SIMATIC
Communication Library LCom for Ethernet Communication

Library Manual

02/2013

Safety Guidelines

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

⚠DANGER
indicates that death or severe personal injury **will** result if proper precautions are not taken.

⚠WARNING
indicates that death or severe personal injury **may** result if proper precautions are not taken.

⚠CAUTION
with a safety alert symbol, indicates that minor personal injury can result if proper precautions are not taken.

CAUTION
without a safety alert symbol, indicates that property damage can result if proper precautions are not taken.

NOTICE
indicates that an unintended result or situation can occur if the corresponding information is not taken into account.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

Qualified Personnel

The device/system/software may only be set up and used in conjunction with this documentation. Commissioning and operation of a device/system/software may only be performed by **qualified personnel**. Within the context of the safety notes in this documentation qualified persons are defined as persons who are authorized to commission, ground and label devices, systems and circuits in accordance with established safety practices and standards.

Prescribed Usage

Note the following:

⚠WARNING
This device/software may only be used for the applications described in the catalog or the technical description and only in connection with devices or components from other manufacturers which have been approved or recommended by Siemens. Correct, reliable operation of the product requires proper transport, storage, positioning and assembly as well as careful operation and maintenance.

Trademarks

All names identified by ® are registered trademarks of the Siemens AG. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

Siemens AG
Industry Sector
Postfach 48 48
90327 NÜRNBERG
GERMANY

Foreword

General Notes

NOTICE

The standard applications are not binding and do not claim to be complete regarding the circuits shown, equipping and any eventuality. The standard applications do not represent customer-specific solutions. They are only intended to provide support for typical applications. You are responsible in ensuring that the described products are correctly used. These standard applications do not relieve you of the responsibility in safely and professionally using, installing, operating and servicing equipment. When using these standard applications, you recognize that we cannot be made liable for any damage/claims beyond the liability clause described. We reserve the right to make changes to these standard applications at any time without prior notice. If there are any deviations between the recommendations provided in these standard applications and other Siemens publications - e.g. catalogs - then the contents of the other documents have priority

Guarantee, Liability and Support

If the application is provided free of charge the following shall apply:

We shall not be liable for the information contained in this document.

Any and all further rights and remedies against Siemens AG for whatsoever legal reason, shall be excluded; this shall refer in particular to claims for loss of production, loss of use, loss of orders or profit and other direct, indirect or consequential damage.

The aforesaid shall not apply if liability is mandatory, e.g. in accordance with the Product Liability Act, in cases of intent, gross negligence by directors and officers of Siemens AG or in the case of willful hiding of a defect.

These limitations of liability shall also apply for the benefit of the Siemens AG's subcontractors, suppliers, agents, directors, officers and employees.

This Contract shall be subject to German law if customer’s place of business is in Germany. If customer’s place of business is outside of Germany the Contract shall be subject to Swiss law. The application of the UN Convention on Contracts for the International Sale of Goods (CISG) shall be excluded.

If the application is provided in return for payment the alternative shall apply which fits the respective business case:

- Alternative 1: (internal business)

  If not explicitly stated otherwise below, the "Terms and Conditions for Deliveries and Services for Siemens Internal Transactions", valid at the time of sale, are applicable.
- Alternative 2: (domestic business of Siemens AG)

  If not explicitly stated otherwise below, the "General License Conditions for Software Products for Automation and Drives for Customers with a Seat or registered Office in Germany", valid at the time of sale, are applicable.
- Alternative 3: (direct export business of Siemens AG)

  If not explicitly stated otherwise below, the "General License Conditions for Software Products for Automation and Drives for Customers with a Seat or Registered Office outside of Germany", valid at the time of sale, are applicable.

It is not permissible to transfer or copy these standard applications or excerpts of them in unmodified form without first having prior explicit authorization from Siemens Industry Sector in writing.

Export Procedure Indicator

AL: N

ECCN: N

About this manual

Target group / objective

The target group is programmers and application engineers of machine manufacturers who want to connect their machine to other controllers or devices with Industrial Ethernet and use the TCP protocol for the data exchange. Information, e.g. the machine status, line speed, etc. can be exchanged with upstream and downstream machines as well as higher-level visualization and MES/MIS systems during integration in a production line.

Third-party systems, such as cameras, PCs or controllers can also be integrated on the basis of the standard Industrial Ethernet.

Core contents

This document describes the **LCom** library. This contains a function block for TCP communication via Industrial Ethernet. An extended transport protocol, the LCom protocol, has been implemented for extended communication functions.

A separate **LCom** library is provided for SIMOTION controllers that has the same functions and enables the data exchange between SIMOTION and SIMATIC systems.

Differentiation

This documentation refers to the **LCom** library for SIMATIC controllers.
The **LCom** library for SIMOTION is described in a separate document.

The communication is based on the TCP protocol. An extension to UDP communication is not implemented.

In the case of controllers with integrated PROFINET interface, the **FBLComMachineCom** function block is based on the communication blocks **FB63** to **FB66**.

With this version V1.1, the **LCom** library can also be used for CP modules. The functions **FC5**, **FC6** and **FC10** are used for this, whereby a distinction is made between **CP343** and **CP443**.

Table of contents

Library description 1

1.1 Field of application

1.1.1 Controllers and devices

All types of controllers and devices that have an Ethernet interface and support the TCP protocol can be connected, e.g. cameras, printers, scanners and PCs.

For the extended communication functions, the LCom protocol must be implemented on the peer side.

1.1.2 Area of application

Standard mechanisms can be used for networks that include devices from different system families. The widespread and versatile Industrial Ethernet and the standard protocols TCP, UDP and IP in use worldwide, are suitable as a bus system.

In the field of automation, the TCP protocol based on Industrial Ethernet is typically deployed for safe, time non-critical (> 25 ms) and multi-system communication.

In the SIMATIC automation system, certain controller types have the required interfaces and system functions for TCP communication.

The **FBLComMachineCom** function block in the **LCom** library provides the function of a data record-oriented transport protocol, LCom protocol. The TCP transport protocol is used for data transmission via Ethernet.

Further communication functions via TCP are also implemented in the **FBLComMachineCom** function block.

Currently, only function blocks for SIMOTION and SIMATIC devices are available for extended communication functions (with a separate library for SIMOTION). Adopting the standard functions of the TCP protocol simplifies the implementation of additional functions on other devices, e.g. the PC via VisualBasic.

1.2 Objective

1.2.1 Description of the task

The **FBLComMachineCom** function block of the **LCom** library is to be used to implement a point-to-point communication via Industrial Ethernet based on the TCP protocol.

The TCP protocol ensures the point-to-point transmission of a continuous flow of data between two devices. The TCP protocol does not provide a mechanism for transmitting files (data record with defined length, ID for start and end). The transport protocol defined in the **LCom** library is to be used to define such a mechanism (comparable to FTP and RFC1006, for example).

For further functions not offered by the TCP protocol or the controller (e.g. transmission of data up to 64 KB, flow control, time synchronization, etc.), it is necessary to extend the mechanisms of the TCP protocol.

Data can be published in different ways depending on its significance.

- Control values are usually transmitted when the data is changed.
- Status data, e.g. actual values, is usually transmitted cyclically.
- Large data quantities, e.g. for interpolation points of a cam, are usually transmitted once.

The different transmission modes let the user define how the data is published.

The function block is also intended for use in standard TCP communication (native TCP).

1.2.2 Benefits

The **LCom** library implements a point-to-point connection in both directions (full duplex) between two devices.

The function block can be used for standard TCP communication. The widespread deployment of the TCP standard allows data exchange to be implemented between two different devices.

Since the functional scope of TCP is not sufficient for many applications, the **LCom** library defines a separate transport protocol (the LCom protocol).
The following additional functions have been defined:

- Data records with defined length up to 64 KB.
- Different transmission modes.
- Monitoring of the connection through cyclic transmission of a sign-of-life (faster response times on connection failure). In the case of a pure TCP communication, this is typically in the range of seconds.
- Prevention of a full working memory on the system level through frequent sending of data via integrated flow control on the application level.
- Feedback from receiver on successful data transmission on application level.
- Simple time synchronization.
- Data transport in both transmission directions can be started and stopped by both communication peers.

The function block can be instantiated and thereby used for several communication relationships of a SIMATIC controller (e.g. in a production line both to the upstream and to the downstream machine).

1.3 Concept

1.3.1 General

Industrial Ethernet is used as the transmission medium for the communication with the **FBLComMachineCom** function block and TCP as protocol.

The TCP transport protocol guarantees a continuous flow of data. TCP is not packet-oriented and therefore does not permit transmission of data records with a defined overall length.

The communication block **FB63** TSEND in SIMATIC allows a specific maximum volume of data (depending on the controller type) to be sent with each call. This data is divided into TCP message frames (max. 1460 bytes of useful data, with CP, a maximum of 240 bytes of user data) by the the TCP stack. The actual number of bytes of user data per message frame is defined autonomously by the TCP stack and cannot be influenced by the user. This is why several packets of different sizes can arrive at the peer device. This means that it is not possible to establish where a contiguous message starts and stops without applying further mechanisms.

The situation is shown in Figure 1–1 Sending and receiving via TCP [Page 11].
Data packets of 4096 bytes are transferred to the communication block **FB63** TSEND. These data packets are divided into TCP message frames at the network level, transferred, and entered in the TCP receive buffer of the receiver side.

This buffer is output by the receiver application. Depending on the time at which the data is read out, this buffer can contain a different number of bytes. The task of the application is to assign the received bytes to the original data packets again.

If the LCom protocol is used, the user does not have to perform this task.

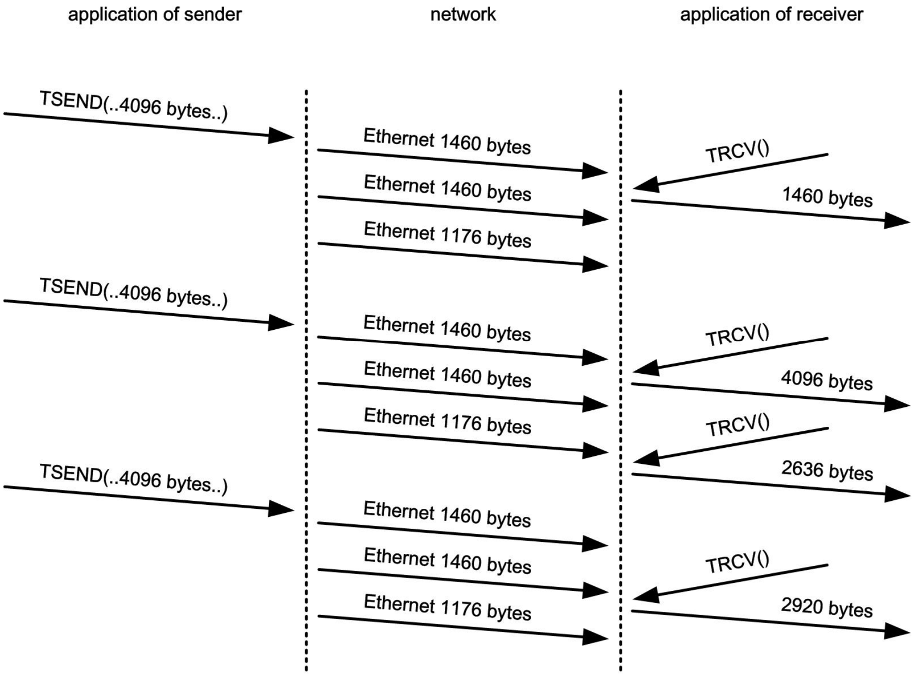


Figure 1–1 Sending and receiving via TCP

It is possible to establish a point-to-point connection between two devices. Data is exchanged in both directions (full duplex mode). Each device can operate as sender and receiver simultaneously.

An IP address and port number via which the communication is performed, must be assigned for each device. Note that both devices are in the same network, and that when configuring the two devices are assigned the IP addresses of the same subnet. If routers are used, the IP addresses of the two devices may be in different networks.

In TCP communication, one device is the server and the other device is the client. This should be defined at the time of configuration. The port number and IP address of the server must be known to the client.

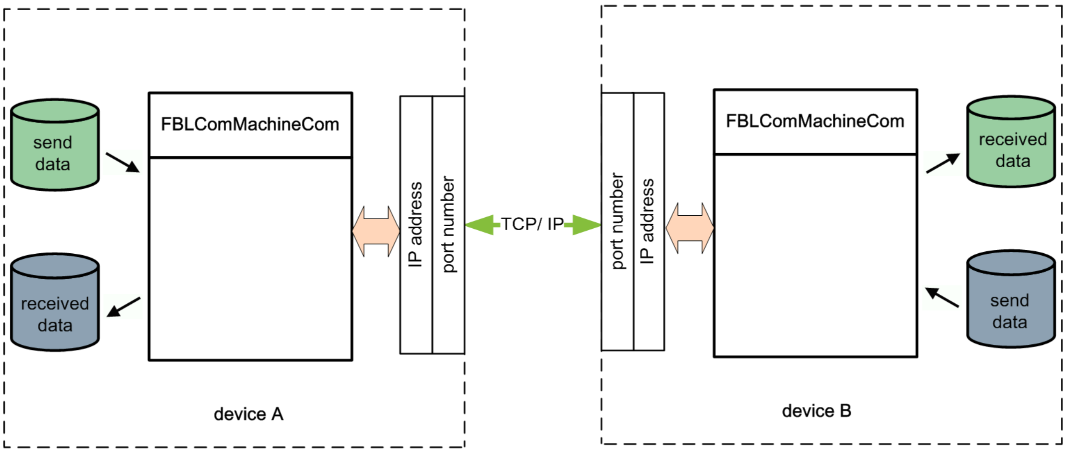


Figure 1–2 Communication link between two devices via TCP/IP

In addition to the standard TCP functions, the **FBLComMachineCom** function block also provides further communication functions. The communication peer must support these additional properties with the LCom protocol. These extended communication functions are described in detail in Section Using the LCom protocol [page 14].

Alternatively, it is also possible to exchange data with the **FBLComMachineCom** function block via TCP without using the LCom protocol. However, the supplementary conditions for the TCP protocol mentioned above must be taken into account. These are explained in Section TCP communication [page 26].

1.3.2 Type of data transmission

The **FBLComMachineCom** function block supports three different types of data publishing.

Cyclic transmission

For cyclic data transmission, the user defines a cycle time. The data is sent to the communication peer in this fixed time grid. This type of transmission is used primarily for status data that changes constantly, e.g. axis position or temperature values.

Event-driven transmission

In event-driven data transmission, the data is sent once when communication is activated. Then the send data is only published when the data is changed. When a change is detected, all the data is sent to the communication peer.

This transmission mode can be used for control data. This data normally changes less often than status data.

The check for changes places a greater load on the controller, but reduces the communication load on the bus system.

When the previous data has been sent completely, a further check for changes is made. If the data changes continuously, it is sent to the peer as quickly as possible.

Single transmission

With a single transmission, the data is sent once and immediately. Republishing requires explicit re-selection by the operator.

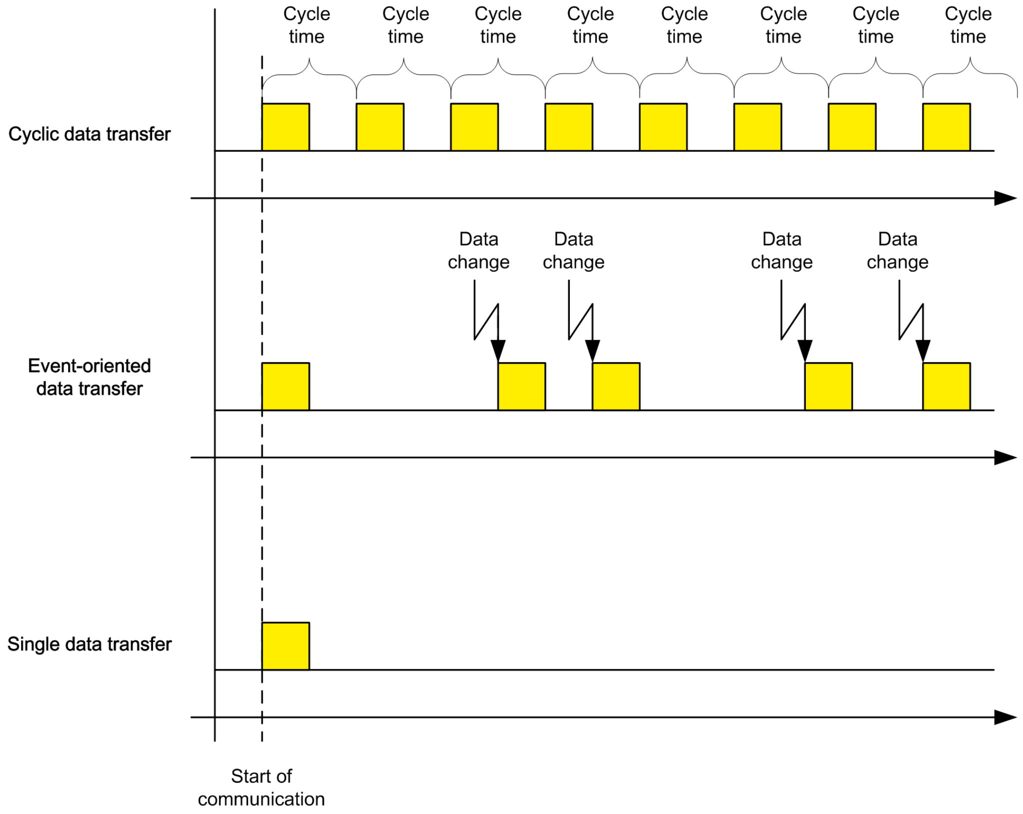


Figure 1–3 Overview of the three types of data transmission

Mixed operation for data transmission

The **FBLComMachineCom** function block can set up one send channel and one receive channel. A separate transmission mode can be defined for each channel. In a communication relationship between a master controller and a subordinate device, for example, this allows event-driven transmission of control data from the master controller, and cyclic transmission of status data from the subordinate controller.

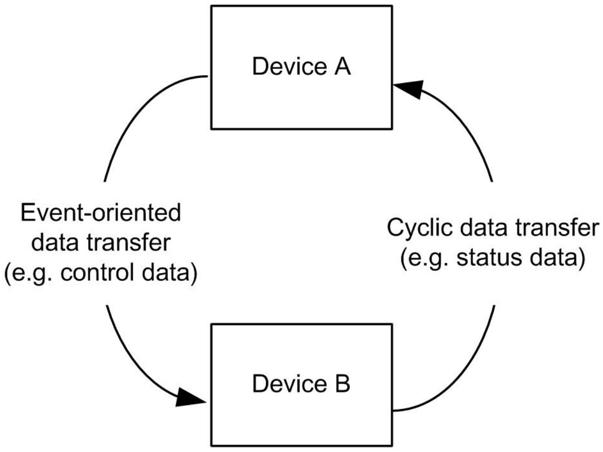

Device A
Event-oriented data transfer (e.g. control data)
Cyclic data transfer (e.g. status data)
Device B

Figure 1–4 Different transmission modes for a connection

1.3.3 Using the LCom protocol

Additional information must be transferred for the integration of further functions by the LCom protocol. This information is entered in a separate header between sender and receiver. This LCom header is evaluated by the **FBLComMachineCom** function block and the appropriate functions processed.

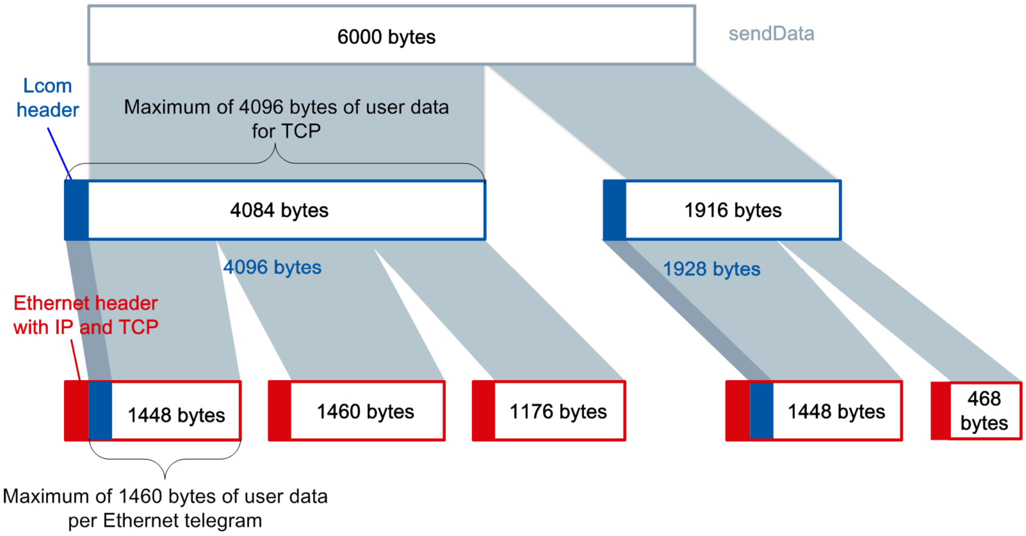

6000 bytes
sendData
Lcom header
Maximum of 4096 bytes of user data for TCP
4084 bytes
1916 bytes
4096 bytes
1928 bytes
Ethernet header with IP and TCP
1448 bytes
1460 bytes
1176 bytes
1448 bytes
468 bytes
Maximum of 1460 bytes of user data per Ethernet telegram

Figure 1–5 Example: Sending of a 6 KB large data record using the LCom protocol

Figure 1–5 Example: Sending of a 6 KB large data record using the LCom protocol [Page 14] illustrates how the protocols involved in data transmission function. A 6000-byte data record is sent via Ethernet using the LCom protocol and the TCP protocol.

In SIMATIC, data packets of maximum 4096 bytes are transferred to the TCP stack at every call of the communication block **FB63** TSEND. The LCom header (12 bytes) is inserted in each of these data packets. Therefore, when using the LCom protocol a maximum of 4084 bytes of user data can be sent per **FB63** TSEND call. The 6000-byte data record is divided into two data packets (4084 and 1916 bytes) in accordance with this maximum size.

Each Ethernet message frame (1500 bytes) which also contains an IP header and TCP header (in each case 20 bytes), can contain a maximum of 1460 bytes of user data. In this case, the LCom header also belongs to the user data. Correspondingly, the first data packet is divided into three TCP message frames and the second data packet into two TCP message frames. The user has no influence over these mechanisms. The size of the user data is also specified by the system (TCP stack). In the example, see Figure 1–5 Example: Sending of a 6 KB large data record using the LCom protocol [Page 14], the user data of each TCP message frame is 1460 bytes large (or the number of remaining bytes).

The division of the user data to be transmitted into individual data packets (4096 bytes incl. LCom header) is performed in the **FBLComMachineCom** function block. The division of these data packets into TCP message frames is performed by the SIMATIC system.

The extended functions, as compared to the TCP standard, are described below.

1.3.3.1 Sending contiguous data

If the LCom protocol is used, information about the start and end of the message is also transmitted. The function block signals when message transmission (data record transmission) is completed and only then does it output the complete message. In this way, data transfer is comparable to FTP.

1.3.3.2 Extended data size

With TCP communication in SIMATIC, a specific maximum data length, depending on the type of controller, can be transferred to the communication block **FB63** TSEND.

The **FBLComMachineCom** function block contained in the **LCom** library can be used to transfer data records up to a length of 64 KB. In this way, it is possible, for example, to safely transfer the interpolation points of a cam. To ensure that the data can be transferred by the communication block **FB63** TSEND, a segmentation of the data record is performed in the **FBLComMachineCom** function block of the sender side. The receiver side combines the received data packets into the complete message (data record).

1.3.3.3 Sliding window

To confirm successful transmission of data packets, the receiver must send acknowledgements to the sender. No further data packets are sent to the receiver until the acknowledgements are received. This ensures that the receiver actually receives the packet and that the sender cannot overfill the working memory of the receiver. This would happen if the receiver collected data slower than it was sent.

In order to reduce the bus load and waiting times for acknowledgement message frames, it is possible to transfer a certain number of data message frames without acknowledgement. In this case, each data packet is not acknowledged explicitly. With acknowledgement of a subsequent packet, it is assumed that the previous packet has been received.

This function is called "Sliding window". The number of consecutive packets that can be transferred without acknowledgement can be configured.

This function is part of a flow control that ensures optimum data exchange between the two communication peers.

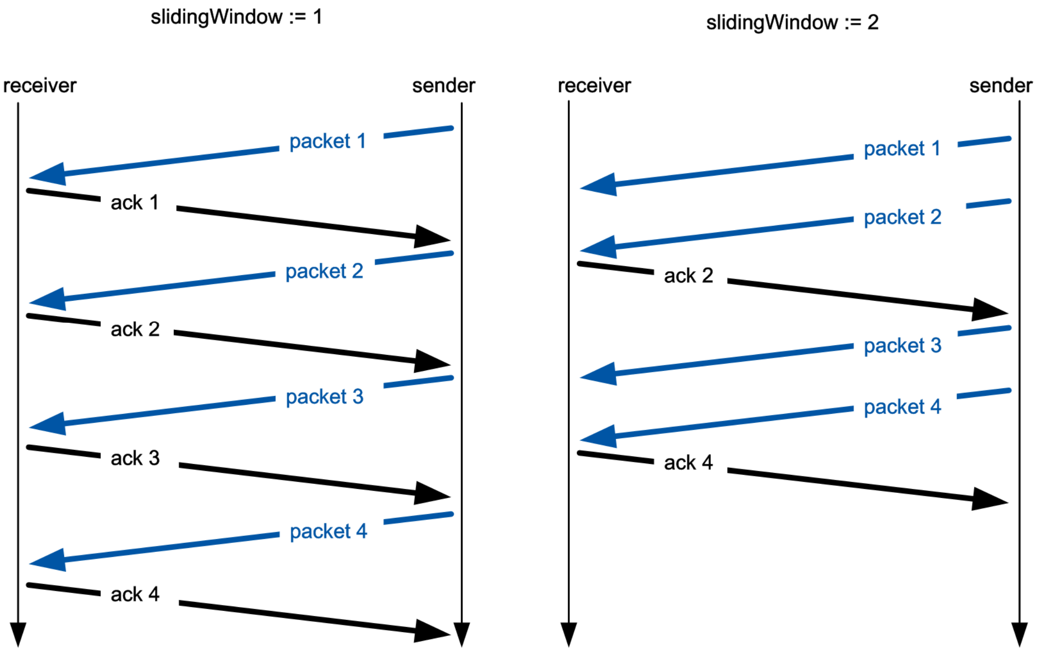


Figure 1–6 Sliding window function with values 1 and 2

1.3.3.4 Acknowledgement of data packets

If the receiver does not acknowledge a data packet within a defined period (ackTimeout), an error must be assumed in the communication with LCom protocol. An error is output on the **FBLComMachineCom** function block and the connection is closed. An attempt is then made to re-establish the connection.

This monitoring method on the application level enables faster detection of a failure in the connection during data transmission than with standard TCP communication. A sign-of-life is used to monitor the connection when there is no active exchange of data.

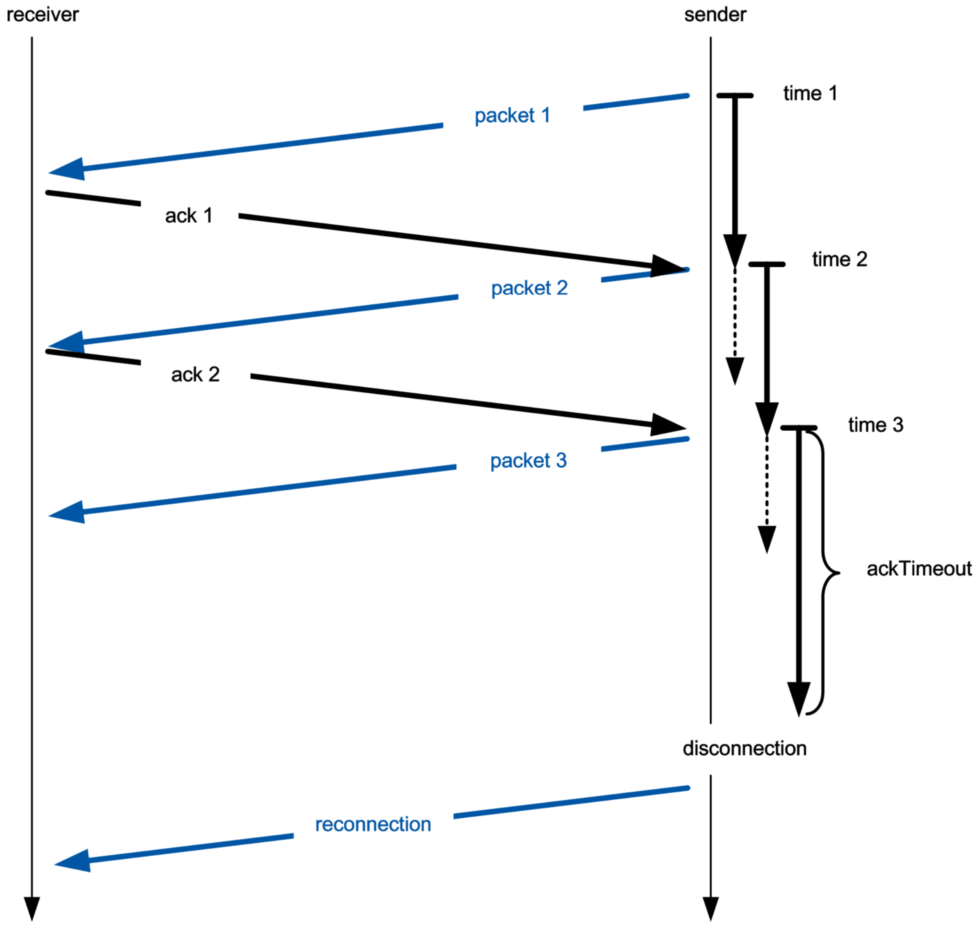


Figure 1–7 Acknowledgement monitoring and reaction

1.3.3.5 Sign-of-life

In standard TCP communication, it can take up to 10 s to detect a failure in the communication peer. Sign-of-life monitoring reduces this period considerably. The sign-of-life is only sent when the LCom protocol is used.

If data is being transmitted, it can be assumed that the connection is error-free. This information is not available during a break in transmission. This is relevant for all three transmission modes:

- In cyclic transmission, this can occur when a long cycle time is used.
- In event-driven transmission, it may be that the data does not change for a longer period and is therefore not transmitted.
- If single transmission is used, no more data is sent after data exchange has been completed.

In theses phases in which no user data is exchanged, a sign-of-life is generated automatically. The cycle time of the periodic sign-of-life can be configured. If no new sign-of-life or new user data is received after three sign-of-life cycles, it is assumed that the connection has been broken and an appropriate message is issued.

NOTE
The number of sign-of-life cycles before a broken connection is reported is specified as 3.

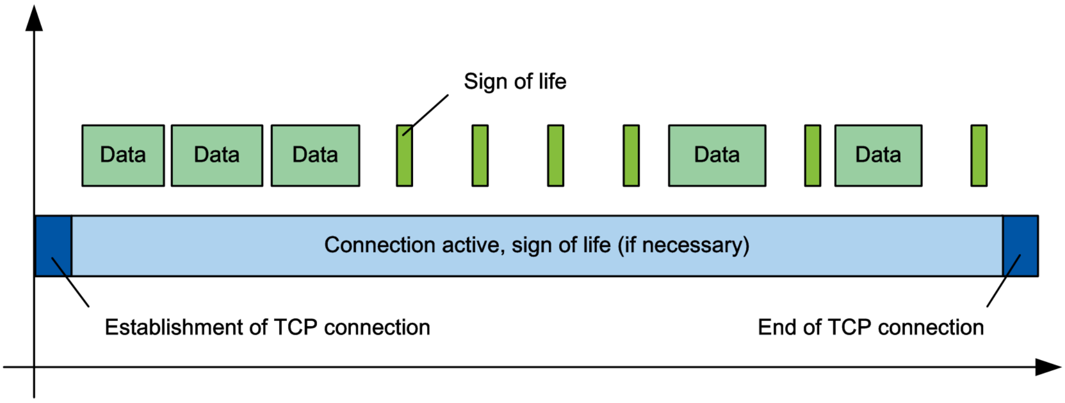

Sign of life
Data
Data
Data
Data
Data
Connection active, sign of life (if necessary)
Establishment of TCP connection
End of TCP connection

Figure 1–8 Sign-of-life monitoring

1.3.3.6 Time synchronization

For time synchronization of two controllers, you can send the current time of one controller to the peer and use it there as the system time. The typical deviation after synchronization of the two system times is 50 ms.

A distinction is made between the following types of time synchronization:

- Cyclic time synchronization
  Specification of a cycle time for renewed synchronization (1..65535 min)
- Daily time synchronization
  Specification of a time-of-day at which the daily synchronization is performed

1.3.3.7 Version identifier

The LCom protocol contains a version identifier. This version identifier is used to check which function block versions are communicating with each another. If the versions are different, an alarm is issued by the **FBLComMachineCom** function block.

This allows future, possibly incompatible library versions to be detected.

1.3.3.8 Configuring the communication parameters

The following communication parameters can be defined:

- Type of data transmission (cyclic, at data change or single)
- Cycle time for the cyclic data transmission
- Time for timeout after which a data packet has to be acknowledged
- Value for the sliding window
- These values can be entered in the parameter structure both for the send and the receive channels. In this way, data exchange can be initiated in both transmission directions from one device.

Alignment of communication parameters

When a connection is established, data can be exchanged between both nodes. This is done independently in each data direction. At the start of data exchange, the communication parameters of both devices are aligned. On completion of this configuration phase, both nodes (receiver and sender) have the same connection parameters. The currently active settings are displayed in a diagnostic structure. The communication can be established from both sides.

The mechanism for alignment of the communication parameters is explained for one data direction below.

Assigning communication parameters to the peer device

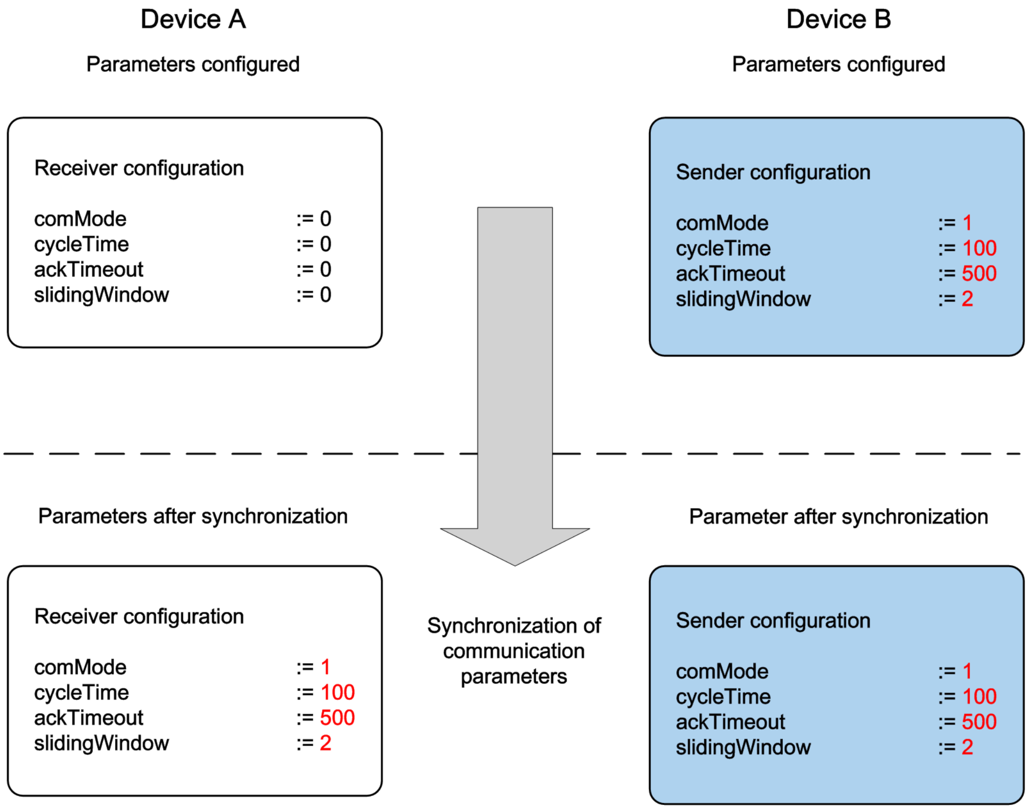


Figure 1–9 Assignment of the communication parameters from device B to device A

In the figure above, the communication parameters are set for device B only; they are assigned to device A on commencement of the communication relationship. After alignment, both devices have the same communication parameters for this transmission direction.

NOTE
To avoid errors, it is recommended that the communication parameters are assigned to only one device. The peer device adopts these settings when setting up the communication relationship. Because the setting up of the communication relationship (alignment of the communication parameters) has been optimized in the **LCom** library V1.1, the receiver settings should be set to 0 for mixed operation with different versions (less than V1.1).

Confirmation of the communication parameters by the peer device

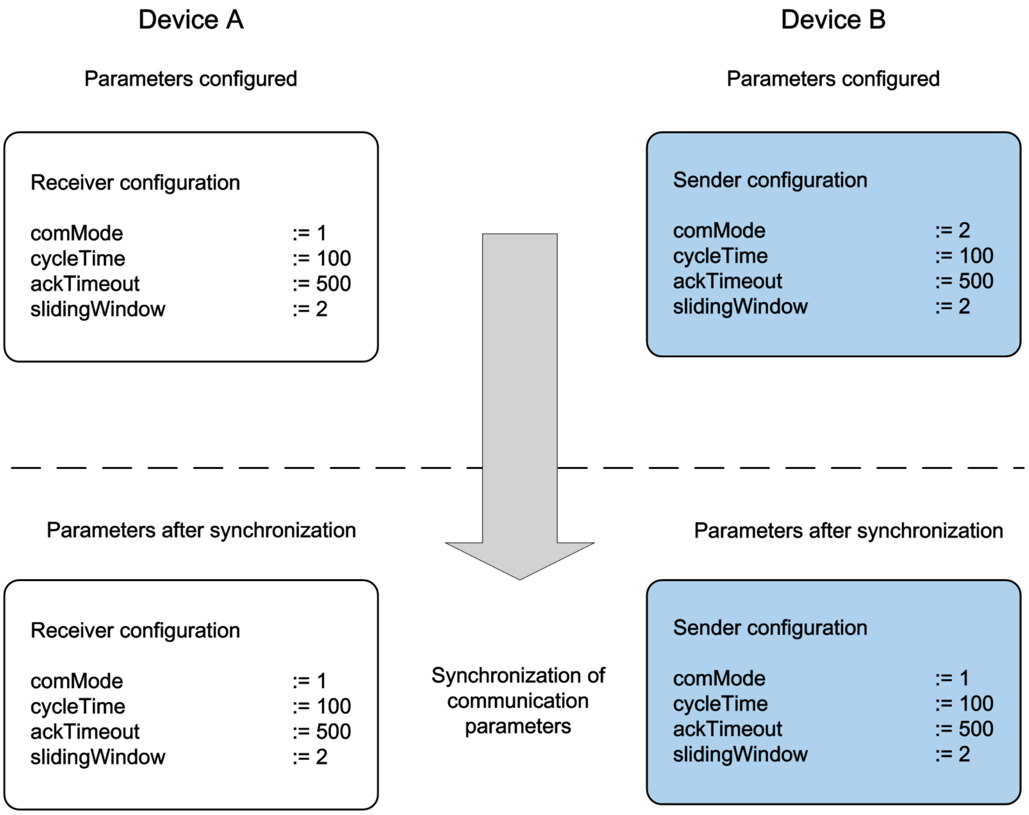


Figure 1–10 Confirmation of the parameters

The same communication parameters are set in both devices. When the connection is established by one device, no differences are detected and the settings remain unchanged.

Adapting the communication parameters in the case of differences

If the parameters are different on both sides, the values must be adapted. Adaptation is necessary, for example, when one node cannot send as frequently and is restricted by its settings.

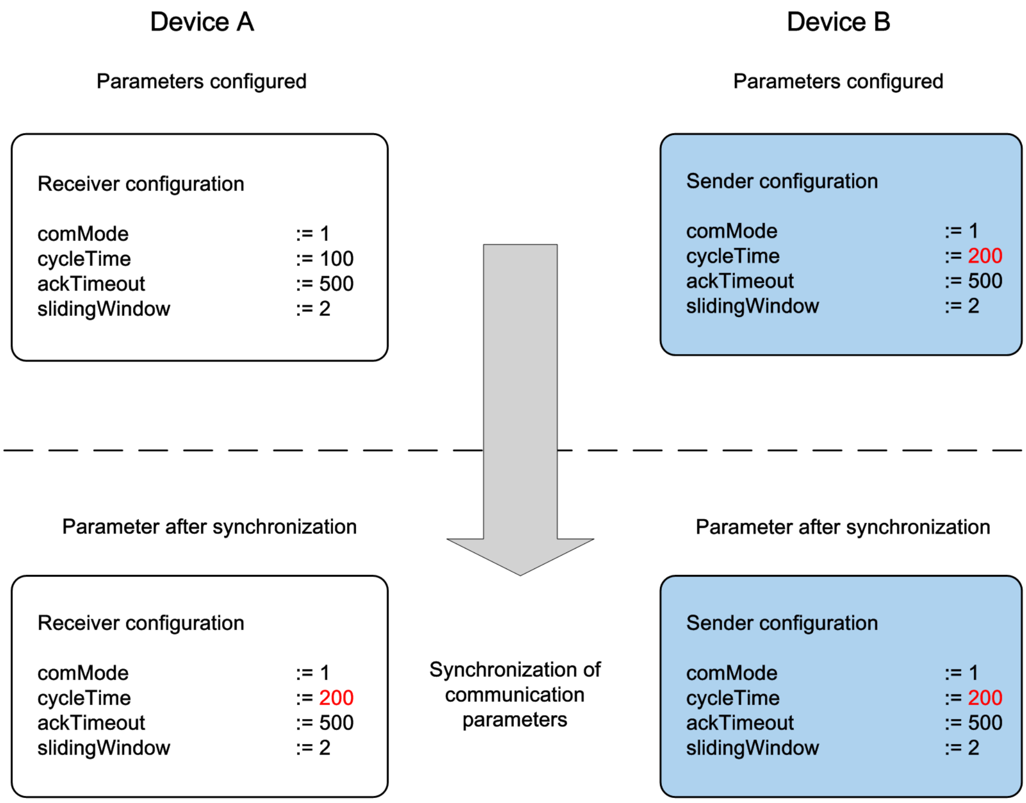


Figure 1–11 Adapting the communication parameters

The adaptation is carried out according to the following rules:

- The cycle time (*cycleTime*) and the monitoring time for the acknowledgement of the data packets (*ackTimeout*) are set to the larger of the two values.
- The sliding window (*slidingWindow*) is set to the lower of the two values.

In the example above, the cycle time of device B is longer than that of device A. The cycle time is set to the larger of the two values.

The rules for the various transmission modes are explained separately.

Rejecting the communication parameters

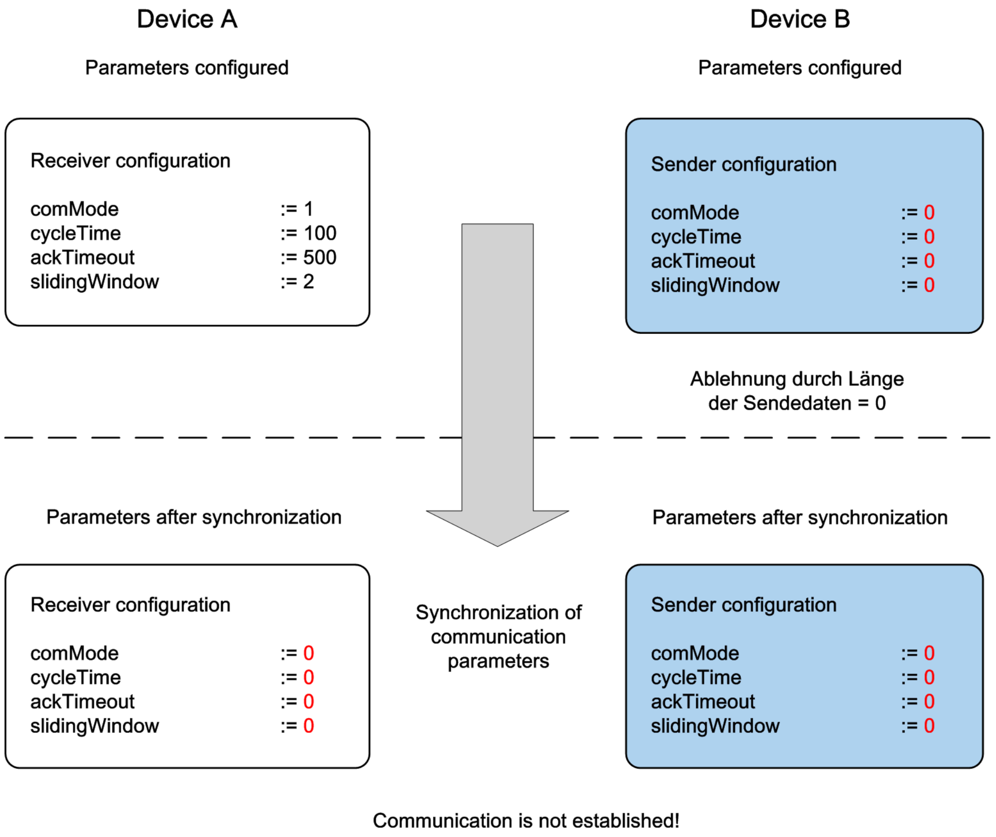


Figure 1–12 Rejecting the communication parameters

A node can also reject a communication request if no data is present for transmission (length of the send data = 0). Communication is not established.

Establishment of communication with changes on both sides

The parameters are adapted in accordance with the rules even when they have to be changed on both sides (e.g. the cycle time is longer on the receiver side and the monitoring time is longer on the sender side).

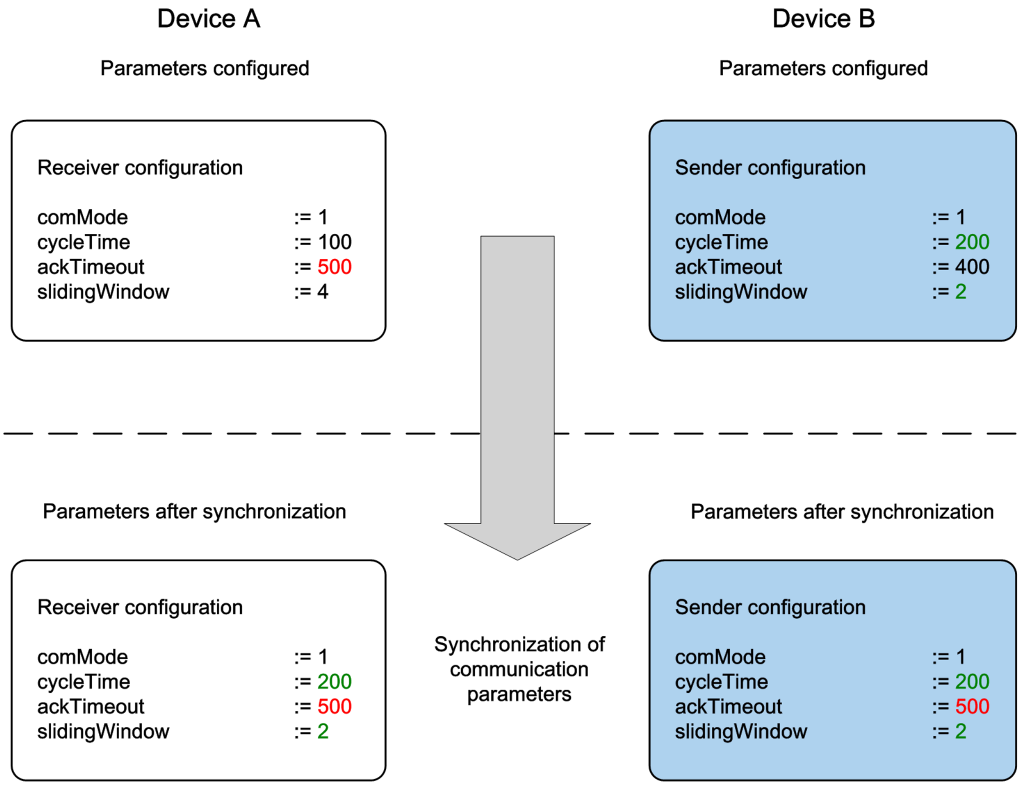


Figure 1–13 Changing the parameters of both communication peers

Parameterizing different transmission modes

If both nodes attempt to transmit simultaneously with different transmission modes, the transmission modes are prioritized. If the values in the other parameters are different, they are adapted as described above.

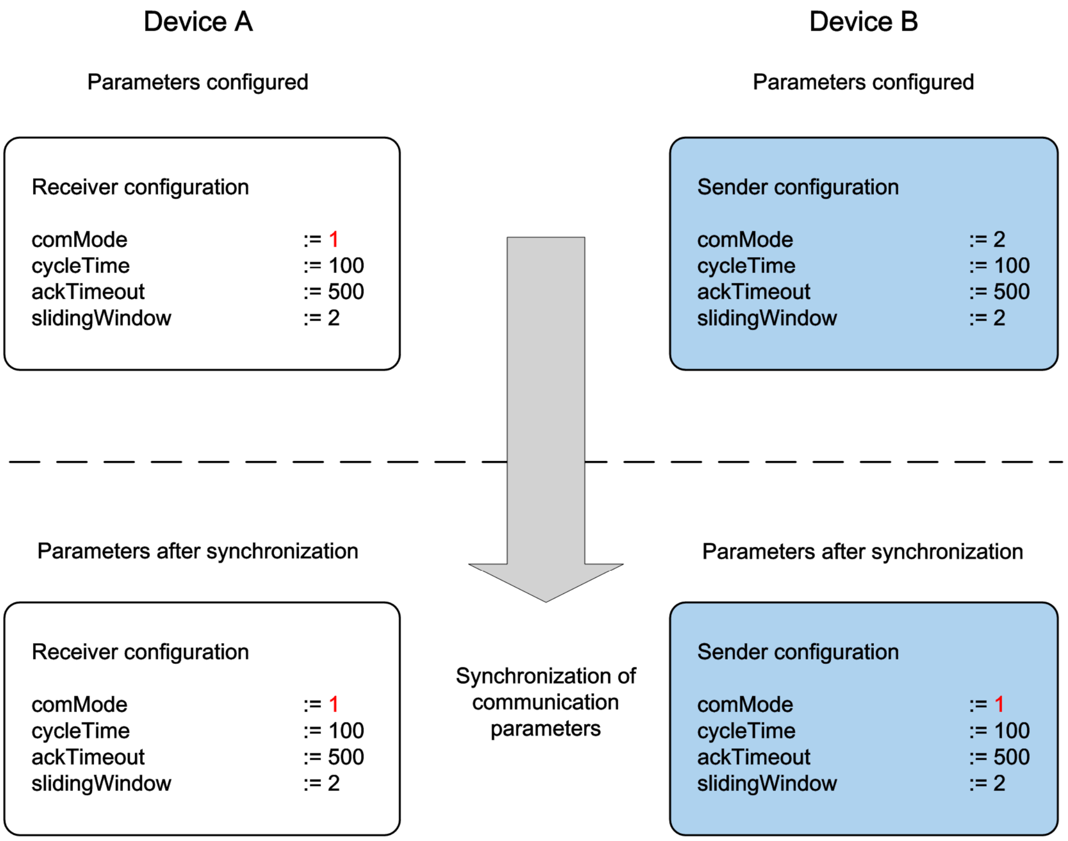


Figure 1–14 Prioritizing the transmission modes

The following priorities have been defined:

- Single transmission (3) - lowest priority
- Transmission on data change (2) - medium priority
- Cyclic transmission (1) - highest priority

Correspondingly, the transmission mode in the example above is set to cyclic (1).

NOTE
If the transmission modes are set differently on each side, the transmission mode with the highest priority is used. Data is exchanged.

Changing the configuration during operation

The communication data described above can also be changed while the connection is active. Therefore, it is possible to change cycle times, for example, during operation. The new values are entered in a diagnostic structure.

1.3.3.9 Establishment of communication

In order to establish the connection, you must configure which side is the client and which is the server. Both peers have equal priority when the connection is active. The data transmission can therefore be initiated by both peers.

1.3.3.10 Communication discontinuation

Discontinuation of communication can be initiated by both communication peers. The other side is informed of this discontinuation and the connection is closed by both peers. The status of the closed connection is indicated via an output variable.

A non-notified clearing of a connection produces an error message on the peer device.

1.3.4 TCP communication

Many systems offer functions for data exchange via TCP. The **LCom** library is available for SIMOTION and SIMATIC controllers with the **FBLComMachineCom** function block, which is also based on TCP communication. For this purpose, the communication blocks **FB63** to **FB66** are encapsulated in a function block. This especially facilitates their use in the LAD/FBD programming languages.

With pure TCP communication without LCom protocol, the **FBLComMachineCom** function block outputs the data that is currently in the TCP receive buffer. The user must carry out a function assignment of the data to form a coherent message in the context of a specific application. This behavior is described in Section General [page 11].

The additional functions described in Section Using the LCom protocol [page 14], such as

- Extended user data sizes
- Flow control and
- Configuration by both sides

cannot be used in pure TCP communication.

NOTE
Use of the LCom protocol is recommended. Otherwise, the user must be familiar with the supplementary conditions for the standard TCP communication.

1.4 Automation overview

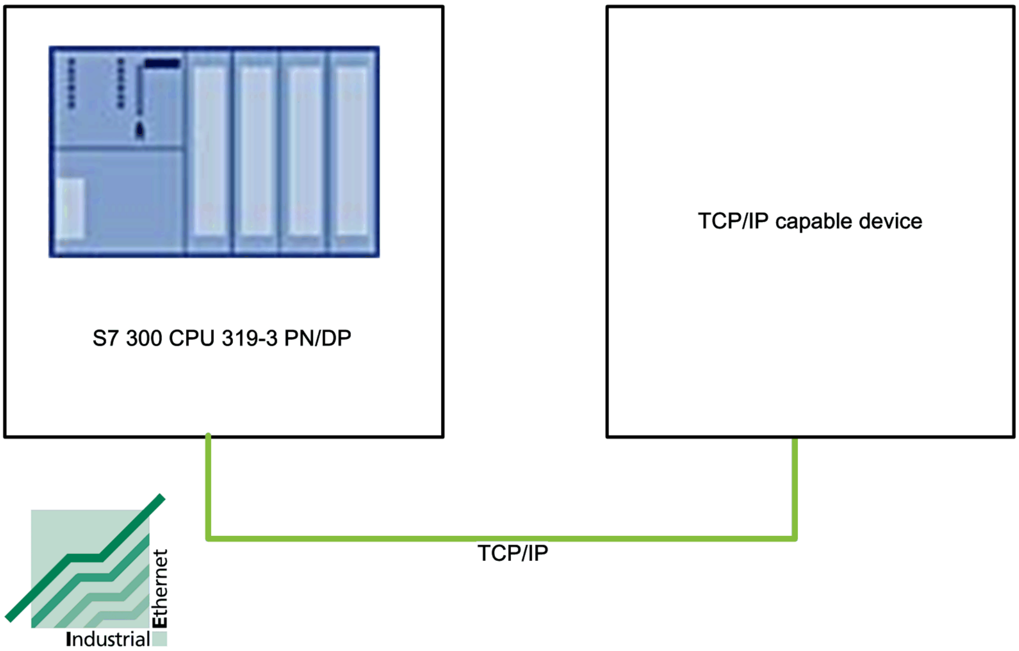


Figure 1–15 Connection example with a TCP/IP-capable device

The **FBLComMachineCom** function block sets up a point-to-point connection between a SIMATIC controller and a TCP/IP-capable device, e.g. another SIMATIC controller, a SIMOTION controller, or a PC.

Several point-to-point connections to different devices can be set up simultaneously via an Ethernet interface.

1.4.1 System requirements

Development tool

SIMATIC STEP 7 as of version V5.4 SP2

Controllers

Table 1–1 Controller types

Controller type	b8CpuType	MLFB
CPU 315(F)-2 PN/DP firmware >= 2.4	B#16#02	6ES7 315-2EH13-0AB0 6ES7 315-2EH14-0AB0 6ES7 315-2FH13-0AB0 6ES7 315-2FJ14-0AB0
CPU 317(F)-2 PN/DP firmware >= 2.4	B#16#02	6ES7 317-2EK13-0AB0 6ES7 317-2EK14-0AB0 6ES7 317-2FK13-0AB0 6ES7 317-2FK14-0AB0
CPU 319(F)-3 PN/DP firmware >= 2.4	B#16#03	6ES7 318-3EL00-0AB0 6ES7 318-3EL01-0AB0 6ES7 318-3FL00-0AB0 6ES7 318-3FL01-0AB0
CPU 412-2 PN firmware >= 5.1	B#16#05	6ES7 412-2EK06-0AB0
CPU 414(F)-3 PN/DP firmware >= 5.1	B#16#05	6ES7 414-3EM05-0AB0 6ES7 414-3EM06-0AB0 6ES7 414-3FM06-0AB0
CPU 416(F)-3 PN/DP firmware >= 5.1	B#16#05	6ES7 416-3ER05-0AB0 6ES7 416-3ES06-0AB0 6ES7 416-3FR05-0AB0 6ES7 416-3FS06-0AB0
IM151-8(F) PN/DP CPU firmware >= 2.4	B#16#01	6ES7 151-8AB00-0AB0 6ES7 151-8AB01-0AB0 6ES7 151-8FB00-0AB0 6ES7 151-8FB01-0AB0
WinAC RTX, IE interface on IF1	B#16#01	---
WinAC RTX, IE interface on IF2	B#16#06	---
WinAC RTX, IE interface on IF3	B#16#0B	---
WinAC RTX, IE interface on IF4	B#16#0F	---

NOTICE
The **FBLComMachineCom** function block can automatically identify all controller types with the MLFB from Table 1–1 Controller types [Page 28]. The controller type is required internally for the communication blocks. If the controller type (parameter *b8CpuType*) is not identified automatically (*enable* = TRUE or *b8CpuType* = 16#FF), this must be initialized by the user.

Comunications Processor (CP)

The function block **FBLComMachineCom** uses internal the function FC10 (AG_CNTRL, CMD 1 and 2) to check and reset the connection. The function FC10 is just supported by following CPs:

Table 1–1 CP types

CP type	MLFB	Firmware	b8CpuType
CP343-1 Lean	6GK7343-1CX10-0XE0	V2.1	B#16#00
CP343-1	6GK7343-1EX21-0XE0	V1.0.17	
	6GK7343-1EX30-0XE0	V2.0.16	
CP343-1 Advanced	6GK7343-1GX21-0XE0	V1.0.24	
	6GK7343-1GX30-0XE0	V1.0.23	
	6GK7343-1GX31-0XE0	V3.0	
CP443-1	6GK7443-1EX20-0XE0	V1.0.26	
	6GK7443-1EX30-0XE0	V3.0	
CP443-1 Advanced	6GK7443-1EX40-0XE0	V2.2.35	
	6GK7443-1EX41-0XE0	V1.0.24	
	6GK7443-1GX20-0XE0	V2.0	
	6GK7443-1GX30-0XE0	V3.0	

NOTICE
Please take care of following entry: http://support.automation.siemens.com/WW/view/en/33414377

NOTICE
With CP and WinAC RTX controllers, parameter *b8CpuType* must always be initialized by the user. The required value for parameter *b8CpuType* can be found in Table 1–1 Controller types [Page 28].

1.4.2 Scope of delivery

The **LCom** library is supplied for the TCP communication via Industrial Ethernet. This library contains the **FBLComMachineCom** function block as well as further data required for communication.

The **LCom** library is supplied as an archive.

Library structure 2

2.1 Library structure

2.1.1 Overview

The **LCom** library contains the following function blocks.

Table 2–1 Blocks in the LCom library

Block name	Number	Description
TSEND	FB63	FB for sending data via TCP
TRCV	FB64	FB for receiving data via TCP
TCON	FB65	FB for opening a TCP connection
TDISCON	FB66	FB for closing a TCP connection
FBLComMachineCom	FB105	FB for the data exchange via Ethernet
TCON_PAR	UDT65	Data structure for the communication blocks
UDTLComParameter	UDT110	Data structure for parameters of the **FBLComMachineCom** function block
UDTLComDiagnostics	UDT111	Data structure for diagnostics of the **FBLComMachineCom** function block
AG_CNTRL	FC10	Check and reset connection (CP)
AG_SEND	FC5	FC for sending data via TCP (CP)
AG_RECV	FC6	FC for receiving data via TCP (CP)
SET_CLK	SFC0	System function for setting the system time
BLKMOV	SFC20	System function for copying data storage areas
READ_CLK	SFC1	System function for reading the system time
RDSYSST	SFC51	System function for reading out the system status
TIME_TCK	SFC64	System function for reading out the system time

Communication blocks from the standard library

Parts of the communication blocks from the standard library for the TCP communication are contained in the **LCom** library. These blocks provide the basic functions for TCP communication.

FBLComMachineCom function block

The **FBLComMachineCom** function block is assigned the FB number 105. This FB is described in detail in Section 4 Function description [Page 45].

UDT

UDT65 TCON_PAR is used for the connection parameters.

Two created UDT are also available:

- UDT110, for the parameterization of the connection and the communication
- UDT111, for the diagnostics of the **FBLComMachineCom** function block

System functions

The system functions (SET_CLK, BLKMOV, READ_CLK, RDSYSST and TIME_TCK) required for the **FBLComMachineCom** function block are also contained in the library.

2.2 Core functions

2.2.1 Overview

The **FBLComMachineCom** function block is provided by the **LCom** library. This enables the TCP communication via Industrial Ethernet between SIMOTION and SIMATIC controllers (and CP) as well as other TCP-capable devices.

The LCom protocol was defined for additional communication functions, such as transmission of data records, flow control on the application level and the selection of various transmission modes.

The FB can have multiple instances, which allows several connections to one SIMATIC control.

Integration 3

3.1 Integration into the user program

3.1.1 Required communication blocks

The communication blocks **FB63** to **FB66** are required for the use of the **LCom** library with SIMATIC controllers with onboard Ethernet interface. **UDT65** is required for the transfer of the connection parameters to these function blocks.

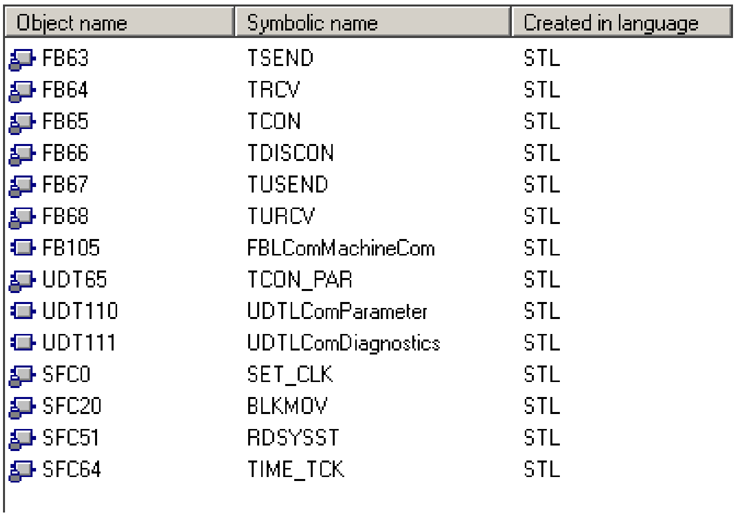
The functions **FC5**, **FC6** and **FC10** are required for CP. Whereby a distinction is made between CP343 and CP434.

3.1.2 Integration of the LCom library

The **LCom** library is supplied as an archive and must be de-archived via the SIMATIC Manager. All of the library objects must then be copied to the desired SIMATIC project.

For CP, the objects must be copied from the CP343 or CP434 folder depending on the type of CP.

It is irrelevant from which folder the objects are copied for a SIMATIC controller with onboard Ethernet interface.



Object name	Symbolic name	Created in language
FB63	TSEND	STL
FB64	TRCV	STL
FB65	TCON	STL
FB66	TDISCON	STL
FB67	TUSEND	STL
FB68	TURCV	STL
FB105	FBLComMachineCom	STL
UDT65	TCON_PAR	STL
UDT110	UDTLComParameter	STL
UDT111	UDTLComDiagnostics	STL
SFC0	SET_CLK	STL
SFC20	BLKMOV	STL
SFC51	RDSYSST	STL
SFC64	TIME_TCK	STL

Figure 3–1 Objects in the LCom library

The **FBLComMachineCom** function block can have multiple instances, which allows several simultaneously active connections.

Table 3–1 Maximum number of connections

Controller type	Maximum number of connections
CPU 315(F)-2 PN/DP	8
CPU 317(F)-2 PN/DP	16
CPU 319(F)-2 PN/DP	32
CPU 412-2 PN	46
CPU 414(F)-3 PN/DP	62
CPU 416(F)-3 PN/DP	94
IM151-8(F) PN/DP CPU	8

NOTE

The **FBLComMachineCom** function block must be called cyclically. It is recommended that a call be executed in **OB1**. For consistency reasons, it is recommended that send data is supplied and received data read in the same execution level in which the function block is called.

Transmitting the send and receive data

The **FBLComMachineCom** function block expects the send data in the form of an ANY pointer. This means that the storage location and the data length do not have to be defined at the time of configuration, but can instead be changed during operation.

The destination for the receive data is also transferred to the communication block via an ANY pointer. You must make sure that the data storage area indicated by the pointer is large enough for the receive data. This should be taken into account at the time of configuration.

NOTE

The send and receive data is accessed directly via the ANY pointer. This is why the user must ensure data consistency when processing the communication block. The **FBLComMachineCom** function block signals whether send data is currently being read or receive data being written via output variables (*sending* and *receiving*).

NOTE

If the LCom protocol is not used, the size of the byte array for the receive buffer must be set to 4096 bytes.

3.1.3 Call example in statement list and ladder logic (STL and LAD)

Description of the task

Two controllers (device A and device B) should be connected to each other. The connection should be made directly after power up of both controllers and must be maintained during operation. 1000 bytes should be sent cyclically every 800 ms from device A to device B. From device B, 300 bytes should be sent cyclically every 500 ms to device A. Port 4000 should be used on both controllers. Device A should be parameterized as the client, device B as the server.

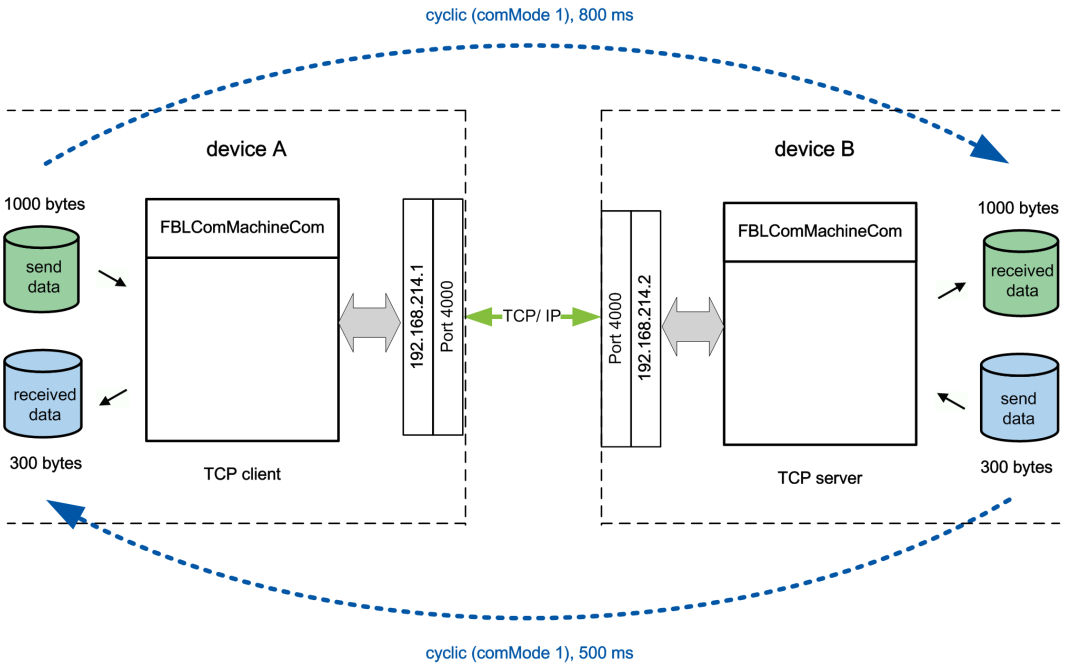


Figure 3–2 Communication relationship between two devices for FBD call example

Device A

The program shown is called cyclically in OB1.

Table 3–2 Assignment of the connection parameters to device A in STL

```
      A     #boInitDone
      JC    IEND
//do initialization only in first cycle
      SET
      S     #boInitDone

//connection configuration
      S     "DBLComParameter".parameter[0].sCfgConnection.boWithLComProtocol
      S     "DBLComParameter".parameter[0].sCfgConnection.boIsTcpClient

      L     W#16#FF    //W#16#FF: identify CPU type via SZL, W#16#0: CP
      T     "DBLComParameter".parameter[0].sCfgConnection.b8CpuType

      L     W#16#FA0    //4000
      T     "DBLComParameter".parameter[0].sCfgConnection.b16LocalPort
      L     W#16#FA0    //4000
      T     "DBLComParameter".parameter[0].sCfgConnection.b16RemotePort

      L     W#16#C0     //192
      T     "DBLComParameter".parameter[0].sCfgConnection.ab8RemoteAddress[0]
      L     W#16#A8     //168
      T     "DBLComParameter".parameter[0].sCfgConnection.ab8RemoteAddress[1]
      L     W#16#D6     //214
      T     "DBLComParameter".parameter[0].sCfgConnection.ab8RemoteAddress[2]
      L     W#16#2      //2
      T     "DBLComParameter".parameter[0].sCfgConnection.ab8RemoteAddress[3]

      L     W#16#1      //1
      T     "DBLComParameter".parameter[0].sCfgConnection.b16ConnectionId

      L     W#16#3E8   //1000 ms (1..60000 ms)
      T     "DBLComParameter".parameter[0].sCfgConnection.b16LifeSignCycle

//sender parameter
      L     W#16#1     //0: inactive, 1: cyclic, 2: on change, 3: once
      T     "DBLComParameter".parameter[0].sCfgSender.b8ComMode
      L     W#16#1     //max. number of message frames with summary confirmation
      T     "DBLComParameter".parameter[0].sCfgSender.b8SlidingWindow
      L     W#16#320   //800 ms
      T     "DBLComParameter".parameter[0].sCfgSender.b16CycleTime
      L     W#16#3E8   //1000 ms
      T     "DBLComParameter".parameter[0].sCfgSender.b16AckTimeout

IEND: NOP   0
```

The parameters for the connection are entered in the STL programming language. The network above is processed in OB1. Through the query of a previous initialization (local variable *boInitDone*), this assignment is only processed during power up of the controller.

As device A is the TCP client, the peer IP address and the peer port must be specified and the bit for *boIsTcpClient* set here.

The communication parameters are defined for the send direction, i.e. from device A to device B. This data is entered in device B for the transmission direction from device B to device A, so that parameters are assigned for the send direction in both devices. If the LCom protocol is used, parameters can also be assigned for communication in both transmission directions.

Two variables are defined and set to TRUE for the control variables *enable* and *communicate* of the FB. As a result, once the controller has powered up an attempt is made to establish the connection (*enable* = TRUE) and, once the connection is open, data transmission is started (*communicate* = TRUE).

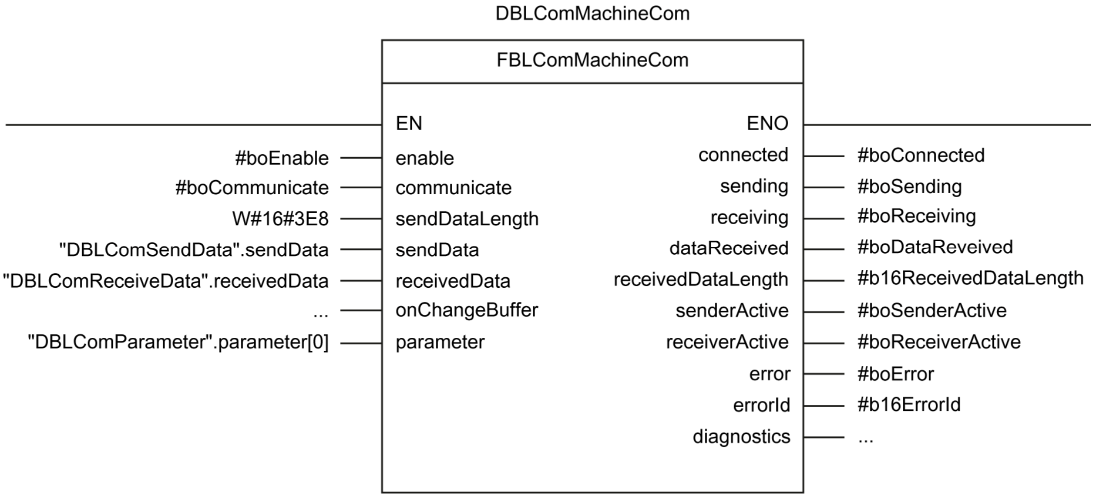


Figure 3–3 Call of the FBLComMachineCom function block in device A

In this example, the **FBLComMachineCom** function block for communication is called in the LAD programming language in **OB1**. Six data blocks are used:

- DBLComMachineCom
  Instance data block of the **FBLComMachineCom** function block
- DBLComSendData
  Data block with the data to be sent
- DBLComReceiveData
  Data block for the data to be received
- DBLComOnChangeBuffer
  Data block for comparison of changes, only relevant for transmission mode **On data change** and a data length to be sent greater than 4096 bytes
- DBLComParameter
  Data block for connection and communication parameters
- DBLComDiagnostics
  Data block for saving diagnostics data
- The length of the data to be sent - 1000 bytes (16#3E8) in this example - is specified via the *sendDataLength* input.

Device B

The program shown is called cyclically in OB1.

Table 3–3 Assignment of connection parameters to device B

```
      A     #boInitDone
      JC    IEND
//do initialization only in first cycle
      SET
      S     #boInitDone

//connection configuration
      S     "DBLComParameter".parameter[0].sCfgConnection.boWithLComProtocol
      R     "DBLComParameter".parameter[0].sCfgConnection.boIsTcpClient

      L     W#16#FF    //W#16#FF: identify CPU type via SZL, W#16#0: CP
      T     "DBLComParameter".parameter[0].sCfgConnection.b8CpuType

      L     W#16#FA0   //4000
      T     "DBLComParameter".parameter[0].sCfgConnection.b16LocalPort
      L     W#16#FA0   //4000
      T     "DBLComParameter".parameter[0].sCfgConnection.b16RemotePort

      L     W#16#C0    //192
      T     "DBLComParameter".parameter[0].sCfgConnection.ab8RemoteAddress[0]
      L     W#16#A8    //168
      T     "DBLComParameter".parameter[0].sCfgConnection.ab8RemoteAddress[1]
      L     W#16#D6    //214
      T     "DBLComParameter".parameter[0].sCfgConnection.ab8RemoteAddress[2]
      L     W#16#1     //1
      T     "DBLComParameter".parameter[0].sCfgConnection.ab8RemoteAddress[3]

      L     W#16#2     //2
      T     "DBLComParameter".parameter[0].sCfgConnection.b16ConnectionId

      L     W#16#3E8   //1000 ms (1..60000 ms)
      T     "DBLComParameter".parameter[0].sCfgConnection.b16LifeSignCycle

//sender parameter
      L     W#16#1     //0: inactive, 1: cyclic, 2: on change, 3: once
      T     "DBLComParameter".parameter[0].sCfgSender.b8ComMode
      L     W#16#1     //Max. number of message frames with summary confirmation
      T     "DBLComParameter".parameter[0].sCfgSender.b8SlidingWindow
      L     W#16#1F4   //500 ms
      T     "DBLComParameter".parameter[0].sCfgSender.b16CycleTime
      L     W#16#3E8   //1000 ms
      T     "DBLComParameter".parameter[0].sCfgSender.b16AckTimeout

IEND: NOP   0
```

The parameters for the connection are entered in the STL programming language. The network above is processed in OB1. Through the query of a previous initialization (local variable *boInitDone*), this assignment is only processed during power up of the controller.

As device B is the TCP server, only the local port needs to be specified here.

The communication parameters are defined for the send direction, i.e. from device B to device A. This data is entered in device A for the transmission direction from device A to device B, so that parameters are assigned for the send direction in both devices.

Two variables are defined and set to TRUE for the control variables *enable* and *communicate* of the FB. As a result, once the controller has powered up an attempt is made to establish the connection (*enable* = TRUE) and, once the connection is open, data transmission is started (*communicate* = TRUE).

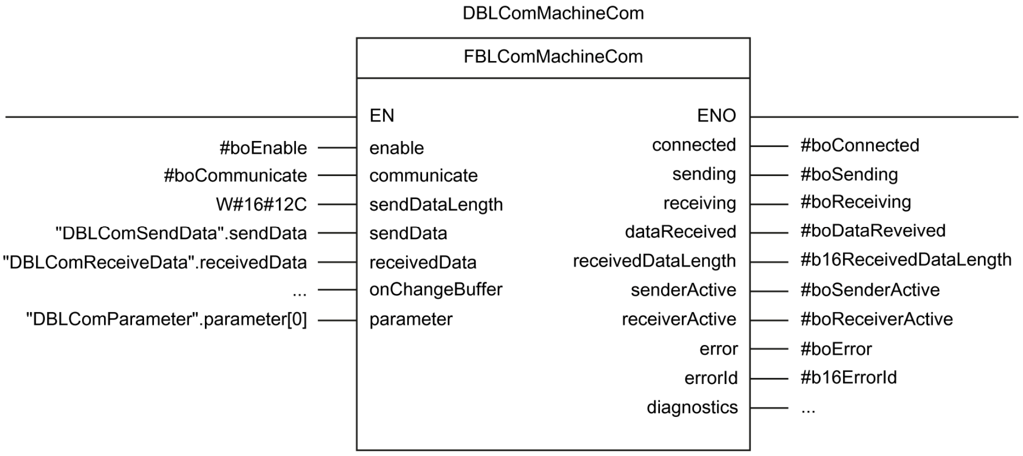


Figure 3–4 Call of the FBLComMachineCom function block in device B

The **FBLComMachineCom** function block is called in the LAD programming language in **OB1**. Six data blocks are accessed in the example shown:

- DBLComMachineCom
  Instance data block of the **FBLComMachineCom** function block
- DBLComSendData
  Data block with the data to be sent
- DBLComReceiveData
  Data block for the data to be received
- DBLComOnChangeBuffer
  Data block for comparison of changes, only relevant for transmission mode **On data change** and a data length to be sent greater than 4096 bytes
- DBLComParameter
  Data block for connection and communication parameters
- DBLComDiagnostics
  Data block for saving diagnostics data
- The length of the data to be sent - 300 bytes (16#12C) in this example - is specified via the *sendDataLength* input.

NOTICE

The connection parameters should not be initialized in each cycle but only, for example, after startup of the controller.
To interrupt the connection or data exchange and restart, the *enable* and *communicate* input parameters must be set to FALSE. The connection is then also cleared on the peer device.

In the call example, these parameters were preassigned the value TRUE for clarity.

3.1.4 Special features when using a CP

NOTICE

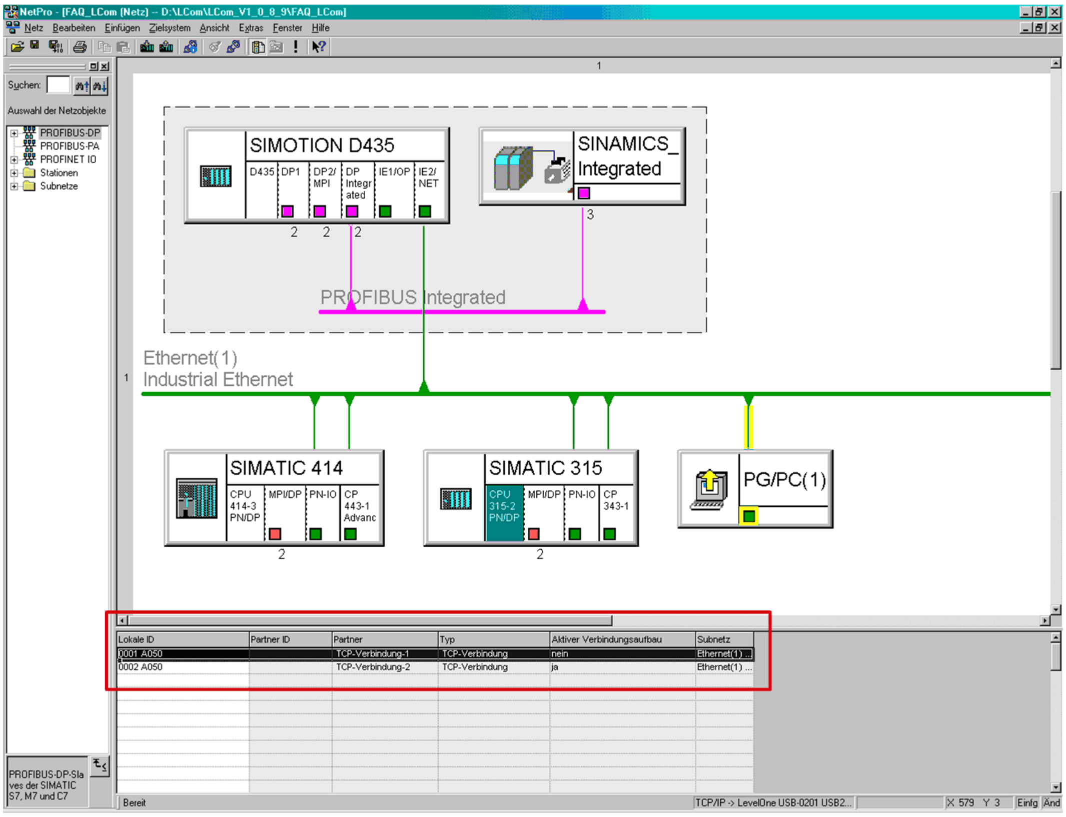
LCom supports CPs, which are listed in chapter 1.4.1 System requirements!

3.1.4.1 Connection configuration in NetPro

Creating a connection

When using a CP, an unspecified connection to the communication peer must be configured in NetPro. This configuration corresponds to the *sLComConnectionType* connection parameters.

This configuration is not required in NetPro for TCP communication via the onboard Ethernet interfaces.



Lokale ID	Partner ID	Partner	Typ	Aktiver Verbindungsaufbau	Subnetz
0001 A050		TCP-Verbindung-1	TCP-Verbindung	nein	Ethernet(1) ...
0002 A050		TCP-Verbindung-2	TCP-Verbindung	ja	Ethernet(1) ...

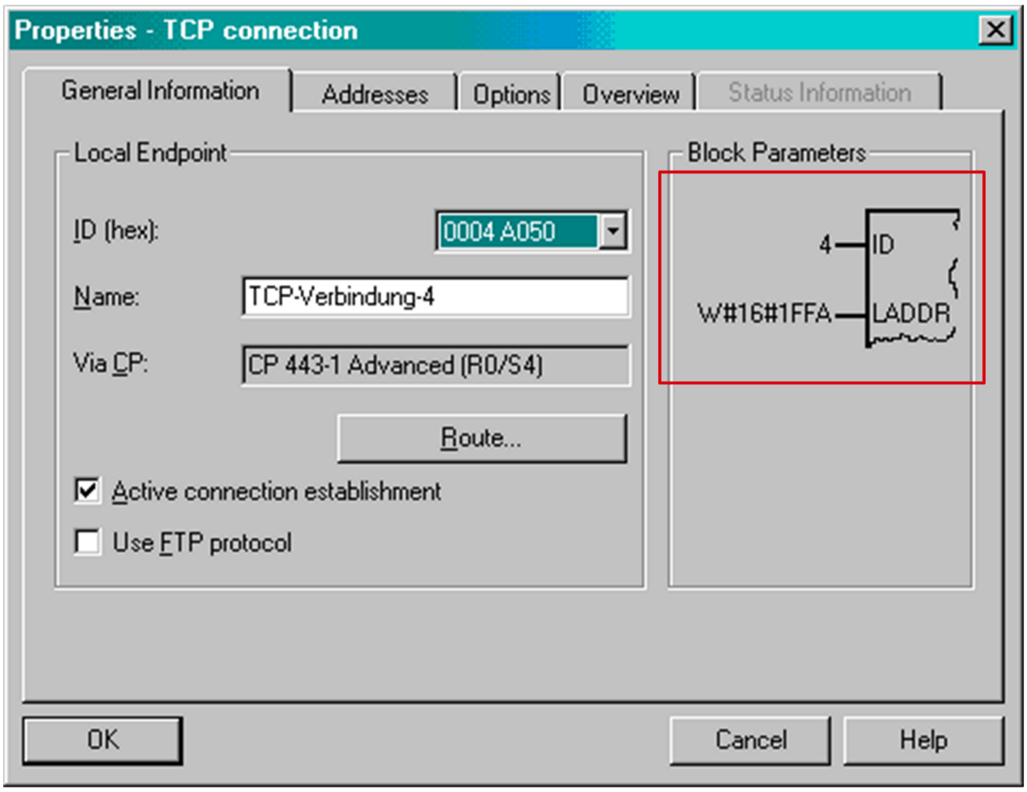
Figure 3–5 Inserting the CP connection configuration in NetPro

General information

The name of the connection can be defined in the **General information** tab, see Figure 3–6 Inserting properties of the TCP connection [Page 41].

The CP is also configured as server or client here. As client, the **Active connection establishment** checkbox must be activated.

The **Use FTP protocol** checkbox must be inactive.



ID (hex): 0004 A050

Name: TCP-Verbindung-4

Via CP: CP 443-1 Advanced (R0/S4)

☑ Active connection establishment

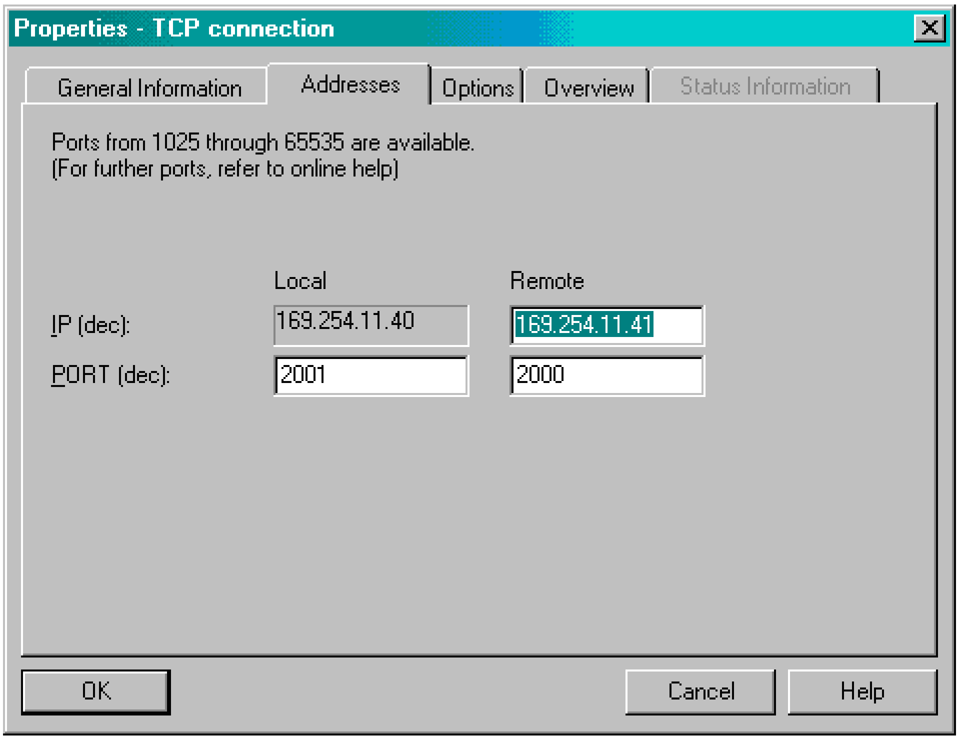
□ Use FTP protocol

Figure 3–6 Inserting properties of the TCP connection

The block parameters must be used for the parametrisation of the function block **FBLComMachineCom** (see chapter 3.1.4.2 Parameterization of the FBLComMachineCom function block).

Addresses

The IP addresses and port numbers are defined in the **Addresses** tab. It means that the remote IP address and port numbers must not be configured for the function block **FBLComMachineCom**.



Ports from 1025 through 65535 are available.
(For further ports, refer to online help)

	Local	Remote
IP (dec):	169.254.11.40	169.254.11.41
PORT (dec):	2001	2000

Figure 3–7 Configuration of the IP addresses and port numbers

3.1.4.2 Parameterization of the FBLComMachineCom function block

Table 3–4 Assignment of connection parameters when using a CP

```
      A     #boInitDone
      JC    IEND
//do initialization only in first cycle
      SET
      S     #boInitDone

//connection configuration
      S     "DBLComParameter".parameter[0].sCfgConnection.boWithLComProtocol
      S     "DBLComParameter".parameter[0].sCfgConnection.boIsTcpClient

      L     W#16#0     //W#16#FF: identify CPU type via SZL, W#16#0: CP
      T     "DBLComParameter".parameter[0].sCfgConnection.b8CpuType

      L     W#16#1FFA  //LADDR of block parameters, logical address of CP
      T     "DBLComParameter".parameter[0].sCfgConnection.b16LocalPort
//    L     W#16#0     //not used, must be defined in NetPro
//    T     "DBLComParameter".parameter[0].sCfgConnection.b16RemotePort

//    L     W#16#0     //not used, must be defined in NetPro
//    T     DBLComParameter".parameter[0].sCfgConnection.ab8RemoteAddress[0]
//    L     W#16#0     //not used, must be defined in NetPro
//    T     DBLComParameter".parameter[0].sCfgConnection.ab8RemoteAddress[1]
//    L     W#16#0     //not used, must be defined in NetPro
//    T     DBLComParameter".parameter[0].sCfgConnection.ab8RemoteAddress[2]
//    L     W#16#0     //not used, must be defined in NetPro
//    T     DBLComParameter".parameter[0].sCfgConnection.ab8RemoteAddress[3]

      L     W#16#4     //ID of block parameters
      T     "DBLComParameter".parameter[0].sCfgConnection.b16ConnectionId

      L     W#16#3E8   //1000 ms (1..60000 ms)
      T     "DBLComParameter".parameter[0].sCfgConnection.b16LifeSignCycle

//sender parameter
      L     W#16#1     //0: inactive, 1: cyclic, 2: on change, 3: once
      T     "DBLComParameter".parameter[0].sCfgSender.b8ComMode
      L     W#16#1     //max. number of message frames with summary confirmation
      T     "DBLComParameter".parameter[0].sCfgSender.b8SlidingWindow
      L     W#16#1F4   //500 ms
      T     "DBLComParameter".parameter[0].sCfgSender.b16CycleTime
      L     W#16#3E8   //1000 ms
      T     "DBLComParameter".parameter[0].sCfgSender.b16AckTimeout

IEND: NOP   0
```

The following parameters must be transferred from the Figure 3–6 Inserting properties of the TCP connection [Page 41] to the*sLComConnectionType* structure.

- Active connection establishment *boIsTcpClient*
- LADDR *b16LocalPort*
  This parameter is used to transfer the logical address of the CP to the FBLComMachineCom function block.
- ID *b16ConnectionId* connection ID

Function description

4

4.1 FBLComMachineCom function block

4.1.1 Functionality

This function block is used for data exchange on the basis of the TCP protocol. A description of the extended communication functions can be found in Section 1.3 Concept [Page 11].

4.1.2 Schematic LAD representation

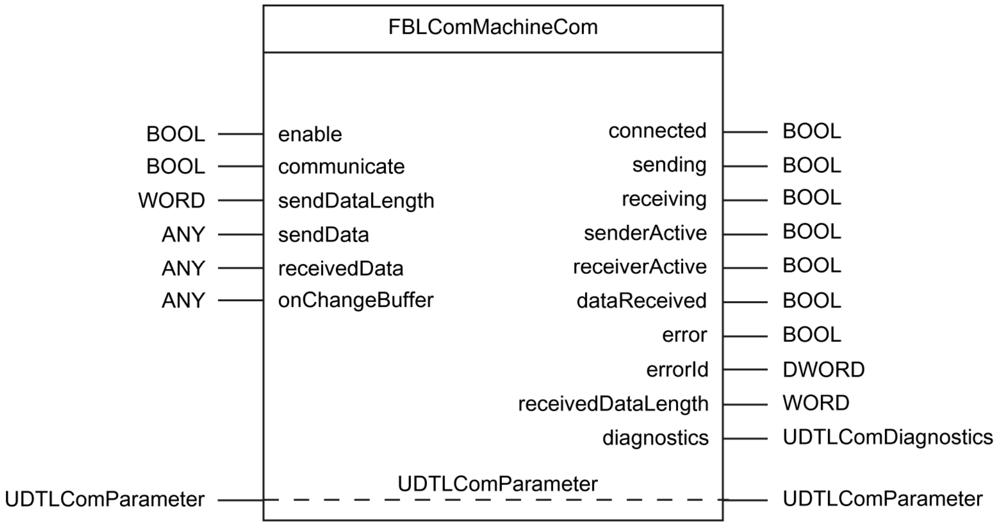


Figure 4–1 Schematic LAD representation of the FBLComMachineCom function block

With a rising edge of the Boolean *enable* input, an attempt is made to establish a connection or a server is opened. A connection is cleared with a falling edge of *enable*.

The *connected* output parameter is set when a connection to the connection peer exists on the application level.

When a connection exists, a rising edge of *communicate* starts the parameterized data exchange; the falling edge stops it.

NOTE

The positive edge at the *communicate* input can be used to start data transmission from a device in both directions, provided that the appropriate communication parameters have been set (see Section Configuring the communication parameters [page 19]). A negative edge at *communicate* stops communication in both directions. To stop transmission in one direction, the corresponding transmission mode must be set to inactive (0).

The in/out parameter *parameter* provides the block with all the information required to establish a connection. At the *diagnostic* output parameter, you can poll detailed information about the current connection or errors that have occurred. The individual elements of the created structures are described in a separate Section (Overview of the structures [page 66]).

The *sendDataLength* input parameter notifies the FB of how much data in the send buffer (*sendData* input parameter) is ready for transmission.
Received data is written to the output buffer (*receivedData* input parameter) and the length of the received data output via the *receivedDataLength* output parameter.

In order to avoid inconsistent access to send data (writing on data that is currently being sent) and receive data (reading data even though the entire message has not yet been transmitted), the Boolean output variable *sending* indicates that a send operation is active and the Boolean output variable *receiving* indicates that a receive operation is active.

The data blocks for send and receive data must not be accessed during sending and receiving.

The *dataReceived* output parameter is set for one cycle to signal the arrival of new data.

NOTE

The *onChangeBuffer* input only has to be connected when the send data is over 4096 bytes long (only relevant when the LCom protocol is being used). This buffer is required in order to perform a comparison with previous data in the "on data change" transmission mode. The size of this data buffer must be at least as large as the size of the send data.

4.1.3 Input and output parameters

The **FBLComMachineCom** function block has the following input and output parameters:

Table 4–1 Parameters of the FBLComMachineCom function block

Name	Type [1)]	Data type	Description
enable	IN	BOOL	TRUE: Activates processing and establishes a connection FALSE: Clears the connection
communicate	IN	BOOL	TRUE: Activates the configured data exchange FALSE: Stops the configured data exchange
sendDataLength	IN	WORD	Data length to be sent, maximum 65534 bytes
sendData	IN	ANY	Pointer to send data
receivedData	IN	ANY	Pointer to receive data
onChangeBuffer	IN	ANY	Pointer to data storage area, only relevant when send data is over 4096 bytes, data storage area must be the same size as the send data
parameter	IN_OUT	UDTLCom Parameter	Parameter structure
connected	OUT	BOOL	TRUE: Connected to communication peer
sending	OUT	BOOL	TRUE: Data is being sent, send data should not be written
receiving	OUT	BOOL	TRUE: Data is being received, receive data should not be read
senderActive	OUT	BOOL	TRUE: Transmission mode (cycl., on change, once) of the sender is active
receiverActive	OUT	BOOL	TRUE: Transmission mode (cycl., on change, once) of the receiver is active (relevant only when LCom protocol is used)
dataReceived	OUT	BOOL	TRUE: New data has been received TRUE is present for one cycle
error	OUT	BOOL	TRUE: Error occurred

Name	Type [1)]	Data type	Description
errorId	OUT	DWORD	Error number
receivedDataLength	OUT	WORD	Received data length
diagnostic	OUT	UDTLCom Diagnostics	Diagnostic structure
1)Parameter data: IN = input parameter, OUT = output parameter, IN_OUT = in/out parameter			

4.1.4 Time sequence charts

If the *enable* input is set for both communication peers, a connection is established. The *connected* output variable is set to TRUE as response for an active connection.

To start data transmission using one of the three modes, the desired communication peers are entered into the parameter structure (*sConfigSender* or *sConfigReceiver*) and the *communicate* input is set. If the LCom protocol is used, the *sConfigSender* or *sConfigReceiver* parameters between both devices are aligned. The data is then transmitted in the appropriate mode. The data is transmitted immediately in the appropriate mode without the LCom protocol.

An active data exchange on the sender side is indicated by setting the *senderActive* output (only with LCom protocol); on the receiver side, by setting the *receiverActive* output. This applies to communication negotiated by both peers in one of the three transmission modes. Both outputs remain set even if there is a break in transmission (waiting for a new cycle to start, no data change).

If data is currently being sent or received, the *sending* or *receiving* variable is set.

Completed transmission of a message is signaled in the receiver by setting *dataReceived* . This output remains active for one cycle.

Resetting the transmission mode to the default value 0 stops data exchange; the connection remains active. This means that *senderActive* and *receiverActive* are reset; but *connected* retains the value TRUE.

Resetting the *enable* input closes the connection and the *connected* output is assigned the value FALSE.

The figures below show the time sequence charts for the three transmission modes with and without use of the LCom protocol.

4.1.4.1 Time sequence chart for cyclic data transmission with LCom protocol

In the figure below, data exchange is initiated by the sender. Similarly, data exchange can also be initiated by the receiver. The signal response is identical.

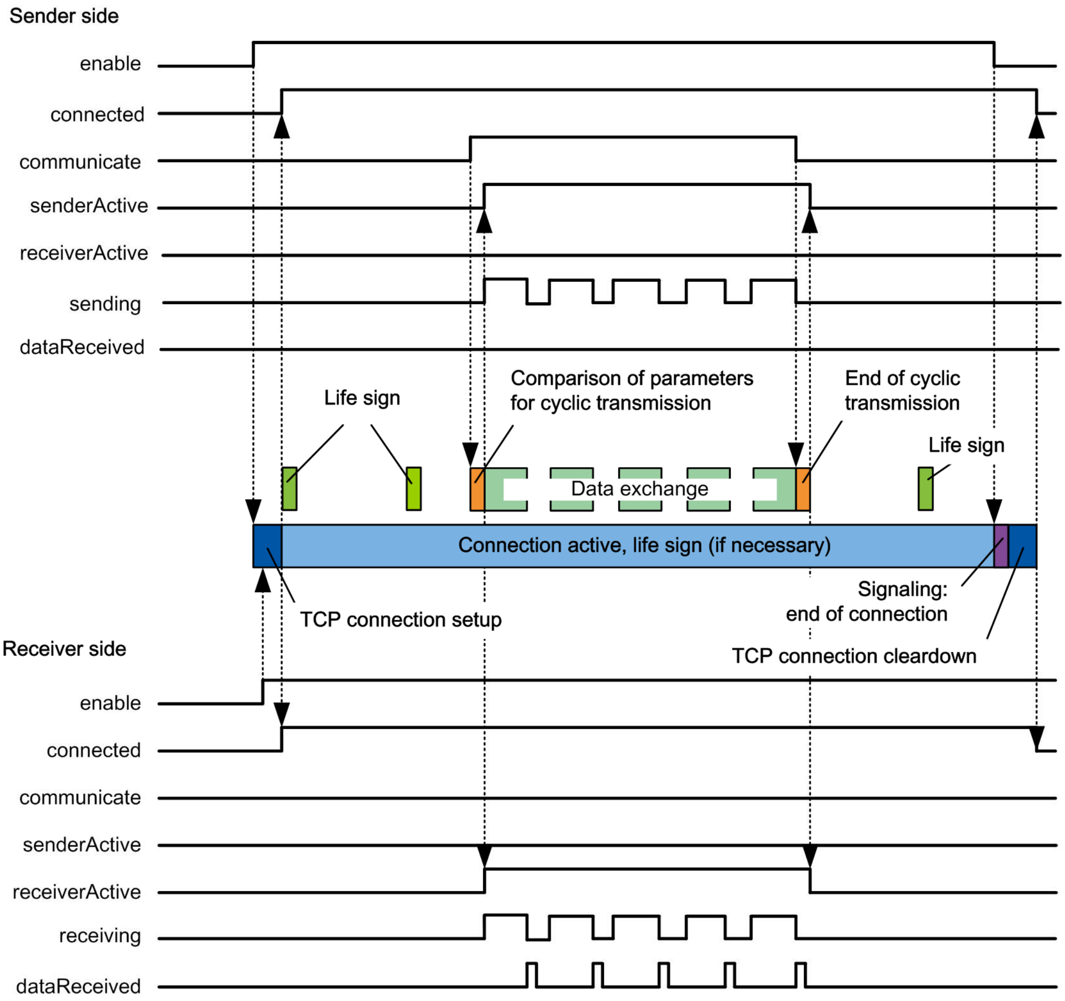


Figure 4–2 Time sequence chart for cyclic data transmission

4.1.4.2 Time sequence chart for data transmission on change with LCom protocol

In the figure below, data exchange is initiated by the sender. Similarly, data exchange can also be initiated by the receiver. The signal response is identical.

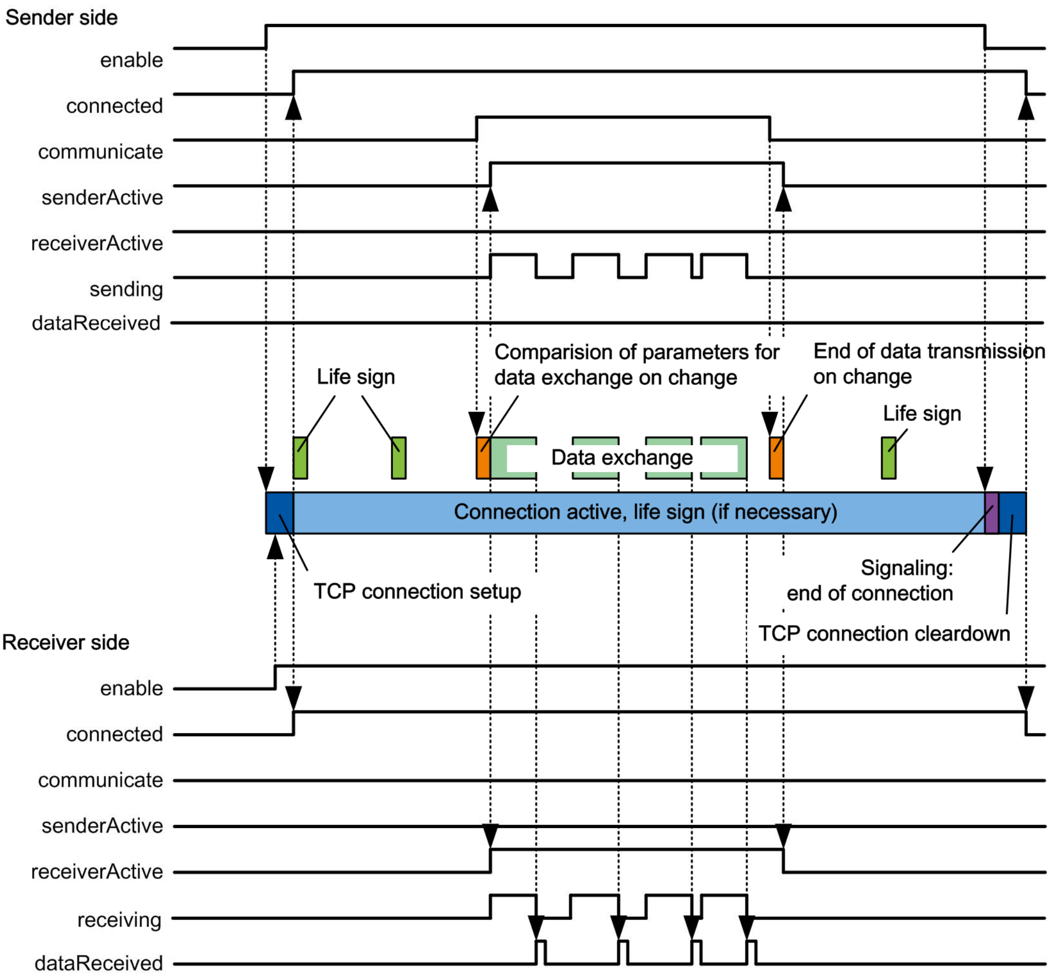


Figure 4–3 Time sequence chart for data transmission on change

4.1.4.3 Time sequence chart for single data transmission with LCom protocol

In the figure below, data exchange is initiated by the sender. Similarly, data exchange can also be initiated by the receiver. The signal response is identical.

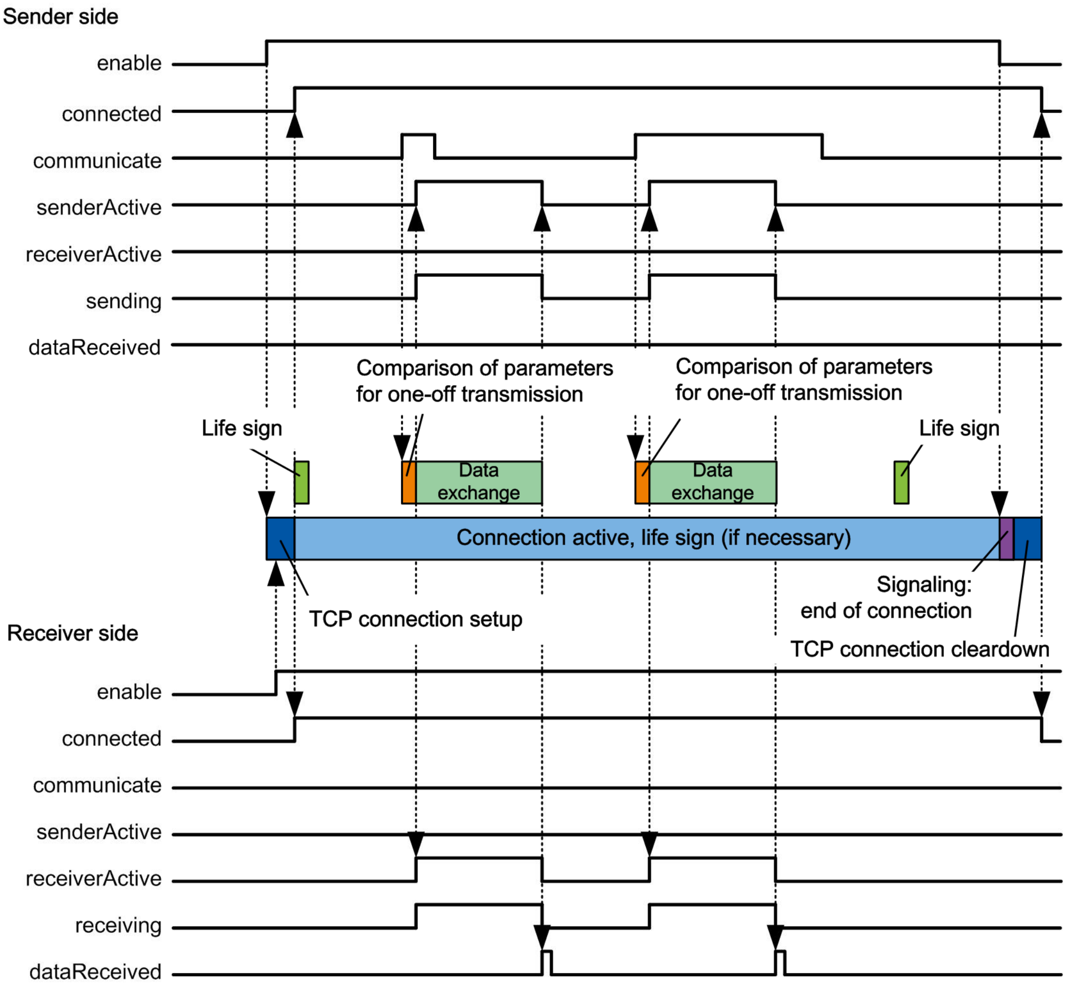


Figure 4–4 Time sequence chart for single data transmission

4.1.4.4 Time sequence chart for cyclic transmission without LCom protocol

In the figure below, data exchange is initiated by the sender. It is not possible to initiate data exchange from the receiver side without using the LCom protocol.

Without the LCom protocol, only contiguous data up to 4096 bytes in length can be sent.

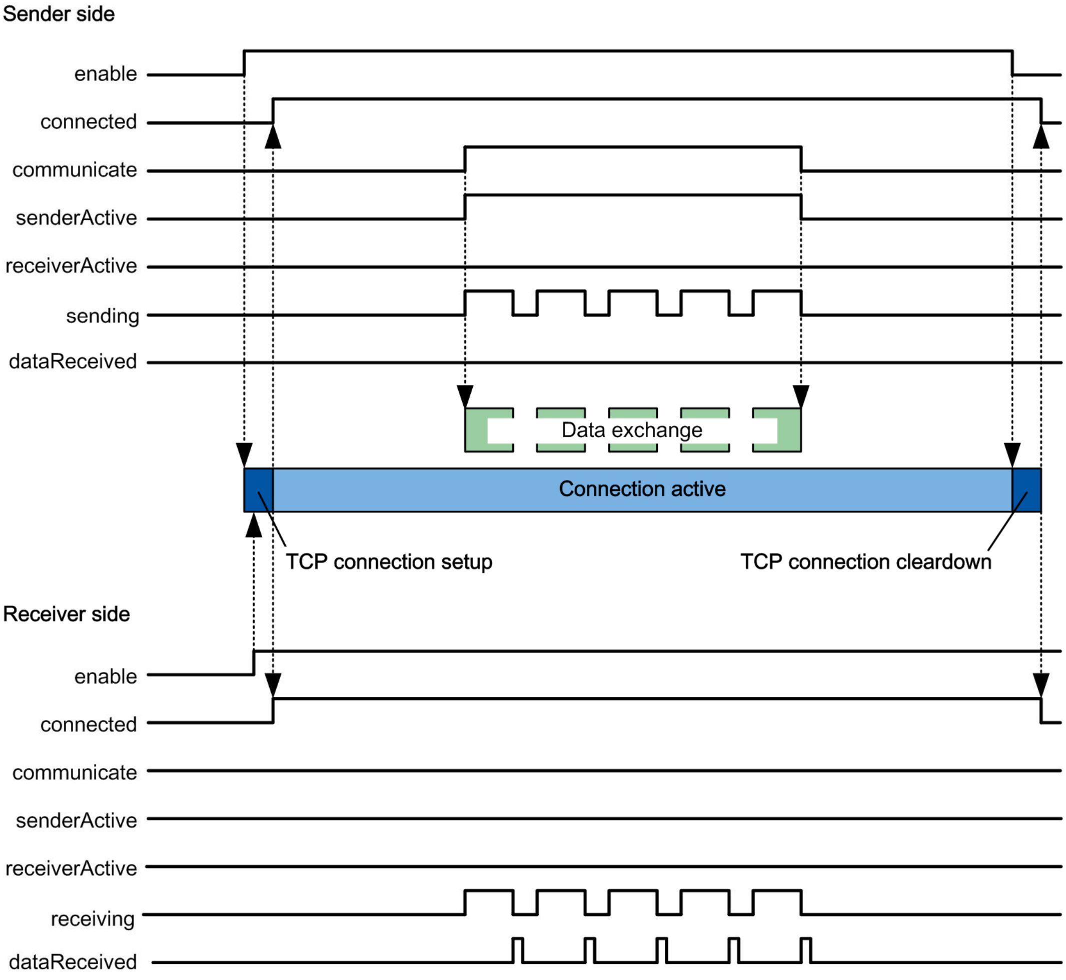


Figure 4–5 Time sequence chart for cyclic data transmission without using the LCom protocol

The *receiverActive* output is not set without using the LCom protocol, since the receiver does not know the set transmission mode of the sender. The receiver therefore does not know how long the data exchange will take.

The sent data may arrive at the receiver in several segments, but must be in the correct order. The user program must recombine these segments.

4.1.4.5 Time sequence chart for data transmission on change without LCom protocol

In the figure below, data exchange is initiated by the sender. It is not possible to initiate data exchange from the receiver side without using the LCom protocol.

Without the LCom protocol, only contiguous data up to 4096 bytes in length can be sent.

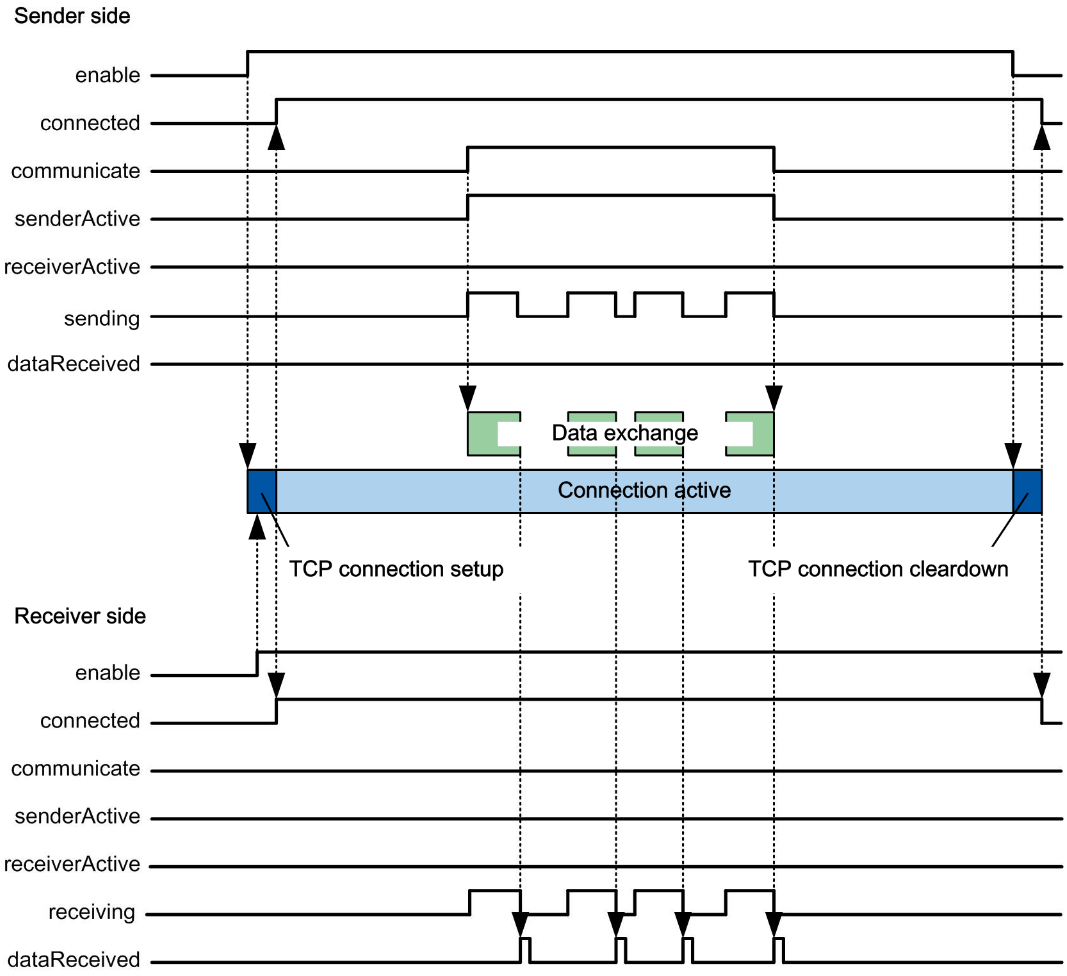


Figure 4–6 Time sequence chart for data transmission on change without using the LCom protocol

The *receiverActive* output is not set without using the LCom protocol, since the receiver does not know the set transmission mode of the sender. The receiver therefore does not know how long the data exchange will take.

The sent data may arrive at the receiver in several segments, but must be in the correct order. The user program must recombine these segments.

4.1.4.6 Time sequence chart for single data transmission without LCom protocol

In the figure below, data exchange is initiated by the sender. It is not possible to initiate data exchange from the receiver side without using the LCom protocol.

Without the LCom protocol, only contiguous data up to 4096 bytes in length can be sent.

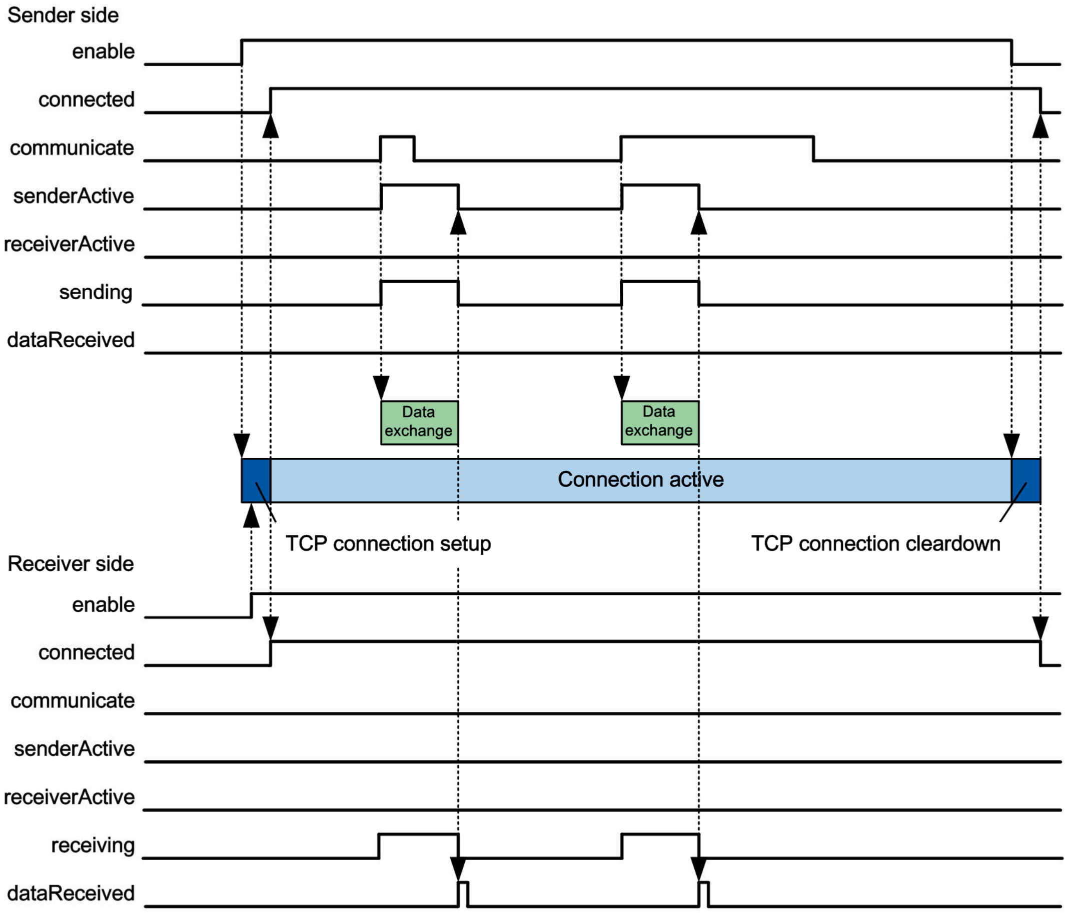


Figure 4–7 Time sequence chart for single data transmission without using the LCom protocol

The *receiverActive* output is not set without using the LCom protocol, since the receiver does not know the set transmission mode of the sender. The receiver therefore does not know how long the data exchange will take.

The sent data may arrive at the receiver in several segments, but must be in the correct order. The user program must recombine these segments.

4.1.5 Error messages

Indication of an error

If an error occurs, the Boolean output parameter *error* is set at the block and an error number output at the output parameter *errorId*. If the displayed error becomes inactive, the last still active error is output at the *errorID* output. A diagnostic structure contains the error history.

The error number is present at the output parameter until the error is reset or a more current error occurs (e.g. on re-establishment of connection).

A diagnostic structure can be evaluated for a detailed error description.

Diagnostic structure

The *diagnostic* output parameter includes a substructure called *asDiagnostics*. This is an array of the *sDiagnosticsType* structure. The array depth is predefined as 64 entries. This array operates as a ring buffer. Error messages that occur are stored in this buffer. The *i32DiagnosticIndex* variable points to the last (most recent) entry.

If an active error occurs again, this error is not written to the buffer again as an active error.

The structure consists of the following elements: The error number, date and time of the occurrence and the current status of the error. The status is always ACTIVE (1) when the error is entered into the buffer. When the error is reset, the status in the buffer is set to INACTIVE (0).
The structure also provides the return value of the system function and three additional values, which can contain detailed information, depending on the particular error. The meaning of the additional values for the corresponding error messages is explained in Section Error messages [page 55].

	Error code	Time of occurrence	Error status	Return value of the system function (if relevant)	Auxiliary value 1	Auxiliary value 2	Auxiliary value 3
0	2000	DT#2008-01-01-0:0:0	INACTIVE	0	1	0	0
1	4003	DT#2008-01-01-1:0:0	ACTIVE	FFFF8FFC	0	0	0
2	4001	DT#2008-01-01-2:0:0	ACTIVE	FFFFFFFD	0	0	0
63	0	DT#0001-01-01-0:0:0	INACTIVE	0	0	0	0

Index to recent error
(i32DiagnosticIndex)

Figure 4–8 Example of error messages in the *asDiagnostic* diagnostic structure

NOTE

The diagnostic buffer is not initialized with a rising edge of *enable*. The error history is therefore retained for later diagnostics. In the event of a power failure, the data is lost for an S7-400 CPU without a backup battery.

Error handling and error acknowledgement

The response to errors and their acknowledgement is configured in the function block in such a way, that the exchange of data is continued without a new control signal once the cause of the error has been remedied. For example, communication should be established again automatically after interruption of the physical connection.

But it is also possible that errors occur, where the cause must be removed and the function block then started with a positive edge of the *enable* input in order to re-establish the connection. This can be necessary after incorrect parameter assignment.

Table 4–2 Response and acknowledgement in the case of an error

Error	Response	Acknowledgement
Error when calling a system function for TCP communication	Closing of the connection and automatic attempt to re-establish it	Reset of the error message when the system function causing the error has been executed without error
Incorrect communication parameters (transmission mode, cycle time, timeout for acknowledgement and sliding window)	Exchange of user data is not performed, the connection is maintained, data exchange starts automatically when the parameters are correct	Reset of the error message when the correct parameters have been entered
Incorrect connection parameters (IP address, port number, send data length too long, receive data length too long, etc.)	Connection is not established	Reset of the error message through falling edge of *enable*; correct parameters and rising edge of *enable* required for re-establishment of connection

Error when calling a communication block

If a communication block signals an error, the connection is closed and a new attempt made to establish it. The corresponding error is reset when the communication block that caused the error is called without error. Several errors can occur simultaneously should a connection fail (error when sending, error when receiving, error when opening the connection, etc.). The most recent error is written to *errorId*. The other errors are sorted in the diagnostic structure according to the time of occurrence and marked with the status ACTIVE.

Example of error handling for communication blocks

If the Ethernet cable is disconnected during data transmission between the devices, the error message about the failure to send data is output. The connection is closed and the function block then attempts to re-establish it. This is not possible because the cable has been disconnected. The most recent error message is now the error when opening the connection. The relevant error code is output in the *errorId* output variable. The status of the error message for sending still remains ACTIVE in the diagnostic structure. The index of the ring buffer points to the most recent error, now the error when opening the connection.

When the Ethernet cable is reconnected, the connection is re-established. The error when opening the connection is reset in the diagnostic structure. The last active error is shown as the error code – Error when sending. When the data has been sent successfully, this error is also reset in the output variables and in the diagnostics structure.

Incorrect configuration of the communication parameters

The communication parameters include the transmission mode, the cycle time, the time for acknowledging the packets, as well as the value for the sliding window. These values are necessary for data exchange and are not checked until after the connection has been established. If this data is incorrect, data exchange will not take place. However, a TCP connection between the devices is active. After the values have been corrected, data transmission is carried out automatically and the error message is reset.

Incorrect configuration of the connection parameters

The connection parameters are only checked at a rising edge of the *enable* input. If errors are detected in the parameterization, a TCP connection is not established. The parameters must be changed to the correct values and a new rising edge applied to the *enable* input. The error message is removed at the falling edge of *enable* and set again in the case of further incorrect parameters with a rising edge.

The connection parameters include the IP address, port number and length of the send data (less than / equal to the send byte array), but also the information received about the size of the received data (less than / equal to the receive byte array).

Error number ranges

The error numbers used are divided into number ranges according to the cause of the error and listed below.

These error messages are described in the sections below.

4.1.5.1 Errors due to incorrect parameterization

Errors during the parameterization (2000h .. 2FFFh):

Table 4–3 Errors due to incorrect parameterization

errorId DWORD [Hex]	**Constant / meaning**
2000	LCOM_ERR_CFG_CONNECTION The parameterization in the *sCfgParameter* substructure is incorrect and could not be accepted by the FB. **Remedy:** The relevant parameter is corrected and the error reset with *enable* = FALSE.
	Additional information: dtActivationTime: Time of error occurrence eState: Error state: ACTIVE/INACTIVE b32SysRetVal: - r32AdditionalValue1: Indication of incorrect parameter: 1: Unknown service (*b16ComService*) 2: Invalid port (*b16LocalPort*) 3: Invalid target port (*b16RemotePort*) 4: Invalid target address (*ab8RemoteAddress*) (x.y.z.0 and x.y.z.255 are not permissible) 5: Invalid connection ID (*b16ConnectionId*) 6: Invalid sign-of-life cycle (*b16LifeSignCycle*) 7: Unknown CPU type (*b8CpuType*) r32AdditionalValue2: - r32AdditionalValue3: -
2001	LCOM_ERR_CFG_SENDER The parameterization in the *sCfgSender* substructure is incorrect and could not be accepted by the FB. **Remedy:** The relevant parameter is corrected and the error thereby acknowledged.
	Additional information: dtActivationTime: Time of error occurrence eState: Error state: ACTIVE/INACTIVE b32SysRetVal: - r32AdditionalValue1: Indication of incorrect parameter: 1: Transmission type (*b8ComMode*) 2: Cycle time (*b16CycleTime*) 3: Monitoring time (*b16AckTimeout*) 4: Transmission window (*b8SlidingWindow*) r32AdditionalValue2: - r32AdditionalValue3: -

errorId DWORD [Hex]	**Constant / meaning**
2002	LCOM_ERR_CFG_RECEIVER The parameterization in the *sCfgReceiver* substructure is incorrect and could not be accepted by the FB. **Remedy:** The relevant parameter is corrected and the error thereby acknowledged.
	Additional information: dtActivationTime: Time of error occurrence eState: Error state: ACTIVE/INACTIVE b32SysRetVal: - r32AdditionalValue1: Indication of incorrect parameter: 1: Transmission type (*b8ComMode*) 2: Cycle time (*b16CycleTime*) 3: Monitoring time (*b16AckTimeout*) 4: Transmission window (*b8SlidingWindow*) r32AdditionalValue2: - r32AdditionalValue3: -
2003	LCOM_ERR_CFG_TIME_SYNC The parameterization in the *sCfgTimeSync* substructure is incorrect and could not be accepted by the FB. **Remedy:** The relevant parameter is corrected and the error acknowledged with a rising edge at *enable*.
	Additional information: dtActivationTime: Time of error occurrence eState: Error state: ACTIVE/INACTIVE b32SysRetVal: - r32AdditionalValue1: Indication of incorrect parameter: 1: Synchronization type (*b8SendModeTimeSync*) invalid 2: Cycle time (*b16TimeSyncCycleTime*) must be greater 0 r32AdditionalValue2: - r32AdditionalValue3: -
2004	LCOM_ERR_BUFFERLENGTH The configured lengths of the send and receive buffers do not match the peer buffer lengths. The send buffer must be smaller than or the same size as the associated receive buffer. **Remedy:** The buffers are adapted and the error reset with *enable* = FALSE.
	Additional information: dtActivationTime: Time of error occurrence eState: Error state: ACTIVE/INACTIVE b32SysRetVal: - r32AdditionalValue1: Indication of incompatibility: 1: Send buffer with receive buffer of the peer 2: Receive buffer with send buffer of the peer r32AdditionalValue2: Own buffer length r32AdditionalValue3: Peer buffer length

errorId DWORD [Hex]	**Constant / meaning**
2005	LCOM_ERR_SENDDATALENGTH The configured send length (*sendDataLength*) is too long and is limited by the FB. The maximum possible data length is limited by the size of your own send buffer. When the LCom protocol is used, the size of the receive buffer on the peer side is also taken into account. Without LCom protocol, the selected transmission service (without LCom protocol = 4096 Bytes) also limits the maximum send length. The configured send length is reduced and data is sent with this maximum length. **Remedy**: Adapt the send length or the buffer lengths. The error is reset automatically.
	Additional information: dtActivationTime: Time of error occurrence eState: Error state: ACTIVE/INACTIVE b32SysRetVal: - r32AdditionalValue1: Limitation 1: Own send buffer 2: Receive buffer on peer side 3: TCP data length (for operation without LCom protocol) r32AdditionalValue2: Set send length (*sendDataLength*) r32AdditionalValue3: Limited/maximum possible send length

4.1.5.2 Errors during processing

Errors during internal processing, e.g. when calling a SIMATIC communication (4000h .. 4FFFh):

Table 4–4 Errors during processing

errorId DWORD [Hex]	**Constant / meaning**
4000	LCOM_ERR_WRONG_PARTNER An unknown peer has logged on to the server. The configuration does not permit this (*boAcceptUnknownPartner* = FALSE). The connection is closed by the server. **Remedy:** Check whether the peer connection parameters match or accept the unknown connection peer (*boAcceptUnknownPartner* = TRUE).
	Additional information: dtActivationTime: Time of error occurrence b16State: Error state: ACTIVE/INACTIVE r32AdditionalValue1: First and second bytes of the IP address of the unknown peer r32AdditionalValue2: Third and fourth bytes of the IP address of the unknown peer r32AdditionalValue3: Port of the unknown peer

errorId DWORD [Hex]	**Constant / meaning**
4001	LCOM_ERR_TCP_OPEN_SERVER An error occurred when using the **FB65** TCON function block (local controller is server). **Remedy**: The function block automatically attempts to open the server again. If successful, the error is acknowledged automatically.
	Additional information: dtActivationTime: Time of error occurrence b16State: Error state: ACTIVE/INACTIVE b32SysRetVal: Return value of function block
4002	LCOM_ERR_TCP_OPEN_CLIENT An error occurred when using the **FB65** TCON function block (local controller is client). **Remedy**: The function block automatically attempts to open the connection. If successful, the error is acknowledged automatically.
	Additional information: dtActivationTime: Time of error occurrence b16State: Error state: ACTIVE/INACTIVE b32SysRetVal: Return value of function block: 16#FFFF8FFA: No communication peer present or can be reached. Check the Ethernet cable and connection ID
4003	LCOM_ERR_TCP_SEND An error occurred when using the **FB63** TSEND function block. **Remedy**: The function block automatically attempts to send again. If successful, the error is acknowledged automatically. If fatal errors occur, the connection is closed and re-opened.
	Additional information: dtActivationTime: Time of error occurrence b16State: Error state: ACTIVE/INACTIVE b32SysRetVal: Return value of function block
4004	LCOM_ERR_TCP_RECEIVE An error occurred when using the **FB64** TRCV function block. **Remedy**: The function block automatically attempts to receive again. If successful, the error is acknowledged automatically. If fatal errors occur, the connection is closed and re-opened.
	Additional information: dtActivationTime: Time of error occurrence b16State: Error state: ACTIVE/INACTIVE b32SysRetVal: Return value of function block

errorId DWORD [Hex]	**Constant / meaning**
4005	LCOM_ERR_TCP_CLOSE_CONNECTION An error occurred when using the **FB66** TDISCON function block. **Remedy**: The function block automatically attempts to close the connection again.
	Additional information: dtActivationTime: Time of error occurrence b16State: Error state: ACTIVE/INACTIVE b32SysRetVal: Return value of the function block r32AdditionalValue1: - r32AdditionalValue2: - r32AdditionalValue3: -
4006	LCOM_ERR_HEADER_NOT_FOUND The packet header could not be identified. Either an invalid ID has been received, the peer is not using the LCom protocol, or data has been lost.
	Additional information: dtActivationTime: Time of error occurrence b16State: Error state: ACTIVE/INACTIVE r32AdditionalValue1: Received, incorrect packet identifier r32AdditionalValue2: - r32AdditionalValue3: -
4007	LCOM_ERR_RECEIVED_DATALENGTH Operation without LCom protocol: Received data exceeds the configured size of the receive buffer. The data is discarded. **Remedy**: Check whether the size of the receive buffer has been configured correctly.
	Additional information: dtActivationTime: Time of error occurrence b16State: Error state: ACTIVE/INACTIVE r32AdditionalValue1: Received data length r32AdditionalValue2: Configured reception buffer length r32additionalvalue3: -
4008	LCOM_ERR_TIMEOUT_LIFE_SIGN It has been too long since the sign-of-life was received. No connection closure was notified, however. With TCP, the connection is closed and re-established. If a new connection is then established, the error is acknowledged automatically. **Remedy**: The connection has been physically separated and must be established again. The peer device is in the CPU mode STOP and must be started.
	Additional information: dtActivationTime: Time of error occurrence b16State: Error state: ACTIVE/INACTIVE b32SysRetVal: - r32AdditionalValue1: Send cycle of the sign-of-life (by peer) in ms r32AdditionalValue2: Resulting timeout in ms r32AdditionalValue3: -

errorId DWORD [Hex]	**Constant / meaning**
4009	LCOM_ERR_RECEIVED_CFG_SENDER Invalid configuration message frame received for the sender. One or more parameters are incorrect.
	Additional information: dtActivationTime: Time of error occurrence b16State: Error state: ACTIVE/INACTIVE r32AdditionalValue1: Indication of incorrect parameter: 1: Transmission type (*b8ComMode*) 2: Cycle time (*b16CycleTime*) 3: Monitoring time (*b16AckTimeout*) 4: Transmission window (*b8SlidingWindow*) r32AdditionalValue2: Value of the incorrect parameter r32AdditionalValue3: State of communication establishment: -1: Inactive 1: Wait for sender configuration 2: Wait for receiver configuration 99: Error
400A	LCOM_ERR_RECEIVED_CFG_RECEIVER Invalid configuration message frame received for the receiver. One or more parameters are incorrect.
	Additional information: dtActivationTime: Time of error occurrence b16State: Error state: ACTIVE/INACTIVE r32AdditionalValue1: Indication of incorrect parameter: 1: Transmission type (*b8ComMode*) 2: Cycle time (*b16CycleTime*) 3: Monitoring time (*b16AckTimeout*) 4: Transmission window (*b8SlidingWindow*) r32AdditionalValue2: Value of the incorrect parameter r32AdditionalValue3: State of communication establishment: -1: Inactive 1: Wait for sender configuration 2: Wait for receiver configuration 99: Error
400B	LCOM_ERR_COM_DENIED_REMOTE_RECEIVER The receiver on the peer device has rejected communication establishment. The error is reset when the next communication is established successfully or when resetting the *enable* input.
	Additional information: dtActivationTime: Time of error occurrence b16State: Error state: ACTIVE/INACTIVE r32AdditionalValue1: - r32AdditionalValue2: - r32AdditionalValue3: -

errorId DWORD [Hex]	**Constant / meaning**
400C	LCOM_ERR_COM_DENIED_REMOTE_SENDER The sender on the peer device has rejected the communication establishment. It may be that there is no data to send (*datalength* = 0). **Remedy**: Data must be present for sending on the sender side (*sendDataLength* = 0). The error is reset the next time communication is established successfully or when the FB is reset (*enable* = FALSE).
	Additional information: dtActivationTime: Time of error occurrence b16State: Error state: ACTIVE/INACTIVE r32AdditionalValue1: - r32AdditionalValue2: - r32AdditionalValue3: -
400D	LCOM_ERR_SENDER_CYCLE_TIME Error during cyclic transmission: The specified send cycle (*b16CycleTime)* cannot be maintained because a previous transmission has not yet been completed. This can happen when the block is not called frequently enough or when the selected send cycle is too short. **Remedy**: Check whether the block is called frequently enough. This can be done by monitoring the mean call cycle (additional information). If necessary, the configured send cycle must be increased. The error is reset automatically the next time the cycle time is maintained.
	Additional information: dtActivationTime: Time of error occurrence b16State: Error state: ACTIVE/INACTIVE b32SysRetVal: - r32AdditionalValue1: Currently configured send cycle (*b16CycleTime*) in ms r32AdditionalValue2: Mean call cycle of the FB in ms r32AdditionalValue3: Reason on timeout: 1: Monitoring window (*SlidingWindow*) reached: Further message frames only after acknowledgement 2: Call for sending already made 3: Reason 1 + 2 4: Buffer read-out not yet completed 5: Reason 1 + 4 6: Reason 2 + 4 7: Reason 1 + 2 + 4
400E	LCOM_ERR_RECEIVER_CYCLE_TIME Error during cyclic receiving: The specified send cycle (*b16CycleTime*) is not maintained. This can happen when the FB is not called frequently enough or when the selected send cycle is too short. **Remedy**: Extend the send cycle.
	Additional information: dtActivationTime: Time of error occurrence b16State: Error state: ACTIVE/INACTIVE b32SysRetVal: - r32AdditionalValue1: Currently configured send cycle (*b16CycleTime*) in ms r32AdditionalValue2: Mean call cycle of the FB in ms r32AdditionalValue3: Mean (data) receive cycle of the FB in ms

errorId DWORD [Hex]	**Constant / meaning**
400F	LCOM_ERR_ACK_TIMEOUT The monitoring time for a sent data packet has expired but no acknowledgement has been received. With TCP communication, the connection is closed. **Remedy**: Increase the monitoring time (*b16AckTimeout*) or check the network load. The error is reset automatically when sender communication is inactive.
	Additional information: dtActivationTime: Time of error occurrence b16State: Error state: ACTIVE/INACTIVE b32SysRetVal: - r32AdditionalValue1: Configured monitoring time (*b32AckTimeout*) in ms r32AdditionalValue2: Mean call cycle of the FB in ms r32AdditionalValue3: -
4010	LCOM_ERR_DIFFERENT_VERSIONS An attempt is being made to communicate with different (protocol) versions of the **LCom** library. This can cause problems. Communication will be continued. **Remedy**: Both connection peers should use the same version of the protocol header. The error can only be reset by deactivating the FB (*enable* = FALSE).
	Additional information: dtActivationTime: Time of error occurrence b16State: Error state: ACTIVE/INACTIVE b32SysRetVal: - r32AdditionalValue1: Local library version Coding: xxyy.0 (LREAL) = Vxx.yy r32AdditionalValue2: Version of communication peer Coding: xxyy.0 (LREAL) = Vxx.yy r32AdditionalValue3: -

4.2 Overview of the structures

4.2.1 UDTLComParameterType data type for parameters

A data structure of type *UDTLComParameter* notifies all relevant settings to the **FBLComMachineCom** function block. The data structure is divided into the following substructures:

Table 4–5 Structure of UDTLComParameterType

Name	Data type	Description
sCfgConnection	sLComConnectionType	Connection settings
sCfgSender	sLComConfigurationType	Current sender settings
sCfgReceiver	sLComConfigurationType	Current receiver settings
sCfgTimeSync	sLComTimeSyncType	Settings for time synchronization

The *sCfgConnection* substructure contains all connection settings. It is checked by the function block with a positive edge of the *enable* input parameter and then accepted.

The *sCfgSender* and *sCfgReceiver* substructures contain all the parameters for a transmission to the communication peer when the standard header is used.

If no LCom protocol is used, the parameters in the *sCfgSender* substructure are accepted directly and the *sCfgReceiver* substructure is not used.
The *sCfgTimeSync* substructure contains the settings for time synchronization. The substructures are described in the following sections.

4.2.1.1 sLComConnectionType structure

The structure contains basic settings for the connection. These cannot be changed during a data transmission:

Table 4–6 Structure of the sLComConnectionType data type

Name	Data type	Description
boWithLComProtocol	BOOL	TRUE: Use LCom protocol
boAcceptUnknownPartner	BOOL	TRUE: Previously non-specified communication peers are accepted
boBigEndian	BOOL	TRUE: The data orientation used for packet headers is Big Endian FALSE: The data orientation used for packet headers is Little Endian Default: TRUE
boIsTcpClient	BOOL	Only relevant for TCP communication FALSE: Server TRUE: Client (active connection establishment) Default: FALSE
b8CPUType	BYTE	Coding of the CPU type used 0: CP 2: CPU 315 PN, 317 PN 3: CPU 319 PN 5: CPU 41x PN See Table 1–1 Controller types [Page 28] Default: FF
b16ComService	WORD	1: TCP Default: 1
b16LocalPort	WORD	Port on own side (in SIMATIC, port numbers between 2000 and 5000 are permissible)
b16RemotePort	WORD	Port on peer side (in SIMATIC, port numbers between 2000 and 5000 are permissible)
ab8RemoteAddress	ARRAY [0..3] OF BYTE	IP address of peer (xxx.xxx.xxx.xxx = [0].[1].[2].[3])
b16ConnectionId	WORD	ID number of the connection
b16LifeSignCycle	WORD	Sign-of-life cycle 1..60000 ms

boWithLComProtocol

The *boWithLComProtocol* variable defines whether the LCom protocol is used. This is recommended in order to utilize all the functions of the function block.

b16ComService

The *b16ComService* variable defines which communication service is used. Currently only TCP is supported.

biIsTcpClient

The *boIsTcpClient* variable defines whether it is the TCP client which is establishing a connection or the TCP server which is waiting for the connection to be established.

NOTE
In a TCP connection, one peer must always be configured as the client, the other as the server.

ab8RemoteAdress, b16LocalPort, b16RemotePort, boAcceptUnknownPartner

The *ab8RemoteAddress*, *b16LocalPort* and *b16RemotePort* variables define the IP address of the peer and the ports of both connection peers. The TCP server does not require all the connection parameters. However, if all parameters are specified, the *boAcceptUnknownPartner* variable can be used to configure whether connection attempts should only be accepted by the specified connection peer. Other connection peers are then rejected.

boBigEndian

boBigEndian can be used to configure how the orientation of the data types in the packet header is to be interpreted. SIMATIC works internally with the Big Endian format. Big Endian is selected here per default. This corresponds to the STEP 7 format and should not be changed. Without adaptation, direct communication is possible with both STEP 7 and SIMOTION.

b8CPUType

The *b8CPUType* parameter is used to define the CPU type of the controller. If this input is not connected or the default value of W#16#FF is applied, the **FBLComMachineCom** function block reads out the CPU based on the system description list. The CPU type can also be specified via this parameter; in this case, no readout is performed. If a CPU type that does not match the actual CPU is entered, an error is output and the connection is not established.

b16ConnectionId

Each SIMATIC CPU connection needs a separate connection ID. This is notified via the *b16ConnectionId* structural element.

NOTE
A connection ID may only be assigned once within a controller.

b16LifeSignCycle

The *b16LifeSignCycle* variable defines the cycle time for a sign-of-life. The sign-of-life is used to monitor the connection; it is sent when no user data is transmitted. If no sign-of-life or user data is received within 3-times the value of *u16LifeSignCycle*, it is assumed that the connection has failed.

4.2.1.2 sLComConfigurationType structure

This structure contains the current transmission settings. These can be changed during a data transmission.

Table 4–7 Structure of the sLComConfigurationType data type

Name	Data type	Description
b8ComMode	BYTE	0: Inactive 1: Cyclic 2: On data change 3: Once
b8SlidingWindow	BYTE	Send window 1…10
b16CycleTime	WORD	Cycle time for cyclic communication 1…60000 ms
b16AckTimeout	WORD	Monitoring time for connection 1…60000 ms

b8ComMode

The *b8ComMode* variable defines which transmission mode is to be used.

b16CycleTime

The *b16CycleTime* element is relevant only for cyclic transmission; it defines the cycle time.

b16AckTimeout

The *b16AckTimeout* variable defines the monitoring time after which an acknowledgement of the sent packet is expected at the latest. If this period is exceeded, the connection is closed and re-established.

b8SlidingWindow

The *b8SlidingWindow* parameter indicates how many data packets may be sent before an acknowledgement arrives from the receiver. This is an effective method to prevent the receiver being overloaded.

NOTICE

The last two parameters are only relevant when the LCom protocol is used. These functions are described in detail in Section 1.3.3 Using the LCom protocol [Page 14].

4.2.1.3 sLComTimeSyncType structure

This structure contains all settings for the time synchronization. These can be changed during a data transmission.

Table 4–8 Structure of the sLComTimeSyncType data type

Name	Data type	Description
boUseReceivedTimeStamps	BOOL	TRUE: Incoming time synchronization message frames are accepted as the system time.
b8SendModeTimeSync	BYTE	Send mode time synchronization 0: Inactive 1: Cyclic 2: Time
b16TimeSyncCycleTime	WORD	For mode 1 only: Send cycle of time stamp 1...65535 min
todTimeSyncAtTime	TIME_OF_DAY	For mode 2 only: Time at which a time synchronization message frame is sent.

NOTE

Time synchronization can only be used for communication with the LCom protocol.

boUseReceivedTimeStamp

You can select whether the time of the communication peer is to be adopted for the own system clock. If the *boUseReceivedTimeStamp* variable is set (default = FALSE), the received time is used. Runtimes are not compensated in this case. Accuracy below one second can be achieved.

NOTE

Only one of the two connection peers can perform time synchronization. If the *boUseReceivedTimeStamp* variable is set for both peers and the "cyclic" or "time" mode is set, synchronization is not performed and an error is output.

b8SendModeTimeSync, b16TimeSnycCycleTime, todTimeSyncAtTime

The *b8SendModeTimeSync* variable can be used to choose whether the function block is to perform time synchronization. This can be done cyclically or daily at a specified time. If cyclic mode is selected, the *b16TimeSyncCycleTime* variable defines the cycle in minutes (0 … 65535). If a time synchronization message frame is to be sent at a particular time, the time is specified using the*todTimeSyncAtTime* variable.

NOTE

If *boUseReceivedTimeStamps* is set, the times are synchronized at the start of data transmission irrespective of the mode (*b8SendModeTimeSync*).

4.2.2 UDTLComDiagnostics data type for diagnostics

Diagnostics data is stored in the *UDTLComDiagnosticsType* data type. This data type is divided into substructures and variables.

Table 4–9 Structure of the UDTLComDiagnostics data type

Name	Data type	Description
sActCfgSender	sLComConfigurationType	Current sender settings; for a description see Section sLComConfigurationType structure [Page 69]
sActCfgReceiver	sLComConfigurationType	Current receiver settings; for a description see Section sLComConfigurationType structure [Page 69]
sStatistics	sLComStatisticType	Connection information
sRemoteCfg	sLComRemoteCfgType	Settings for the connection peer
sLocalCfg	sLComLocalCfgType	Own settings
b32DiagnosticIndex	WORD	Pointer to the last diagnostic buffer entry (-1: no entry yet)
asDiagnostics	Array [0.. 63] OF STRUCT	Diagnostic buffer

4.2.2.1 sLComStatisticType structure

This structure contains statistical information for operation of the block.

Table 4–10 Structure of the sLComStatisticType data type

Name	Data type	Description
r32AvgTaskCycle	REAL	Mean value of FB call cycle
r32AvgReceiveMsgCycle	REAL	Mean value of message receive cycle, calculated only when the LCom protocol is used
r32MaxReceivedMsgCycle	REAL	Maximum receive cycle that occurred
b8LastReceivedMsgNumber	BYTE	Last message number received
b16NumberOfAckTimeOuts	WORD	Number of elapsed timeouts
dtLastTimeSync	DATE_AND_TIME	Last time synchronization performed
dtLastCfgSender	DATE_AND_TIME	Last sender configuration performed
dtLastCfgReceiver	DATE_AND_TIME	Last receiver configuration performed

The *sLComStatisticType* substructure contains general information that is useful for communication monitoring during operation.

r32AvgTaskCycle

The *r32AvgTaskCycle* indicates the mean value for the time between two block calls.

r32AvgReceiveMsgCycle

The *r32AvgReceiveMsgCycle* variable outputs the mean value for the cycle in which the data is received. The number of measurements used to calculate the mean value is specified as 20.

r32MaxReceivedMsgCycle

The *r32MaxReceivedMsgCycle* variable supplies the maximum receive cycle that occurred. The value is reset as soon as transmission type or transmission parameters are changed.

b8LastReceivedMsgNumber

The *b8LastReceivedMsgNumber* variable contains the message number of the last message received completely.

b16NumberOfAckTimeOuts

The *b16NumberOfAckTimeOuts* variable indicates how often the monitoring time was exceeded.

dtLastTimeSync

The instant of the last time synchronization is stored in *dtLastTimeSync*.

dtLastCfgSender, dtLastCfgReceiver

The last configuration of the sender and receiver is stored under *dtLastCfgSender* and *dtLastCfgReceiver*.

4.2.2.2 sLComRemoteCfgType structure

This structure contains the current information of the connection peer.

Table 4–11 Structure of the sLComRemoteCfgType data type

Name	Data type	Description
ab8Address	ARRAY [0..3] of BYTE	IP address of the peer
b16Port	WORD	Port on peer side
b16LifeSignCycle	WORD	Sign-of-life cycle of peer station
boUsesTimeStamps	BOOL	Incoming time synchronization message frames are accepted as the system time at the peer.
b8SendModeTimeSync	BYTE	Send mode of time synchronization at the communication peer 0: Inactive 1: Cyclic 2: Time
b8HeaderVersion	BYTE	Version in hexadecimal format (currently 10hex = V1.0)
b16SendBufferLength	WORD	Send data length of the peer
b16ReceiveBufferLength	WORD	Receive data length of the peer

This information is determined while the connection is being established or on receipt of a configuration message frame.

b16LifeSignCycle

The *b16LifeSignCycle* variable contains the sign-of-life cycle of the peer station. The monitoring of the communication path is derived from this. If no sign-of-life is received for three cycles, an error is issued and the *connected* connection status is reset.

ab8Address, b16Port

The IP address and the port of the current connection peer are stored in the elements *ab8Address* and *b16Port*.

boUsesTimeStamps

If *boUsesTimeStamps* is set, the peer uses the incoming time synchronization programs. The send mode for the time synchronization is specified by *b8SendModeTimeSync* (0: Inactive, 1: Cyclic or 2: Time).

b8HeaderVersion

The *b8HeaderVersion* variable contains the version of the message frame header. If the versions of the two devices are different, an error is issued.

b16SendBufferLength, b16ReceiveBufferLength

The data lengths of the peer are displayed under *b16SendBufferLength* and *b16ReceiveBufferLength*. The data length of the own sender must match the receive data size of the peer (same principle as for receiver); otherwise, an error is issued and no data is transmitted.

4.2.2.3 sLComLocalCfgType structure

This structure contains the current information of the connection peer.

Table 4–12 Structure of the sLComRemoteCfgType data type

Name	Data type	Description
b8HeaderVersion	BYTE	Version in hexadecimal format (currently 10hex = V1.0)
b16SendBufferLength	WORD	Send data length of the peer
b16ReceiveBufferLength	WORD	Receive data length of the peer

b8HeaderVersion

The *b8HeaderVersion* variable contains the version of the message frame header. If the versions of the two devices are different, an error is issued.

b16SendBufferLength, b16ReceiveBufferLength

The lengths specified for send and receive data in the own device are displayed under *b16SendBufferLength* and *b16ReceiveBufferLength*. The data length of the own sender must match or be less than the receive data size of the peer (same principle as for receiver); otherwise, an error is issued and no data is transmitted.

4.2.2.4 sLComDiagnosticsType structure

This structure contains diagnostic information about the errors that occurred.

Table 4–13 Structure of the sLComDiagnosticsType data type

Name	Data type	Description
b32ErrorId	DWORD	Error number
dtActivationTime	DATE_AND_TIME	Date and time of error
b16State	WORD	Error state: 1: ACTIVE 0: INACTIVE
b32SysRetVal	WORD	Return value of the system function
r32AdditionalValue1	LREAL	Other, error-dependent information according to the list
r32AdditionalValue2	LREAL	Other, error-dependent information according to the list
r32AdditionalValue3	LREAL	Other, error-dependent information according to the list

b32ErrorId

This element stores the error number output at the *errorId* output.

dtActivationTime

For error time statistics, the time and date of error activation is written to the *dtActivationTime* variable.

b16State

The *b16State* element indicates whether an error is active (*b16State = 1*) or inactive (*b16State = 0*). This can be helpful if several errors are present but only the last one that occurred can be indicated at the *errorId* output.

b32SysRetVal

If an error is initiated by calling a system function, the error code of this system function is output in the *b32SysRetVal* variable. For the meaning of the error number, refer to the SIMATIC documentation.

r32AdditionalValue

The three elements *r32AdditionalValue1*, *r32AdditionalValue2* and *r32AdditionalValue3* contain additional information about the error. The meanings of these additional values are explained in the list in Section Error messages [page 55].

Appendix A

A.1. Contacts

Siemens AG
Industry Sector
I DT MC PMA APC
Frauenauracher Str. 80
D-91056 Erlangen, Germany
Fax: +49 (9131) 98-1297
E-mail: profinet.team.motioncontrol.i-dt@siemens.com

A.2. Internet addresses

www.siemens.com/simotion
www.siemens.com/sinamics
www.siemens.com/motioncontrol/apc
www.siemens.com/packaging