Technical Information

SICK
RFH6xx Function Block

Block Version V2.X

SICK RFH6XX TCP Function Block for
Siemens S7-Controls (Step7 V5.5)

SICK
Sensor Intelligence.

Table of contents

1 About this document

Please read this chapter carefully before you start working with these operating instructions and the SICK RFH6XX function block.

1.1 Purpose of this document

These operating instructions describe how to use the SICK RFH6XX TCP function block. They are intended to guide technical personnel working for the machine manufacturer/operator through the processes of configuring and commissioning the function block.

1.2 Target group

This Operation Manual is aimed for specialists, such as technicians and engineers.

2 General information

The "SICK RFH6XX TCP" function block is used to facilitate communication between a SIMATIC controller and a SICK RFH6xx RFID interrogator. The RFH6xx communicates with the controller via a TCP connection.

The following figure shows how the function block is represented in the function block diagram (FBD) view.

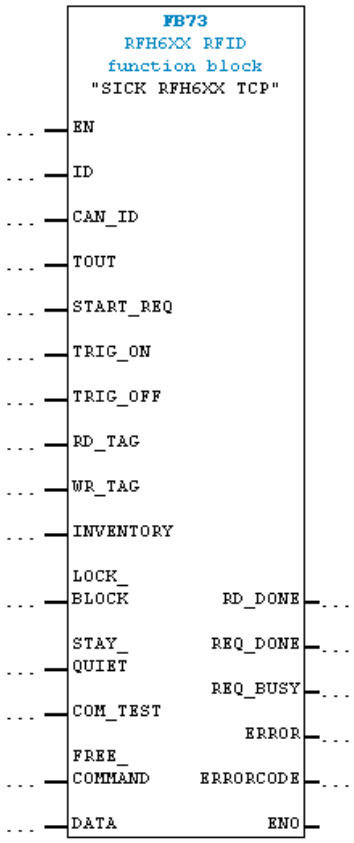

Figure 1: SICK RFH6XX TCP function block

Block functions:

- Send a trigger command (CoLa[i] command) via the PLC
- Receive reading results (defined in SOPAS-ET[ii] output format)
- Read and write transponder content
- Execute an inventory command (show all transponders in the read field)
- Permanent blocking of transponder blocks
- Execute a communication test
- Communicate via freely selectable CoLa commands (CoLa-A protocol)
- Address devices that communicate with each other via CAN bus

[i] The command language (CoLa) is a protocol internal to SICK for communicating with SOPAS devices.
[ii] SOPAS-ET is an engineering tool for configuring SICK sensors.

3 Hardware configuration

3.1 Supported PLC controllers

The function block may only be operated with Simatic S7-300 and S7-400 controllers with an integrated TCP interface. Communication via a communication processor (CP module) is not supported.

3.2 Establishing a connection

A TCP connection to the sensor must be established before the function block can be used. For S7 controllers with an integrated IE interface, Siemens provides the following communication blocks (Standard Library → Communication Blocks):

- FB65 (TCON): To establish a TCP connection
- FB65 (TDISCON): To close down a TCP connection
- FB63 (TSEND): To send data
- FB64 (TRCV): To receive data

Figure 2 shows the FB65 call with the associated instance (DB65) in OB1. When the controller starts up, OB100 is executed once. The start bit (REQ) of FB65 is set in OB100 to establish the connection via FB65. Successful connection setup is indicated by the bit DONE = TRUE. The connection parameters for establishing a connection are saved in a data structure (UDT65).

The ID parameter of the TCON block and the instantiated UDT structure must be identical to the ID of the SICK RFU TCP block.

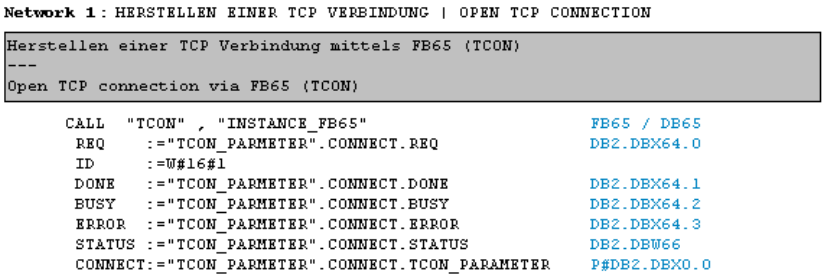

Figure 2: Using FB65 (TCON) to establish a TCP connection

The table below shows an example configuration of the UDT65.

Byte	Parameter	Data type	Start value	Description
0 - 1	block_length	WORD	W#16#0040	Length of the UDT65: 64 bytes (fixed)
2 - 3	id	WORD	W#16#0001	Reference to TCP connection. This value must be identical to the ID parameter at TCON (FB65) and to the SICK RFH TCP (FB73) block.
4	connection_type	BYTE	B#16#11	Connection type = TCP
5	active_est	BOOL	TRUE	Active connection establishment

Byte	Parameter	Data type	Start value	Description
6	local_device_id	BYTE	B#16#02	Type of TCP connection In this case: Communication via the integrated Ethernet interface for **CPUs 315-2 PN/DP and 317-2 PN/DP**
7	local_tsap_id_len	BYTE	B#16#02	For connection type B#16#11 and a passive end point
8	rem_subnet_id_ len	BYTE	B#16#00	Not used
9	rem_staddr_len	BYTE	B#16#04	Length of the IP address of the station (SICK device)
10	rem_tsap_id_len	BYTE	B#16#02	Fixed for connection type B#16#11
11	Next_staddr_len	BYTE	B#16#00	Used length of the next_staddr parameter (not used)
12 - 27	Local_tsap_id	Array [1..16] of BYTE	-	Number of the local port used: [1] = High byte of the port number used represented in hex format [2] = Low byte of the port number used represented in hex format [3..16] = B#16#00
28 - 33	rem_subnet_id	Array [1..6] of BYTE	-	Not used [1..16] = B#16#0
34 - 39	Rem_staddr	Array [1..6] of BYTE	-	IP address of the RFH For example: 192.168.10.15 [1] = B#16#C0 (192) [2] = B#16#A8 (168) [3] = B#16#0A (10) [4] = B#16#0F (15) [5-6] = B#16#00
40 - 55	Rem_tsap_id	Array [1..16] of BYTE	-	Port number of connected RFH SICK communication port: 2112 (decimal) [1] = B#16#08 (high byte) [2] = B#16#40 (low byte) [3..16] = B#16#0
56 - 61	Next_staddr	Array [1..6] of BYTE	-	Not used [1..6] = B#16#0
62 - 63	spare	WORD	W#16#0000	Not used

Please refer to the Step7 help system for a precise description of the UDT parameters.

The function block will only work if there is an active TCP connection to the RFU. Figure 3 shows the Step7 diagnostics screen for open communication via Industrial Ethernet. To access this screen, select "Module Status → Diagnostics → Occupied Connection Resources".

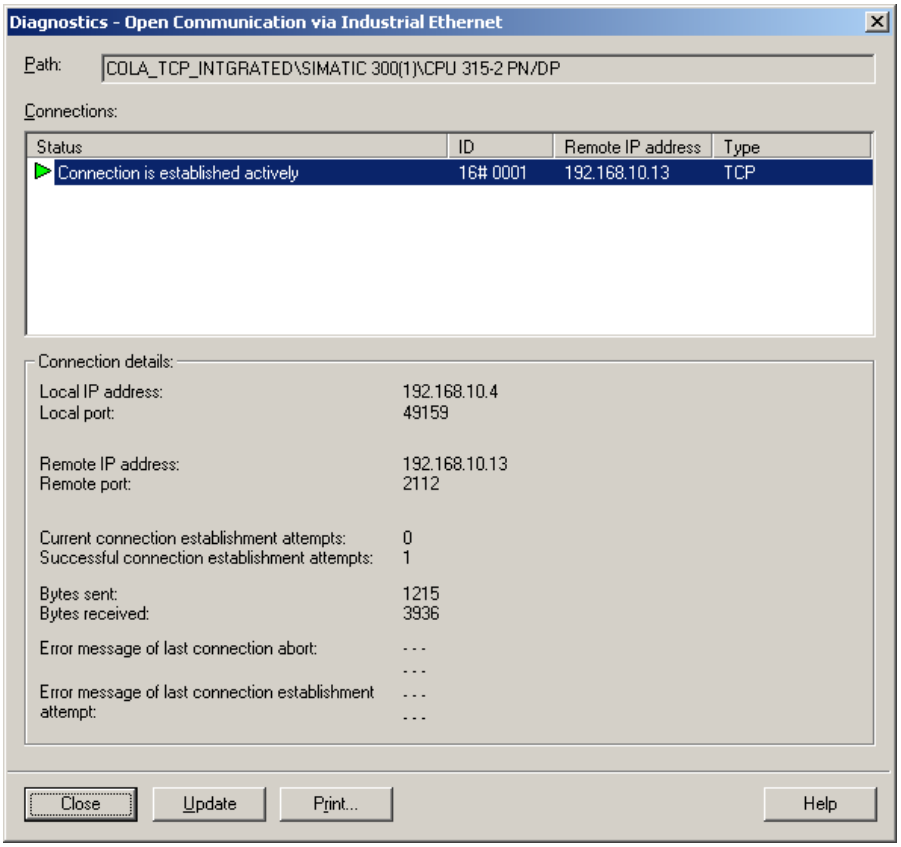

Figure 3: Communication diagnostics

4 Block description

The function block is an asynchronous FB, i.e., processing encompasses several function block calls. This means that the block must be called in the user program on a cyclical basis.

The RFU block encapsulates the "SICK CCOM TCP" (FB13) function block, which facilitates communication between the PLC and the sensor. FC10 (SICK COLA ACCESS) is used internally to interpret CoLa telegrams.

4.1 Block specifications

Block number:	FB73
Block name:	SICK RFH6XX TCP
Version:	2.1
Blocks called:	FB63 (TSEND) FB64 (TRCV) SFC20 (BLKMOV) SFB4 (TON) FB13 (SICK CCOM TCP) FC10 (SICK COLA ACCESS)
Data blocks used:	DB73 (SICK RFH DATA)
Block call:	Cyclical
Flags used:	None
Counters used:	None
Registers used:	AR1, AR2 (for multi-instance calls)
Capable of multi-instancing:	Yes
Language used for block creation:	Step7 STL
Step7 version:	Simatic Step7 V5.5

The system functions (SFCs) used in the function block must exist on the controller that is being used.

If block numbers are changed, then the corresponding calls in the SICK RFH6XX TCP block must be updated accordingly.

4.2 Operating principle

The following communication parameters must be specified before the RFH block can be used:

ID: Connection ID of the TCP connection. The value specified here must be the same as for the ID parameter of the TCON block and the instantiated UDT structure. See also Figure 3.

DATA: The data block (DB73) that accompanies the function block contains input and output parameters for the supported block functions. The data block must be transferred to the "DATA" input parameter of the function block.

Executable block functions:

- Trigger on → Uses a CoLa command to open the device reading gate
- Trigger off → Uses a CoLa command to close the device reading gate
- Read tag → Reads out the transponder data
- Write tag → Writes transponder data
- Inventory → The inventory function searches for active transponders in the RFH reading range and returns their UIDs.
- Lock block → Permanent blocking of a selected transponder block
- Stay quiet → Mutes the RFID tag located in the field.
- Communication test → Checks whether the device can be contacted by sending command "sRI0"
- Free command → Executes a freely selectable CoLa command

To execute a block function (TRIG_ON, RD_TAG, etc.), the desired function must first be selected. Only one function can be executed at a time. The START_REQ parameter must be triggered with a rising edge (signal change from logical zero to one) in order for the function to be executed. Until a valid device response is received, the REQ_BUSY parameter signals that a response is still pending.

If the block's REQ_DONE output parameter = TRUE, it means that the function has been successfully completed. If data was requested from the device during this function (e.g., RD_TAG), this data is copied to the relevant data area of the accompanying user data block (DATA).

Data sent via a trigger command (TRIG_ON, TRIG_OFF) or directly by the device (e.g., direct trigger via a photoelectric sensor) is stored in the data block (ReadingResult.arrResult). For one PLC cycle, the RD_DONE output parameter indicates that new data has been received. The data sent by the device can be changed or adapted in SOPAS output format.

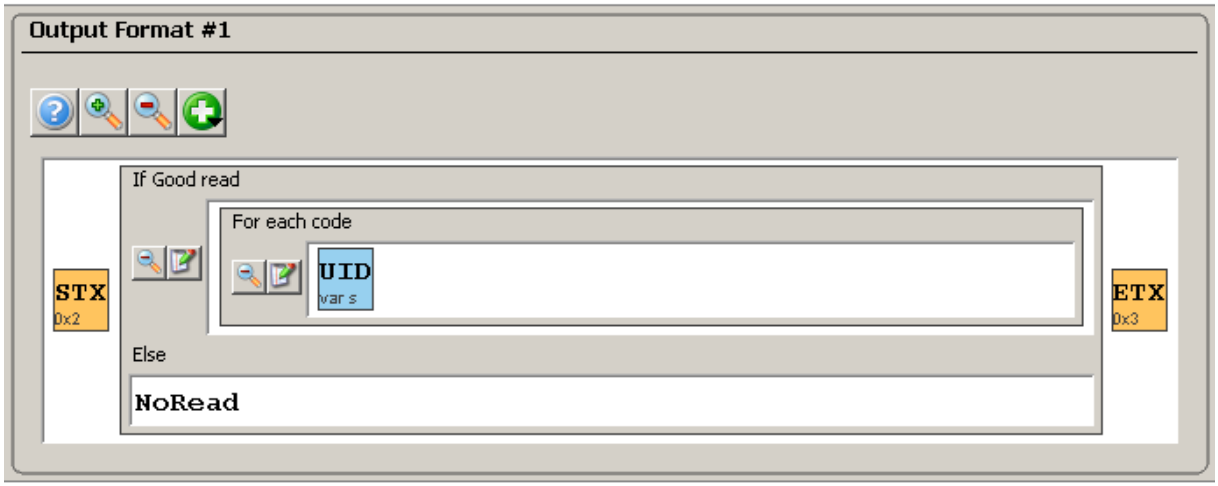

Figure 4: SOPAS output format

4.3 Response to errors

If the function block has an incorrect input value or if the input has been connected incorrectly, an error bit (ERROR) is set and an error code (ERRORCODE) is output. In this case, no further processing is carried out. The diagnostic parameters (ERROR, ERRORCODE) of the function block retain their values until a new command is started.

4.4 Timing

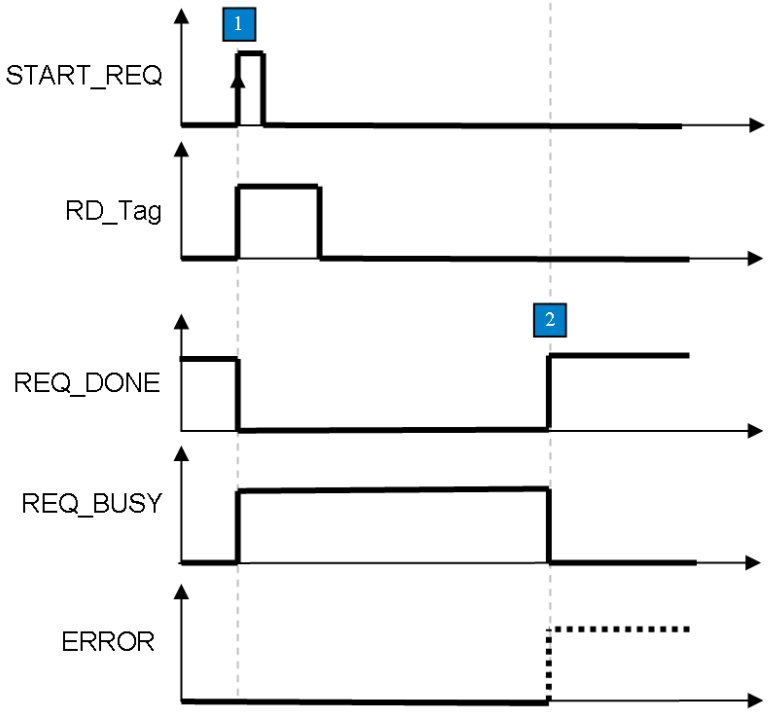

Figure 5: Timing diagram

1: Request triggered by rising edge at START_REQ
The desired function (RD_TAG in this case) must be selected at the same time/in advance. Only one function may be selected at once; otherwise, the function will be aborted with "ERROR".

2: Once all commands have been sent and all responses received, the function is terminated with "REQ_DONE". If an error occurred during the function, the function is terminated with "ERROR". "ERRORCODE" contains information on the error that occurred if the function is aborted with "ERROR".

4.5 Value transfer

The supplied data block "SICK RFH DATA" (DB73) contains input and output parameters for all supported block functions. The data block can be renamed according to the user program. The data structure has a fixed definition and may not be modified except for the last entry (ReadingResult.arrResult) (see chapter 4.6: Receiving reading results > 200 bytes).

DB73 -- "SICK RFH DATA" -- SICK_RFH6XX_PNDP\SIMATIC 300(1)\CPU 315-2 PN/DP\...\DB73

Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	Mode	STRUCT		-- MODE --
+0.0	bMode	BOOL	FALSE	1: Use a fixed UID | 0: Use the UID of the transponder in the field (IN)
+2.0	arrUID	ARRAY[1..8]		If bMode=1, this UID will be used for a Read/Write/Lock/Stay quit job (IN/OUT)
*1.0		BYTE		
=10.0		END_STRUCT		
+10.0	iLockBlock	INT	0	Number of the block that should be locked (IN)
+12.0	Inventory	STRUCT		-- INVENTORY --
+0.0	iNumRetTags	INT	0	Number of returned transponders (OUT)
+2.0	arrTagInfo	ARRAY[1..5]		Max. 5 transponder (OUT)
*0.0		STRUCT		
+0.0	nError	BYTE	B#16#0	Error code (OUT)
+1.0	nRSSI	BYTE	B#16#0	RSSI RX value (OUT)
+2.0	nDSFID	BYTE	B#16#0	DSFID (OUT)
+4.0	arrUID	ARRAY[1..8]		UID (OUT)
*1.0		BYTE		
=12.0		END_STRUCT		
=62.0		END_STRUCT		
+74.0	ReadTag	STRUCT		-- READ TAG --
+0.0	iStartBlock	INT	0	Number of the first block that should be read (IN)
+2.0	iNumBlocks	INT	0	Number of blocks that should be read (IN)
+4.0	iDataLength	INT	0	Content length in bytes (OUT)
+6.0	arrData	ARRAY[1..128]		Data to be read (OUT)
*1.0		BYTE		
=134.0		END_STRUCT		
+208.0	WriteTag	STRUCT		-- WRITE TAG --
+0.0	iStartBlock	INT	0	Number of the first block that should be written (IN)
+2.0	iNumBlocks	INT	0	Number of blocks that should be written (IN)
+4.0	iBlockSize	INT	4	Block size in bytes (IN)
+6.0	arrData	ARRAY[1..128]		Data to be write (IN)
*1.0		BYTE		
=134.0		END_STRUCT		
+342.0	FreeCommand	STRUCT		-- FREE COMMAND --
+0.0	iCommandLength	INT	0	Byte length of the free command (IN)
+2.0	arrCommand	ARRAY[1..100]		Command (SICK CoLa-A protocol without [STX]/[ETX] framing) (IN)
*1.0		CHAR		
+102.0	iResultLength	INT	0	Byte length of the free command result (OUT)
+104.0	arrResult	ARRAY[1..100]		Result (SICK CoLA-A protocol) (OUT)
*1.0		CHAR		
=204.0		END_STRUCT		
+546.0	ReadingResult	STRUCT		-- READING RESULT --
+0.0	nCounter	BYTE	B#16#0	This counter is incremented if a new reading result has arrived (OUT)
+2.0	iLength	INT	0	Byte length of the reading result (OUT)
+4.0	arrResult	ARRAY[1..200]		Reading result data (OUT)
*1.0		CHAR		
=204.0		END_STRUCT		
=750.0		END_STRUCT		

Figure 6: Structure of SICK RFH DATA user data DB

4.5.1 Mode

The RFH can only communicate with a single transponder at any one time. For this reason, read and write commands are always addressed. The function block uses the UID (unique identifier) to identify the transponder.

The function block supports two different modes in order to determine which transponder UID is to be communicated with:

Mode 1: The system always communicates with the transponder which is currently in the read field. This mode can only be used when precisely one tag is located within the field.
Mode 2: A user-defined transponder UID is used for the purpose of communication.

Parameter	Declaration	Data type	Description
Mode.bMode	Input	BOOL	Addressing mode FALSE: Mode 1 active TRUE: Mode 2 active
Mode.arrUID	Input/Output	INT	Transponder identifier (UID) *The UID is read out automatically in Mode 1.*

Table 1: Mode parameters

4.5.2 Lock block

The lock block function allows you to protect any block on the RFID tag by preventing it from being overwritten. The block number is specified via the iLockBlock parameter before the FB function is executed. The function permanently locks the selected block. The block cannot be unlocked.

Parameter	Declaration	Data type	Description
iLockBlock	INPUT	INT	Number of the block to be locked

4.5.3 Inventory

The inventory function searches for active transponders within the receiving range of the sensor. The function block provides the following information for each detected transponder (max. 5 transponders).

Parameter	Declaration	Data type	Description
Inventory. iNumRetTags	Output	INT	Number of detected transponders
Inventory. arrTagInfo[ ].nError	Output	BYTE	Transponder error code (see RFH operating instructions)
Inventory. arrTagInfo[ ].nRSSI	Output	BYTE	RSSI (signal strength of detected transponder)
Inventory. arrTagInfo[ ].nDSFID	Output	BYTE	DSFID of detected transponders
Inventory. arrTagInfo[ ].arrUID	Output	ARRAY [1..8] OF BYTE	UID of detected transponders in HEX format

4.5.4 Read tag

The read tag function is used to read a defined data area of a tag. This function can only ever be applied to one tag. The selected mode determines which transponder the system communicates with (see chapter 4.5.1).

Prior to each read process, it is necessary to define which blocks are to be read out of the transponder. Once the read process is successfully completed, the byte length of the read data is stored in the user data DB along with the user data.

Parameter	Declaration	Data type	Description
ReadTag. iStartBlock	Input	INT	Number of block at which the read process is to start
ReadTag. iNumBlocks	Input	INT	Number of blocks to be read
ReadTag. iDataLength	Output	INT	Length of read content in bytes
ReadTag.arrData	Output	ARRAY [1..128] OF BYTE	Content of read blocks

Table 2: Read tag parameters

4.5.5 Write tag

The write tag function is used to write to a defined data area of a tag. This function can only ever be applied to one tag. The selected mode determines which transponder the system communicates with (see chapter 4.5.1).

Prior to each write process, it is necessary to define the block at which the write process is to start and how many blocks are to be written. The block length of the transponder must also be specified, because these changes depending on the tag type (see information from tag manufacturer).

Parameter	Declaration	Data type	Description
WriteTag. iStartBlock	Input	INT	Number of block at which the write process is to start
WriteTag. iNumBlocks	Input	INT	Number of blocks to be written
WriteTag. iBlockSize	Input	INT	Block length in bytes Valid range: [4,8,12,16,…]
WriteTag.arrData	Input	ARRAY [1..128] OF BYTE	Data to be written to the transponder blocks

Table 3: Write tag parameters

4.5.6 Free command

The free command allows you to communicate with the RFH via a valid CoLa command. For this to happen, the command must be stored in the "arrCommand" parameter of the "FreeCommand" structure. The character length of the command to be transmitted is written to the "iCommandLength" parameter. The commands can be obtained from the device description or SOPAS-ET.

Parameter	Declaration	Data type	Description
FreeCommand. iCommandLength	Input	INT	Character length of the CoLa command to be transmitted Valid range [1..100]
FreeCommand. arrCommand	Input	ARRAY [1..100] OF CHAR	Freely selectable CoLa command (for commands, see device documentation)
FreeCommand. iResultLength	Output	INT	Byte length of received CoLa telegram
FreeCommand. arrResult	Output	ARRAY [1..100] OF CHAR	Response to the transmitted CoLa telegram

Table 4: Free command parameters

4.5.7 Reading result

The "ReadingResult.arrResult" array stores data that is sent via a trigger command (TRIG_ON, TRIG_OFF) or directly from the device (e.g., direct trigger via photoelectric sensor). The RD_DONE output parameter signals whether data has been received.

Parameter	Declaration	Data type	Description
ReadingResult. nCounter	Output	BYTE	The receive counter is incremented by one as soon as a new reading result is received. Value range: [0x00...0xFF]
ReadingResult. iLength	Output	INT	Byte length of received reading result
ReadingResult. arrResult	Output	ARRAY [1..200] of BYTE	Response to a trigger signal (can be defined via the SOPAS output format) The maximum length of the received data is 200 bytes. Chapter 4.6 describes the procedure for receiving longer data telegrams.

Table 5: Reading result parameters

4.6 Receiving reading results > 200 bytes

The function block is designed to receive reading results up to a length of 200 bytes. If longer data is to be received, the function block must be changed at the points indicated below.

Changes in SICK RFH DATA data block:

The length of the "ReadingResult.arrResult" array in the user data block supplied (DB73) must be set so that the reading result to be received fits into the data area of the variable.

+426.0	ReadingResult	STRUCT		-- READING RESULT --
+0.0	nCounter	BYTE	B#16#0	This counter is incremented if a new reading result has arrived (OUT)
+2.0	iLength	INT	0	Byte length of the reading result (OUT)
+4.0	arrResult	ARRAY[1..200]		Reading result data (OUT)
*1.0		CHAR		
=204.0		END_STRUCT		

Figure 7: Receiving reading results > 200 bytes (change to data block)

Changes in SICK RFH6XX TCP function block:

In the static area of the variable overview, the length of the "arrRecord" variable must be adapted so that the reading result fits into the data area of the variable. The array is not allowed to be less than 500 bytes in length, but must be greater than or equal to the length of "ReadingResult.arrResult".

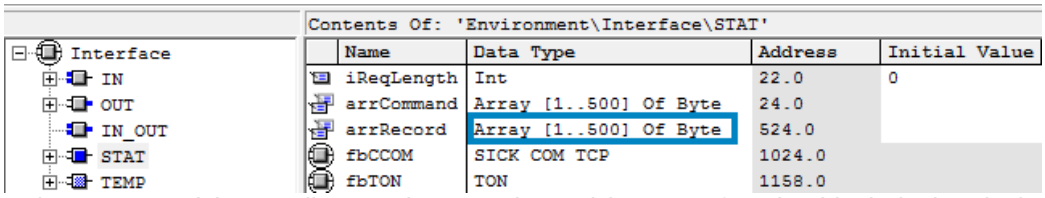

Figure 8: Receiving reading results > 200 bytes (change to function block declaration)

The newly defined array lengths must be entered into network 3 of the SICK RFH6XX TCP function block.

```
Network 3: CONFIGURATION
- Configure the length of the "Record" array
- Configure the length of the "Command" array
- Configure the length of the "Reading Result" array
- Configure [STX]/[ETX] framing flag

PLEASE NOTE:
"Record" array >= "Command" array
"Record" array >= "Reading Result" array
```

```
//-- LENGTH OF THE RECORD ARRAY
      L     500
      T     #iArrayRecLen

//-- LENGTH OF THE COMMAND ARRAY
      L     500
      T     #iArrayComLen

//-- LENGTH OF THE READING RESULT ARRAY
      L     200
      T     #iArrayReadLen

//-- FRAMING
      CLR                                  // Set telegram framing
      =     #bAddFraming

//-- RESET READING RESULT FLAG
      CLR
      =     #RD_DONE
```

Figure 9: Receiving reading results > 200 bytes (change to block code)

After modification, the instance of the function block must be updated. Subsequently, the modified user data block and the function block must be transferred to the PLC again, together with the updated instance.

5 Parameters

Parameter	Declaration	Data type	Memory area	Description
EN	INPUT	BOOL	I,M,D,L, const.	Enable input (LD and FBD)
ID	INPUT	WORD	I,M,D,L, const.	Connection ID for configured TCP connection (see communication diagnostics Figure 3 or ID parameter TCON FB)
CAN_ID	INPUT	INT	I,M,D,L, const.	CAN ID of the sensor to be addressed If no CAN network is used, the CAN ID is 0. The master or multiplexer is always addressed with CAN ID 0, even if it has been assigned another CAN ID.
TOUT	INPUT	TIME	I,M,D,L, const.	Period of time, after which a timeout error is triggered
START_REQ	INPUT	BOOL	I,M,D,L	Rising edge: Selected block function is executed
TRIG_ON	INPUT	BOOL	I,M,D,L, const.	Block function: Execute a device trigger (open trigger window).
TRIG_OFF	INPUT	BOOL	I,M,D,L, const.	Block function: Execute a device trigger (close trigger window). The result sent from the device (SOPAS output format) is stored in the "ReadingResult.arrResult" variable of the user data DB (DB73).
RD_TAG	INPUT	BOOL	I,M,D,L, const.	Block function: Read tag content. This function only works if the parameters of the "ReadTag" structure for the transferred data block have been assigned valid values (see chapter 4.5.4). The selected addressing mode determines which transponder is to be read (see chapter 4.5.1).
WR_TAG	INPUT	BOOL	I,M,D,L, const.	Block function: Write tag content. This function only works if the parameters of the "WriteTag" structure for the transferred data block have been assigned valid values (see chapter 4.5.5). The selected addressing mode determines which transponder is to be written to (see chapter 4.5.1).

Parameter	Declaration	Data type	Memory area	Description
INVENTORY	INPUT	BOOL	I,M,D,L, const.	Searches for active transponders within the receiving range and indicates their UID, DSFID, and RSSI signal strengths
LOCK_ BLOCK	INPUT	BOOL	I,M,D,L, const.	Protects a defined block by locking it so that it cannot be overwritten This function only works if a valid block number has been assigned to the iLockBlock parameter in the data block being transferred (see chapter 4.5.5). The function permanently locks the selected block. The block cannot be unlocked.
STAY_QUIET	INPUT	BOOL	I,M,D,L, const.	Mutes the RFID tag located in the field This function can only be used if the HF field of the RFID device is permanently switched on (see SOPAS → Transponder Communication → HF Field).
COM_TEST	INPUT	BOOL	I,M,D,L, const.	Block function: Execute a communication test. REQ_DONE = TRUE: Communication OK REQ_DONE = FALSE: Communication not OK
FREE_ COMMAND	INPUT	BOOL	I,M,D,L, const.	Block function: Execute a free command. This function only works if valid data has been assigned to the iCommandLength and arrCommand parameters in the structure (FreeCommand) within the user data block (DB73) (see chapter 4.5.6). Following successful transfer, the command response (REQ_DONE = TRUE) is made available in the RESULT area of the data block.
DATA	INPUT	BLOCK_ DB	Const.	Transfers the accompanying user data block that is required to configure the block functions and store the reading results (DB73)
RD_DONE	OUTPUT	BOOL	Q,M,D,L	Rising edge: New reading result received

Parameter	Declaration	Data type	Memory area	Description
REQ_DONE	OUTPUT	BOOL	Q,M,D,L	Indicates whether the selected block function has been successfully completed TRUE: Successfully completed FALSE: Not completed
REQ_BUSY	OUTPUT	BOOL	Q,M,D,L	Command in progress
ERROR	OUTPUT	BOOL	Q,M,D,L	Error bit: 0: No error 1: Aborted with error
ERROR CODE	OUTPUT	WORD	Q,M,D,L	Error status (see "Error codes")
ENO	OUTPUT	BOOL	Q,M,D,L	Enable output (LD and FBD)

Table 6: Block parameters

6 Error codes

The ERRORCODE parameter contains the following error information:

Error code	Brief description	Description
W#16#0000	No error	No error
W#16#0001	Timeout error	Command could not be executed within the selected timeout period Possible causes: - Device is not connected to the PLC - Incorrect communication parameters - CAN bus station not present
W#16#0002	Internal block error	Internal block error
W#16#0003	No block function selected, or more than one block function selected	Only one block function can be executed at a time.
W#16#0004	Received reading result > reading result array	The reading result received is longer than 200 bytes. See chapter 4.6 for information on how to receive longer reading results.
W#16#0005	100 < FreeCommand. iCommandLength <=0	Length of free command is invalid Valid range: [1...100]
W#16#0006	Free command response > 100 bytes	The response to the free command sent is longer than 100 bytes.
W#16#0007	63 < CAN_ID < 0	Invalid CAN ID Valid range: [0..63]
W#16#0008	Reserved	Reserved
W#16#0009	Communication error	Communication could not be established with the device. Possible causes: - Invalid ID parameter - Connection not established - A telegram > arrRecord was received.
W#16#**XX**0A	Device error	A device error occurred ("sFA XX"). **XX** = device error (see device documentation)

Error code	Brief description	Description
W#16#000B	Invalid command response	The selected function was not executed. The following causes are possible, depending on the function: - Incorrect trigger setting in the SOPAS device configuration - Device is not in "Run mode" - Tag not long enough in field - Attempt to access a non-existent tag area (check iStartBlock and iNumBlocks parameters) - Invalid UID (check Mode.arrUID)
W#16#000C – W#16#000F	Reserved	Reserved
W#16#0010	Tags in field > 5 (Inventory)	Inventory cannot be executed because there are more than 5 transponders in the read field of the RFH.
W#16#0011	ReadTag.iStartBlock < 0	Invalid start of reading (read tag)
W#16#0012	32 < ReadTag. iNumBlocks <= 0	A maximum of 128 bytes of transponder data can be read per function call (32 blocks of 4 bytes). Valid range: [1..32]
W#16#0013	Content to be read > 128 bytes	A maximum of 128 bytes of data can be read per function call. The RD_TAG function must be executed several times in succession in order to read more than 128 bytes of data.
W#16#0014	WriteTag.iStartBlock < 0	Invalid parameter Valid range: [0 .. max. number of transponder blocks]
W#16#0015	32 < WriteTag. iNumBlocks <= 0	A maximum of 128 bytes of transponder data can be written per function call (32 blocks of 4 bytes). Valid range: [1..32]
W#16#0016	WriteTag.iBlockSize <> 4,8,12,16,…	Invalid block size Valid range: [4,8,12,16,…]
W#16#0017	Content to be written > 128 bytes	A maximum of 128 bytes of data can be written per function call. The WR_TAG function must be executed several times in succession in order to write more than 128 bytes of data.

Error code	Brief description	Description
W#16#0018	iLockBlock < 0	Invalid iLockBlock parameter Valid range: [0 .. max. number of transponder blocks]
W#16#**XX**19	Transponder error	A transponder error has come up. **XX** = Transponder- / Device errors <u>Transponder errors:</u> 16#00: No error 16#01: Command not supported 16#02: Command nor recognized 16#03: Option not supported 16#0F: Unknown error 16#10: Block not available 16#11: Block already locked 16#13: Block write error 16#14: Block lock error <u>Device errors:</u> 16#1E: Unknown error 16#1F: CRC error 16#20: Parity error 16#21: Timeout error 16#22: No response error 16#23: Collision error 16#24: Content check error 16#25: Framing error 16#26: Verify error 16#27: Transmit error 16#28: Receive error 16#29: Non addressed error 16#2A: Tag type selection error 16#2B: Max block count error 16#2C: Block length mismatch error 16#46: Slot detect warning For further error codes please have a look at the device description.
W#16#001A	No tag in field	There are no tags in the receiving range of the RFH.
W#16#001B	More than one tag in field	There is more than one tag in the receiving range of the RFH. This error can only occur in Mode 1.

Table 7: Error codes

7 Examples

Figure 12 shows an example of a connected SICK RFH6XX TCP function block. The TCP connection to the SICK sensor is established with FB65 (TCON) during PLC startup (see Figure 10 / Figure 11). A zero is entered for the CAN ID because the RFH is not operating on a CAN network.

Program call:

```
OB100 :  "Complete Restart"
Comment:

Network 1: TCP VERBINDUNG HERSTELLEN | ESTABLISHing A TCP CONNECTION
Herstellen einer TCP Verbindung nach jedem SPS restart.
---
Open TCP connection after every PLC restart.

      SET
      =     "TCON_PARMETER".CONNECT.REQ    DB2.DBX64.0
```

Figure 10: Start of the connection setup in OB100

```
Network 1: HERSTELLEN EINER TCP VERBINDUNG | OPEN TCP CONNECTION
Herstellen einer TCP Verbindung mittels FB65 (TCON)
---
Open TCP connection via FB65 (TCON)

// SET IP ADDRESS OF THE RFU
      L     192
      T     "TCON_PARAMETER".CONNECT.TCON_PARAMETER.rem_staddr[1]
      L     168
      T     "TCON_PARAMETER".CONNECT.TCON_PARAMETER.rem_staddr[2]
      L     10
      T     "TCON_PARAMETER".CONNECT.TCON_PARAMETER.rem_staddr[3]
      L     152
      T     "TCON_PARAMETER".CONNECT.TCON_PARAMETER.rem_staddr[4]

// SET ETHERNET CONNECTION PORT OF THE RFU
      L     2112
      T     DB65.DBW   40

      CALL  "TCON" , DB165
       REQ    :="TCON_PARAMETER".CONNECT.REQ
       ID     :=W#16#1
       DONE   :="TCON_PARAMETER".CONNECT.DONE
       BUSY   :="TCON_PARAMETER".CONNECT.BUSY
       ERROR  :="TCON_PARAMETER".CONNECT.ERROR
       STATUS :="TCON_PARAMETER".CONNECT.STATUS
       CONNECT:="TCON_PARAMETER".CONNECT.TCON_PARAMETER

      CLR
      =     "TCON_PARAMETER".CONNECT.REQ
```

Figure 11: FB65 (TCON) call for creating a TCP connection

```
Network 3 : CALL SICK RFH PNDP FUNCTION BLOCK
Comment:

      CALL  "SICK RFH6XX TCP" , "INSTANCE_FB73"     FB73 / DB173
       ID           :=W#16#1
       CAN_ID       :="iCanID"                     MW16
       TOUT         :=T#5S
       START_REQ    :="bRequest"                   M10.0
       TRIG_ON      :="bTriggerOn"                 M12.1
       TRIG_OFF     :="bTriggerOff"                M12.2
       RD_TAG       :="bRdTag"                     M12.3
       WR_TAG       :="bWrTag"                     M12.5
       INVENTORY    :="bInventory"                 M12.7
       LOCK_BLOCK   :="bLockBlock"                 M13.0
       STAY_QUIET   :="bStayQuit"                  M13.1
       COM_TEST     :="bComTest"                   M12.4
       FREE_COMMAND:="bFreeCommand"                M12.6
       DATA         :="SICK RFH DATA"              DB73
       RD_DONE      :="bRdDone"                    M10.1
       REQ_DONE     :="bReqDone"                   M10.2
       REQ_BUSY     :="bReqBusy"                   M10.3
       ERROR        :="bError"                     M10.4
       ERRORCODE    :="nErrorcode"                 MW14
```

Figure 12: Example of a connected SICK RFH6XX TCP function block

7.1 Reading tag content

First, it is necessary to determine which transponder the system is to communicate with. If bit Mode.bMode = FALSE, then the system will communicate with the transponder that is currently located in the RFID sensor's reading range.

// ========== Mode ==========			
DB73.DBX 0.0	"SICK RFH DATA".Mode.bMode	BOOL	false

Figure 13: Selection of communication mode

Then, it is necessary to define what content is to be read out of the transponder.

Start block: 0
Number of blocks: 2 (number of blocks to read)

// ========== Read Tag ==========			
DB73.DBW 74	"SICK RFH DATA".ReadTag.iStartBlock	DEC	0
DB73.DBW 76	"SICK RFH DATA".ReadTag.iNumBlocks	DEC	2

Figure 14: Read tag parameters

The reading function (bRdTag) is executed as soon as the "bRequest" bit is triggered with a rising edge.

// SICK RFH6XX TCP Function Block Example			
MW 16	"iCanID"	DEC	0
M 10.0	"bRequest"	BOOL	true
M 10.2	"bReqDone"	BOOL	true
M 10.3	"bReqBusy"	BOOL	false
M 10.4	"bError"	BOOL	false
MW 14	"nErrorcode"	HEX	W#16#0000
// Selection of the FB action to be execute			
M 12.1	"bTriggerOn"	BOOL	false
M 12.2	"bTriggerOff"	BOOL	false
M 12.3	"bRdTag"	BOOL	true
M 12.5	"bWrTag"	BOOL	false
M 12.7	"bInventory"	BOOL	false
M 13.0	"bLockBlock"	BOOL	false
M 13.1	"bStayQuit"	BOOL	false
M 12.4	"bComTest"	BOOL	false
M 12.6	"bFreeCommand"	BOOL	false

Figure 15: Starting the block function

The reading function is completed as soon as bit bReqDone = TRUE. The read tag content is available in the "ReadTag.arrData" array of the user data block. The "ReadTag.iDataLength" variable specifies how many bytes have been received and are valid.

// ========== Read Tag ==========				
DB73.DBW	74	"SICK RFH DATA".ReadTag.iStartBlock	DEC	0
DB73.DBW	76	"SICK RFH DATA".ReadTag.iNumBlocks	DEC	2
DB73.DBW	78	"SICK RFH DATA".ReadTag.iDataLength	DEC	8
DB73.DBB	80	"SICK RFH DATA".ReadTag.arrData[1]	CHARACTER	'S'
DB73.DBB	81	"SICK RFH DATA".ReadTag.arrData[2]	CHARACTER	'I'
DB73.DBB	82	"SICK RFH DATA".ReadTag.arrData[3]	CHARACTER	'C'
DB73.DBB	83	"SICK RFH DATA".ReadTag.arrData[4]	CHARACTER	'K'
DB73.DBB	84	"SICK RFH DATA".ReadTag.arrData[5]	CHARACTER	' '
DB73.DBB	85	"SICK RFH DATA".ReadTag.arrData[6]	CHARACTER	'A'
DB73.DBB	86	"SICK RFH DATA".ReadTag.arrData[7]	CHARACTER	'G'
DB73.DBB	87	"SICK RFH DATA".ReadTag.arrData[8]	CHARACTER	' '
DB73.DBB	88	"SICK RFH DATA".ReadTag.arrData[9]	CHARACTER	B#16#00
DB73.DBB	89	"SICK RFH DATA".ReadTag.arrData[10]	CHARACTER	B#16#00

Figure 16: Read tag content

7.2 Writing tag content

First, it is necessary to determine which transponder the system is to communicate with. If bit Mode.bMode = TRUE, then the system will communicate with the specified transponder, the UID of which must be known in advance (in this case: E0 04 01 00 06 D2 37 45).

// ========== Mode ==========				
DB73.DBX	0.0	"SICK RFH DATA".Mode.bMode	BOOL	true
DB73.DBB	2	"SICK RFH DATA".Mode.arrUID[1]	HEX	B#16#E0
DB73.DBB	3	"SICK RFH DATA".Mode.arrUID[2]	HEX	B#16#04
DB73.DBB	4	"SICK RFH DATA".Mode.arrUID[3]	HEX	B#16#01
DB73.DBB	5	"SICK RFH DATA".Mode.arrUID[4]	HEX	B#16#00
DB73.DBB	6	"SICK RFH DATA".Mode.arrUID[5]	HEX	B#16#06
DB73.DBB	7	"SICK RFH DATA".Mode.arrUID[6]	HEX	B#16#D2
DB73.DBB	8	"SICK RFH DATA".Mode.arrUID[7]	HEX	B#16#37
DB73.DBB	9	"SICK RFH DATA".Mode.arrUID[8]	HEX	B#16#45

Figure 17: Specification of transponder UID

Then, it is necessary to define what content is to be written to the tag and where it should be stored.

Start block: 0
Number of blocks: 3 (number of blocks to write)
Block size: 4 (transponder-dependent)
Data: "Hello World"

// ========== Write Tag ==========			
DB73.DBW 208	"SICK RFH DATA".WriteTag.iStartBlock	DEC	0
DB73.DBW 210	"SICK RFH DATA".WriteTag.iNumBlocks	DEC	3
DB73.DBW 212	"SICK RFH DATA".WriteTag.iBlockSize	DEC	4
DB73.DBB 214	"SICK RFH DATA".WriteTag.arrData[1]	CHARACTER	'H'
DB73.DBB 215	"SICK RFH DATA".WriteTag.arrData[2]	CHARACTER	'e'
DB73.DBB 216	"SICK RFH DATA".WriteTag.arrData[3]	CHARACTER	'l'
DB73.DBB 217	"SICK RFH DATA".WriteTag.arrData[4]	CHARACTER	'l'
DB73.DBB 218	"SICK RFH DATA".WriteTag.arrData[5]	CHARACTER	'o'
DB73.DBB 219	"SICK RFH DATA".WriteTag.arrData[6]	CHARACTER	' '
DB73.DBB 220	"SICK RFH DATA".WriteTag.arrData[7]	CHARACTER	'W'
DB73.DBB 221	"SICK RFH DATA".WriteTag.arrData[8]	CHARACTER	'o'
DB73.DBB 222	"SICK RFH DATA".WriteTag.arrData[9]	CHARACTER	'r'
DB73.DBB 223	"SICK RFH DATA".WriteTag.arrData[10]	CHARACTER	'l'
DB73.DBB 224	"SICK RFH DATA".WriteTag.arrData[11]	CHARACTER	'd'
DB73.DBB 225	"SICK RFH DATA".WriteTag.arrData[12]	CHARACTER	' '

Figure 18: Write tag parameters

The write function (bWrTag) is executed as soon as the "bRequest" bit is triggered with a rising edge.

// SICK RFH6XX TCP Function Block Example			
MW 16	"iCanID"	DEC	0
M 10.0	"bRequest"	BOOL	true
M 10.2	"bReqDone"	BOOL	true
M 10.3	"bReqBusy"	BOOL	false
M 10.4	"bError"	BOOL	false
MW 14	"nErrorcode"	HEX	W#16#0000
// Selection of the FB action to be execute			
M 12.1	"bTriggerOn"	BOOL	false
M 12.2	"bTriggerOff"	BOOL	false
M 12.3	"bRdTag"	BOOL	false
M 12.5	"bWrTag"	BOOL	true
M 12.7	"bInventory"	BOOL	false
M 13.0	"bLockBlock"	BOOL	false
M 13.1	"bStayQuit"	BOOL	false
M 12.4	"bComTest"	BOOL	false
M 12.6	"bFreeCommand"	BOOL	false

Figure 19: Starting the block function

The write function is completed as soon as bit bReqDone = TRUE.