

# **SICK LECTOR / CLV6xx Function Block**

**Version V2.X**

SICK LECTOR CLV PN CP Function Block for  
Siemens Step7 Controls



# Table of Content

<b>1 About this document</b>	<b>3</b>
1.1 Function of this document	3
1.2 Target group	3
<b>2 General Information</b>	<b>4</b>
<b>3 Hardware configuration</b>	<b>5</b>
3.1 Supported PLC controls	5
3.2 Supported Fieldbus Gateways / Sensors	5
3.3 Configuration in Step7	5
3.3.1 Hardware Configuration	6
3.3.2 Access to the E/A area	6
3.4 SOPAS device configuration	9
<b>4 Description of function block</b>	<b>11</b>
4.1 Function block specification	11
4.2 Operation Mode	12
4.3 Behavior in the case of an error	13
4.4 Timing	13
4.5 Value Transfer	14
4.5.1 Matchcode	15
4.5.2 Free Command	16
4.5.3 Reading Result	16
4.6 Receipt of read results > 200 Byte	17
<b>5 Parameter</b>	<b>19</b>
<b>6 Error Codes</b>	<b>22</b>
<b>7 Example</b>	<b>24</b>
7.1 Create / change Matchcode	25
7.2 Send trigger signal	27

## **1 About this document**

Please read this chapter carefully before you start working with this Technical Information and with SICK Lector/CLV6xx function block.

### **1.1 Function of this document**

This Technical Information describes how to use SICK LECTOR CLV6XX PN CP function block. It is used for guiding technical personnel working for the machine manufacturer / operator in project planning and commissioning.

### **1.2 Target group**

This Technical Information is aimed for specialists, such as technicians and engineers.

## 2 General Information

The function block „SICK LECTOR CLV PN CP“ is used for the communication between a SIMATIC control and a SICK Lector 2D code reader or a CLV6xx barcode reader.

The following image shows the function block in the view of the function block diagram (FBD).

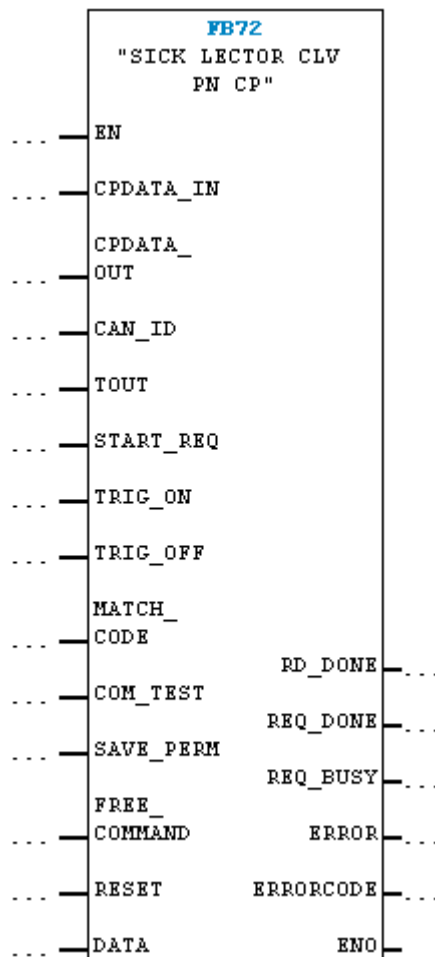


Image 1: Diagram of SICK LECTOR CLV PN CP

### Features of function blocks:

- Sending of a trigger (CoLa<sup>i</sup> command) via the PLC
- Receiving of read results (defined in SOPAS-ET<sup>ii</sup> output format)
- Create / change an evaluation condition for a matchcode
- Carrying out a communication test
- Permanent saving of all device parameters in the device
- Communication via free selectable CoLa commands (CoLa-A protocol)
- Addressing of devices which communicate via CAN-Bus

<sup>i</sup> The Command Language (CoLa) is an internal SICK protocol for the communication with SOPAS devices

<sup>ii</sup> SOPAS-ET is an engineering tool for the configuration of SICK sensors

## 3 Hardware configuration

### 3.1 Supported PLC controls

The function block must only be used with a Simatic S7 control family 300. Only controls which use a CP module as Profinet controller are being supported. Controls with integrated Profinet controller are not supported.

### 3.2 Supported Fieldbus Gateways / Sensors

The SICK sensor communicates via fieldbus (Profibus/Profinet) with the PLC. If the sensor cannot support the fieldbusses mentioned above, Gateway modules can be used.

The following Gateways / Sensors are supported from the function block:

- CDM 425 (Profinet), starting with firmware version V3.31
- CDF 600 (Profibus), starting with firmware version V1.15
- CDM 420 incl. CMF400 Profibus Module, starting with firmware version V1.100
- CLV6xx (Profinet), starting with firmware version V5.11

Necessary CLV6xx firmware version:

- CLV6xx (Profibus), starting with firmware version V4.00

Necessary LECTOR firmware version:

- LECTOR620, starting with firmware version V1.40

### 3.3 Configuration in Step7

Before the function block can be used, the corresponding sensor or the corresponding gateway has to be projected in the hardware configuration in Step7. Therefore the corresponding device file (GSD file) has to be imported in the hardware library in Step7.

The function block is laid out especially for the handshake mode (Confirmed Messaging Protocol). Please do only use HS-Modules with a length between 8...128 Bytes. The used addresses can be projected in the periphery or outside. An address assignment on the periphery to which a partly process image with OB6x-connection (alarm of asynchronous trigger) is assigned, must not be used.

### 3.3.1 Hardware Configuration

Image 2 shows an example configuration with a CDM425 Profinet Gateway.

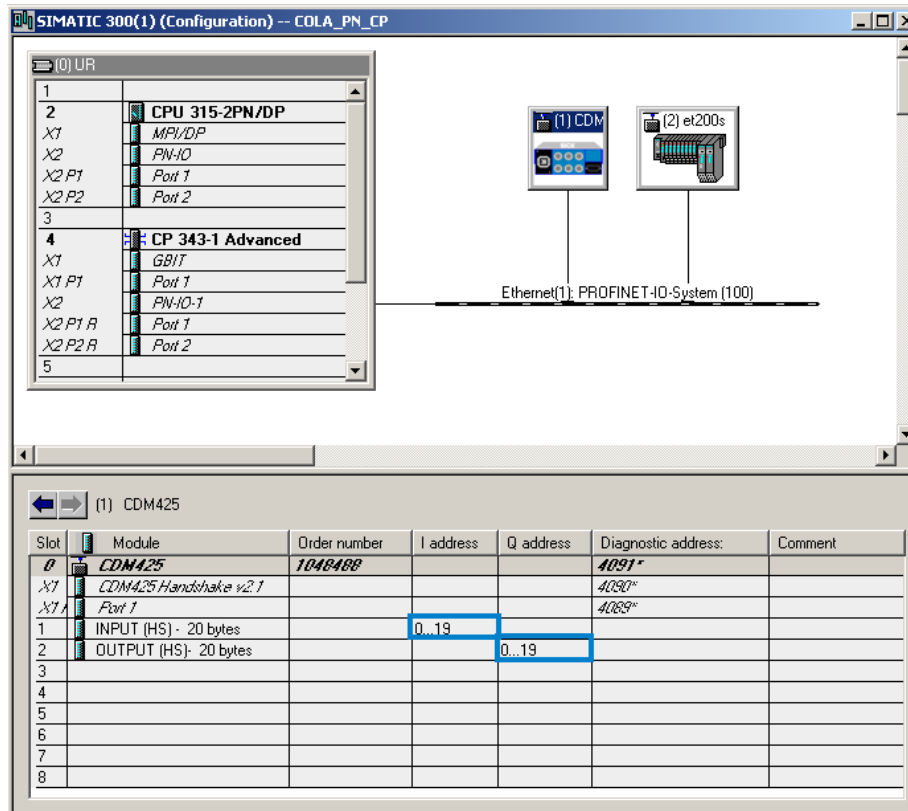


Image 2: Step7 Hardware configuration

Please take notice that the E/A-Addresses of the CP-Module are not identical with the E/A-Addresses of the CPU, since the CP-Module has an own address area. Therefore it is not possible to communicate directly out of the S7-Program with the Profinet participant.

The access to the complete E/A area of the CP-Module (here participants SICK CDM425 and Siemens ET200S) is realized via the functions FC11 (PNIO\_SEND) and FC12 (PNIO\_RECV). These functions make a consistent E/A image of all devices connected to the CP module.

In order to use Siemens FCs it is necessary, that the projected E/A areas of the connected periphery are configured coherently and that all start with address 0.

### 3.3.2 Access to the E/A area

The E/A area of the connected participants should be read out or written at each PLC cycle. In this example the functions FC11/FC12 are called up cyclically in OB1.

The function FC12 (PNIO\_RECV) has to be configured in the following way:

Technical Information

CPLADDR: Hardware address of the projected CP-Module (see Image 4)  
 MODE: 0 = IO-Controller Mode  
 LEN: Byte length of all entry data starting with address 0.  
       CDM425 (0..19) = 20 Byte  
       ET200S (20..23) = 4 Byte  
       Total = 24 Byte  
 RECV: Pointer onto the data area (DB) in which the incoming data should be stored.  
 IOPS: Pointer onto the data area (DB) in which the IOPS (IO Producer Status)  
       should be stored. The length of the data area for the IOPS data has to be LEN  
       /8 (24/8= 3 Byte) long.

Network 2 : RECEIVE DATA

Receive the whole CP-PN data via FC12

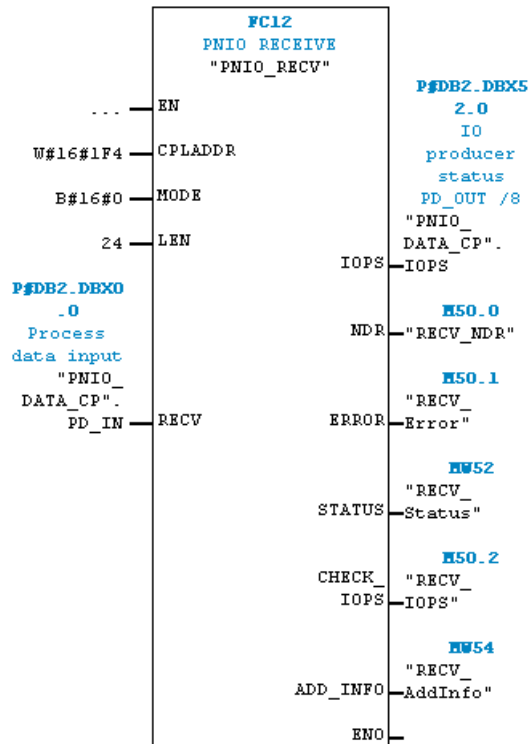


Image 3: Call up FC12 (PNIO\_RECV) in OB1

Image 4 shows where you can find the hardware address of the projected CP-Module in the Simatic hardware configuration.

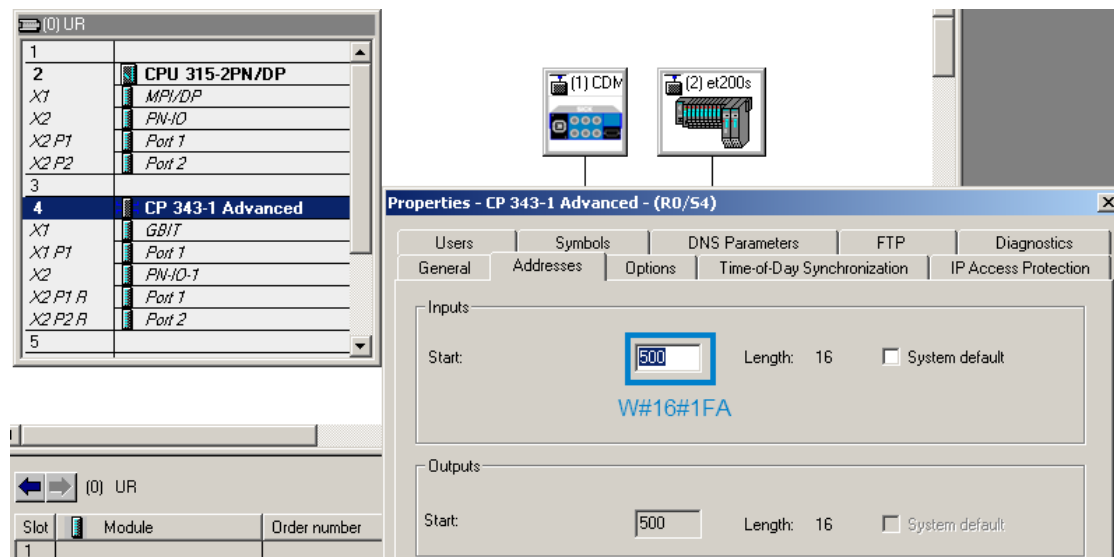


Image 4: Hardware address of the projected CP-Module



The function FC11 (PNIO\_SEND) has to be configured in the following way:

CPLADDR: Hardware address of the projected CP-Module (see Image 4)  
 MODE: 0 = IO-Controller Mode  
 LEN: Byte length of all output data starting with address 0.  
       CDM425 (0..19) = 20 Byte  
       ET200S (20..23) = 4 Byte  
       Total = 24 Byte  
 SEND: Pointer onto the data area (DB) in which the output data has to be stored.  
 IOCS: Pointer onto the data area (DB) in which the IOCS (IO Consumer Status) has to be stored. The length of the data area for IOCS data has to be LEN / 8 (24/8=3 Byte) long.

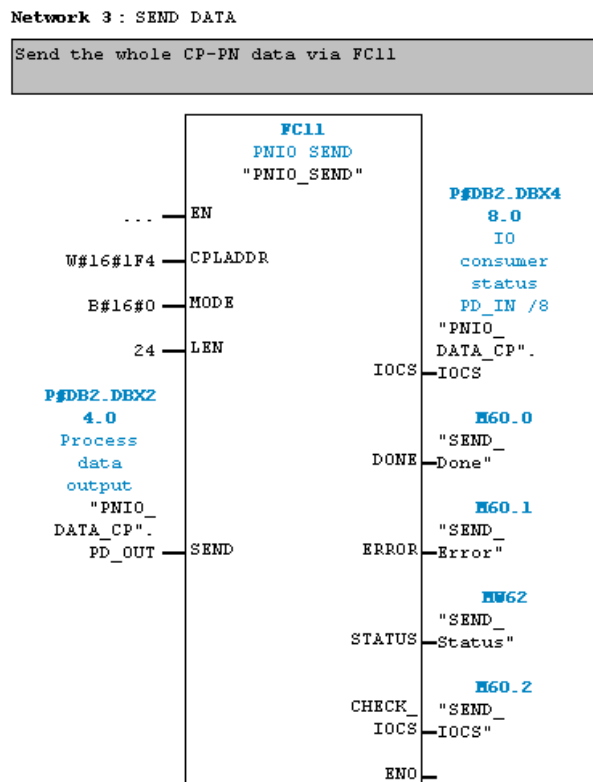


Image 5: Call-up FC11 (PNIO\_SEND) in OB1

### 3.4 SOPAS device configuration

That a trigger via the PLC can take place, the trigger source has to be set with SOPAS-ET to „command“ before. Image 6 shows how to configure CLV6xx or Lector at the menu point „Object Trigger“.

- Start with "SOPAS command" (TRIG\_ON command has to be used)
  - Stop with "SOPAS command" (TRIG\_OFF command has to be used)
- Optionally the trigger window can automatically be closed, if the sensor has read a code with „Good Read“ or, in the case of a „No Reads“, after a defined timeout (here 500ms).

Image 6: SOPAS Trigger Settings

If the device should be triggered directly, e.g. via a light switch or a hardware signal at the Sensor1-entry of LECTOR/CLV, the function blocks TRIG\_ON / TRIG\_OFF cannot be used anymore. If a trigger result has been received from the function block, this is signaled via the output parameter „RD\_DONE“.

The read results always have to be differentiated to the SOPAS commands. If the read result can start with „s“ (lower case character „s“ as a SOPAS command), the output format has to be changed by placing e.g. a „D“ in front.

Image 7: SOPAS output format settings

## 4 Description of function block

The function block is working asynchronously, which means the processing is done via various function block call ups. Therefore it is necessary that the function block is called up cyclically in the user program.

The function block encapsulates „SICK CCOM PN CP“ (FB14), which allows the communication between PLC and sensor.

### 4.1 Function block specification

Number of function block:	FB72
Name of function block:	SICK LECTOR CLV PN CP
Version:	2.0
Called up function blocks:	SFC20 (BLKMOV) SFB4 (TON) FB14 (SICK CCOM PN CP)
Used data blocks:	DB72 (SICK LECTOR CLV DATA)
Function block call up:	Cyclically
Used flag:	no
Used counter:	none
Used register:	AR1, AR2 (for multi instance call up)
Multi instant capable:	yes
Language:	Step7-AWL
Step7 Version:	Simatic Step7 V5.5

The system functions (SFCs) used in the function block have to be available on the respective PLC.

When changing the function block numbers, the respective calls in the function block SICK LECTOR CLV PN CP have to be updated.

## 4.2 Operation Mode

In order to use the function block, the following communication parameters have to be set:

CPDATA\_IN: Pointer on the input data of the sensor/gateway. The input data has to be got before with the function FC12 (PNIO\_RECV).

CPDATA\_OUT: Pointer on the output data of the sensor/gateway. The output data has to be transferred before to the device by using the function FC12 (PNIO\_SEND).

DATA: The data block (DB72) belonging to the function block contains in- and output parameter of the supported function block actions. The data block has to be transferred to the input parameter „DATA“ of the function block.

### Realizable function block functions:

- Trigger on → Opens the reading gate of the device per CoLa command
- Trigger off → Closes the reading gate of the device per CoLa command
- Match Code → Creates / changes a new evaluation condition for a matchcode
- Communication test → Checks if the device can be reached via „sRI0“ command
- Save Permanent → Saves all device parameters permanently in the device.
- Free Command → Executes a free selectable CoLa command
- Reset → Resetting the device communication

In order to execute a function block action (TRIG\_ON, TRIG\_OFF, etc.), the desired action has to be selected first. Only one action can be executed at the same time. In order to do the action, the parameter START\_REQ has to be triggered with a positive edge (signal change from a logical zero to one). As long as no valid device answer has to be received, this is signalized via the parameter REQ\_BUSY.

If the function block signalizes REQ\_DONE = TRUE at the output parameter, the action has been done successfully. If, for this action (e.g. FREE\_COMMAND) data has been requested from the device, it will be copied in the respective data area of the UDTs.

Data that is sent per trigger (TRIG\_ON, TRIG\_OFF) or directly from the device (e.g. direct trigger via a light switch), is stored in the data function block (ReadingResult.arrResult). The output parameter RD\_DONE indicates for one PLC cycle, that new data has been received. The from the device sent data can be changed in the SOPAS output format (see chapter 3.4).

### 4.3 Behavior in the case of an error

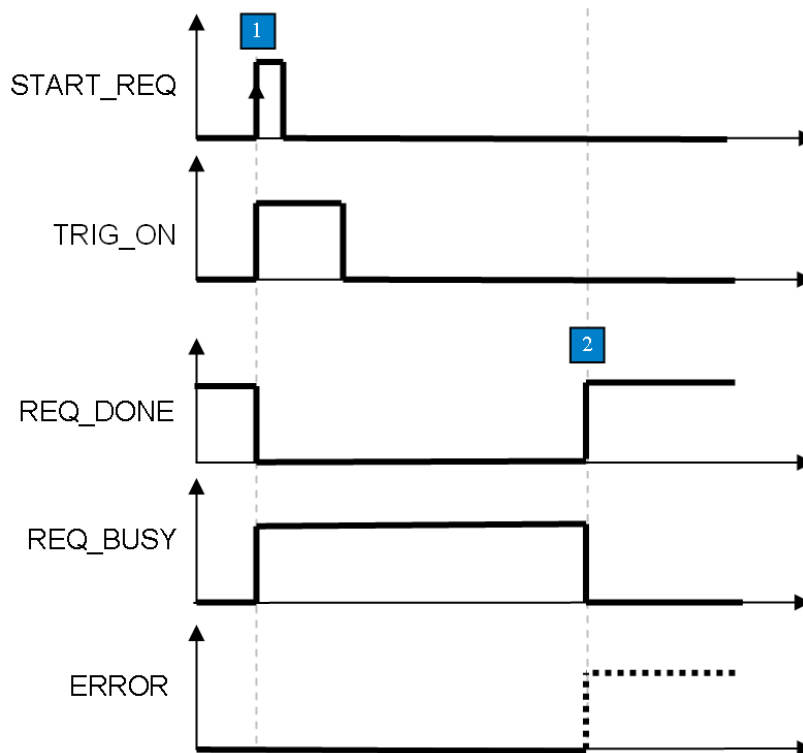
If there is a wrong input value or a wrong input circuit of the function block, an error bit (*bError*) is set and an error code (*iErrorcode*) will be given out. In this case there is no further processing. The diagnosis parameter (*bError* and *iErrorcode*) of the routine maintain their value until a new request has been started.

Via the RESET Bit the communication between the sensor and the PLC can be reset. The reset order is carried out as soon as the RESET Bit is chosen and the START\_REQ Bit is triggered with a positive edge (signal change from zero to one). The bit REQ\_BUSY signalizes that the order is executed. As soon as the reset routine is terminated, the bit REQ\_DONE is set.

Because of the reset the following actions are carried out:

- Reset of the counter from confirmed messaging protocol (device communication)
- Reset of all error information

### 4.4 Timing



1: Request via Pos edge to bRequest

The desired action (here *bTriggerOn*) has to be selected in advance / at the same time. Only one action must be selected at the same time, otherwise there is a break down with *bError = TRUE*.

2: If all commands are sent and all replies are received, the action is ended with *bReqDone = TRUE*. If the action is faulty, it will be terminated with *bError = TRUE*. If terminated with *bError*, you can find the error in *iErrorcode*.

## 4.5 Value Transfer

The data function block „SICK LECTOR CLV DATA“ (DB72) contains input and output parameters of all supported function block actions. The data function block can be re-named according to the user program. The data structure is pre-defined and must not be changed (except for the last entry (ReadingResult.arrResult), (see chapter 4.6: Receipt of read results Receipt of read results > 200 Byte).

Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	Matchcode	STRUCT		-- MATCH CODE --
+0.0	nMatchNumber	BYTE	B#16#1	Matchcode number (Match1..9) (IN)
+1.0	nCodeType	CHAR	' '	Codetype see device docu. (Example: 'd' = EAN-Code; '*' = Don't care) (IN)
+2.0	nLength	BYTE	B#16#0	Sets min and max length. B#16#0 = Don't care (IN)
+4.0	iContentLength	INT	0	Content length (IN)
+6.0	arrContent	ARRAY[1..75]		Matchcode content (IN)
+1.0		CHAR		
+82.0		END_STRUCT		
+82.0	FreeCommand	STRUCT		-- FREE COMMAND --
+0.0	iCommandLength	INT	0	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		
+286.0	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		
+490.0		END_STRUCT		

Image 8: Structure of the SICK LECTOR CLV DATA

### 4.5.1 Matchcode

With the help of the matchcode action you have the possibility to create a new evaluation condition or to change an existing one. Before the matchcode action can be carried out, the following parameters have to be set in the structure *Matchcode*.

Parameter	Declaration	Data type	Description
Matchcode. iMatchNumber	Input	BYTE	The matchcode number defines the name of the evaluation condition (e.g. 1=Match1, 2=Match2).  Valid value area: [1..9]
Matchcode. CodeType	Input	CHAR	Desired code type which the evaluation condition should refer to (e.g. 'w'= Data matrix, 's'= QR-Code, '*' = any code type).  A selection of all code types can be found in the device documentation.
Matchcode. nLength	Output	BYTE	Minimal and maximal code length 0 = any code length
Matchcode. iContentLength	Output	INT	Byte length of the given matchcode
Matchcode. arrContent	Output	ARRAY [1..75] OF CHAR	Matchcode content

Table 1: Matchcode Parameter

### 4.5.2 Free Command

With the help of a free command you have the possibility to communicate via a valid CoLa command with the device. Hence it is necessary to store the command in the parameter *sCommand* of the structure *FreeCommand*. The character length of the transferring command is written in the parameter „iCommandLength“. The commands can be looked up in the device description or SOPAS-ET.

Parameter	Declaration	Data type	Description
FreeCommand. iCommandLength	Input	INT	Character length of the transferring CoLa command.  Valid value area: [1..100]
FreeCommand. arrCommand	Input	ARRAY [1..100] OF CHAR	Free selectable CoLa command (Commands can be looked up in the device documentation).
FreeCommand. iResultLength	Output	INT	Byte length of the receiving CoLa telegram.
FreeCommand. arrResult	Output	ARRAY [1..100] OF CHAR	Receiving answer of the sent CoLa telegram.

Table 2: Free Command Parameter

### 4.5.3 Reading Result

In the data string *ReadingResult.sResult* data is stored, which is sent via trigger order (*bTriggerOn*, *bTriggerOff*) or directly from the device (e.g. direct trigger via a light switch or fieldbus). The output parameter *bRdDone* signalsizes whether the data has been received.

Parameter	Declaration	Data type	Description
ReadingResult. nCounter	Output	BYTE	The receipt counter is incremented by one as soon as a new read result has been received.  Value area: [0x00..0xFF]
ReadingResult. iLength	Output	INT	Byte length of the receiving read result.
ReadingResult. arrResult	Output	ARRAY [1..200] of BYTE	Receiving answer of a trigger signal (can be defined via the SOPAS output format).  The maximal length of the receiving data is 200 Bytes. Chapter 4.6 describes the procedure when receiving longer data telegrams.

Table 3: Reading Result Parameter



## 4.6 Receipt of read results > 200 Byte

The function block is laid out to receive read results up to a length of 200 Bytes. If longer data has to be received, the routine has to be changed at the following positions:

### Change in SICK LECTOR CLV DATA UDT:

In the delivered UDT (DB72) the length of the array „ReadingResult.arrResult“ has to be adapted in such a way, that the read result which has 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		

Image 9: Receipt of read results > 200 Bytes (Change in the UDT)

### Change in SICK LECTOR CLV PN CP function block:

In the static area of the variable survey, the length of the variable „arrRecord“ has to be adapted in such a way, that the read result fits into the data area of the variable. The array must not be below a length of 250 bytes, but it has to be larger or equal to the length of the ReadingResult.arrResult.

Contents Of: 'Environment\Interface\STAT'			
	Name	Data Type	Address
Interface	iReqLength	Int	42.0
	arrCommand	Array [1..250] Of Byte	44.0
	arrRecord	Array [1..250] Of Byte	294.0
	fbCCOM	SICK CCOM PN CP	544.0

Image 10: Receipt of read results > 200 Bytes (change in FB declaration)

The new defined lengths of the array have to be put into network 3 of SICK LECTOR CLV PN CP FBs.

```

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 250
T #iArrayRecLen          #iArrayRecLen

/-- LENGTH OF THE COMMAND ARRAY
L 250
T #iArrayComLen          #iArrayComLen

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

/-- FRAMING
SET
= #bAddFraming           #bAddFraming

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

```

Image 11: Receipt of read results > 200 Bytes (change in function block code)

After the change the instance of the function block has to be actualized. Afterwards the changed UDT and the function block have to be transferred together with the updated instance to the PLC.

## 5 Parameter

Parameter	Declaration	Data type	Storing area	Description
EN	INPUT	BOOL	E,M,D,L, Konst.	Enable input (KOP and FUP)
CPDATA_IN	INPUT	ANY	D	<p>Pointer on the input area of the sensor / gateway. Only one data type Byte is allowed.</p> <p><u>Hint:</u> Please note that the parameter always needs the complete info of the DB – parameter. (Example: P#DB13.DBX0.0 BYTE 100). Leaving out an explicit DB-No. is not allowed and leads to a function block error.</p>
CPDATA_OUT	INPUT	ANY	D	<p>Pointer on the output area of the sensors / gateway. Only one data type Byte is allowed.</p> <p><u>Hint:</u> Please note that the parameter always needs the complete info of the DB – parameter. (Example: P#DB13.DBX0.0 BYTE 100). Leaving out an explicit DB-No. is not allowed and leads to a function block error.</p>
CAN_ID	INPUT	INT	E,M,D,L, Konst.	<p>CAN-ID of the sensor to be contacted.</p> <p>If no CAN-network is used, the CAN-ID = 0.</p> <p>The master resp. the multiplexer is always contacted with CAN-ID = 0, even if another CAN-ID is assigned.</p>
TOUT	INPUT	TIME	E,M,D,L, Konst.	Time after which a timeout error is provoked.
START_REQ	INPUT	BOOL	E,M,D,L	<p>Positive edge: Carrying out the selected function block action.</p>
TRIG_ON	INPUT	BOOL	E,M,D,L, Konst.	Function block action: Carrying out a device trigger (open trigger window)

Parameter	Declaration	Data type	Storing area	Description
TRIG_OFF	INPUT	BOOL	E,M,D,L, Konst.	Function block action: Carrying out a device trigger (close trigger window)  The from the device sent result (SOPAS output format) is stored in the variable „ReadingResult.sResult“ of the transferring data structure (DB72).
MATCH_CODE	INPUT	BOOL	E,M,D,L Konst.	Creating a matchcode condition.  This requires that the parameters described in chapter 4.5.1 are set in the structure (Matchcode).
COM_TEST	INPUT	BOOL	E,M,D,L, Konst.	Function block action: Carrying out a communication test.  REQ_DONE= TRUE: Communication OK  REQ_DONE= FALSE: Communication not OK
SAVE_PERMANENT	INPUT	BOOL	E,M,D,L, Konst.	Permanent saving of all device parameters in the device.
FREE_COMMAND	INPUT	BOOL	E,M,D,L, Konst.	Function block action: Carrying out a free command.  This action requires a valid CoLa command in the data structure (FreeCommand.sCommand) (see chapter 4.5.2).  After a successful transfer (bReqDone=TRUE) the command reply is available in the data structure.
RESET	INPUT	BOOL	E,M,D,L, Konst.	Function block action: Reset of the communication to the device.
DATA	INPUT	BLOCK_DB	Konst.	Transfer of the respective UDT which is necessary for the configuration of the function block and for storing the read results (DB72).
RD_DONE	OUTPUT	BOOL	A,M,D,L	Positive edge: New read result is received. The content of the read result can be configured with SOPAS-ET (see chapter 3.4).
REQ_DONE	OUTPUT	BOOL	A,M,D,L	Indicates if the chosen function block action can be carried out without error.  TRUE: processing terminated FALSE: processing not terminated
REQ_BUSY	OUTPUT	BOOL	A,M,D,L	Request is in process.

Parameter	Declaration	Data type	Storing area	Description
ERROR	OUTPUT	BOOL	A,M,D,L	Error Bit:  0: No error 1: Break-off with error
ERROR CODE	OUTPUT	WORD	A,M,D,L	Error status (see error codes)
ENO	OUTPUT	BOOL	A,M,D,L	Enable output (KOP and FUP)

*Table 4: Function block parameters*

## 6 Error Codes

The parameter ERRORCODE contains the following error information:

Error code	Short description	Description
W#16#0000	No error	No error
W#16#0001	Timeout error	Order has not been finished within the chosen timeout.  This could be because of: - Device is not connected with PLC - CAN-Bus participant is not available - Wrong communication parameter
W#16#0002	Internal function block error	Internal function block error
W#16#0003	No or more than one function block action selected	Only one function block action can be carried out at the same time
W#16#0004	Received read result > Reading Result Array	The received read result is longer than 200 Bytes. For the receipt of longer read results, please have a look at chapter 4.6
W#16#0005	100 < FreeCommand. iCommandLength <=0	Invalid length of the free command  Valid value area: [1...100]
W#16#0006	Answer of the free command > 100 Byte	The answer to the sent free command is longer than 100 Byte.
W#16#0007	63 < CAN_ID < 0	Invalid CAN-ID  Valid value area: [0..63]
W#16#0008	Reserved	Reserved
W#16#0009	Communication error	Communication to the device cannot be realized.  This could be because of: - Invalid length of E/A data - A telegram > arrRecord has been received
W#16#XX0A	Device error	A device error has come up ('sFA XX')  <b>XX</b> = device error (see device documentation)
W#16#000B	Invalid Command answer	The selected action has not been carried out.  This could be because of: - Wrong trigger setting in the SOPAS object - Device is not in „Run-Mode“
W#16#000C – W#16#000F	Reserved	Reserved

Error code	Short description	Description
W#16#0010	9 < Matchcode.nMatchNumber <= 0	Invalid Matchcode number.  Valid value area: [1..9]
W#16#0011	75 < Matchcode. iLength <= 0	Invalid Matchcode length.  Valid value area: [1..75]
W#16#XX12	Matchcode / Save Permanently have not been carried out.	The selected action has not been carried out. The device has put into „RUN-Mode“ once again.  <b>XX</b> = Error code of the shown errors
W#16#0013	Change into „RUN-Mode“ is not possible	The device cannot be put back into the „RUN-Mode“.  This could be because of: - Command needs more time than the set Timeout of the routine.

Table 5: Error codes

## 7 Example

Image 12 shows an example of a circuit of SICK LECTOR CLV FBs. In the hardware configuration, a SICK device with a process data width of 20 Byte Input/Output is projected. The E/A-area of the device is described by the functions FC11/FC12 (see chapter 3.3). Since the device is not in a CAN network, a zero is put as CAN-ID.

For the trigger result (defined in SOPAS Output format) 200 Bytes are reserved in DB72. If the read result is longer, this is indicated by an error at the function block.

### Program selection:

```

Network 1: AUFRUF LECTOR FB | CALL LECTOR FB

Aufwurf des LECTOR/CLV Funktionsbausteins (Profinet-Anbindung über CP-Modul)
---
Call of the LECTOR/CLV function block (Profinet-Connection via CP-Module)

CALL "SICK LECTOR CLV PN CP" , "INSTANCE_FB72"

CPDATA_IN  := "PNIO_DATA_CP".PD_IN
CPDATA_OUT := "PNIO_DATA_CP".PD_OUT
CAN_ID     := "iCanID"
TOUT       := T#10S
START_REQ  := "bRequest"
TRIG_ON    := "bTriggerOn"
TRIG_OFF   := "bTriggerOff"
MATCH_CODE := "bMatchCode"
COM_TEST   := "bComTest"
SAVE_PERM  := "bSavePermanent"
FREE_COMMAND := "bFreeCommand"
RESET      := "bReset"
DATA       := "SICK LECTOR CLV DATA"
RD_DONE    := "bRdDone"
REQ_DONE   := "bReqDone"
REQ_BUSY   := "bReqBusy"
ERROR      := "bError"
ERRORCODE  := "nErrorcode"

FB72 / DB172
P#DB2.DBX0.0
P#DB2.DBX20.0
MW16

M10.0
M12.1
M12.2
M12.3
M12.4
M12.5
M12.6
M12.0
DB72
M10.1
M10.2
M10.3
M10.4
MW14

```

Image 12: Example of a circuit of FB SICK LECTOR CLV function block

Slot	Module	Order number	I address	Q address	Diagnostic address:
0	CDM425	1048488			4091"
X1	CDM425 Handshake v2.1				4090"
X1A	Port 1				4089"
1	INPUT (HS) - 20 bytes		0...19		
2	OUTPUT (HS) - 20 bytes			0...19	

Image 13: Step7 Projecting of hardware



## 7.1 Create / change Matchcode

In order to create a new matchcode evaluation condition or to change an existing one, the necessary parameter values have to be put first.

nMatchNumber: Matchcode Number (here: Match5)  
 nCodeType: Code Type ("\*" = All Code Types)  
 nLength: Length of the codes  
 iContentLength: Code content

// ===== Match Code =====

DB72.DBB	0	"SICK LECTOR CLV6XX DATA".Matchcode.nMatchNumber	DEC	5
DB72.DBB	1	"SICK LECTOR CLV6XX DATA".Matchcode.nCodeType	CHARACTER	'*'
DB72.DBB	2	"SICK LECTOR CLV6XX DATA".Matchcode.nLength	DEC	5
DB72.DBB	4	"SICK LECTOR CLV6XX DATA".Matchcode.iContentLength	DEC	4
DB72.DBB	6	"SICK LECTOR CLV6XX DATA".Matchcode.arrContent[1]	CHARACTER	'A'
DB72.DBB	7	"SICK LECTOR CLV6XX DATA".Matchcode.arrContent[2]	CHARACTER	'B'
DB72.DBB	8	"SICK LECTOR CLV6XX DATA".Matchcode.arrContent[3]	CHARACTER	'C'
DB72.DBB	9	"SICK LECTOR CLV6XX DATA".Matchcode.arrContent[4]	CHARACTER	'*'
DB72.DBB	10	"SICK LECTOR CLV6XX DATA".Matchcode.arrContent[5]	CHARACTER	' '

Image 14: Matchcode Parameter

The matchcode action *bMatchcode* is carried out as soon as the bit *bRequest* is triggered with a positive edge.

// SICK Lector / CLV Function Block Example

M/W	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
M/W	14	"nErrorcode"	HEX	Vv#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	"bMatchCode"	BOOL	true
M	12.4	"bComTest"	BOOL	false
M	12.5	"bSavePermanent"	BOOL	false
M	12.6	"bFreeCommand"	BOOL	false

Image 15: Start Matchcode action

The Matchcode action is terminated as soon as the bit *bReqDone* = *TRUE*. In this example the following evaluation condition is put onto the device:

**Match code condition**

Condition type: Match-Code Condition

Name: Match5

Condition state: Code related

Code content: ABC\* >

☒ Wildcards (? and \*) ☐ Regular expression Test...

Code length: min: 5 max: 5 ☐ Don't care

Code type: Codabar ☒ Don't care

Code validity: ☒ Match only valid codes ☐ Match all codes

Restrict to devices with ID: ☒ Don't care

☐ Invert condition

☐ Deactivate condition As "false"

OK Cancel

Image 16: Put evaluation condition

## 7.2 Send trigger signal

That a trigger via the PLC can take place, the trigger source has to be set before to „command“ via SOPAS-ET.

In this example the reading gate can be opened and closed via the function block. Optionally, the reading gate will be closed automatically in the case of a „Good Read“.

Image 17: SOPAS Object trigger settings

This is done, as soon as the bit „bRequest“ is triggered with a positive edge. The reading gate is open as soon as „bReqDone = TRUE“.

// SICK Lector / CLV Function Block Example			
M/W 16	"iCanID"	DEC	0
M 10.0	"bRequest"	BOOL	<input checked="" type="checkbox"/> true
M 10.2	"bReqDone"	BOOL	<input checked="" type="checkbox"/> true
M 10.3	"bReqBusy"	BOOL	<input type="checkbox"/> false
M 10.4	"bError"	BOOL	<input type="checkbox"/> false
M/W 14	"nErrorcode"	HEX	VW#16#0000
// Selection of the FB action to be execute			
M 12.1	"bTriggerOn"	BOOL	<input checked="" type="checkbox"/> true
M 12.2	"bTriggerOff"	BOOL	<input type="checkbox"/> false
M 12.3	"bMatchCode"	BOOL	<input type="checkbox"/> false
M 12.4	"bComTest"	BOOL	<input type="checkbox"/> false
M 12.5	"bSavePermanent"	BOOL	<input type="checkbox"/> false
M 12.6	"bFreeCommand"	BOOL	<input type="checkbox"/> false

Image 18: Open reading gate

If the code has been read successfully („Good Read“), the device closes the reading gate automatically and sends the read code to the PLC. The function block saves the read code in the array „ReadingResult.arrResult“ of the UDT (DB72). The parameter RD\_DONE indicates

for one PLC cycle, that new data has been received. The parameter ReadingResult.iLength indicates, how many bytes have been received resp. how many are valid.

```
// ===== Reading Result =====
```

DB72.DBB	286	"SICK LECTOR CLV6XX DATA".ReadingResult.iCounter	HEX	B#16#18
DB72.DBB	288	"SICK LECTOR CLV6XX DATA".ReadingResult.iLength	DEC	16
DB72.DBB	290	"SICK LECTOR CLV6XX DATA".ReadingResult.arrResult[1]	CHARACTER	'I'
DB72.DBB	291	"SICK LECTOR CLV6XX DATA".ReadingResult.arrResult[2]	CHARACTER	'S'
DB72.DBB	292	"SICK LECTOR CLV6XX DATA".ReadingResult.arrResult[3]	CHARACTER	'I'
DB72.DBB	293	"SICK LECTOR CLV6XX DATA".ReadingResult.arrResult[4]	CHARACTER	'C'
DB72.DBB	294	"SICK LECTOR CLV6XX DATA".ReadingResult.arrResult[5]	CHARACTER	'K'
DB72.DBB	295	"SICK LECTOR CLV6XX DATA".ReadingResult.arrResult[6]	CHARACTER	' '
DB72.DBB	296	"SICK LECTOR CLV6XX DATA".ReadingResult.arrResult[7]	CHARACTER	'T'
DB72.DBB	297	"SICK LECTOR CLV6XX DATA".ReadingResult.arrResult[8]	CHARACTER	'e'
DB72.DBB	298	"SICK LECTOR CLV6XX DATA".ReadingResult.arrResult[9]	CHARACTER	's'
DB72.DBB	299	"SICK LECTOR CLV6XX DATA".ReadingResult.arrResult[10]	CHARACTER	't'
DB72.DBB	300	"SICK LECTOR CLV6XX DATA".ReadingResult.arrResult[11]	CHARACTER	' '
DB72.DBB	301	"SICK LECTOR CLV6XX DATA".ReadingResult.arrResult[12]	CHARACTER	'C'
DB72.DBB	302	"SICK LECTOR CLV6XX DATA".ReadingResult.arrResult[13]	CHARACTER	'o'
DB72.DBB	303	"SICK LECTOR CLV6XX DATA".ReadingResult.arrResult[14]	CHARACTER	'd'
DB72.DBB	304	"SICK LECTOR CLV6XX DATA".ReadingResult.arrResult[15]	CHARACTER	'e'
DB72.DBB	305	"SICK LECTOR CLV6XX DATA".ReadingResult.arrResult[16]	CHARACTER	'L'
DB72.DBB	306	"SICK LECTOR CLV6XX DATA".ReadingResult.arrResult[17]	CHARACTER	' '

Image 19: Read result