BECKHOFF New Automation Technology

Handbuch | DE

TS6340

TwinCAT 2 PLC Serial Communication

Supplement | Communication

20.10.2022 | 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, für jede Installation und Inbetriebnahme die zu dem betreffenden Zeitpunkt veröffentliche Dokumentation zu verwenden.

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

Disclaimer

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

Marken

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

Patente

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

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

Copyright

© Beckhoff Automation GmbH & Co. KG, Deutschland.
Weitergabe sowie Vervielfältigung dieses Dokuments, Verwertung und Mitteilung seines Inhalts sind verboten, soweit nicht ausdrücklich gestattet.
Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patent-, Gebrauchsmuster- oder Geschmacksmustereintragung vorbehalten.

1.2 Sicherheitshinweise

Sicherheitsbestimmungen

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

Haftungsausschluss

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

Qualifikation des Personals

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

Erklärung der Symbole

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

⚠ GEFAHR

Akute Verletzungsgefahr!

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

⚠ WARNUNG

Verletzungsgefahr!

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

⚠ VORSICHT

Schädigung von Personen!

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

HINWEIS

Schädigung von Umwelt oder Geräten

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

Tipp oder Fingerzeig

Dieses Symbol kennzeichnet Informationen, die zum besseren Verständnis beitragen.

1.3 Hinweise zur Informationssicherheit

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

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

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

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

2 Übersicht

Die TwinCAT SPS Bibliothek COMlibV2 bietet Funktionsbausteine und Datenstrukturen für die serielle Datenkommunikation. Die Bibliothek unterstützt die seriellen Beckhoff Busklemmen KL6xxx, EtherCAT Klemmen EL60xx, die Standard PC-Schnittstellen COMx und virtuelle COM Ports.

Im Einzelnen bietet die Bibliothek Bausteine für folgende Funktionen:

- Byteweises senden und empfangen
- Senden und empfangen von Strings
- Senden und empfangen von transparenten Daten
- Konfiguration der seriellen Busklemmen KL6xxx (Baudrate, Handshake etc.)
- Bedienung der seriellen Busklemmen
- Bedienung der seriellen EtherCAT Klemmen

Alle Bausteine sind mehrfach instanzierbar, so dass je nach Leistung des PC beliebig viele serielle Schnittstellen parallel bedient werden können.

Der Vorgänger der TwinCAT SPS Bibliothek COMlibV2 war die TwinCAT SPS Bibliothek COMlib.

Systemvoraussetzungen bei Verwendung mit virtuellen COM Ports

mit Installation >= v2.2.4 verfügbar

Die Installation des TwinCAT PLC Serial Communication Produktes bietet optional die Möglichkeit einen ADS Server zu installieren. Dies ist nur nötig, falls auf virtuelle COM Ports zugegriffen werden soll. Andernfalls muss der Server nicht installiert werden.

Für die Verwendung der SPS Bibliothek mit virtuellen COM Ports in Windows gelten spezielle Systemvoraussetzungen:

- Programmierumgebung:
  - XP, XPe, WES, Win7, WES7, Win10
  - TwinCAT Installation Level: TwinCAT PLC oder höher
  - TwinCAT System Version 2.11 Build 1544 oder höher
  - **COMlibV2.lib** Diese SPS Bibliothek muss in dem SPS-Projekt eingebunden werden. (Die Bibliotheken Standard.lib; TcBase.lib; TcSystem.lib werden automatisch ins Projekt eingebunden.)
    Bis zur Version < v2.3.0 der Bibliothek COMlibV2.lib wurde nur die Bibliothek Standard.lib eingebunden. Es ist nun sicherzustellen, dass alle drei Bibliotheken verfügbar sind!
- Zielplattform:
  - PC oder CX (x86, x64): XP, XPe, WES, Win7, WES7, Win10, CE 6.0
  - TwinCAT SPS-Laufzeitsystem Version 2.10 Build 1340 oder höher
- Installation:
  - Die Produktinstallation bietet optional die Möglichkeit der Installation des TcAdsSerialCommServer.
  - Der ADS Server muss auf dem System installiert werden, auf dem sich der virtuelle COM Port befindet.
  - Um den ADS Server auf einem Windows CE System zu installieren müssen folgende Schritte ausgeführt werden:

Installieren Sie das Produkt zuerst wie gewohnt auf Ihrem Programmier-PC.

Nach der Installation finden Sie im Ordner: **...\TwinCAT\CE\TcPLCSerialComm** ein Cabinet-File für das CE-Laufzeitsystem.

Kopieren Sie das Cabinet-File in einen Ordner auf dem CE-Laufzeitsystem.

Auf dem CE-System: Installieren Sie (durch einen Doppelklick auf das das Cabinet-File) die CE-Komponenten nach '\Hard Disk\System'.

Starten Sie das CE-Gerät erneut. Der TcAdsSerialCommServer wird mit dem CE-Betriebssystem automatisch gestartet. (Beachten Sie bitte, dass der ADS Server erst automatisch startet, wenn Sie die "StartUp.exe" ab der Version 1.0.0.47 verwenden.)

Stand der Dokumentation: 11.04.2012

3 Unterstützte Hardware

Serielle Busklemme

KL6xxx im 3-Byte Modus

Die serielle Beckhoff Busklemme wird in der ausgelieferten Standardausführung im 3-Byte Modus betrieben. D. h. es können in einem Bustelegramm 3 Datenbytes zur Klemme übertragen und von ihr empfangen werden. Da für jeden Datenaustausch zwischen SPS und Busklemme 3 SPS-Zyklen notwendig sind, kann effektiv in jedem Zyklus ein Byte übertragen werden.

Die maximale effektive Datenübertragungsrate Bps ist abhängig von der Zykluszeit T der SPS und der Anzahl der Bits pro übertragenem Datenbyte LB:

Bps = LB / T

LB = 1 Startbit + n Datenbits + p Paritätsbits + m Stoppbits

Die maximale effektive Datenübertragungsrate wird nach oben durch die in der Busklemme programmierte physikalische Baudrate begrenzt.

Bei der Wahl der Zykluszeit muss bei Busklemmen die K-Bus-Update-Zeit des Buskopplers berücksichtigt werden (siehe Taskkonfiguration [▶ 47]).

KL6xxx im 5-Byte Modus

Die serielle Busklemme kann offline durch ein Konfigurationsprogramm (Beckhoff KS2000) umprogrammiert werden, so dass im 5-Byte Modus jeweils 5 Datenbyte von und zur Klemme übertragen werden können. Dabei sind ebenfalls 3 SPS-Zyklen für einen Austausch notwendig. Die effektive Datenrate liegt bei gleicher Zykluszeit der SPS um 5/3 höher als im 3-Byte Modus.

Bps = (LB * 5/3) / T

Die Umprogrammierung der Busklemmen kann nicht zur Laufzeit der SPS erfolgen, da sich der 3-Byte und der 5-Byte Modus im Register-Mapping und in der Konfiguration im TwinCAT System Manager unterscheiden.

Bei der Wahl der Zykluszeit muss bei Busklemmen die K-Bus-Update-Zeit des Buskopplers berücksichtigt werden (siehe Taskkonfiguration [▶ 47]).

KL6xxx im 22-Byte Modus

Als Sondertyp kann die serielle Busklemme mit einem 24 Byte großen Prozessabbild geliefert werden, so dass jeweils 22 Datenbyte von und zur Klemme übertragen werden können. Dabei sind ebenfalls 3 SPS-Zyklen für einen Austausch notwendig.

Bps = (LB * 22/3) / T

Diese Busklemme wird erst ab Version 2.0 der seriellen Kommunikationsbibliothek (ComLibV2) unterstützt. Bei der Wahl der Zykluszeit muss bei Busklemmen die K-Bus-Update-Zeit des Buskopplers berücksichtigt werden (siehe Taskkonfiguration [▶ 47]).

Serielle EtherCAT Klemme

EL60xx im 22-Byte Modus

Die serielle EtherCAT Klemme wird im 22-Byte Modus betrieben, so dass jeweils 22 Datenbyte von und zur Klemme übertragen werden können. Dabei sind ebenfalls 3 SPS-Zyklen für einen Austausch notwendig.

Bps = (LB * 22/3) / T

Die Parametrierung der Klemme wird über den CoE-Online Reiter im TwinCAT System Manager (mit Doppelklick auf das entsprechende Objekt) vorgenommen.

Serielle PC-Schnittstelle

Die serielle PC-Schnittstelle (COM1, COM2 etc.) wird durch das TwinCAT System analog zur seriellen Busklemme bedient und benutzt größere Datenübertragungspuffer als die serielle Busklemme. Die Bibliothek nutzt einen 64-Byte Puffer, so dass gleichzeitig bis zu 64 Datenbytes zwischen SPS und Schnittstellentreiber übertragen werden. Auch bei der seriellen PC-Schnittstelle werden 3 SPS-Zyklen für den Austausch eines Datenblocks benötigt.

Bps = (LB * 64/3) / T

Virtueller serieller COM Port

Ein in Windows verfügbarer virtueller serieller COM Port (COM1, ..., COM255) wird durch das TwinCAT System ebenfalls unterstützt. Dazu bedarf es keiner Konfiguration des Prozessabbildes im TwinCAT System Manager.
Die Parametrierung erfolgt direkt in der SPS mit den zur Verfügung gestellten Funktionsbausteinen.

Diese Kommunikationsverbindung ist nicht echtzeitfähig.

Die Baudrate ist einstellbar von 150 Baud bis zu 115200 Baud.

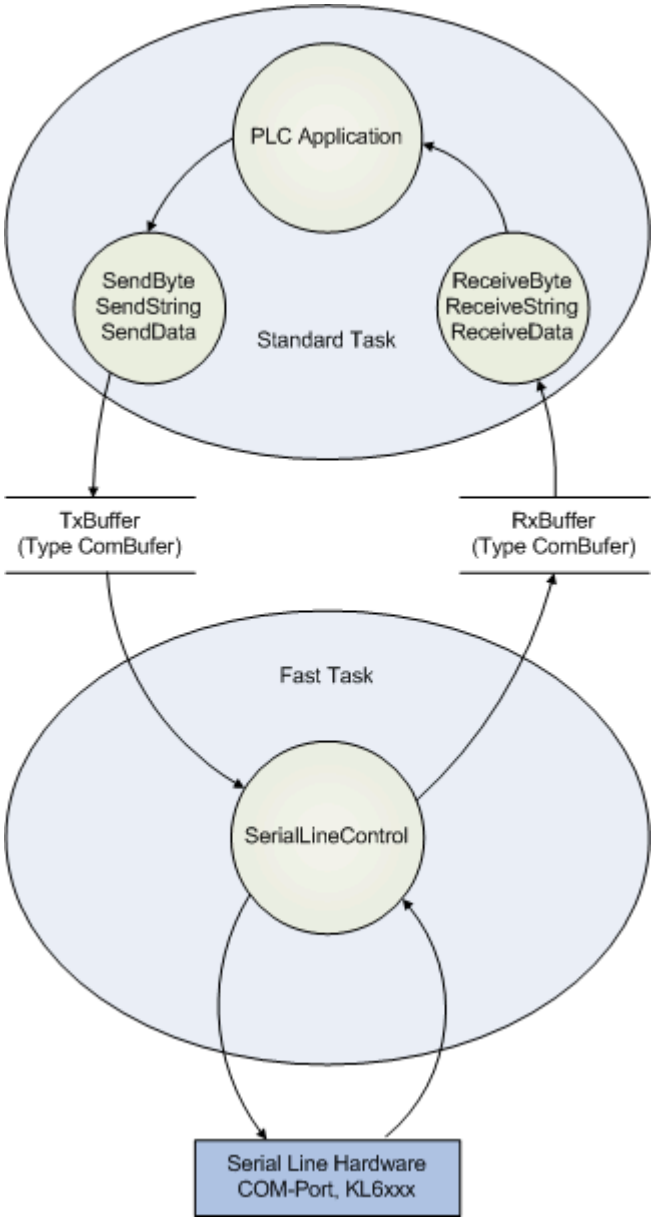
Das Produkt ist für die Anwendung mit einem einzigen virtuellen COM Port pro Zielsystem ausgerichtet. Die Verwendung mehrerer virtueller COM Ports an einem System ist mit Funktionstest jedoch generell möglich.

4 Kommunikationskonzept

Hintergrundkommunikation

Wie im Kapitel Unterstützte Hardware [▶ 10] ausgeführt, ist die maximale effektive Datenübertragungsrate unter anderem von der Zykluszeit der SPS abhängig. So ist zum Beispiel bei der seriellen Busklemme für eine Kommunikation mit effektiv 9600 bps bereits eine Zykluszeit von 1 ms notwendig. In vielen größeren Anwendungen würde eine solch kurze Zykluszeit für die gesamte SPS den Steuerungsrechner stark belasten.

Da für die meisten Anwendungen längere Zykluszeiten von z. B. 10 ms vollkommen ausreichend sind, ist es mit Hilfe der Bibliothek möglich, den Datenverkehr zwischen SPS und Hardware von der restlichen SPS-Anwendung zu entkoppeln. Dazu werden im SPS-Programm zwei Tasks angelegt. Die Standard-Task läuft in dem gewünschten langsamen SPS-Zyklus von z. B. 10 ms und eine zweite Kommunikations-Task läuft in einem schnelleren Zyklus von z. B. 2 ms.



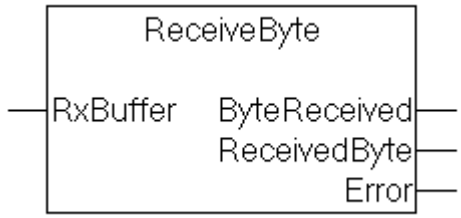
Zur Entkopplung der unterschiedlichen Geschwindigkeiten zwischen der schnellen Kommunikations-Task und der Standard-Task werden Datenpuffer vom Typ ComBuffer [▶ 35] verwendet, die asynchron beschrieben und gelesen werden.

Die später beschriebenen Funktionsbausteine zum Empfangen und Senden von Daten (SendByte [▶ 14], SendString [▶ 16], SendData [▶ 19] etc.) benutzen nur noch die Datenpuffer zum Datenaustausch und sind damit unabhängig von der verwendeten Hardware. In jedem Fall wird zusätzlich zu den Sende- und Empfangsbausteinen ein Kommunikationsbaustein SerialLineControl [▶ 21] in der schnellen Task aufgerufen, der den Datenverkehr zwischen Datenpuffer und Hardware mit maximaler Geschwindigkeit im Hintergrund abwickelt. Immer dann, wenn ein COM-Port oder eine KL60xx mit großem 22 Byte Dateninterface verwendet wird, kann bei kleinen Baudraten evtl. auf die zweite Task verzichtet werden. Der Kommunikationsbaustein SerialLineControl [▶ 21] kann dann auch in der Standard-Task aufgerufen werden.

Die Kommunikation mit virtuellen COM Ports mit Hilfe des Funktionsbausteines SerialLineControlADS [▶ 22] verwendet einen ADS Server. Über ADS werden die zu sendenden Daten asynchron ausgetauscht und in Zwischenpuffern verwaltet. Der Server seinerseits empfängt die Daten des virtuellen COM Ports unabhängig von der SPS. Somit hängt die maximale effektive Datenübertragungsrate nicht von der Zykluszeit der SPS ab.
Allein die Reaktionszeit auf angekommene Daten wird durch eine schnelle Task reduziert. Unabhängig davon sind die Virtual-Com-Port Treiber sowie der ADS Server nicht echtzeitfähig, wodurch sich die Gesamtverzögerung nicht definieren lässt.
Im Normalfall wird der Kommunikationsbaustein *SerialLineControlADS* in der Standard-Task aufgerufen und somit auf eine zweite Task verzichtet.

5 Funktionsbausteine

5.1 ReceiveByte



Interface

```
VAR_OUTPUT
    ByteReceived: BOOL;
    ReceivedByte: BYTE;
    Error: ComError_t;
END_VAR
VAR_IN_OUT
    RxBuffer : ComBuffer;
END_VAR
```

Beschreibung

Der Baustein ReceiveByte empfängt ein einzelnes Zeichen von der mit der Eingangsvariablen RxBuffer korrespondierenden Schnittstelle. Ist nach dem Aufruf ByteReceived=TRUE, dann steht in der Ausgangsvariablen ReceivedByte das empfangene Datenbyte zur Verfügung. Anderenfalls wurden keine Daten empfangen.

Sobald der Baustein ReceiveByte in einer langsameren SPS-Task abgearbeitet wird, als die Kommunikation mit der Hardware, ist zu beachten, dass in jedem SPS-Zyklus mehr als ein Zeichen bereitstehen kann. Die empfangenen Zeichen sollten daher in einer Schleife ausgelesen werden:

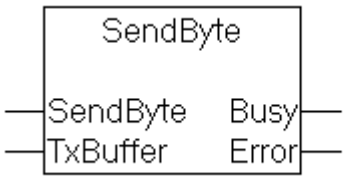
```
REPEAT
    Receive(RXbuffer:=RXbuffer);
    IF Receive.ByteReceived THEN
    (* Zeichen auswerten *)
    END_IF
UNTIL NOT Receive.ByteReceived
END_REPEAT
```

Die Anzahl der Schleifendurchläufe ist dabei grundsätzlich auf die Größe des Empfangsdatenpuffers (zur Zeit 300 Byte) begrenzt, so dass eine Endlosschleife nicht zu befürchten ist.

Sehen Sie dazu auch

- Fehlercodes - ComError_t [▶ 41]
- ComBuffer [▶ 35]

5.2 SendByte



Interface

```
VAR_INPUT
    SendByte: BYTE;
END_VAR
VAR_OUTPUT
    Busy : BOOL;
    Error: ComError_t;
END_VAR
VAR_IN_OUT
    TxBuffer: ComBuffer;
END_VAR
```

Beschreibung

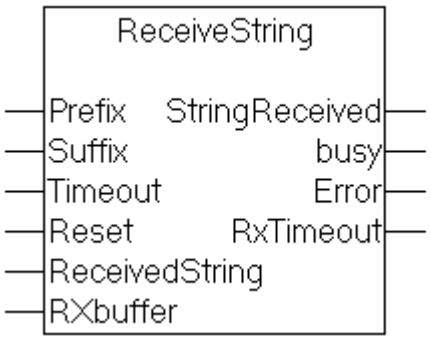
SendByte sendet ein einzelnes Zeichen an die mit der Eingangsvariablen **TxBuffer** korrespondierende Schnittstelle. Solange der Ausgang **Busy**=TRUE ist, wurde das Senden noch nicht abgeschlossen. Das Zeichen wurde erfolgreich gesendet, sobald **Busy**=FALSE und Error=0 ist. Der Ausgang Busy wird demnach nicht TRUE, wenn die Daten bereits mit dem ersten Aufruf abgeschickt werden konnten.

Solange der Sendedatenpuffer noch Daten aufnehmen kann, können in einem SPS-Zyklus mehrere Zeichen gesendet werden. Das ist jedoch nur sinnvoll, wenn die gepufferten Zeichen durch eine schnellere Kommunikations-Task zur Hardware übertragen werden.

Sehen Sie dazu auch

- Fehlercodes - ComError_t [▶ 41]
- ComBuffer [▶ 35]

5.3 ReceiveString



Interface

```
VAR_INPUT
    Prefix : STRING;
    Suffix : STRING;
    Timeout : TIME;
    Reset :BOOL;
END_VAR
VAR_OUTPUT
    StringReceived: BOOL;
    busy : BOOL;
    Error: ComError_t;
    RxTimeout : BOOL;
END_VAR
VAR_IN_OUT
    ReceivedString : STRING;
    RXbuffer : ComBuffer;
END_VAR
```

Beschreibung

ReceiveString empfängt eine Zeichenkette von der mit der Eingangsvariablen **RxBuffer** korrespondierenden Schnittstelle und speichert sie in der Ausgangsvariablen **ReceivedString**. Anfang und Ende der Zeichenkette werden über verschiedene miteinander kombinierbare Mechanismen erkannt:

- **Präfix**

  Wird in der Eingangsvariablen Prefix ein String übergeben, so müssen die ersten Zeichen der empfangenen Daten mit diesem Präfix übereinstimmen. Andere Zeichen werden verworfen. Wird kein Präfix übergeben (Leerstring), so beginnt der Empfangsstring mit dem ersten empfangenen Zeichen.
- **Suffix**

  Wird in der Eingangsvariablen Suffix ein String übergeben, so werden die Eingangsdaten solange gelesen, bis das Ende des Empfangsstrings mit dem Suffix übereinstimmt. Erreichen die empfangenen Daten dabei die Maximallänge des Empfangsstrings, so wird ein Fehler COMERROR_STRINGOVERRUN generiert. Wenn ein Leerstring als Suffix übergeben wird, so muss alternativ ein Timeout definiert werden, da anderenfalls das Ende der Zeichenkette nicht erkannt werden kann.
- **Timeout**

  Wird ein Timeout an den Baustein übergeben, so werden solange Zeichen empfangen, bis nach einem Zeichen eine entsprechend große Zeitlücke folgt. Der Empfangsstring besteht aus den bis dahin empfangenen Zeichen. Suffix und Timeout dürfen kombiniert werden. Wird ein Suffix übergeben, so darf der Timeout 0 sein.

Sobald der Ausgang StringReceived TRUE wird, stehen in der Variablen ReceivedString die empfangenen Daten bereit.

Reset

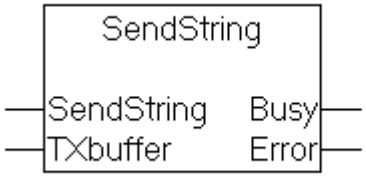
Durch Setzen des Eingangs Reset wird der Baustein aus dem Empfangszustand in den Grundzustand zurückgesetzt. Das Zurücksetzen ist nur in Ausnahmefällen notwendig, wenn zum Beispiel der erwartete String nicht empfangen werden konnte und der Baustein Busy bleibt.

Der String ReceivedString hat eine Standardlänge von 80 Zeichen. Für manche Anwendungen kann diese Länge zu kurz sein. In diesem Fall kann der Baustein **ReceiveString255** verwendet werden. Der einzige Unterschied ist eine Stringlänge von 255 Zeichen für den ReceivedString.

Sehen Sie dazu auch

- Fehlercodes - ComError_t [▶ 41]
- ComBuffer [▶ 35]

5.4 SendString



Interface

```
VAR_INPUT
    SendString: STRING;
END_VAR
VAR_OUTPUT
    Busy : BOOL;
```

```
    Error: ComError_t;
END_VAR
VAR_IN_OUT
    TXbuffer: ComBuffer;
END_VAR
```

Beschreibung

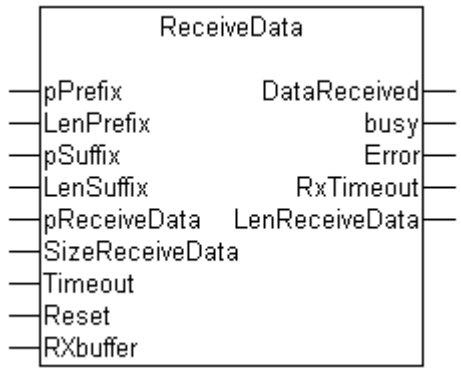
SendString sendet eine Zeichenkette an die mit der Eingangsvariablen **TxBuffer** korrespondierende Schnittstelle. Solange der Ausgang **Busy**=TRUE ist, wurde das Senden noch nicht abgeschlossen. Die Daten wurden erfolgreich gesendet, sobald **Busy**=FALSE und Error=0 ist. Der Ausgang Busy wird demnach nicht TRUE, wenn die Daten bereits mit dem ersten Aufruf abgeschickt werden konnten.

Der String SendString hat eine Standardlänge von 80 Zeichen. Für manche Anwendungen ist eine größere Länge wünschenswert. In diesem Fall kann der Baustein **SendString255** verwendet werden. Der einzige Unterschied ist eine Stringlänge von 255 Zeichen für den SendString.

Sehen Sie dazu auch

- Fehlercodes - ComError_t [▶ 41]
- ComBuffer [▶ 35]

5.5 ReceiveData



Interface

```
VAR_INPUT
    pPrefix         : POINTER TO BYTE;
    LenPrefix       : BYTE;
    pSuffix         : POINTER TO BYTE;
    LenSuffix       : BYTE;
    pReceiveData    : POINTER TO BYTE;
    SizeReceiveData : DINT;
    Timeout         : TIME;
    Reset           :BOOL;
END_VAR
VAR_OUTPUT
    DataReceived    : BOOL;
    busy            : BOOL;
    Error           : ComError_t;
    RxTimeout       : BOOL;
    LenReceiveData  : UDINT;
END_VAR
VAR_IN_OUT
    RXbuffer        : ComBuffer;
END_VAR
```

Beschreibung

ReceiveData empfängt Daten beliebigen Typs von der mit der Eingangsvariablen **RxBuffer** korrespondierenden Schnittstelle und speichert sie in der Variablen ReceiveData. Anfang und Ende des Datenstromes werden über verschiedene miteinander kombinierbare Mechanismen erkannt:

- **Präfix**

  Wird in der Eingangsvariablen Präfix eine Variable übergeben, so müssen die ersten Zeichen der empfangenen Daten mit diesem Präfix übereinstimmen. Andere Zeichen werden verworfen. Wird kein Präfix übergeben (Null), so beginnen die Empfangsdaten mit dem ersten empfangenen Zeichen.

- **Suffix**

  Wird eine Eingangsvariable Suffix übergeben, so werden die Eingangsdaten solange gelesen, bis das Ende der Empfangsdaten mit dem Suffix übereinstimmt. Erreichen die empfangenen Daten dabei die Maximallänge SizeReceiveData, so wird ein Fehler COMERROR_DATASIZEOVERRUN generiert.

- **Blockgröße**
  Wird kein Suffix angegeben, so werden bis zu SizeReceiveData Zeichen empfangen.

- **Timeout**

  Wird ein Timeout an den Baustein übergeben, so werden solange Zeichen empfangen, bis nach einem Zeichen eine entsprechend große Zeitlücke folgt. Die Empfangsdaten bestehen aus den bis dahin empfangenes Zeichen. Ist Timeout 0, dann werden bis zu SizeReceiveData Zeichen ohne Zeitüberwachung empfangen.

Sobald der Ausgang **DataReceived** TRUE wird, stehen in der Variablen ReceiveData die empfangenen Daten bereit. Die Anzahl der empfangenen Zeichen wird in **LenReceiveData** angegeben.

pPrefix

pPrefix ist die Adresse einer beliebigen Datenstruktur, die mit ADR(Variablenname) an den Baustein übergeben wird. LenPrefix gibt die Anzahl der Datenbytes des Präfixes an.

LenPrefix

LenPrefix gibt die Anzahl der Datenbytes des Präfixes an.

pSuffix

pSuffix ist die Adresse einer beliebigen Datenstruktur, die mit ADR(Variablenname) an den Baustein übergeben wird.

LenSuffix

LenSuffix gibt die Anzahl der Datenbytes des Suffixes an.

pReceiveData

pReceiveData ist die Adresse der Empfangsdaten und wird mit ADR(Empfangsdaten) ermittelt. Die empfangenen Daten werden in der Variablen auf die pReceiveData zeigt abgelegt.

SizeReceiveData

SizeReceiveData wird mit SIZEOF(Empfangsdaten) ermittelt und gibt die maximale Größe der Empfangdaten an.

Timeout

Timeout definiert die maximale Zeitlücke zwischen zwei empfangenen Zeichen. Die Timeoutüberwachung wird nach dem ersten Zeichen wirksam. Somit kann mit Timeout nicht überwacht werden, ob ein erwartetes Telegramm eintrifft oder nicht. Diese Überwachung erfolgt extern.

Reset

Durch Setzen des Eingangs Reset wird der Baustein aus dem Empfangszustand in den Grundzustand zurückgesetzt. Das Zurücksetzen ist nur in Ausnahmefällen notwendig, wenn zum Beispiel die erwarteten Daten nicht empfangen werden konnten und der Baustein Busy bleibt.

DataReceived

DataReceived wird TRUE sobald die Empfangsdaten gültig sind. Der Ausgang ist für genau einen Zyklus TRUE, sodass die empfangenen Daten sofort ausgewertet werden müssen.

Busy

Busy wird ab dem ersten empfangenen Zeichen TRUE und wird FALSE sobald die Daten empfangen wurden oder ein Fehler aufgetreten ist.

Error

Error gibt im Fehlerfall einen Fehlercode aus. Der Fehlercode ist durch den Datentype ComError_t definiert und wird dadurch zur Laufzeit in Textform angezeigt.

RxTimeout

RxTimeout wird TRUE wenn die maximale Zeitlücke zwischen zwei empfangenen Zeichen überschritten wird. Der Datenempfang wird dadurch abgebrochen und die bis dahin empfangenes Zeichen liegen bereit. Wenn ohne Suffix gearbeitet wird, so ist die Timeout-Erkennung kein Fehler, sondern kennzeichnet das normale Ende der Empfangsdaten. Wird jedoch ein Suffix verwendet, so konnte dieses nicht empfangen werden.

LenReceiveData

LenReceiveData gibt die tatsächliche Anzahl der empfangenen Datenbytes an und kann kleiner oder gleich SizeReceiveData sein.

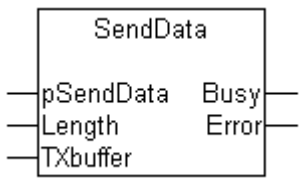
RxBuffer

RxBuffer ist der Empfangsdatenpuffer, der mit der verwendeten Schnittstelle korrespondiert.

Sehen Sie dazu auch

- Fehlercodes - ComError_t [▶ 41]
- ComBuffer [▶ 35]

5.6 SendData



Interface

```
VAR_INPUT
    pSendData : POINTER TO BYTE;
    Length : UDINT;
END_VAR
VAR_OUTPUT
    Busy : BOOL;
    Error: ComError_t;
END_VAR
VAR_IN_OUT
    TXbuffer: ComBuffer;
END_VAR
```

Beschreibung

SendData sendet den Inhalt einer Variablen beliebigen Typs an die mit der Eingangsvariablen **TxBuffer** korrespondierende Schnittstelle. Solange der Ausgang **Busy**=TRUE ist, wurde das Senden noch nicht abgeschlossen. Die Daten wurden erfolgreich gesendet, sobald **Busy**=FALSE und Error=0 ist.

pSendData

pSendData ist die Adresse der Sendedaten und wird mit ADR(Sendedaten) ermittelt.

Die Sendedaten dürfen nicht verändert werden, solange Busy=TRUE ist und die Daten noch nicht vollständig gesendet wurden.

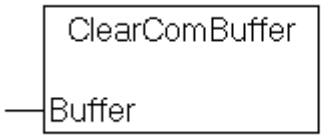
Length

Length ist die Anzahl der zu sendenden Datenbytes und kann kleiner oder gleich der Größe der verwendeten Datenstruktur sein. Wenn der gesamte Inhalt einer Variablen gesendet werden soll, kann Length mit SIZEOF(Sendedaten) ermittelt werden.

Sehen Sie dazu auch

- Fehlercodes - ComError_t [▶ 41]
- ComBuffer [▶ 35]

5.7 ClearComBuffer



Interface

```
VAR_IN_OUT
    Buffer : ComBuffer;
END_VAR
```

Beschreibung

Der SPS-interne Kommunikationspuffer **Buffer** wird gelöscht.

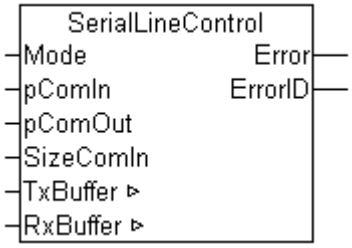
Sehen Sie dazu auch

- ComBuffer [▶ 35]

5.8 Hardwareabhängige Funktionsbausteine

5.8.1 Hintergrundkommunikation

5.8.1.1 SerialLineControl



Der Funktionsbaustein SerialLineControl wickelt die Kommunikation zwischen einer seriellen Schnittstelle (KL60xx, EL60xx oder COM-Schnittstelle) und der SPS ab. Der Funktionsbaustein wird zyklisch aufgerufen und stellt empfangene Daten im *RxBuffer* bereit. Gleichzeitig werden im Sendebuffer *TxBuffer* bereitgestellte Daten zur Schnittstelle übertragen.

Da die Funktion unabhängig von der Applikation abgewickelt wird, wird sie als Hintergrundkommunikation bezeichnet und kann, insbesondere bei seriellen Busklemmen, auch in einer schnellen Task abgewickelt werden (siehe Kommunikationskonzept [▶ 12] und Unterstützte Hardware [▶ 10]).

Der Funktionsbaustein ersetzt ab der Version 2.0 der Bibliothek die hardwareabhängigen Funktionsbausteine KL6Control [▶ 24], KL6Control5B [▶ 25] und PcComControl [▶ 24].

Interface

```
VAR_INPUT
    Mode            : ComSerialLineMode_t;
    pComIn          : POINTER TO BYTE
    pComOut         : POINTER TO BYTE
    SizeComIn       : UINT;
END_VAR
VAR_OUTPUT
    Error           : BOOL;
    ErrorID         : ComError_t;
END_VAR
VAR_IN_OUT
    TxBuffer        : ComBuffer;
    RxBuffer        : ComBuffer;
END_VAR
```

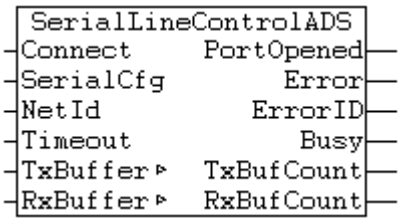
Mode	Der Mode-Eingang legt eindeutig fest, welche serielle Hardware verwendet wird.
pComIn	Universeller Pointer auf die Eingangsvariable der Prozessdaten der seriellen Hardware (Datentypen KL6inData [▶ 37], KL6inData5b [▶ 37], KL6inData22b [▶ 38], EL6inData22b [▶ 38], PcComInData [▶ 39]). Der Pointer wird mit der *ADR*() Funktion zugewiesen.
pComOut	Universeller Pointer auf die Ausgangsvariable der Prozessdaten der seriellen Hardware (Datentypen KL6outData [▶ 37], KL6outData5b [▶ 37], KL6outData22b [▶ 38], EL6outData22b [▶ 38], PcComOutData [▶ 39]). Der Pointer wird mit der *ADR*() Funktion zugewiesen.
SizeComIn	Größe des Eingangs-Prozessabbildes der verwendeten seriellen Hardware. Die Größe wird mit der *SIZEOF*() Funktion ermittelt und zugewiesen.
Error	Der *Error* Ausgangs wird TRUE sobald ein Fehler auftritt.
ErrorID	Der *ErrorID* Ausgang gibt im Fehlerfall einen Fehlercode aus.

Mode	Der Mode-Eingang legt eindeutig fest, welche serielle Hardware verwendet wird.
TxBuffer	Puffer mit Sendedaten für die verwendete serielle Hardware. Der Sendepuffer wird durch Funktionen wie SendByte [▶ 14], SendData [▶ 19] oder SendString [▶ 16] gefüllt.
RxBuffer	Puffer in dem die Empfangsdaten abgelegt werden. Der Empfangspuffer wird durch Funktionen wie ReceiveByte [▶ 14], ReceiveData [▶ 17] oder ReceiveString [▶ 15] ausgelesen.

Sehen Sie dazu auch

- Aufzählungstypen für die serielle Kommunkationsbibliothek [▶ 39]
- Fehlercodes - ComError_t [▶ 41]
- ComBuffer [▶ 35]

5.8.1.2 SerialLineControlADS



Der Funktionsbaustein *SerialLineControlADS* wickelt die Kommunikation zwischen einer virtuellen seriellen Schnittstelle und der SPS ab. Der Funktionsbaustein wird zyklisch aufgerufen und stellt empfangene Daten im *RxBuffer* bereit. Gleichzeitig werden im Sendepuffer *TxBuffer* bereitgestellte Daten zur Schnittstelle übertragen.

Da die Funktion unabhängig von der Applikation abgewickelt wird, wird sie als Hintergrundkommunikation bezeichnet und kann, ebenso wie der Funktionsbaustein *SerialLineControl*, auch in einer schnellen Task abgewickelt werden (siehe Kommunikationskonzept [▶ 12] und Unterstützte Hardware [▶ 10]).

Sobald der Funktionsbaustein zyklisch aufgerufen und der Eingang *Connect* gesetzt wird, wird automatisch der parametrierte serielle COM Port geöffnet.
Dadurch ist dieser COM Port für andere Applikationen geblockt. Möchten Sie den COM Port zwischenzeitlich freigeben, um von einer anderen Applikation aus darauf zuzugreifen, so können Sie den Eingang *Connect* zurücksetzen. Dadurch wird der bisherige Port geschlossen. Wird in der Eingangsstruktur SerialCfg [▶ 35] ein anderer COM Port oder eine andere Parametrierung angewählt, wird automatisch der vorherige Port geschlossen und daraufhin der neue Port geöffnet.

Interface

```
FUNCTION_BLOCK SerialLineControlADS
VAR_INPUT
    Connect         :BOOL;              (* connect to serial port [TRUE=connect, FALSE=disconnect] *
)
    SerialCfg       :ComSerialConfig;
    NetId           :T_AmsNetId :='';               (* host NetId *)
    Timeout         :TIME :=DEFAULT_ADS_TIMEOUT;    (* Timeout for ADS calls *)
END_VAR
VAR_IN_OUT
    TxBuffer        :ComBuffer;       (* serial Tx ComBuffer *)
    RxBuffer        :ComBuffer;       (* serial Rx ComBuffer *)
END_VAR
VAR_OUTPUT
    PortOpened      :BOOL;          (* Indicates if selected serial port is opened *)
    Error           :BOOL;          (* 'TRUE' if an error occurred *)
    ErrorID         :UDINT;         (* Displays the error code; 0 = no error *)
    Busy            :BOOL;          (* 'TRUE' if internal ADS communication is busy *)
    TxBufCount      :UDINT;         (* number of bytes in internal Tx buffer *)
    RxBufCount      :UDINT;         (* number of bytes in internal Rx buffer *)
END_VAR
```

Eingangsvariablen

Connect	Um eine Verbindung zu einem seriellen Port zu initialisieren, muss *Connect* TRUE am Funktionsbaustein anliegen. Liegt *Connect* FALSE an, so wird ein geöffneter Port wieder geschlossen. Wird ein Wechsel dieser Eingangsvariablen vollzogen, kann es maximal das 6-fache der an Timeout angegebenen Zeitspanne dauern, bis die Aktion vollständig durchgeführt wurde. **Die Applikation muss deshalb auf den Ausgang PortOpened achten und warten, bis dieser den gewünschten Zustand annimmt.**
SerialCfg	Diese Eingangsstruktur definiert welcher COM Port mit welchen Parametern verwendet und geöffnet werden soll. Details sind in der Beschreibung von ComSerialConfig [▶ 35] zu finden.
NetId	Um die Anfrage auf dem lokalen Gerät durchzuführen, bedarf es keiner Angabe dieser Eingangsvariablen. Alternativ kann ein leerer String angegeben werden. Um die Anfrage an ein anderes TwinCAT Zielgerät zu richten kann hier die entsprechende AMS Net Id angegeben werden.
Timeout	Gibt eine maximale Zeitdauer für die Ausführung des Funktionsbausteines an. Der Defaultwert ist 5 Sekunden. (Es sollte ein Wert von mindestens 1000ms angegeben werden.)

Ein-/Ausgangsvariablen

TxBuffer	Puffer mit Sendedaten für die verwendete serielle Hardware. Der Sendepuffer wird durch Funktionen wie SendByte [▶ 14], SendData [▶ 19] oder SendString [▶ 16] gefüllt.
RxBuffer	Puffer in dem die Empfangsdaten abgelegt werden. Der Empfangspuffer wird durch Funktionen wie ReceiveByte [▶ 14], ReceiveData [▶ 17] oder ReceiveString [▶ 15] ausgelesen.

Ausgangsvariablen

PortOpened	Dieser Ausgang gibt an, ob der gewählte serielle Port geöffnet und verbunden ist.
Error	Der *Error* Ausgangs wird TRUE sobald ein Fehler auftritt.
ErrorID	Der *ErrorID* Ausgang gibt im Fehlerfall einen Fehlercode aus. Im Kapitel Fehlercodes [▶ 41] finden Sie eine Auflistung möglicher Werte sowie Informationen zur Fehlerbehebung.
Busy	Dieser Ausgang ist TRUE solange die interne ADS Kommunikation des Funktionsbausteines aktiv ist.
TxBufCount	Am Ausgang *TxBufCount* lässt sich feststellen, ob sich noch Datenbytes im internen SPS Puffer befinden, welche noch nicht versendet wurden.
RxBufCount	Am Ausgang *RxBufCount* lässt sich feststellen, ob sich noch empfangene Datenbytes im internen SPS Puffer befinden, welche noch nicht zum *RxBuffer* übertragen wurden. Es muss von der Applikation sichergestellt werden, dass die empfangenen Daten schnell genug aus dem *RxBuffer* herausgelesen werden.

Einzubindende SPS Bibliotheken

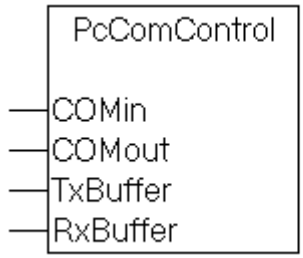
COMlibV2.lib [Version 2.003.008 oder höher]

Sehen Sie dazu auch

- ComSerialConfig [▶ 35]
- ComBuffer [▶ 35]

5.8.1.3 Bausteine bis Version 1.7

5.8.1.3.1 PcComControl



Interface

```
VAR_INPUT
    COMin : PcComInData;
END_VAR
VAR_IN_OUT
    COMout : PcComOutData;
    TxBuffer: ComBuffer;
    RxBuffer: ComBuffer;
END_VAR
```

Beschreibung

Der Funktionsbaustein **PcComControl** kontrolliert die Datenübertragung zwischen den SPS-internen Datenpuffern **TxBuffer** und **RxBuffer** und der Hardware. Die Datenstrukturen PcComInData [▶ 39] und PcComOutData [▶ 39] sind global deklariert und werden im TwinCAT System Manager mit der PC-Schnittstelle verknüpft.

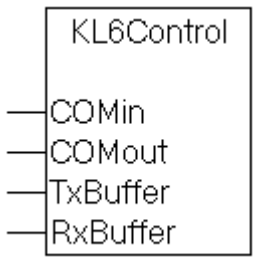
PcComControl wird unabhängig von der Absicht Daten zu senden oder zu empfangen permanent aufgerufen (Hintergrundkommunikation). Dieser Baustein sollte in einer genügend schnellen Task aufgerufen werden, um eine effektive Baudrate nahe an der physikalisch eingestellten Baudrate zu erreichen.

Die Funktionsbausteine zur Hintergrundkommunikation werden vorzugsweise in einer schnellen Task abgearbeitet (vergleiche Unterstützte Hardware [▶ 10]).
Ab der Version 2 der Kommunikationsbibliothek wird dieser Baustein durch den Baustein SerialLine-Control [▶ 21] ersetzt.

Sehen Sie dazu auch

- Datenstrukturen für serielle PC-Schnittstellen COM [▶ 39]
- ComBuffer [▶ 35]

5.8.1.3.2 KL6Control



Interface

```
VAR_INPUT
    COMin : KL6inData;
END_VAR
VAR_IN_OUT
    COMout : KL6outData;
    TxBuffer: ComBuffer;
    RxBuffer: ComBuffer;
END_VAR
```

Beschreibung

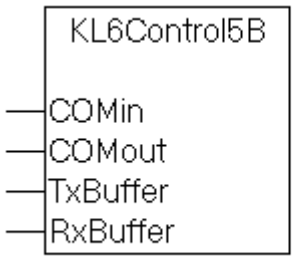
KL6Control kontrolliert analog zum Baustein **PcComControl** die Datenübertragung zwischen den SPS-internen Datenpuffern **TxBuffer** und **RxBuffer** und der Hardware.

> Die Funktionsbausteine zur Hintergrundkommunikation werden vorzugsweise in einer schnellen Task abgearbeitet (vergleiche Unterstützte Hardware [▶ 10]).
> Ab der Version 2 der Kommunikationsbibliothek wird dieser Baustein durch den Baustein SerialLine-Control [▶ 21] ersetzt.

Sehen Sie dazu auch

- Datenstrukturen für serielle Busklemme KL6xxx im 3-Byte Modus [▶ 37]
- ComBuffer [▶ 35]

5.8.1.3.3 KL6Control5B



Interface

```
VAR_INPUT
    COMin : KL6inData5B;
END_VAR
VAR_IN_OUT
    COMout : KL6outData5B;
    TxBuffer: ComBuffer;
    RxBuffer: ComBuffer;
END_VAR
```

Beschreibung

KL6Control5B kontrolliert analog zum Baustein **KL6Control** die Datenübertragung zwischen den SPS-internen Datenpuffern **TxBuffer** und **RxBuffer** und der Hardware, hier jedoch für die serielle Busklemme im 5-Byte Modus

> Die Funktionsbausteine zur Hintergrundkommunikation werden vorzugsweise in einer schnellen Task abgearbeitet (vergleiche Unterstützte Hardware [▶ 10]).
> Ab der Version 2 der Kommunikationsbibliothek wird dieser Baustein durch den Baustein SerialLine-Control [▶ 21] ersetzt.

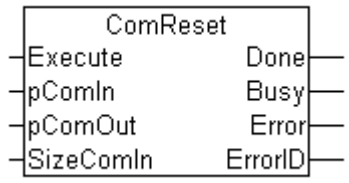
Sehen Sie dazu auch

- Datenstrukturen für serielle Busklemme KL6xxx im 5-Byte Modus [▶ 37]

ComBuffer [▶ 35]

5.8.2 Konfiguration

5.8.2.1 ComReset



ComReset führt einen Reset der angeschlossenen seriellen Hardware durch. Dadurch werden die Hardware-internen Sende- und Empfangspuffer gelöscht. Der Funktionsbaustein unterstützt verschiedene serielle Hardware wie die serielle PC-Schnittstelle und die seriellen Busklemmen KL6xxx.

Der Funktionsbaustein ersetzt ab der Version 2.0 der Bibliothek den Baustein KL6Init [▶ 31].

> **i** Der Funktionsbaustein löscht nicht die SPS-internen Datenpuffer vom Typ ComBuffer [▶ 35]. Diese können zur Initialisierung separat mit dem Funktionsbaustein ClearComBuffer [▶ 20] gelöscht werden.

Interface

```
VAR_INPUT
    Execute     : BOOL;
    pComIn      : POINTER TO BYTE
    pComOut     : POINTER TO BYTE
    SizeComIn   : UINT;
END_VAR
VAR_OUTPUT
    Done        : BOOL;
    Busy        : BOOL;
    Error       : BOOL;
    ErrorID     : UDINT;
END_VAR
```

Execute : Eine steigende Flanke am Eingang Execute führt einen Reset der angeschlossenen seriellen Hardware durch.

pComIn : Universeller Pointer auf die Eingangsvariable der Prozessdaten der seriellen Hardware (Datentypen KL6inData [▶ 37], KL6inData5b [▶ 37], PcComInData [▶ 39]). Der Pointer wird mit der *ADR*() Funktion zugewiesen.

pComOut : Universeller Pointer auf die Ausgangsvariable der Prozessdaten der seriellen Hardware (Datentypen KL6outData [▶ 37], KL6outData5b [▶ 37], PcComOutData [▶ 39]). Der Pointer wird mit der *ADR*() Funktion zugewiesen.

SizeComIn : Größe des Eingangs-Prozessabbildes der verwendeten seriellen Hardware. Die Größe wird mit der *SIZEOF*() Funktion ermittelt und zugewiesen.

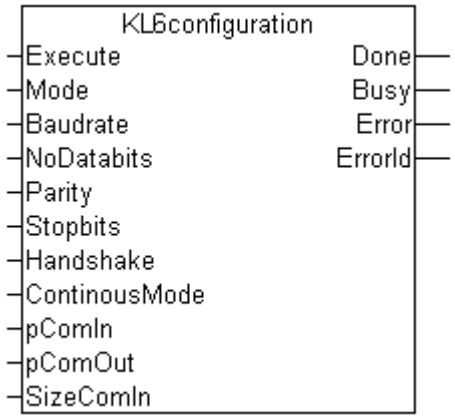
Done : Der Ausgang Done wird TRUE wenn die Funktion fehlerfrei durchgeführt wurde.

Busy : Der Ausgang Busy wird mit steigender Flanke an Execute TRUE und bleibt solange TRUE wie der Baustein seine Funktion ausführt.

Error : Der *Error* Ausgangs wird TRUE sobald ein Fehler auftritt.

ErrorID : Der *ErrorID* Ausgang gibt im Fehlerfall einen Fehlercode aus.

5.8.2.2 KL6Configuration



Der Funktionsbaustein KL6configuration initialisiert und konfiguriert eine serielle Busklemme KL6xxx.

Der Funktionsbaustein ersetzt ab der Version 2.0 der Bibliothek die hardwareabhängigen Funktionsbausteine KL6Config [▶ 31] und KL6Config5B [▶ 32].

Konfiguration der EtherCAT-Klemmen

Der Funktionsbaustein verwendet die bei KL-Klemmen übliche Registerkommunikation zur Konfiguration. Bei EtherCAT Klemmen EL ist diese Registerkommunikation nicht möglich. EL-Klemmen können mit Funktionsbausteinen aus der EtherCAT-Bibliothek konfiguriert werden (FB_EcCoeSdoWrite).

Interface

```
VAR_INPUT
    Execute         : BOOL;
    Mode            : ComSerialLineMode_t;
    Baudrate        : UDINT;            (* 115200, 57600, 38400, 19200, 9600, 4800, 2400, 1200 *)
    NoDatabits      : BYTE;             (* 7 or 8 *)
    Parity          : ComParity_t;        (* PARITY_NONE=0, PARITY_EVEN=1, PARITY_ODD=2 *)
    Stopbits        : BYTE;             (* 1 or 2 *)
    Handshake       : ComHandshake_t;         (* HANDSHAKE_NONE=0, HANDSHAKE_RTSCTS=1, HANDSHAKE_XON
XOFF=2 *)
    ContinousMode   : BOOL;             (* don't start transmission before transmit buffer is filled
 *)
    pComIn          : POINTER TO BYTE;      (* for universal register communication *)
    pComOut         : POINTER TO BYTE;      (* for universal register communication *)
    SizeComIn       : UINT;
END_VAR
VAR_OUTPUT
    Done        : BOOL;
    Busy        : BOOL;
    Error       : BOOL;
    ErrorId     : ComError_t;
END_VAR
```

Execute : Eine steigende Flanke am Eingang Execute führt einen Reset der angeschlossenen seriellen Hardware durch.

Mode : Der Mode-Eingang legt eindeutig fest, welche serielle Hardware verwendet wird.

Baudrate : Die Baudrate, soweit durch die serielle Hardware unterstützt

NoDatabits : Anzahl der Nutzdatenbits in einem Datenbyte

Parity : Typ des Paritybits eines Datenbytes

Stopbits : Anzahl der Stoppbits pro Datenbyte

Handshake : Typ des verwendeten Handshakes soweit durch die serielle Hardware unterstützt

ContinousMode : Schaltet das kontinuierliche Senden ein, wenn es durch die serielle Hardware unterstützt wird.

Wenn ContinousMode TRUE ist, werden gesendete Daten erst dann aus der seriellen Hardware abgeschickt, wenn der Hardware-Sendepuffer voll ist. Dadurch wird ein zeitlückenfreies Senden gewährleistet, solange die Datenmenge in der Größenordnung des Hardware-Sendepuffers liegt. Der continous mode wird nur in besonderen Fällen benötigt, wenn das Endgerät auf Zeitlücken mit einem Timeout reagiert.

pComIn : Universeller Pointer auf die Eingangsvariable der Prozessdaten der seriellen Hardware (Datentypen KL6inData [▶ 37], KL6inData5b [▶ 37], KL6inData22b [▶ 38], PcComInData [▶ 39]). Der Pointer wird mit der *ADR*() Funktion zugewiesen.

pComOut : Universeller Pointer auf die Ausgangsvariable der Prozessdaten der seriellen Hardware (Datentypen KL6outData [▶ 37], KL6outData5b [▶ 37], KL6outData22b [▶ 38], PcComOutData [▶ 39]). Der Pointer wird mit der *ADR*() Funktion zugewiesen.

SizeComIn : Größe des Eingangs-Prozessabbildes der verwendeten seriellen Hardware. Die Größe wird mit der *SIZEOF*() Funktion ermittelt und zugewiesen.

Done : Der Ausgang Done wird TRUE wenn die Funktion fehlerfrei durchgeführt wurde.

Busy : Der Ausgang Busy wird mit steigender Flanke an Execute TRUE und bleibt solange TRUE wie der Baustein seine Funktion ausführt.

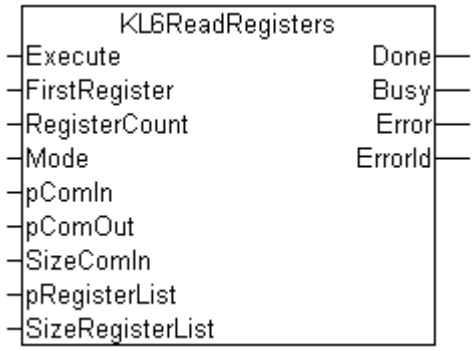
Error : Der *Error* Ausgangs wird TRUE sobald ein Fehler auftritt.

ErrorID : Der *ErrorID* Ausgang gibt im Fehlerfall einen Fehlercode aus.

Sehen Sie dazu auch

- Aufzählungstypen für die serielle Kommunkationsbibliothek [▶ 39]
- Fehlercodes - ComError_t [▶ 41]

5.8.2.3 KL6ReadRegisters



Der Funktionsbaustein KL6ReadRegisters liest ein oder mehrere Register einer seriellen Busklemme KL6xxx.

Interface

```
VAR_INPUT
    Execute             : BOOL;
    FirstRegister       : UINT;
    RegisterCount       : UINT;
    Mode                : ComSerialLineMode_t;
    pComIn              : POINTER TO ARRAY[0..65] OF BYTE; (* for universal register communication *
)
    pComOut             : POINTER TO ARRAY[0..65] OF BYTE; (* for universal register communication *
)
    SizeComIn           : UINT;
    pRegisterList       : POINTER TO ARRAY[0..63] OF ComRegisterData_t;
    SizeRegisterList    : UINT;
```

```
END_VAR
VAR_OUTPUT
    Done        : BOOL;
    Busy        : BOOL;
    Error       : BOOL;
    ErrorId     : ComError_t;
END_VAR
```

Execute : Eine steigende Flanke am Eingang Execute startet den Lesevorgang.

FirstRegister : Legt das erst zu lesende Register fest. Ab dieser Registernummer, die zwischen 1 und 64 liegen darf, werden *RegisterCount* Daten gelesen und in der Registerliste vom Typ ComRegisterData_t abgelegt.

Wenn kein zusammenhängender Registerbereich gelesen werden soll, so kann *FirstRegister* mit dem Wert 16#FFFF belegt werden. In diesem Fall muss der Anwender die zu lesenden Registernummern in der Registerliste initialisieren, bevor der Baustein getriggert wird. In diesem Fall wird auch *RegisterCount* nicht verwendet.

RegisterCount : Legt die Anzahl der zu lesenden Register fest. Der Baustein liest einen zusammenhängenden Registerbereich ab FirstRegister und legt die Daten in der Registerliste ab.

Mode : Der Mode-Eingang legt eindeutig fest, welche serielle Hardware verwendet wird (ComSerialLineMode_t [▶ 39]).

pComIn : Universeller Pointer auf die Eingangsvariable der Prozessdaten der seriellen Hardware (Datentypen KL6inData [▶ 37], KL6inData5b [▶ 37], KL6inData22b [▶ 38], PcComInData [▶ 39]). Der Pointer wird mit der *ADR*() Funktion zugewiesen.

pComOut : Universeller Pointer auf die Ausgangsvariable der Prozessdaten der seriellen Hardware (Datentypen KL6outData [▶ 37], KL6outData5b [▶ 37], KL6outData22b [▶ 38], PcComOutData [▶ 39]). Der Pointer wird mit der *ADR*() Funktion zugewiesen.

SizeComIn : Größe des Eingangs-Prozessabbildes der verwendeten seriellen Hardware. Die Größe wird mit der *SIZEOF*() Funktion ermittelt und zugewiesen.

pRegisterList : Startadresse einer Registerliste vom Typ ComRegisterData_t [▶ 35]. Die Startadresse kann mit ADR(Registerliste) ermittelt werden.

SizeRegisterList : Größe der Registerliste in Bytes. Die Größe kann mit SIZE(Registerliste) ermittelt werden. Die Liste darf zwischen 1 und 64 Einträge haben.

Done : Der Ausgang Done wird TRUE wenn die Funktion fehlerfrei durchgeführt wurde.

Busy : Der Ausgang Busy wird mit steigender Flanke an Execute TRUE und bleibt solange TRUE, wie der Baustein seine Funktion ausführt.

Error : Der *Error* Ausgangs wird TRUE sobald ein Fehler auftritt.

ErrorID : Der *ErrorID* Ausgang gibt im Fehlerfall einen Fehlercode aus.

Sehen Sie dazu auch

- Fehlercodes - ComError_t [▶ 41]

5.8.2.4 KL6WriteRegisters

Der Funktionsbaustein KL6WrtieRegisters beschreibt ein oder mehrere Register einer seriellen Busklemme KL6xxx.

Interface

```
VAR_INPUT
    Execute         : BOOL;
    Mode            : ComSerialLineMode_t;
    pComIn          : POINTER TO ARRAY[0..65] OF BYTE; (* for universal register communication *)
    pComOut         : POINTER TO ARRAY[0..65] OF BYTE; (* for universal register communication *)
    SizeComIn       : UINT;
    pRegisterList   : POINTER TO ARRAY[0..63] OF ComRegisterData_t;
    SizeRegisterList: UINT;
END_VAR
VAR_OUTPUT
    Done        : BOOL;
    Busy        : BOOL;
    Error       : BOOL;
    ErrorId     : ComError_t;
END_VAR
```

Execute : Eine steigende Flanke am Eingang Execute startet den Schreibvorgang.
Die *Registerliste* muss initialisiert werden, bevor der Baustein getrigger wird. Das heißt es müssen Registernummern und Registerinhalt in die Liste eingetragen werden.

Mode : Der Mode-Eingang legt eindeutig fest, welche serielle Hardware verwendet wird (ComSerialLineMode_t [▶ 39]).

pComIn : Universeller Pointer auf die Eingangsvariable der Prozessdaten der seriellen Hardware (Datentypen KL6inData [▶ 37], KL6inData5b [▶ 37], KL6inData22b [▶ 38], PcComInData [▶ 39]). Der Pointer wird mit der *ADR*() Funktion zugewiesen.

pComOut : Universeller Pointer auf die Ausgangsvariable der Prozessdaten der seriellen Hardware (Datentypen KL6outData [▶ 37], KL6outData5b [▶ 37], KL6outData22b [▶ 38], PcComOutData [▶ 39]). Der Pointer wird mit der *ADR*() Funktion zugewiesen.

SizeComIn : Größe des Eingangs-Prozessabbildes der verwendeten seriellen Hardware. Die Größe wird mit der *SIZEOF*() Funktion ermittelt und zugewiesen.

pRegisterList : Startadresse einer Registerliste vom Typ ComRegisterData_t [▶ 35]. Die Startadresse kann mit ADR(Registerliste) ermittelt werden. Die *Registerliste* muss initialisiert werden, bevor der Baustein getrigger wird. Das heißt es müssen Registernummern und Registerinhalt in die Liste eingetragen werden.

SizeRegisterList : Größe der Registerliste in Bytes. Die Größe kann mit SIZE(Registerliste) ermittelt werden. Die Liste darf zwischen 1 und 64 Einträge haben.

Done : Der Ausgang Done wird TRUE wenn die Funktion fehlerfrei durchgeführt wurde.

Busy : Der Ausgang Busy wird mit steigender Flanke an Execute TRUE und bleibt solange TRUE, wie der Baustein seine Funktion ausführt.

Error : Der *Error* Ausgangs wird TRUE sobald ein Fehler auftritt.

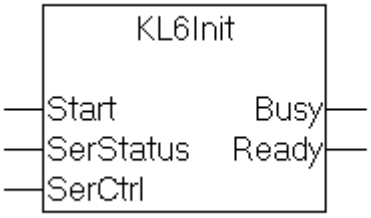
ErrorID : Der *ErrorID* Ausgang gibt im Fehlerfall einen Fehlercode aus.

Sehen Sie dazu auch

Fehlercodes - ComError_t [▶ 41]

5.8.2.5 Bausteine bis Version 1.7

5.8.2.5.1 KL6Init



Interface

```
VAR_INPUT
    Start : BOOL;
    SerStatus : BYTE;
END_VAR
VAR_OUTPUT
    Busy : BOOL;
    Ready : BOOL;
END_VAR
```

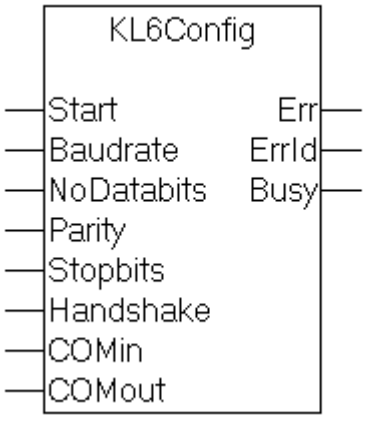
Beschreibung

KL6Init initialisiert die serielle Busklemme KL6xxx ohne deren Konfiguration zu verändern. Es werden dadurch die Klemmen-internen Puffer gelöscht.

Die Eingangsdaten **SerStatus** und **SerCtrl** entsprechen den mit der Klemme im TwinCAT SystemManager verknüpften Daten. Sie sind in den Datenstrukturen KL6inData [▶ 37] bzw. KL6outData [▶ 37] der COMlib definiert.

Die Klemme wurde nach dem Startsignal **Start** erfolgreich initialisiert, sobald Busy=FALSE und Ready=TRUE ist.

5.8.2.5.2 KL6Config



Interface

```
VAR_INPUT
    Start : BOOL; (* Flankengetriggert *)
    Baudrate : INT;     (* 19200, 9600, 4800, 2400, 1200 *)
    NoDatabits : BYTE;      (* 7 or 8 *)
    Parity : BYTE;      (* 0=no 1= even 2 = odd *)
    Stopbits : BYTE;    (* 1 or 2 *)
    Handshake : BYTE;       (*0=none,1=RTS/CTS,2=XON/XOFF*)
    COMin : KL6inData;
END_VAR
VAR_OUTPUT
    Err : BOOL;
    ErrId : WORD;
    Busy : BOOL;
END_VAR
VAR_IN_OUT
    COMout : KL6outData;
END_VAR
```

Beschreibung

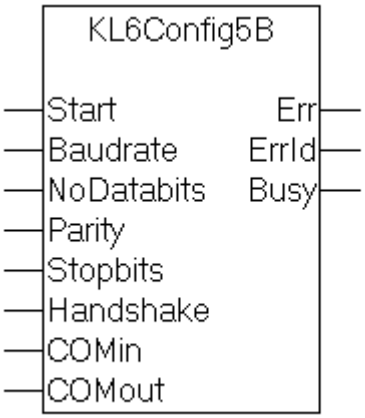
KL6Config konfiguriert die Schnittstellenparameter der seriellen Busklemme KL6xxx.

Während der Baustein **KL6Config** aktiv (**busy**) ist, darf der Baustein KL6Control zur Hintergrundkommunikation **nicht** aufgerufen werden! Initialisierung und normale Kommunikation mit der seriellen Busklemme benutzen dieselben Register.
Ab der Version 2 der Kommunikationsbibliothek wird dieser Baustein durch den Baustein KL6configuration [▶ 27] ersetzt.

Sehen Sie dazu auch

- Datenstrukturen für serielle Busklemme KL6xxx im 3-Byte Modus [▶ 37]

5.8.2.5.3 KL6Config5B



Interface

```
VAR_INPUT
    Start : BOOL; (* Flankengetriggert *)
    Baudrate : INT;     (* 19200, 9600, 4800, 2400, 1200 *)
    NoDatabits : BYTE;      (* 7 or 8 *)
    Parity : BYTE;      (* 0=no, 1= even, 2 = odd *)
    Stopbits : BYTE;        (* 1 or 2 *)
    Handshake : BYTE;    (*0=none,1=RTS/CTS,2=XON/XOFF*)
    COMin : KL6inData5B;
END_VAR
VAR_OUTPUT
    Err : BOOL;
    ErrId : WORD;
    Busy : BOOL;
END_VAR
```

```
VAR_IN_OUT
    COMout : KL6outData5B;
END_VAR
```

Beschreibung

KL6Config5B arbeitet analog zum Baustein KL6Config, bedient jedoch die serielle Busklemme im 5-Byte Modus.

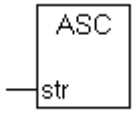
Während der Baustein **KL6Config5B** aktiv (**busy**) ist, darf der Baustein KL6Control5B zur Hintergrundkommunikation nicht aufgerufen werden! Initialisierung und normale Kommunikation mit der seriellen Busklemme benutzen dieselben Register.
Ab der Version 2 der Kommunikationsbibliothek wird dieser Baustein durch den Baustein KL6configuration [▶ 27] ersetzt.

Sehen Sie dazu auch

- Datenstrukturen für serielle Busklemme KL6xxx im 5-Byte Modus [▶ 37]

6 Hilfsfunktionen

6.1 ASC



Interface

```
FUNCTION ASC : BYTE
VAR_INPUT
    str : STRING;
END_VAR
```

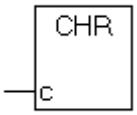
Beschreibung

Die Funktion Asc gibt den ASCII-Code des ersten Zeichens des Eingangsstrings in einem Byte zurück.

Hintergrund

Sendedaten liegen oft als Zeichenkette in einem String vor. Sollen sie gesendet werden, so werden die einzelnen Zeichen als Byte benötigt.

6.2 CHR



Interface

```
FUNCTION CHR : STRING
VAR_INPUT
    c : BYTE;
END_VAR
```

Beschreibung

Die Funktion Chr gibt das Zeichen, dass dem ASCII-Code in der Eingangsvariablen c entspricht in einem String zurück.

Hintergrund

Empfangene Zeichen kommen als Byte in das SPS-System und müssen häufig als Zeichenkette weiterverarbeitet werden.

7 Datentypen

7.1 Strukturen

7.1.1 ComBuffer

Die Datenstruktur **ComBuffer** ist ein Datenpuffer zur Entkopplung der Hardware-abhängigen Kommunikationsbausteine von den Hardware-unabhängigen Bausteinen (siehe auch Kommunikationskonzept [▶ 12]). Gleichzeitig wird damit eine schnelle Kommunikations-Task von der Standard-Task entkoppelt. Datenpuffer vom Typ ComBuffer werden vom Anwender niemals direkt beschrieben oder gelesen, sondern dienen nur als Zwischenspeicher für die Kommunikationsbausteine.

```
TYPE ComBuffer
STRUCT
    Buffer: ARRAY[0..300] OF BYTE;
    RdIdx: INT;
    WrIdx: INT;
    Count: INT; (* Anzahl der Zeichen im Ringpuffer *)
    FreeByte:INT; (* Anzahl freie Plätze im Ringpuffer *)
    Error: INT; (* Fehlercode der Schnittstelle *)
    blocked : BOOL;
END_STRUCT
END_TYPE
```

7.1.2 Datenstrukturen Registerliste

Für das Lesen und Schreiben von Registern einer seriellen Busklemme ist der Datentyp *ComRegisterList_t* definiert. Jeder Eintrag der *Registerliste* enthält die Registernummer und den Inhalt des Registers.

ComRegisterList_t

```
TYPE ComRegisterList_t : ARRAY[0..63] OF ComRegisterData_t;
END_TYPE
```

ComRegisterData_t

```
TYPE ComRegisterData_t :
STRUCT
    Register : BYTE;
    Value : WORD;
END_STRUCT
END_TYPE
```

7.1.3 ComSerialConfig

Diese Eingangsstruktur definiert welcher COM Port mit welchen Parametern verwendet und geöffnet werden soll.

Wird ein Parameter während dem zyklischen Aufrufen von SerialLineControlADS [▶ 22] geändert, so wird automatisch die bestehende COM Port Verbindung geschlossen und der serielle COM Port mit dem neuen Parameter neu geöffnet. Es ist dabei nicht nötig den Port durch ein Rücksetzen des Einganges *Connect* explizit zu schließen.

```
TYPE ComSerialConfig :
(* contains the configuration parameters of the com port to be opened. *)
STRUCT
    ComPort     :UDINT      :=1;            (* Serial port number [1..255] *)
    Baudrate    :UDINT      :=9600;
    Parity      :ComParity_t    :=PARITY_NONE;
    DataBits    :INT        :=8;            (* [4..8] *)
    StopBits    :ComStopBits_t  :=STOPBITS_ONE;

    DTR         :ComDTRCtrl_t   :=DTR_CTRL_HANDSHAKE;   (* 'Data Terminal Ready' signal *)
    RTS         :ComRTSCtrl_t   :=RTS_CTRL_HANDSHAKE;   (* 'Request to Send' signal (= RFR 'Ready fo
```

```
r Receiving') *)
    CTS          :BOOL       :=FALSE;        (* 'Clear to Send' signal *)
    DSR          :BOOL       :=FALSE;        (* 'Dataset Ready' signal *)

    TraceLevel      :BYTE       :=0;            (* None=0;Error=1,Warning=2,Info=3,Verbose=4,Noise=5
 *)

    Reserved1       :BYTE;
    Reserved2       :BYTE;
    Reserved3       :BYTE;
END_STRUCT
END_TYPE
```

ComPort	Es kann ein beliebiger COM Port, COM1 .. COM255, ausgewählt werden. Welche Nummer ein Treiber (z.B. ein USB-To-Virtual-Com-Port Treiber) vergeben hat, lässt sich im Windows Hardware Geräte Manager nachvollziehen.
Baudrate	Es können alle Standardbaudraten eingestellt werden, von 150 Baud bis zu 115200 Baud. Default ist 9600 Baud.
Parity	Hier wird die Einstellung der Paritätsprüfung für die serielle Datenübertragung vorgenommen. Mögliche Werte sind in der Enumeration ComParity_t [▶ 39] zusammengefasst.
DataBits	Hier wird die Einstellung der Anzahl an Datenbits für die serielle Datenübertragung vorgenommen. Mögliche Werte sind 4,5,6,7 und 8. Bei einem Wert kleiner 8 Bit werden zwar dennoch komplette Datenbytes an den SPS Puffer übergeben, jedoch werden nicht alle 8 Bit eines Bytes über die serielle Schnittstelle übertragen. Die höchstwertigsten Bits werden vor der Übertragung abgeschnitten, wodurch sich die Bitmenge je zu übertragenen Datenbytes verringert. Diese Einstellung war hauptsächlich zu früheren Zeiten sinnvoll, falls man wusste, dass die höchstwertigsten Bits je Byte nie verwendet werden und man die Übertragungsgeschwindigkeit erhöhen wollte. Sie wird selten verwendet, so dass empfohlen wird die Anzahl an Datenbits bei 8 [default] zu belassen.
StopBits	Hier wird die Einstellung der Anzahl an Stoppbits für die serielle Datenübertragung vorgenommen. Mögliche Werte sind in der Enumeration ComStopBits_t [▶ 39] zusammengefasst.
RTS	Hier wird die Einstellung des 'Request to Send' (= RFR 'Ready for Receiving') Signals für die serielle Datenübertragung vorgenommen. Mögliche Werte sind in der Enumeration ComRTSCtrl_t [▶ 39] zusammengefasst.
DTR	Hier wird die Einstellung des 'Data Terminal Ready' Signals für die serielle Datenübertragung vorgenommen. Mögliche Werte sind in der Enumeration ComDTRCtrl_t [▶ 39] zusammengefasst.
CTS	Hier wird die Einstellung des 'Clear to Send' Signals für die serielle Datenübertragung vorgenommen. Wenn der Wert TRUE ist, so werden keine Daten gesendet, falls das CTS Eingangssignal der Datenübertragung nicht gesetzt ist.
DSR	Hier wird die Einstellung des 'Dataset Ready' Signals für die serielle Datenübertragung vorgenommen. Wenn der Wert TRUE ist, so werden keine Daten gesendet, falls das DSR Eingangssignal der Datenübertragung nicht gesetzt ist. Ebenso werden, wenn *DSR* TRUE ist, empfangene Datenbytes ignoriert, falls das DSR Eingangssignal der Datenübertragung nicht gesetzt ist.
TraceLevel	Mit diesem Eingang kann die Ausgabe von Meldungen (debug traces) seitens des TcAdsSerialCommServer konfiguriert werden. Dies betrifft nicht die Fehlermeldungen am Ausgang vom SPS Funktionsbaustein *SerialLineControlADS*. Es handelt sich demnach um eine zusätzliche Diagnosemöglichkeit. Folgende Trace Level sind möglich (default = 0): • 0 None (default) • 1 Error • 2 Warning • 3 Info • 4 Verbose

	• 5 Noise Alle Meldungen mit einem Level kleiner oder gleich dem angegebenen Level werden ausgegeben. Falls Level 'None' gewählt ist, werden keine Meldungen ausgegeben. Als Ausgabe werden Debug Traces verwendet. Unter win32 werden Error-/Warning-/Info-Messages ebenfalls als Log Events (Applikation Log) ausgegeben. Mit Level 'Noise' werden sogar am seriellen Port empfangene Daten in den Meldungen ausgegeben. Dafür werden einige System Ressourcen benötigt und die Einstellung sollte somit nur für vorübergehende Tests gewählt werden. **Bei Verwendung des Tools DebugView (von SysInternals) muss die Einstellung CaptureGlobalWin32 aktiv sein, um die Meldungen zu empfangen.**

Nicht jede Einstellung der Parameter für die serielle Datenübertragung ist zwangsweise immer möglich. Manche Einstellungen oder Kombinationen werden von Windows oder von Com-Port-Treibern nicht unterstützt.
(Beispiel: Oft wird die Möglichkeit der 1,5 Stoppbits oder der 4 Datenbits oder auch der Kombination von 5 Datenbits & 2 Stoppbits nicht unterstützt. Oder die Baudrate ist auf maximal 115200 Baud begrenzt.)

Weitere Informationen zu den Parametern einer seriellen Datenübertragung finden sich in der Microsoft MSDN Beschreibung zur DCB Struktur.

Einzubindende SPS-Bibliotheken

COMlibV2.lib [Version 2.003.008 oder höher]

7.1.4 Datenstrukturen für serielle Busklemme KL6xxx im 3-Byte Modus

Jede serielle Busklemme benötigt zum Datenaustausch über den I/O-Bus je eine Variable des Typs KL6inData und KL6outData. Diese Variablen werden im Speicherabbild auf eine feste Adresse gelegt und mit dem TwinCAT System Manager mit der Hardware verbunden.

KL6inData

```
TYPE KL6inData
STRUCT
    Status: BYTE;
    SerStatus: BYTE;
    D: ARRAY[0..2] OF BYTE;
END_STRUCT
END_TYPE
```

KL6outData

```
TYPE KL6outData
STRUCT
    Ctrl: BYTE;
    SerCtrl: BYTE;
    D: ARRAY[0..2] OF BYTE;
END_STRUCT
END_TYPE
```

7.1.5 Datenstrukturen für serielle Busklemme KL6xxx im 5-Byte Modus

Jede serielle Busklemme benötigt zum Datenaustausch über den I/O-Bus je eine Variable des Typs KL6inData5B und KL6outData5B. Diese Variablen werden im Speicherabbild auf eine feste Adresse gelegt und mit dem TwinCAT System Manager mit der Hardware verbunden.

KL6inData5B

```
TYPE KL6inData5B
STRUCT
    Status: BYTE;
    D: ARRAY[0..4] OF BYTE;
END_STRUCT
END_TYPE
```

KL6outData5B

```
TYPE KL6outData5B
STRUCT
    Ctrl: BYTE;
    D: ARRAY[0..4] OF BYTE;
END_STRUCT
END_TYPE
```

7.1.6 Datenstrukturen für serielle Busklemme KL6xxx im 22-Byte Modus

Jede serielle Busklemme benötigt zum Datenaustausch über den I/O-Bus je eine Variable des Typs KL6inData22B und KL6outData22B. Diese Variablen werden im Speicherabbild auf eine feste Adresse gelegt und mit dem TwinCAT System Manager mit der Hardware verbunden.

KL6inData22B

```
TYPE KL6inData22B
STRUCT
    Status  : WORD;
    D       : ARRAY[0..21] OF BYTE;
END_STRUCT
END_TYPE
```

KL6outData22B

```
TYPE KL6outData22B
STRUCT
    Ctrl    : WORD;
    D       : ARRAY[0..21] OF BYTE;
END_STRUCT
END_TYPE
```

7.1.7 Datenstrukturen für serielle EtherCAT Klemme EL60xx im 22-Byte Modus

Jede serielle EtherCAT Klemme benötigt zum Datenaustausch über den I/O-Bus je eine Variable des Typs EL6inData22B und EL6outData22B. Diese Variablen werden im Speicherabbild auf eine feste Adresse gelegt und mit dem TwinCAT System Manager mit der Hardware verbunden.

EL6inData22B

```
TYPE EL6inData22B
STRUCT
    Status  : WORD;
    D       : ARRAY[0..21] OF BYTE;
END_STRUCT
END_TYPE
```

EL6outData22B

```
TYPE EL6outData22B
STRUCT
    Ctrl    : WORD;
    D       : ARRAY[0..21] OF BYTE;
END_STRUCT
END_TYPE
```

7.1.8 Datenstrukturen für serielle PC-Schnittstellen COM

Jede serielle PC-Schnittstelle benötigt zum Datenaustausch je eine Variable des Typs PcComInData und PcComOutData . Diese Variablen werden im Speicherabbild auf eine feste Adresse gelegt und mit dem TwinCAT SystemManager mit der Hardware verbunden.

PcComInData

```
TYPE PcComInData
STRUCT
    SerStatus: WORD;
    D: ARRAY[0..63] OF BYTE;
END_STRUCT
END_TYPE
```

PcComOutData

```
TYPE PcComOutData
STRUCT
    SerCtrl: WORD;
    D: ARRAY[0..63] OF BYTE;
END_STRUCT
END_TYPE
```

7.2 Aufzählungstypen für die serielle Kommunkationsbibliothek

ComSerialLineMode_t

Der Aufzählungstyp ComSerialLineMode_t legt den Typ der verwendeten seriellen Hardware für verschiedene hardwareabhängige Funktionsbausteine der seriellen Kommunikationsbibliothek fest.

```
TYPE ComSerialLineMode_t :
(
    SERIALLINEMODE_DEFAULT,
    SERIALLINEMODE_KL6_3B_ALTERNATIVE,
    SERIALLINEMODE_KL6_5B_STANDARD,
    SERIALLINEMODE_KL6_22B_STANDARD,
    SERIALLINEMODE_PC_COM_PORT,
    SERIALLINEMODE_EL6_22B,
    SERIALLINEMODE_IE6_11B
);
END_TYPE
```

ComHandshake_t

```
TYPE ComHandshake_t :
(
    HANDSHAKE_NONE,
    HANDSHAKE_RTSCTS,
    HANDSHAKE_XONXOFF,
    RS485_FULLDUPLEX,
    RS485_HALFDUPLEX,
    RS485_FULLDUPLEX_XONXOFF,
    RS485_HALFDUPLEX_XONXOFF
);
END_TYPE
```

ComParity_t

```
TYPE ComParity_t :
(
    PARITY_NONE,
    PARITY_EVEN,
    PARITY_ODD,
    PARITY_MARK,    (* only available with SerialLineControlADS *)
    PARITY_SPACE    (* only available with SerialLineControlADS *)
);
END_TYPE
```

ComStopBits_t

```
TYPE ComStopBits_t :
(
    STOPBITS_ONE    := 1,
    STOPBITS_TWO    := 2,
    STOPBITS_ONE5   := 3
);
END_TYPE
```

ComDTRCtrl_t

```
TYPE ComDTRCtrl_t :
(
    DTR_CTRL_DISABLE,
    DTR_CTRL_ENABLE,
    DTR_CTRL_HANDSHAKE
);
END_TYPE
```

ComRTSCtrl_t

```
TYPE ComRTSCtrl_t :
(
    RTS_CTRL_DISABLE,
    RTS_CTRL_ENABLE,
    RTS_CTRL_HANDSHAKE,
    RTS_CTRL_TOGGLE
);
END_TYPE
```

8 Fehlercodes

8.1 Fehlercodes - ComError_t

ComError_t

```
TYPE ComError_t :
(
    COMERROR_NOERROR                := 0,
    COMERROR_PARAMETERCHANGED       := 1,       (* input parameters changed during reception *)
    COMERROR_TXBUFFOVERRUN          := 2,       (* string > transmit buffer *)
    COMERROR_STRINGOVERRUN          := 10,      (* end of string *)
    COMERROR_ZEROCHARINVALID        := 11,      (* string cannot receive zero characters *)
    COMERROR_INVALIDPOINTER         := 20,      (* invalid data pointer, e. g. zero *)
    COMERROR_INVALIDRXPOINTER       := 21,      (* invalid data pointer for ReceiveData *)
    COMERROR_INVALIDRXLENGTH        := 22,      (* invalid length for ReceiveData, e. g. zero *)
    COMERROR_DATASIZEOVERRUN        := 23,      (* end of data block *)
    COMERROR_INVALIDPROCESSDATASIZE := 24,
    COMERROR_MODENOTSUPPORTED       := 16#0101,     (* mode not supported (3-
Byte Terminals connectd to bus controllers) *)
    COMERROR_INVALIDCHANNELNUMBER   := 16#0102,
    COMERROR_INVALIDBAUDRATE        := 16#1001,
    COMERROR_INVALIDNUMDATABITS     := 16#1002,
    COMERROR_INVALIDNUMSTOPBITS     := 16#1003,
    COMERROR_INVALIDPARITY          := 16#1004,
    COMERROR_INVALIDHANDSHAKE       := 16#1005,
    COMERROR_INVALIDNUMREGISTERS    := 16#1006,
    COMERROR_INVALIDREGISTER        := 16#1007,
    COMERROR_TIMEOUT                := 16#1008
);
END_TYPE
```

Fehlercodes vom Type *ComError_t* werden vom Funktionsbaustein *SerialLineControlADS* nicht verwendet.

8.2 Fehlercodes - AdsSerialComm

Übersicht der Fehlercodes des Funktionsbausteines SerialLineControlADS

Offset + Fehlercode	Bereich	Beschreibung
0x00000000 + TwinCAT System Fehler	0x00000000-0x00007800	TwinCAT System Fehler (ADS-Fehlercodes inklusive)
0x00000000 + TcAdsSerialCommServer Fehler	0x00009000-0x000091FF	Fehler des TwinCAT ADS Serial Comm Servers
0x3D090000 + Win32 system error code	0x3D090000-0x3D09FFFF	Win32 Systemfehler

Fehlercodes vom Type *ComError_t* werden von diesem Funktionsbaustein nicht verwendet.

TcAdsSerialCommServer Fehler

Code (hex)	Code (dec)	Beschreibung	Symbolischer Name
0x00009001	36865	Der angegebene COM Port ist ungültig. gültiger Wertebereich: 1 .. 255	COMERRORADS_INVALID_COMPORT
0x00009002	36866	Das Kommando für den TcAdsSerialCommServer ist ungültig.	COMERRORADS_INVALID_CMD
0x00009003	36867	interner Fehler	COMERRORADS_INVALID_DATAPOINTER
0x00009011	36881	Die übergebene Parameterstruktur ist dem TcAdsSerialCommServer unbekannt.	COMERRORADS_INVALID_CFGSTLEN
0x00009012	36882	Die übergebene Parameterstruktur ist dem TcAdsSerialCommServer unbekannt.	COMERRORADS_INVALID_CFGSTVER
0x00009013	36883	Das Trace Level (Variable *TraceLevel* in der Eingangsstruktur SerialCfg [▶ 35]) für die Ausgabe von Meldungen ist ungültig.	COMERRORADS_INVALID_TL
0x00009021	36897	Die angegebene Baudrate ist wird nicht unterstützt.	COMERRORADS_INVALID_BAUDRATE
0x00009022	36898	Die angegebene Parität ist ungültig.	COMERRORADS_INVALID_PARITY
0x00009023	36899	Die angegebene Anzahl an Datenbits ist ungültig.	COMERRORADS_INVALID_BYTESIZE
0x00009024	36900	Die angegebene Anzahl an Stoppbits ist ungültig.	COMERRORADS_INVALID_STOPBIT
0x00009025	36901	Dtr Control ist ungültig.	COMERRORADS_INVALID_DTR_CTRL
0x00009026	36902	Rts Control ist ungültig.	COMERRORADS_INVALID_RTS_CTRL
0x00009027	36903	Cts ist ungültig.	COMERRORADS_INVALID_CTS_OUTCTRL
0x00009028	36904	Dsr ist ungültig.	COMERRORADS_INVALID_DSR_OUTCTRL
0x00009029	36905	Dsr ist ungültig.	COMERRORADS_INVALID_DSR_SENS
0x00009031	36913	interner Fehler	COMERRORADS_NOT_INIT
0x00009032	36914	Der Empfangspuffer im TcAdsSerialCommServer ist übergelaufen. Eingehende Daten gehen verloren. **Empfangene Daten müssen umgehend abgefragt werden.** Es ist dafür zu sorgen, dass sich im SPS Funktionsbaustein SerialLineControlADS [▶ 22] keine Daten ansammeln. Dies kann mit dem Ausgang	COMERRORADS_RD_BUFFER_OVERRUN

Code (hex)	Code (dec)	Beschreibung	Symbolischer Name
		RxBufCount überwacht werden. Bei stabiler Kommunikationsverbindung steigt dieser Wert nicht über 1000.	
0x00009033	36915	Der COM Port ist bereits geöffnet. *SerialLineControlADS* wird automatisch versuchen den Port zu schließen und mit den angegebenen Parametern neu zu öffnen. Liegen in den darauf folgenden Zyklen keine Fehler am Ausgang an und wird der Ausgang *PortOpened* = TRUE, so ist ein Öffnen des COM Ports erfolgreich abgeschlossen.	COMERRORADS_PORT_CONNECTED
0x00009034	36916	Die Interaktion mit dem COM Port ist nicht möglich, weil der COM Port noch nicht vom TcAdsSerialCommServer geöffnet wurde. *SerialLineControlADS* wird automatisch versuchen den Port mit den angegebenen Parametern neu zu öffnen. Liegen in den darauf folgenden Zyklen keine Fehler am Ausgang an und wird der Ausgang *PortOpened* = TRUE, so ist ein Öffnen des COM Ports erfolgreich abgeschlossen.	COMERRORADS_PORT_NOT_CONNECTED
0x00009035	36917	Der COM Port konnte nicht korrekt geschlossen werden.	COMERRORADS_RD_THREAD_TIMEOUT
0x00009036	36918	Der COM Port konnte nicht korrekt geschlossen werden.	COMERRORADS_WR_THREAD_TIMEOUT
0x00009037	36919	Während einer bestehenden Kommunikationsverbindung kann dieser Fehler auftreten, wenn das USB Gerät getrennt wird. Vor dem Trennen des USB Gerätes muss der Eingang *bConnect* = FALSE gesetzt werden und damit der COM Port	COMERRORADS_RD_FAILURE

Code (hex)	Code (dec)	Beschreibung	Symbolischer Name
		geschlossen wird. Ebenso kann ein Lesefehler der Auslöser sein. Details lassen sich mit Hilfe der Variablen *TraceLevel* in der Eingangsstruktur SerialCfg [▶ 35] ausgeben. **SerialLineControlADS hat den Port geschlossen und wird automatisch versuchen ihn mit den angegebenen Parametern neu zu öffnen.** Liegen in den darauf folgenden Zyklen keine Fehler am Ausgang an und wird der Ausgang *PortOpened* = TRUE, so ist ein Öffnen des COM Ports erfolgreich abgeschlossen.	
0x00009038	36920	Ein Schreibfehler kann der Auslöser für diesen Fehlercode sein. Details lassen sich mit Hilfe der Variablen *TraceLevel* in der Eingangsstruktur SerialCfg [▶ 35] ausgeben. Möglicherweise sind Daten nicht übertragen worden. Es findet keine automatische Wiederholung des Schreibversuches statt.	COMERRORADS_WR_FAILURE
0x000090E0 - 0x000090FF	37088 - 37119	interne Fehler	
0x00009101	37121	Die Version des TcAdsSerialCommServer ist inkompatibel. Eine offizielle Produktinstallation behebt den Fehler.	COMERRORADS_SERVER_INCOMPATIBLE

zusätzliche Erläuterungen zu den wichtigsten Win32 Systemfehlern

Code (hex)	Code (dec)	Beschreibung	zusätzliche Erläuterung	Symbolischer Name
0x3D090002	1024000002	Das System kann die spezifizierte Datei nicht finden.	Dieser Fehler kann auftreten, wenn der angegebene COM Port nicht verfügbar ist. Überprüfen Sie, ob	ERROR_FILE_NOT_FOUND

Code (hex)	Code (dec)	Beschreibung	zusätzliche Erläuterung	Symbolischer Name
			der Wert für den verwendeten COM Port sowie die weiteren Parameter in ComSerialConfig [▶ 35] korrekt angegeben sind.	
0x3D090005	1024000005	Zugriff ist verweigert.	Prüfen Sie, ob der entsprechende serielle COM Port bereits von einem anderen Programm belegt/ geöffnet wurde. In diesem Fall müssen Sie den Port vom anderen Programm aus freigeben, um eine Kommunikation zu ermöglichen.	ERROR_ACCESS_DENIED
0x3D090057	1024000087	Der Parameter ist nicht korrekt.	Dieser Fehler tritt auf, falls ein Eingangsparameter für die serielle Datenübertragung (siehe ComSerialConfig [▶ 35]) ungültig ist. Nicht jede Einstellung der Parameter für die serielle Datenübertragung ist zwangsweise immer möglich. Manche Einstellungen oder Kombinationen werden von Windows oder von Com-Port-Treibern nicht unterstützt. In diesem Fall sollten Sie prüfen, ob die Kommunikation mit einer anderen Parametereinstellung funktioniert.	ERROR_INVALID_PARAMETER

Für eine detailliertere Fehleranalyse können zusätzliche debug Ausgaben des TcAdsSerialComm-Server konfiguriert werden. Dies geschieht mit der Variablen *TraceLevel* in der Eingangsstruktur SerialCfg [▶ 35].

9 Einbinden in ein SPS-Programm

9.1 Installation

COMlibV2

Zur Installation wird die Bibliothek COMlibV2.lib in das TwinCAT Verzeichnis TwinCAT\PLC\LIB kopiert.

COMlib

Zur Installation werden die Bibliothek COMlib.LIB und die mitgelieferte Hilfsbibliothek ChrAsc.LIB und ChrAsc.OBJ in das TwinCAT Verzeichnis TwinCAT\PLC\LIB kopiert.

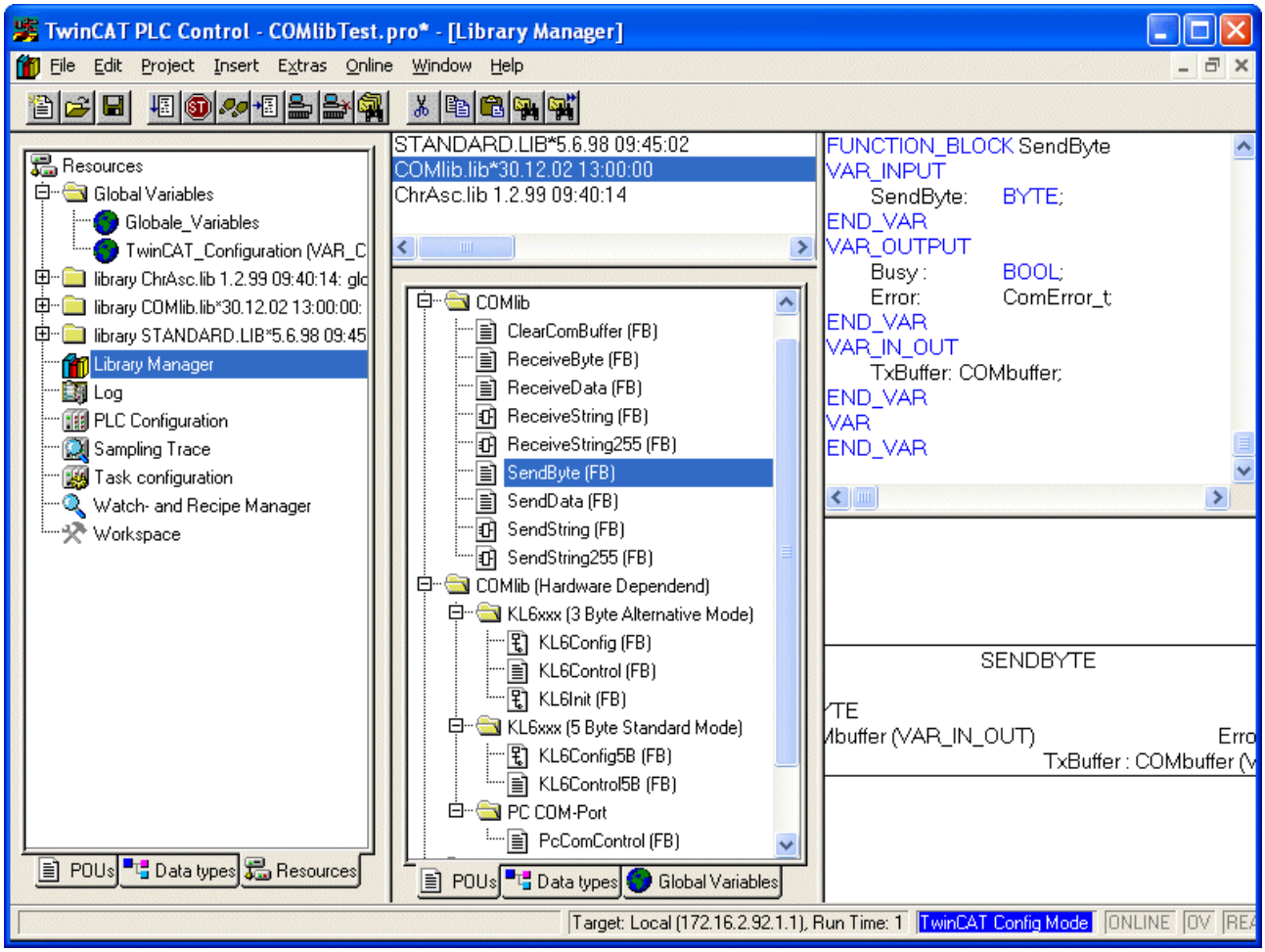
Das beiliegende Testprogramm kopieren Sie in ein beliebiges Projektverzeichnis, z. B. nach TwinCAT\PLC.

- https://infosys.beckhoff.com/content/1031/tcplclibserialcom/Resources/11388462091.zip

Bibliotheken einbinden

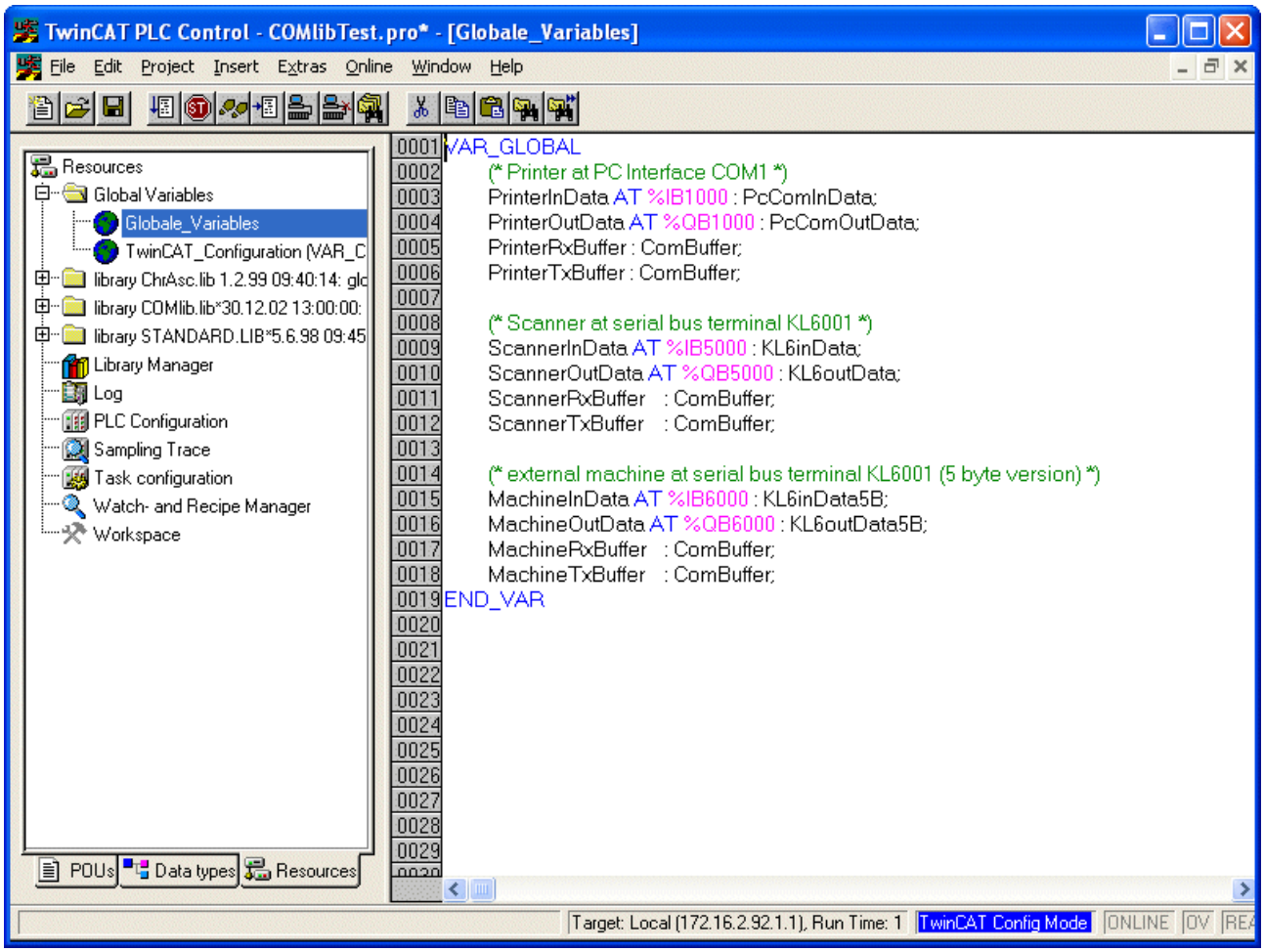
Legen Sie ein neues SPS-Projekt mit dem TwinCAT PLC Control an, um die Einbindung der Bibliothek nachzuvollziehen.

Wechseln Sie in die Bibliotheksverwaltung und fügen Sie die Bibliotheken ChrAsc.LIB und ComLib.LIB ein.



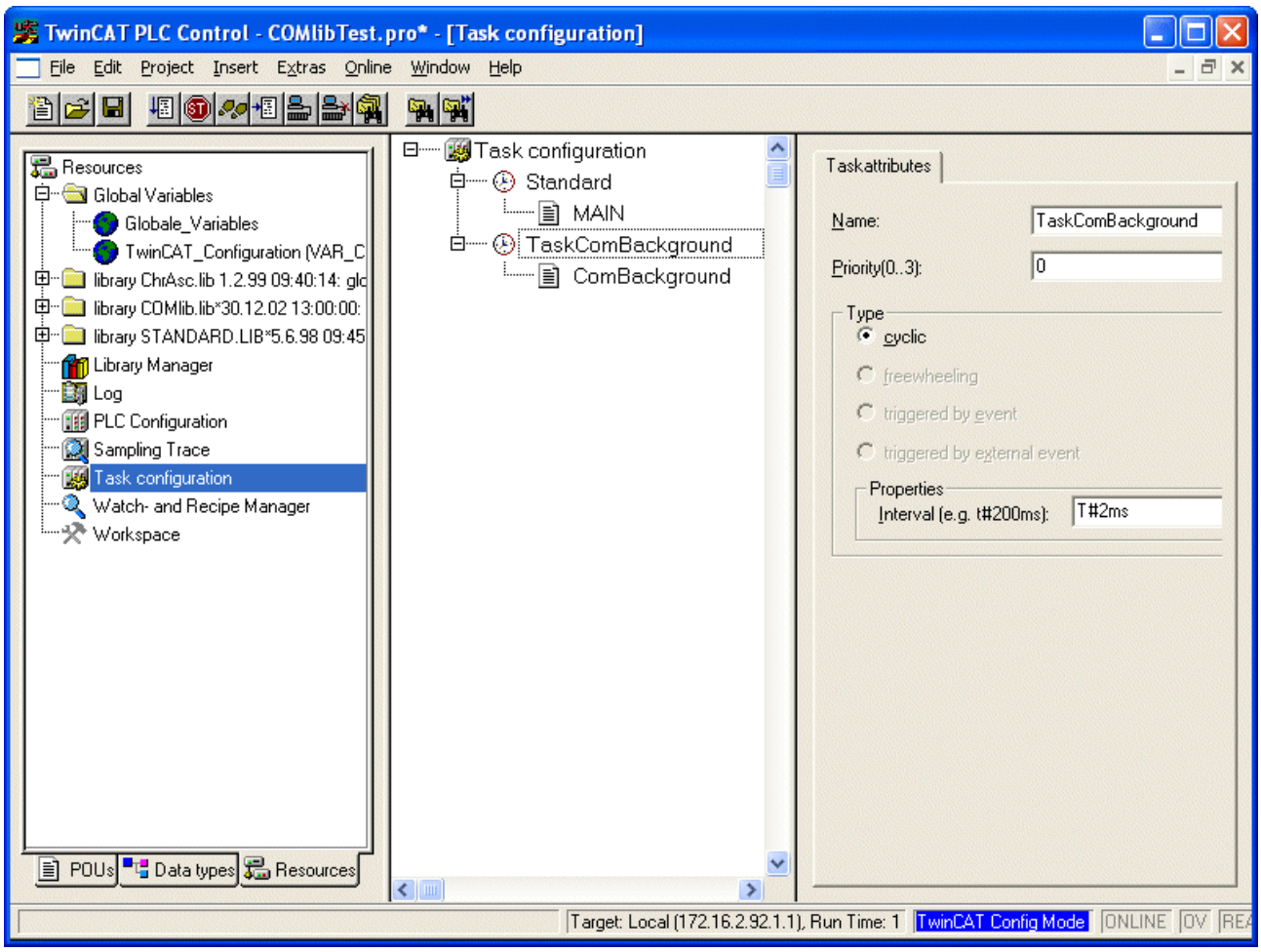
9.2 Globale Variablen

Um eine serielle Schnittstelle anzusprechen, sind vier globale Datenstrukturen notwendig. Zwei stellen die Verbindung zur Hardware in Sende- und Empfangsrichtung her. Zusätzlich werden zwei Datenpuffer als Zwischenspeicher benötigt.



9.3 Taskkonfiguration

Für die Taskkonfiguration muss die Geschwindigkeit der seriellen Schnittstellen beachtet werden (siehe Unterstützte Hardware [▶ 10] und Kommunikationskonzept [▶ 12]). Damit zum Beispiel bei 9600 bps an der seriellen Busklemme alle Daten tatsächlich mit dieser Geschwindigkeit verarbeitet werden können, muss der zugehörige Kommunikationsbaustein mindestens einmal pro Millisekunde aktiv werden. Dementsprechend schnell ist die Task, die den Baustein bedient. einzustellen. Im einfachsten Fall läuft das gesamte PLC-Programm in dieser schnellen Task. Wird die Task langsamer eingestellt funktioniert die Kommunikation, solange die Schnittstelle mit Hardware Handshake arbeitet, in diesem Fall jedoch nicht mit voller Geschwindigkeit. Ohne Handshake können dann Empfangsdaten verloren gehen.



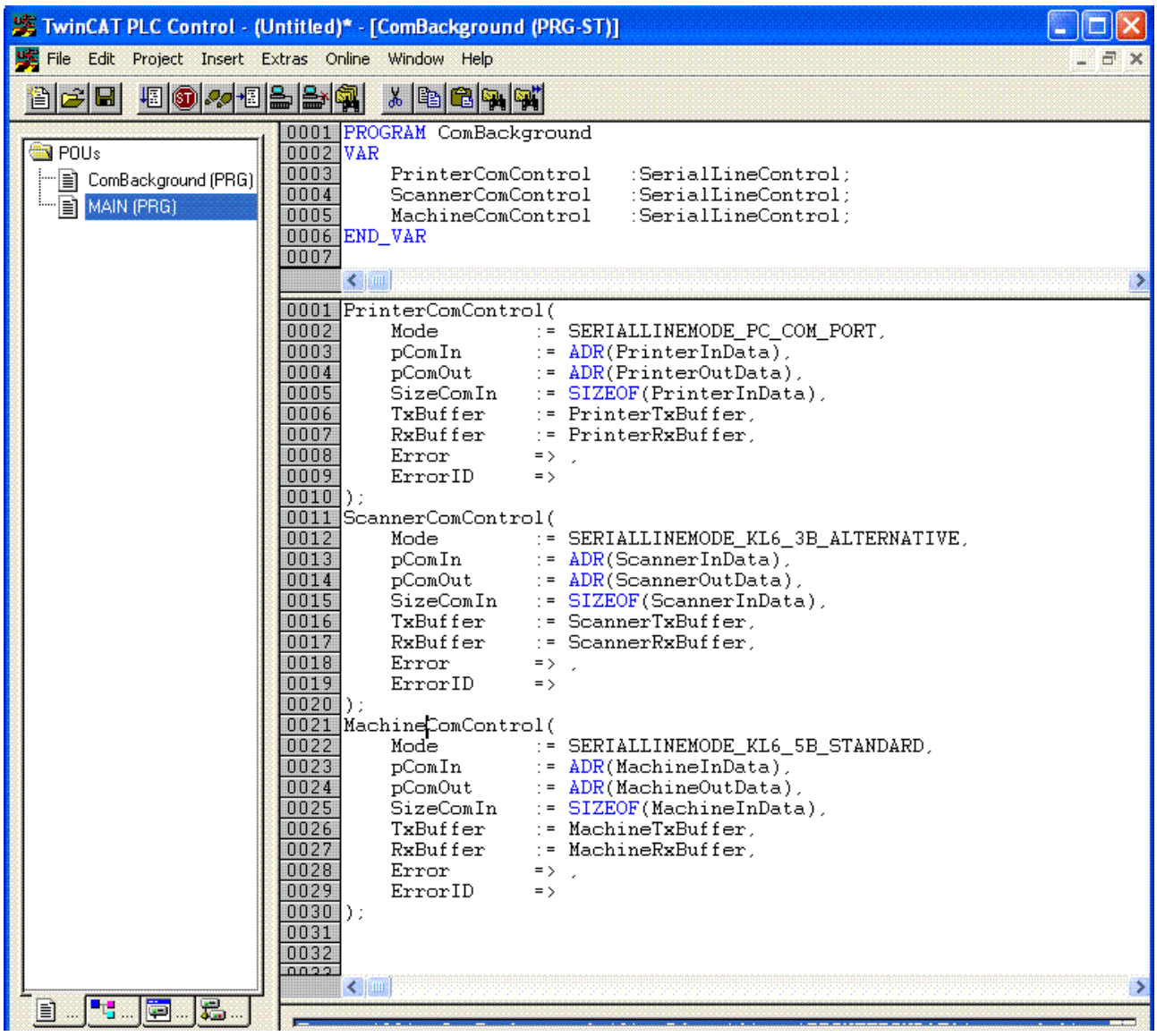
Bei Verwendung von Busklemmen KL6xxx an Buskopplern BKxxxx ist zu beachten, dass die K-Bus-Update-Zeit unterhalb der Zykluszeit der Task liegen muss. Die K-Bus-Update-Zeit kann im TwinCAT SystemManager nach Klick auf den Buskoppler unterhalb der E/A Konfiguration abgelesen werden. Dabei sollte eine Reserve von 10 % bis 20 % berücksichtigt werden. Bei vielen Busklemmen an einem Buskoppler muss die **Zykluszeit der Task** eventuell auf **mindestens 2 ms** eingestellt werden.

9.4 Hintergrundkommunikation

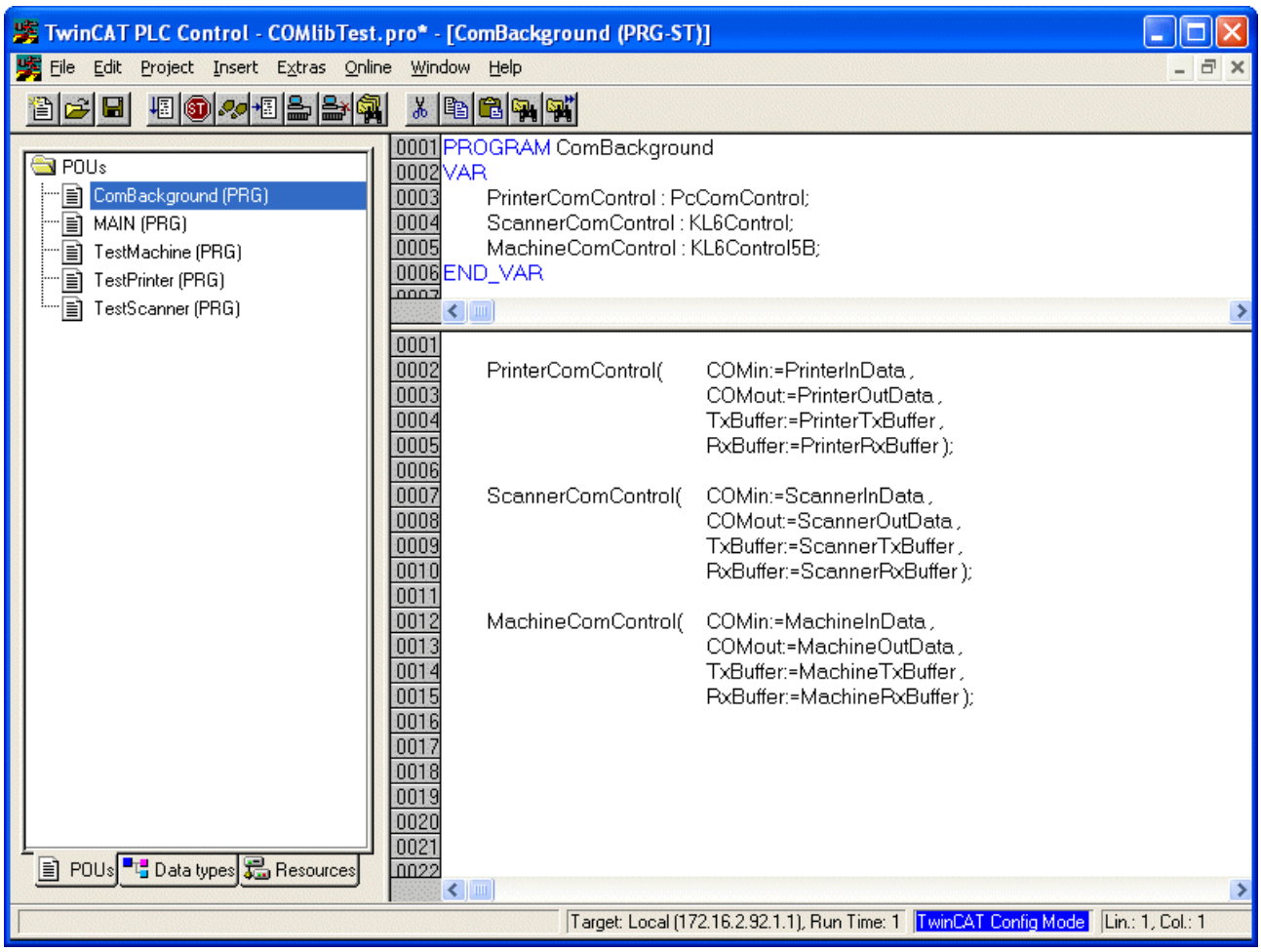
Die Kommunikation zwischen der seriellen Hardware und den Datenpuffer vom Typ ComBuffer [▶ 35] wird in einer separaten schnellen Task abgewickelt.

Siehe auch Kommunikationskonzept [▶ 12].

mit COMlibV2:


TwinCAT PLC Control - (Untitled)* - [ComBackground (PRG-ST)]
File Edit Project Insert Extras Online Window Help
POUs
ComBackground (PRG)
MAIN (PRG)
0001 PROGRAM ComBackground
0002 VAR
0003 PrinterComControl :SerialLineControl;
0004 ScannerComControl :SerialLineControl;
0005 MachineComControl :SerialLineControl;
0006 END_VAR
0007
0001 PrinterComControl(
0002 Mode := SERIALLINEMODE_PC_COM_PORT,
0003 pComIn := ADR(PrinterInData),
0004 pComOut := ADR(PrinterOutData),
0005 SizeComIn := SIZEOF(PrinterInData),
0006 TxBuffer := PrinterTxBuffer,
0007 RxBuffer := PrinterRxBuffer,
0008 Error => ,
0009 ErrorID =>
0010);
0011 ScannerComControl(
0012 Mode := SERIALLINEMODE_KL6_3B_ALTERNATIVE,
0013 pComIn := ADR(ScannerInData),
0014 pComOut := ADR(ScannerOutData),
0015 SizeComIn := SIZEOF(ScannerInData),
0016 TxBuffer := ScannerTxBuffer,
0017 RxBuffer := ScannerRxBuffer,
0018 Error => ,
0019 ErrorID =>
0020);
0021 MachineComControl(
0022 Mode := SERIALLINEMODE_KL6_5B_STANDARD,
0023 pComIn := ADR(MachineInData),
0024 pComOut := ADR(MachineOutData),
0025 SizeComIn := SIZEOF(MachineInData),
0026 TxBuffer := MachineTxBuffer,
0027 RxBuffer := MachineRxBuffer,
0028 Error => ,
0029 ErrorID =>
0030);
0031
0032

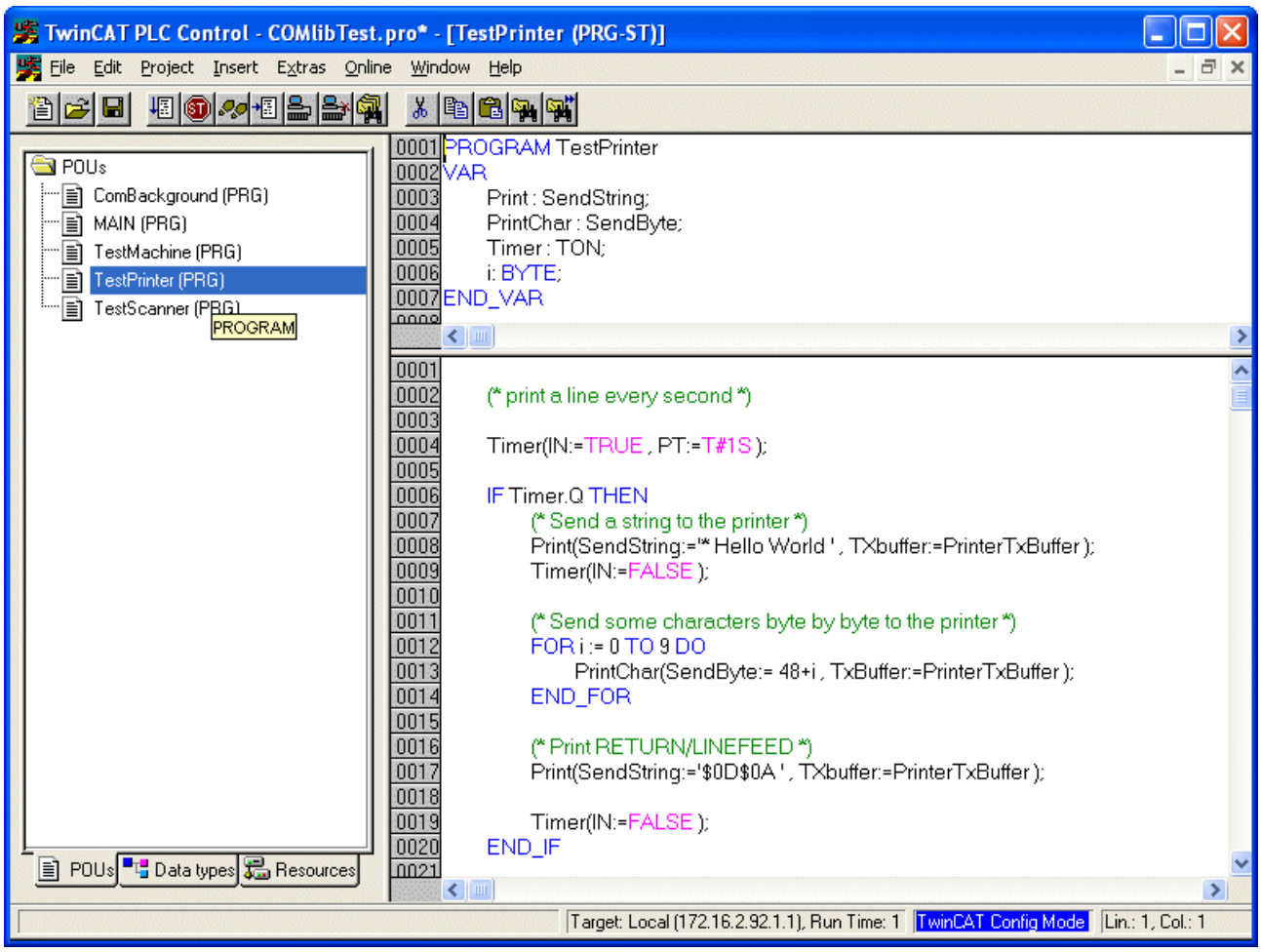
mit COMlib:



9.5 Senden und Empfangen

Daten senden

Daten werden in der Standard-Task gesendet. Im beiliegenden Beispielprogramm werden Daten über die serielle PC-Schnittstelle an einen Drucker gesendet.

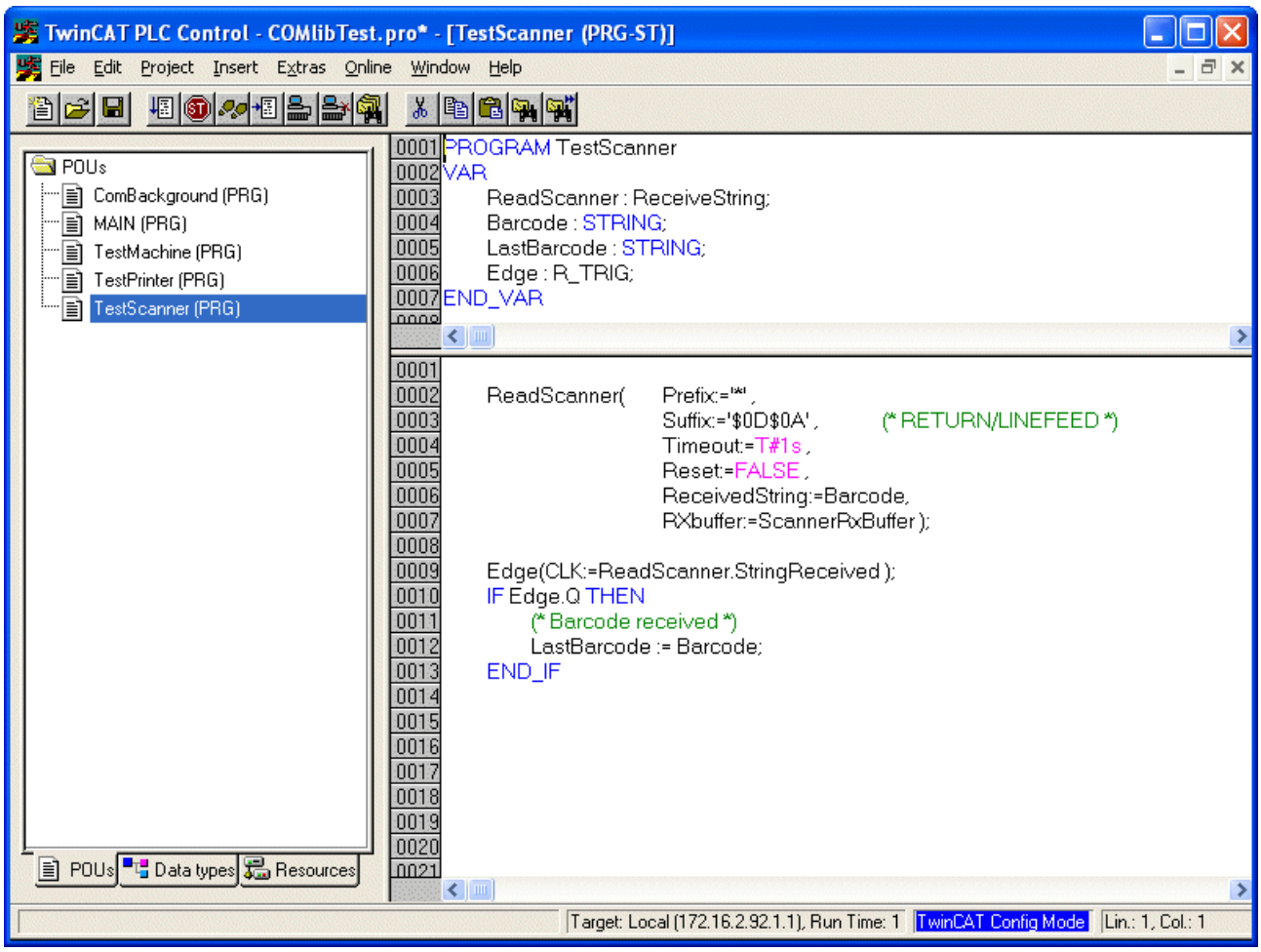


Mögliche Fehler

Während eines SPS-Zyklus dürfen mehrere Zeichen gesendet werden, solange sie vom Sendepuffer aufgenommen werden können. Bei einem Überlauf des Sendepuffers wird der Busy-Ausgang des Sendebausteins nach dem Aufruf TRUE bleiben. Das letzte Zeichen wurde dann nicht gesendet und der Baustein muss im nächsten SPS-Zyklus mit unveränderten Eingangsdaten erneut aufgerufen werden. Der Füllstand eines Puffers kann jederzeit ermittelt werden (z. B. TxBuffer.Count bzw. TxBuffer.FreeByte).

Datenempfang

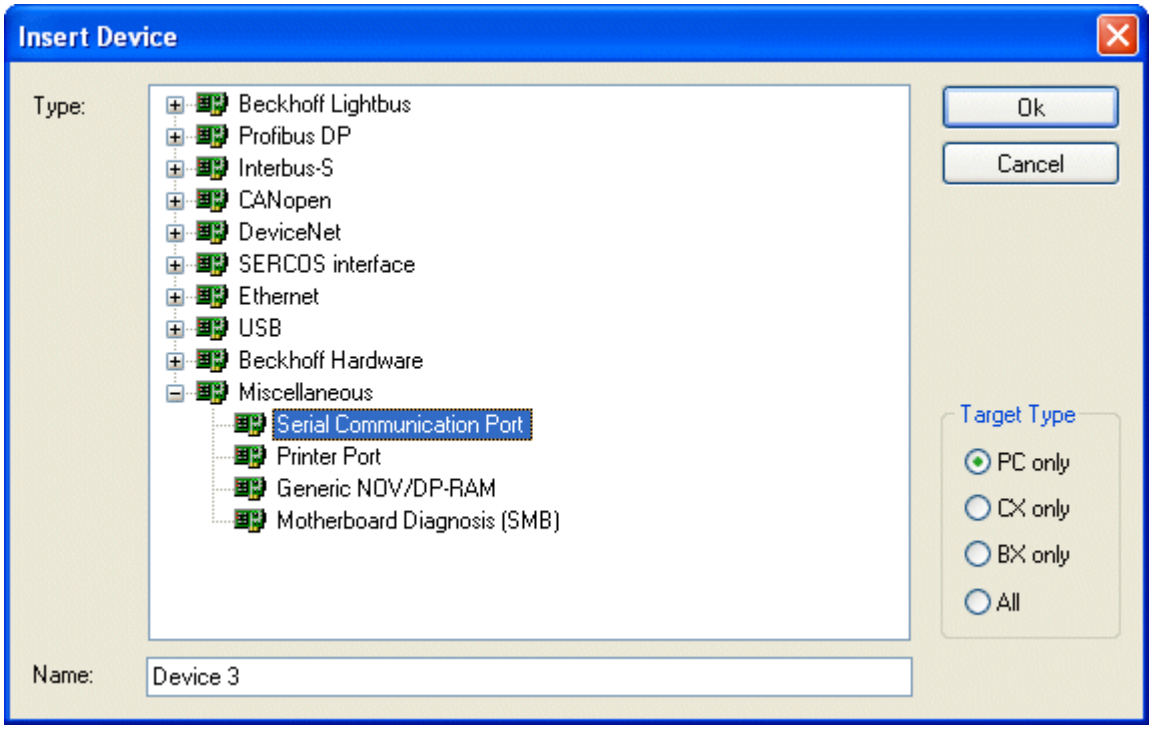
Das Beispielprogramm empfängt einen Barcode von einem Scanner, der an eine serielle Busklemme angeschlossen ist.


TwinCAT PLC Control - COMlibTest.pro* - [TestScanner (PRG-ST)]
File Edit Project Insert Extras Online Window Help
POUs
ComBackground (PRG)
MAIN (PRG)
TestMachine (PRG)
TestPrinter (PRG)
TestScanner (PRG)
POUs Data types Resources
0001 PROGRAM TestScanner
0002 VAR
0003 ReadScanner : ReceiveString;
0004 Barcode : STRING;
0005 LastBarcode : STRING;
0006 Edge : R_TRIG;
0007 END_VAR
0001 ReadScanner(Prefix:='*',
0002 Suffix:='$0D$0A', (* RETURN/LINEFEED *)
0003 Timeout:=T#1s,
0004 Reset:=FALSE,
0005 ReceivedString:=Barcode,
0006 RXbuffer:=ScannerRxBuffer);
0007
0008 Edge(CLK:=ReadScanner.StringReceived);
0009 IF Edge.Q THEN
0010 (* Barcode received *)
0011 LastBarcode := Barcode;
0012 END_IF
0013
0014
0015
0016
0017
0018
0019
0020
0021
Target: Local (172.16.2.92.1.1), Run Time: 1
TwinCAT Config Mode
Lin.: 1, Col.: 1

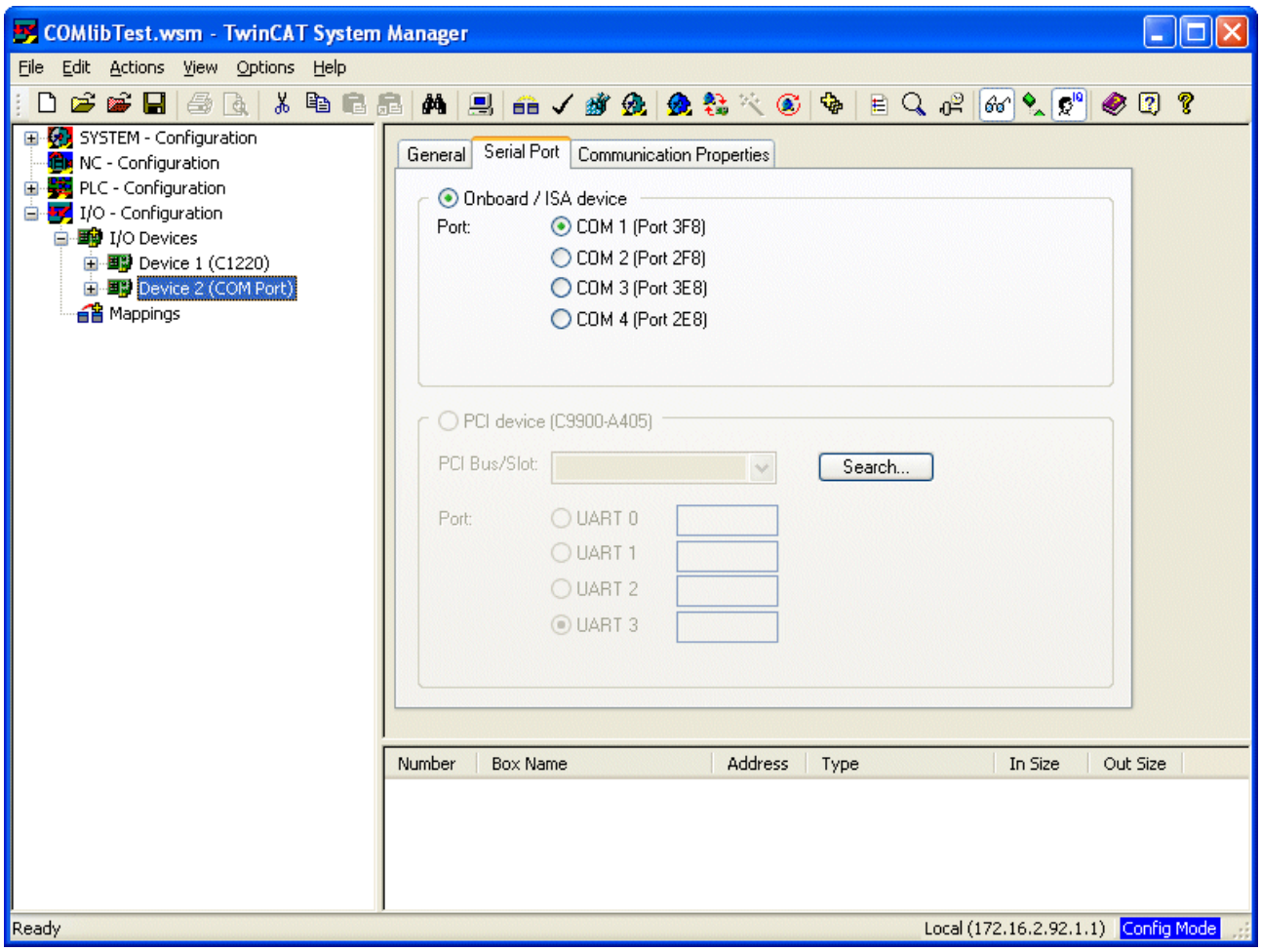
10 Konfiguration im TwinCAT System Manager

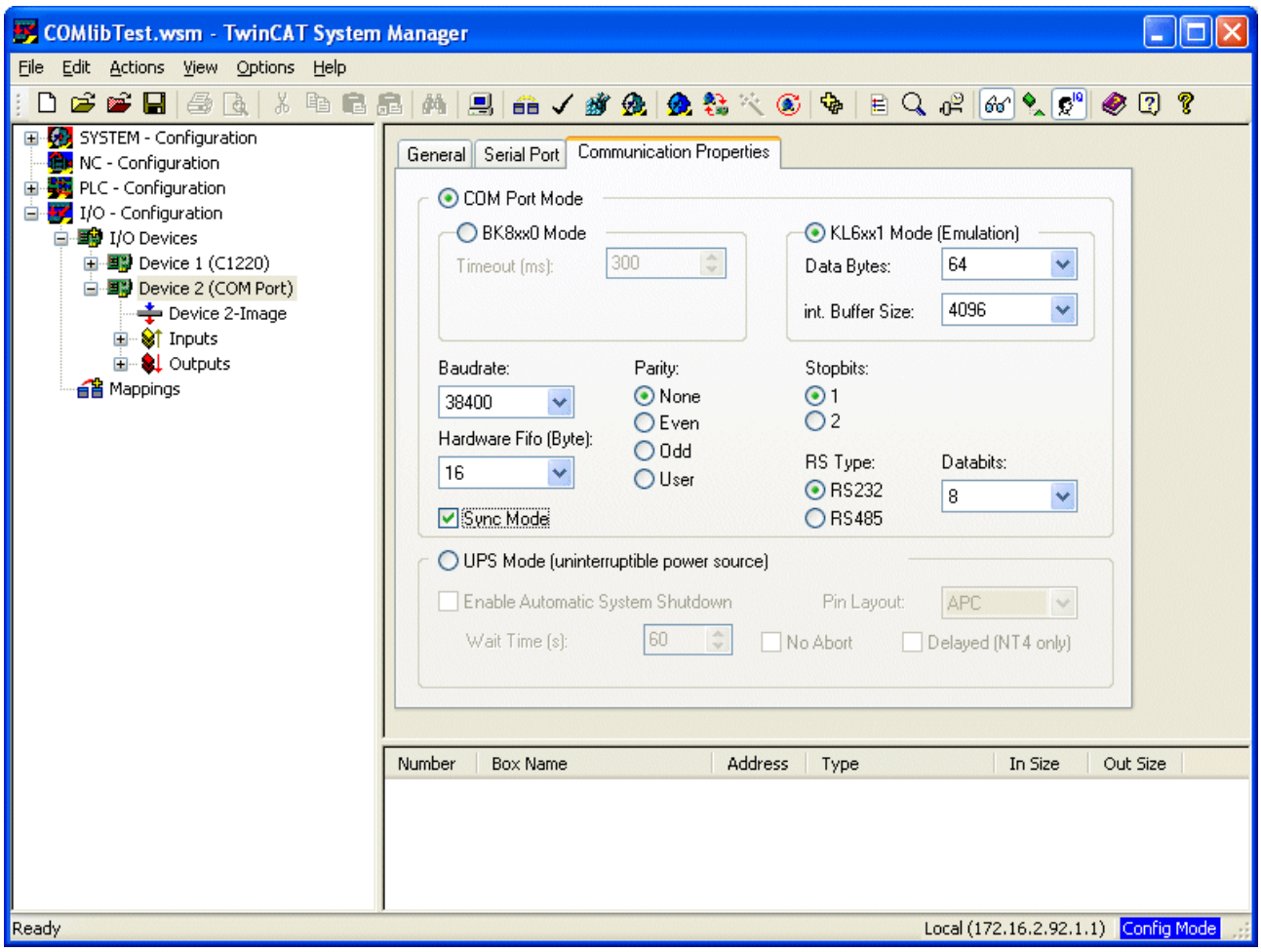
10.1 Serielle PC Schnittstelle

Die serielle Standardschnittstelle des PC wird als neues E/A-Gerät in die E/A-Konfiguration eingetragen.



Anschließend wird die Schnittstelle konfiguriert:


COMlibTest.wsm - TwinCAT System Manager
File Edit Actions View Options Help
SYSTEM - Configuration
NC - Configuration
PLC - Configuration
I/O - Configuration
I/O Devices
Device 1 (C1220)
Device 2 (COM Port)
Mappings
General
Serial Port
Communication Properties
Onboard / ISA device
Port:
COM 1 (Port 3F8)
COM 2 (Port 2F8)
COM 3 (Port 3E8)
COM 4 (Port 2E8)
PCI device (C9900-A405)
PCI Bus/Slot:
Search...
Port:
UART 0
UART 1
UART 2
UART 3
Number
Box Name
Address
Type
In Size
Out Size
Ready
Local (172.16.2.92.1.1)
Config Mode


COMlibTest.wsm - TwinCAT System Manager
File Edit Actions View Options Help
SYSTEM - Configuration
NC - Configuration
PLC - Configuration
I/O - Configuration
I/O Devices
Device 1 (C1220)
Device 2 (COM Port)
Device 2-Image
Inputs
Outputs
Mappings
General
Serial Port
Communication Properties
COM Port Mode
BK8xx0 Mode
Timeout (ms):
300
KL6xx1 Mode (Emulation)
Data Bytes:
64
int. Buffer Size:
4096
Baudrate:
38400
Parity:
None
Even
Odd
User
Stopbits:
1
2
Hardware Fifo (Byte):
16
RS Type:
RS232
RS485
Databits:
8
Sync Mode
UPS Mode (uninterruptible power source)
Enable Automatic System Shutdown
Pin Layout:
APC
Wait Time (s):
60
No Abort
Delayed (NT4 only)
Number
Box Name
Address
Type
In Size
Out Size
Ready
Local (172.16.2.92.1.1)
Config Mode

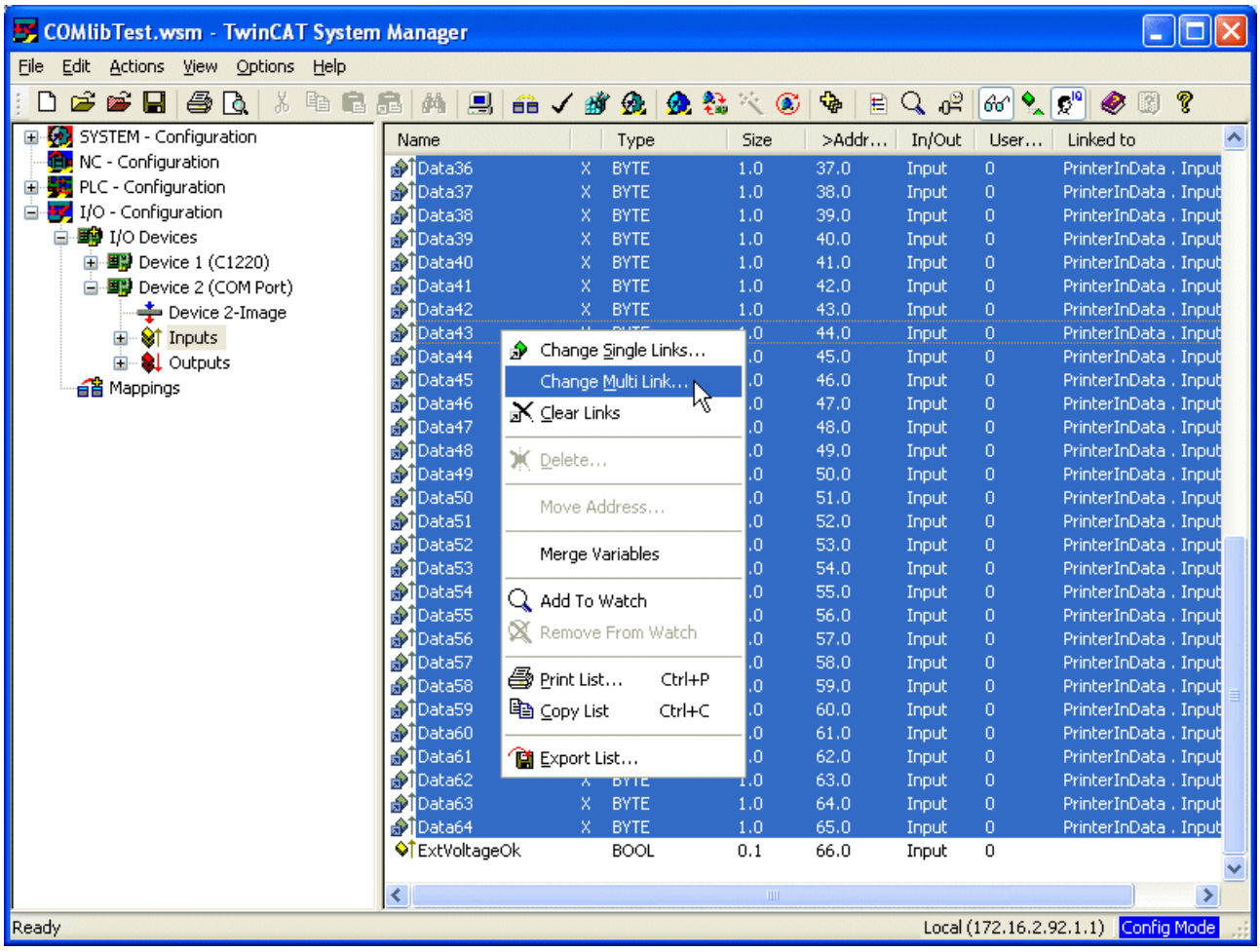
Die COMlib unterstützt nur den KL6xx1 Emulationsbetrieb. Die Anzahl der **Datenbytes** muss auf **64 Byte** eingestellt werden. Die Schnittstellenparameter werden der Anwendung entsprechend eingestellt.

Sync Mode: Im *Sync Mode* wird die Kommunikation mit der Schnittstellenhardware synchron zur Kommunikationstask abgewickelt. Diese Einstellung bietet normalerweise Vorteile bei hohen Baudraten, solange die Zykluszeit der Kommunikationstask kurz genug ist. Bei 115 kBaud werden beispielsweise pro Millisekunde 12 Zeichen empfangen. Die Schnittstelle muss also mindestens in einem 1 ms Zyklus bedient werden, damit das Hardware-FIFO von 16 Bytes nicht überlaufen kann. Bei zu langen Zykluszeiten besteht die Gefahr eines Pufferüberlaufes.

Bei abgeschaltetem *Sync Mode* wird die Schnittstelle unabhängig von der Task-Zykluszeit jede Millisekunde im Windows-Timer-Interrupt bedient. Diese Betriebsart ist nicht echtzeitfähig und es kann bei sehr hoher Rechnerauslastung auch zu längeren Bedienintervallen kommen. Bei sehr hohen Baudraten besteht dann ebenfalls die Gefahr eines Pufferüberlaufes.

Es wird empfohlen, den *Sync Mode* zu aktivieren und die Zykluszeit der Kommunikationstask so an die Baudrate anzupassen, dass der 16 Byte Hardware Puffer nicht überläuft. Bei kleinen Baudraten und gleichzeitig langsamer Kommunkationstask kann der *Sync Mode* evtl. deaktiviert werden.

Im nächsten Schritt werden die Eingänge und Ausgänge mittels Multiverknüpfung mit den korrespondierenden Datenstrukturen in der SPS verknüpft (Typ PcComInData [▶ 39] bzw. PcComOutData [▶ 39]).

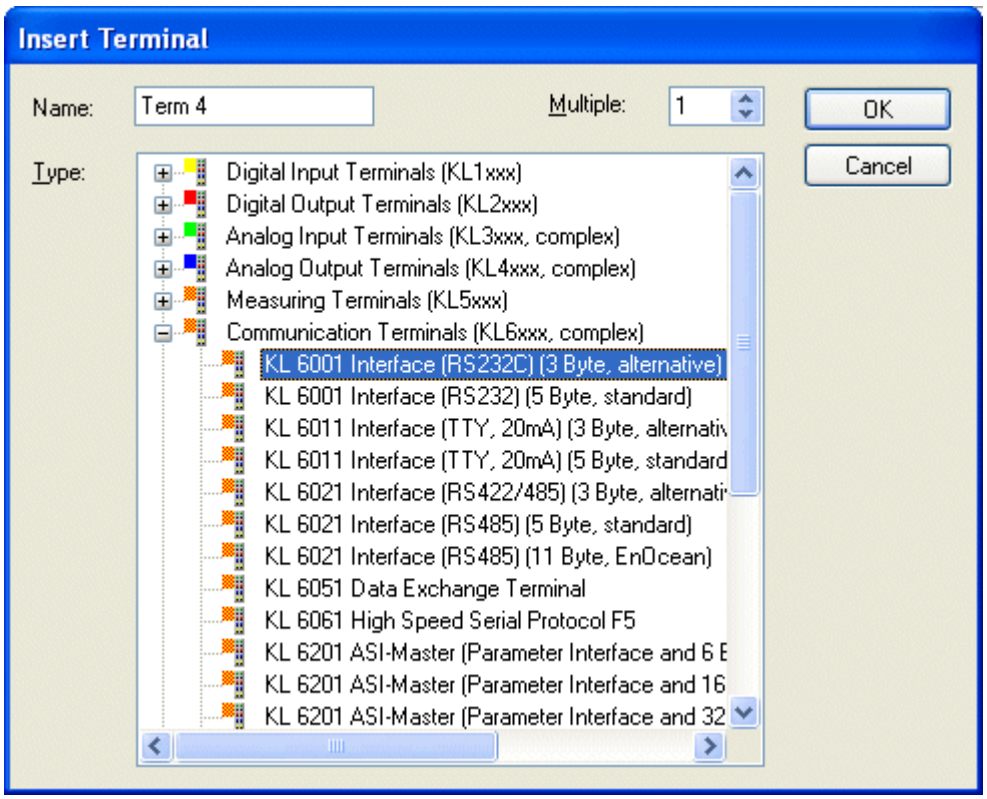


Die neue Konfiguration wird aktiviert und das System damit neu gestartet.

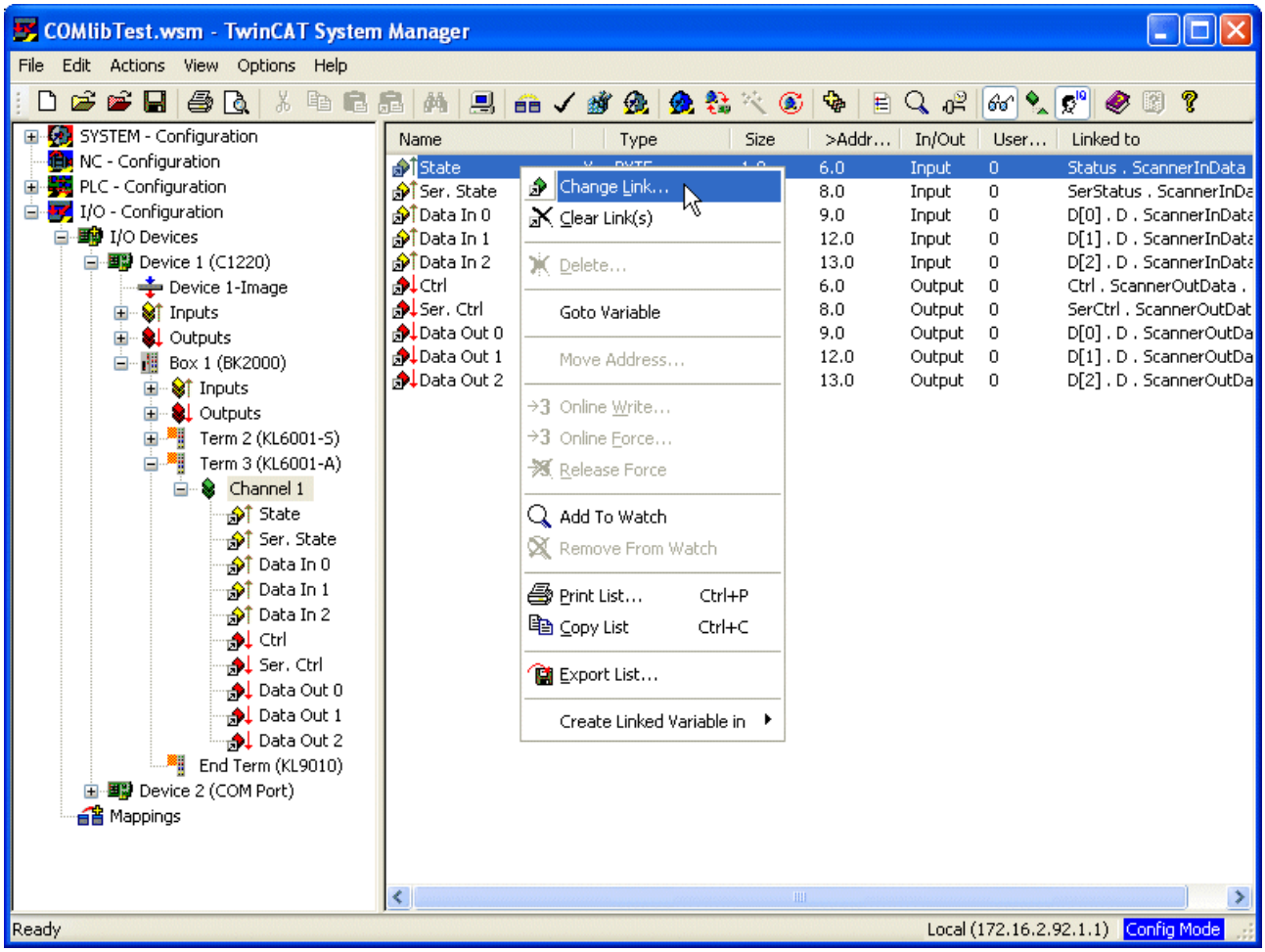
10.2 Serielle Busklemme

KL6xxx im 3-Byte Modus

Die serielle Busklemme wird unterhalb eines Buskopplers in das System eingefügt.



Anschließend werden die Ein-/Ausgangsdaten einzeln mit den korrespondierenden Variablen der SPS verknüpft (Typ KL6inData [▶ 37] bzw. KL6outData [▶ 37]).



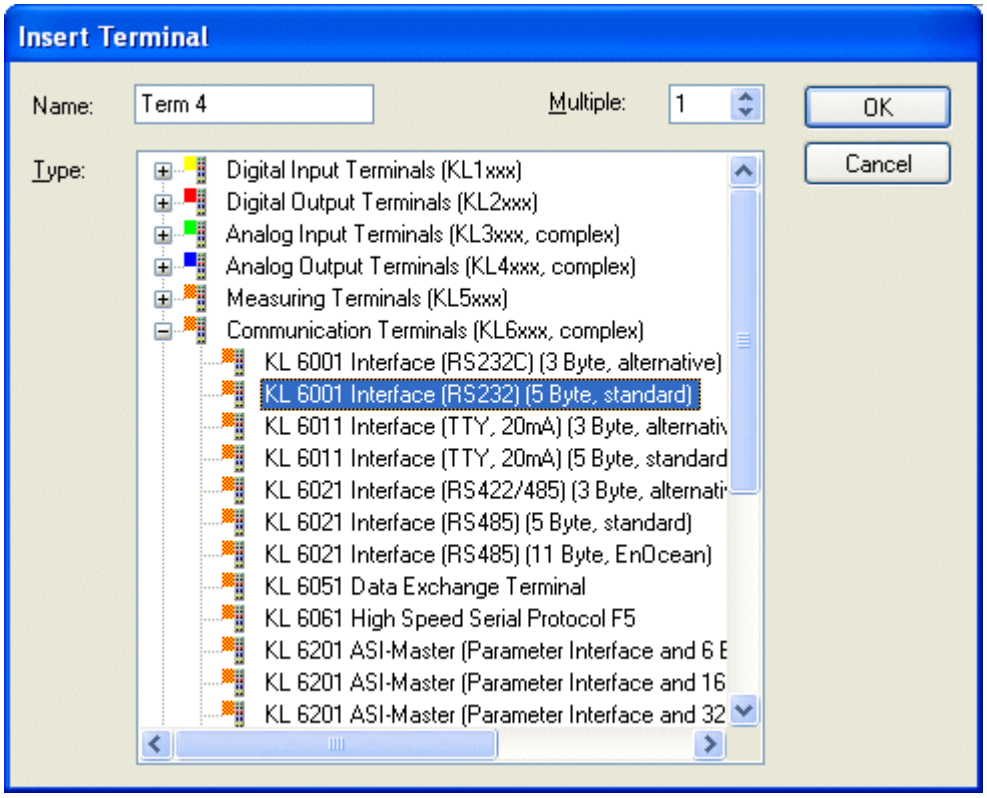
Die neue Konfiguration wird aktiviert und das System damit neu gestartet.

10.3 Serielle Busklemme

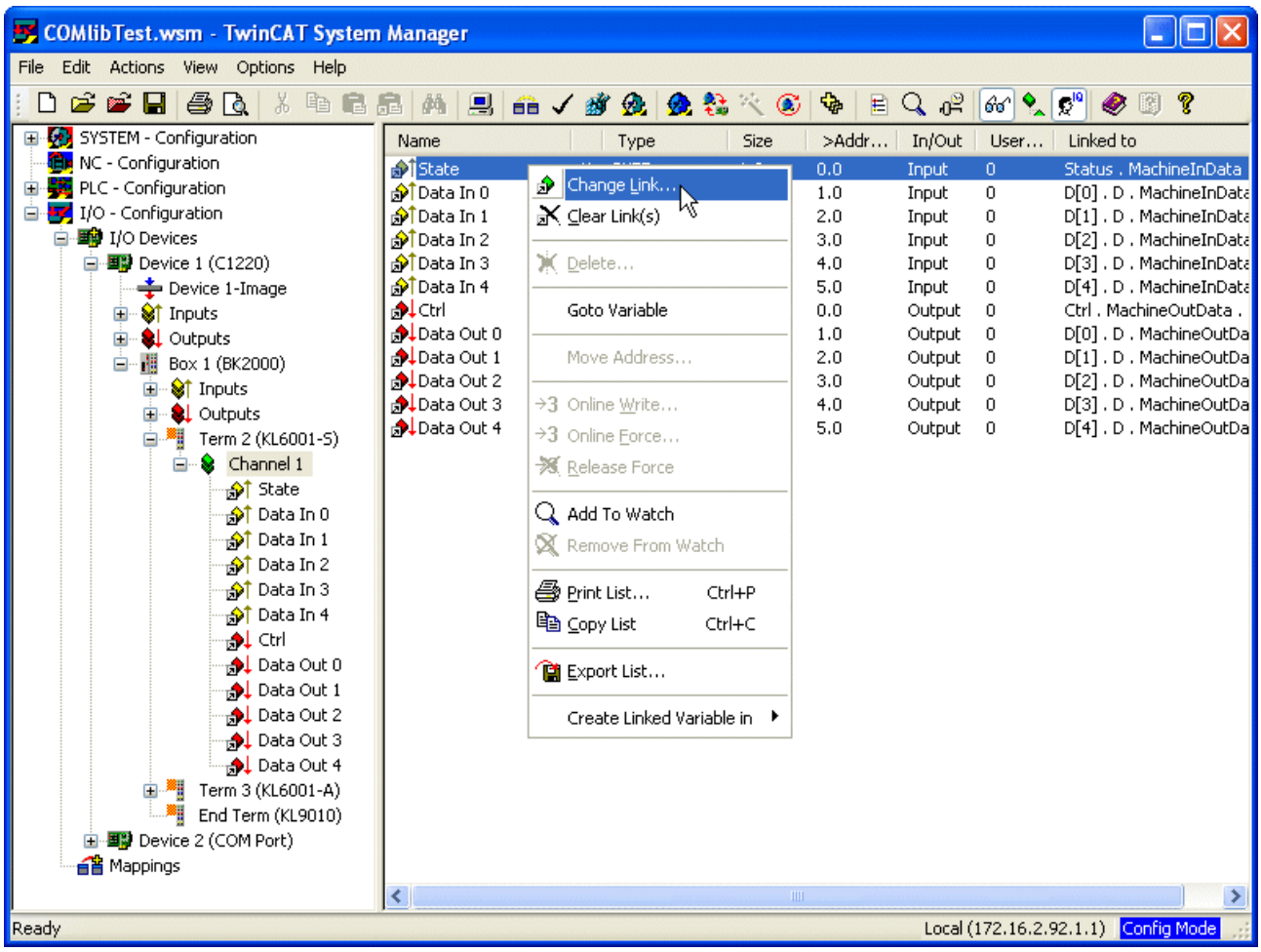
KL6xxx im 5-Byte Modus

Bevor eine Busklemme KL6xxx im 5-Byte Modus verwenden können, muss sie entsprechend umkonfiguriert werden. Diese Konfiguration kann nicht zur Laufzeit durch die ComLib geschehen, sondern durch das Konfigurationsprogramm Beckhoff KS2000. Die Klemme wird damit permanent für den 5-Byte Modus eingestellt.

Die serielle Busklemme wird nun unterhalb eines Buskopplers in das System eingefügt.



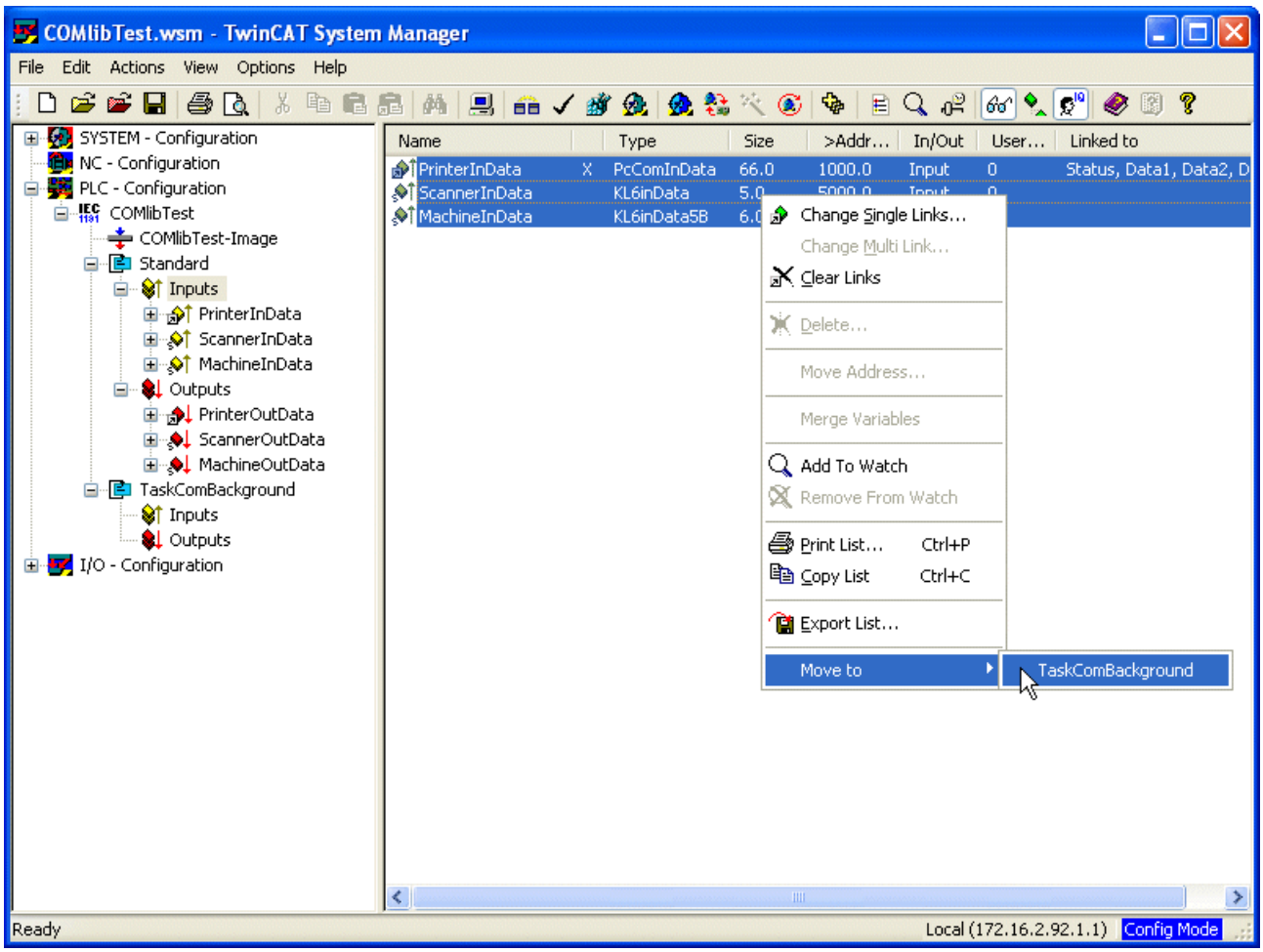
Anschließend werden die Ein-/Ausgangsdaten einzeln mit den korrespondierenden Variablen der SPS verknüpft (Typ KL6inData5B [▶ 37] bzw. KL6outData5B [▶ 37]).



Die neue Konfiguration wird aktiviert und das System damit neu gestartet.

10.4 Task-Zuordnung

Bei der Deklaration globaler Variablen im TwinCAT PLC Control werden die einzelnen Variablen nicht einer bestimmten Task zugeordnet. Daher erscheinen sie im TwinCAT System Manager zunächst unterhalb der Standard-Task.

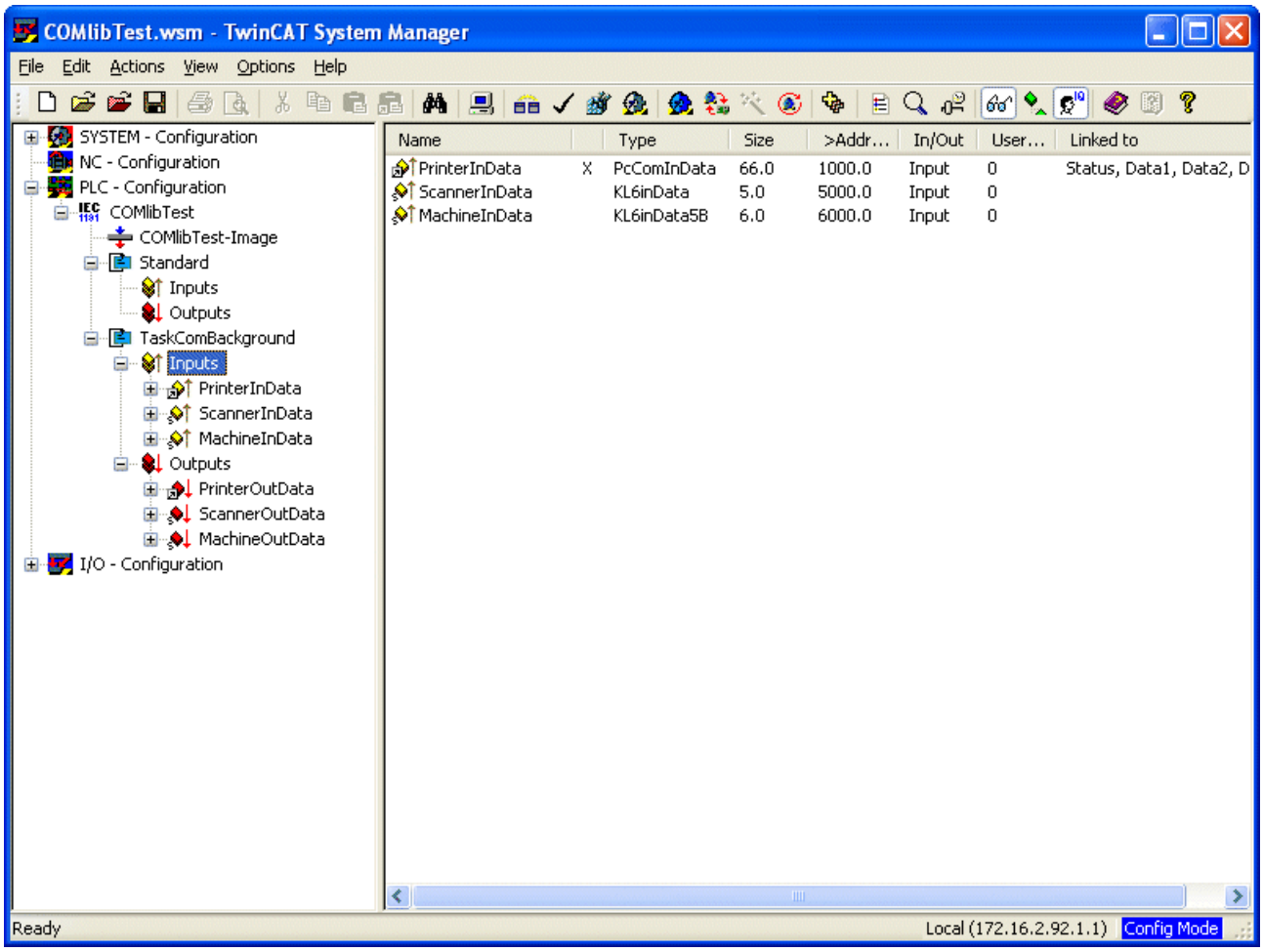


Da die Zykluszeit der Task über den Update-Zyklus auf dem I/O-Bus entscheidet, ist es unbedingt erforderlich, die Datenstrukturen zur Kommunikation zwischen schneller Kommunikations-Task und der Hardware dieser Task zuzuordnen. Die entsprechenden Einträge unterhalb der Ein- und Ausgänge der Standard-Task werden mit der Maus oder über das *Move to* Kommando im Kontextmenü in die Kommunikations-Task verschoben.

Betroffen sind alle Datenstrukturen folgenden Typs:

- KL6inData [▶ 37]
- KL6outData [▶ 37]
- KL6inData5B [▶ 37]
- KL6outData5B [▶ 37]
- KL6inData22B [▶ 38]
- KL6outData22B [▶ 38]
- EL6inData22B [▶ 38]
- EL6outData22B [▶ 38]
- PcComInData [▶ 39]
- PcComOutData [▶ 39]

Kommunikationsdatenstrukturen in der schnellen Kommunikations-Task



11 Beispiele

Die folgenden Beispiele wurden mit unterschiedlicher Hardware entwickelt.

Beispiel 2 - Tutorial

Der Einsatz der Bibliotheksbausteine wird als Tutorial in den Kapiteln 'Einbinden in ein SPS-Programm [▶ 46]' und 'Konfiguration im TwinCAT System Manager [▶ 53]' erläutert. Dabei wird auch auf die Verwendung unterschiedlicher Hardware hingewiesen.

Beispiel 3 - Verwendung von virtuellen Com Ports

mit Installation > v2.3.0 unterstützt

Dieses Beispiel kann für unterschiedliche Anwendungen mit virtuellen Com Ports genutzt werden.
Es ist möglich beliebige Daten zu senden und zu empfangen.

Es handelt sich weniger um ein Beispiel als viel mehr um ein Testprogramm mit dem die Kommunikationsverbindung zu dem USB-Gerät getestet werden kann.
In einer Applikation mit Verwendung eines virtuellen Com Ports ist allein der Aufruf von *SerialLineControlAds* spezifisch. Der Rest der seriellen Kommunikationsaufrufe, wie Daten senden und empfangen, ist identisch zur Verwendung echter serieller Ports.

Beachten Sie die Installationshinweise auf der Übersichtsseite [▶ 8].
Es empfiehlt sich, den VirtualComPort Treiber Ihres Gerätes als ersten Schritt mit einem Windows-Terminalprogramm auf Funktionsfähigkeit zu prüfen.

Es muss keine Verlinkung im TwinCAT System Manager erfolgen.
Das Beispiel kann genutzt werden, um das Gerät komfortabel in Betrieb zu nehmen und zu testen. Das Beispiel verfügt über eine Visualisierung.

Projekt https://infosys.beckhoff.com/content/1031/tcplclibserialcom/Resources/11388463499.zip

Mehr Informationen:
www.beckhoff.de/ts6340

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