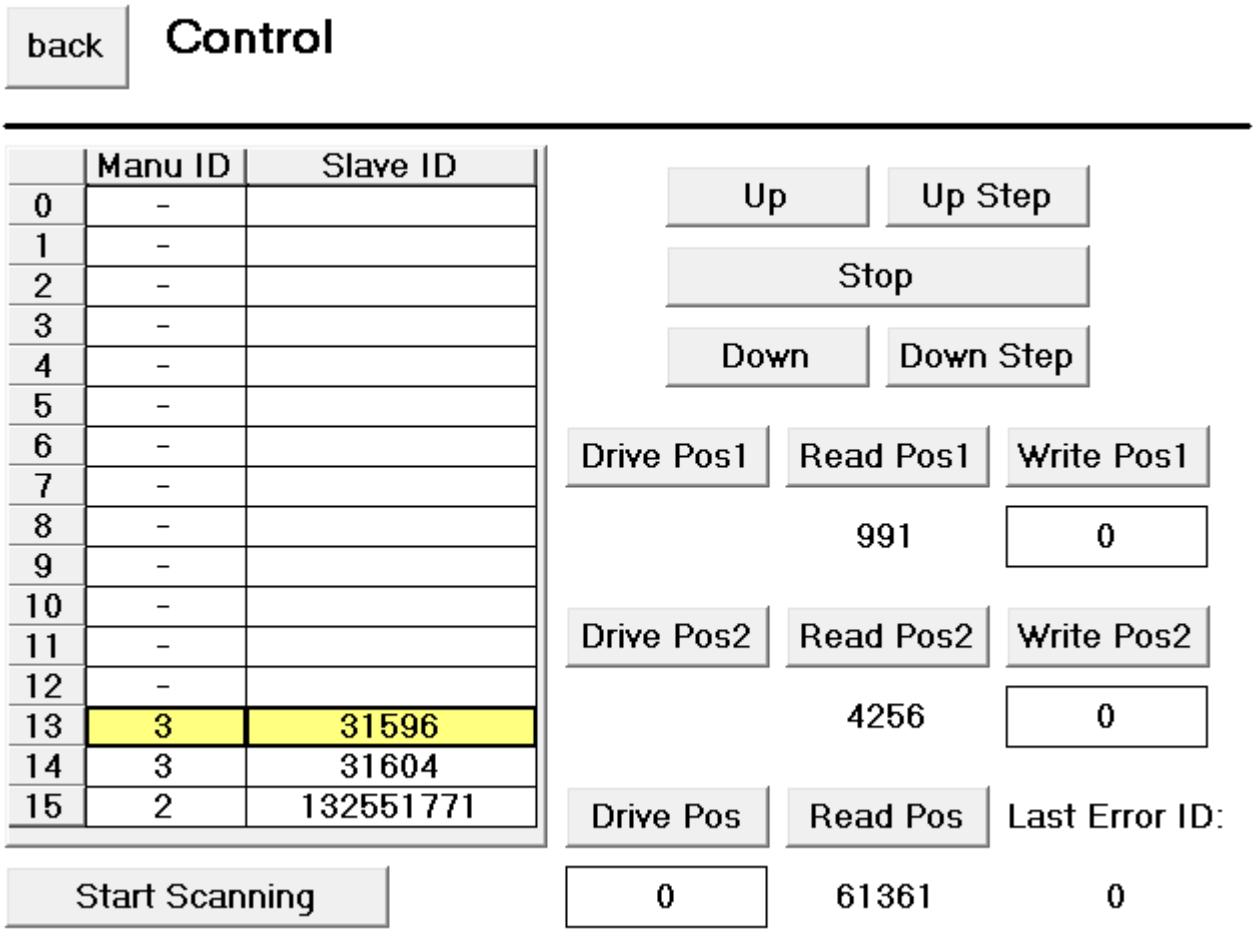
Mit dem Beispiel ist es möglich SMI-Geräte zu adressieren oder eine vorhandene Installation zu erweitern. Des Weiteren können die Dialoge zur Diagnose und Fehleranalyse genutzt werden. Insgesamt stehen 5 Dialoge zur Verfügung.

Start



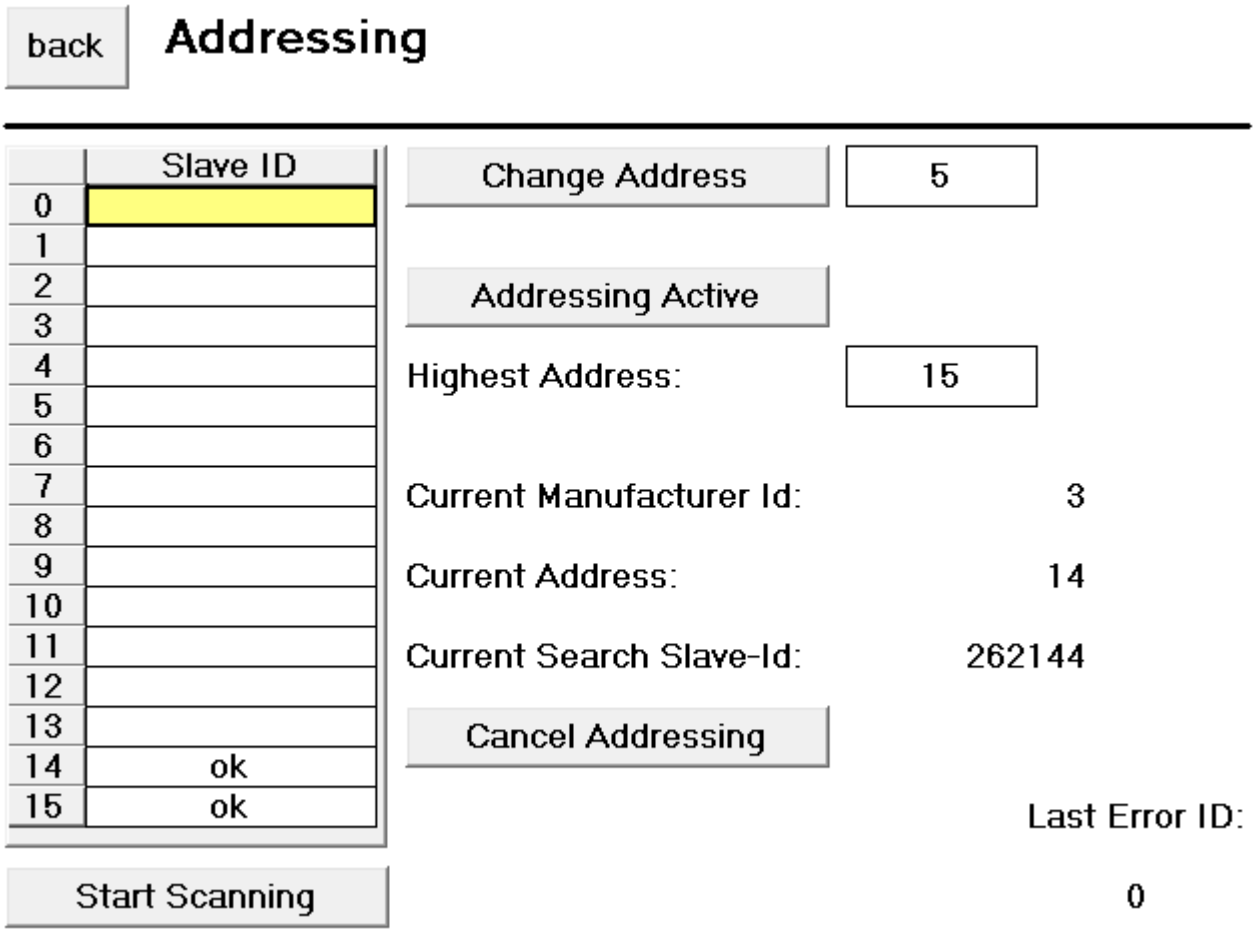
Unter *SMI_Start* befindet sich das Hauptmenü, über das die vier Untermenüs erreichbar sind.

Control



Mit der Schaltfläche *Start Scanning* wird an der SMI-Linie nach adressierten SMI-Geräten gesucht. Alle gefundenen SMI-Geräte werden in der linken Liste durch den Herstellercode [▶ 67] und der Slave-Id angezeigt. Durch Anklicken eines Eintrags wird dieser selektiert. Die anderen Schaltflächen beziehen sich immer auf den selektierten Eintrag. Hierdurch können alle wichtigen SMI-Kommandos an jedes adressierte SMI-Gerät versendet werden. Sollte bei einem SMI-Kommando ein Fehler erkannt werden, so wird dieser in der rechten unteren Ecke durch den Fehlercode [▶ 58] angezeigt.

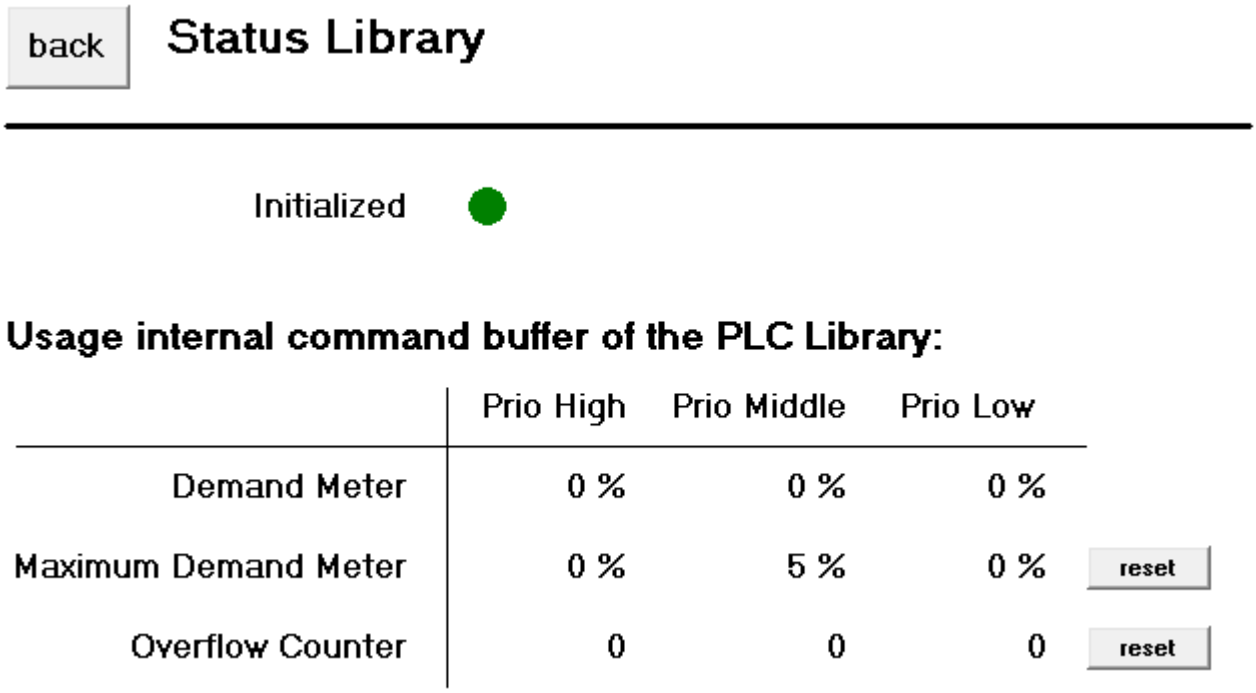
Addressing



Jedem SMI-Gerät kann eine Adresse von 0 bis 15 zugewiesen werden. Über diese Adresse kann jedes SMI-Gerät angesprochen werden. Es gibt zwar noch andere Methoden SMI-Geräte anzusprechen (siehe hier [▶ 9]), jedoch ist für die Gruppenadressierung eine eindeutige Adresse notwendig. Von daher empfiehlt es sich, jedem SMI-Gerät eine Adresse zuzuweisen. Die Schaltfläche *Start Scanning* dient zum Suchen aller adressierten SMI-Geräte. Soll eine Adresse geändert werden, so muss das entsprechende SMI-Gerät in der Liste selektiert werden. Über das Eingabefeld rechts neben der Schaltfläche *Change Address* kann die gewünschte Adresse vorgegeben werden, die durch das Betätigen der Schaltfläche übernommen wird.

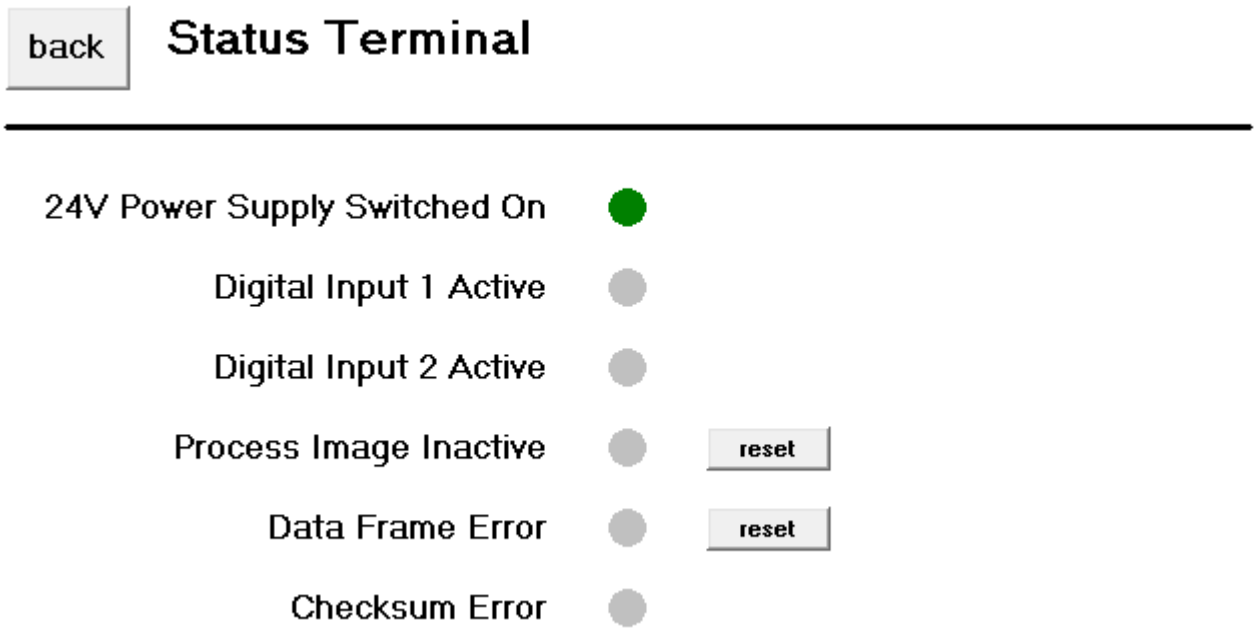
Besitzen SMI-Geräte noch keine Adresse, so bekommen alle SMI-Geräte durch das Betätigen der Schaltfläche *Start Addressing* eine eindeutige Adresse. Der Anwender hat keinen Einfluss darauf, welches SMI-Gerät welche Adresse zugewiesen bekommt. Die Vergabe der Adressen erfolgt absteigend, beginnend bei der Adresse, die durch den Parameter *Highest Address* vorgegeben wird. Die Felder *Current Manufacturer Id*, *Current Address* und *Current Search Slave-Id* geben Auskunft über den Status der Adressierung. Durch *Cancel Addressing* kann die Adressierung vorzeitig abgebrochen werden.

Status Library



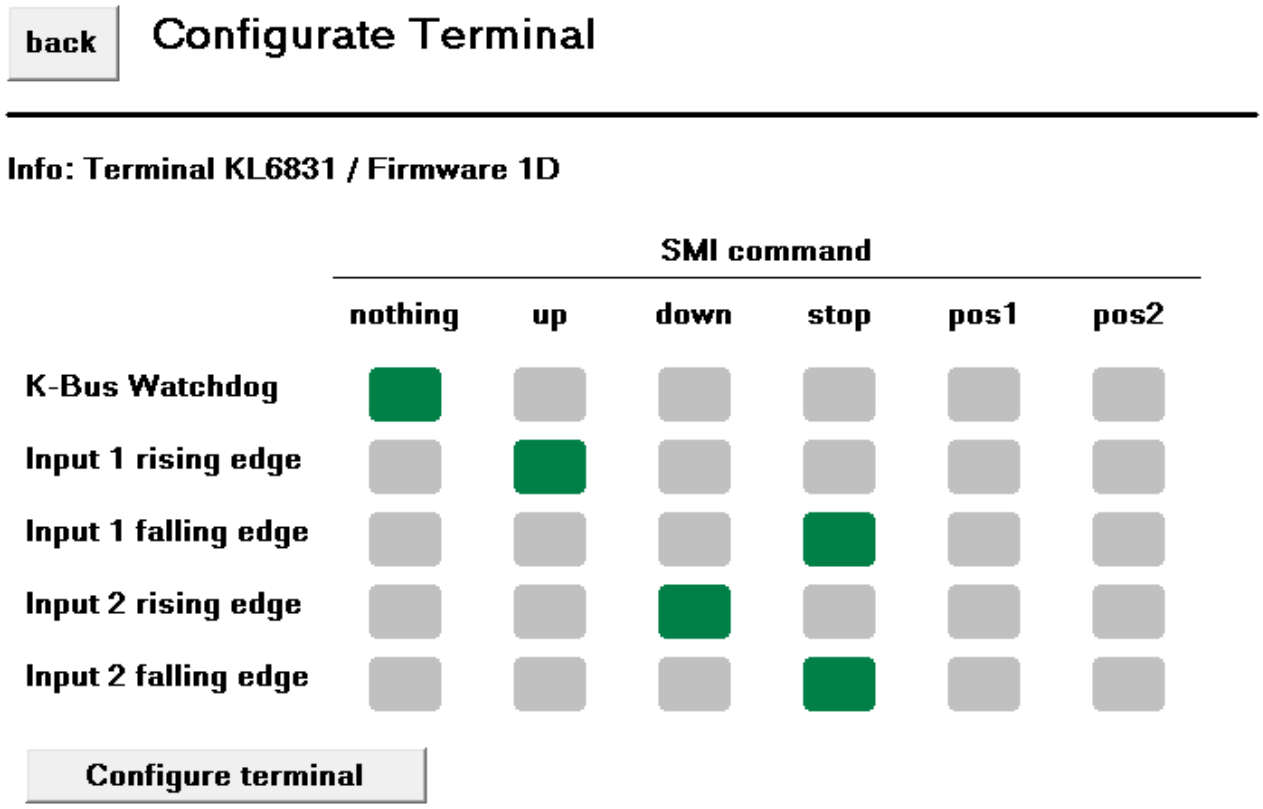
Innerhalb der SPS-Bibliothek erfolgt die Kommunikation zwischen den einzelnen SPS-Bausteinen und der Busklemme über drei zentrale Puffer (je SMI-Klemme). Die Auslastung der Puffer und evtl. auftretende Überläufe können in der abgebildeten Tabelle ermittelt werden.

Status Terminal



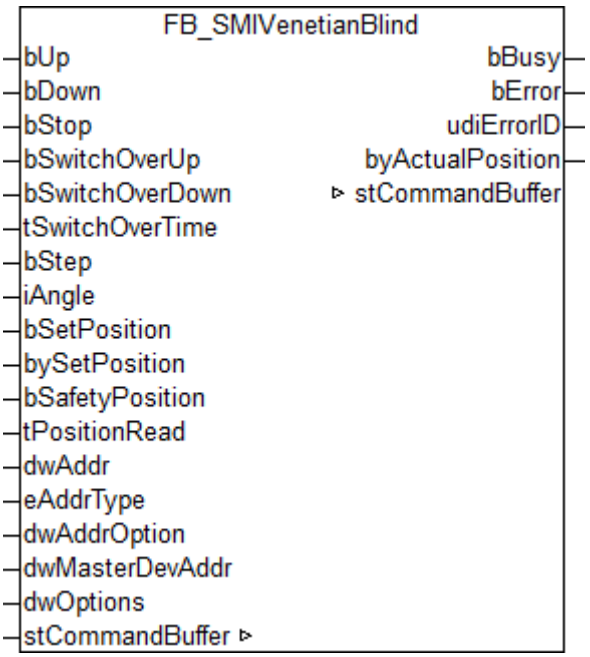
Die Statusinformationen aus dem Prozessabbild der Klemme werden in diesem Dialog angezeigt. Auch lassen sich die quittierungspflichtigen Meldungen in diesen Dialog wieder zurücksetzen.

Configurate Terminal



Über zwei digitale Eingänge der SMI-Klemme können SMI-Befehle versendet werden. Welche SMI-Befehle mit der steigenden und der fallenden Flanke ausgelöst werden, lässt sich in diesen Dialog festlegen. Des Weiteren kann ein SMI-Befehl versendet werden, wenn auf dem K-Bus der Klemme keine Kommunikation mehr vorhanden ist (z.B. SPS ist gestoppt oder Verbindung zum Feldbus-Master ist unterbrochen).

6.2 Beispiel: FB_SMIVenetianBlind



Jalousiesteuerung.

https://infosys.beckhoff.com/content/1031/tcplclibsmi/Resources/12074365323.zip

VAR_INPUT

```
bUp                 : BOOL;
bDown               : BOOL;
bStop               : BOOL;
bSwitchOverUp       : BOOL;
bSwitchOverDown     : BOOL;
tSwitchOverTime     : TIME := t#400ms;
bStep               : BOOL;
iAngle              : INT := 0;
bSetPosition        : BOOL;
bySetPosition       : BYTE := 0;
bSafetyPosition     : BOOL;
tPositionRead       : TIME := t#1s;
dwAddr              : DWORD := 0;
eAddrType           : E_SMIAddrType := eSMIAddrTypeAddress;
dwAddrOption        : DWORD := 0;
dwMasterDevAddr     : DWORD := 0;
dwOptions           : DWORD := 0;
```

bUp: Fährt die Jalousie hoch. Beginnt im Schleichgang und erhöht dann die Geschwindigkeit.

bDown: Fährt die Jalousie runter. Beginnt im Schleichgang und erhöht dann die Geschwindigkeit.

bStop: Stoppt die Jalousie.

bSwitchOverUp: Fährt die Jalousie schrittweise hoch. Liegt das Signal länger an als *tSwitchOverTime*, geht die Jalousie in Selbsthaltung.

bSwitchOverDown: Fährt die Jalousie schrittweise runter. Liegt das Signal länger an als *tSwitchOverTime*, geht die Jalousie in Selbsthaltung.

tSwitchOverTime: Gibt an, wie lange *bSwitchOverUp* und *bSwitchOverDown* anliegen müssen, bevor die Jalousie in Selbsthaltung geht. Wenn der Wert 0 beträgt, geht die Jalousie sofort in Selbsthaltung.

bStep: Fährt die Jalousie an die angegebene Position.

iAngle: Winkel (-510° ... +510°) auf den die Jalousie gefahren wird, nachdem eine steigende Flanke am Eingang *bStep* erkannt wurde. Negative Werte entsprechen hochfahren, positive Werte runterfahren.

bSetPosition: Fährt die Jalousie an die unter *bySetPosition* angegebene Position.

bySetPosition: Position in Prozent, zu der die Jalousie gefahren wird, nachdem eine steigende Flanke am Eingang *bSetPosition* erkannt wurde. 0 % entspricht vollständig eingefahren, 100 % vollständig ausgefahren.

bSafetyPosition: Die Sicherheitsposition wird angefahren. Es ist nicht möglich die Jalousie zu bedienen während dieser Eingang gesetzt ist.

tPositionRead: Intervall, in dem die aktuelle Position vom SMI-Motor ausgelesen wird, wenn keine Positionierungsbefehle verarbeitet werden.

dwAddr: Herstellercode [▶ 67] (0-15), Adresse eines Teilnehmers (0-15), Bitfeld (16 Bit) für die Gruppenadressierung oder Slave-Id (32 Bit Key-Id). Wird ein Sammelruf (Broadcast) versendet, so hat dieser Eingang keine Bedeutung.

eAddrType: Legt fest, ob der Eingang *dwAddr* als Herstellercode [▶ 67], Adresse [▶ 55] eines Teilnehmers, zur Gruppenadressierung oder als Slave-Id ausgewertet werden soll.

dwAddrOption: Wird das SMI-Gerät per Slave-Id adressiert (*eAddrType* = *eSMIAddrTypeSlaveId*), so muss über diesen Eingang der Herstellercode [▶ 67] angegeben werden.

dwMasterDevAddr: Adresse (0-15) vom SMI-Motor, von dem die aktuelle Position ausgelesen werden soll, wenn *eAddrType* != *eSMIAddrTypeAddress* ist.

dwOptions: Reserviert für zukünftige Entwicklungen.

VAR_OUTPUT

```
bBusy               : BOOL;
bError              : BOOL;
udiErrorId          : UDINT;
byActualPosition    : BYTE;
```

bBusy: Bei der Aktivierung des Bausteins wird der Ausgang gesetzt und bleibt so lange aktiv, bis der Befehl abgearbeitet wurde.

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

udiErrorId: Enthält den befehlsspezifischen Fehlercode [▶ 58] des zuletzt ausgeführten Befehls.

byActualPosition: Die ausgelesene Position in Prozent. 0 % entspricht der oberen Endlage und 100 % der unteren Endlage.

VAR_IN_OUT

```
stCommandBuffer     : ST_SMICommandBuffer;
```

stCommandBuffer: Verweis auf die Struktur [▶ 57] zur Kommunikation (Puffer) mit dem FB_KL6831KL6841Communication() [▶ 19]-Baustein.

Voraussetzungen

Entwicklungsumgebung	Zielsystem	erforderliche Bibliotheken
TwinCAT 2.11 R3/x64 ab Build 2238	PC/CX, BX oder BC	TcSMI-Bibliothek ab V1.0.0

6.3 Herstellercodes

Wert (hex)	Wert (dez)	Beschreibung
0x00	0	alle Hersteller
0x01	1	Fa. Dunkermotoren GmbH
0x02	2	Fa. Becker Antriebe GmbH
0x03	3	Fa. elero GmbH
0x04	4	Fa. Selve GmbH & Co. KG
0x05	5	unbenutzt
0x06	6	Fa. Vestamatic GmbH
0x07	7	Fa. WAREMA Renkhoff GmbH
0x08	8	Fa. Groeninger Antriebstechnik GmbH
0x09	9	Fa. Gerhard Geiger GmbH & Co. KG
0x0A	10	Fa. Griesser AG
0x0B	11	unbenutzt
0x0C	12	unbenutzt
0x0D	13	unbenutzt
0x0E	14	unbenutzt
0x0F	15	reserviert für Erweiterungen

6.4 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.

Downloadfinder

Unser Downloadfinder beinhaltet alle Dateien, die wir Ihnen zum Herunterladen anbieten. Sie finden dort Applikationsberichte, technische Dokumentationen, technische Zeichnungen, Konfigurationsdateien und vieles mehr.

Die Downloads sind in verschiedenen Formaten erhältlich.

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 unserer Internetseite: www.beckhoff.com

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 5246 963-157
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 5246 963-460
E-Mail:	service@beckhoff.com

Beckhoff Unternehmenszentrale

Beckhoff Automation GmbH & Co. KG

Hülshorstweg 20
33415 Verl
Deutschland

Telefon:	+49 5246 963-0
E-Mail:	info@beckhoff.com
Internet:	www.beckhoff.com

Mehr Informationen:
www.beckhoff.de/tx1200

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