BECKHOFF New Automation Technology

Manual | EN

TS6340

TwinCAT 2 PLC Serial Communication

Supplement | Communication

2022-10-20 | Version: 1.1

Table of contents

1 Foreword 5
1.1 Notes on the documentation 5
1.2 Safety instructions 6
1.3 Notes on information security 7
2 Overview 8
3 Supported Hardware 10
4 Communication Principle 12
5 Function Blocks 14
5.1 ReceiveByte 14
5.2 SendByte 14
5.3 ReceiveString 15
5.4 SendString 16
5.5 ReceiveData 17
5.6 SendData 19
5.7 ClearComBuffer 20
5.8 Hardware-dependent Function Blocks 20
5.8.1 Background Communication 20
5.8.2 Configuration 25
6 Help Functions 33
6.1 ASC 33
6.2 CHR 33
7 Data Types 34
7.1 Structures 34
7.1.1 ComBuffer 34
7.1.2 Data structures RegisterList 34
7.1.3 ComSerialConfig 34
7.1.4 Data structures for the KL6xxx serial bus terminal in 3-byte mode 36
7.1.5 Data structures for the KL6xxx serial bus terminal in 5-byte mode 36
7.1.6 Data structures for the KL6xxx serial bus terminals in 22-byte mode 37
7.1.7 Data structures for the EL60xx serial EtherCAT terminal in 22-byte mode 37
7.1.8 Data structures for the COM serial PC interfaces 37
7.2 Enumarated Types for the Communication Library 38
8 Error codes 40
8.1 Error codes: ComError_t 40
8.2 Error codes: AdsSerialComm 40
9 Linking into a PLC Program 45
9.1 Installation 45
9.2 Global Variables 45
9.3 Task Configuration 46
9.4 Background Communication 47
9.5 Sending and Receiving 49
10 Configuration in the TwinCAT System Manager 52

10.1 Serial PC Interface 52
10.2 Serial Bus Terminal 54
10.3 Serial Bus Terminal 56
10.4 Task Assignment 57

11 Examples 60

1 Foreword

1.1 Notes on the documentation

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

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

Disclaimer

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

Trademarks

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

Patent Pending

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

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

Copyright

1.2 Safety instructions

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Description of symbols

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

⚠ DANGER

Serious risk of injury!

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

⚠ WARNING

Risk of injury!

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

⚠ CAUTION

Personal injuries!

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

NOTE

Damage to the environment or devices

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

Tip or pointer

This symbol indicates information that contributes to better understanding.

1.3 Notes on information security

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

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

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

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

2 Overview

The TwinCAT PLC library COMlibV2 supplies function blocks and data structures for serial data communication. The library supports the serial Beckhoff KL6xxx Bus Terminals, EL60xx EtherCAT Terminals, the standard COMx PC interfaces and virtual COM ports.

In detail, the library provides function blocks for the following functions:

- Sending and receiving byte by byte
- Sending and receiving strings
- Sending and receiving transparent data
- Configuration of the serial KL6xxx Bus Terminals (baud rate, handshake etc.)
- Operation of the serial bus terminals
- Operation of the serial EtherCAT terminals

All function blocks support multiple instances, so that, depending on the processing capacity of the PC, any number of serial interfaces can be operated in parallel.

The predecessor of the TwinCAT PLC library COMlibV2 was the TwinCAT PLC library COMlib.

System requirements for use with virtual COM ports

available with installation >= v2.2.4

The installation of the TwinCAT PLC Serial Communication product offers the option to install an ADS server. This is only necessary if virtual COM ports are to be accessed. Otherwise, the server does not need to be installed.

There are special system requirements for using the PLC library with virtual COM ports in Windows:

- Programming environment:
  - XP, XPe, WES, Win7, WES7, Win10
  - TwinCAT installation level: TwinCAT PLC or higher
  - TwinCAT system version 2.11 build 1544 or higher
  - **COMlibV2.lib** This PLC library must be included in the PLC project.
    (The libraries Standard.lib; TcBase.lib; TcSystem.lib are automatically included in the project.)
    Up to version < v2.3.0 of the library COMlibV2.lib only the library Standard.lib was included. It is now necessary to ensure that all three libraries are available!
- Target platform:
  - PC or CX (x86, x64): XP, XPe, WES, Win7, WES7, Win10, CE 6.0
  - TwinCAT PLC runtime system version 2.10 build 1340 or higher
- Installation:
  - The product installation optionally offers the possibility to install the TcAdsSerialCommServer.
  - The ADS server must be installed on the system on which the virtual COM port is located.
  - To install the ADS server on a Windows CE system, the following steps must be performed:

First install the product as usual on your programming PC.

After the installation you will find in the folder: **...\TwinCAT\CE\TcPLCSerialComm** a cabinet file for the CE runtime system.

Copy the cabinet file to a folder on the CE runtime system.

On the CE system: install (by double-clicking on the cabinet file) the CE components after '\Hard Disk\System'.

Restart the CE device. The TcAdsSerialCommServer is started automatically with the CE operating system. (Please note that the ADS server will not start automatically until you use the "StartUp.exe" from version 1.0.0.47)

Status of the documentation: 11.04.2012

3 Supported Hardware

Serial Bus Terminal

KL6xxx in 3-byte mode

The standard version, as supplied, of the serial Beckhoff Bus Terminal is operated in 3-byte mode. In other words, a bus telegram can transmit or receive 3 data bytes to or from the terminal. Since every data exchange between the PLC and the bus terminal requires 3 PLC cycles, the effective transfer rate is one byte per cycle.

The maximum effective data transmission rate in bits per second depends on the cycle time, T, of the PLC, and on the number of useful bits in each data byte transferred, LB:

Bps = LB / T

LB = 1 start bit + n data bits + p parity bits + m stop bits

The maximum effective data transmission rate is limited by the physical baud rate programmed into the bus terminal.

In the case of bus terminals, the K-bus update time of the bus coupler must be considered when selecting the cycle time (see Task configuration [▶ 46]).

KL6xxx in 5-byte mode

The serial bus terminal can be reprogrammed offline by means of a configuration program (Beckhoff KS2000), so that, in 5-byte mode, 5 data bytes at a time can be transferred to or from the terminal. 3 PLC cycles are still necessary for each exchange. The effective data rate for a given cycle time of the PLC is thus 5/3 greater than in 3-byte mode.

Bps = (LB * 5/3) / T

The bus terminals cannot be re-programmed while the PLC is running, since the 3-byte and 5-byte modes differ in the register mapping and in the TwinCAT System Manager configuration.

In the case of bus terminals, the K-bus update time of the bus coupler must be considered when selecting the cycle time (see Task configuration [▶ 46]).

KL6xxx in 22-byte mode

The serial bus terminal can be supplied with a 24-byte process image as a special type, so that packets of 22 data bytes can be transferred to and from each terminal. 3 PLC cycles are still necessary for each exchange.

Bps = (LB * 22/3) / T

This bus terminal is only supported from version 2.0 of the serial communication library (ComLibV2). In the case of bus terminals, the K-bus update time of the bus coupler must be considered when selecting the cycle time (see Task configuration [▶ 46]).

Serial EtherCAT Terminal

EL60xx in 22-Byte Mode

The serial EtherCAT terminal is operated in 22-byte mode, so that 22 data bytes at a time can be transferred to or from the terminal. 3 PLC cycles are still necessary for each exchange.

Bps = (LB * 22/3) / T

Terminals' parameterization will be conducted by the CoE-Online tab in the TwinCAT System Manager (double-click on the referring object).

Serial PC Interface

The serial PC interface (COM1, COM2 etc.) is handled by the TwinCAT system similarly to the serial bus terminal, but use larger data transfer buffers than the serial bus terminal. The library uses a 64 byte buffer, so up to 64 data bytes are transferred at once between the PLC and the interface driver. 3 PLC cycles are again needed for the exchange of a data block with the serial PC interface.

Bps = (LB * 64/3) / T

Virtual serial COM port

The TwinCAT system also supports a virtual serial COM port (COM1, ..., COM255), which is available in Windows. This does not require any configuration of the process image in the TwinCAT System Manager. Parameterization takes place directly in the PLC with the function blocks provided.

This communication connection is not real-time capable.

The baud rate can be set between 150 baud and 115200 baud.

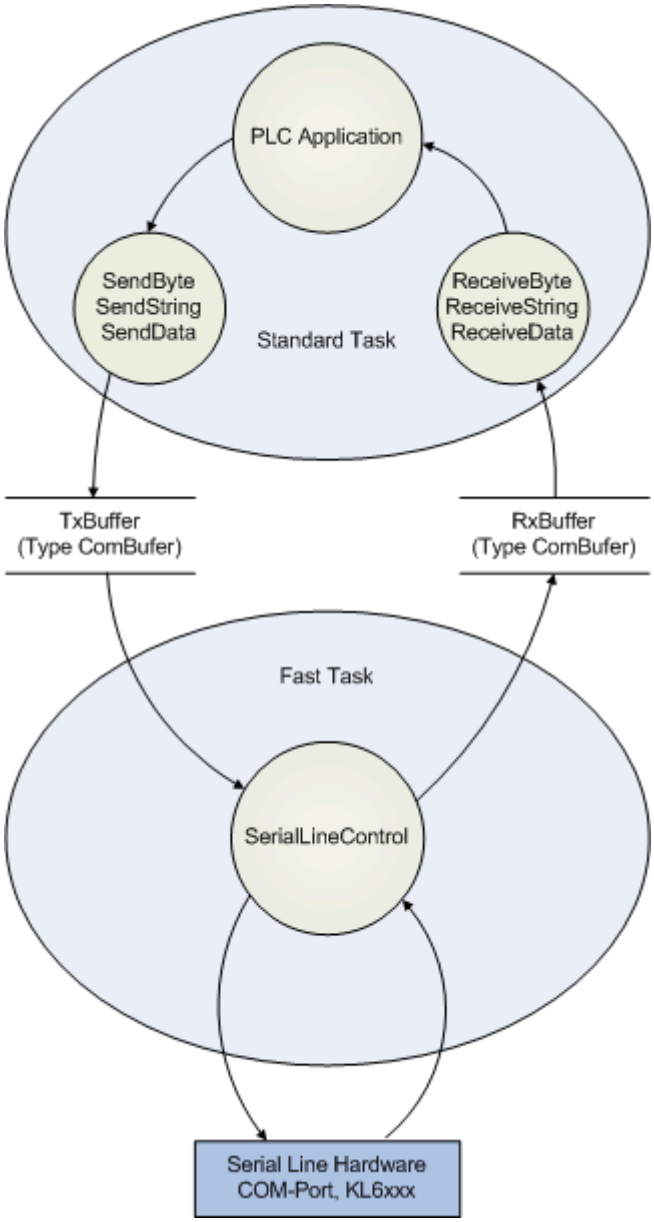
The product is designed for use with a single virtual COM port per target system. However, in principle it is possible to use of several virtual COM ports on one system, provided function tests are carried out.

4 Communication Principle

Background Communication

As described in the section on Supported Hardware [▶ 10], the maximum effective data transfer rate depends in part on the PLC cycle time. So, for example, for communication with the serial bus terminal at an effective rate of 9600 bps, a cycle time of 1 ms is required. In many larger applications such a short cycle time for the whole PLC would heavily load the control computer.

Since for most applications longer cycle times of, for instance, 10 ms are more than adequate, it is possible with the aid of this library to decouple the data traffic between the PLC and the hardware from the rest of the PLC application. Two tasks are created in the PLC program for this purpose. The standard task runs with the conveniently long PLC cycle time of, for example, 10 ms, while a second communication task runs with a faster cycle taking, for example, 2 ms.



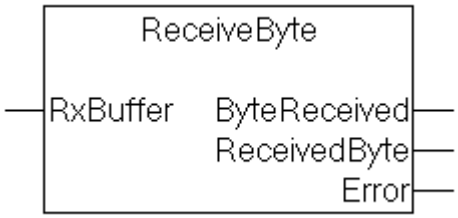
Data buffers of type ComBuffer [▶ 34] are used to decouple the different speeds of the fast communication task and the standard task. They are written and read asynchronously.

The function blocks described later for receiving and sending data (SendByte [▶ 14], SendString [▶ 16], SendData [▶ 19] etc.) make use of only an additional data buffer for data exchange, and are thus independent of the hardware being used. In all cases, a communication block SerialLineControl [▶ 20] is called in the fast task, as well as the send and receive blocks. This handles the data traffic between the data buffer and the hardware with maximum speed in the background. If a COM port or terminal with 22 byte data interface is used a second task is not needed for low baud rates. Then the communication block SerialLineControl [▶ 20] can be called in the standard task.

The communication with virtual COM ports via SerialLineControlADS [▶ 21] uses an ADS server. Data exchange is handled asynchronously via ADS and is managed in intermediate buffers. The server itselfs receives the incoming data of the virtual COM port independent of the PLC. So the maximal effective data transfer don't depend on the PLC task cycle time.
Only the reaction time for incoming data would be decreased by a faster task cycle time. The full delay can't be defined because the virtual-com-port-drivers and the ADS server do not run in realtime.
The communication block *SerialLineControlADS* usually is called in the standard task. A second task is not needed.

5 Function Blocks

5.1 ReceiveByte



Interface

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

Description

The ReceiveByte block receives a single character from the interface corresponding to the input variable RxBuffer. If ByteReceived=TRUE after the call, then the data byte received is available in the output variable ReceivedByte. Otherwise no data has been received.

Whenever a the ReceiveByte function block is processed in a PLC task that is running slower than the communication with the hardware, it must be remembered that more than one character can be made available in each PLC cycle. The characters received should therefore be read out within a loop:

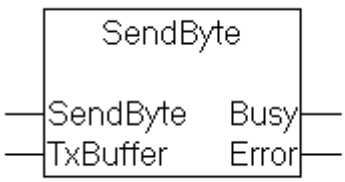
```
REPEAT
    Receive(RXbuffer:=RXbuffer);
    IF Receive.ByteReceived THEN
    (* Zeichen auswerten *)
    END_IF
UNTIL NOT Receive.ByteReceived
END_REPEAT
```

The number of passes through the loop is inevitably limited by the size of the receive data buffer (presently 300 bytes), so that an infinite loop need not be feared.

Also see about this

- Error codes: ComError_t [▶ 40]
- ComBuffer [▶ 34]

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
```

Description

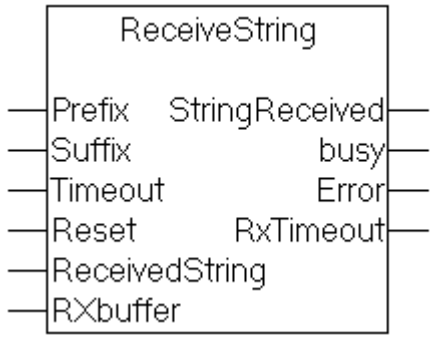
SendByte sends a single character to the interface corresponding to the input variable **TxBuffer**. For as long as **Busy**=TRUE, the transmission is not completed. The character was successfully sent when **Busy**=FALSE and Error=0.

As long as the send data buffer can still accept data, more than one character can be sent in a single PLC cycle. This is, however, only worthwhile if the buffered characters will be transmitted to the hardware by a faster communication task.

Also see about this

- Error codes: ComError_t [▶ 40]
- ComBuffer [▶ 34]

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
```

Description

ReceiveString receives a string of characters from the interface corresponding to the input variable **RxBuffer**, storing it in the output variable **ReceivedString**. The start and end of the string are recognised by various mechanisms, which can be combined with one another:

- **Prefix** If a string is supplied to the input variable prefix, the first characters must be the same as this prefix. Other characters are disallowed. If no prefix is supplied (an empty string), the received string starts with the first received character.
- **Suffix** If a string is supplied to the input variable suffix, the input data is read until the end of the received string agrees with the suffix. If during this process the received data reach the maximum length of the receive string, a COMERROR_STRINGOVERRUN error is generated. If an empty string is supplied as the suffix, a timeout must be defined instead, since otherwise the end of the character string cannot be recognised.
- **Timeout** If a timeout is supplied to the block, then characters will be received until a correspondingly long interval has elapsed after reception of a character. The received string consists of the characters received up to that point. Suffix and timeout may be combined. If a suffix is supplied, the timeout may be 0.

As soon as the output StringReceived becomes TRUE, the received data are ready in the ReceivedString variable.

Reset

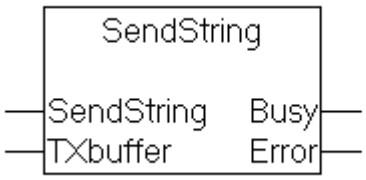
Setting the Reset input will reset the block from the receive state into the initial state. Reset is only necessary in exceptional cases, such as when the expected string can not be received.

ReceivedString has a standard length of 80 characters. For some applications this length may be too short. In this case the **ReceiveString255** function block can be used. The only difference is a length of 255 characters for the ReceivedString.

Also see about this

- Error codes: ComError_t [▶ 40]
- ComBuffer [▶ 34]

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
```

Description

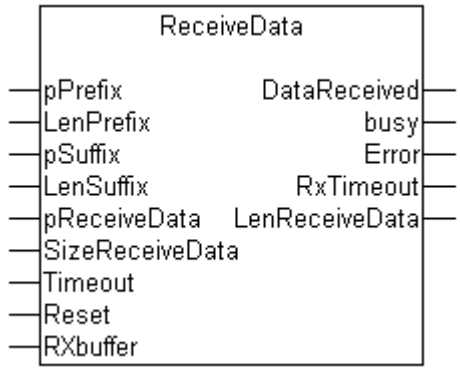
SendString sends a string of characters to the interface corresponding to the input variable **TxBuffer**. For as long as **Busy**=TRUE, the transmission is not completed. The data has successfully been sent when **Busy**=FALSE and Error=0.

SendString has a standard length of 80 characters. For some applications a longer string is desirable. In this case the function block **SendString255** can be used. The only difference is a length of 255 characters for the SendString.

Also see about this

- Error codes: ComError_t [▶ 40]
- ComBuffer [▶ 34]

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
```

Description

ReceiveData receives data of any type from the interface corresponding to the input variable **RxBuffer**, storing it in the ReceiveData variable. The start and end of the data stream are recognised by various mechanisms, which can be combined with one another:

- **Prefix**

  If a variable is passed to the input variable prefix, the first characters of the received data must be the same as this prefix. Other characters are discarded. If no prefix is supplied (null), the receive data starts with the first received character.

- **Suffix**

If an input variable suffix is supplied, the input data is read until the end of the receive data agrees with the suffix. If during this process the received data reach the maximum length of the receive string, a COMERROR_STRINGOVERRUN error is generated. If no suffix (null) is supplied, a timeout must be defined instead, since otherwise the end of the data stream cannot be recognised.

- **Block Size**

  If no suffix is supplied, up to SizeReceiveData characters are received.

- **Timeout**

  If a timeout is supplied to the block, then characters will be received until a correspondingly long interval has elapsed after reception of a character. The receive data consists of the characters received up to that point. Suffix and timeout may be combined. If a suffix is supplied, the timeout may be 0.

As soon as the output **DataReceived** becomes TRUE, the receive data is ready in the ReceiveData variable. The number of characters received is given in **LenReceiveData**.

pPrefix

pPrefix is the address of any kind of data structure that is passed to the block using ADR(variable name). LenPrefix indicates the number of data bytes in the prefix.

LenPrefix

LenPrefix indicates the number of data bytes in the prefix.

pSuffix

pSuffix is the address of any kind of data structure that is passed to the block using ADR(variable name).

LenSuffix

LenSuffix indicates the number of data bytes in the suffix.

pReceiveData

pReceiveData is the address of the receive data, and is found by means of ADR(receive data). The receive data is placed in the variables to which pReceiveData points.

SizeReceiveData

SizeReceiveData is found by means of SIZEOF(receive data), and indicates the maximum size of the receive data.

Timeout

Timeout defines the maximum interval between two received characters. Timeout monitoring becomes effective after the first character. This means that timeout cannot be used to detect whether an expected telegram arrives or not. This is monitored externally.

Reset

Setting the Reset input will reset the block from the receive state into the initial state. Resetting is only necessary in exceptional cases, if, for example, the expected data could not be received and the block remains busy.

DataReceived

DataReceived becomes TRUE as soon as the receive data is valid. The output remains TRUE for precisely one cycle, which means that it is necessary to evaluate the received data immediately.

Busy

Busy becomes TRUE after the first character has been received, and goes FALSE as soon as the data has been received or an error has occurred.

Error

If a fault occurs, Error will contain an error code. The error code is defined by the data type ComError_t, and this is used to display it in text form at runtime.

RxTimeout

RxTimeout becomes TRUE if the maximum interval between two received characters is exceeded. This causes data reception to be aborted, and the characters received up to this point are available. If a suffix is not being used, then detection of a timeout does not represent a fault, but indicates the normal end of the receive data. If, on the other hand, a suffix was being used, it was not possible to receive this.

LenReceiveData

LenReceiveData indicates the actual number of data bytes received, and can be less than or equal to SizeReceiveData.

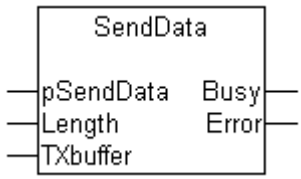
RxBuffer

RxBuffer is the receive data buffer corresponding to the interface in use.

Also see about this

- Error codes: ComError_t [▶ 40]
- ComBuffer [▶ 34]

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
```

Description

SendDatasends the contents of a variable of any type to the interface corresponding to the input variable **TxBuffer**. For as long as **Busy**=TRUE, the transmission is not completed. The data has successfully been sent when **Busy**=FALSE and Error=0.

pSendData

pSendData is the address of the send data, and is determined by means of ADR(send data).

The data must not be changed as long as Busy is TRUE and the transmission is not completed.

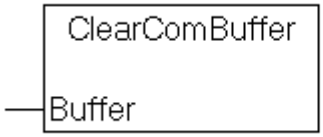
Length

Length is the number of data bytes to be sent, and can be smaller than or equal to the size of the data structure being used. If the entire contents of a variable are to be sent, then the length can be determined using SIZEOF(send data).

Also see about this

- Error codes: ComError_t [▶ 40]
- ComBuffer [▶ 34]

5.7 ClearComBuffer



Interface

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

Description

The communication buffer, **Buffer**, internal to the PLC, is cleared.

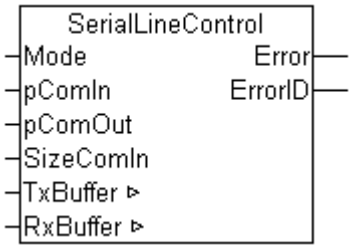
Also see about this

- ComBuffer [▶ 34]

5.8 Hardware-dependent Function Blocks

5.8.1 Background Communication

5.8.1.1 SerialLineControl



The SerialLineControl function block looks after the communication between a serial interface (KL60xx, EL60xx or COM interface) and the PLC. The function block is called cyclically, and places received data into the *RxBuffer*. Data made available in the *TxBuffer* transmit buffer is sent to the interface at the same time.

Because the function operates independently of the application, it is referred to as background communication and can, particularly in the case of serial bus terminals, also operate in a fast task (see Communication concept [▶ 12] and Supported hardware [▶ 10]).

In Version 2.0 and above of the library the function block replaces the hardware-dependent KL6Control [▶ 24], KL6Control5B [▶ 24] and PcComControl [▶ 23] function blocks.

Interface

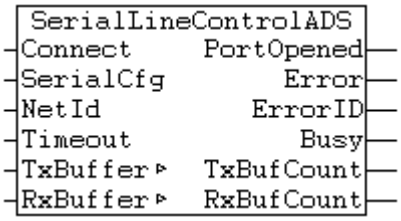
```
VAR_INPUT
    Mode          : ComSerialLineMode_t;
    pComIn        : POINTER TO BYTE
    pComOut       : POINTER TO BYTE
    SizeComIn     : UINT;
END_VAR
VAR_OUTPUT
    Error         : BOOL;
    ErrorID       : ComError_t;
END_VAR
VAR_IN_OUT
    TxBuffer      : ComBuffer;
    RxBuffer      : ComBuffer;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

Mode	The Mode input specifies unambiguously which serial hardware is being used.
pComIn	Universal pointer to the input variable of the process data for the serial hardware (data types KL6inData [▶ 36], KL6inData5b [▶ 36], KL6inData22b [▶ 37], EL6inData22b [▶ 37], PcComInData [▶ 37]). The pointer is assigned with the *ADR*() function.
pComOut	Universal pointer to the output variable of the process data for the serial hardware (data types KL6outData [▶ 36], KL6outData5b [▶ 36], KL6outData22b [▶ 37], EL6outData22b [▶ 37], PcComOutData [▶ 37]). The pointer is assigned with the *ADR*() function.
SizeComIn	Size of the input process image of the serial hardware being used. The size is determined and assigned with the *SIZEOF*() function.
Error	The *Error* output becomes TRUE as soon as an error occurs.
ErrorID	The *ErrorID* output provides an error code when an error occurs.
TxBuffer	Buffer with transmit data for the serial hardware being used. The transmit buffer is filled by functions such as SendByte [▶ 14], SendData [▶ 19] or SendString [▶ 16].
RxBuffer	Buffer into which received data is placed. The receive buffer is read by functions such as ReceiveByte [▶ 14], ReceiveData [▶ 17] or ReceiveString [▶ 15].

Also see about this

- Enumarated Types for the Communication Library [▶ 38]
- Error codes: ComError_t [▶ 40]
- ComBuffer [▶ 34]

5.8.1.2 SerialLineControlADS



The *SerialLineControlADS* function block looks after the communication between a virtual serial interface and the PLC. The function block is called cyclically, and places received data into the *RxBuffer*. Data made available in the *TxBuffer* transmit buffer is sent to the interface at the same time.

Because the function operates independently of the application, it is referred to as background communication and can, like the function block *SerialLineControl*, also operate in a fast task (see Communication concept [▶ 12] and Supported hardware [▶ 10]).

As soon as the function block is called cyclically and the input *Connect* is set, the parameterized serial COM port is opened automatically.
Thereby this COM port is blocked for other applications. Do you want to unblock the COM port to get access with another application, you can reset the input *Connect*. Thereby the current port will be closed. Is the COM port number or another parameter in the input structure SerialCfg [▶ 34] changed, the previous port will get closed automatically and after it the new port will get opened.

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
```

Input variables

Connect	To initialize a connection to a serial port, *Connect* TRUE must be applied to the function block. If *Connect* is FALSE, an opened port is closed again. If a change of this input variable is executed, it can take a maximum of 6 times the time specified at Timeout until the action has been completely executed. **Therefore, the application must pay attention to the PortOpened output and wait until it assumes the desired state.**
SerialCfg	This input structure defines which COM port is to be used and opened with which parameters. Details can be found in the description of ComSerialConfig [▶ 34].
NetId	To execute the query on the local device, it is not necessary to specify this input variable. Alternatively, an empty string can be specified. To direct the request to another TwinCAT target device, the corresponding AMS Net Id can be specified here.
Timeout	Specifies a maximum length of time for the execution of the function block. The default value is 5 seconds. (A value of at least 1000 ms should be specified)

Input/output variables

TxBuffer	Buffer with transmit data for the serial hardware being used. The transmit buffer is filled by functions such as SendByte [▶ 14], SendData [▶ 19] or SendString [▶ 16].
RxBuffer	Buffer into which received data is placed. The receive buffer is read by functions such as ReceiveByte [▶ 14], ReceiveData [▶ 17] or ReceiveString [▶ 15].

Output variables

PortOpened	This output indicates whether the selected serial port is opened and linked.
Error	The *Error* output becomes TRUE as soon as an error occurs.

ErrorID	The *ErrorID* output provides an error code when an error occurs. See chapter Error codes [▶ 40] for a list of possible values and troubleshooting information.
Busy	This output is TRUE as long as the internal ADS communication of the function block is active.
TxBufCount	The output *TxBufCount* indicates whether the internal PLC buffer still contains data bytes that have not yet been sent.
RxBufCount	The output *RxBufCount* can be used to determine whether there are still received data bytes in the internal PLC buffer that have not yet been transferred to the *RxBuffer*. The application must ensure that the received data is read out of the *RxBuffer* fast enough.

PLC libraries to include

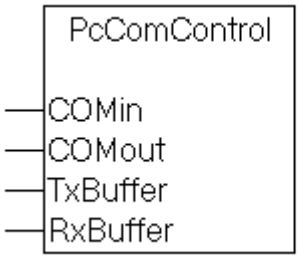
COMlibV2.lib [Version 2.003.008 or higher]

Also see about this

- ComSerialConfig [▶ 34]
- ComBuffer [▶ 34]

5.8.1.3 Library Version 1.7 Compatibility Blocks

5.8.1.3.1 PcComControl



Interface

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

Description

The **PcComControl** function block controls the data transfer between the data buffers **TxBuffer** and **RxBuffer** internal to the PLC and the hardware. The data structures PcComInData [▶ 37] and PcComOutData [▶ 37] are declared globally, and are linked in the TwinCAT System Manager with the PC interface.

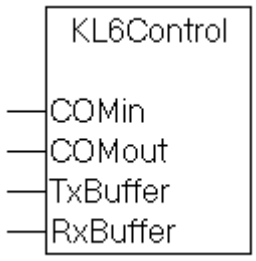
PcComControl is always called, regardless of whether there is any intention to send or receive data (Background Communication). This block should be called in a task that is fast enough to achieve an effective baud rate close to the physically determined baud rate.

The function blocks for background communication are preferably handled in a fast task (compare Supported Hardware [▶ 10]).
As of version 2 of the communication library, this function block is replaced by the function block SerialLineControl [▶ 20].

Also see about this

- Data structures for the COM serial PC interfaces [▶ 37]
- ComBuffer [▶ 34]

5.8.1.3.2 KL6Control



Interface

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

Description

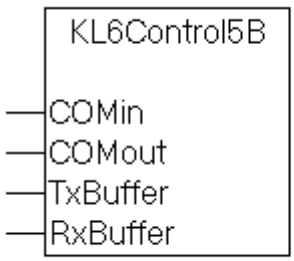
KL6Control controls the data transfer between the PLC's internal data buffers **TxBuffer** and **RxBuffer**, and the hardware, in a manner similar to the **PcComControl** block.

The function blocks for background communication are preferably handled in a fast task (compare Supported Hardware [▶ 10]).
As of version 2 of the communication library, this function block is replaced by the function block SerialLineControl [▶ 20].

Also see about this

- Data structures for the KL6xxx serial bus terminal in 3-byte mode [▶ 36]
- ComBuffer [▶ 34]

5.8.1.3.3 KL6Control5B



Interface

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

Description

KL6Control5B controls the data transfer between the data buffers **TxBuffer** and **RxBuffer**, internal to the PLC and the hardware in a manner similar to the **KL6Control** block, but in this case is for the serial bus terminal in 5-byte mode.

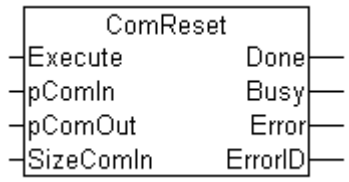
The function blocks for background communication are preferably handled in a fast task (compare Supported Hardware [▶ 10]).
As of version 2 of the communication library, this function block is replaced by the function block SerialLineControl [▶ 20].

Also see about this

- Data structures for the KL6xxx serial bus terminal in 5-byte mode [▶ 36]
- ComBuffer [▶ 34]

5.8.2 Configuration

5.8.2.1 ComReset



ComReset resets the connected serial hardware. This clears the internal hardware send and receive buffers. The function block supports a variety of serial hardware, such as the serial PC interface and the serial KL6xxx Bus Terminals.

The function block replaces the KL6Init [▶ 30] function block as of library version 2.0.

The function block does not clear the data buffer of type ComBuffer [▶ 34] internal to the PLC. For initialization purposes, this can be separately cleared with the ClearComBuffer [▶ 20] function block.

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 : a rising edge at the Execute input resets the connected serial hardware.

pComIn : universal pointer to the input variable of the process data for the serial hardware (data types KL6inData [▶ 36], KL6inData5b [▶ 36], PcComInData [▶ 37]). The pointer is assigned with the *ADR*() function.

pComOut : universal pointer to the output variable of the process data for the serial hardware (data types KL6outData [▶ 36], KL6outData5b [▶ 36], PcComOutData [▶ 37]). The pointer is assigned with the *ADR*() function.

SizeComIn : size of the input process image of the serial hardware being used. The size is determined and assigned with the *SIZEOF*() function.

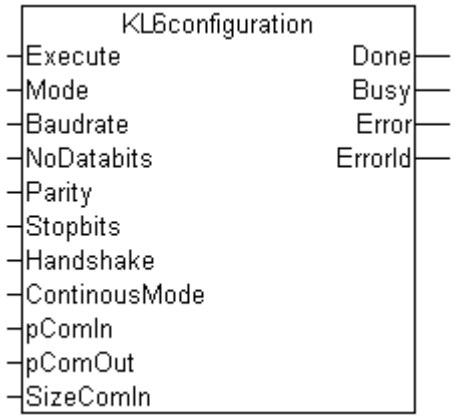
Done : the Done output becomes TRUE when the function has been carried out without error.

Busy : the Busy output becomes TRUE with a rising edge at Execute and remains TRUE as long as the function block executes its function.

Error : the *Error* output becomes TRUE as soon as an error occurs.

ErrorID : the *ErrorID* output gives an error code in case of an error.

5.8.2.2 KL6Configuration



The KL6configuration function block initialises and configures a KL6xxx Serial Bus Terminal.

In Version 2.0 and above of the library the function block replaces the hardware-dependent KL6Config [▶ 31] and KL6Config5B [▶ 32] function blocks.

Configuration of EtherCAT terminals

The function block uses register communication to configure KL6 terminals. Register communication is not available with EtherCAT terminals. EL terminal can be configured with function blocks (FB_EcCoeSdoWrite) from the TwincAT EtherCAT library.

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: A rising edge at the Execute input resets the connected serial hardware.

Mode: The Mode input specifies unambiguously which serial hardware is being used. (ComSerialLineMode_t [▶ 38]).

Baudrate: The baud rate, provided it is supported by the serial hardware.

NoDatabits: The number of user data bits in one data byte.

Parity: The type of the parity bit in a data byte.

Stopbits: The number of stop bits per data byte.

Handshake: The type of handshake used, provided it is supported by the serial hardware.

ContinousMode: Switches on continuous transmission, provided this is supported by the serial hardware.

If ContinousMode is TRUE, transmitted data is not sent out by the serial hardware until the hardware transmit buffer is full. This means that there are no time gaps in the transmission, provided the quantity of data is similar in size to the hardware transmit buffer. Continuous mode is only necessary in special cases in which the end device reacts to time gaps with a time-out.

pComIn: Universal pointer to the input variable of the process data for the serial hardware (data types KL6inData [▶ 36], KL6inData5b [▶ 36], KL6inData22b [▶ 37], PcComInData [▶ 37]). The pointer is assigned with the *ADR*() function.

pComOut: Universal pointer to the output variable of the process data for the serial hardware (data types KL6outData [▶ 36], KL6outData5b [▶ 36], KL6outData22b [▶ 37], PcComOutData [▶ 37]). The pointer is assigned with the *ADR*() function.

SizeComIn: Size of the input process image of the serial hardware being used. The size is determined and assigned with the *SIZEOF*() function.

Done: The Done output becomes TRUE when the function has been carried out without error.

Busy: The Busy output becomes TRUE in response to rising edge at Execute, and remains TRUE for as long as the block is performing its function.

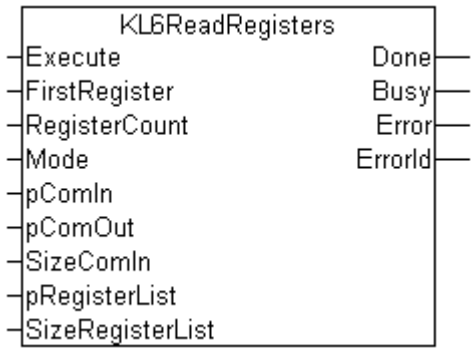
Error: The *Error* output becomes TRUE as soon as an error occurs.

ErrorID: The *ErrorID* output provides an error code when an error occurs.

Also see about this

- Error codes: ComError_t [▶ 40]

5.8.2.3 KL6ReadRegisters



The KL6ReadRegisters function block reads one or several registers of a KL6xxx serial Bus Terminal.

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: A rising edge at the Execute input resets the connected serial hardware.

FirstRegister : Specifies the first register to be read. From this register number (which may range between 1 and 64), *RegisterCount* data are read and stored in the register list of type ComRegisterData_t.

If no coherent register range is to be read, the value 16#FFFF can be stored in *FirstRegister*. In this case the user has to initialise the register numbers to be read in the register list before the block is triggered. In this case *RegisterCount* is not used.

RegisterCount : Specifies the number of registers to be read. The block reads a coherent register range from FirstRegister and stores the data in the register list.

Mode : The Mode input specifies unambiguously which serial hardware is being used. (ComSerialLineMode_t [▶ 38]).

pComIn: Universal pointer to the input variable of the process data for the serial hardware (data types KL6inData [▶ 36], KL6inData5b [▶ 36], KL6inData22b [▶ 37], PcComInData [▶ 37]). The pointer is assigned with the *ADR*() function.

pComOut: Universal pointer to the output variable of the process data for the serial hardware (data types KL6outData [▶ 36], KL6outData5b [▶ 36], KL6outData22b [▶ 37], PcComOutData [▶ 37]). The pointer is assigned with the *ADR*() function.

SizeComIn: Size of the input process image of the serial hardware being used. The size is determined and assigned with the *SIZEOF*() function.

pRegisterList : Start address of a register list of type ComRegisterData_t [▶ 34]. The start address can be determined with ADR(register list).

SizeRegisterList : Size of the register list in bytes. The size can be determined with SIZE(register list). The list may have between 1 and 64 entries.

Done: The Done output becomes TRUE when the function has been carried out without error.

Busy: The Busy output becomes TRUE in response to rising edge at Execute and remains TRUE for as long as the block is performing its function.

Error: The *Error* output becomes TRUE as soon as an error occurs.

ErrorID: The *ErrorID* output provides an error code when an error occurs.

Also see about this

- Error codes: ComError_t [▶ 40]

5.8.2.4 KL6WriteRegisters

The function block KL6WriteRegisters writes data to one or several registers of a serial KL6xxx Bus Terminal.

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: A rising edge at the Execute input resets the connected serial hardware.
The *register list* must be initialised before the block is triggered. This means the register numbers and register content must be entered in the list.

Mode: The Mode input specifies unambiguously which serial hardware is being used. (ComSerialLineMode_t [▶ 38]).

pComIn: Universal pointer to the input variable of the process data for the serial hardware (data types KL6inData [▶ 36], KL6inData5b [▶ 36], KL6inData22b [▶ 37], PcComInData [▶ 37]). The pointer is assigned with the *ADR*() function.

pComOut: Universal pointer to the output variable of the process data for the serial hardware (data types KL6outData [▶ 36], KL6outData5b [▶ 36], KL6outData22b [▶ 37], PcComOutData [▶ 37]). The pointer is assigned with the *ADR*() function.

SizeComIn: Size of the input process image of the serial hardware being used. The size is determined and assigned with the *SIZEOF*() function.

pRegisterList : Start address of a register list of type ComRegisterData_t [▶ 34]. The start address can be determined with ADR(register list). The *register list* must be initialised before the block is triggered. This means the register numbers and register content must be entered in the list.

SizeRegisterList : Size of the register list in bytes. The size can be determined via SIZE(register list). The list may have between 1 and 64 entries.

Done: The Done output becomes TRUE when the function has been carried out without error.

Busy: The Busy output becomes TRUE in response to rising edge at Execute, and remains TRUE for as long as the block is performing its function.

Error: The *Error* output becomes TRUE as soon as an error occurs.

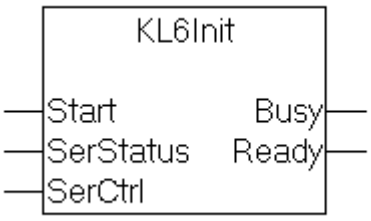
ErrorID: The *ErrorID* output provides an error code when an error occurs.

Also see about this

- Error codes: ComError_t [▶ 40]

5.8.2.5 Library Version 1.7 Compatibility Blocks

5.8.2.5.1 KL6Init



Interface

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

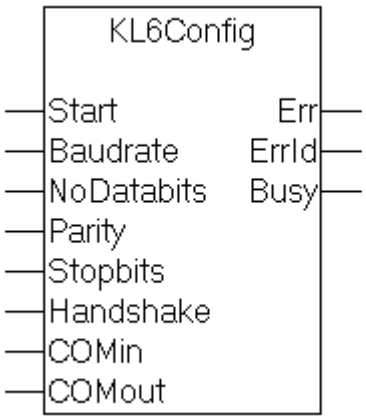
Description

KL6Init initialises the KL6xxx serial bus terminal without altering its configuration. The terminal's internal buffers are cleared.

The **SerStatus** and **SerCtrl** input data correspond to the data linked with the terminal in the TwinCAT System Manager. They are defined in the COMlib KL6inData [▶ 36] and KL6outData [▶ 36] data structures respectively.

The terminal has been successfully initialised after the **Start** signal when Busy=FALSE and Ready=TRUE.

5.8.2.5.2 KL6Config



Interface

```
VAR_INPUT
    Start : BOOL; (* Edge triggered *)
    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
```

Description

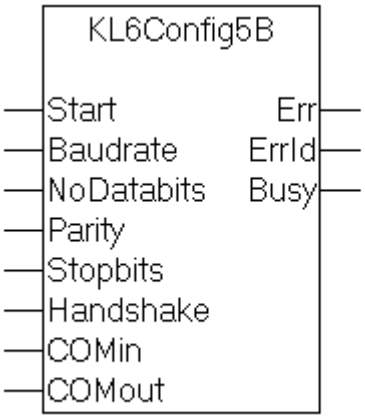
KL6Config configures the interface parameters of the KL6xxx serial bus terminal.

> While the **KL6Config** function block is active (**busy**), the KL6Control function block for background communication must **not** be called! Initialization and normal communication with the serial bus terminal use the same registers.
> From version 2 of the communication library, this function block is replaced by the function block KL6configuration [▶ 26].

Also see about this

- Data structures for the KL6xxx serial bus terminal in 3-byte mode [▶ 36]

5.8.2.5.3 KL6Config5B



Interface

```
VAR_INPUT
    Start : BOOL; (* Edge triggered *)
    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
```

Description

KL6Config5B works in a manner similar to that of the KL6Config block, but operates the serial bus terminal in 5-byte mode.

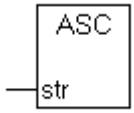
> While the **KL6Config5B** function block is active (**busy**), the KL6Control5B function block for background communication must not be called! Initialization and normal communication with the serial bus terminal use the same registers.
> From version 2 of the communication library, this function block is replaced by the function block KL6configuration [▶ 26].

Also see about this

- Data structures for the KL6xxx serial bus terminal in 5-byte mode [▶ 36]

6 Help Functions

6.1 ASC



Interface

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

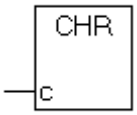
Description

The function Asc returns the ASCII code of the first character of the input sting as one byte.

Background

Transmission data is often available as a string of characters. If it is to be sent, the individual characters are required as bytes.

6.2 CHR



Interface

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

Description

The function Chr returns the character corresponding to the ASCII code in the input variable c to a string.

Background

Received characters arrive as bytes at the PLC system, and must often be further processed in the form of strings.

7 Data Types

7.1 Structures

7.1.1 ComBuffer

The **ComBuffer** data structure is a data buffer that decouples the hardware-dependent communication blocks from the hardware-independent blocks (see Communication Principle [▶ 12]). At the same time this decouples a fast communication task from the standard task. Data buffers of type ComBuffer are never directly written or read by the user, but are merely used as intermediate storage for the communication blocks.

```
TYPE ComBuffer
STRUCT
    Buffer: ARRAY[0..300] OF BYTE;
    RdIdx: INT;
    WrIdx: INT;
    Count: INT; (* Number of characters in the ring buffer *)
    FreeByte:INT; (* Number of free spaces in the ring buffer *)
    Error: INT; (* Interface error code *)
    blocked : BOOL;
END_STRUCT
END_TYPE
```

7.1.2 Data structures RegisterList

Data type *ComRegisterList_t* has been defined for reading and writing of registers of a serial Bus Terminal. Each entry of the *register list* contains the register number and the register content.

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

This input structure defines which COM port is to be used and opened with which parameters.

If a parameter is changed during cyclic calls of SerialLineControlADS [▶ 21], the existing COM port connection is closed automatically, and the serial COM port is opened with the new parameter. It is not necessary to close the port explicitly by resetting the input *Connect*.

```
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	Any COM port (COM1 .. COM255) can be selected. The Windows hardware device manager indicates which number a driver (e.g. a USB-to-virtual-COM-port driver) has assigned.
Baud rate	All standard baud rates can be set, from 150 baud up to 115200 baud. The default is 9600 baud.
Parity	The parity check for the serial data transmission is set here. Possible values are summarized in the enumeration ComParity_t [▶ 38].
DataBits	The number of data bits for the serial data transmission is set here. Possible values are 4, 5, 6, 7 and 8. Although complete data bytes are still transferred to the PLC buffer if the value is smaller than 8 bits, not all 8 bits of a byte are transferred via the serial interface. The most significant bits are truncated before transmission, which reduces the amount of bits per byte of data to be transmitted. This setting was mainly useful in earlier times if you knew that the most significant bits in each byte would never be used and you wanted to increase the data transfer rate. Nowadays it is rarely used, so that it is recommended to leave the number of data bits at 8 [default].
StopBits	The number of stop bits for the serial data transmission is set here. Possible values are summarized in the enumeration ComStopBits_t [▶ 38].
RTS	The 'Request to Send' signal (= RFR 'Ready for Receiving') for the serial data transmission is set here. Possible values are summarized in the enumeration ComRTSCtrl_t [▶ 38].
DTR	The 'Data Terminal Ready' signal for the serial data transmission is set here. Possible values are summarized in the enumeration ComDTRCtrl_t [▶ 38].
CTS	The 'Clear to Send' signal for the serial data transmission is set here. If the value is TRUE, no data are sent if the CTS input signal for the data transmission is not set.
DSR	The 'Dataset Ready' signal for the serial data transmission is set here. If the value is TRUE, no data are sent if the DSR input signal for the data transmission is not set. If *DSR* is TRUE, received data bytes are ignored, if the DSR input signal for the data transmission is not set.
TraceLevel	This input can be used to configure messages (debug traces) that are issued by the TcAdsSerialCommServer. This does not apply to the error messages at the output of the PLC function block *SerialLineControlADS*. It is therefore an additional diagnostic option. The following trace levels are possible (default = 0): • 0 None (default) • 1 Error • 2 Warning • 3 Info • 4 Verbose • 5 Noise All messages with a level less or equal the specified level are output. If level 'None' is selected, no messages are output. Debug traces are used as output. Under Win32 error/warning/info messages are also output as log events (application log). With level 'Noise' even data received at the serial port are output in the messages. Some system resources are needed for this and the setting should therefore only be

	selected for temporary tests. **When using the DebugView tool (from SysInternals) the CaptureGlobalWin32 setting must be active to receive the messages.**

Not all parameter settings for serial data transmission may be available in all cases. Some settings or combinations are not supported by Windows or by Com-Port drivers.
(Example: often the possibility of 1.5 stop bits or 4 data bits or even the combination of 5 data bits & 2 stop bits is not supported. Or the baud rate may be limited to a maximum of 115200 baud.)

Further information on the parameters for serial data transmission can be found in the Microsoft MSDN description of the DCB structure.

PLC libraries to include

COMlibV2.lib [Version 2.003.008 or higher]

7.1.4 Data structures for the KL6xxx serial bus terminal in 3-byte mode

For data exchange by way of the I/O bus, every serial bus terminal needs variables of type KL6inData and KL6outData. These variables are placed at a fixed address in the memory map, and are linked to the hardware with the TwinCAT System Manager.

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 Data structures for the KL6xxx serial bus terminal in 5-byte mode

For data exchange by way of the I/O bus, every serial bus terminal need variables of type KL6inData5B and KL6outData5B. These variables are placed at a fixed address in the memory map and are linked to the hardware with the TwinCAT System Manager.

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 Data structures for the KL6xxx serial bus terminals in 22-byte mode

For data exchange by way of the I/O bus, every serial bus terminal needs variables of type KL6inData22B and KL6outData22B. These variables are placed at a fixed address in the memory map, and are linked to the hardware with the TwinCAT System Manager.

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 Data structures for the EL60xx serial EtherCAT terminal in 22-byte mode

For data exchange by way of the I/O bus, every serial EtherCAT terminal needs variables of type EL6inData22B and EL6outData22B. These variables are placed at a fixed address in the memory map and are linked to the hardware with the TwinCAT System Manager.

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 Data structures for the COM serial PC interfaces

Every serial PC interface needs a variable of type PcComInData and one of type PcComOutData for data exchange. These variables are placed at a fixed address in the memory map and are linked to the hardware with the TwinCAT System Manager.

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 Enumarated Types for the Communication Library

ComSerialLineMode_t

The enumeration type ComSerialLineMode_t defines the type of used serial hardware for different hardware dependent function blocks of the serial communication library.

```
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 Error codes

8.1 Error codes: 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
```

Error codes of type *ComError_t* are not used by the function block *SerialLineControlADS*.

8.2 Error codes: AdsSerialComm

Overview of the error codes of the function block SerialLineControlADS

Offset + error code	Range	Description
0x00000000 + TwinCAT system error	0x00000000-0x00007800	TwinCAT system error (including ADS error codes)
0x00000000 + TcAdsSerialCommServer error	0x00009000-0x000091FF	Error in the TwinCAT ADS Serial Comm Server
0x3D090000 + Win32 system error code	0x3D090000-0x3D09FFFF	Win32 system error

Error codes of the type *ComError_t* are not used by this function block.

TcAdsSerialCommServer error

Code (hex)	Code (dec)	Description	Symbolic name
0x00009001	36865	The specified COM port is invalid. Valid value range: 1 .. 255	COMERRORADS_INVALID_COMPORT
0x00009002	36866	The command for the TcAdsSerialCommServer is invalid.	COMERRORADS_INVALID_CMD
0x00009003	36867	internal error	COMERRORADS_INVALID_DATAPOINTER
0x00009011	36881	The parameter structure passed is unknown to the TcAdsSerialCommServer.	COMERRORADS_INVALID_CFGSTLEN
0x00009012	36882	The parameter structure passed is unknown to the TcAdsSerialCommServer.	COMERRORADS_INVALID_CFGSTVER
0x00009013	36883	The trace level (variable *TraceLevel* in the input structure SerialCfg [▶ 34]) for the output of messages is invalid.	COMERRORADS_INVALID_TL
0x00009021	36897	The specified baud rate is not supported.	COMERRORADS_INVALID_BAUDRATE
0x00009022	36898	The specified parity is invalid.	COMERRORADS_INVALID_PARITY
0x00009023	36899	The specified number of data bits is invalid.	COMERRORADS_INVALID_BYTESIZE
0x00009024	36900	The specified number of stop bits is invalid.	COMERRORADS_INVALID_STOPBIT
0x00009025	36901	Dtr control is invalid.	COMERRORADS_INVALID_DTR_CTRL
0x00009026	36902	Rts control is invalid.	COMERRORADS_INVALID_RTS_CTRL
0x00009027	36903	Cts is invalid.	COMERRORADS_INVALID_CTS_OUTCTRL
0x00009028	36904	Dsr is invalid.	COMERRORADS_INVALID_DSR_OUTCTRL
0x00009029	36905	Dsr is invalid.	COMERRORADS_INVALID_DSR_SENS
0x00009031	36913	internal error	COMERRORADS_NOT_INIT
0x00009032	36914	The receive buffer in the TcAdsSerialCommServer has overflowed. Incoming data is lost. **Received data must be queried immediately.** It must be ensured that no data accumulates in the PLC function block SerialLineControlADS [▶ 21]. This can be monitored with the output *RxBufCount*. With a stable communication connection, this value does not rise above 1000.	COMERRORADS_RD_BUFFER_OVERRUN

Code (hex)	Code (dec)	Description	Symbolic name
0x00009033	36915	The COM port is already open. *SerialLineControlADS* will automatically try to close the port and reopen it with the specified parameters. If no errors are present at the output in the following cycles and the output *PortOpened* = TRUE, an opening of the COM port has been successfully completed.	COMERRORADS_PORT_CONNECTED
0x00009034	36916	Interaction with the COM port is not possible because the COM port has not yet been opened by the TcAdsSerialCommServer. *SerialLineControlADS* will automatically try to reopen the port with the specified parameters. If no errors are present at the output in the following cycles and the output *PortOpened* = TRUE, an opening of the COM port has been successfully completed.	COMERRORADS_PORT_NOT_CONNECTED
0x00009035	36917	The COM port could not be closed correctly.	COMERRORADS_RD_THREAD_TIMEOUT
0x00009036	36918	The COM port could not be closed correctly.	COMERRORADS_WR_THREAD_TIMEOUT
0x00009037	36919	During an existing communication connection, this error may occur if the USB device is disconnected. Before disconnecting the USB device the input *bConnect* must be set to FALSE and thus the COM port is closed. Also a read error can be the trigger. Details can be output using the variable *TraceLevel* in the input structure SerialCfg [▶ 34]. **SerialLineControlADS has closed the port and will automatically try to reopen it with the specified parameters.** If no errors are present at the output in the	COMERRORADS_RD_FAILURE

Code (hex)	Code (dec)	Description	Symbolic name
		following cycles and the output *PortOpened* = TRUE, an opening of the COM port has been successfully completed.	
0x00009038	36920	A write error can be the trigger for this error code. Details can be output with the help of the variable *TraceLevel* in the input structure SerialCfg [▶ 34]. It is possible that data has not been transferred. There is no automatic repetition of the write attempt.	COMERRORADS_WR_FAILURE
0x000090E0 - 0x000090FF	37088 - 37119	Internal errors	
0x00009101	37121	The version of the TcAdsSerialCommServer is incompatible. An official product installation fixes the error.	COMERRORADS_SERVER_INCOMPATIBLE

Additional information regarding the main Win32 system errors

Code (hex)	Code (dec)	Description	additional information	Symbolic name
0x3D090002	1024000002	The system cannot find the specified file.	This error can occur if the specified COM port is not available. Make sure the value for the COM port and the other parameters in ComSerialConfig [▶ 34] are specified correctly.	ERROR_FILE_NOT_FOUND
0x3D090005	1024000005	Access is denied.	Check whether the respective serial COM port was already accessed/ opened by another program. In this case you have to enable the port from the other program, in order to enable communication.	ERROR_ACCESS_DENIED
0x3D090057	1024000087	The parameter is incorrect.	This error occurs if an input parameter for serial data	ERROR_INVALID_PARAMETER

Code (hex)	Code (dec)	Description	additional information	Symbolic name
			transmission is invalid (see ComSerialConfig [▶ 34]). Not all parameter settings for serial data transmission may be available in all cases. Some settings or combinations are not supported by Windows or the COM port drivers. In this case you should check whether the communication works with a different parameter setting.	

For more detailed error analysis additional debug outputs can be configured in the TcAdsSerialCommServer. Use the variable *TraceLevel* in the input structure SerialCfg [▶ 34] for this purpose.

9 Linking into a PLC Program

9.1 Installation

COMlibV2

In order to install, the library COMlibV2.lib must be copied into the TwinCAT directory TwinCAT\PLC\LIB.

COMlib

In order to install, the library COMlib.LIB and the auxiliary libraries ChrAsc.LIB and ChrAsc.OBJ, which are also supplied, must be copied into the TwinCAT directory TwinCAT\PLC\LIB.

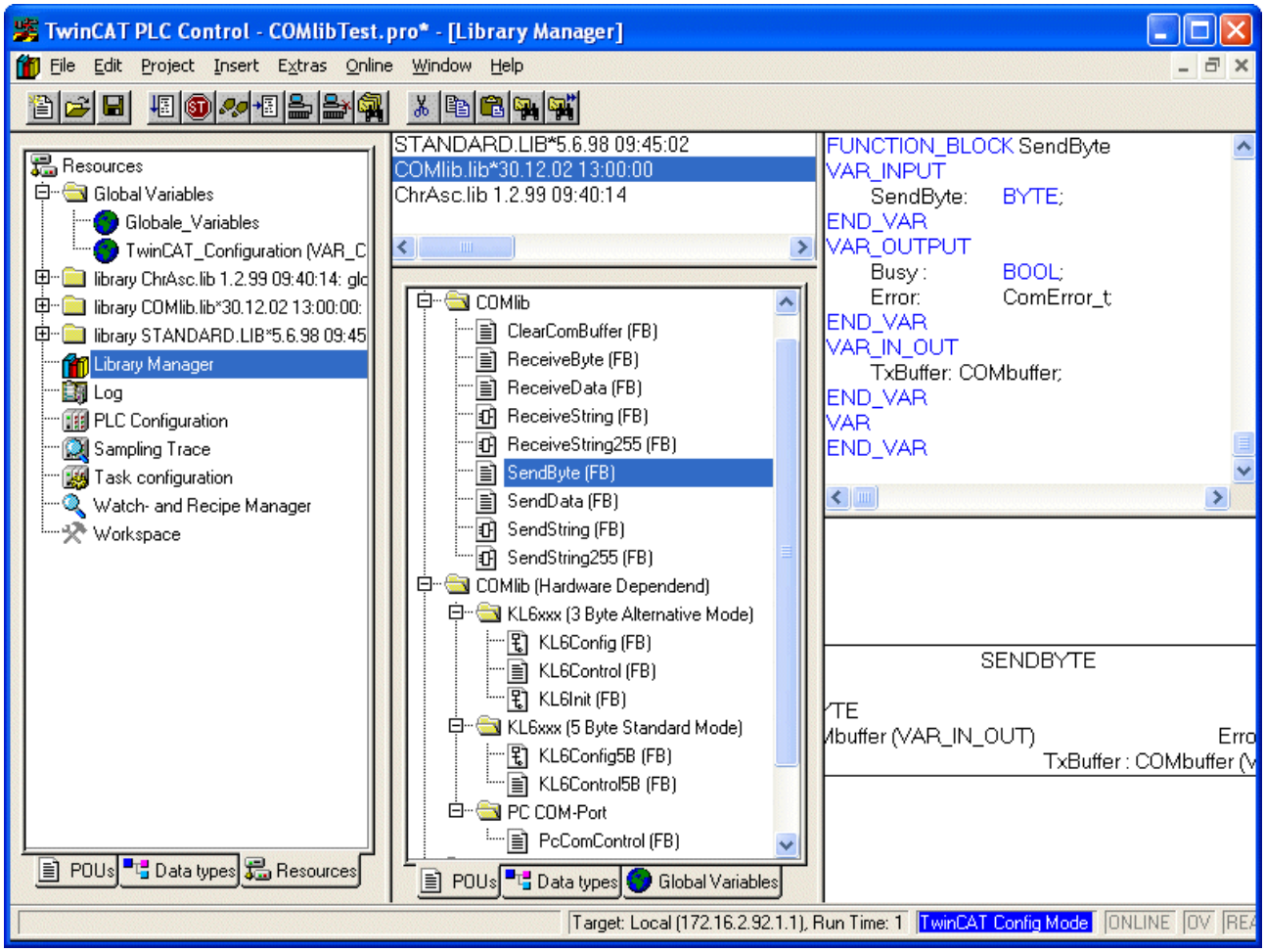
The associated test program should be copied to any project directory of your choice, e.g. to TwinCAT\PLC.

- https://infosys.beckhoff.com/content/1033/tcplclibserialcom/Resources/11388462091/.zip

Link libraries

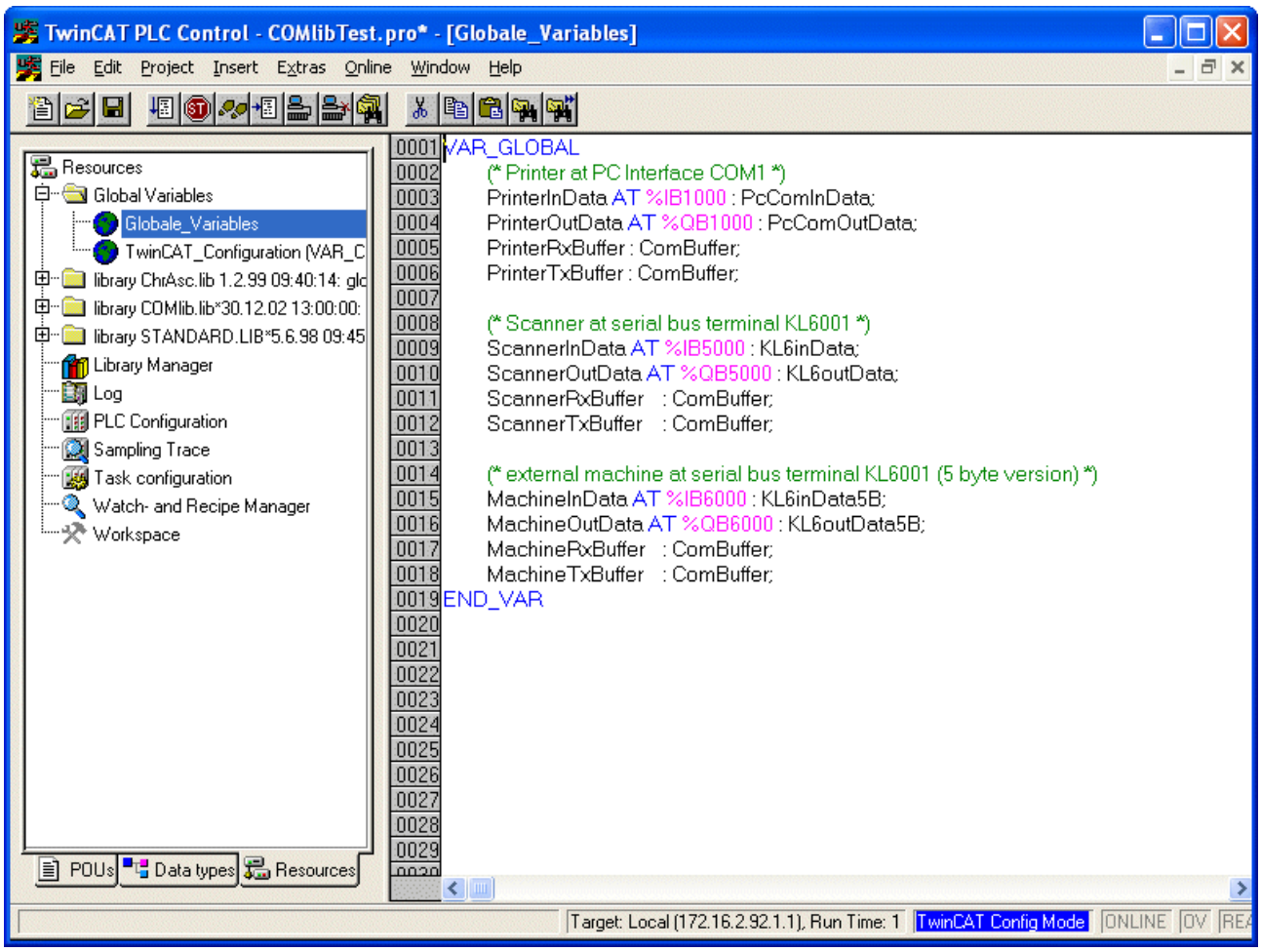
Create a new PLC project with TwinCAT PLC Control in order to perform the library linking.

Go to Library Management and add the libraries ChrAsc.LIB and ComLib.LIB.



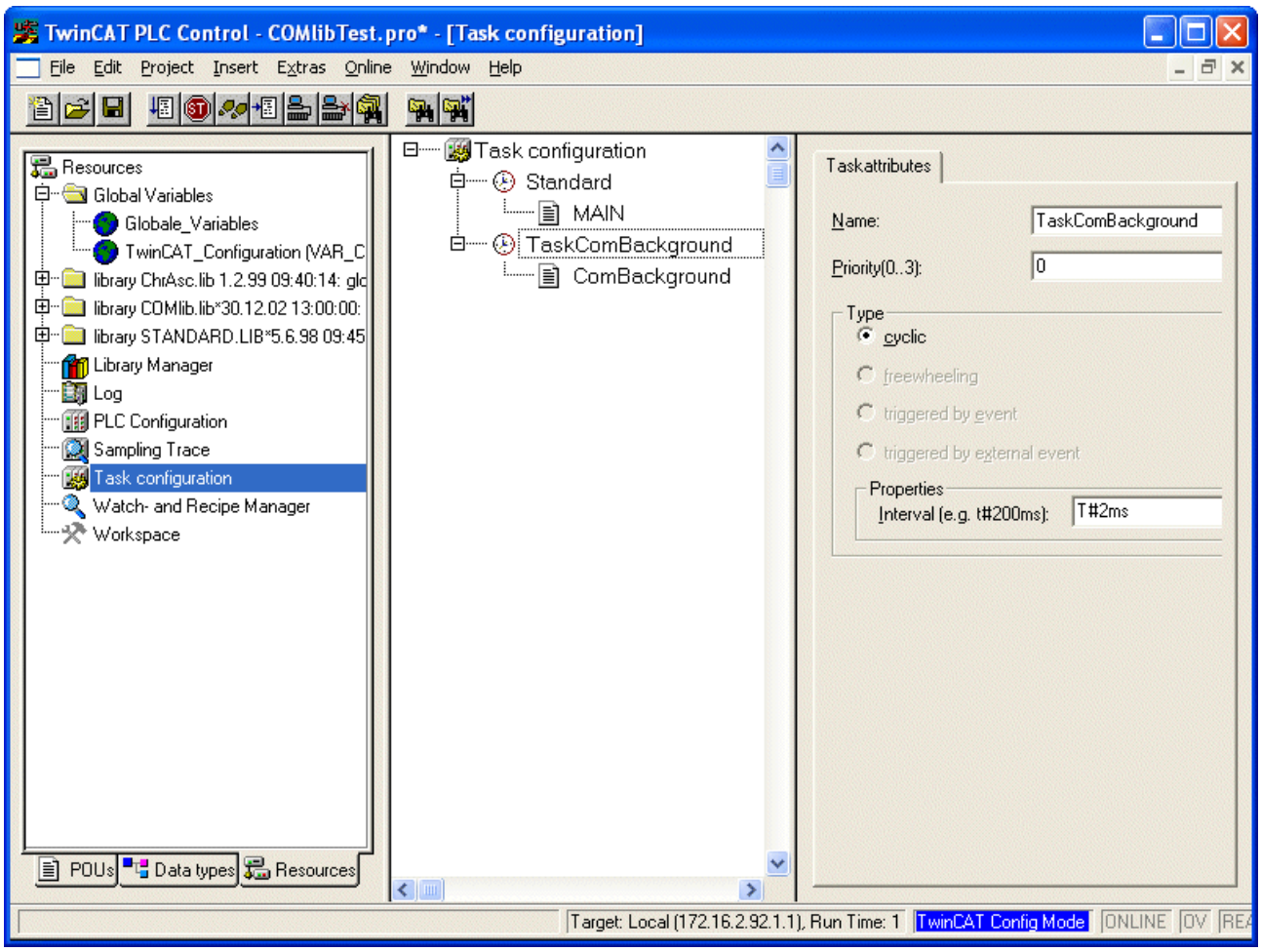
9.2 Global Variables

Four global data structures are needed to access a serial interface. Two provide the connection to the hardware in the send and receive directions. Two data buffers are also necessary for intermediate storage.



9.3 Task Configuration

The speed of the serial interfaces must be considered for the task configuration (see the section on Supported Hardware [▶ 10] and the Communication Principle [▶ 12]). For example, in order that, with 9600 bps at the serial bus terminal, the data can all actually be processed at this speed, the associated communication block must be active at least once per millisecond. The task that operates the function block must have a correspondingly fast setting. The simplest case is when the entire PLC program runs in this fast task. If the task is set slower, then as long as the interface operates with a hardware handshake, the communication will function at reduced speed. Without handshake, data to be received can be lost.



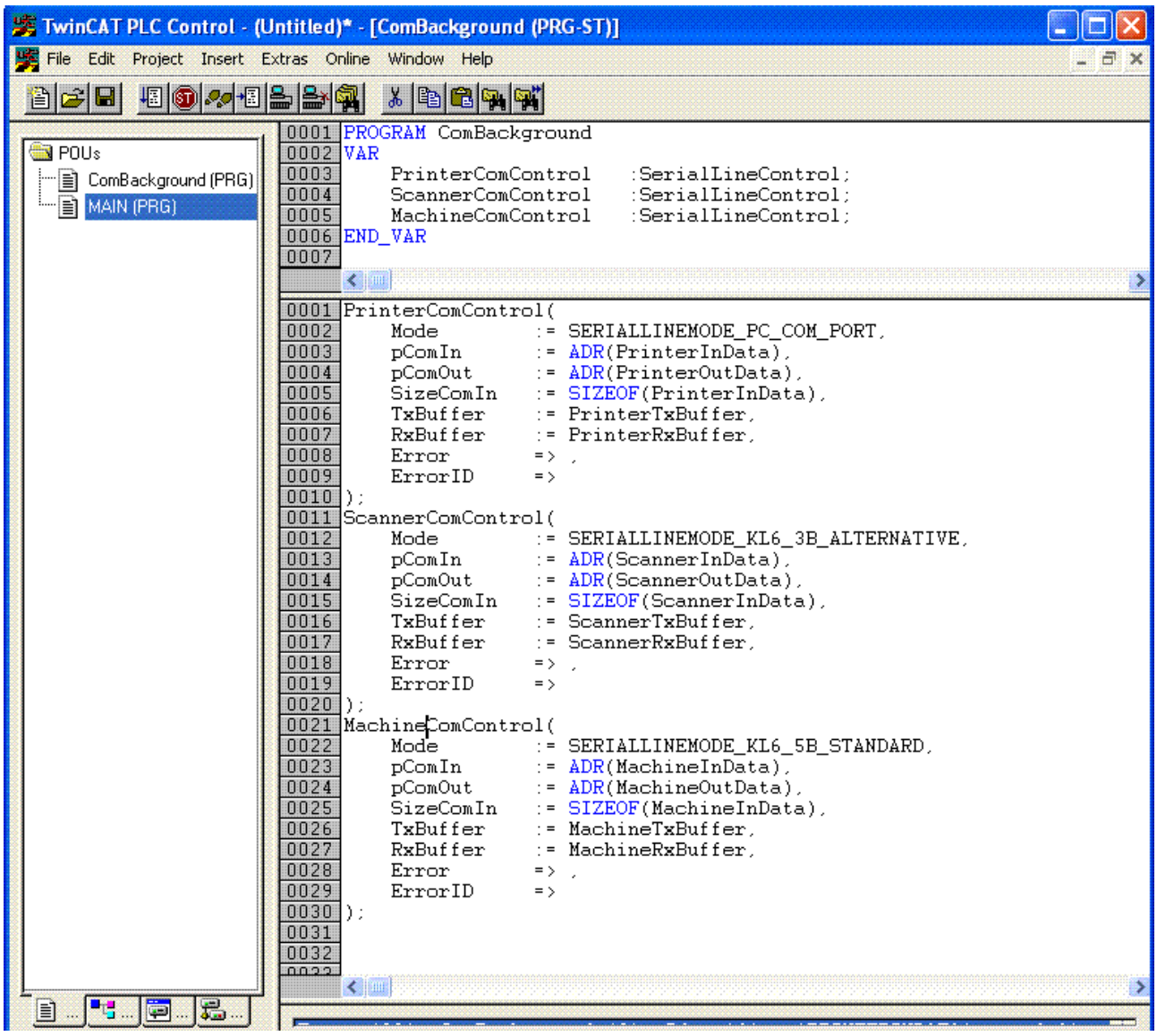
When using KL6xxx Bus Terminals on BKxxxx Bus Couplers, please note that the K-bus update time must be below the cycle time of the task. The K-Bus update time can be read in the TwinCAT System Manager after clicking on the bus coupler below the I/O configuration. A reserve of 10 % to 20 % should be taken into account. If there are many bus terminals on one bus coupler, the **cycle time of the task** may have to be set to **at least 2 ms**.

9.4 Background Communication

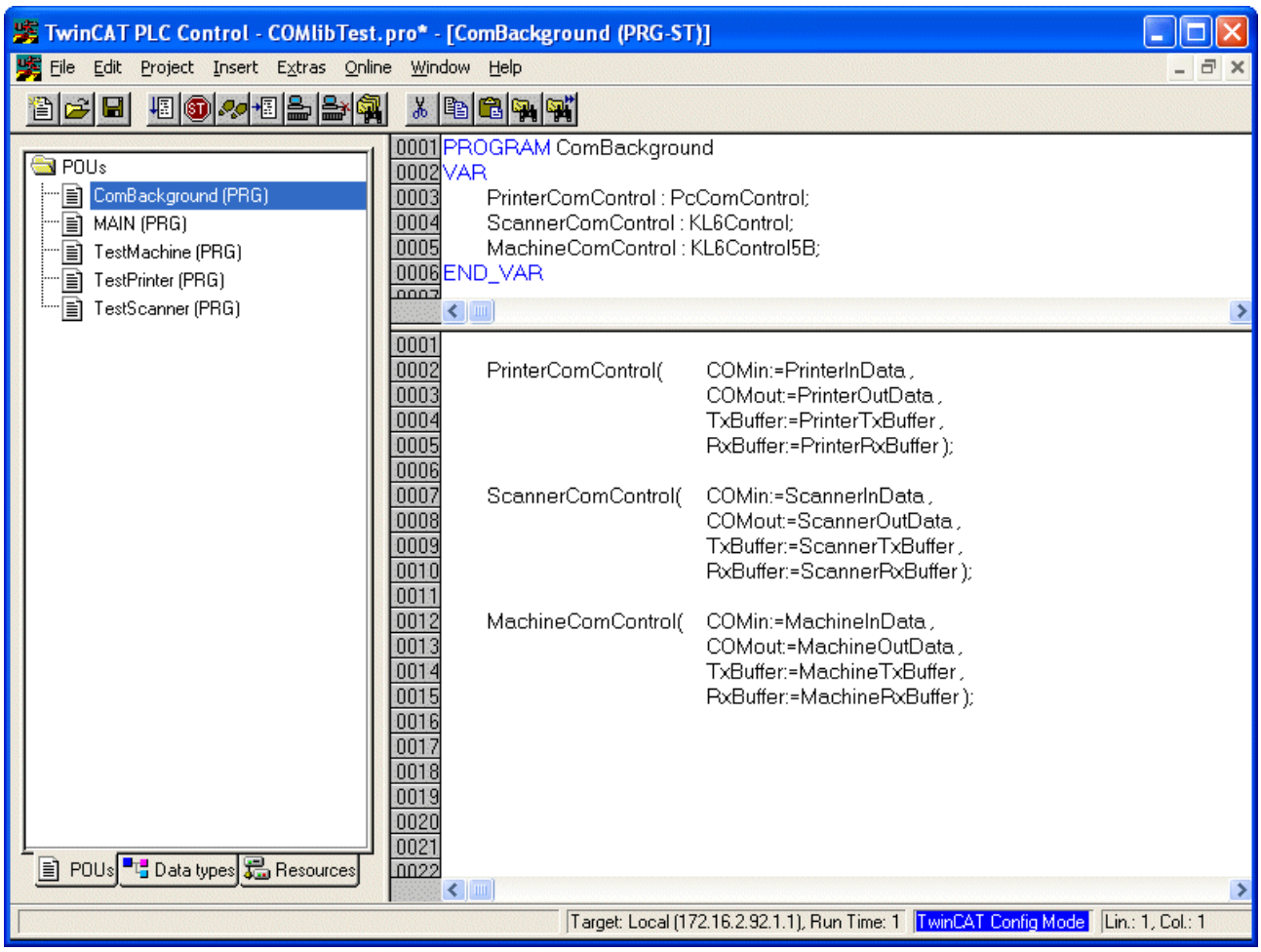
Communication between the serial hardware and the data buffer, whose type is ComBuffer [▶ 34], is handled in a separate fast task.

See also the Communication Principle [▶ 12].

with 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

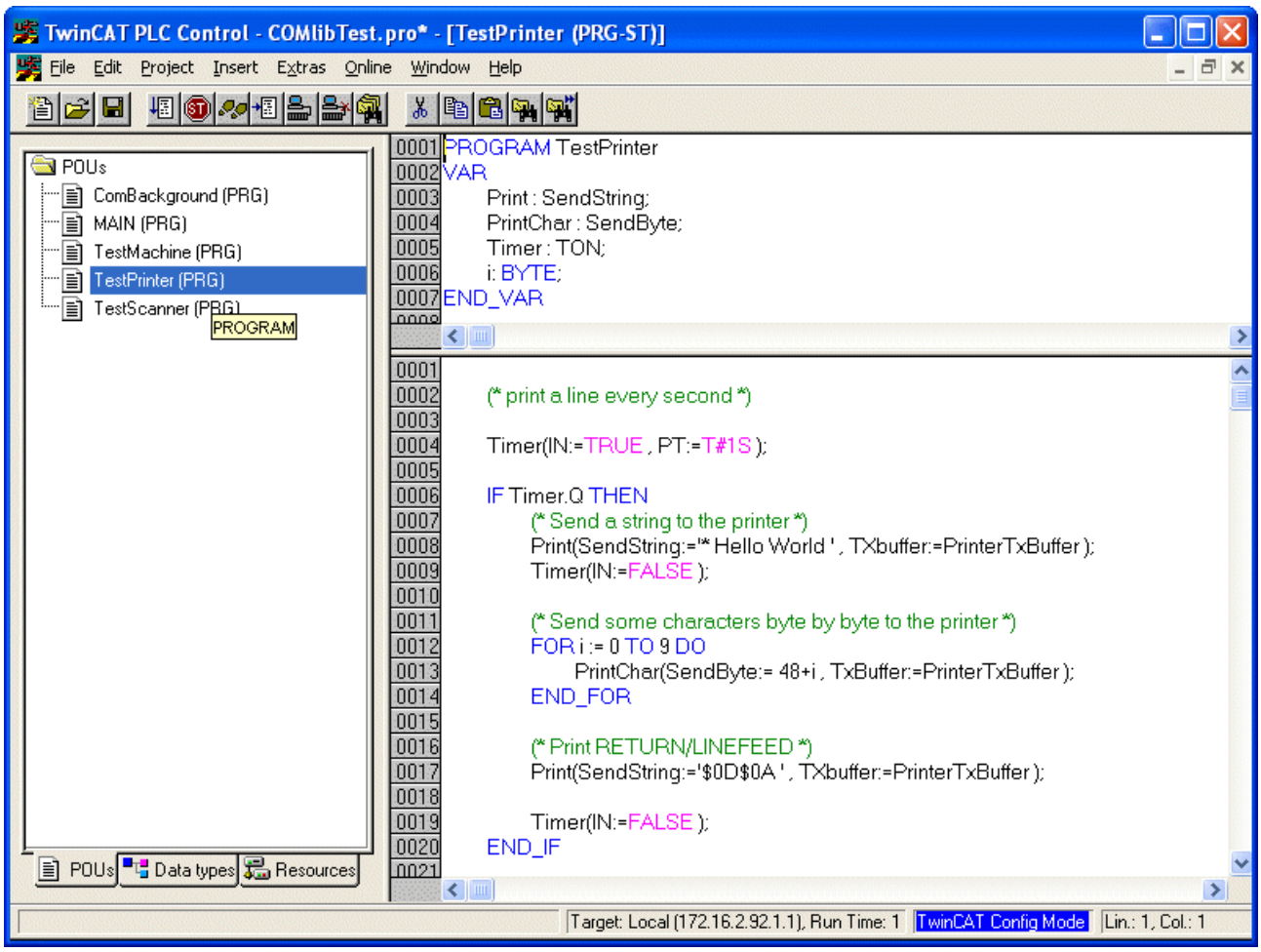
with COMlib:



9.5 Sending and Receiving

Sending data

Data is sent from the standard task. The example program included sends data by way of the serial PC interface to a printer.

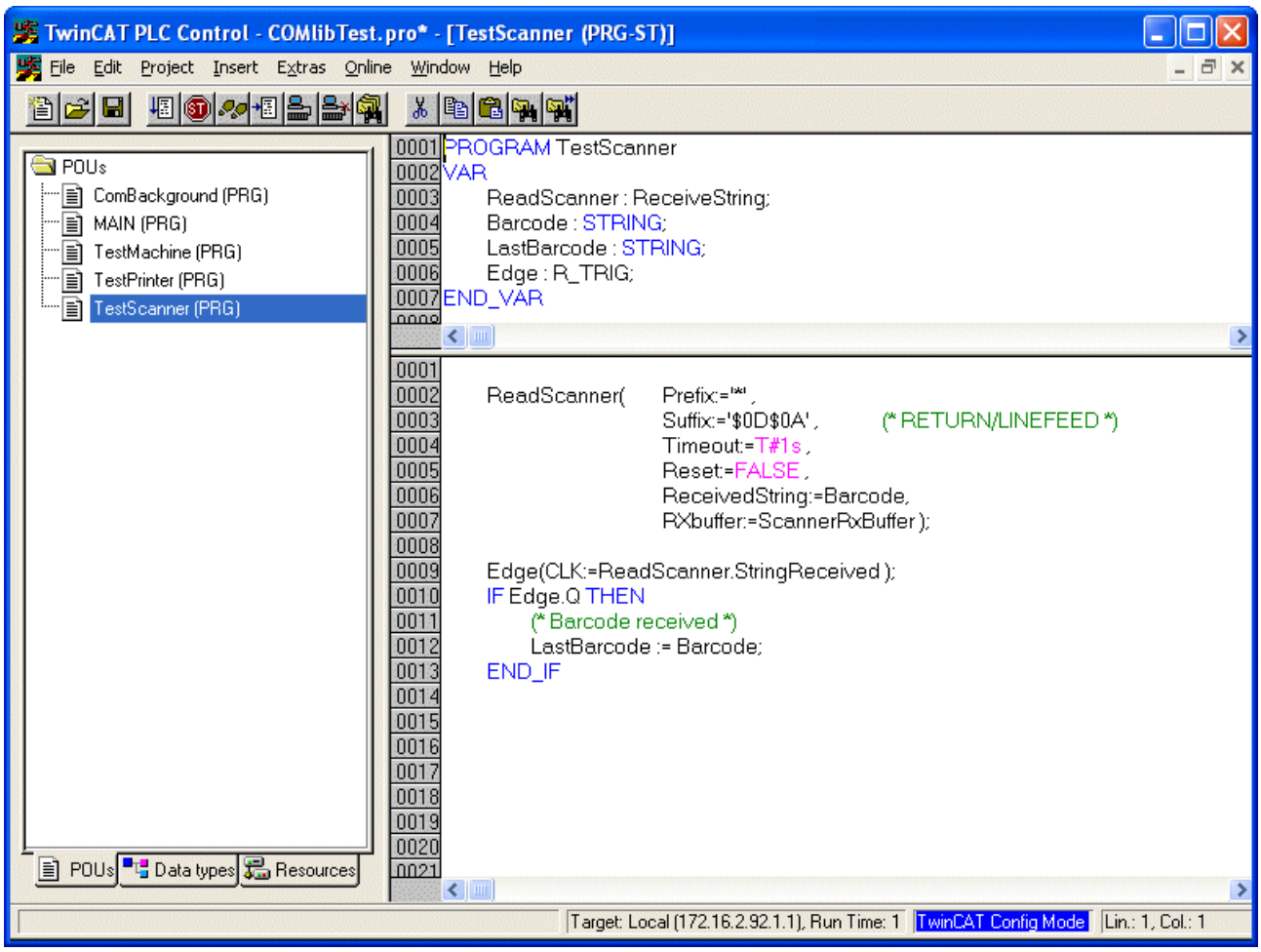


Possible errors

More than one character may be transmitted during one PLC cycle, provided they can be accepted by the send buffer. If the send buffer overflows, the busy output of the send block will remain TRUE after it has been called. In that case the last character is not sent, and the block must be called again with the same input data in the next PLC cycle. How full a buffer is can be determined at any time (e.g., TxBuffer.Count or TxBuffer.FreeByte).

Data reception

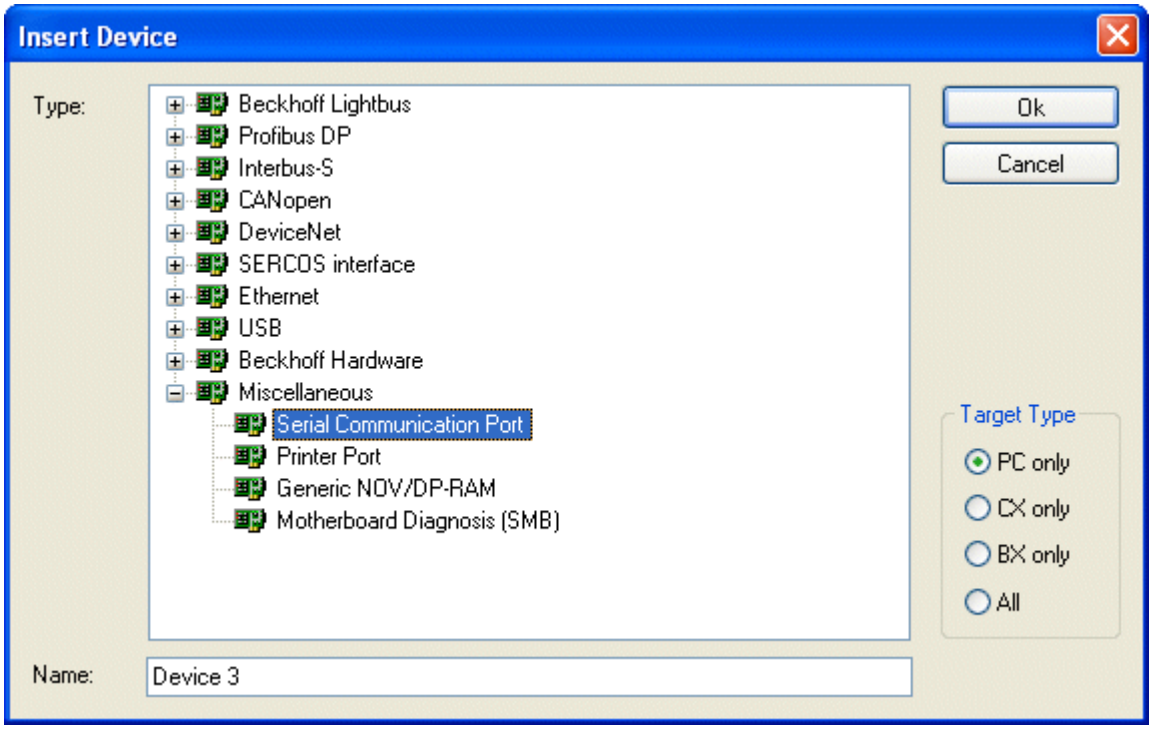
The example program receives a bar code from a scanner that is connected to a serial bus terminal.


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

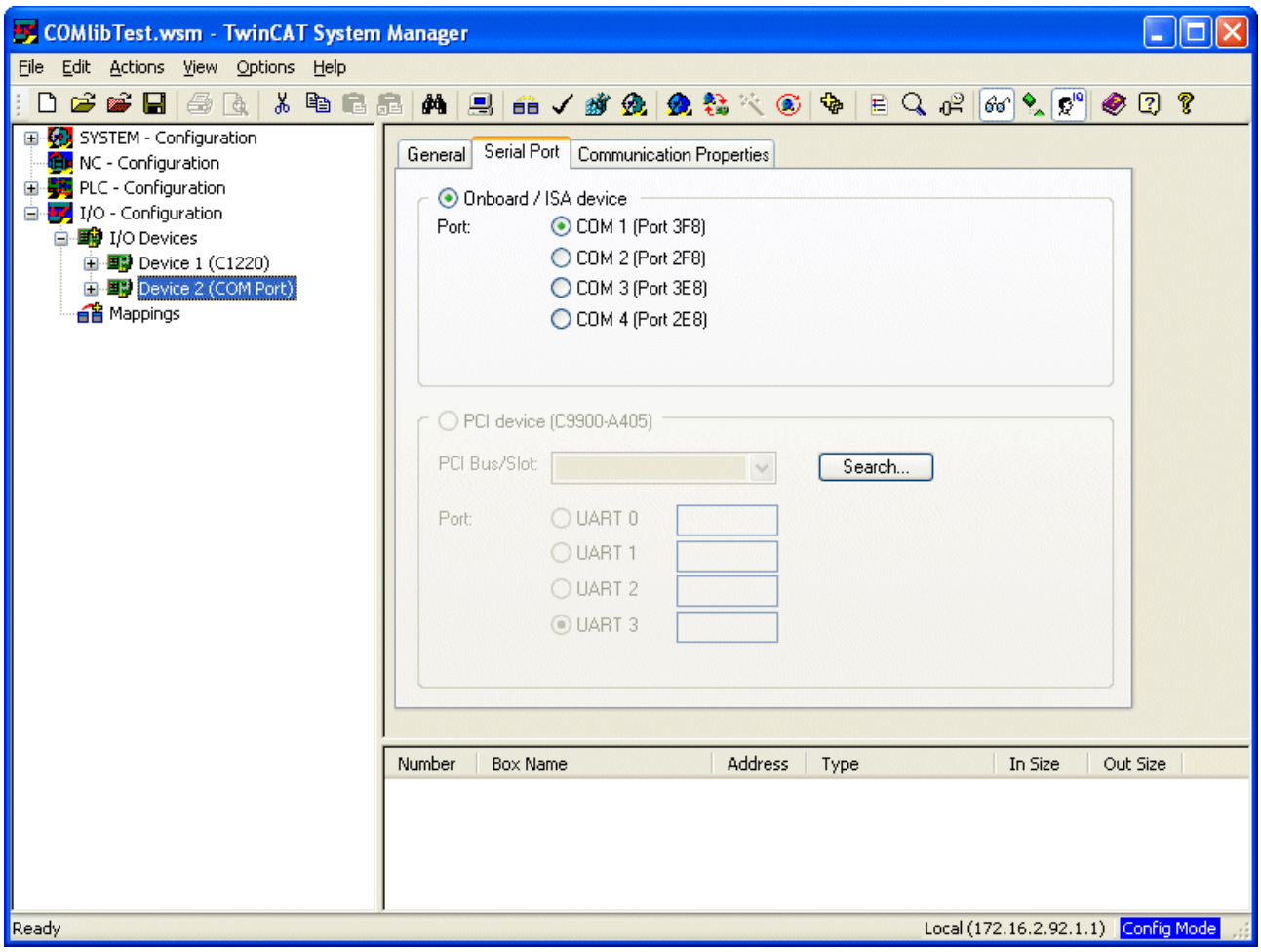
10 Configuration in the TwinCAT System Manager

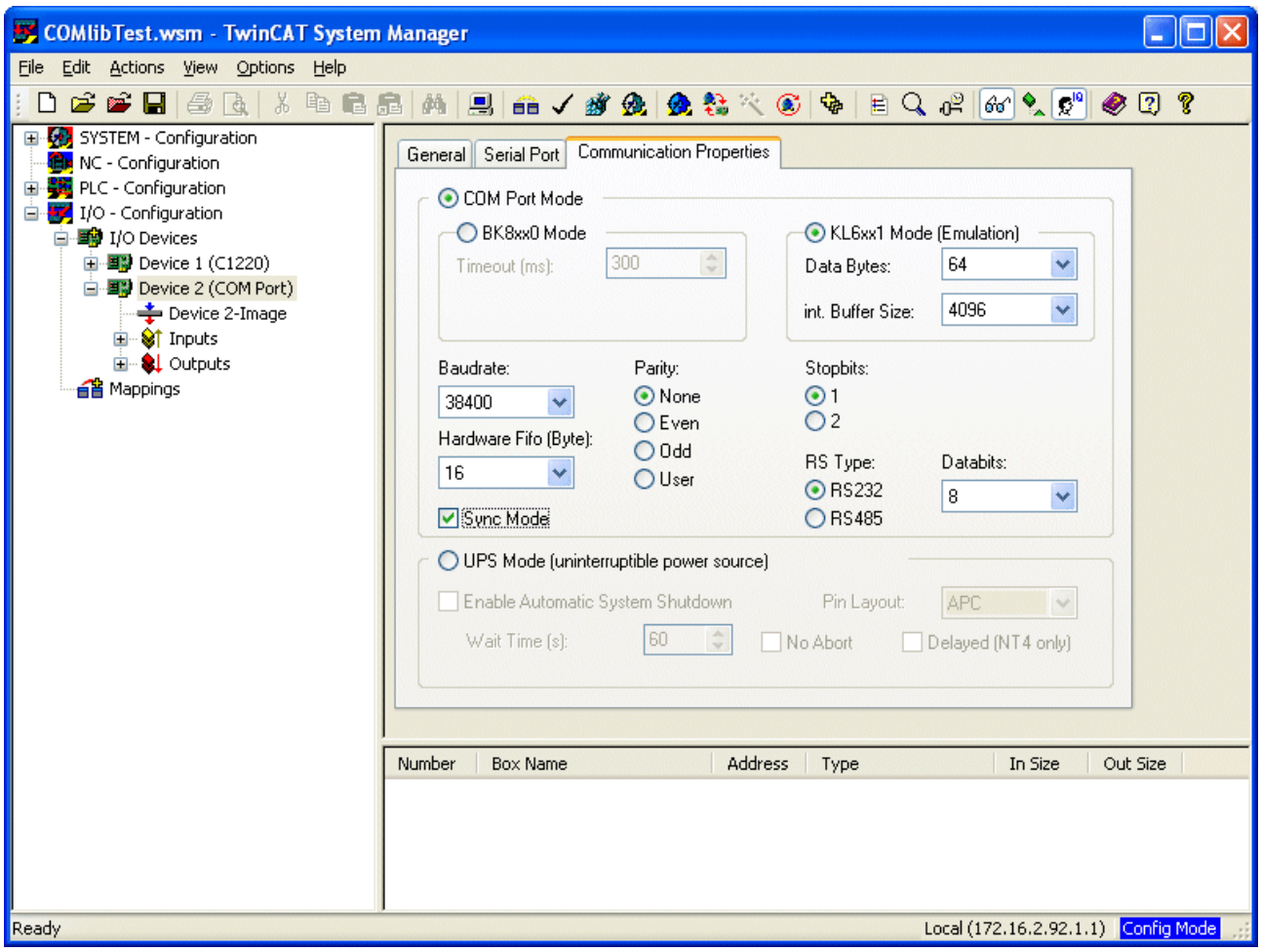
10.1 Serial PC Interface

The PC's standard serial interface is entered into the I/O configuration as a new I/O device.



After this the interface is configured:


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

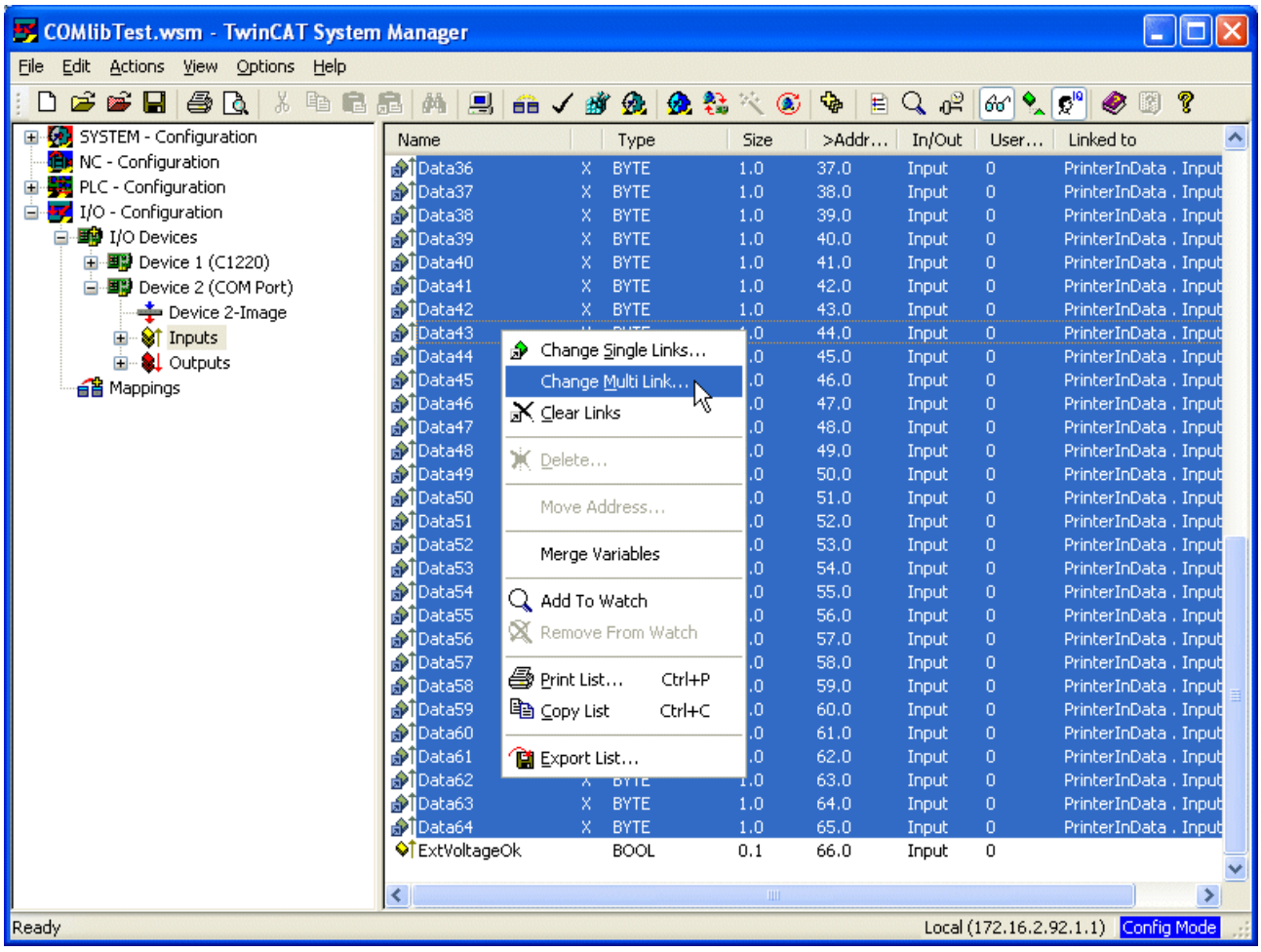
COMlib only supports KL6xx1 emulation operation. *Data bytes* must be set to 64 bytes. The interface parameters are set appropriately in the application.

Sync mode: In *Sync mode*, communication with the interface hardware is synchronised with the communication task. This setting usually offers benefits at high baud rates, as long the cycle time of the communication task is short enough. At 115 kBaud, for example, 12 characters are received each millisecond. The interface therefore has to be operated with a maximum cycle time of 1 ms, in order to avoid overflow of the 16-byte hardware FIFO. If the cycle times are too long, there is a risk of buffer overflow.

If *Sync mode* is switched off, the interface is served via the Windows timer interrupt every millisecond, irrespective of the task cycle time. This mode is not real-time capable, and with high computer utilisation longer operating intervals may be experienced. In this case, with very high baud rates there is also a risk of buffer overflow.

It is recommended to activate *Sync mode* and to adjust the cycle time of the communication task to the baud rate in such a way that overflow of the 16-byte hardware buffer is avoided. For smaller baud rates and with a slower communication task, *Sync mode* may perhaps be deactivated.

The next step is to link the inputs and outputs to the corresponding data structures in the PLC (types PcComInData [▶ 37] and PcComOutData [▶ 37]) by means of multiple linking.

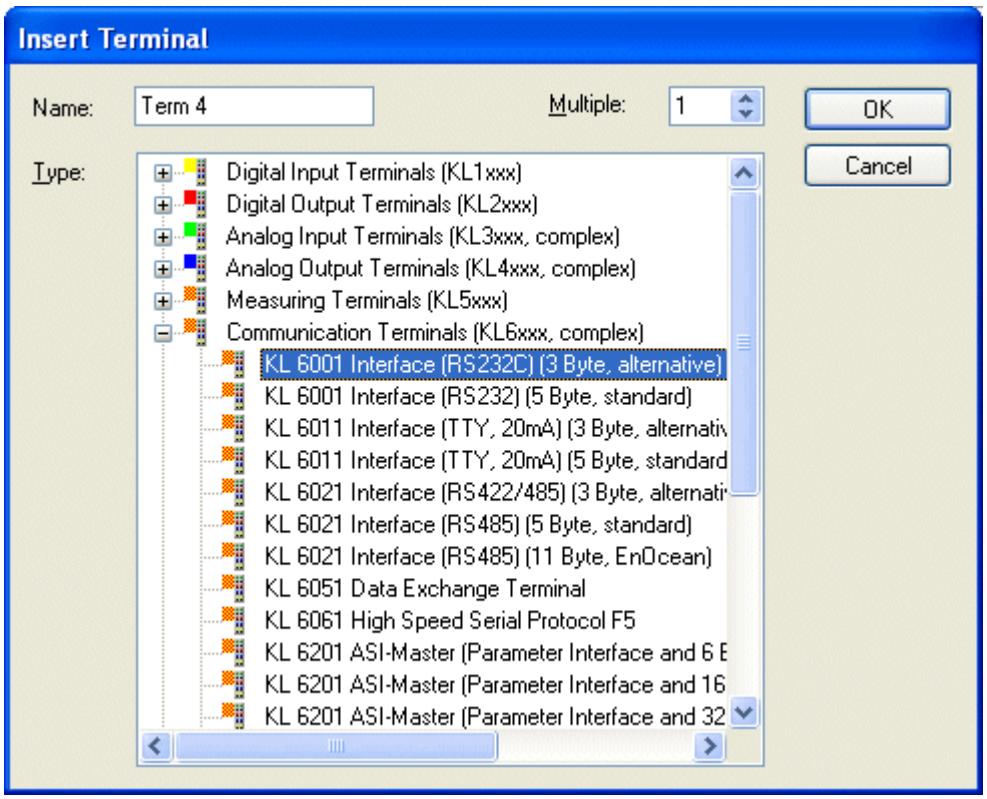


The new configuration is activated, and the system is restarted.

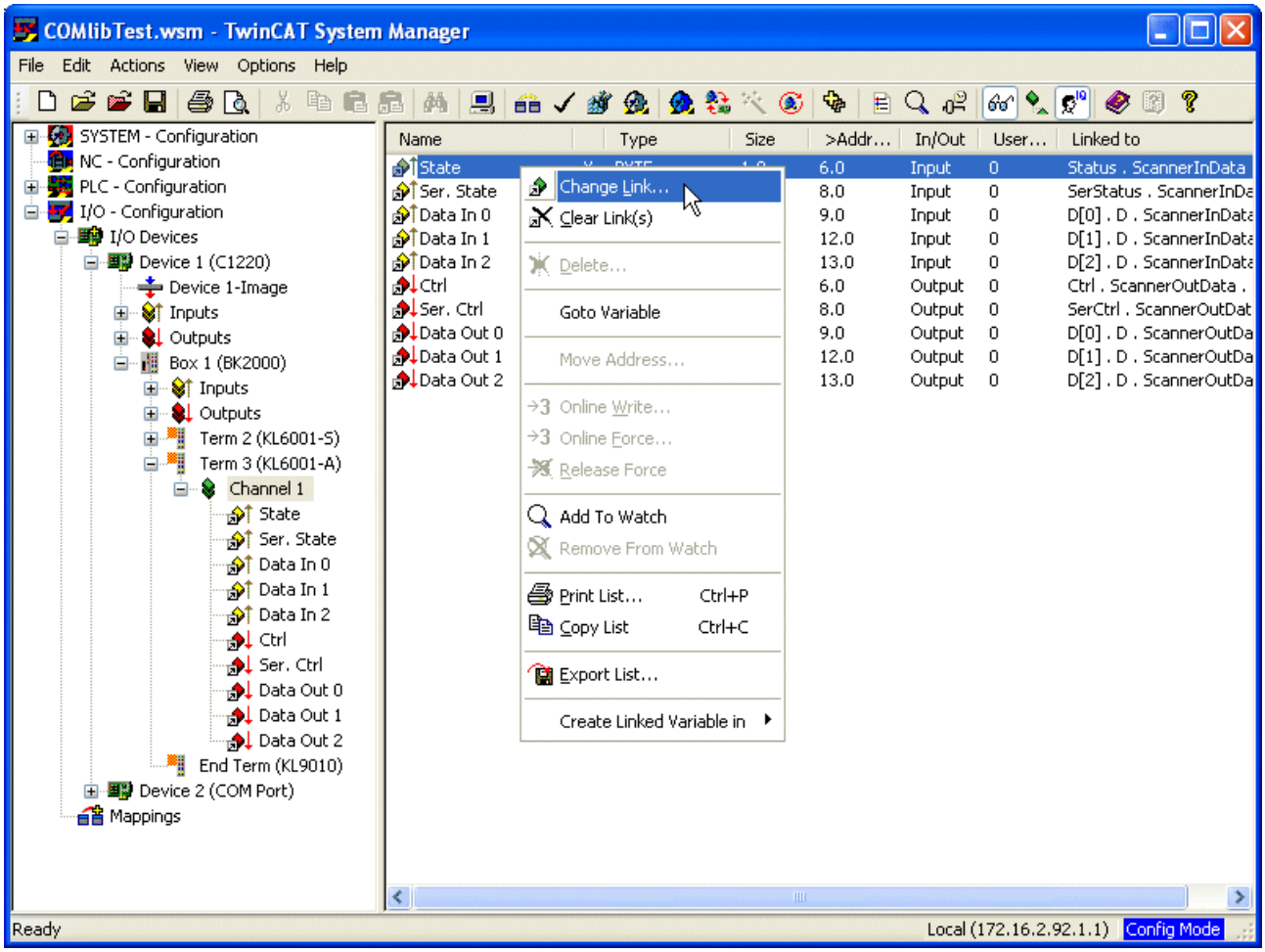
10.2 Serial Bus Terminal

KL6xxx in 3-Byte Mode

The serial bus terminal is entered into the system under a bus coupler.



Then the input and output data is linked with the corresponding PLC variables (of types KL6inData [▶ 36] and KL6outData [▶ 36]).



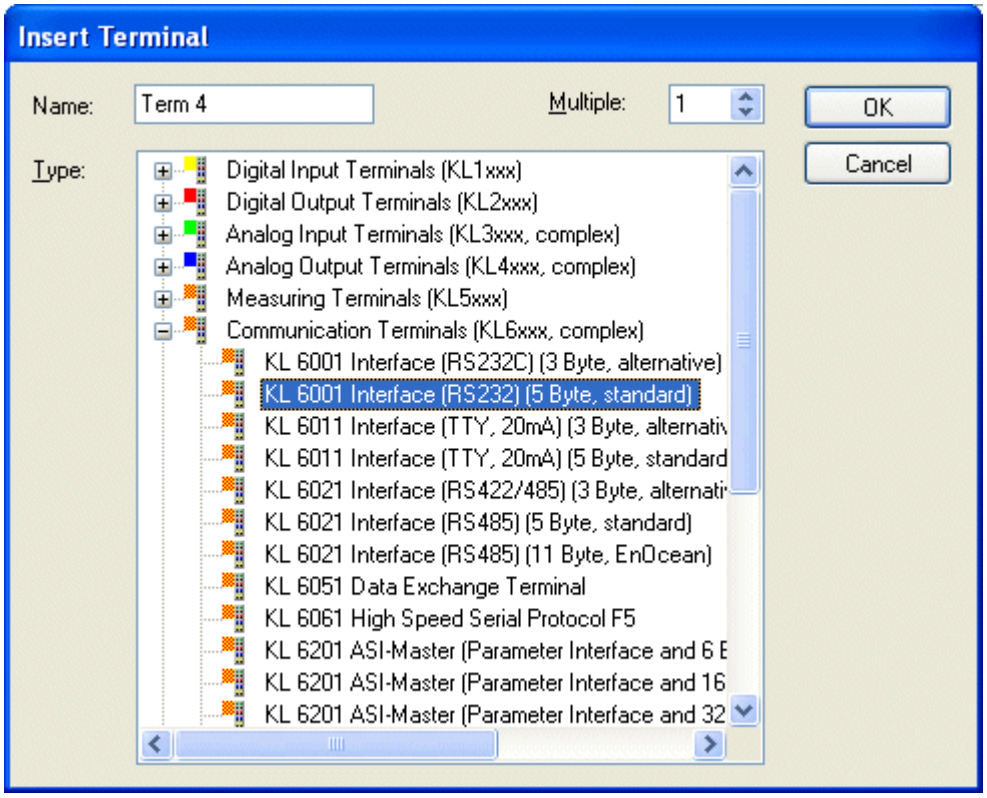
The new configuration is activated, and the system is restarted.

10.3 Serial Bus Terminal

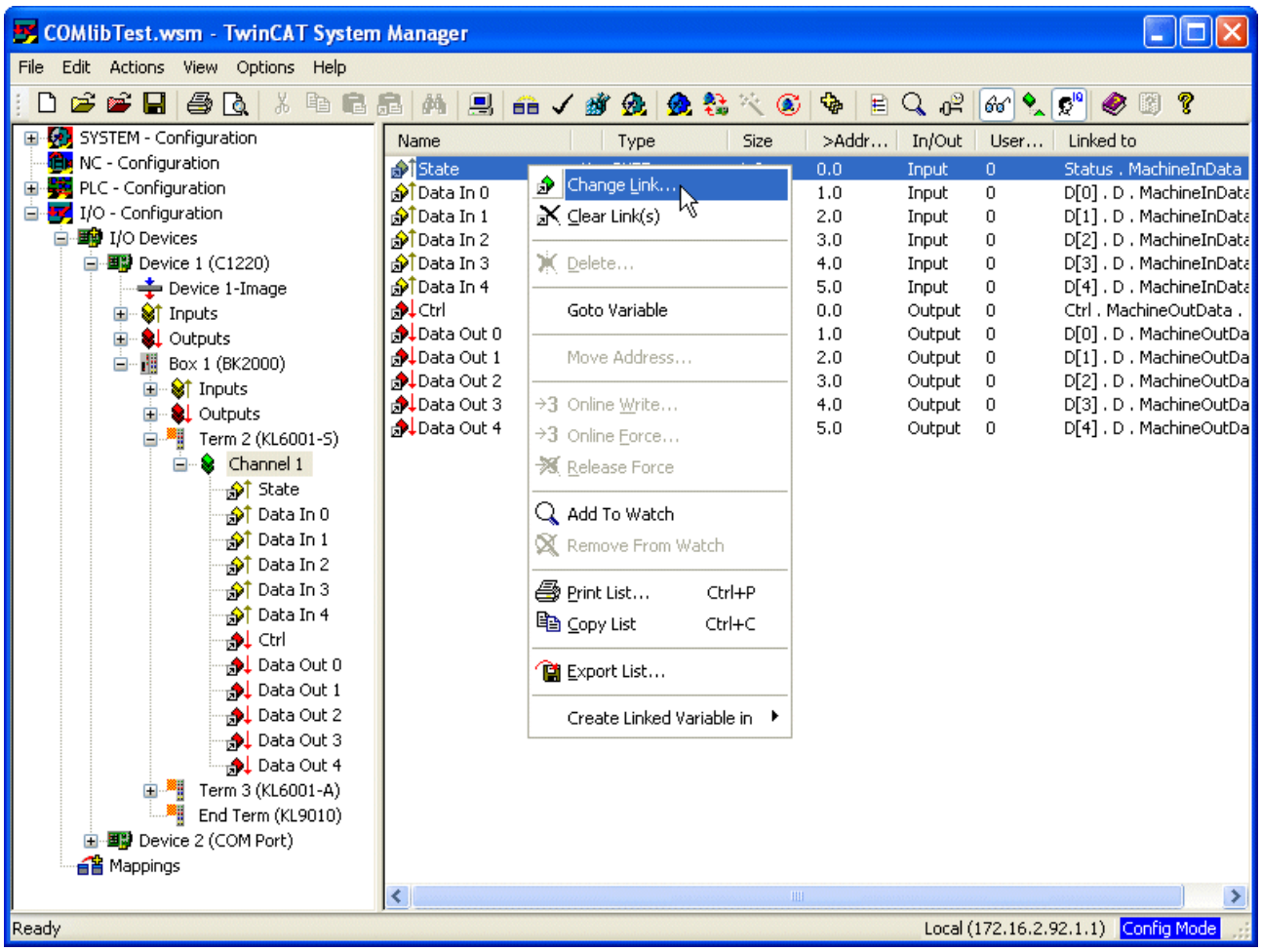
KL6xxx in 5-Byte Mode

Before a KL6xxx bus terminal can be used in 5-byte mode, it must be appropriately reconfigured. This can not be done through ComLib while running, but must be performed with the Beckhoff KS2000 configuration program. The terminal is then permanently set for 5-byte mode.

The serial bus terminal is now entered into the system under a bus coupler.



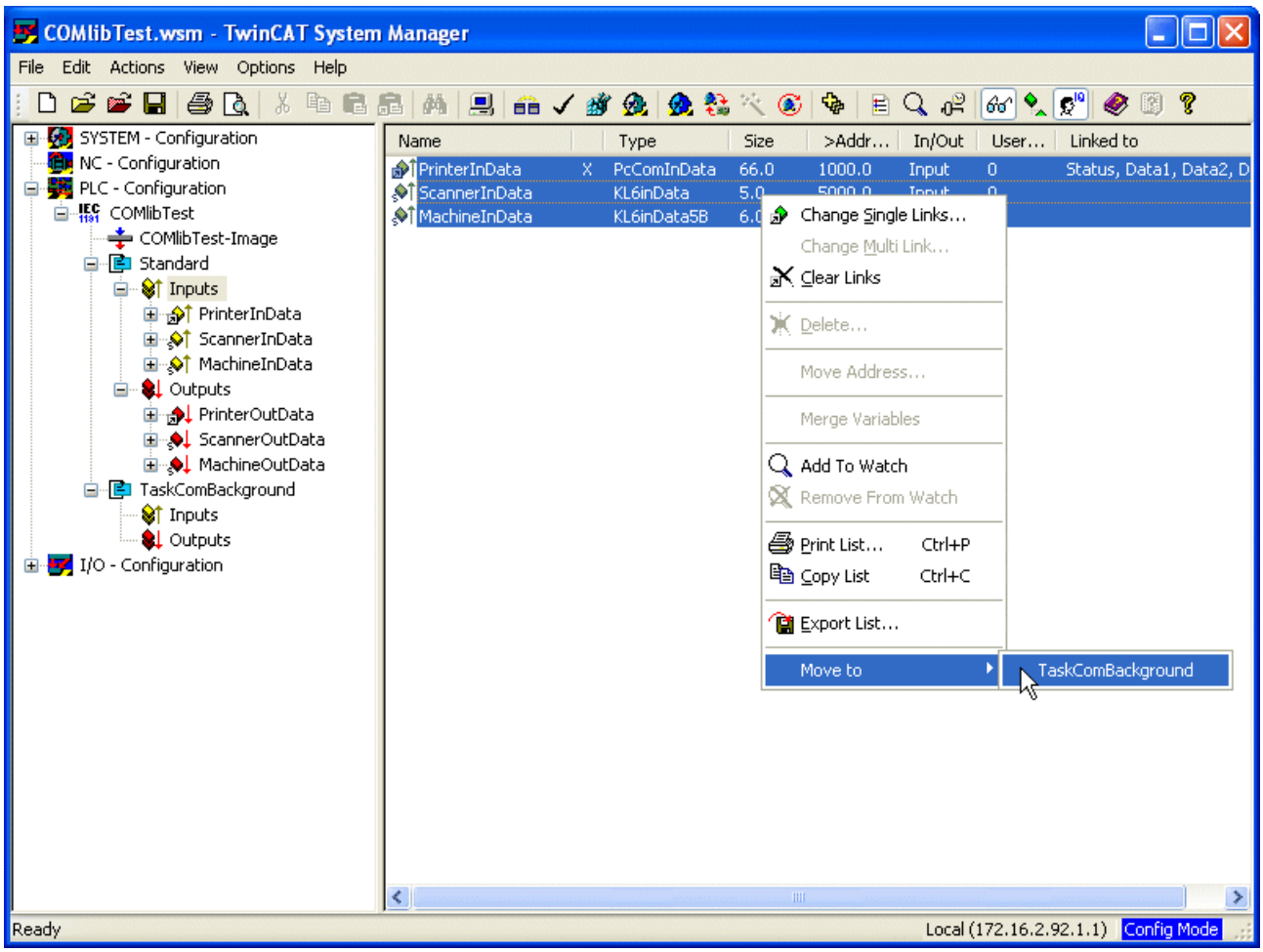
Then the input and output data are individually linked to the corresponding PLC variables (of types KL6inData5B [▶ 36] and KL6outData5B [▶ 36]).



The new configuration is activated, and the system is restarted.

10.4 Task Assignment

When declaring global variables in TwinCAT PLC Control, the individual variables are not assigned to a particular task. Therefore they initially appear in the TwinCAT System Manager under the standard task.

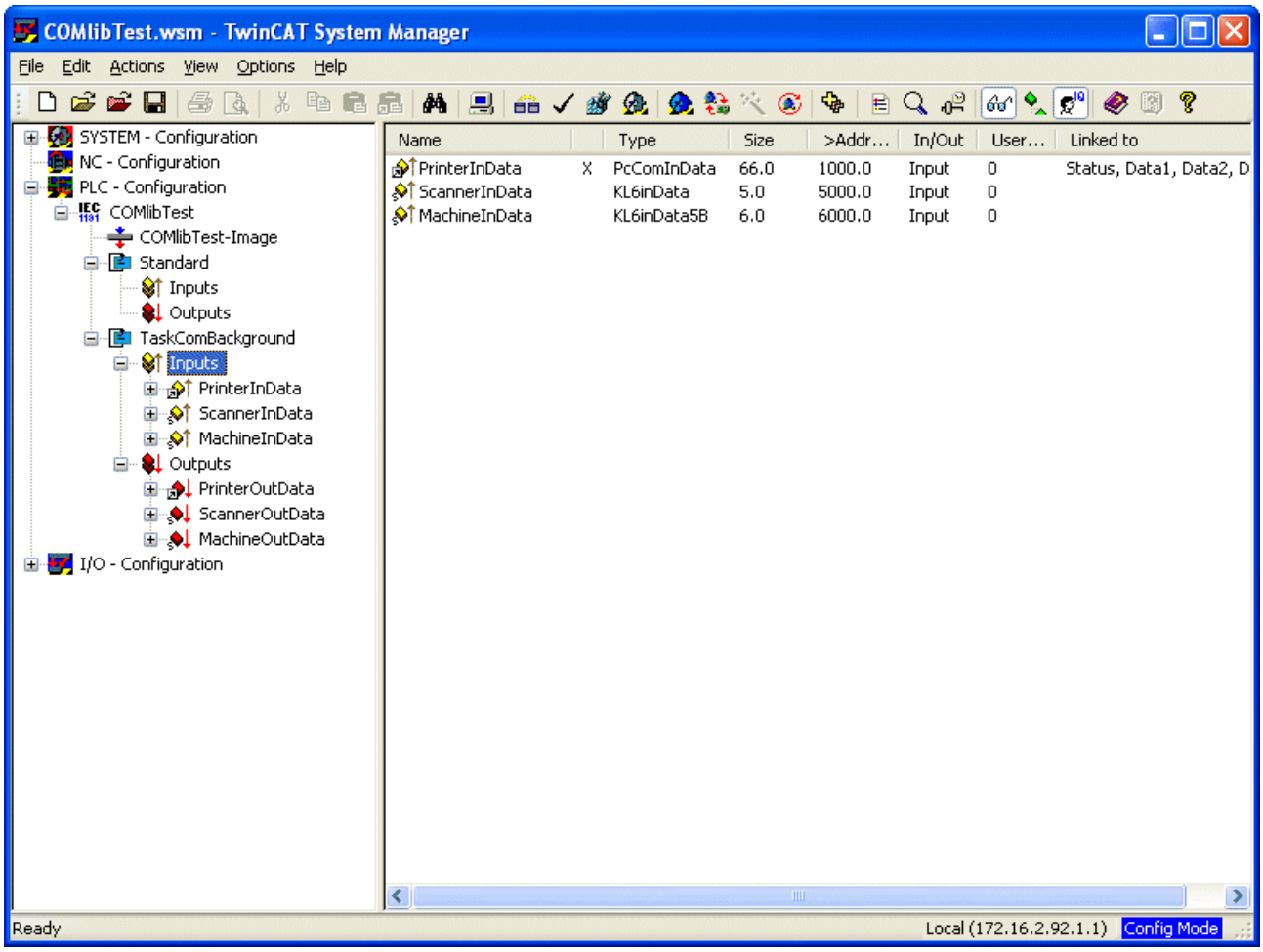


Since the task's cycle time determines the update cycle on the I/O bus, it is essential that the data structures for communication between the fast communication task and the hardware be assigned to that task. The corresponding entries under the standard task's inputs and outputs are dragged with the mouse or with the *Move to* context menu into the communication task.

All data structures of the following types are affected:

- KL6inData [▶ 36]
- KL6outData [▶ 36]
- KL6inData5B [▶ 36]
- KL6outData5B [▶ 36]
- KL6inData22B [▶ 37]
- KL6outData22B [▶ 37]
- EL6inData22B [▶ 37]
- EL6outData22B [▶ 37]
- PcComInData [▶ 37]
- PcComOutData [▶ 37]

Communication data structures in the fast communication task



11 Examples

The following examples have been developed with different hardware.

Sample 2 - Tutorial

The usage of the library function blocks is described as tutorial in the chapters 'Linking into a PLC Program [▶ 45]' and 'Configuration in the TwinCAT System Manager [▶ 52]'. The employment of different hardware is mentioned here too.

Example 3 - Application of virtual Com ports

supported with installation > v2.3.0

This example can be used for different applications with virtual Com ports.
It is possible to send and receive any data.

It is not so much an example as a test program that can be used to test the communication link to the USB device.
In an application using a virtual Com port, only the call to *SerialLineControlAds* is specific. The other serial communication calls, such as send and receive data, are identical to actual serial ports.

Observe the installation instructions on the overview page [▶ 8].
It is recommended to check the VirtualComPort driver of your device for functionality with a Windows terminal program as a first step.

No linking in the TwinCAT System Manager is required.
The example can be used to conveniently commission and test the device. The example features a visualization.

Project https://infosys.beckhoff.com/content/1033/tcplclibserialcom/Resources/11388463499/.zip

More Information:
www.beckhoff.com/ts6340

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