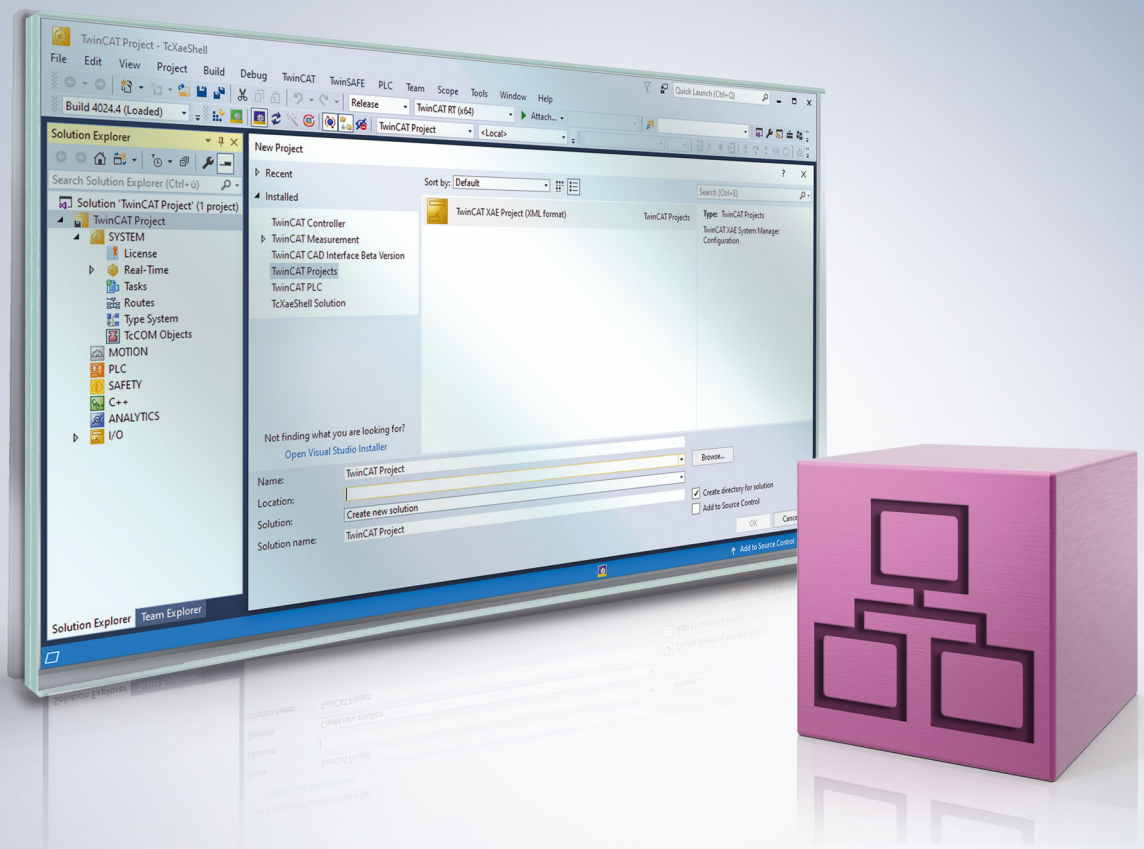
BECKHOFF New Automation Technology

Handbuch | DE

TF6255

TwinCAT 3 | Modbus RTU



07.05.2024 | Version: 1.4.1

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.

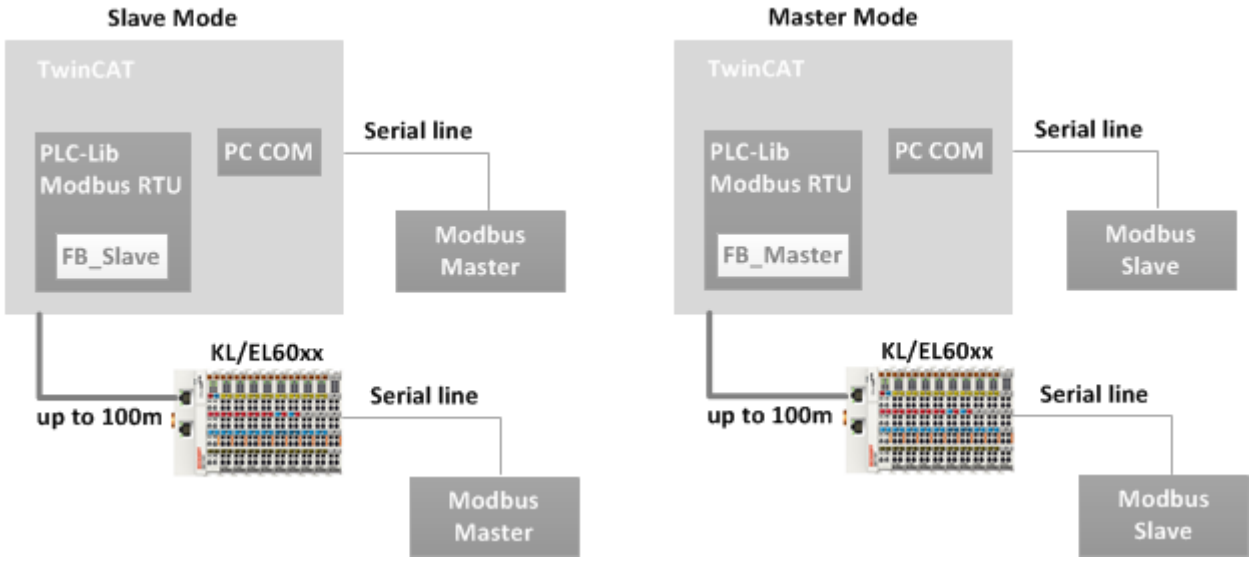
Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem https://www.beckhoff.de/secguide.

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 Übersicht

TwinCAT 3 Modbus RTU bietet Funktionsbausteine zur seriellen Kommunikation mit Modbus-Endgeräten.



Modbus RTU Geräte werden per serieller Schnittstelle mit einem Beckhoff Controller verbunden. Die TwinCAT SPS verwendet Slave Funktionsbausteine der Modbus RTU Bibliothek um mit meinem Modbus Master zu kommunizieren (Slave Mode). Zusätzlich sind Master Funktionsbausteine verfügbar, um mehrere Modbus Slaves anzusprechen (Master Mode)

Unterstützte Schnittstellen

- Serieller COM-Port eines PC oder CX
- Serielle Busklemmen KL60xx
- Serielle EtherCAT-Klemmen EL60xx
- Virtueller serieller COM-Port (USB-Port) eines PC oder CX
  - Bei zusätzlicher Verwendung (und Lizenzierung) von TF6340 TC3 Serial Communication

Weiterführende Dokumentation

Technische Details und Spezifikationen über Modbus finden Sie unter: http://www.modbus.org

Randbedingungen

Das Modbus-Protokoll definiert ein exaktes Timing, so sollen z. B. alle Zeichen eines Telegramms lückenlos übertragen werden. Da die Kommunikation Modbus RTU auf einer SPS-Steuerung realisiert wird, kann wegen der zyklischen Abarbeitung des SPS-Programms, dieses exakte Timing nicht garantiert werden. Die meisten Endgeräte sind sehr tolerant und verhalten sich problemlos, falls kurze Zeitlücken zwischen den Zeichen auftreten. Im Einzelfall muss das Verhalten des Endgerätes überprüft werden.

Bei einer EL60x2 ist der zweite Kanal nicht für ModbusRTU-Kommunikation geeignet, da dieser Low Prior bearbeitet wird und dadurch die Frames mit Lücken versendet, was wiederum die Gegenstelle als Frame Error detektieren könnte.

Bei einigen seriellen Schnittstellen Klemmen kann ein Interner Buffer vor dem Senden gefüllt werden (Option *kontinuierliches Senden*). Die Bibliothek ModbusRTU kann diese Funktion verwenden, wenn dies in der entsprechenden seriellen Klemme eingestellt ist. Zum Beispiel kann bei der KL6031 mit dem Konfigurationsbaustein *KL6configuration* der ContinuousMode aktiviert werden (Register 34 Bit 6). Damit werden dann bis zu 128 Byte in den internen Buffer der Busklemme gelegt und kontinuierlich gesendet.

3 Installation

3.1 Systemvoraussetzungen

Technische Daten	TF6255 TC3 Modbus-RTU
Zielsystem	Windows XP / 7 / 10 PC or CX (x86, x64, ARM)
Min. TwinCAT-Version	3.1.0
Min. TwinCAT-Level	TC1200 TC3 PLC

3.2 Installation

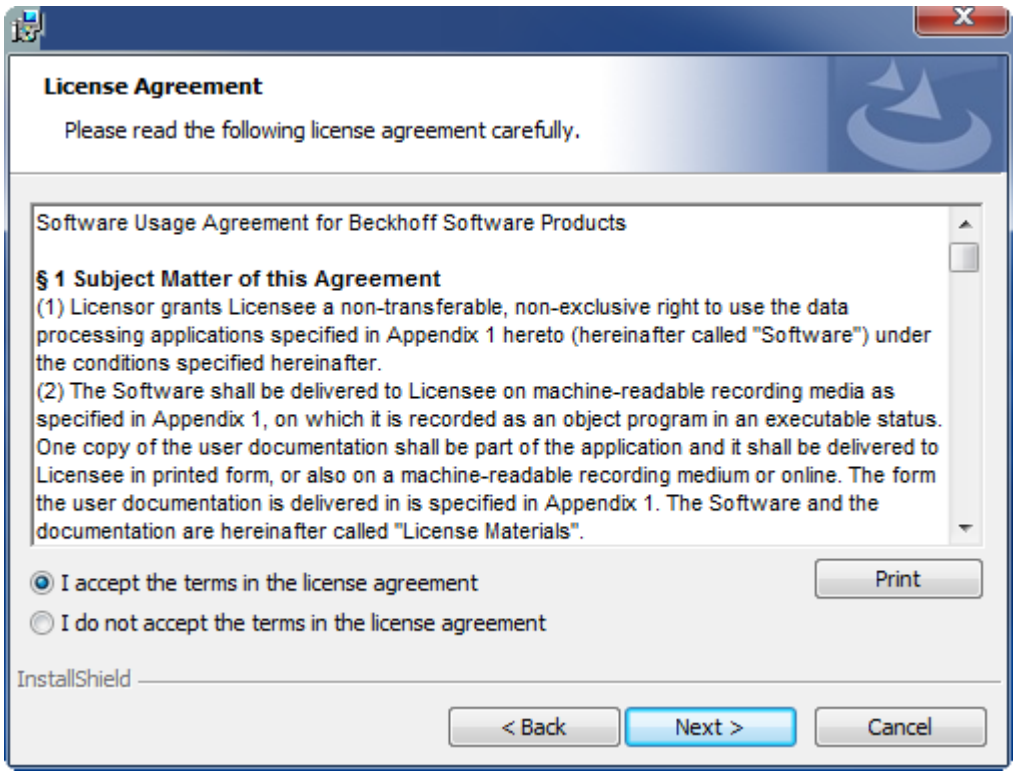
Nachfolgend wird beschrieben, wie die TwinCAT 3 Function für Windows-basierte Betriebssysteme installiert wird.

✓ Die Setup-Datei der TwinCAT 3 Function wurde von der Beckhoff-Homepage heruntergeladen.

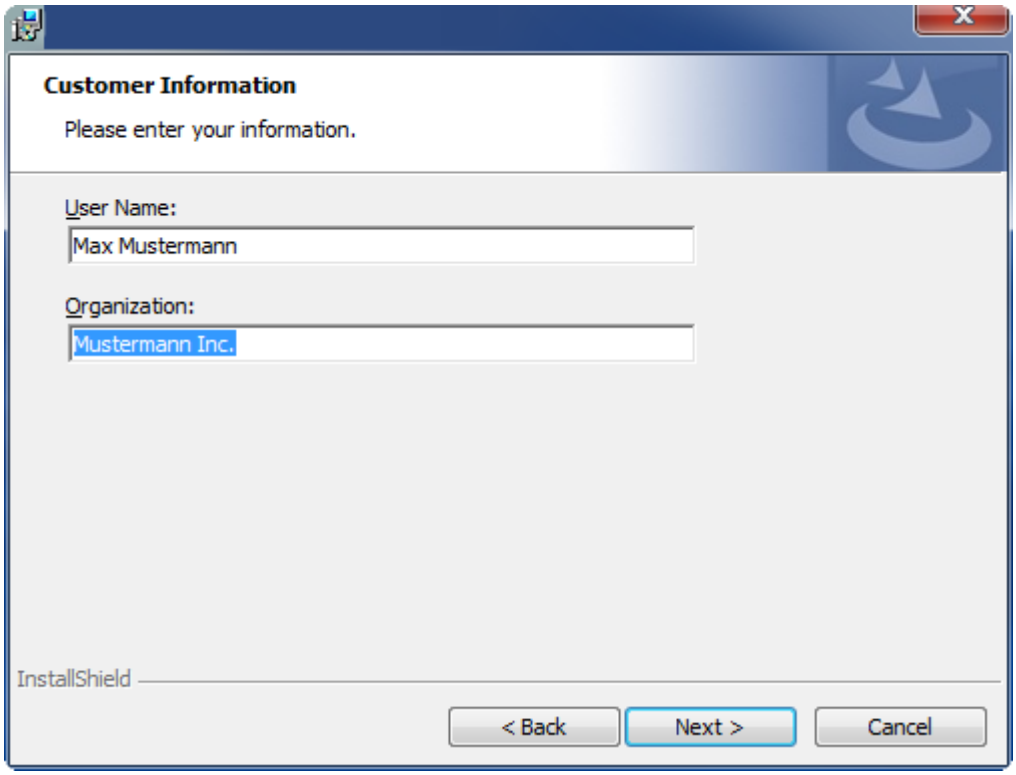
1. Führen Sie die Setup-Datei als Administrator aus. Wählen Sie dazu im Kontextmenü der Datei den Befehl **Als Administrator ausführen**.

   ⇨ Der Installationsdialog öffnet sich.

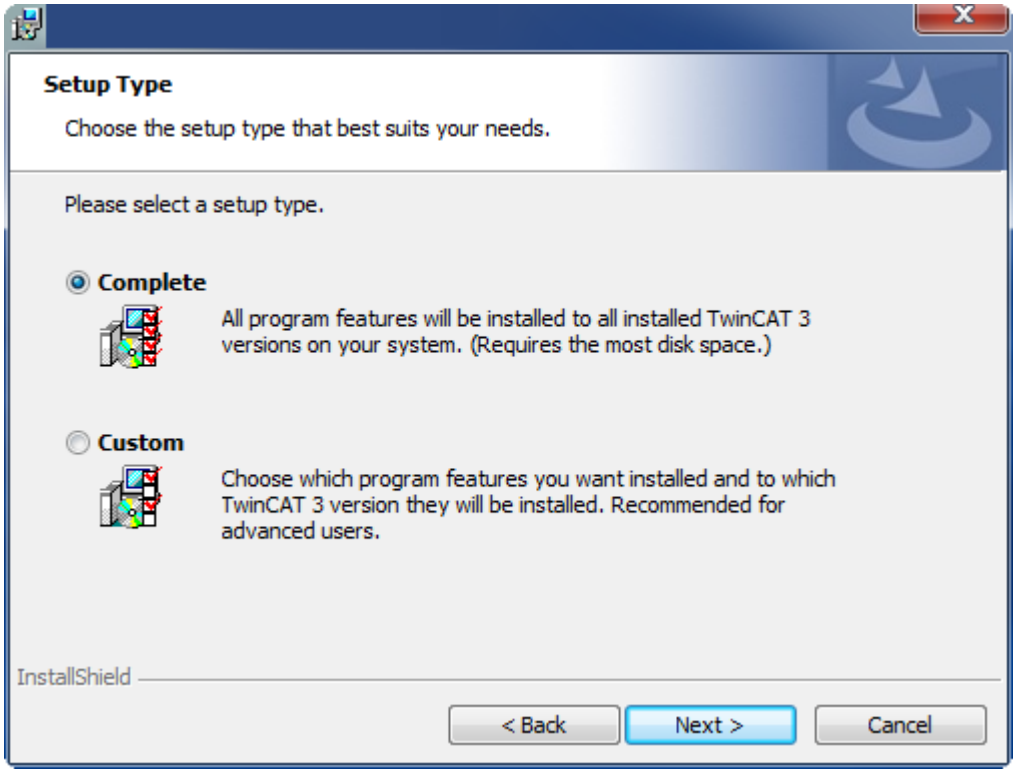
2. Akzeptieren Sie die Endbenutzerbedingungen und klicken Sie auf **Next**.



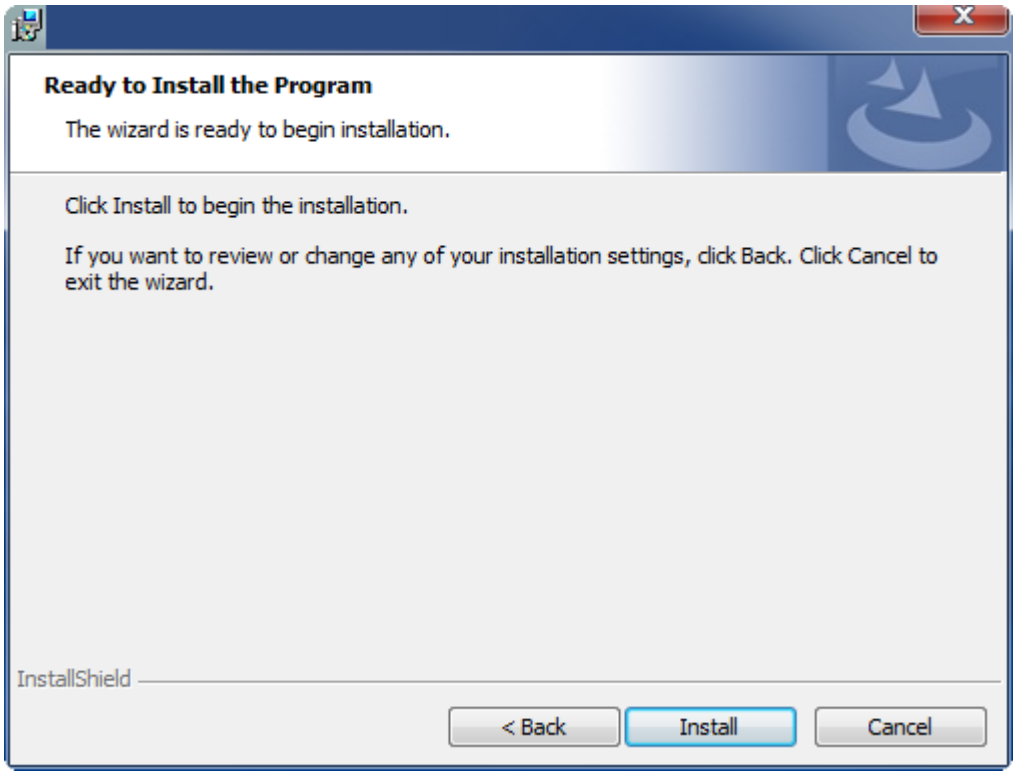
3. Geben Sie Ihre Benutzerdaten ein.



4. Wenn Sie die TwinCAT 3 Function vollständig installieren möchten, wählen Sie **Complete** als Installationstyp. Wenn Sie die Komponenten der TwinCAT 3 Function separat installieren möchten, wählen Sie **Custom**.

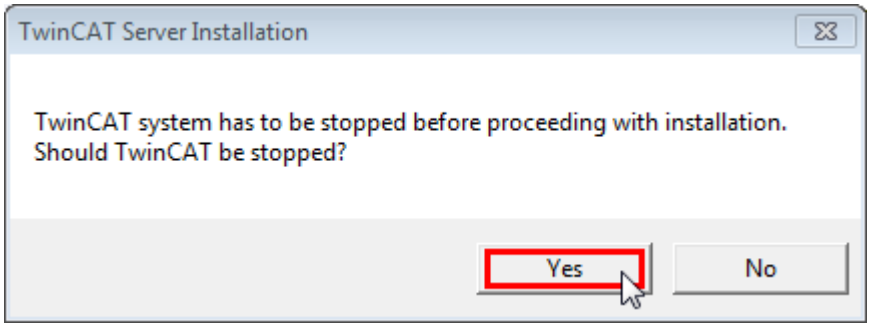


5. Wählen Sie **Next** und anschließend **Install**, um die Installation zu beginnen.

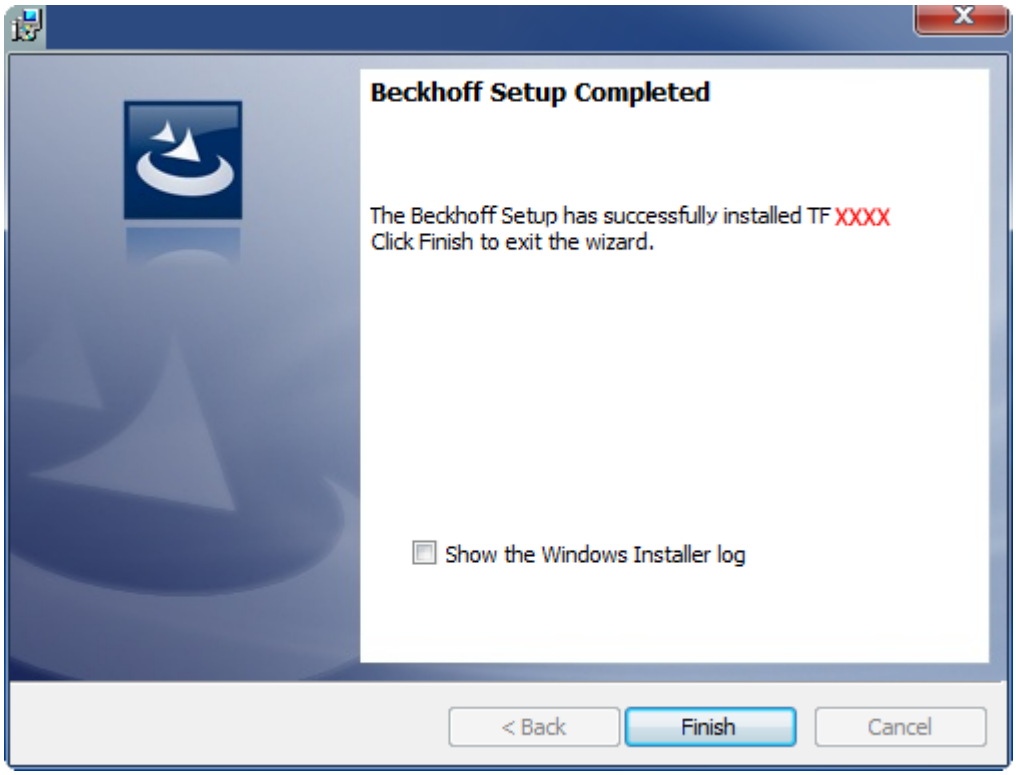


⇨ Ein Dialog weist Sie darauf hin, dass das TwinCAT-System für die weitere Installation gestoppt werden muss.

6. Bestätigen Sie den Dialog mit **Yes.**



7. Wählen Sie **Finish**, um das Setup zu beenden.



⇨ Die TwinCAT 3 Function wurde erfolgreich installiert und kann lizenziert werden (siehe Lizenzierung [▶ 12]).

3.3 Lizenzierung

Die TwinCAT 3 Function ist als Vollversion oder als 7-Tage-Testversion freischaltbar. Beide Lizenztypen sind über die TwinCAT-3-Entwicklungsumgebung (XAE) aktivierbar.

Lizenzierung der Vollversion einer TwinCAT 3 Function

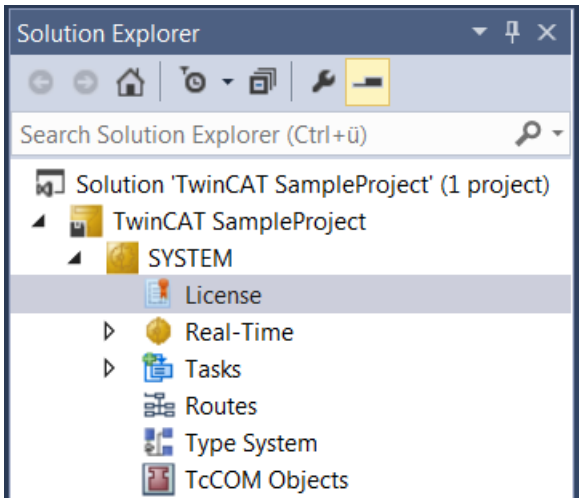
Die Beschreibung der Lizenzierung einer Vollversion finden Sie im Beckhoff Information System in der Dokumentation „TwinCAT-3-Lizenzierung“.

Lizenzierung der 7-Tage-Testversion einer TwinCAT 3 Function

Eine 7-Tage-Testversion kann nicht für einen TwinCAT-3-Lizenz-Dongle freigeschaltet werden.

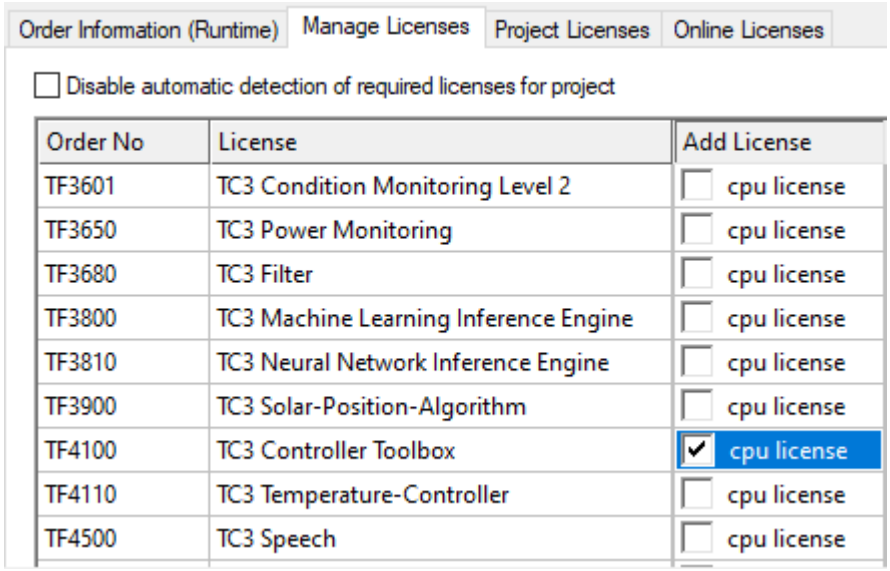
1. Starten Sie die TwinCAT-3-Entwicklungsumgebung (XAE).
2. Öffnen Sie ein bestehendes TwinCAT-3-Projekt oder legen Sie ein neues Projekt an.
3. Wenn Sie die Lizenz für ein Remote-Gerät aktivieren wollen, stellen Sie das gewünschte Zielsystem ein. Wählen Sie dazu in der Symbolleiste in der Drop-down-Liste **Choose Target System** das Zielsystem aus.
   ⇨ Die Lizenzierungseinstellungen beziehen sich immer auf das eingestellte Zielsystem. Mit der Aktivierung des Projekts auf dem Zielsystem werden automatisch auch die zugehörigen TwinCAT-3-Lizenzen auf dieses System kopiert.

4. Klicken Sie im **Solution Explorer** im Teilbaum **SYSTEM** doppelt auf **License.**



⇨ Der TwinCAT-3-Lizenzmanager öffnet sich.

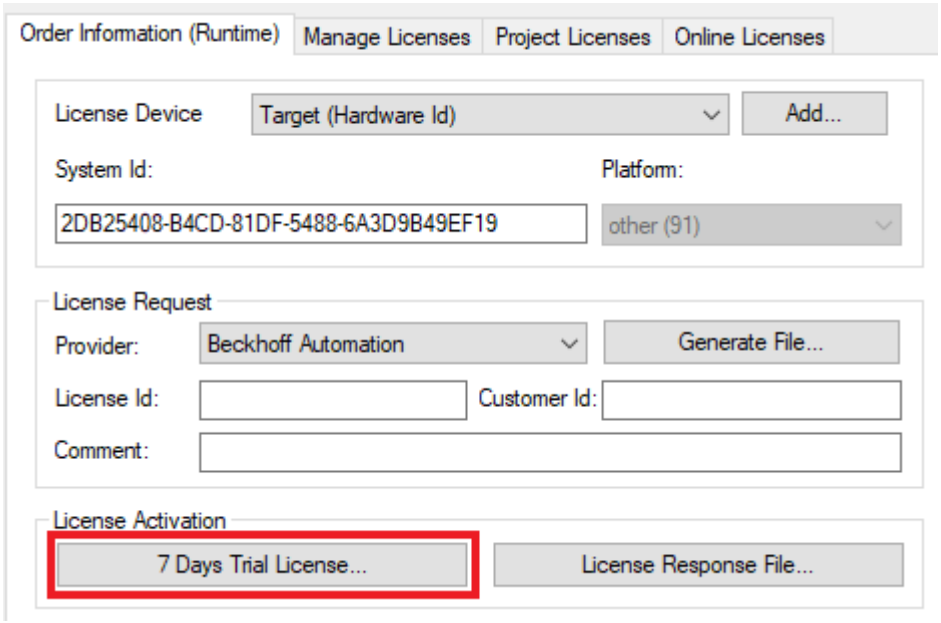
5. Öffnen Sie die Registerkarte **Manage Licenses.** Aktivieren Sie in der Spalte **Add License** das Auswahlkästchen für die Lizenz, die Sie Ihrem Projekt hinzufügen möchten (z. B. „TF4100 TC3 Controller Toolbox“).



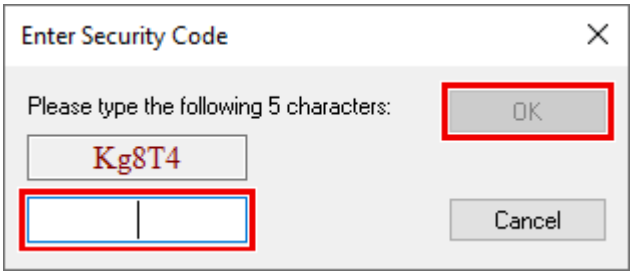
6. Öffnen Sie die Registerkarte **Order Information (Runtime).**

⇨ In der tabellarischen Übersicht der Lizenzen wird die zuvor ausgewählte Lizenz mit dem Status „missing“ angezeigt.

7. Klicken Sie auf **7 Days Trial License...**, um die 7-Tage-Testlizenz zu aktivieren.



⇨ Es öffnet sich ein Dialog, der Sie auffordert, den im Dialog angezeigten Sicherheitscode einzugeben.



8. Geben Sie den Code genauso ein, wie er angezeigt wird, und bestätigen Sie ihn.
9. Bestätigen Sie den nachfolgenden Dialog, der Sie auf die erfolgreiche Aktivierung hinweist.
   ⇨ In der tabellarischen Übersicht der Lizenzen gibt der Lizenzstatus nun das Ablaufdatum der Lizenz an.
10. Starten Sie das TwinCAT-System neu.

⇨ Die 7-Tage-Testversion ist freigeschaltet.

4 Konfiguration

4.1 Klemmenkonfiguration

Die Busklemmen KL6001, KL6011, KL6021, KL6031 und KL6041 können mit der Konfigurationssoftware KS2000 parametriert werden.

Alternativ ist auch eine Konfiguration über SPS-Bausteine möglich, die in der seriellen Kommunikationsbibliothek Tc2_SerialCom enthalten sind. So kann der Baustein KL6configuration lizenzfrei zur Konfiguration der Busklemmen verwendet werden.

4.2 Modbus Adressbereiche

Modbus definiert Zugriffsfunktionen für verschiedene Datenbereiche. Diese Datenbereiche werden in einem TwinCAT SPS-Programm als Variablen, beispielsweise als Word-Arrays, deklariert und dem Modbus-Slave-Funktionsbaustein als Eingangsparameter übergeben. Um die Bereiche eindeutig zu unterscheiden, hat jeder Bereich eine andere Modbus-Startadresse. Dieser Offset muss bei der Adressierung berücksichtigt werden.

Inputs

Der Datenbereich *Inputs* beschreibt üblicherweise die physikalischen Eingangsdaten, auf die nur lesend zugegriffen werden kann. Das können digitale Eingänge (Bit) oder Analogeingänge (Word) sein. Es liegt in der Hand des SPS-Programmierers, ob er dem Kommunikationspartner den direkten Zugriff auf die physikalischen Eingänge erlauben möchte. Es ist ebenfalls möglich, für die Modbus-Kommunikation einen Input-Bereich zu definieren, der nicht mit den physikalischen Eingängen identisch ist:

Definition der Modbus-Input-Daten als direktes Abbild der physikalischen Eingänge. Anfang und Größe des Datenbereichs können frei festgelegt werden und sind durch die reale Größe des Eingangsprozessabbildes der verwendeten Steuerung begrenzt.

```
VAR
Inputs AT%IW0 : ARRAY[0..255] OF WORD;
END_VAR
```

Definition der Modbus-Input-Daten als separater Modbus-Datenbereich unabhängig von den physikalischen Eingängen

```
VAR
Inputs : ARRAY[0..255] OF WORD;
END_VAR
```

Die maximale Größe des *Input*-Datenbereiches beträgt 2048 Words (ARRAY[0..2047] OF WORD).

Der Zugriff auf den *Input*-Bereich durch einen Modbus-Master ist mit folgenden Modbus-Funktionen möglich:

```
2 : Read Input Status
4 : Read Input Registers
```

Adressierung

Der *Input*-Bereich wird mit einem Offset 0 adressiert, das heißt, dass die im Telegramm übertragene Adresse 0 das erste Element im Input-Datenbereich anspricht.

Beispiele:

SPS-Variable	Zugriffsart	Adresse im Modbus-Telegramm	Adresse im Endgerät (Geräte-abhängig)
Inputs[0]	Wort	16#0	30001
Inputs[1]	Wort	16#1	30002
Inputs[0], Bit 0	Bit	16#0	10001
Inputs[1], Bit 14	Bit	16#1E	1001F

Outputs

Der Datenbereich *Outputs* beschreibt üblicherweise die physikalischen Ausgangsdaten, auf die lesend und schreibend zugegriffen werden kann. *Outputs* können digitale Ausgänge (Coils) oder Analogausgänge (Output Register) sein. Wie bei den *Inputs* ist kann der Bereich als physikalische Ausgangsvariable oder als einfache Variable deklariert werden.

Definition der Modbus-Output-Daten als direktes Abbild der physikalischen Ausgänge. Anfang und Größe des Datenbereichs können frei festgelegt werden und sind durch die reale Größe des Ausgangsprozessabbildes der verwendeten Steuerung begrenzt.

```
VAR
Outputs AT%QW0 : ARRAY[0..255] OF WORD;
END_VAR
```

Definition der Modbus-Output-Daten als separater Modbus-Datenbereich unabhängig von den physikalischen Ausgängen

```
VAR
Outputs : ARRAY[0..255] OF WORD;
END_VAR
```

Die maximale Größe des *Output*-Datenbereiches beträgt 14336 Words (ARRAY[0..14335] OF WORD).

Der Zugriff auf den *Output*-Bereich durch einen Modbus-Master ist mit folgenden Modbus-Funktionen möglich:

```
1 : Read Coil Status
3 : Read Holding Registers
5 : Force Single Coil
6 : Preset Single Register
15 : Force Multiple Coils
16 : Preset Multiple Registers
```

Adressierung

Der *Output*-Bereich wird mit einem Offset 16#800 adressiert, das heißt, dass die im Telegramm übertragene Adresse 16#800 das erste Element im Output-Datenbereich anspricht.

Beispiele:

SPS-Variable	Zugriffsart	Adresse im Modbus-Telegramm	Adresse im Endgerät (Geräte-abhängig)
Outputs[0]	Wort	16#800	40801
Outputs[1]	Wort	16#801	40802
Outputs[0], Bit 0	Bit	16#800	00801
Outputs[1], Bit 14	Bit	16#81E	0081F

Memory

Der Datenbereich *Memory* beschreibt einen SPS Variablenbereich ohne physikalische I/O-Zuordnung.

Definition der Modbus-Memory-Daten als SPS-Merker. Anfang und Größe des Datenbereichs können frei festgelegt werden.

```
VAR
Memory AT%MW0 : ARRAY[0..255] OF WORD;
END_VAR
```

Definition der Modbus-Memory-Daten als Variable ohne Merkeradresse

```
VAR
Memory : ARRAY[0..255] OF WORD;
END_VAR
```

Die maximale Größe des *Memory*-Datenbereiches beträgt 16384 Words (ARRAY[0..16383] OF WORD).

Der Zugriff auf den *Memory*-Bereich durch einen Modbus-Master ist mit folgenden Modbus-Funktionen möglich:

```
3 : Read Holding Registers
6 : Preset Single Register
16 : Preset Multiple Registers
```

Adressierung

Der *Memory*-Bereich wird mit einem Offset 16#4000 adressiert, das heißt, dass die im Telegramm übertragene Adresse 16#4000 das erste Wort im Memory-Datenbereich anspricht.

Beispiele:

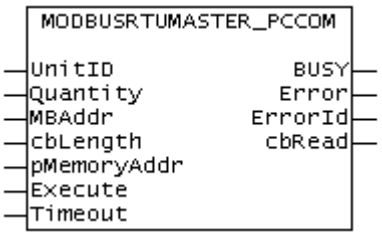
SPS-Variable	Zugriffsart	Adresse im Modbus-Telegramm	Adresse im Endgerät (Geräte-abhängig)
Memory[0]	Wort	16#4000	44001
Memory[1]	Wort	16#4001	44002

5 SPS API

5.1 Funktionsbausteine

5.1.1 [veraltet]

5.1.1.1 ModbusRtuMaster_PcCOM



Der Funktionsbaustein `ModbusRtuMaster_PcCOM` realisiert einen Modbus-Master, der über eine serielle PC-Schnittstelle (COM-Port) kommuniziert. Zur Kommunikation über eine serielle Busklemme KL6001, KL6011 oder KL6021 steht der Funktionsbaustein ModbusRtuMaster_KL6x5B [▶ 20] zur Verfügung.

Verbindung zur Hardware

Die zur Verknüpfung mit dem Kommunikations-Port notwendigen Datenstrukturen sind im Funktionsbaustein enthalten. Sie werden im TwinCAT System Manager angezeigt nachdem das SPS-Programm eingebunden ist und können dort mit einem COM-Port verbunden werden. Die Vorgehensweise ist analog zur Beschreibung im Kapitel Serielle PC-Schnittstelle der Dokumentation TF6340 TC3 Serial Communication.

Der Baustein wird nicht in seiner Grundform aufgerufen, sondern es werden in einem SPS-Programm einzelne Aktionen des Funktionsbausteins verwendet. Jede Modbus-Funktion ist als Aktion realisiert.

Unterstützte Modbus-Funktionen (Aktionen)

- **ModbusMaster.ReadCoils**
  Modbus-Funktion 1 = *Read Coils*

  Liest binäre Ausgänge (Coils) von einem angeschlossenen Slave. Die Daten werden in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` abgelegt.
- **ModbusMaster.ReadInputStatus**
  Modbus-Funktion 2 = *Read Input Status*

  Liest binäre Eingänge von einem angeschlossenen Slave. Die Daten werden in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` abgelegt.
- **ModbusMaster.ReadRegs**
  Modbus-Funktion 3 = *Read Holding Registers*

  Liest Daten von einem angeschlossenen Slave.
- **ModbusMaster.ReadInputRegs**
  Modbus-Funktion 4 = *Read Input Registers*

  Liest Eingangsregister von einem angeschlossenen Slave.
- **ModbusMaster.WriteSingleCoil**
  Modbus-Funktion 5 = *Write Single Coil*

  Sendet einen binären Ausgang (Coil) an einen angeschlossenen Slave. Die Daten müssen in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` zum Senden bereit liegen.
- **ModbusMaster.WriteSingleRegister**
  Modbus-Funktion 6 = *Write Single Register*

Sendet ein einzelnes Datenwort an einen angeschlossenen Slave

- **ModbusMaster.WriteMultipleCoils**
  Modbus-Funktion 15 = *Write Multiple Coils*

  Sendet binäre Ausgänge (Coils) an einen angeschlossenen Slave. Die Daten müssen in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` zum Senden bereit liegen.
- **ModbusMaster.WriteRegs**
  Modbus-Funktion 16 = *Preset Multiple Registers*

  Sendet Daten an einen angeschlossenen Slave
- **ModbusMaster.Diagnostics**
  Modbus-Funktion 8 = *Diagnostics*

  Sendet eine Diagnoseanforderung mit einem von Anwender angegebenen Funktionscode (subfunction code) an den Slave. Da bei dieser Funktion kein Speicher adressiert wird, wird hier der Funktionscode im Datenwort `MBAddr` übergeben. Eventuelle für die Funktion notwendige Daten werden über `pMemoryAddr` mitgegeben.

Eingänge

```
VAR_INPUT
    UnitID      : UINT;
    Quantity    : WORD;
    MBAddr      : WORD;
    cbLength    : UINT;
    pMemoryAddr : POINTER TO BYTE;
    Execute     : BOOL;
    Timeout     : TIME;
END_VAR
```

Name	Typ	Beschreibung
UnitID	UINT	Modbus Stationsadresse [▶ 45] (1..247). Der Modbus-Slave antwortet nur, wenn er Telegramme mit seiner eigenen Stationsadresse empfängt. Optional können hier auch Sammeladressen eingestellt werden, um auf beliebige Anfragen zu antworten. Die Adresse 0 ist für Broadcast-Telegramme reserviert und somit keine gültige Stationsadresse (siehe UnitID [▶ 45])
Quantity	WORD	Anzahl der zu lesenden oder zu schreibenden Datenworte bei Wort-orientierten Modbus-Funktionen. Bei Bit-orientierten Modbus-Funktionen gibt Quantity die Anzahl der Bits (Inputs oder Coils) an.
MBAddr	WORD	Modbus-Datenadresse, von der die Daten aus dem Endgerät (Slave) gelesen werden. Diese Adresse wird unverändert zum Slave übertragen und wird dort als Datenadresse interpretiert. Bei der *Diagnostics*-Funktion (8) wird hier der Funktionscode (subfunction code) übergeben.
cbLength	UINT	Größe der verwendeten Datenvariable für Sende- oder Lese-Aktionen in Bytes. cbLength muss größer oder gleich der durch Quantity bestimmten übertragenen Datenmenge sein. Bei Wortzugriffen gilt z. B.: *[cbLength >= Quantity * 2]*. *cbLength* kann mit SIZEOF(ModbusDaten) berechnet werden.
pMemoryAddr	BYTE	Speicheradresse in der SPS, die mit ADR (ModbusDaten) berechnet wird. Bei Leseaktionen werden die gelesenen Daten in der adressierten Variablen abgelegt. Bei Sende-Aktionen werden die Daten aus der adressierten Variablen zum Endgerät übertragen.
Execute	BOOL	Startsignal. Mit steigender Flanke am Eingang Execute wird die Aktion ausgelöst.
Timeout	TIME	Timeout-Wert für das Warten auf eine Reaktion des angesprochenen Slaves.

Ausgänge

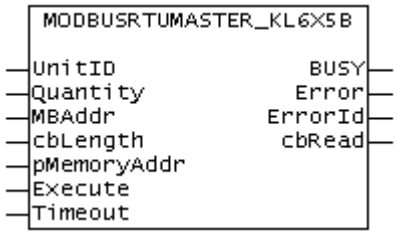
```
VAR_OUTPUT
    BUSY    : BOOL;
    Error   : BOOL;
    ErrorId : MODBUS_ERRORS;
    cbRead  : UINT;
END_VAR
```

Name	Typ	Beschreibung
Busy	BOOL	Zeigt an, dass der Funktionsbaustein aktiv ist. *Busy* wird mit steigender Flanke an *Execute* TRUE und wird wieder FALSE nachdem die gestartete Aktion beendet ist. Es kann immer nur eine Aktion gleichzeitig aktiv sein.
Error	BOOL	Zeigt an, dass bei der Bearbeitung einer Aktion ein Fehler aufgetreten ist.
ErrorId	MODBUS_ERRORS	Zeigt eine Fehlernummer [▶ 46] im Falle einer gestörten oder fehlerhaften Kommunikation an.
cbRead	UINT	Liefert die Anzahl der gelesenen Datenbytes bei einer Lese-Aktion.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_ModbusRTU

5.1.1.2 ModbusRtuMaster_KL6x5B



Der Funktionsbaustein `ModbusRtuMaster_KL6x5B` realisiert einen Modbus-Master, der über eine serielle Busklemme KL6001, KL6011 oder KL6021 kommuniziert.

Verbindung zur Hardware

Die zur Verknüpfung mit dem Kommunikations-Port notwendigen Datenstrukturen sind im Funktionsbaustein enthalten. Auf einem PC erfolgt die Zuweisung im TwinCAT System Manager analog zur Beschreibung im Kapitel Serielle Busklemme der Dokumentation TF6340 TC3 Serial Communication.

Der Baustein wird nicht in seiner Grundform aufgerufen, sondern es werden in einem SPS-Programm einzelne Aktionen des Funktionsbausteins verwendet. Jede Modbus-Funktion ist als Aktion realisiert.

Unterstützte Modbus-Funktionen (Aktionen)

- **ModbusMaster.ReadCoils**
  Modbus-Funktion 1 = *Read Coils*

  Liest binäre Ausgänge (Coils) von einem angeschlossenen Slave. Die Daten werden in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` abgelegt.
- **ModbusMaster.ReadInputStatus**
  Modbus-Funktion 2 = *Read Input Status*

  Liest binäre Eingänge von einem angeschlossenen Slave. Die Daten werden in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` abgelegt.

- **ModbusMaster.ReadRegs**
  Modbus-Funktion 3 = *Read Holding Registers*

  Liest Daten von einem angeschlossenen Slave.
- **ModbusMaster.ReadInputRegs**
  Modbus-Funktion 4 = *Read Input Registers*

  Liest Eingangsregister von einem angeschlossenen Slave.
- **ModbusMaster.WriteSingleCoil**
  Modbus-Funktion 5 = *Write Single Coil*

  Sendet einen binären Ausgang (Coil) an einen angeschlossenen Slave. Die Daten müssen in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` zum Senden bereit liegen.
- **ModbusMaster.WriteSingleRegister**
  Modbus-Funktion 6 = *Write Single Register*

  Sendet ein einzelnes Datenwort an einen angeschlossenen Slave
- **ModbusMaster.WriteMultipleCoils**
  Modbus-Funktion 15 = *Write Multiple Coils*

  Sendet binäre Ausgänge (Coils) an einen angeschlossenen Slave. Die Daten müssen in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` zum Senden bereit liegen.
- **ModbusMaster.WriteRegs**
  Modbus-Funktion 16 = *Preset Multiple Registers*

  Sendet Daten an einen angeschlossenen Slave
- **ModbusMaster.Diagnostics**
  Modbus-Funktion 8 = *Diagnostics*

  Sendet eine Diagnoseanforderung mit einem von Anwender angegebenen Funktionscode (subfunction code) an den Slave. Da bei dieser Funktion kein Speicher adressiert wird, wird hier der Funktionscode im Datenwort `MBAddr` übergeben. Eventuelle für die Funktion notwendige Daten werden über `pMemoryAddr` mitgegeben.

Eingänge

```
VAR_INPUT
    UnitID      : UINT;
    Quantity    : WORD;
    MBAddr      : WORD;
    cbLength    : UINT;
    pMemoryAddr : POINTER TO BYTE;
    Execute     : BOOL;
    Timeout     : TIME;
END_VAR
```

Name	Typ	Beschreibung
UnitID	UINT	Modbus Stationsadresse [▶ 45] (1..247). Der Modbus-Slave antwortet nur, wenn er Telegramme mit seiner eigenen Stationsadresse empfängt. Optional können hier auch Sammeladressen eingestellt werden, um auf beliebige Anfragen zu antworten. Die Adresse 0 ist für Broadcast-Telegramme reserviert und somit keine gültige Stationsadresse (siehe UnitID [▶ 45])
Quantity	WORD	Anzahl der zu lesenden oder zu schreibenden Datenworte bei Wort-orientierten Modbus-Funktionen. Bei Bit-orientierten Modbus-Funktionen gibt Quantity die Anzahl der Bits (Inputs oder Coils) an.
MBAddr	WORD	Modbus-Datenadresse, von der die Daten aus dem Endgerät (Slave) gelesen werden. Diese Adresse wird unverändert zum Slave übertragen und wird dort als Datenadresse interpretiert. Bei der *Diagnostics*-Funktion (8) wird hier der Funktionscode (subfunction code) übergeben.
cbLength	UINT	Größe der verwendeten Datenvariable für Sende- oder Lese-Aktionen in Bytes. cbLength muss größer oder gleich der durch Quantity bestimmten übertragenen Datenmenge sein. Bei Wortzugriffen gilt z. B.: *[cbLength >= Quantity * 2]. cbLength* kann mit SIZEOF(ModbusDaten) berechnet werden.
pMemoryAddr	BYTE	Speicheradresse in der SPS, die mit ADR (ModbusDaten) berechnet wird. Bei Leseaktionen werden die gelesenen Daten in der adressierten Variablen abgelegt. Bei Sende-Aktionen werden die Daten aus der adressierten Variablen zum Endgerät übertragen.
Execute	BOOL	Startsignal. Mit steigender Flanke am Eingang Execute wird die Aktion ausgelöst.
Timeout	TIME	Timeout-Wert für das Warten auf eine Reaktion des angesprochenen Slaves.

Ausgänge

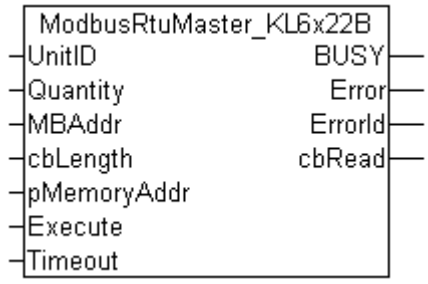
```
VAR_OUTPUT
    BUSY    : BOOL;
    Error   : BOOL;
    ErrorId : MODBUS_ERRORS;
    cbRead  : UINT;
END_VAR
```

Name	Typ	Beschreibung
Busy	BOOL	Zeigt an, dass der Funktionsbaustein aktiv ist. *Busy* wird mit steigender Flanke an *Execute* TRUE und wird wieder FALSE nachdem die gestartete Aktion beendet ist. Es kann immer nur eine Aktion gleichzeitig aktiv sein.
Error	BOOL	Zeigt an, dass bei der Bearbeitung einer Aktion ein Fehler aufgetreten ist.
ErrorId	MODBUS_ERRORS	Zeigt eine Fehlernummer [▶ 46] im Falle einer gestörten oder fehlerhaften Kommunikation an.
cbRead	UINT	Liefert die Anzahl der gelesenen Datenbytes bei einer Lese-Aktion.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_ModbusRTU

5.1.1.3 ModbusRtuMaster_KL6x22B



Der Funktionsbaustein `ModbusRtuMaster_KL6x22B` realisiert einen Modbus-Master, der über eine serielle Busklemme KL6031 oder KL6041 kommuniziert. Zur Kommunikation über eine serielle PC-Schnittstelle (COM-Port) steht der Funktionsbaustein ModbusRtuMaster_PcCOM [▶ 18] zur Verfügung.

Verbindung zur Hardware

Die zur Verknüpfung mit dem Kommunikations-Port notwendigen Datenstrukturen sind im Funktionsbaustein enthalten. Auf einem PC erfolgt die Zuweisung im TwinCAT System Manager analog zur Beschreibung im Kapitel Serielle Busklemme der Dokumentation TF6340 TC3 Serial Communication.

Der Baustein wird nicht in seiner Grundform aufgerufen, sondern es werden in einem SPS-Programm einzelne Aktionen des Funktionsbausteins verwendet. Jede Modbus-Funktion ist als Aktion realisiert.

Unterstützte Modbus-Funktionen (Aktionen)

- **ModbusMaster.ReadCoils**
  Modbus-Funktion 1 = *Read Coils*

  Liest binäre Ausgänge (Coils) von einem angeschlossenen Slave. Die Daten werden in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` abgelegt.
- **ModbusMaster.ReadInputStatus**
  Modbus-Funktion 2 = *Read Input Status*

  Liest binäre Eingänge von einem angeschlossenen Slave. Die Daten werden in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` abgelegt.
- **ModbusMaster.ReadRegs**
  Modbus-Funktion 3 = *Read Holding Registers*

  Liest Daten von einem angeschlossenen Slave.
- **ModbusMaster.ReadInputRegs**
  Modbus-Funktion 4 = *Read Input Registers*

  Liest Eingangsregister von einem angeschlossenen Slave.
- **ModbusMaster.WriteSingleCoil**
  Modbus-Funktion 5 = *Write Single Coil*

  Sendet einen binären Ausgang (Coil) an einen angeschlossenen Slave. Die Daten müssen in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` zum Senden bereit liegen.
- **ModbusMaster.WriteSingleRegister**
  Modbus-Funktion 6 = *Write Single Register*

  Sendet ein einzelnes Datenwort an einen angeschlossenen Slave
- **ModbusMaster.WriteMultipleCoils**
  Modbus-Funktion 15 = *Write Multiple Coils*

  Sendet binäre Ausgänge (Coils) an einen angeschlossenen Slave. Die Daten müssen in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` zum Senden bereit liegen.
- **ModbusMaster.WriteRegs**
  Modbus-Funktion 16 = *Preset Multiple Registers*

Sendet Daten an einen angeschlossenen Slave

- **ModbusMaster.Diagnostics**
  Modbus-Funktion 8 = *Diagnostics*

  Sendet eine Diagnoseanforderung mit einem von Anwender angegebenen Funktionscode (subfunction code) an den Slave. Da bei dieser Funktion kein Speicher adressiert wird, wird hier der Funktionscode im Datenwort `MBAddr` übergeben. Eventuelle für die Funktion notwendige Daten werden über `pMemoryAddr` mitgegeben.

Eingänge

```
VAR_INPUT
    UnitID      : UINT;
    Quantity    : WORD;
    MBAddr      : WORD;
    cbLength    : UINT;
    pMemoryAddr : POINTER TO BYTE;
    Execute     : BOOL;
    Timeout     : TIME;
END_VAR
```

Name	Typ	Beschreibung
UnitID	UINT	Modbus Stationsadresse [▶ 45] (1..247). Der Modbus-Slave antwortet nur, wenn er Telegramme mit seiner eigenen Stationsadresse empfängt. Optional können hier auch Sammeladressen eingestellt werden, um auf beliebige Anfragen zu antworten. Die Adresse 0 ist für Broadcast-Telegramme reserviert und somit keine gültige Stationsadresse (siehe UnitID [▶ 45])
Quantity	WORD	Anzahl der zu lesenden oder zu schreibenden Datenworte bei Wort-orientierten Modbus-Funktionen. Bei Bit-orientierten Modbus-Funktionen gibt Quantity die Anzahl der Bits (Inputs oder Coils) an.
MBAddr	WORD	Modbus-Datenadresse, von der die Daten aus dem Endgerät (Slave) gelesen werden. Diese Adresse wird unverändert zum Slave übertragen und wird dort als Datenadresse interpretiert. Bei der *Diagnostics*-Funktion (8) wird hier der Funktionscode (subfunction code) übergeben.
cbLength	UINT	Größe der verwendeten Datenvariable für Sende- oder Lese-Aktionen in Bytes. cbLength muss größer oder gleich der durch Quantity bestimmten übertragenen Datenmenge sein. Bei Wortzugriffen gilt z. B.: *[cbLength >= Quantity * 2]*. *cbLength* kann mit SIZEOF(ModbusDaten) berechnet werden.
pMemoryAddr	BYTE	Speicheradresse in der SPS, die mit ADR (ModbusDaten) berechnet wird. Bei Leseaktionen werden die gelesenen Daten in der adressierten Variablen abgelegt. Bei Sende-Aktionen werden die Daten aus der adressierten Variablen zum Endgerät übertragen.
Execute	BOOL	Startsignal. Mit steigender Flanke am Eingang Execute wird die Aktion ausgelöst.
Timeout	TIME	Timeout-Wert für das Warten auf eine Reaktion des angesprochenen Slaves.

Ausgänge

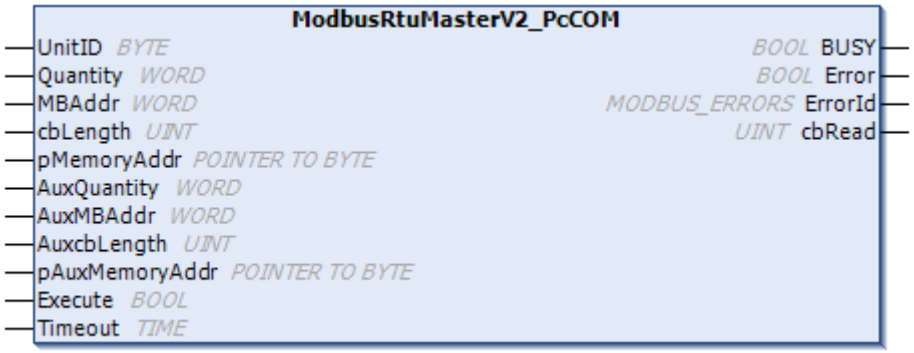
```
VAR_OUTPUT
    BUSY    : BOOL;
    Error   : BOOL;
    ErrorId : MODBUS_ERRORS;
    cbRead  : UINT;
END_VAR
```

Name	Typ	Beschreibung
Busy	BOOL	Zeigt an, dass der Funktionsbaustein aktiv ist. *Busy* wird mit steigender Flanke an *Execute* TRUE und wird wieder FALSE nachdem die gestartete Aktion beendet ist. Es kann immer nur eine Aktion gleichzeitig aktiv sein.
Error	BOOL	Zeigt an, dass bei der Bearbeitung einer Aktion ein Fehler aufgetreten ist.
ErrorId	MODBUS_ERRORS	Zeigt eine Fehlernummer [▶ 46] im Falle einer gestörten oder fehlerhaften Kommunikation an.
cbRead	UINT	Liefert die Anzahl der gelesenen Datenbytes bei einer Lese-Aktion.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_ModbusRTU

5.1.2 ModbusRtuMasterV2_PcCOM



Der Funktionsbaustein `ModbusRtuMasterV2_PcCOM` realisiert einen Modbus-Master, der über eine serielle PC-Schnittstelle (COM-Port) kommuniziert. Zur Kommunikation über eine serielle Busklemme stehen weitere Funktionsbausteine zur Verfügung.

Verbindung zur Hardware

Die zur Verknüpfung mit dem Kommunikations-Port notwendigen Datenstrukturen sind im Funktionsbaustein enthalten. Sie werden im TwinCAT System Manager angezeigt nachdem das SPS-Programm eingebunden ist und können dort mit einem COM-Port verbunden werden. Die Vorgehensweise ist analog zur Beschreibung im Kapitel Serielle PC-Schnittstelle der Dokumentation TF6340 TC3 Serial Communication.

Der Baustein wird nicht in seiner Grundform aufgerufen, sondern es werden in einem SPS-Programm einzelne Aktionen des Funktionsbausteins verwendet. Jede Modbus-Funktion ist als Aktion realisiert.

Unterstützte Modbus-Funktionen (Aktionen)

- **ModbusMaster.ReadCoils**
  Modbus-Funktion 1 = *Read Coils*

  Liest binäre Ausgänge (Coils) von einem angeschlossenen Slave. Die Daten werden in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` abgelegt.
- **ModbusMaster.ReadInputStatus**
  Modbus-Funktion 2 = *Read Input Status*

  Liest binäre Eingänge von einem angeschlossenen Slave. Die Daten werden in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` abgelegt.

- **ModbusMaster.ReadRegs**
  Modbus-Funktion 3 = *Read Holding Registers*

  Liest Daten von einem angeschlossenen Slave.
- **ModbusMaster.ReadInputRegs**
  Modbus-Funktion 4 = *Read Input Registers*

  Liest Eingangsregister von einem angeschlossenen Slave.
- **ModbusMaster.WriteSingleCoil**
  Modbus-Funktion 5 = *Write Single Coil*

  Sendet einen binären Ausgang (Coil) an einen angeschlossenen Slave. Die Daten müssen in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` zum Senden bereit liegen.
- **ModbusMaster.WriteSingleRegister**
  Modbus-Funktion 6 = *Write Single Register*

  Sendet ein einzelnes Datenwort an einen angeschlossenen Slave
- **ModbusMaster.WriteMultipleCoils**
  Modbus-Funktion 15 = *Write Multiple Coils*

  Sendet binäre Ausgänge (Coils) an einen angeschlossenen Slave. Die Daten müssen in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` zum Senden bereit liegen.
- **ModbusMaster.WriteRegs**
  Modbus-Funktion 16 = *Preset Multiple Registers*

  Sendet Daten an einen angeschlossenen Slave
- **ModbusMaster.Diagnostics**
  Modbus-Funktion 8 = *Diagnostics*

  Sendet eine Diagnoseanforderung mit einem von Anwender angegebenen Funktionscode (subfunction code) an den Slave. Da bei dieser Funktion kein Speicher adressiert wird, wird hier der Funktionscode im Datenwort `MBAddr` übergeben. Eventuelle für die Funktion notwendige Daten werden über `pMemoryAddr` mitgegeben.

Unterstützte Modbus-Funktionen (Aktionen) der MasterV2 Funktionsbausteine

- **ModbusMaster.ReadWriteRegs**
  Modbus-Funktion 23 = *Read/Write Multiple Registers*

  Sendet die über die Aux-Parmeter spezifizierten Daten an einen angeschlossenen Slave und empfängt gleichzeitig Daten von dem Slave. Die empfangenen Daten werden an der durch pMemoryAddr spezifizierten Adresse abgelegt.
- **ModbusMaster.UserReadWrite**
  Universelles Anwendertelegramm

  Der Modbus-Funktion Code wird vom Anwender im ersten Byte spezifizierten Daten (pMemoryAddr) angegeben. Der Anwender ist mit dieser Funktion in der Lage Modbus-Telegramm mit beliebigem Funktionscode zu senden. Evtl. vom Slave empfangene Daten werden an der durch pAuxMemoryAddr spezifizierten Adresse abgelegt.

Eingänge

```
VAR_INPUT
    UnitID      : BYTE;
    Quantity    : WORD;
    MBAddr      : WORD;
    cbLength    : UINT;
    pMemoryAddr : POINTER TO BYTE;

    AuxQuantity    : WORD;
    AuxMBAddr      : WORD;
    AuxcbLength    : UINT;
    pAuxMemoryAddr : POINTER TO BYTE;

    Execute     : BOOL;
    Timeout     : TIME;
END_VAR
```

Name	Typ	Beschreibung
UnitID	UINT	Modbus Stationsadresse [▶ 45] (1..247). Der Modbus-Slave antwortet nur, wenn er Telegramme mit seiner eigenen Stationsadresse empfängt. Optional können hier auch Sammeladressen eingestellt werden, um auf beliebige Anfragen zu antworten. Die Adresse 0 ist für Broadcast-Telegramme reserviert und somit keine gültige Stationsadresse (siehe UnitID [▶ 45])
Quantity	WORD	Anzahl der zu lesenden oder zu schreibenden Datenworte bei Wort-orientierten Modbus-Funktionen. Bei Bit-orientierten Modbus-Funktionen gibt Quantity die Anzahl der Bits (Inputs oder Coils) an.
MBAddr	WORD	Modbus-Datenadresse, von der die Daten aus dem Endgerät (Slave) gelesen werden. Diese Adresse wird unverändert zum Slave übertragen und wird dort als Datenadresse interpretiert. Bei der *Diagnostics*-Funktion (8) wird hier der Funktionscode (subfunction code) übergeben.
cbLength	UINT	Größe der verwendeten Datenvariable für Sende- oder Lese-Aktionen in Bytes. cbLength muss größer oder gleich der durch Quantity bestimmten übertragenen Datenmenge sein. Bei Wortzugriffen gilt z. B.: *[cbLength >= Quantity * 2]. cbLength* kann mit SIZEOF(ModbusDaten) berechnet werden.
pMemoryAddr	BYTE	Speicheradresse in der SPS, die mit ADR (ModbusDaten) berechnet wird. Bei Leseaktionen werden die gelesenen Daten in der adressierten Variablen abgelegt. Bei Sende-Aktionen werden die Daten aus der adressierten Variablen zum Endgerät übertragen.
AuxQuantity	WORD	Zusätzlicher Parameter, welcher nur bei Read/Write Funktionen verwendet wird, siehe ReadWriteRegs/ UserReadWrite [▶ 26]
AuxMBAddr	WORD	Zusätzlicher Parameter, welcher nur bei Read/Write Funktionen verwendet wird, siehe ReadWriteRegs/ UserReadWrite [▶ 26].
AuxcbLength	UINT	Zusätzlicher Parameter, welcher nur bei Read/Write Funktionen verwendet wird, siehe ReadWriteRegs/ UserReadWrite [▶ 26]
pAuxMemoryAddr	BYTE	Zusätzlicher Parameter, welcher nur bei Read/Write Funktionen verwendet wird, siehe ReadWriteRegs/ UserReadWrite [▶ 26]
Execute	BOOL	Startsignal. Mit steigender Flanke am Eingang Execute wird die Aktion ausgelöst.
Timeout	TIME	Timeout-Wert für das Warten auf eine Reaktion des angesprochenen Slaves.

Ausgänge

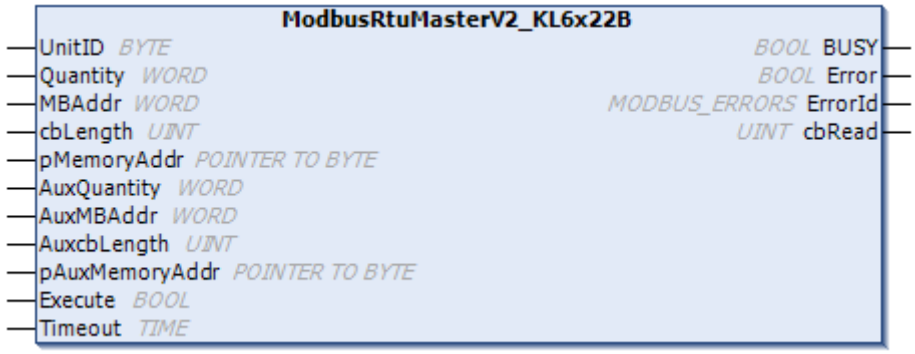
```
VAR_OUTPUT
    BUSY    : BOOL;
    Error   : BOOL;
    ErrorId : MODBUS_ERRORS;
    cbRead  : UINT;
END_VAR
```

Name	Typ	Beschreibung
Busy	BOOL	Zeigt an, dass der Funktionsbaustein aktiv ist. *Busy* wird mit steigender Flanke an *Execute* TRUE und wird wieder FALSE nachdem die gestartete Aktion beendet ist. Es kann immer nur eine Aktion gleichzeitig aktiv sein.
Error	BOOL	Zeigt an, dass bei der Bearbeitung einer Aktion ein Fehler aufgetreten ist.
ErrorId	MODBUS_ERRORS	Zeigt eine Fehlernummer [▶ 46] im Falle einer gestörten oder fehlerhaften Kommunikation an.
cbRead	UINT	Liefert die Anzahl der gelesenen Datenbytes bei einer Lese-Aktion.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_ModbusRTU

5.1.3 ModbusRtuMasterV2_KL6x22B



Der Funktionsbaustein `ModbusRtuMasterV2_KL6x22B` realisiert einen Modbus-Master, der über eine serielle Busklemme KL6031 oder KL6041 kommuniziert. Es werden ebenso serielle EtherCAT-Klemmen mit 22 Bytes Daten Prozessabbild unterstützt. Zur Kommunikation über eine serielle PC-Schnittstelle (COM-Port) steht der Funktionsbaustein ModbusRtuMasterV2_PcCOM [▶ 25] zur Verfügung.

Verbindung zur Hardware

Die zur Verknüpfung mit dem Kommunikations-Port notwendigen Datenstrukturen sind im Funktionsbaustein enthalten. Auf einem PC erfolgt die Zuweisung im TwinCAT System Manager analog zur Beschreibung im Kapitel Serielle Busklemme der Dokumentation TF6340 TC3 Serial Communication.

Der Baustein wird nicht in seiner Grundform aufgerufen, sondern es werden in einem SPS-Programm einzelne Aktionen des Funktionsbausteins verwendet. Jede Modbus-Funktion ist als Aktion realisiert.

Unterstützte Modbus-Funktionen (Aktionen)

- **ModbusMaster.ReadCoils**
  Modbus-Funktion 1 = *Read Coils*

  Liest binäre Ausgänge (Coils) von einem angeschlossenen Slave. Die Daten werden in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` abgelegt.
- **ModbusMaster.ReadInputStatus**
  Modbus-Funktion 2 = *Read Input Status*

  Liest binäre Eingänge von einem angeschlossenen Slave. Die Daten werden in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` abgelegt.

- **ModbusMaster.ReadRegs**
  Modbus-Funktion 3 = *Read Holding Registers*

  Liest Daten von einem angeschlossenen Slave.
- **ModbusMaster.ReadInputRegs**
  Modbus-Funktion 4 = *Read Input Registers*

  Liest Eingangsregister von einem angeschlossenen Slave.
- **ModbusMaster.WriteSingleCoil**
  Modbus-Funktion 5 = *Write Single Coil*

  Sendet einen binären Ausgang (Coil) an einen angeschlossenen Slave. Die Daten müssen in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` zum Senden bereit liegen.
- **ModbusMaster.WriteSingleRegister**
  Modbus-Funktion 6 = *Write Single Register*

  Sendet ein einzelnes Datenwort an einen angeschlossenen Slave
- **ModbusMaster.WriteMultipleCoils**
  Modbus-Funktion 15 = *Write Multiple Coils*

  Sendet binäre Ausgänge (Coils) an einen angeschlossenen Slave. Die Daten müssen in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` zum Senden bereit liegen.
- **ModbusMaster.WriteRegs**
  Modbus-Funktion 16 = *Preset Multiple Registers*

  Sendet Daten an einen angeschlossenen Slave
- **ModbusMaster.Diagnostics**
  Modbus-Funktion 8 = *Diagnostics*

  Sendet eine Diagnoseanforderung mit einem von Anwender angegebenen Funktionscode (subfunction code) an den Slave. Da bei dieser Funktion kein Speicher adressiert wird, wird hier der Funktionscode im Datenwort `MBAddr` übergeben. Eventuelle für die Funktion notwendige Daten werden über `pMemoryAddr` mitgegeben.

Unterstützte Modbus-Funktionen (Aktionen) der MasterV2 Funktionsbausteine

- **ModbusMaster.ReadWriteRegs**
  Modbus-Funktion 23 = *Read/Write Multiple Registers*

  Sendet die über die Aux-Parmeter spezifizierten Daten an einen angeschlossenen Slave und empfängt gleichzeitig Daten von dem Slave. Die empfangenen Daten werden an der durch pMemoryAddr spezifizierten Adresse abgelegt.
- **ModbusMaster.UserReadWrite**
  Universelles Anwendertelegramm

  Der Modbus-Funktion Code wird vom Anwender im ersten Byte spezifizierten Daten (pMemoryAddr) angegeben. Der Anwender ist mit dieser Funktion in der Lage Modbus-Telegramm mit beliebigem Funktionscode zu senden. Evtl. vom Slave empfangene Daten werden an der durch pAuxMemoryAddr spezifizierten Adresse abgelegt.

Eingänge

```
VAR_INPUT
    UnitID      : BYTE;
    Quantity    : WORD;
    MBAddr      : WORD;
    cbLength    : UINT;
    pMemoryAddr : POINTER TO BYTE;

    AuxQuantity    : WORD;
    AuxMBAddr      : WORD;
    AuxcbLength    : UINT;
    pAuxMemoryAddr : POINTER TO BYTE;

    Execute     : BOOL;
    Timeout     : TIME;
END_VAR
```

Name	Typ	Beschreibung
UnitID	UINT	Modbus Stationsadresse [▶ 45] (1..247). Der Modbus-Slave antwortet nur, wenn er Telegramme mit seiner eigenen Stationsadresse empfängt. Optional können hier auch Sammeladressen eingestellt werden, um auf beliebige Anfragen zu antworten. Die Adresse 0 ist für Broadcast-Telegramme reserviert und somit keine gültige Stationsadresse (siehe UnitID [▶ 45])
Quantity	WORD	Anzahl der zu lesenden oder zu schreibenden Datenworte bei Wort-orientierten Modbus-Funktionen. Bei Bit-orientierten Modbus-Funktionen gibt Quantity die Anzahl der Bits (Inputs oder Coils) an.
MBAddr	WORD	Modbus-Datenadresse, von der die Daten aus dem Endgerät (Slave) gelesen werden. Diese Adresse wird unverändert zum Slave übertragen und wird dort als Datenadresse interpretiert. Bei der *Diagnostics*-Funktion (8) wird hier der Funktionscode (subfunction code) übergeben.
cbLength	UINT	Größe der verwendeten Datenvariable für Sende- oder Lese-Aktionen in Bytes. cbLength muss größer oder gleich der durch Quantity bestimmten übertragenen Datenmenge sein. Bei Wortzugriffen gilt z. B.: *[cbLength >= Quantity * 2]. cbLength* kann mit SIZEOF(ModbusDaten) berechnet werden.
pMemoryAddr	BYTE	Speicheradresse in der SPS, die mit ADR (ModbusDaten) berechnet wird. Bei Leseaktionen werden die gelesenen Daten in der adressierten Variablen abgelegt. Bei Sende-Aktionen werden die Daten aus der adressierten Variablen zum Endgerät übertragen.
AuxQuantity	WORD	Zusätzlicher Parameter, welcher nur bei Read/Write Funktionen verwendet wird, siehe ReadWriteRegs/ UserReadWrite [▶ 29]
AuxMBAddr	WORD	Zusätzlicher Parameter, welcher nur bei Read/Write Funktionen verwendet wird, siehe ReadWriteRegs/ UserReadWrite [▶ 29].
AuxcbLength	UINT	Zusätzlicher Parameter, welcher nur bei Read/Write Funktionen verwendet wird, siehe ReadWriteRegs/ UserReadWrite [▶ 29]
pAuxMemoryAddr	BYTE	Zusätzlicher Parameter, welcher nur bei Read/Write Funktionen verwendet wird, siehe ReadWriteRegs/ UserReadWrite [▶ 29]
Execute	BOOL	Startsignal. Mit steigender Flanke am Eingang Execute wird die Aktion ausgelöst.
Timeout	TIME	Timeout-Wert für das Warten auf eine Reaktion des angesprochenen Slaves.

Ausgänge

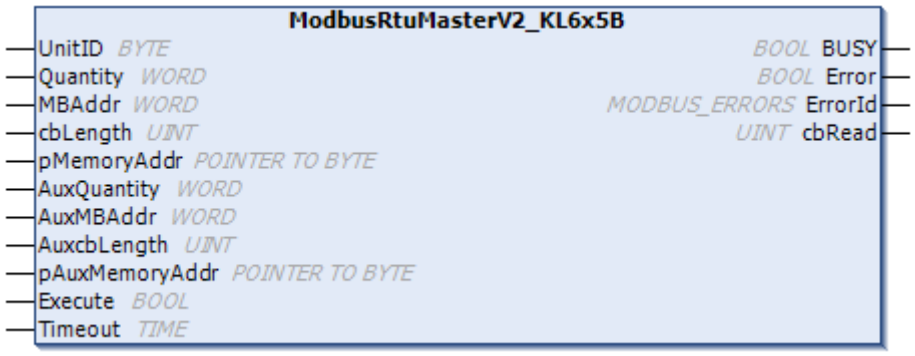
```
VAR_OUTPUT
    BUSY    : BOOL;
    Error   : BOOL;
    ErrorId : MODBUS_ERRORS;
    cbRead  : UINT;
END_VAR
```

Name	Typ	Beschreibung
Busy	BOOL	Zeigt an, dass der Funktionsbaustein aktiv ist. *Busy* wird mit steigender Flanke an *Execute* TRUE und wird wieder FALSE nachdem die gestartete Aktion beendet ist. Es kann immer nur eine Aktion gleichzeitig aktiv sein.
Error	BOOL	Zeigt an, dass bei der Bearbeitung einer Aktion ein Fehler aufgetreten ist.
ErrorId	MODBUS_ERRORS	Zeigt eine Fehlernummer [▶ 46] im Falle einer gestörten oder fehlerhaften Kommunikation an.
cbRead	UINT	Liefert die Anzahl der gelesenen Datenbytes bei einer Lese-Aktion.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_ModbusRTU

5.1.4 ModbusRtuMasterV2_KL6x5B



Der Funktionsbaustein `ModbusRtuMasterV2_KL6x5B` realisiert einen Modbus-Master, der über eine serielle Busklemme KL6001, KL6011 oder KL6021 kommuniziert. Zur Kommunikation über eine serielle PC-Schnittstelle (COM-Port) steht der Funktionsbaustein ModbusRtuMasterV2_PcCOM [▶ 25] zur Verfügung.

Verbindung zur Hardware

Die zur Verknüpfung mit dem Kommunikations-Port notwendigen Datenstrukturen sind im Funktionsbaustein enthalten. Auf einem PC erfolgt die Zuweisung im TwinCAT System Manager analog zur Beschreibung im Kapitel Serielle Busklemme der Dokumentation TF6340 TC3 Serial Communication.

Der Baustein wird nicht in seiner Grundform aufgerufen, sondern es werden in einem SPS-Programm einzelne Aktionen des Funktionsbausteins verwendet. Jede Modbus-Funktion ist als Aktion realisiert.

Unterstützte Modbus-Funktionen (Aktionen)

- **ModbusMaster.ReadCoils**
  Modbus-Funktion 1 = *Read Coils*

  Liest binäre Ausgänge (Coils) von einem angeschlossenen Slave. Die Daten werden in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` abgelegt.
- **ModbusMaster.ReadInputStatus**
  Modbus-Funktion 2 = *Read Input Status*

  Liest binäre Eingänge von einem angeschlossenen Slave. Die Daten werden in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` abgelegt.
- **ModbusMaster.ReadRegs**
  Modbus-Funktion 3 = *Read Holding Registers*

Liest Daten von einem angeschlossenen Slave.

- **ModbusMaster.ReadInputRegs**
  Modbus-Funktion 4 = *Read Input Registers*

  Liest Eingangsregister von einem angeschlossenen Slave.
- **ModbusMaster.WriteSingleCoil**
  Modbus-Funktion 5 = *Write Single Coil*

  Sendet einen binären Ausgang (Coil) an einen angeschlossenen Slave. Die Daten müssen in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` zum Senden bereit liegen.
- **ModbusMaster.WriteSingleRegister**
  Modbus-Funktion 6 = *Write Single Register*

  Sendet ein einzelnes Datenwort an einen angeschlossenen Slave
- **ModbusMaster.WriteMultipleCoils**
  Modbus-Funktion 15 = *Write Multiple Coils*

  Sendet binäre Ausgänge (Coils) an einen angeschlossenen Slave. Die Daten müssen in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` zum Senden bereit liegen.
- **ModbusMaster.WriteRegs**
  Modbus-Funktion 16 = *Preset Multiple Registers*

  Sendet Daten an einen angeschlossenen Slave
- **ModbusMaster.Diagnostics**
  Modbus-Funktion 8 = *Diagnostics*

  Sendet eine Diagnoseanforderung mit einem von Anwender angegebenen Funktionscode (subfunction code) an den Slave. Da bei dieser Funktion kein Speicher adressiert wird, wird hier der Funktionscode im Datenwort `MBAddr` übergeben. Eventuelle für die Funktion notwendige Daten werden über `pMemoryAddr` mitgegeben.

Unterstützte Modbus-Funktionen (Aktionen) der MasterV2 Funktionsbausteine

- **ModbusMaster.ReadWriteRegs**
  Modbus-Funktion 23 = *Read/Write Multiple Registers*

  Sendet die über die Aux-Parmeter spezifizierten Daten an einen angeschlossenen Slave und empfängt gleichzeitig Daten von dem Slave. Die empfangenen Daten werden an der durch pMemoryAddr spezifizierten Adresse abgelegt.
- **ModbusMaster.UserReadWrite**
  Universelles Anwendertelegramm

  Der Modbus-Funktion Code wird vom Anwender im ersten Byte spezifizierten Daten (pMemoryAddr) angegeben. Der Anwender ist mit dieser Funktion in der Lage Modbus-Telegramm mit beliebigem Funktionscode zu senden. Evtl. vom Slave empfangene Daten werden an der durch pAuxMemoryAddr spezifizierten Adresse abgelegt.

Eingänge

```
VAR_INPUT
    UnitID      : BYTE;
    Quantity    : WORD;
    MBAddr      : WORD;
    cbLength    : UINT;
    pMemoryAddr : POINTER TO BYTE;

    AuxQuantity    : WORD;
    AuxMBAddr      : WORD;
    AuxcbLength    : UINT;
    pAuxMemoryAddr : POINTER TO BYTE;

    Execute     : BOOL;
    Timeout     : TIME;
END_VAR
```

Name	Typ	Beschreibung
UnitID	UINT	Modbus Stationsadresse [▶ 45] (1..247). Der Modbus-Slave antwortet nur, wenn er Telegramme mit seiner eigenen Stationsadresse empfängt. Optional können hier auch Sammeladressen eingestellt werden, um auf beliebige Anfragen zu antworten. Die Adresse 0 ist für Broadcast-Telegramme reserviert und somit keine gültige Stationsadresse (siehe UnitID [▶ 45])
Quantity	WORD	Anzahl der zu lesenden oder zu schreibenden Datenworte bei Wort-orientierten Modbus-Funktionen. Bei Bit-orientierten Modbus-Funktionen gibt Quantity die Anzahl der Bits (Inputs oder Coils) an.
MBAddr	WORD	Modbus-Datenadresse, von der die Daten aus dem Endgerät (Slave) gelesen werden. Diese Adresse wird unverändert zum Slave übertragen und wird dort als Datenadresse interpretiert. Bei der *Diagnostics*-Funktion (8) wird hier der Funktionscode (subfunction code) übergeben.
cbLength	UINT	Größe der verwendeten Datenvariable für Sende- oder Lese-Aktionen in Bytes. cbLength muss größer oder gleich der durch Quantity bestimmten übertragenen Datenmenge sein. Bei Wortzugriffen gilt z. B.: *[cbLength >= Quantity * 2]. cbLength* kann mit SIZEOF(ModbusDaten) berechnet werden.
pMemoryAddr	BYTE	Speicheradresse in der SPS, die mit ADR (ModbusDaten) berechnet wird. Bei Leseaktionen werden die gelesenen Daten in der adressierten Variablen abgelegt. Bei Sende-Aktionen werden die Daten aus der adressierten Variablen zum Endgerät übertragen.
AuxQuantity	WORD	Zusätzlicher Parameter, welcher nur bei Read/Write Funktionen verwendet wird, siehe ReadWriteRegs/ UserReadWrite [▶ 32]
AuxMBAddr	WORD	Zusätzlicher Parameter, welcher nur bei Read/Write Funktionen verwendet wird, siehe ReadWriteRegs/ UserReadWrite [▶ 32].
AuxcbLength	UINT	Zusätzlicher Parameter, welcher nur bei Read/Write Funktionen verwendet wird, siehe ReadWriteRegs/ UserReadWrite [▶ 32]
pAuxMemoryAddr	BYTE	Zusätzlicher Parameter, welcher nur bei Read/Write Funktionen verwendet wird, siehe ReadWriteRegs/ UserReadWrite [▶ 32]
Execute	BOOL	Startsignal. Mit steigender Flanke am Eingang Execute wird die Aktion ausgelöst.
Timeout	TIME	Timeout-Wert für das Warten auf eine Reaktion des angesprochenen Slaves.

Ausgänge

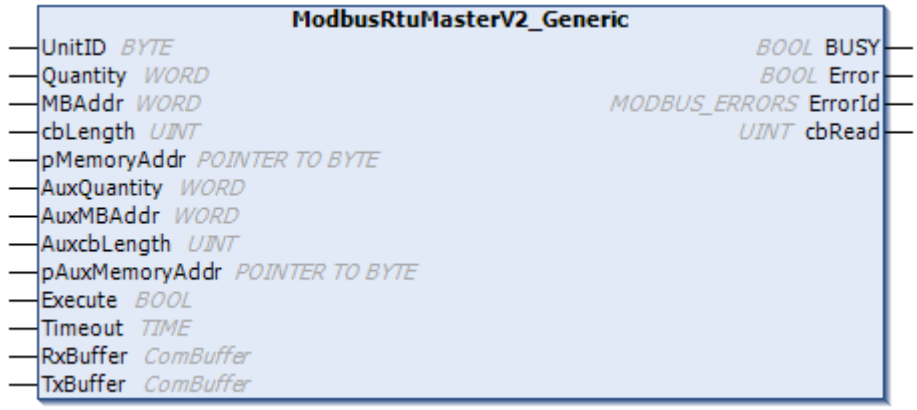
```
VAR_OUTPUT
    BUSY    : BOOL;
    Error   : BOOL;
    ErrorId : MODBUS_ERRORS;
    cbRead  : UINT;
END_VAR
```

Name	Typ	Beschreibung
Busy	BOOL	Zeigt an, dass der Funktionsbaustein aktiv ist. *Busy* wird mit steigender Flanke an *Execute* TRUE und wird wieder FALSE nachdem die gestartete Aktion beendet ist. Es kann immer nur eine Aktion gleichzeitig aktiv sein.
Error	BOOL	Zeigt an, dass bei der Bearbeitung einer Aktion ein Fehler aufgetreten ist.
ErrorId	MODBUS_ERRORS	Zeigt eine Fehlernummer [▶ 46] im Falle einer gestörten oder fehlerhaften Kommunikation an.
cbRead	UINT	Liefert die Anzahl der gelesenen Datenbytes bei einer Lese-Aktion.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_ModbusRTU

5.1.5 ModbusRtuMasterV2_Generic



Der Funktionsbaustein `ModbusRtuMasterV2_Generic` realisiert einen Modbus-Master, der über verschiedenste serielle Schnittstellen (COM-Port, virtueller COM-Port, EtherCAT-Klemmen, ...) kommuniziert.

Aufgrund der Hardware-unabhängigen Eigenschaft des `ModbusRtuMasterV2_Generic` ist die Verwendung etwas aufwendiger als bei den Hardware-abhängigen Funktionsbausteinen `ModbusRtuMasterV2_PcCOM`, `ModbusRtuMasterV2_KL6x22B`, `ModbusRtuMasterV2_ KL6x5B`. Alle Funktionsbausteinen bieten die gleiche ModbusRTU Funktionalität. Allein der `ModbusRtuMasterV2_Generic` ermöglicht jedoch die Verwendung von virtuellen COM-Ports.

Verbindung zur Hardware mittels TF6340 TC3 Serial Communication (Lizenz notwendig)

Die zur Verknüpfung mit dem Kommunikations-Port notwendigen Datenstrukturen müssen separat instanziiert werden. Die an diesem Funktionsbaustein vorhandenen Datenstrukturen vom Typ ComBuffer sind Datenpuffer zur Entkopplung der Hardware-abhängigen Hintergrundkommunikation. Diese Hintergrundkommunikation muss über entsprechende Funktionsbausteine (SerialLineControl, SerialLineControlADS) der Tc2_SerialCom SPS Bibliothek realisiert werden, wozu diese SPS Bibliothek zuerst im Programm eingebunden werden muss. Hiermit wird auch die Lizenz für TF6340 TC3 Serial Communication zusätzlich notwendig.

Der Baustein wird nicht in seiner Grundform aufgerufen, sondern es werden in einem SPS-Programm einzelne Aktionen des Funktionsbausteins verwendet. Jede Modbus-Funktion ist als Aktion realisiert.

Unterstützte Modbus-Funktionen (Aktionen)

- **ModbusMaster.ReadCoils**
  Modbus-Funktion 1 = *Read Coils*

  Liest binäre Ausgänge (Coils) von einem angeschlossenen Slave. Die Daten werden in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` abgelegt.
- **ModbusMaster.ReadInputStatus**
  Modbus-Funktion 2 = *Read Input Status*

  Liest binäre Eingänge von einem angeschlossenen Slave. Die Daten werden in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` abgelegt.
- **ModbusMaster.ReadRegs**
  Modbus-Funktion 3 = *Read Holding Registers*

  Liest Daten von einem angeschlossenen Slave.
- **ModbusMaster.ReadInputRegs**
  Modbus-Funktion 4 = *Read Input Registers*

  Liest Eingangsregister von einem angeschlossenen Slave.
- **ModbusMaster.WriteSingleCoil**
  Modbus-Funktion 5 = *Write Single Coil*

  Sendet einen binären Ausgang (Coil) an einen angeschlossenen Slave. Die Daten müssen in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` zum Senden bereit liegen.
- **ModbusMaster.WriteSingleRegister**
  Modbus-Funktion 6 = *Write Single Register*

  Sendet ein einzelnes Datenwort an einen angeschlossenen Slave
- **ModbusMaster.WriteMultipleCoils**
  Modbus-Funktion 15 = *Write Multiple Coils*

  Sendet binäre Ausgänge (Coils) an einen angeschlossenen Slave. Die Daten müssen in komprimierter Form (8 Bit pro Byte) ab der angegebenen Adresse `pMemoryAddr` zum Senden bereit liegen.
- **ModbusMaster.WriteRegs**
  Modbus-Funktion 16 = *Preset Multiple Registers*

  Sendet Daten an einen angeschlossenen Slave
- **ModbusMaster.Diagnostics**
  Modbus-Funktion 8 = *Diagnostics*

  Sendet eine Diagnoseanforderung mit einem von Anwender angegebenen Funktionscode (subfunction code) an den Slave. Da bei dieser Funktion kein Speicher adressiert wird, wird hier der Funktionscode im Datenwort `MBAddr` übergeben. Eventuelle für die Funktion notwendige Daten werden über `pMemoryAddr` mitgegeben.

Unterstützte Modbus-Funktionen (Aktionen) der MasterV2 Funktionsbausteine

- **ModbusMaster.ReadWriteRegs**
  Modbus-Funktion 23 = *Read/Write Multiple Registers*

  Sendet die über die Aux-Parmeter spezifizierten Daten an einen angeschlossenen Slave und empfängt gleichzeitig Daten von dem Slave. Die empfangenen Daten werden an der durch pMemoryAddr spezifizierten Adresse abgelegt.
- **ModbusMaster.UserReadWrite**
  Universelles Anwendertelegramm

  Der Modbus-Funktion Code wird vom Anwender im ersten Byte spezifizierten Daten (pMemoryAddr) angegeben. Der Anwender ist mit dieser Funktion in der Lage Modbus-Telegramm mit beliebigem Funktionscode zu senden. Evtl. vom Slave empfangene Daten werden an der durch pAuxMemoryAddr spezifizierten Adresse abgelegt.

Eingänge

```
VAR_INPUT
    UnitID       : BYTE;
    Quantity     : WORD;
    MBAddr       : WORD;
    cbLength     : UINT;
    pMemoryAddr : POINTER TO BYTE;

    AuxQuantity     : WORD;
    AuxMBAddr       : WORD;
    AuxcbLength     : UINT;
    pAuxMemoryAddr : POINTER TO BYTE;

    Execute      : BOOL;
    Timeout      : TIME;
END_VAR
```

Name	Typ	Beschreibung
UnitID	UINT	Modbus Stationsadresse [▶ 45] (1..247). Der Modbus-Slave antwortet nur, wenn er Telegramme mit seiner eigenen Stationsadresse empfängt. Optional können hier auch Sammeladressen eingestellt werden, um auf beliebige Anfragen zu antworten. Die Adresse 0 ist für Broadcast-Telegramme reserviert und somit keine gültige Stationsadresse (siehe UnitID [▶ 45])
Quantity	WORD	Anzahl der zu lesenden oder zu schreibenden Datenworte bei Wort-orientierten Modbus-Funktionen. Bei Bit-orientierten Modbus-Funktionen gibt Quantity die Anzahl der Bits (Inputs oder Coils) an.
MBAddr	WORD	Modbus-Datenadresse, von der die Daten aus dem Endgerät (Slave) gelesen werden. Diese Adresse wird unverändert zum Slave übertragen und wird dort als Datenadresse interpretiert. Bei der *Diagnostics*-Funktion (8) wird hier der Funktionscode (subfunction code) übergeben.
cbLength	UINT	Größe der verwendeten Datenvariable für Sende- oder Lese-Aktionen in Bytes. cbLength muss größer oder gleich der durch Quantity bestimmten übertragenen Datenmenge sein. Bei Wortzugriffen gilt z. B.: *[cbLength >= Quantity * 2]. cbLength* kann mit SIZEOF(ModbusDaten) berechnet werden.
pMemoryAddr	BYTE	Speicheradresse in der SPS, die mit ADR (ModbusDaten) berechnet wird. Bei Leseaktionen werden die gelesenen Daten in der adressierten Variablen abgelegt. Bei Sende-Aktionen werden die Daten aus der adressierten Variablen zum Endgerät übertragen.
AuxQuantity	WORD	Zusätzlicher Parameter, welcher nur bei Read/Write Funktionen verwendet wird, siehe ReadWriteRegs/ UserReadWrite [▶ 35]
AuxMBAddr	WORD	Zusätzlicher Parameter, welcher nur bei Read/Write Funktionen verwendet wird, siehe ReadWriteRegs/ UserReadWrite [▶ 35].
AuxcbLength	UINT	Zusätzlicher Parameter, welcher nur bei Read/Write Funktionen verwendet wird, siehe ReadWriteRegs/ UserReadWrite [▶ 35]
pAuxMemoryAddr	BYTE	Zusätzlicher Parameter, welcher nur bei Read/Write Funktionen verwendet wird, siehe ReadWriteRegs/ UserReadWrite [▶ 35]
Execute	BOOL	Startsignal. Mit steigender Flanke am Eingang Execute wird die Aktion ausgelöst.
Timeout	TIME	Timeout-Wert für das Warten auf eine Reaktion des angesprochenen Slaves.

Ein-/Ausgänge

```
VAR_IN_OUT
    RxBuffer    : ComBuffer;
    TxBuffer    : ComBuffer;
END_VAR
```

Name	Typ	Beschreibung
TxBuffer	ComBuffer (Tc2_SerialCom SPS Bibliothek)	Puffer mit Sendedaten für die verwendete serielle Hardware. Dieser Datenpuffer wird vom Anwender niemals direkt beschrieben oder gelesen, sondern dient nur als Zwischenspeicher für die Kommunikationsbausteine. Die Hintergrundkommunikation muss über entsprechende Funktionsbausteine der Tc2_SerialCom SPS Bibliothek realisiert werden.
RxBuffer	ComBuffer (Tc2_SerialCom SPS Bibliothek)	Puffer in dem die Empfangsdaten abgelegt werden. Dieser Datenpuffer wird vom Anwender niemals direkt beschrieben oder gelesen, sondern dient nur als Zwischenspeicher für die Kommunikationsbausteine. Die Hintergrundkommunikation muss über entsprechende Funktionsbausteine der Tc2_SerialCom SPS Bibliothek realisiert werden.

Ausgänge

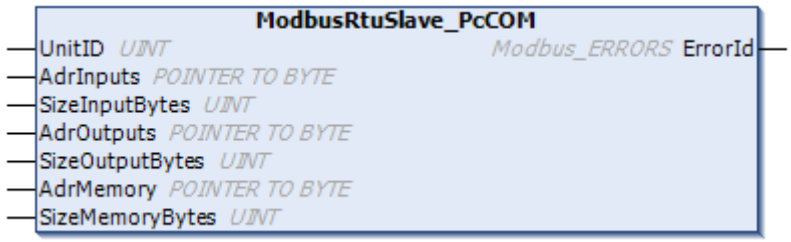
```
VAR_OUTPUT
    BUSY    : BOOL;
    Error   : BOOL;
    ErrorId : MODBUS_ERRORS;
    cbRead  : UINT;
END_VAR
```

Name	Typ	Beschreibung
Busy	BOOL	Zeigt an, dass der Funktionsbaustein aktiv ist. *Busy* wird mit steigender Flanke an *Execute* TRUE und wird wieder FALSE nachdem die gestartete Aktion beendet ist. Es kann immer nur eine Aktion gleichzeitig aktiv sein.
Error	BOOL	Zeigt an, dass bei der Bearbeitung einer Aktion ein Fehler aufgetreten ist.
ErrorId	MODBUS_ERRORS	Zeigt eine Fehlernummer [▶ 46] im Falle einer gestörten oder fehlerhaften Kommunikation an.
cbRead	UINT	Liefert die Anzahl der gelesenen Datenbytes bei einer Lese-Aktion.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4024.0	PC oder CX (x86, x64, ARM)	Tc2_ModbusRTU (>= v3.5.6.0)

5.1.6 ModbusRtuSlave_PcCOM



Der Funktionsbaustein `ModbusRtuSlave_PcCOM` realisiert einen Modbus-Slave, der über eine serielle PC-Schnittstelle (COM-Port) kommuniziert. Zur Kommunikation über eine serielle Busklemme KL6001, KL6011 oder KL6021 steht der Funktionsbaustein ModbusRtuSlave_KL6x5B [▶ 41] zur Verfügung.

Der Baustein verhält sich passiv, bis er von einem angeschlossenen Modbus-Master Telegramme empfängt. Ein Beispielprogramm verdeutlicht die Funktionsweise.

Verbindung zur Hardware

Die zur Verknüpfung mit dem Kommunikations-Port notwendigen Datenstrukturen sind im Funktionsbaustein enthalten. Sie werden im TwinCAT System Manager angezeigt nachdem das SPS-Programm eingebunden ist und können dort mit einem COM-Port verbunden werden. Die Vorgehensweise ist analog zur Beschreibung im Kapitel Serielle PC-Schnittstelle der Dokumentation TF6340 TC3 Serial Communication.

Eingänge

```
VAR_INPUT
    UnitID          : UINT;
    AdrInputs       : POINTER TO BYTE; (* Pointer to the Modbus input area *)
    SizeInputBytes  : UINT;
    AdrOutputs      : POINTER TO BYTE; (* Pointer to the Modbus output area *)
    SizeOutputBytes : UINT;
    AdrMemory       : POINTER TO BYTE; (* Pointer to the Modbus memory area *)
    SizeMemoryBytes : UINT;
END_VAR
```

Name	Typ	Beschreibung
UnitID	UINT	Modbus Stationsadresse [▶ 45] (1..247). Der Modbus-Slave antwortet nur, wenn er Telegramme mit seiner eigenen Stationsadresse empfängt. Optional können hier auch Sammeladressen eingestellt werden, um auf beliebige Anfragen zu antworten. Die Adresse 0 ist für Broadcast-Telegramme reserviert und somit keine gültige Stationsadresse.
AdrInputs	BYTE	Startadresse des Modbus-Input-Bereiches [▶ 15]. Der Datenbereich wird üblicherweise als SPS-Array deklariert und die Adresse kann mit ADR(Input-Variable) berechnet werden.
SizeInputBytes	UINT	Größe des Modbus-Input-Bereiches in Bytes. Die Größe kann mit SIZEOF(Input-Variable) berechnet werden.
AdrOutputs	BYTE	Startadresse des Modbus-Output-Bereiches [▶ 15]. Der Datenbereich wird üblicherweise als SPS-Array deklariert und die Adresse kann mit ADR(Output-Variable) berechnet werden.
SizeOutputBytes	UINT	Größe des Modbus-Output-Bereiches in Bytes. Die Größe kann mit SIZEOF(Output-Variable) berechnet werden.
AdrMemory	BYTE	Startadresse des Modbus-Memory-Bereiches [▶ 15]. Der Datenbereich wird üblicherweise als SPS-Array deklariert und die Adresse kann mit ADR(Memory-Variable) berechnet werden.
SizeMemoryBytes	UINT	Größe des Modbus-Memory-Bereiches in Bytes. Die Größe kann mit SIZEOF(Memory-Variable) berechnet werden.

Ausgänge

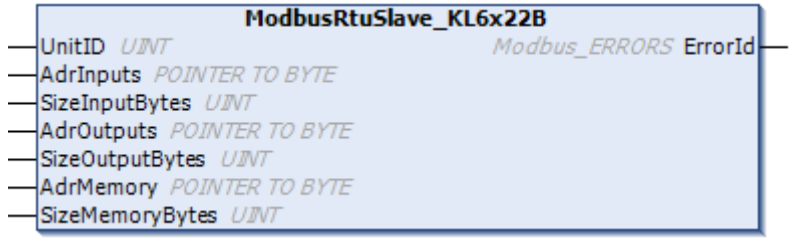
```
VAR_OUTPUT
    ErrorId : MODBUS_ERRORS;
END_VAR
```

Name	Typ	Beschreibung
ErrorId	MODBUS_ERRORS	Zeigt eine Fehlernummer [▶ 46] im Falle einer gestörten oder fehlerhaften Kommunikation an.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_ModbusRTU

5.1.7 ModbusRtuSlave_KL6x22B



Der Funktionsbaustein `ModbusRtuSlave_KL6x22B` realisiert einen Modbus-Slave, der über eine serielle Busklemme KL6031 oder KL6041 kommuniziert. Es werden ebenso serielle EtherCAT-Klemmen mit 22 Bytes Daten Prozessabbild unterstützt. Zur Kommunikation über eine serielle PC-Schnittstelle (COM-Port) steht der Funktionsbaustein ModbusRtuSlave_PcCOM [▶ 38] zur Verfügung.

Der Baustein verhält sich passiv, bis er von einem angeschlossenen Modbus-Master Telegramme empfängt.

> **Verbindung zur Hardware**
>
> Die zur Verknüpfung mit dem Kommunikations-Port notwendigen Datenstrukturen sind im Funktionsbaustein enthalten. Auf einem PC erfolgt die Zuweisung im TwinCAT System Manager analog zur Beschreibung im Kapitel Serielle Busklemme der Dokumentation TF6340 TC3 Serial Communication.

Eingänge

```
VAR_INPUT
    UnitID          : UINT;
    AdrInputs       : POINTER TO BYTE; (* Pointer to the Modbus input area *)
    SizeInputBytes  : UINT;
    AdrOutputs      : POINTER TO BYTE; (* Pointer to the Modbus output area *)
    SizeOutputBytes : UINT;
    AdrMemory       : POINTER TO BYTE; (* Pointer to the Modbus memory area *)
    SizeMemoryBytes : UINT;
END_VAR
```

Name	Typ	Beschreibung
UnitID	UINT	Modbus Stationsadresse [▶ 45] (1..247). Der Modbus-Slave antwortet nur, wenn er Telegramme mit seiner eigenen Stationsadresse empfängt. Optional können hier auch Sammeladressen eingestellt werden, um auf beliebige Anfragen zu antworten. Die Adresse 0 ist für Broadcast-Telegramme reserviert und somit keine gültige Stationsadresse.
AdrInputs	BYTE	Startadresse des Modbus-Input-Bereiches [▶ 15]. Der Datenbereich wird üblicherweise als SPS-Array deklariert und die Adresse kann mit ADR(Input-Variable) berechnet werden.
SizeInputBytes	UINT	Größe des Modbus-Input-Bereiches in Bytes. Die Größe kann mit SIZEOF(Input-Variable) berechnet werden.
AdrOutputs	BYTE	Startadresse des Modbus-Output-Bereiches [▶ 15]. Der Datenbereich wird üblicherweise als SPS-Array deklariert und die Adresse kann mit ADR(Output-Variable) berechnet werden.
SizeOutputBytes	UINT	Größe des Modbus-Output-Bereiches in Bytes. Die Größe kann mit SIZEOF(Output-Variable) berechnet werden.
AdrMemory	BYTE	Startadresse des Modbus-Memory-Bereiches [▶ 15]. Der Datenbereich wird üblicherweise als SPS-Array deklariert und die Adresse kann mit ADR(Memory-Variable) berechnet werden.
SizeMemoryBytes	UINT	Größe des Modbus-Memory-Bereiches in Bytes. Die Größe kann mit SIZEOF(Memory-Variable) berechnet werden.

Ausgänge

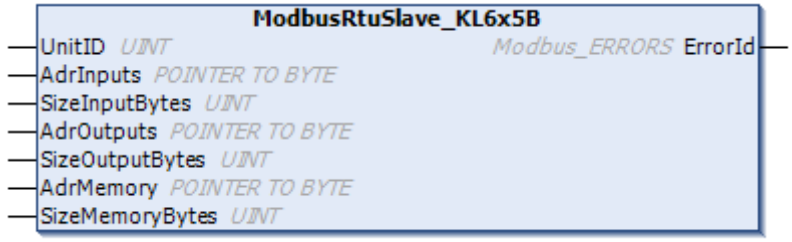
```
VAR_OUTPUT
    ErrorId : MODBUS_ERRORS;
END_VAR
```

Name	Typ	Beschreibung
ErrorId	MODBUS_ERRORS	Zeigt eine Fehlernummer [▶ 46] im Falle einer gestörten oder fehlerhaften Kommunikation an.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_ModbusRTU

5.1.8 ModbusRtuSlave_KL6x5B



Der Funktionsbaustein `ModbusRTUslave_KL6x5B` realisiert einen Modbus-Slave, der über eine serielle Busklemme KL6001, KL6011 oder KL6021 kommuniziert. Zur Kommunikation über eine serielle PC-Schnittstelle (COM-Port) steht der Funktionsbaustein ModbusRtuSlave_PcCOM [▶ 38] zur Verfügung.

Der Baustein verhält sich passiv, bis er von einem angeschlossenen Modbus-Master Telegramme empfängt.

Verbindung zur Hardware

Die zur Verknüpfung mit dem Kommunikations-Port notwendigen Datenstrukturen sind im Funktionsbaustein enthalten. Auf einem PC erfolgt die Zuweisung im TwinCAT System Manager analog zur Beschreibung im Kapitel Serielle Busklemme der Dokumentation TF6340 TC3 Serial Communication.

Eingänge

```
VAR_INPUT
    UnitID          : UINT;
    AdrInputs       : POINTER TO BYTE; (* Pointer to the Modbus input area *)
    SizeInputBytes  : UINT;
    AdrOutputs      : POINTER TO BYTE; (* Pointer to the Modbus output area *)
    SizeOutputBytes : UINT;
    AdrMemory       : POINTER TO BYTE; (* Pointer to the Modbus memory area *)
    SizeMemoryBytes : UINT;
END_VAR
```

Name	Typ	Beschreibung
UnitID	UINT	Modbus Stationsadresse [▶ 45] (1..247). Der Modbus-Slave antwortet nur, wenn er Telegramme mit seiner eigenen Stationsadresse empfängt. Optional können hier auch Sammeladressen eingestellt werden, um auf beliebige Anfragen zu antworten. Die Adresse 0 ist für Broadcast-Telegramme reserviert und somit keine gültige Stationsadresse.
AdrInputs	BYTE	Startadresse des Modbus-Input-Bereiches [▶ 15]. Der Datenbereich wird üblicherweise als SPS-Array deklariert und die Adresse kann mit ADR(Input-Variable) berechnet werden.
SizeInputBytes	UINT	Größe des Modbus-Input-Bereiches in Bytes. Die Größe kann mit SIZEOF(Input-Variable) berechnet werden.
AdrOutputs	BYTE	Startadresse des Modbus-Output-Bereiches [▶ 15]. Der Datenbereich wird üblicherweise als SPS-Array deklariert und die Adresse kann mit ADR(Output-Variable) berechnet werden.
SizeOutputBytes	UINT	Größe des Modbus-Output-Bereiches in Bytes. Die Größe kann mit SIZEOF(Output-Variable) berechnet werden.
AdrMemory	BYTE	Startadresse des Modbus-Memory-Bereiches [▶ 15]. Der Datenbereich wird üblicherweise als SPS-Array deklariert und die Adresse kann mit ADR(Memory-Variable) berechnet werden.
SizeMemoryBytes	UINT	Größe des Modbus-Memory-Bereiches in Bytes. Die Größe kann mit SIZEOF(Memory-Variable) berechnet werden.

Ausgänge

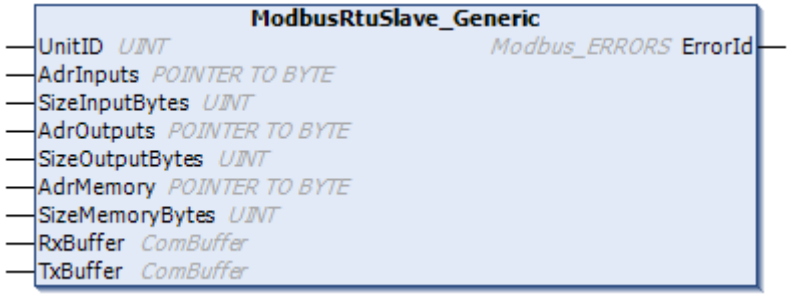
```
VAR_OUTPUT
    ErrorId : MODBUS_ERRORS;
END_VAR
```

Name	Typ	Beschreibung
ErrorId	MODBUS_ERRORS	Zeigt eine Fehlernummer [▶ 46] im Falle einer gestörten oder fehlerhaften Kommunikation an.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_ModbusRTU

5.1.9 ModbusRtuSlave_Generic



Der Funktionsbaustein `ModbusRtuSlave_Generic` realisiert einen Modbus-Slave, der über verschiedenste serielle Schnittstellen (COM-Port, virtueller COM-Port, EtherCAT-Klemmen, ...) kommuniziert.

Aufgrund der Hardware-unabhängigen Eigenschaft des `ModbusRtuSlave_Generic` ist die Verwendung etwas aufwendiger als bei den Hardware-abhängigen Funktionsbausteinen `ModbusRtuSlave_PcCOM`, `ModbusRtuSlave_KL6x22B`, `ModbusRtuSlave_ KL6x5B`. Alle Funktionsbausteinen bieten die gleiche ModbusRTU Funktionalität. Allein der `ModbusRtuSlave_Generic` ermöglicht jedoch die Verwendung von virtuellen COM-Ports.

Der Baustein verhält sich passiv, bis er von einem angeschlossenen Modbus-Master Telegramme empfängt. Ein Beispielprogramm verdeutlicht die Funktionsweise.

Verbindung zur Hardware mittels TF6340 TC3 Serial Communication (Lizenz notwendig)

Die zur Verknüpfung mit dem Kommunikations-Port notwendigen Datenstrukturen müssen separat instanziiert werden. Die an diesem Funktionsbaustein vorhandenen Datenstrukturen vom Typ ComBuffer sind Datenpuffer zur Entkopplung der Hardware-abhängigen Hintergrundkommunikation. Diese Hintergrundkommunikation muss über entsprechende Funktionsbausteine (SerialLineControl, SerialLineControlADS) der Tc2_SerialCom SPS Bibliothek realisiert werden, wozu diese SPS Bibliothek zuerst im Programm eingebunden werden muss. Hiermit wird auch die Lizenz für TF6340 TC3 Serial Communication zusätzlich notwendig.

Eingänge

```
VAR_INPUT
    UnitID          : UINT;
    AdrInputs       : POINTER TO BYTE; (* Pointer to the Modbus input area *)
    SizeInputBytes  : UINT;
    AdrOutputs      : POINTER TO BYTE; (* Pointer to the Modbus output area *)
    SizeOutputBytes : UINT;
    AdrMemory       : POINTER TO BYTE; (* Pointer to the Modbus memory area *)
    SizeMemoryBytes : UINT;
END_VAR
```

Name	Typ	Beschreibung
UnitID	UINT	Modbus Stationsadresse [▶ 45] (1..247). Der Modbus-Slave antwortet nur, wenn er Telegramme mit seiner eigenen Stationsadresse empfängt. Optional können hier auch Sammeladressen eingestellt werden, um auf beliebige Anfragen zu antworten. Die Adresse 0 ist für Broadcast-Telegramme reserviert und somit keine gültige Stationsadresse.
AdrInputs	BYTE	Startadresse des Modbus-Input-Bereiches [▶ 15]. Der Datenbereich wird üblicherweise als SPS-Array deklariert und die Adresse kann mit ADR(Input-Variable) berechnet werden.
SizeInputBytes	UINT	Größe des Modbus-Input-Bereiches in Bytes. Die Größe kann mit SIZEOF(Input-Variable) berechnet werden.
AdrOutputs	BYTE	Startadresse des Modbus-Output-Bereiches [▶ 15]. Der Datenbereich wird üblicherweise als SPS-Array deklariert und die Adresse kann mit ADR(Output-Variable) berechnet werden.
SizeOutputBytes	UINT	Größe des Modbus-Output-Bereiches in Bytes. Die Größe kann mit SIZEOF(Output-Variable) berechnet werden.
AdrMemory	BYTE	Startadresse des Modbus-Memory-Bereiches [▶ 15]. Der Datenbereich wird üblicherweise als SPS-Array deklariert und die Adresse kann mit ADR(Memory-Variable) berechnet werden.
SizeMemoryBytes	UINT	Größe des Modbus-Memory-Bereiches in Bytes. Die Größe kann mit SIZEOF(Memory-Variable) berechnet werden.

Ein-/Ausgänge

```
VAR_IN_OUT
    RxBuffer    : ComBuffer;
    TxBuffer    : ComBuffer;
END_VAR
```

Name	Typ	Beschreibung
TxBuffer	ComBuffer (Tc2_SerialCom SPS Bibliothek)	Puffer mit Sendedaten für die verwendete serielle Hardware. Dieser Datenpuffer wird vom Anwender niemals direkt beschrieben oder gelesen, sondern dient nur als Zwischenspeicher für die Kommunikationsbausteine. Die Hintergrundkommunikation muss über entsprechende Funktionsbausteine der Tc2_SerialCom SPS Bibliothek realisiert werden.
RxBuffer	ComBuffer (Tc2_SerialCom SPS Bibliothek)	Puffer in dem die Empfangsdaten abgelegt werden. Dieser Datenpuffer wird vom Anwender niemals direkt beschrieben oder gelesen, sondern dient nur als Zwischenspeicher für die Kommunikationsbausteine. Die Hintergrundkommunikation muss über entsprechende Funktionsbausteine der Tc2_SerialCom SPS Bibliothek realisiert werden.

Ausgänge

```
VAR_OUTPUT
    ErrorId : MODBUS_ERRORS;
END_VAR
```

Name	Typ	Beschreibung
ErrorId	MODBUS_ERRORS	Zeigt eine Fehlernummer [▶ 46] im Falle einer gestörten oder fehlerhaften Kommunikation an.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4024.0	PC oder CX (x86, x64, ARM)	Tc2_ModbusRTU (>= v3.5.6.0)

5.2 Datentypen

5.2.1 Modbus Stationsadresse

Modbus definiert gültige Stationsadressen im Bereich 1 bis 247. Ein Modbus-Slave antwortet nur auf Telegramme, die seine eigene Adresse enthalten. Die Adresse 0 ist keine gültige Stationsadresse, sondern wird für Broadcast-Telegramme an alle Stationen verwendet, die nicht beantwortet werden. Die Adressen 248 bis 255 sind reserviert.

Die Bibliothek Tc2_ModbusRTU definiert weitere Sammeladressen. Dadurch wird es möglich, eine Station auf mehrere Adressen antworten zu lassen.

```
TYPE MODBUS_UNITID :
(
    MODBUS_UNITID_BROADCAST       := 0,
    MODBUS_UNITID_ALLVALID        := 256, (* response on address 1..247 *)
    MODBUS_UNITID_ALLBUTBROADCAST := 257, (* response on address 1..255 *)
    MODBUS_UNITID_ALL             := 258 (* response on address 0..255 *)
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_ModbusRTU

5.3 Globale Konstanten

5.3.1 Global_Version

Alle Bibliotheken haben eine bestimmte Version. Diese Version ist u. a. im SPS-Bibliotheks-Repository zu sehen. Eine globale Konstante enthält die Information über die Bibliotheksversion:

```
VAR_GLOBAL CONSTANT
stLibVersion_Tc2_Modbus_RTU : ST_LibVersion;
END_VAR
```

Um zu sehen, ob die Version die Sie haben auch die Version ist die Sie brauchen, benutzen Sie die Funktion F_CmpLibVersion (definiert in der Tc2_System SPS Bibliothek).

Alle anderen Möglichkeiten Bibliotheksversionen zu vergleichen, die Sie von TwinCAT 2 kennen, sind veraltet!

6 Anhang

6.1 Modbus RTU Fehlernummern

```
TYPE MODBUS_ERRORS :
(
(* Modbus communication errors *)
MODBUSERROR_NO_ERROR, (* 0 *)
MODBUSERROR_ILLEGAL_FUNCTION, (* 1 *)
MODBUSERROR_ILLEGAL_DATA_ADDRESS, (* 2 *)
MODBUSERROR_ILLEGAL_DATA_VALUE, (* 3 *)
MODBUSERROR_SLAVE_DEVICE_FAILURE, (* 4 *)
MODBUSERROR_ACKNOWLEDGE, (* 5 *)
MODBUSERROR_SLAVE_DEVICE_BUSY, (* 6 *)
MODBUSERROR_NEGATIVE_ACKNOWLEDGE, (* 7 *)
MODBUSERROR_MEMORY_PARITY, (* 8 *)
MODBUSERROR_GATEWAY_PATH_UNAVAILABLE, (* A *)
MODBUSERROR_GATEWAY_TARGET_DEVICE_FAILED_TO_RESPOND, (* B *)

(* additional Modbus error definitions *)
MODBUSERROR_CHARREC_TIMEOUT := 16#20, (* 20 hex *)
MODBUSERROR_ILLEGAL_DATA_SIZE, (* 21 hex *)
MODBUSERROR_ILLEGAL_DEVICE_ADDRESS, (* 22 hex *)
MODBUSERROR_ILLEGAL_DESTINATION_ADDRESS, (* 23 hex *)
MODBUSERROR_ILLEGAL_DESTINATION_SIZE, (* 24 hex *)
MODBUSERROR_NO_RESPONSE, (* 25 hex *)

(* Low level communication errors *)
MODBUSERROR_TXBUFFOVERRUN := 102, (* 102 *)
MODBUSERROR_SENDTIMEOUT := 103, (* 103 *)
MODBUSERROR_DATASIZEOVERRUN := 107, (* 107 *)
MODBUSERROR_STRINGOVERRUN := 110, (* 110 *)
MODBUSERROR_INVALIDPOINTER := 120, (* 120 *)
MODBUSERROR_CRC := 150, (* 150 *)

(* High level PLC errors *)
MODBUSERROR_INVALIDMEMORYADDRESS := 232, (* 232 *)
MODBUSERROR_TRANSMITBUFFERTOOSMALL (* 233 *)
);
END_TYPE
```

Mehr Informationen:
www.beckhoff.com/tf6255

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