Application Note

FESTO

CECC-LK as IO-Link Master connected to Balluff RFID Device and data carriers

The application note contains a step by step explanation how to configure a Balluff RFID read/write system as IO-Link device connected to CECC-LK in Codesys V3

100126

Title ..... CECC-LK as IO-Link Master connected to Balluff RFID Device and data carriers
Version 1.10
Document no. 100126
Original ..... en
Author ..... Festo

Last saved 01.12.2016

Copyright Notice

Legal Notice

Internet: http://www.festo.com
E-Mail: service_international@festo.com

Table of contents

1 Components/Software/ IP address used 4

1.1 Recommended manuals / IODD 4

1.2 Topology 5

1.3 Wiring 5

2 Configuring the IO-Link master 7

2.1 IO-Link configuration 7

2.2 Configuration of Parameters and Testing 9

2.3 Programming in Codesys 12

2.3.1 Program to read parameter by coding 12

2.3.2 Program to write parameter by coding 13

2.3.3 Read stored data from data carrier 15

1 Components/Software/ IP address used

Type/Name	Version Software/Firmware	IP address	Subnet mask
CECC-LK	V 1.4.0.1	192.168.0.20	255.255.0.0
Laptop	--	192.168.0.100	255.255.0.0
Codesys V3.5	SP7 Patch 4	--	--
BIS0103 BIS M-451-072-001-07-S4	V1.70	--	--
BIS0045 BIS M-111-02/L	--	--	--
BIS0046 BIS M-112-02/L	--	--	--

Table 1.1: 1 Components/Software used

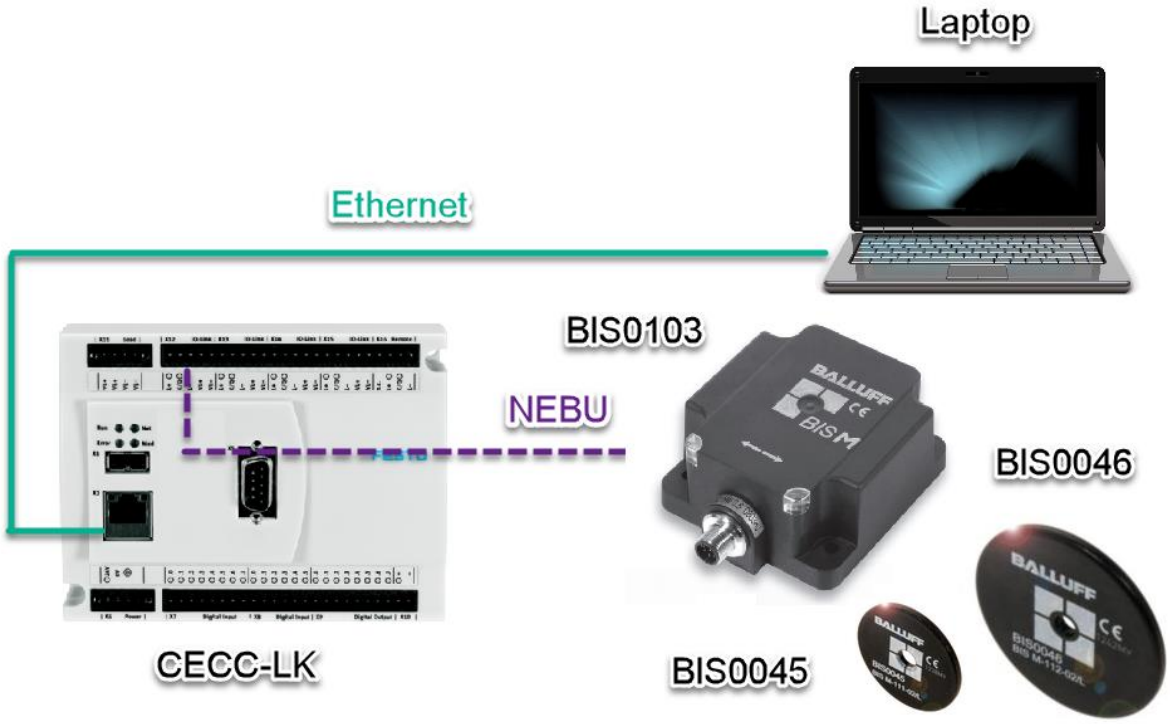
1.1 Recommended manuals / IODD

CECC manual:
https://www.festo.com/net/SupportPortal/Files/407042/CECC_2014-03a_8036062g1.pdf
Target Support:
https://www.festo.com/net/en-gb_gb/SupportPortal/Downloads/415525/443515/CECC_3.5.7.159(ad778b5e1029).package

BIS0103, BIS M-451-072-001-07-S4 manual:
http://asset.balluff.com/std.lang.all/pdf/binary/870554_000_04_DOK.pdf
Datasheet:
http://asset.balluff.com/std.lang.all/pdf/datasheet/6_/gl/Datasheet_BIS0103_228506_GL.pdf
IODD file:
http://asset.balluff.com/std.lang.all/zip/binary/918408_000_00_DRF.zip

NEBU catalog:
https://www.festo.com/net/en-gb_gb/SupportPortal/Downloads/272418/208021/nebu_en.pdf

1.2 Topology

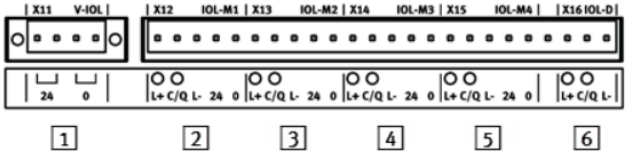


1.3 Wiring

IO-Link is a point to point communication (1 Master -› 1 Slave). In our example, we use port 1 of IO-Link master from CECC-LK. Festo has the NEBU cable series to connect to the IO-Link device and master. To get the correct wiring, please check the following pin assignments.

CECC-LK:

3.5.8 IO-Link on the CECC-LK



1 IO-Link load voltage supply X11

2...5 IO-Link master ports for connecting IO-Link devices X12 ... X15

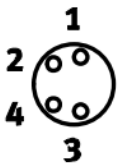
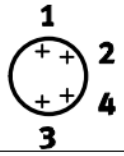
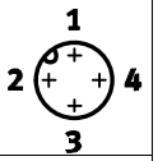
6 IO-Link device port for connecting the CECC-LK as an IO-Link device X16

Figure: CECC-LK, top manifold rail with IO-Link interfaces

IO-Link	Pin	Signal	Comment
1	X11.1	24	Connection for load voltage supply for IO-Link master ports[1]: UA+
	X11.2		
	X11.3	0	Connection for load voltage supply for IO-Link master ports[1]: UA– (GND)
	X11.4		
2... 5	X12...15.1	L+	24 V
	X12...15.2	C/Q	IO-Link communication signal
	X12...15.3	L–	0 V
	X12...15.4	24	UA+
	X12...15.5	0	UA–
6	X16.1	L+	24 V
	X16.2	C/Q	IO-Link communication signal
	X16.3	L–	0 V
1) Port class B			

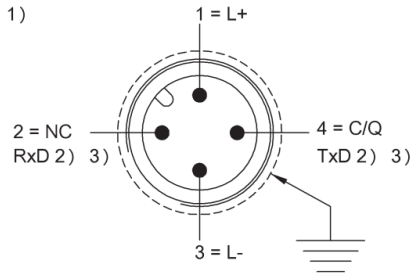
Table: Pin allocation of the IO-Link interfaces on the CECC-LK

NEBU:

Electrical connection: socket, 4-pin, M8 – plug, 4-pin				
	1	BN	1	Plug M8  Plug M12 
	2	WH	2	
	3	BU	3	
	4	BK	4	

BIS0103:

BIS M-451-072-001-07-S4
BIS0103

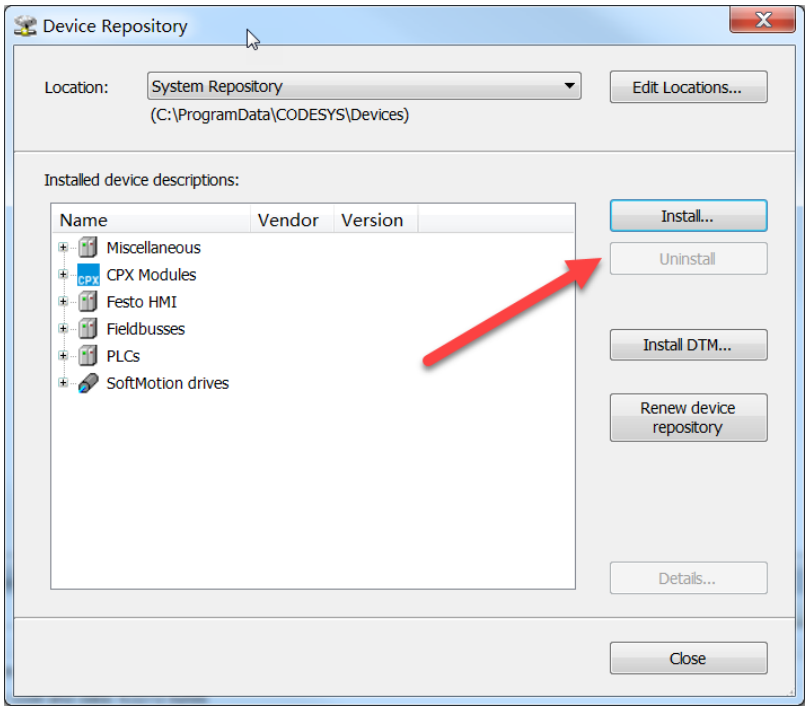


1) View towards connector
2) Service
3) (Only for Balluff Service)

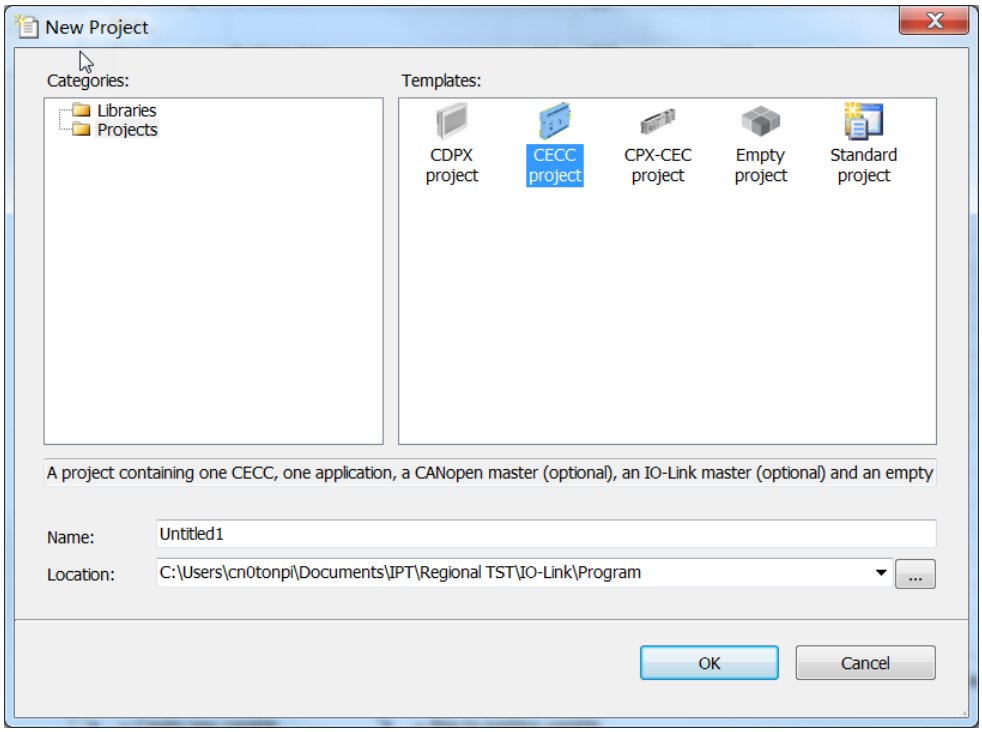
2 Configuring the IO-Link master

2.1 IO-Link configuration

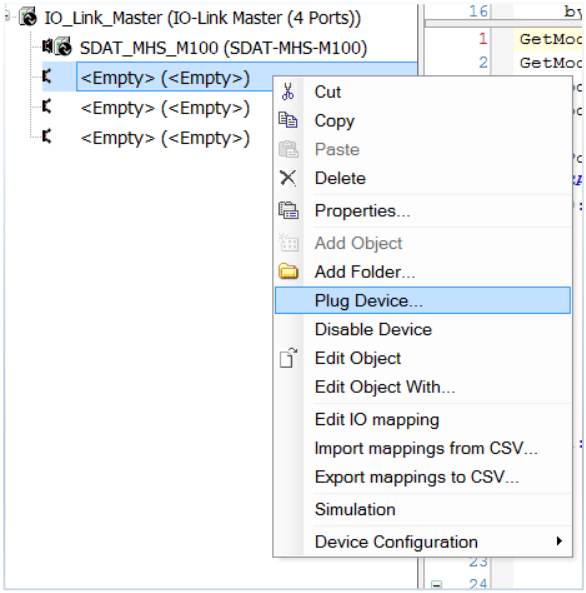
Download the IODD file of Balluff RFID read/write device and install it in the Codesys



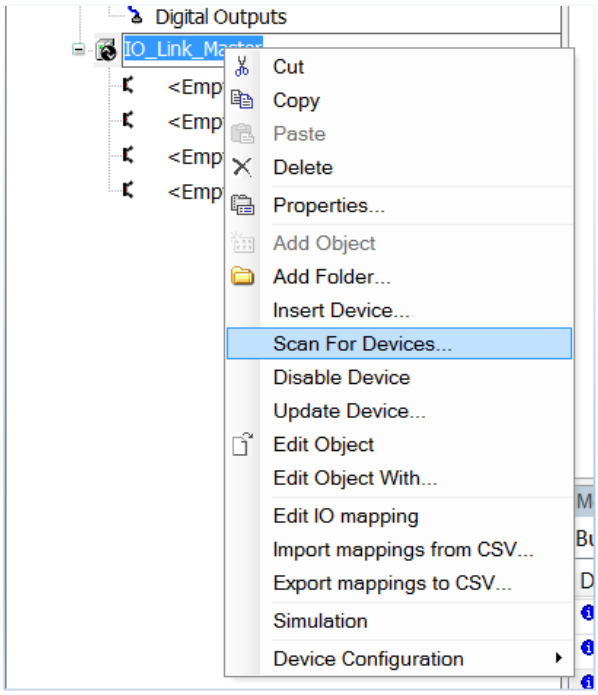
New an IO-Link master project, select CECC-LK and activate IO-Link Master



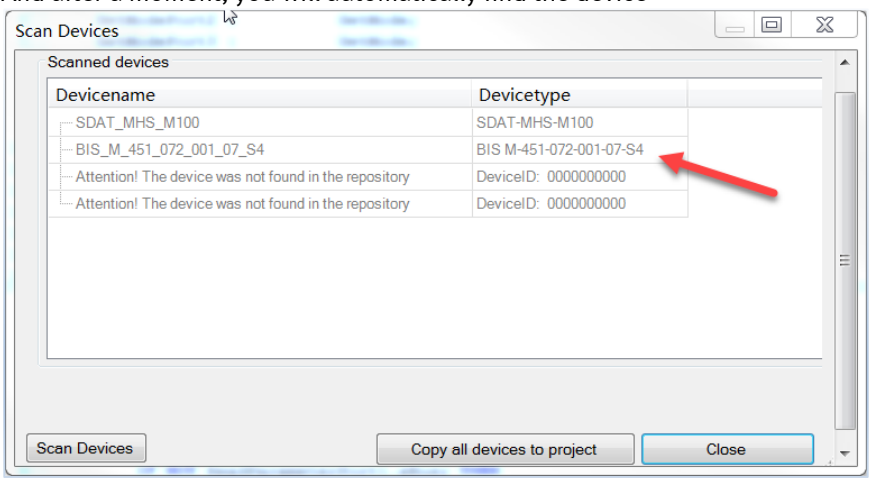
You can either right click the port -› Plug Device to add sensor manually, I use port 2 here.



Or right click the IO_Link_Master -> Scan For Devices



And after a moment, you will automatically find the device

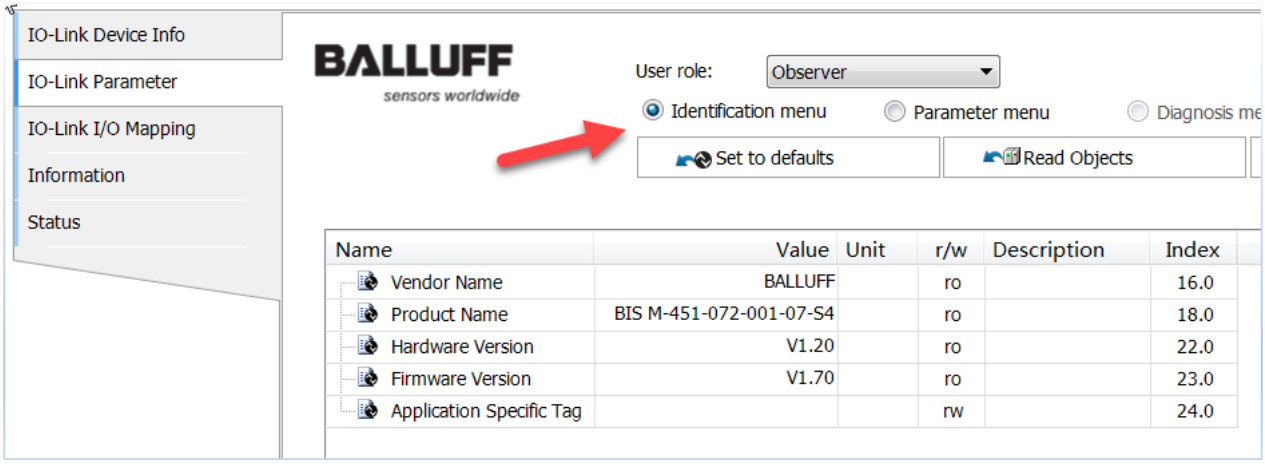


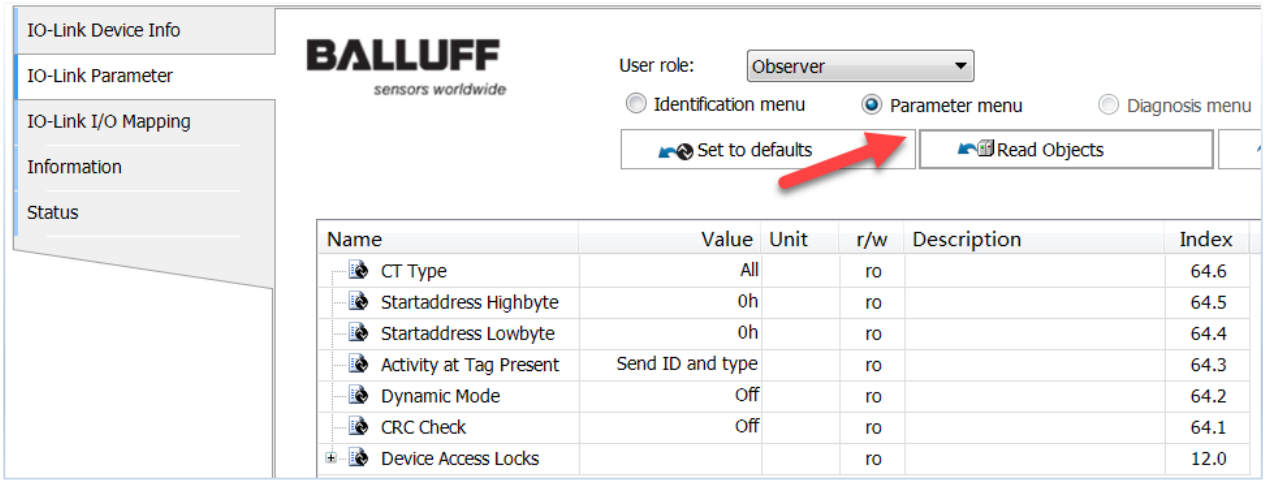
2.2 Configuration of Parameters and Testing

If BIS0103 is working properly, you will find the C/Q of CECC-LK is illuminated, and this read/write device's green LED flashes.

LED	Status	Function
LED 1	Green	Power
LED 2	Yellow	Data carrier detected
LED 1	Green flashing (1 s on / 100 ms off)	IO-Link connection active

Go to the BIS0103 IO-Link configurator, and activate the index column, you will see the parameters saved in Identification Menu and Parameter Menu.



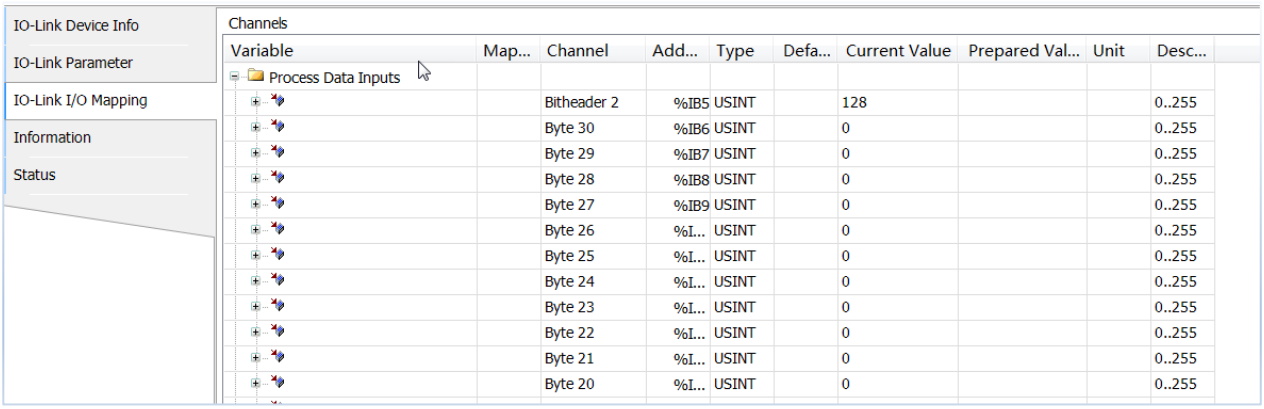


There is no description in the Balluff's IODD file, therefore, we get the description from manual. It is called SPDU in Balluff's manual.

	SPDU		Object name	Length	Information
	Index	Subindex			
Identification data	0_{hex}	8 9	Vendor ID	2 bytes	Balluff Vendor ID = 0378_{hex}
		10 11 12	Device ID	3 bytes	Balluff Device ID = $0602xx_{hex}$
	10_{hex}	0	Vendor name	7 bytes	Balluff
	11_{hex}	0	Vendor text	15 bytes	www.balluff.com
	12_{hex}	0	Product name	23 bytes	Device designation
	13_{hex}	0	Product ID	7 bytes	Ordering code
	14_{hex}	0	Product text	27 bytes	IO-Link RFID read-write head
	16_{hex}	0	Hardware revision	5 bytes	Hardware version
	17_{hex}	0	Firmware revision	5 bytes	Firmware version

	Access SPDU		Description	Data width	Value range	Factory setting
	Index	Subindex				
Parameter data	40_{hex}	1_{hex}	CRC yes/no	1 byte	0 = without CRC 1 = with CRC	0
	40_{hex}	2_{hex}	Dynamic mode - yes/ no	1 byte	0 = no 1 = yes	0
	40_{hex}	3_{hex}	Action if tag present	1 byte	0 = no action 1 = serial number and tag type 7 = automatically read 8 bytes of data beginning at a set start address after subindex 4 and 5	1
	40_{hex}	4_{hex}	Low byte of start address for autoread	2 bytes	Observe data-carrier specifications.	0
	40_{hex}	5_{hex}	High byte of start address for autoread			
	40_{hex}	6_{hex}	Used data-carrier type	1 byte	00_{hex}=ALL FE_{hex}=BIS M1_ _-01 FF_{hex}=BIS M1_ _-02	0

The BIS0103 has 32 bytes output & input, which is very huge, it reaches maximum process data size of the IO-Link defined. You can read & write process data directly in Codesys.



Description is again only available in the manual. To understand, for example, let us put a data carrier BIS0045 close to the range of it, and check the 1st bit string.

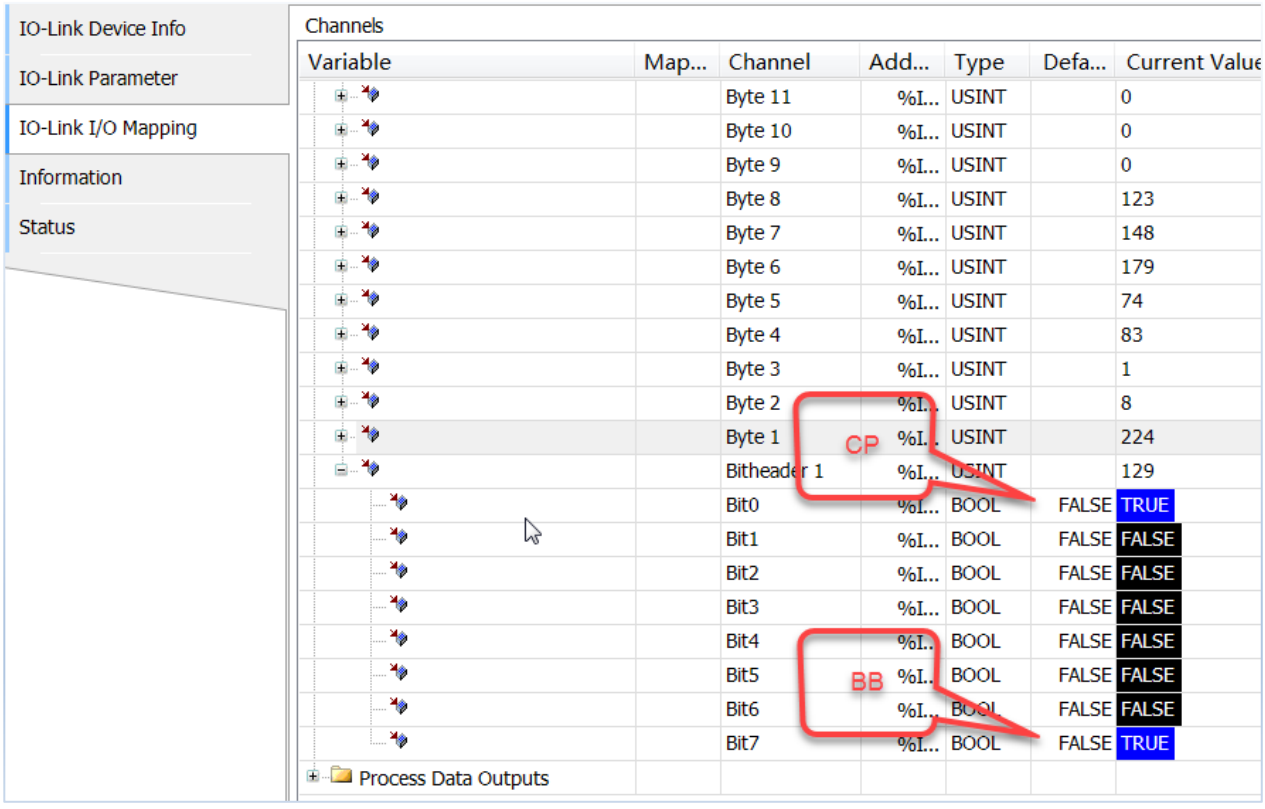
Input buffer:

Bit No. / Subaddress	7	6	5	4	3	2	1	0
00_{hex} - 1st bit string	BB	HF	TO		AF	AE	AA	CP
01_{hex}	Error code or data or high-byte version							
02_{hex}	Data or low-byte version							
03_{hex}	Data							
04_{hex}	Data							
05_{hex}	Data							
06_{hex}	Data							
07_{hex}	Data							
08_{hex}	Data							
Last byte - 2nd bit string	BB	HF	TO		AF	AE	AA	CP

Explanations on the input buffer using 10 bytes as an example:

Subaddress	Bit name	Meaning	Function description
00_{hex}	1st bit string		
	BB	Power	1 = Device is ready
			0 = Device is in ground state
	HF	Head Failure	1 = Head is turned off
			0 = Head is turned on
	TO	Toggle bit	A state change during a job indicates that the read/write device is ready to transfer other data
	AF	Job error	1 = Job incorrectly processed
			0 = Job processed without errors
	AE	Job end	1 = Job processed without errors
			0 = No job or job running
	AA	Job accepted	1 = The job was detected and accepted. Is being processed.
			0 = No job active
	CP	Codetag Present	Data carrier is in the read range of the read/write head
			No data carrier in read range

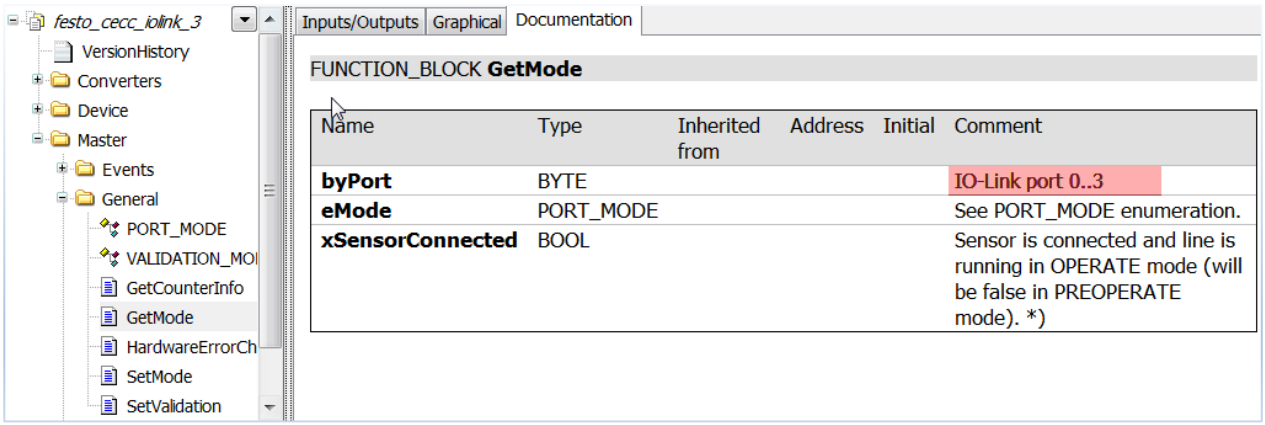
We can see the BB and CP is TRUE, the others are FALSE, which means “device is ready” and “data carrier is in the read range of the read/write head”.



2.3 Programming in Codesys

2.3.1 Program to read parameter by coding

First we test the IOLink connection by the Function Block GetMode.



Being noticed that we are using IOL port 2, but in this FB we should use port number “1”!

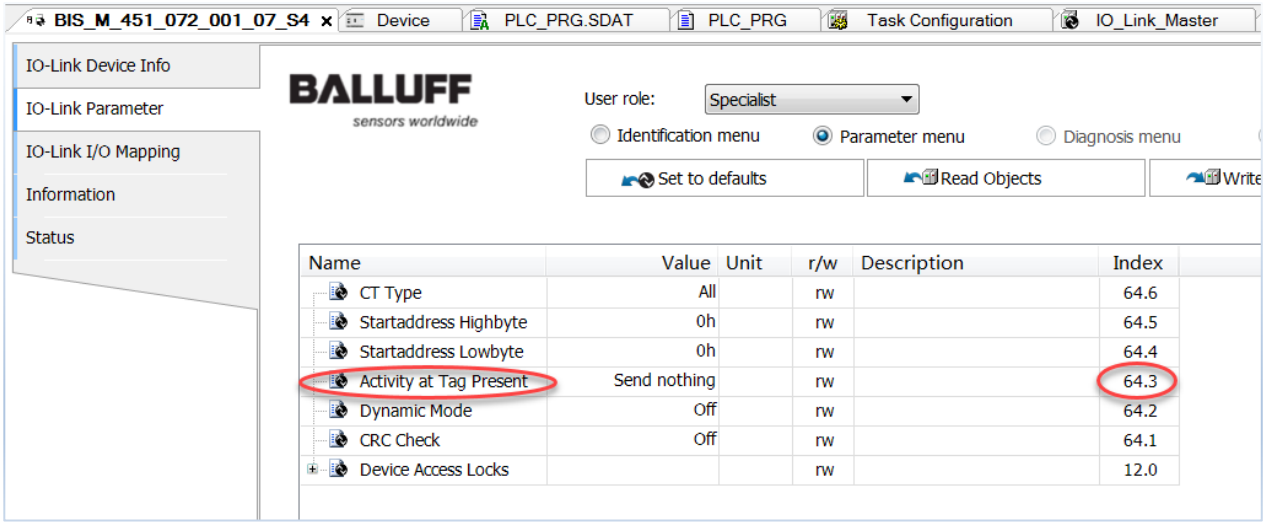
If the connection is OK, you get the return value “True”.

```
1  GetModePort1(byPort:=16#00, xSensorConnected=>xPort1Connected:=TRUE);
2  GetModePort2(byPort:=16#01, xSensorConnected=>xPort2Connected:=TRUE);
3  GetModePort3(byPort:=16#02, xSensorConnected=>xPort3Connected:=FALSE);
4  GetModePort4(byPort:=16#03, xSensorConnected=>xPort4Connected:=FALSE);
```

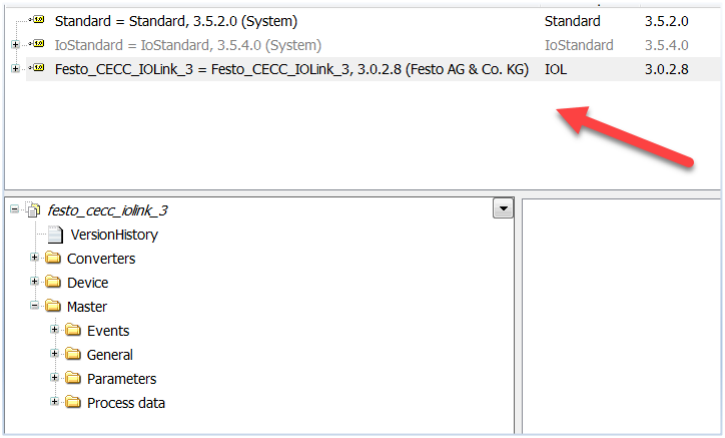
We want the data carrier to send data once the data is in the range of BIS0103.

This is controlled by the parameter Activity at Tag Present at Index 64.3.

In the screenshot, we can see the parameter has currently value of “Send nothing”. Therefore we need to program to firstly read the parameter in code, and then change it accordingly if necessary.



Import the library for IOL CECC-LK



Use FB readParameter to get the value

```
IF xPort2Connected THEN
    CASE byStepPort2 OF
    0:  //check the parameter Activity at Tag Present
        IF NOT readParameterPort2.xBusy THEN
            readParameterPort2(xExecute := TRUE, byPort :=1, wIndex := 64, bySubindex := 3, pbyData := ADR(byActivity));
        ELSE
            ReadParameterPort2(xExecute := FALSE);
        END_IF

        IF ReadParameterPort2.xDone THEN
            IF byActivity = 1 THEN
                byStepPort2 := 2;
            ELSE
                byStepPort2 := 1;
            END_IF
        END_IF
```

The right value of "send UID" is 1, we check this value

2.3.2 Program to write parameter by coding

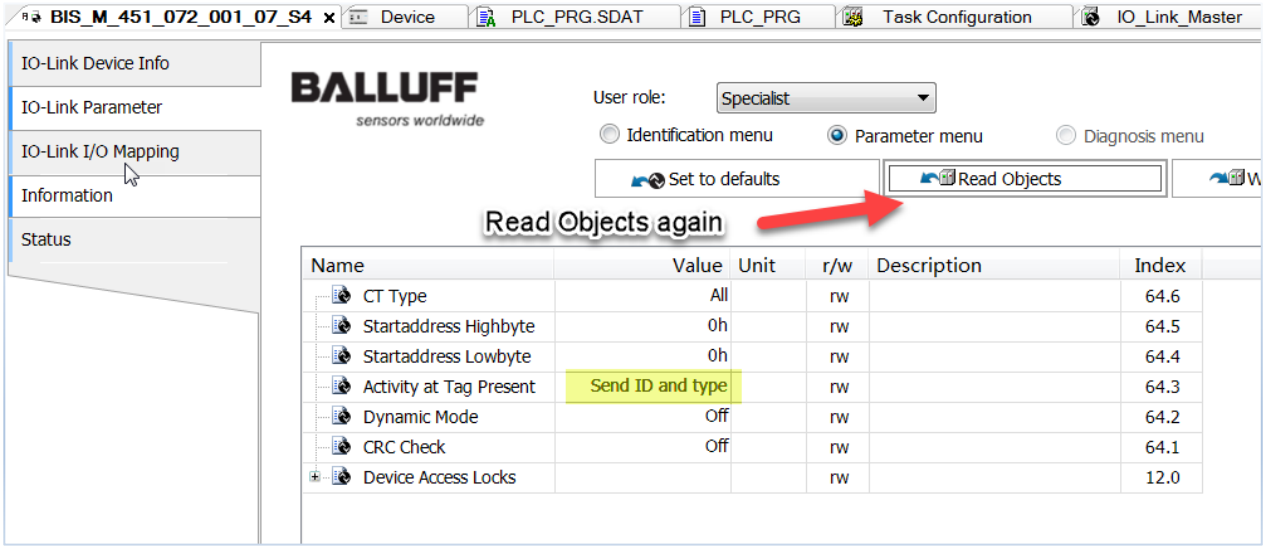
When the data carrier does not have the right parameter to send its UID, then we should use the FB WriteParameter to set the parameter with Index 64.3.

Coding and write operation ok

```
1: //set the mode if mode is not read UID
   byActivityW 16#01 := 1;  //set the mode to read UID
   IF NOT writeParameterPort2.xBusy FALSE THEN
       writeParameterPort2(xExecute FALSE := TRUE, byPort 16#01 := 1, wIndex 16#0040 := 64, bySubindex 16#03 := 3, pbyData 16#404BB364 := ADR(byActivityW 16#01 ), by
   ELSE
       WriteParameterPort2(xExecute FALSE := FALSE);
   END_IF
   IF WriteParameterPort2.xDone TRUE AND WriteParameterPort2.byErrorCode 16#00 = 0 THEN
       byStepPort2 16#02 := 2;
   ELSIF WriteParameterPort2.byErrorCode 16#00 <> 0 THEN
       byStepPort2 16#02 := 99;
   END_IF
2:
   ;
99: //Error handling
   ;
```

OK and no error

Check the parameter again, and we see the value of this parameter has already changed to required one.



2.3.3 Read stored data from data carrier

Let's try to read first 16 bytes.

Note:

As per experiment, you can still send data by sending the command 01 in address 01hex no matter in which "Activity at the Tag Present"!

Output buffer:

Bit No. / Subaddress	7	6	5	4	3	2	1	0
00_{hex} - 1st bit string		TI	KA			GR		AV
01_{hex}	Command designator or data							
02_{hex}	Start address (low byte) or data							
03_{hex}	Start address (high byte) or data							
04_{hex}	Number of bytes (low byte) or data							
05_{hex}	Number of bytes (high byte) or data							
06_{hex}	Data							
07_{hex}	Data							
08_{hex}	Data							
Last byte - 2nd bit string		TI	KA			GR		AV

Explanations on the output buffer using 10 bytes as an example:

Subaddress	Bit name	Meaning	Function description
00_{hex}	1st bit string		
	TI	Toggle bit	A state change during a job indicates that the controller is ready to receive additional data made available by the read/write device.
	KA	Head on/off	1 = Head off (read/write head switched off)
			0 = Head on (read/write head in operation)
	GR	Basic state	1 = Software reset - causes the BIS to switch to the ground state
			0 = Normal operation
	AV	Job	1 = New job pending
			0 = No new job or job no longer pending

01_{hex}		Command designator	00_{hex} = No command
			01_{hex} = Read data carrier
			02_{hex} = Write data carrier
			12_{hex} = Initialize the CRC_16 data check on the data carrier
			32_{hex} = Write a constant value on the data carrier

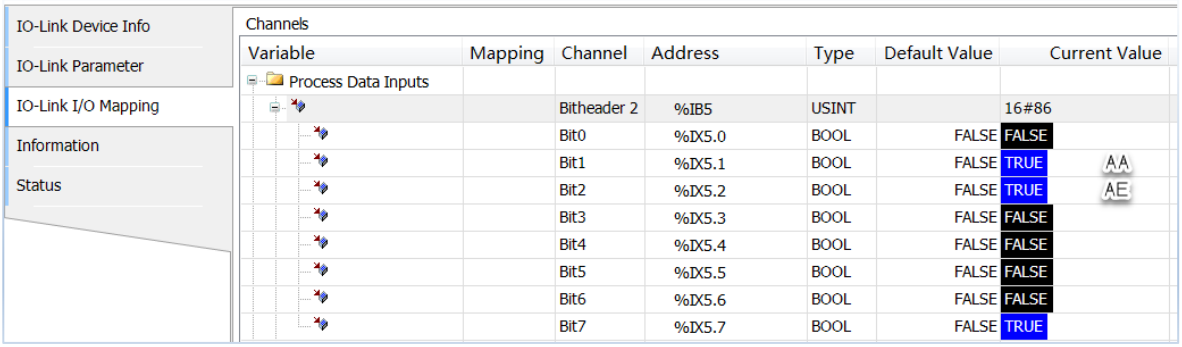
Be careful that both AV in first byte and last byte must be activated! By this device, parameters of in process data are redundant, so you have to set both, otherwise it will cause error.

```
2  //read first 8 byte from data carrier
   IF NOT xInitStep2 TRUE THEN
       %QB31 16#01 := 16#01;     //Output Address 01
       %QB30 16#00 := 16#00;     //Output Address 02
       %QB29 16#00 := 16#00;     //Output Address 03
       %QB28 16#10 := 16#10;     //Output Address 04
       %QX32.0 FALSE := TRUE;    //Set AV True: pend new job
       %QX1.0 FALSE := TRUE;     //Set AV true
```

Here the meaning of parameters is: read the 16 bytes of the data carrier starting from the address 0 (internal address of data carrier!).

If everything is correct you will get activated AA and AE, and the first 16 bytes from the data carrier in **Process Data address** addressed from 01hex to 16hex..

As we can see in the software



Variable	Mapping	Channel	Address	Type	Default Value	Current Value
Process Data Inputs						
		Bitheader 2	%IB5	USINT		16#86
		Bit0	%IX5.0	BOOL	FALSE	FALSE
		Bit1	%IX5.1	BOOL	FALSE	TRUE
		Bit2	%IX5.2	BOOL	FALSE	TRUE
		Bit3	%IX5.3	BOOL	FALSE	FALSE
		Bit4	%IX5.4	BOOL	FALSE	FALSE
		Bit5	%IX5.5	BOOL	FALSE	FALSE
		Bit6	%IX5.6	BOOL	FALSE	FALSE
		Bit7	%IX5.7	BOOL	FALSE	TRUE

Now, all the data from data carrier are:

Variable	Mapping	Channel	Address	Type	Default Value	Current Value
		Byte 19	%IB17	USINT		16#00
		Byte 18	%IB18	USINT		16#00
		Byte 17	%IB19	USINT		16#00
		Byte 16	%IB20	USINT		16#00
		Byte 15	%IB21	USINT		16#00
		Byte 14	%IB22	USINT		16#00
		Byte 13	%IB23	USINT		16#00
		Byte 12	%IB24	USINT		16#00
		Byte 11	%IB25	USINT		16#00
		Byte 10	%IB26	USINT		16#00
		Byte 9	%IB27	USINT		16#99
		Byte 8	%IB28	USINT		16#99
		Byte 7	%IB29	USINT		16#99
		Byte 6	%IB30	USINT		16#99
		Byte 5	%IB31	USINT		16#99
		Byte 4	%IB32	USINT		16#99
		Byte 3	%IB33	USINT		16#99
		Byte 2	%IB34	USINT		16#00
		Byte 1	%IB35	USINT		16#00

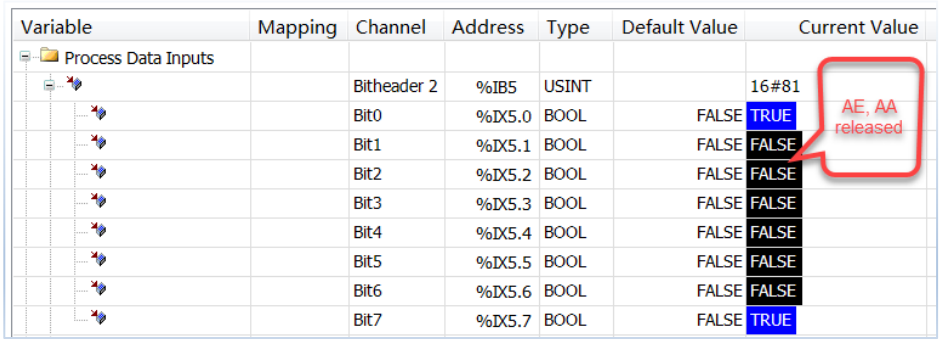
You have the possibility to read data in code.

```
IF %IX5.1 FALSE AND %IX5.2 FALSE THEN
    arrOldReadData[0] 16#00 := %IB35 16#00;
    arrOldReadData[1] 16#00 := %IB34 16#00;
    arrOldReadData[2] 16#99 := %IB33 16#99;
    arrOldReadData[3] 16#99 := %IB32 16#99;
    arrOldReadData[4] 16#99 := %IB31 16#99;
    arrOldReadData[5] 16#99 := %IB30 16#99;
    arrOldReadData[6] 16#99 := %IB29 16#99;
    arrOldReadData[7] 16#99 := %IB28 16#99;
    arrOldReadData[8] 16#99 := %IB27 16#99;
    arrOldReadData[9] 16#00 := %IB26 16#00;
    arrOldReadData[10] 16#00 := %IB25 16#00;
    arrOldReadData[11] 16#00 := %IB24 16#00;
    arrOldReadData[12] 16#00 := %IB23 16#00;
    arrOldReadData[13] 16#00 := %IB22 16#00;
    arrOldReadData[14] 16#00 := %IB21 16#00;
    arrOldReadData[15] 16#00 := %IB20 16#00;
    byStepPort2 16#04 := 3;
END_IF
```

After the read of data is finished, AV should be set to 0(release the new job pending)

```
68      3:  IF NOT xInitStep3 TRUE THEN
69              %QX32.0 FALSE := FALSE;   //Set AV false: release pending new job
70              %QX1.0 FALSE := FALSE;        //Set AV false
71              xInitStep3 TRUE := TRUE;
72          END_IF
```

If everything is correct, you get AA & AE from Process Data Input deactivated.



So the whole process of reading data is finished.
And the process of writing data is quite similar, so I do not write the process in this note.