Function block library

IOL_Basic_7

for PLCnext Engineer

Documentation for
PHOENIX CONTACT function blocks
PHOENIX CONTACT GmbH Co. KG
Flachsmarktstrasse 8
D-32825 Blomberg, Germany

This documentation is available in English only.

Table of Contents

1 Installation hint

Please copy the library data to your PLCnext Engineer (former: PC Worx Engineer) working library directory.

If you did not specify a different directory during **PLCnext Engineer** installation the default PLCnext Engineer working library directory is

C:\Users\Public\Documents\PLCnext Engineer\Libraries

2 General information

IO-Link is a communication system used for the connection of intelligent sensors and actuators to automation systems. An IO-Link system consists of one IO-Link master and one or more IO-Link devices, i.e. sensors or actuators. The IO-Link master provides the interface for the PLC and controls the communication with the connected IO-Link devices.

An IO-Link master can have one or more IO-Link ports, however, only one IO-Link device can be connected to each port.

An IO-Link device has parameter data (objects) that can be read and/or written via asychronous services (commands). The objects are device-specific. Therefore use the current AsynCom library available on the PHOENIX CONTACT website. Integrate both the IOL_Basic and the AsynCom library into your project.

2.1 Articles and function blocks

Article	Local bus	PROFINET
 AXL E PN IOL8 DI4 M12 6M (2701519)	Not possible. PROFINET only.	Use IOL_COM + AsynCom_PN
 AXL E PN IOL8 DI4 M12 6P (2701513)	Not possible. PROFINET only.	Use IOL_COM + AsynCom_PN

 AXL F IOL8 2H (1027843)	Do not use IOL_COM, use AsynCom_AXL only.	Use IOL_COM + AsynCom_PN
 AXL SE IOL4 (1088132)	Do not use IOL_COM, use AsynCom_AXL only.	Use IOL_COM + AsynCom_PN
 IB IL 24 IOL 4 DI 12-2MBD-PAC (2692733) IB IL 24 IOL 4 DI 12-PAC (2692717)	Use IOL_COM + AsynCom_IBS.	Use IOL_COM + AsynCom_PN
 IOL MA8 PN DI8 (1072838)	Not possible. PROFINET only.	Use IOL_COM + AsynCom_PN

3 Change notes

Library version	Library build	PLCnext Engineer version	Change notes	Supported PLCs
7	20220314	2021.0 LTS	Documentation improved	AXC F 1152 (1151412) AXC F 2152 (2404267) AXC F 3152 (1069208)
7	20211103	2021.0 LTS	• xReset: Input deleted. For resetting the function block please deactivate and activate again. • iMode: new input, for details refer to the documentation below. • udtConfig: new input, for details refer to the documentation below. • wDiagCode 16#C100 changed to 16#C101 to 16#C106 • New wDiagCode / dwAddDiagCode = 16#C600 / 16#00000000 = invalid process step	"
6	20200721	2020.0 LTS	Warning removed from IOL_6_EXA_IOL_MA8_PN_DI8.pcwex	"
6	20200408	2020.0 LTS	Corrupted data during upload	"
6	20200330	2020.0 LTS	Hint for AXL SE IOL4 (1088132) added, please refer to “Function blocks”	"
6	20200316	2020.0 LTS	Hint for AXL F IOL8 2H (1027843) added, please refer to “Function blocks”	"
6	20200206	2020.0 LTS	Released for 2020.0 LTS	AXC F 1152 (1151412) AXC F 2152 (2404267)
6	20191001	2019.0 LTS 2019.3 2019.6 2019.9	Adapted to 2019.9	AXC F 2152 (2404267)
5	20190722	2019.0 LTS 2019.3 2019.6	Adapted to 2019.6	"
4	20190607	2019.0 LTS	Added current build of the AsynCom library (to msi-file)	"
4	20190318	2019.0 LTS	• Adapted to IOL MA8 PN DI8 - 1072838 • Example for IOL MA8 PN DI8 - 1072838 added	"
3	20190225	2019.0 LTS	Supports “Allow extended identifiers” = ON	"
3	20190219	2019.0 LTS	• Adapted to PLCnext Engineer 2019.0 LTS	"

2	20180928	7.2.3	• Approval for IB IL 24 IOL 4 DI 12-2MBD-PAC (2692733) • Timeout changed from 2s to 5s. • Function block also works with cycle times up to 500 ms.	"
1	20180712	7.2.2	• Converted from PC Worx 6	"

New version number: Functional changes of at least one function block, incompatibilities (e.g. change of library format)

New build number: No functional changes, but changes in the ZIP file (e.g. documentation update, additional examples)

4 Function blocks

Function block	Description	Version	Supported articles	License
IOL_COM	The block enables the asynchronous communication with the Phoenix Contact IO-Link modules. The function block can be used to write and / or read IO-Link services on the IO-Link-Master or on the IO-Link devices	4	AXL E PN IOL8 DI4 M12 6M (2701519) **FW >= V2.1.0** AXL E PN IOL8 DI4 M12 6P (2701513) **FW >= V2.1.0** AXL F IOL8 2H (1027843) AXL SE IOL4 (1088132) IB IL 24 IOL 4 DI 12-2MBD-PAC (2692733) IB IL 24 IOL 4 DI 12-PAC (2692717) IOL MA8 PN DI8 (1072838)	none

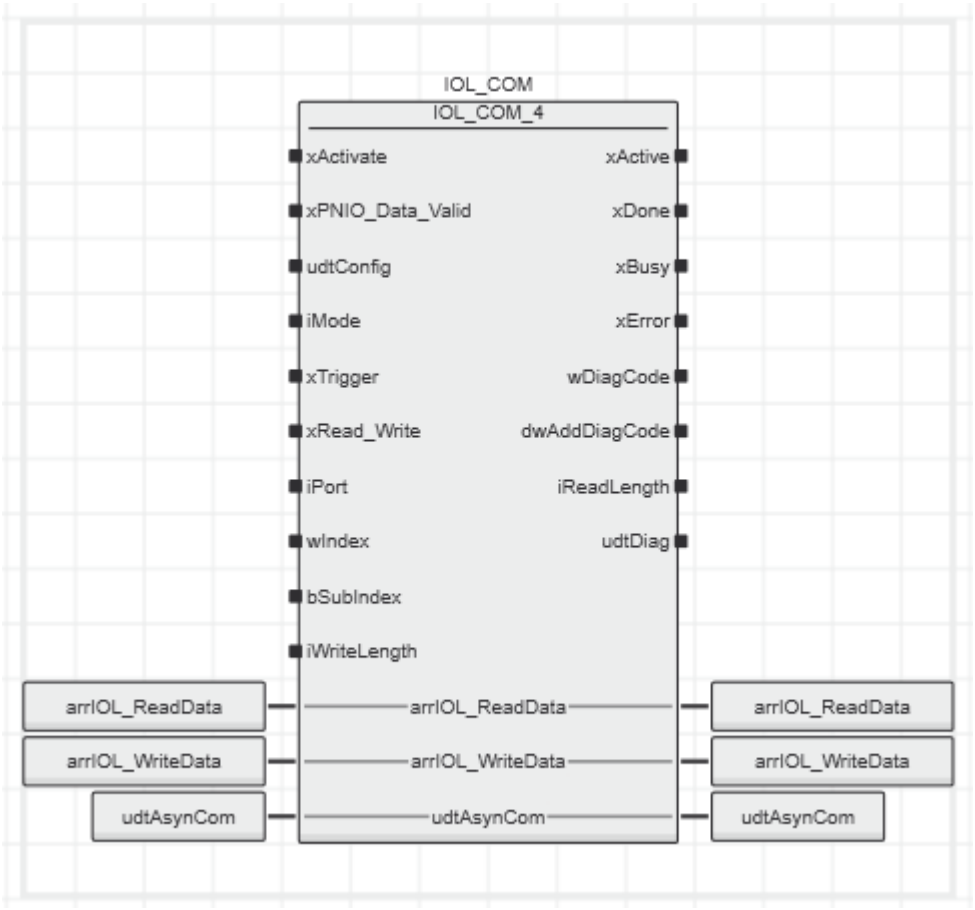
5 IOL_COM

The function block enables the asynchronous communication with the Phoenix Contact IO-Link modules. The function block can be used to write and / or read IO-Link services on the IO-Link master or on the IO-Link devices.

The function block can be used in different bus systems. The AsynCom library contains function blocks for every bus system. The IOL_COM function block communicates with the function blocks from the AsynCom library via the udtAsynCom structure.

If the address (node ID or communication reference) at the AsynCom block has been changed, restart the function block.

5.1 Function block call



5.2 Input parameters

<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>xActivate</td><td>BOOL</td><td>Rising edge: Activates the function block.
FALSE: Deactivates the function block.</td></tr>
<tr><td>xPNIO_Data_Valid</td><td>BOOL</td><td>Status bit for the evaluation of a valid PROFINET connection (optional).</td></tr>
<tr><td>udtConfig</td><td>IOL_UDT_CONF</td><td>Configuration data

These data are only needed if some default settings are not suitable for the current application.

udtConfig.wIndex: Index for PROFINET communication. For details refer to the IO-Link specification.

wIndex has to be set for the following IOL masters.

<table><tr><th>IOL master</th><th>wIndex</th></tr><tr><td>AXL E PN IOL8 DI4 M12 6M (2701519)</td><td>WORD#16#00FF</td></tr><tr><td>AXL E PN IOL8 DI4 M12 6P (2701513)</td><td>WORD#16#00FF</td></tr></table>
udtConfig.tRetry: Time between two retries.

udtConfig.tTimeout: Global timeout time. Default = ms. Needs to be increased for cyclic tasks with high cycle time.</td></tr>
<tr><td>iMode</td><td>INT</td><td>0 (default): IOL master does not require read operation after write operation. 1: IOL master requires read operation after write operation according to the IO-Link specification.

<table><tr><th>IOL master</th><th>iMode</th></tr><tr><td>AXL E PN IOL8 DI4 M12 6M (2701519)</td><td>0</td></tr><tr><td>AXL E PN IOL8 DI4 M12 6P (2701513)</td><td>0</td></tr><tr><td>AXL F IOL8 2H (1027843)</td><td>1</td></tr><tr><td>AXL SE IOL4 (1088132)</td><td>1</td></tr><tr><td>IOL MA8 PN DI8 (1072838)</td><td>0</td></tr></table></td></tr>
<tr><td>xTrigger</td><td>BOOL</td><td>An IO-Link service is read and / or written with a positive edge.</td></tr>
<tr><td>xRead_Write</td><td>BOOL</td><td>TRUE: Write access
FALSE: Read access</td></tr>
<tr><td>iPort</td><td>INT</td><td>Port number on which to read or write
0: IO-Link master
1..8: Port number</td></tr>
<tr><td>wIndex</td><td>WORD</td><td>Index of IOL object to be accessed to</td></tr>
<tr><td>bSubIndex</td><td>BYTE</td><td>Subindex of IOL object</td></tr>
<tr><td>iWriteLength</td><td>INT</td><td>Number of bytes to be written</td></tr>
</table>

5.3 Output parameters

Name	Type	Description
xActive	BOOL	FALSE: Function block is not active. TRUE: Function block is active. Do not start any further action unless xActive is TRUE after activation!
xDone	BOOL	TRUE: Service written/read successfully. The parameter is True as long as xTrigger is True (for at least one cycle).
xBusy	BOOL	FALSE: The function block is ready for the next access. TRUE: The block is busy with the service execution.
xError	BOOL	TRUE: An error has occurred. For more details refer to wDiagCode and wAddDiagCode.
wDiagCode	WORD	Diagnosis code. Refer to diagnostic table.
dwAddDiagCode	DWORD	Additional diagnostic code. Refer to diagnostics table.
iReadLength	INT	Number of data bytes received

5.4 Inout parameters

Name	Type	Description
arrIOL_ReadData	IOL_ARR_B_1_64	The array contains the received data. Every successful read access will overwrite the data in this array by the newly received data.
arrIOL_WriteData	IOL_ARR_B_1_64	The array contains the data to be written.
udtAsynCom	ASYN_UDT_COM	Data exchange structure for asynchronous communication. Connected to AsynCom* function block.

5.5 Diagnosis

<table>
<tr><th>wDiagCode</th><th>wAddDiagCode</th><th>Description</th></tr>
<tr><td>16#0000</td><td>16#00000000</td><td>Function block is deactivated</td></tr>
<tr><td>16#8000</td><td>16#00000000</td><td>Function block is in regular operation</td></tr>
<tr><td>16#C101</td><td>16#00000000</td><td>Timeout or PROFINET connection error. The parameter xPNIO_Data_Valid (optional) is FALSE.</td></tr>
<tr><td>16#C102</td><td>udtAsynCom.dwDiagCodeConnect</td><td>AsynCom connection error during init</td></tr>
<tr><td>16#C103</td><td>udtAsynCom.dwDiagCodeConnect</td><td>AsynCom connection error in state 0</td></tr>
<tr><td>16#C104</td><td>16#00000000</td><td>AsynCom error while writing.</td></tr>
<tr><td>16#C105</td><td>16#00000000</td><td>AsynCom error while reading.</td></tr>
<tr><td>16#C106</td><td>16#00000000</td><td>General timeout. If the reason is that your system is too slow increase udtConfig.tTimeout (e.g. for 500 ms cycle time).</td></tr>
<tr><td>16#C200</td><td>16#00000001</td><td>Configuration error. Invalid parameter at the iWriteLength input.</td></tr>
<tr><td>16#C300</td><td></td><td>IO-Link error</td></tr>
<tr><td></td><td>16#xxxxxxxx</td><td>The parameter is structured as follows:
<table>
<tr><td>WORD 1</td><td colspan="2">WORD 0</td></tr>
<tr><td></td><td>Byte 1</td><td>Byte 0</td></tr>
<tr><td>IOL_M
Error_Code</td><td>IOL
Error_Code</td><td>IOL
Add_Error_Code</td></tr>
</table>
For details refer to appendix.</td></tr>
<tr><td>16#C400</td><td></td><td>Error while sending</td></tr>
<tr><td></td><td>16#xxxxxxxx</td><td>Error in the WRREC or PCP_WRITE firmware block.</td></tr>
<tr><td></td><td>16#DF80A1xx</td><td>Negative acknowledgment when writing to the module: Caused by: Incorrect service content, index, subindex, port number or length of the data to be written.</td></tr>
<tr><td></td><td>16#0000F003</td><td>Invalid node ID</td></tr>
<tr><td></td><td>16#0000F016</td><td>The service used to write the data record could not be run.</td></tr>
<tr><td></td><td>16#06050000</td><td>INTERBUS: Invalid length of the data to be written</td></tr>
<tr><td></td><td>16#06070000</td><td>INTERBUS: Invalid index</td></tr>
<tr><td>16#C500</td><td></td><td>Error while receiving</td></tr>
<tr><td></td><td>16#xxxxxxxx</td><td>Error in the RDREC or PCP_READ firmware block</td></tr>
<tr><td></td><td>16#DE80A0xx</td><td>Negative acknowledgment when reading from the module: Caused by: Incorrect service content, index, subindex or port number</td></tr>
<tr><td></td><td>16#0000F003</td><td>Invalid node ID</td></tr>
<tr><td></td><td>16#0000F015</td><td>The service used to read the data record could not be run.</td></tr>
<tr><td></td><td>16#06070000</td><td>INTERBUS: Invalid index</td></tr>
<tr><td>16#C600</td><td>16#00000000</td><td>Invalid process step. Please contact the technical support.</td></tr>
</table>

6 Startup examples

For the startup instruction please find the following examples:

- IOL_*_EXA_AXL_E.pcwex
- IOL_*_EXA_AXL_F_IOL8_2H_LOC.pcwex
- IOL_*_EXA_AXL_F_IOL8_2H_PN_TPS.pcwex
- IOL_*_EXA_AXL_SE_IOL4_LOC.pcwex
- IOL_*_EXA_AXL_SE_IOL4_PN_TPS.pcwex
- IOL_*_EXA_MA8_PN_DI8.pcwex

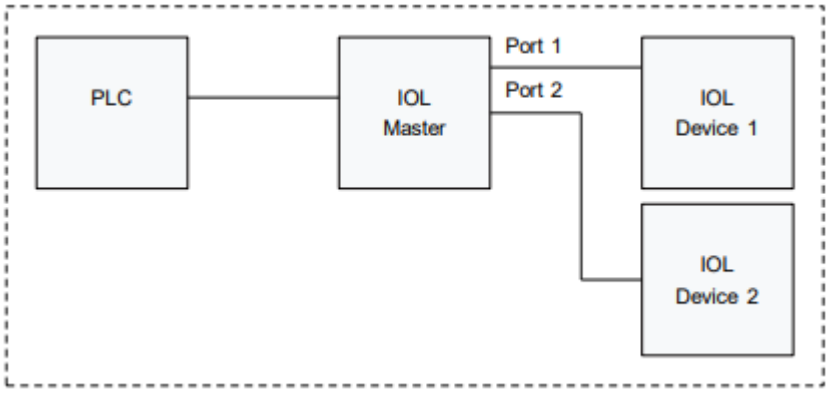
These examples are packed in the zipped Examples folder of the library.

6.1 Example 1: IOL_X_EXA_AXL_E

6.1.1 Plant

- AXC F 2152 (2404267)
  - AXL E PN IOL8 DI4 M12 6M (2701519)

6.1.2 External components



IOL device 1 = AXL E IOL DO8 M12 6P (2702659)

IOL device 2 = AXL E IOL DI8 M12 6P (2702658)

6.1.3 Example description

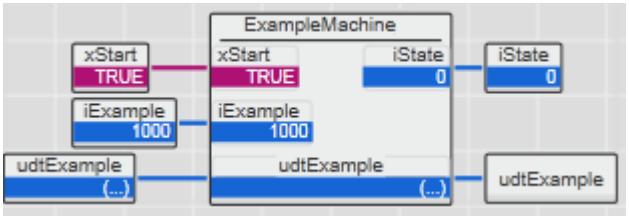
In the example project, we find the function block ExampleMachine. This function block contains a state machine for each example with a detailed description about what we have to do to use the function block correctly.

The following examples can be executed:

Function	iExample	Codesheet
Read / write to IOL device 1	1000	E1000
Read / write to IOL device 2	2000	E2000

6.1.3.1 Example machine

For starting ExampleMachine function block, the requested example can be selected at iExample input and xStart input has set to TRUE.



6.1.3.2 State machine

6.1.3.2.1 E1000

```
(*
** 1st IOL device
*)
IF
    xStart              = TRUE AND
    udtExample.iExample = 1000
THEN
    CASE udtExample.iState OF
        0: (* Init *)
            (* Reset code *)
            wIndex          := WORD#16#0101;
            iWriteLength    := 2;

            (* IOL master *)
            udtExample.dwNodeID                     := WORD#35;
            udtExample.udtIolCom.udtConfig.wIndex   := WORD#16#00FF;
            (* 1st IOL device *)
            udtExample.udtIolCom.iPort              := 5;
            udtExample.udtIolCom.iMode              := 0;
            udtExample.udtIolCom.xRead_Write        := FALSE;
            udtExample.udtIolCom.xTrigger           := FALSE;
            udtExample.udtIolCom.xActivate          := TRUE;
            IF udtExample.udtIolCom.xError  = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtIolCom.xActive  = TRUE THEN
                udtExample.iState   := 100;
            END_IF;

        100: (* Read *)
            udtExample.udtIolCom.wIndex     := wIndex;
            udtExample.udtIolCom.xTrigger   := TRUE;
            IF udtExample.udtIolCom.xError  = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtIolCom.xDone    = TRUE THEN
                udtExample.udtIolCom.xTrigger   := FALSE;
                (* You can find the reset code in udtExample.arrIOL_ReadData *)
                udtExample.iState               := 200;
        END_IF;

        200: (* Write *)
            udtExample.arrIOL_WriteData[0]      := BYTE#16#00;
            udtExample.arrIOL_WriteData[1]      := BYTE#16#02;
            udtExample.udtIolCom.wIndex         := wIndex;
            udtExample.udtIolCom.xRead_Write    := TRUE;
```

```
            udtExample.udtIolCom.iWriteLength   := iWriteLength;
            udtExample.udtIolCom.xTrigger       := TRUE;
            IF udtExample.udtIolCom.xError  = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtIolCom.xDone    = TRUE THEN
                udtExample.udtIolCom.xTrigger   := FALSE;
                udtExample.iState               := 300;
            END_IF;

        300: (* Read back *)
            udtExample.udtIolCom.wIndex         := wIndex;
            udtExample.udtIolCom.xRead_Write    := FALSE;
            udtExample.udtIolCom.xTrigger       := TRUE;
            IF udtExample.udtIolCom.xError  = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF
                udtExample.udtIolCom.xDone          = TRUE
                (* Compare read data with write data *)
                AND udtExample.arrIOL_ReadData[0]   = BYTE#16#00
                AND udtExample.arrIOL_ReadData[1]   = BYTE#16#02
            THEN
                udtExample.udtIolCom.xTrigger   := FALSE;
                udtExample.iState               := 32000;
        END_IF;

        9000: (* Error state *)
            (* Implement your error handling here *)
            udtExample.iState   := udtExample.iState;

        32000: (* Finished *)
            (* This part of the example is successfully finished *)
            udtExample.iState   := 32000;
            udtExample.iExample := 32000;
    END_CASE;
END_IF;
```

6.1.3.2.2 E2000

```
(*
** 2nd IOL device
*)
IF
    xStart              = TRUE AND
    udtExample.iExample = 2000
THEN
    CASE udtExample.iState OF
        0: (* Init *)
            (* Reset code *)
            wIndex          := WORD#16#0101;
            iWriteLength    := 2;

            (* IOL master *)
            udtExample.dwNodeID                     := WORD#35;
            udtExample.udtIolCom.udtConfig.wIndex   := WORD#16#00FF;
            (* 2nd IOL device *)
            udtExample.udtIolCom.iPort              := 1;
            udtExample.udtIolCom.iMode              := 0;
            udtExample.udtIolCom.xRead_Write        := FALSE;
            udtExample.udtIolCom.xTrigger           := FALSE;
            udtExample.udtIolCom.xActivate          := TRUE;
            IF udtExample.udtIolCom.xError  = TRUE THEN
                (* Goto error handling *)
```

```
            udtExample.iState   := 9000;
        ELSIF udtExample.udtIolCom.xActive  = TRUE THEN
            udtExample.iState   := 100;
        END_IF;

    100: (* Read *)
        udtExample.udtIolCom.wIndex     := wIndex;
        udtExample.udtIolCom.xTrigger   := TRUE;
        IF udtExample.udtIolCom.xError  = TRUE THEN
            udtExample.udtIolCom.xTrigger   := FALSE;
            (* Because this 2nd IOL device does not have the parameter "reset code"
            this read access causes an error *)
            udtExample.iState   := 9000;
        ELSIF udtExample.udtIolCom.xDone    = TRUE THEN
            udtExample.udtIolCom.xTrigger   := FALSE;
            (* You can find the reset code in udtExample.arrIOL_ReadData *)
            udtExample.iState               := 200;
        END_IF;

    9000: (* Error state *)
        (* Implement your error handling here.
        In this example we reset the IOL_COM with deactivation and continue *)
        udtExample.iState   := 9010;

    9010: (* Reset *)
        udtExample.udtIolCom.xActivate  := FALSE;
        IF udtExample.udtIolCom.xActive = FALSE THEN
            udtExample.iState   := 32000;
        END_IF;

    32000: (* Finished *)
        (* This part of the example is successfully finished *)
        udtExample.iState   := 32000;
        udtExample.iExample := 32000;
END_CASE;
END_IF;
```

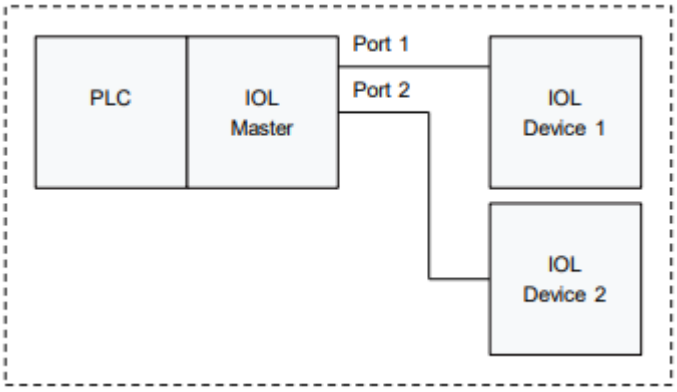
6.2 Example 2: IOL_X_EXA_AXL_F_IOL8_2H_LOC

Important: Only the AsynCom library is needed, the IOL_Com library is not needed.

6.2.1 Plant

- AXC F 2152 (2404267)
  - AXL F IOL8 2H (1027843)

6.2.2 External components



IOL device 1 = AXL E IOL DO8 M12 6P (2702659)

IOL device 2 = AXL E IOL DI8 M12 6P (2702658)

6.2.3 Example description

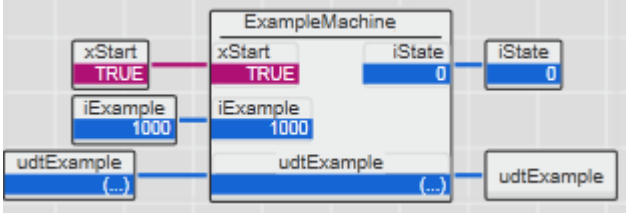
In the example project, we find the function block ExampleMachine. This function block contains a state machine for each example with a detailed description about what we have to do to use the function block correctly.

The following examples can be executed:

Function	iExample	Codesheet
Read / write to IOL master	1000	E1000
Read / write to IOL device 1	2000	E2000
Read / write to IOL device 2	3000	E3000

6.2.3.1 Example machine

For starting ExampleMachine function block, the requested example can be selected at iExample input and xStart input has set to TRUE.



6.2.3.2 State machine

6.2.3.2.1 E1000

```
(*
** IOL master
*)
IF
    xStart              = TRUE AND
    udtExample.iExample = 1000
THEN
    CASE udtExample.iState OF
        0: (* Init *)
            (* IOL master = AXL F IOL8 2H - 1027843 *)
            udtExample.wSlot                := WORD#16#0001;
            (* IOL master = AXL F IOL8 2H - 1027843 *)
            udtExample.bSubSlot             := BYTE#16#00;
            (* Reset code *)
            wIndex                          := WORD#16#0024;
            iLen                            := 16;
            (* Deactivate first to have a reliable start condition *)
            udtExample.udtAsynCom.xActivate := FALSE;
            IF udtExample.udtAsynCom.xActive    = FALSE THEN
                udtExample.iState   := 10;
            END_IF;

        10: (* Activate *)
            udtExample.udtAsynCom.xActivate := TRUE;
            IF udtExample.udtAsynCom.xError = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtAsynCom.xActive = TRUE THEN
                udtExample.iState   := 100;
            END_IF;

        100: (* Read *)
            udtExample.udtAsynCom.udtRead.wIndex    := wIndex;
            udtExample.udtAsynCom.udtRead.iMlen     := iLen;
            udtExample.udtAsynCom.udtRead.xReq      := TRUE;
            IF udtExample.udtAsynCom.xError = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtAsynCom.udtRead.xDone   = TRUE THEN
                udtExample.udtAsynCom.udtRead.xReq  := FALSE;
                (* You can find the reset code in udtExample.udtAsynCom.udtRead.arrData *)
                udtExample.iState   := 200;
            END_IF;

        200: (* Write *)
            udtExample.udtAsynCom.udtWrite.arrData[1]   := BYTE#16#00;
            udtExample.udtAsynCom.udtWrite.arrData[2]   := BYTE#16#02;
            udtExample.udtAsynCom.udtWrite.arrData[3]   := BYTE#16#00;
            udtExample.udtAsynCom.udtWrite.arrData[4]   := BYTE#16#02;
            udtExample.udtAsynCom.udtWrite.arrData[5]   := BYTE#16#00;
            udtExample.udtAsynCom.udtWrite.arrData[6]   := BYTE#16#02;
            udtExample.udtAsynCom.udtWrite.arrData[7]   := BYTE#16#00;
            udtExample.udtAsynCom.udtWrite.arrData[8]   := BYTE#16#02;
            udtExample.udtAsynCom.udtWrite.wIndex       := wIndex;
            udtExample.udtAsynCom.udtWrite.iLen         := iLen;
            udtExample.udtAsynCom.udtWrite.xReq         := TRUE;
            IF udtExample.udtAsynCom.xError = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtAsynCom.udtWrite.xDone  = TRUE THEN
```

```
                udtExample.udtAsynCom.udtWrite.xReq := FALSE;
                udtExample.iState                   := 300;
            END_IF;

        300: (* Read back *)
            udtExample.udtAsynCom.udtRead.wIndex    := wIndex;
            udtExample.udtAsynCom.udtRead.iMlen     := iLen;
            udtExample.udtAsynCom.udtRead.xReq      := TRUE;
            IF udtExample.udtAsynCom.xError = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF
                udtExample.udtAsynCom.udtRead.xDone         = TRUE AND
                (* Compare read data with write data *)
                udtExample.udtAsynCom.udtRead.arrData[1]    = BYTE#16#00 AND
                udtExample.udtAsynCom.udtRead.arrData[2]    = BYTE#16#02 AND
                udtExample.udtAsynCom.udtRead.arrData[3]    = BYTE#16#00 AND
                udtExample.udtAsynCom.udtRead.arrData[4]    = BYTE#16#02 AND
                udtExample.udtAsynCom.udtRead.arrData[5]    = BYTE#16#00 AND
                udtExample.udtAsynCom.udtRead.arrData[6]    = BYTE#16#02 AND
                udtExample.udtAsynCom.udtRead.arrData[7]    = BYTE#16#00 AND
                udtExample.udtAsynCom.udtRead.arrData[8]    = BYTE#16#02
            THEN
                udtExample.udtAsynCom.udtRead.xReq  := FALSE;
                udtExample.iState                   := 32000;
            END_IF;

        9000: (* Error state *)
            (* Implement your error handling here *)
            udtExample.iState   := udtExample.iState;

        32000: (* Finished *)
            (* This part of the example is successfully finished *)
            udtExample.iState   := 32000;
            udtExample.iExample := 32000;
    END_CASE;
END_IF;
```

6.2.3.2.2 E2000

```
(*
** 1st IOL device
*)
IF
    xStart              = TRUE AND
    udtExample.iExample = 2000
THEN
    CASE udtExample.iState OF
        0: (* Init *)
            (* IOL master = AXL F IOL8 2H - 1027843 *)
            udtExample.wSlot                := WORD#16#0001;
            (* 1st IOL device = AXL E IOL DO8 M12 6P - 2702659 *)
            udtExample.bSubSlot             := BYTE#16#01;
            (* Reset code *)
            wIndex                          := WORD#16#0101;
            iLen                            := 2;
            (* Deactivate first to have a reliable start condition *)
            udtExample.udtAsynCom.xActivate := FALSE;
            IF udtExample.udtAsynCom.xActive    = FALSE THEN
                udtExample.iState   := 10;
            END_IF;

        10: (* Activate *)
            udtExample.udtAsynCom.xActivate := TRUE;
```

```
            IF udtExample.udtAsynCom.xError = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtAsynCom.xActive = TRUE THEN
                udtExample.iState   := 100;
            END_IF;

        100: (* Read *)
            udtExample.udtAsynCom.udtRead.wIndex    := wIndex;
            udtExample.udtAsynCom.udtRead.iMlen     := iLen;
            udtExample.udtAsynCom.udtRead.xReq      := TRUE;
            IF udtExample.udtAsynCom.xError = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtAsynCom.udtRead.xDone   = TRUE THEN
                udtExample.udtAsynCom.udtRead.xReq  := FALSE;
                (* You can find the reset code in udtExample.udtAsynCom.udtRead.arrData *)
                udtExample.iState                   := 200;
            END_IF;

        200: (* Write *)
            udtExample.udtAsynCom.udtWrite.arrData[1]   := BYTE#16#00;
            udtExample.udtAsynCom.udtWrite.arrData[2]   := BYTE#16#02;
            udtExample.udtAsynCom.udtWrite.wIndex       := wIndex;
            udtExample.udtAsynCom.udtWrite.iLen         := iLen;
            udtExample.udtAsynCom.udtWrite.xReq         := TRUE;
            IF udtExample.udtAsynCom.xError = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtAsynCom.udtWrite.xDone  = TRUE THEN
                udtExample.udtAsynCom.udtWrite.xReq := FALSE;
                udtExample.iState                   := 300;
            END_IF;

        300: (* Read back *)
            udtExample.udtAsynCom.udtRead.wIndex    := wIndex;
            udtExample.udtAsynCom.udtRead.iMlen     := iLen;
            udtExample.udtAsynCom.udtRead.xReq      := TRUE;
            IF udtExample.udtAsynCom.xError = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF
                udtExample.udtAsynCom.udtRead.xDone         = TRUE AND
                (* Compare read data with write data *)
                udtExample.udtAsynCom.udtRead.arrData[1]    = BYTE#16#00 AND
                udtExample.udtAsynCom.udtRead.arrData[2]    = BYTE#16#02
            THEN
                udtExample.udtAsynCom.udtRead.xReq  := FALSE;
                udtExample.iState                   := 32000;
            END_IF;

        9000: (* Error state *)
            (* Implement your error handling here *)
            udtExample.iState   := udtExample.iState;

        32000: (* Finished *)
            (* This part of the example is successfully finished *)
            udtExample.iState   := 32000;
            udtExample.iExample := 32000;
    END_CASE;
END_IF;
```

6.2.3.2.3 E3000

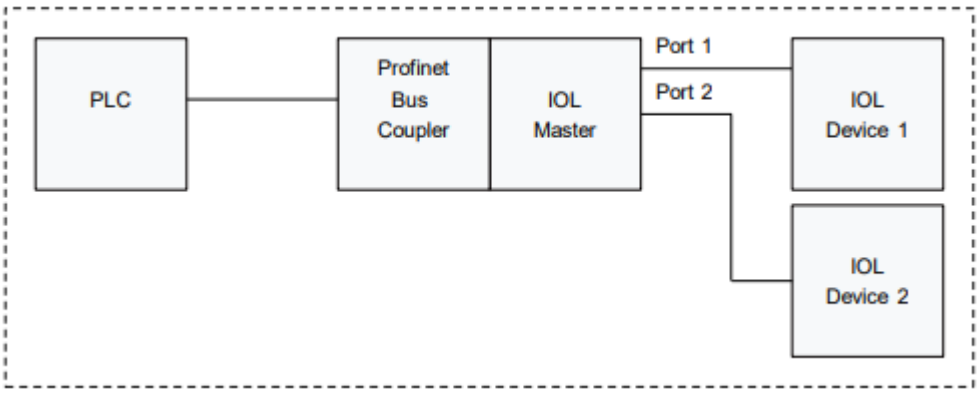
```
(*
** 2nd IOL device
*)
IF
    xStart              = TRUE AND
    udtExample.iExample = 3000
THEN
    CASE udtExample.iState OF
        0: (* Init *)
            (* IOL master = AXL F IOL8 2H - 1027843 *)
            udtExample.wSlot                := WORD#16#0001;
            (* 2nd IOL device = AXL E IOL DI8 M12 6P - 2702658 *)
            udtExample.bSubSlot             := BYTE#16#02;
            (* Reset code *)
            wIndex                          := WORD#16#0101;
            iLen                            := 2;
            (* Deactivate first to have a reliable start condition *)
            udtExample.udtAsynCom.xActivate := FALSE;
            IF udtExample.udtAsynCom.xActive    = FALSE THEN
                udtExample.iState   := 10;
            END_IF;

        10: (* Activate *)
            udtExample.udtAsynCom.xActivate := TRUE;
            IF udtExample.udtAsynCom.xError = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtAsynCom.xActive = TRUE THEN
                udtExample.iState   := 100;
            END_IF;

        100: (* Read *)
            udtExample.udtAsynCom.udtRead.wIndex    := wIndex;
            udtExample.udtAsynCom.udtRead.iMlen     := iLen;
            udtExample.udtAsynCom.udtRead.xReq      := TRUE;
            IF udtExample.udtAsynCom.xError = TRUE THEN
                udtExample.udtAsynCom.udtRead.xReq  := FALSE;
                (* The connected IOL device has no reset code => Error = TRUE *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtAsynCom.udtRead.xDone   = TRUE THEN
                udtExample.udtAsynCom.udtRead.xReq  := FALSE;
                (* You can find the reset code in udtExample.udtAsynCom.udtRead.arrData *)
                udtExample.iState                   := 200;
            END_IF;

        9000: (* Error state *)
            (* The connected IOL device has no reset code => Error = TRUE
            We continue with this example. *)
            udtExample.iState   := 32000;

        32000: (* Finished *)
            (* This part of the example is successfully finished *)
            udtExample.iState   := 32000;
            udtExample.iExample := 32000;
    END_CASE;
END_IF;
```

6.3 Example 3: IOL_X_EXA_AXL_F_IOL8_2H_PN_TPS

6.3.1 Plant

- AXC F 2152 (2404267)
  - AXL F BK PN TPS (2403869)
    - AXL F IOL8 2H (1027843)

6.3.2 External components



IOL device 1 = AXL E IOL DO8 M12 6P (2702659)

IOL device 2 = AXL E IOL DI8 M12 6P (2702658)

6.3.3 Example description

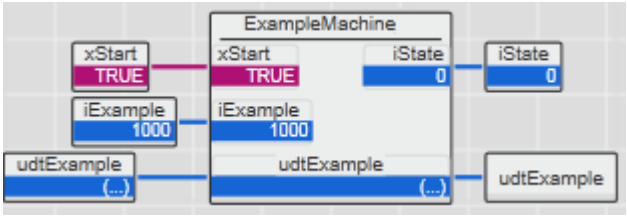
In the example project, we find the function block ExampleMachine. This function block contains a state machine for each example with a detailed description about what we have to do to use the function block correctly.

The following examples can be executed:

Function	iExample	Codesheet
Read / write to IOL master	1000	E1000
Read / write to IOL device 1	2000	E2000
Read / write to IOL device 2	3000	E3000

6.3.3.1 Example machine

For starting ExampleMachine function block, the requested example can be selected at iExample input and xStart input has set to TRUE.



6.3.3.2 State machine

6.3.3.2.1 E1000

```
(*
*** IOL master
*)
IF
    xStart              = TRUE AND
    udtExample.iExample = 1000
THEN
    CASE udtExample.iState OF
        0: (* Init *)
            (* Reset Code *)
            wIndex          := WORD#16#0024;
            iWriteLength    := 16;

            (* IOL master *)
            udtExample.dwNodeID                 := WORD#159;
            (* IOL master *)
            udtExample.udtIolCom.iPort          := 0;
            udtExample.udtIolCom.iMode          := 1;
            udtExample.udtIolCom.xRead_Write    := FALSE;
            udtExample.udtIolCom.xTrigger       := FALSE;
            (* Deactivate to achieve a precise start condition *)
            udtExample.udtIolCom.xActivate      := FALSE;
            IF udtExample.udtIolCom.xActive = FALSE THEN
                udtExample.iState   := 10;
            END_IF;

        10: (* Activate *)
            udtExample.udtIolCom.xActivate  := TRUE;
            IF udtExample.udtIolCom.xError  = TRUE  THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtIolCom.xActive  = TRUE THEN
                udtExample.iState   := 100;
            END_IF;

        100: (* Read *)
            udtExample.udtIolCom.wIndex     := wIndex;
            udtExample.udtIolCom.xTrigger   := TRUE;
            IF udtExample.udtIolCom.xError  = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtIolCom.xDone    = TRUE THEN
                udtExample.udtIolCom.xTrigger   := FALSE;
                (* You can find the reset code in udtExample.arrIOL_ReadData *)
                udtExample.iState               := 200;
            END_IF;

        200: (* Write *)
            udtExample.arrIOL_WriteData[0]      := BYTE#16#00;
            udtExample.arrIOL_WriteData[1]      := BYTE#16#02;
            udtExample.arrIOL_WriteData[2]      := BYTE#16#00;
            udtExample.arrIOL_WriteData[3]      := BYTE#16#02;
```

```
            udtExample.arrIOL_WriteData[4]      := BYTE#16#00;
            udtExample.arrIOL_WriteData[5]      := BYTE#16#02;
            udtExample.arrIOL_WriteData[6]      := BYTE#16#00;
            udtExample.arrIOL_WriteData[7]      := BYTE#16#02;
            udtExample.udtIolCom.wIndex         := wIndex;
            udtExample.udtIolCom.xRead_Write    := TRUE;
            udtExample.udtIolCom.iWriteLength   := iWriteLength;
            udtExample.udtIolCom.xTrigger       := TRUE;
            IF udtExample.udtIolCom.xError  = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtIolCom.xDone    = TRUE THEN
                udtExample.udtIolCom.xTrigger   := FALSE;
                udtExample.iState               := 300;
            END_IF;

        300: (* Read back *)
            udtExample.udtIolCom.wIndex         := wIndex;
            udtExample.udtIolCom.xRead_Write    := FALSE;
            udtExample.udtIolCom.xTrigger       := TRUE;
            IF udtExample.udtIolCom.xError  = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF
                udtExample.udtIolCom.xDone      = TRUE AND
                (* Compare read data with write data *)
                udtExample.arrIOL_ReadData[0]   = BYTE#16#00 AND
                udtExample.arrIOL_ReadData[1]   = BYTE#16#02 AND
                udtExample.arrIOL_ReadData[2]   = BYTE#16#00 AND
                udtExample.arrIOL_ReadData[3]   = BYTE#16#02 AND
                udtExample.arrIOL_ReadData[4]   = BYTE#16#00 AND
                udtExample.arrIOL_ReadData[5]   = BYTE#16#02 AND
                udtExample.arrIOL_ReadData[6]   = BYTE#16#00 AND
                udtExample.arrIOL_ReadData[7]   = BYTE#16#02
            THEN
                udtExample.udtIolCom.xTrigger   := FALSE;
                udtExample.iState               := 32000;
            END_IF;

        9000: (* Error state *)
            (* Implement your error handling here *)
            udtExample.iState   := udtExample.iState;

        32000: (* Finished *)
            (* This part of the example is successfully finished *)
            udtExample.iState   := 32000;
            udtExample.iExample := 32000;
    END_CASE;
END_IF;
```

6.3.3.2.2 E2000

```
(*
** 1st IOL device
*)
IF
    xStart              = TRUE AND
    udtExample.iExample = 2000
THEN
    CASE udtExample.iState OF
        0: (* Init *)
            (* Reset code *)
            wIndex          := WORD#16#0101;
            iWriteLength    := 2;
```

```
    (* IOL master *)
    udtExample.dwNodeID                 := WORD#159;
    (* 1st IOL device *)
    udtExample.udtIolCom.iPort          := 1;
    udtExample.udtIolCom.iMode          := 1;
    udtExample.udtIolCom.xRead_Write    := FALSE;
    udtExample.udtIolCom.xTrigger       := FALSE;
    udtExample.udtIolCom.xActivate      := TRUE;
    IF udtExample.udtIolCom.xError  = TRUE THEN
        (* Goto error handling *)
        udtExample.iState   := 9000;
    ELSIF udtExample.udtIolCom.xActive  = TRUE THEN
        udtExample.iState   := 100;
    END_IF;

100: (* Read *)
    udtExample.udtIolCom.wIndex     := wIndex;
    udtExample.udtIolCom.xTrigger   := TRUE;
    IF udtExample.udtIolCom.xError  = TRUE THEN
        (* Goto error handling *)
        udtExample.iState   := 9000;
    ELSIF udtExample.udtIolCom.xDone    = TRUE THEN
        udtExample.udtIolCom.xTrigger   := FALSE;
        (* You can find the reset code in udtExample.arrIOL_ReadData *)
        udtExample.iState               := 200;
    END_IF;

200: (* Write *)
    udtExample.arrIOL_WriteData[0]      := BYTE#16#00;
    udtExample.arrIOL_WriteData[1]      := BYTE#16#02;
    udtExample.udtIolCom.wIndex         := wIndex;
    udtExample.udtIolCom.xRead_Write    := TRUE;
    udtExample.udtIolCom.iWriteLength   := iWriteLength;
    udtExample.udtIolCom.xTrigger       := TRUE;
    IF udtExample.udtIolCom.xError  = TRUE THEN
        (* Goto error handling *)
        udtExample.iState   := 9000;
    ELSIF udtExample.udtIolCom.xDone    = TRUE THEN
        udtExample.udtIolCom.xTrigger   := FALSE;
        udtExample.iState               := 300;
    END_IF;

300: (* Read back *)
    udtExample.udtIolCom.wIndex         := wIndex;
    udtExample.udtIolCom.xRead_Write    := FALSE;
    udtExample.udtIolCom.xTrigger       := TRUE;
    IF udtExample.udtIolCom.xError  = TRUE THEN
        (* Goto error handling *)
        udtExample.iState   := 9000;
    ELSIF
        udtExample.udtIolCom.xDone          = TRUE AND
        (* Compare read data with write data *)
        udtExample.arrIOL_ReadData[0]   = BYTE#16#00 AND
        udtExample.arrIOL_ReadData[1]   = BYTE#16#02
    THEN
        udtExample.udtIolCom.xTrigger   := FALSE;
        udtExample.iState               := 32000;
    END_IF;

9000: (* Error state *)
    (* Implement your error handling here *)
    udtExample.iState   := udtExample.iState;
```

```
        32000: (* Finished *)
            (* This part of the example is successfully finished *)
            udtExample.iState   := 32000;
            udtExample.iExample := 32000;
    END_CASE;
END_IF;
```

6.3.3.2.3 E3000

```
(*
** 2nd IOL device
*)
IF
    xStart              = TRUE AND
    udtExample.iExample = 3000
THEN
    CASE udtExample.iState OF
        0: (* Init *)
            (* Reset code *)
            wIndex          := WORD#16#0101;
            iWriteLength    := 2;

            (* IOL master *)
            udtExample.dwNodeID                 := WORD#159;
            (* 2nd IOL device *)
            udtExample.udtIolCom.iPort          := 2;
            udtExample.udtIolCom.iMode          := 1;
            udtExample.udtIolCom.xRead_Write    := FALSE;
            udtExample.udtIolCom.xTrigger       := FALSE;
            udtExample.udtIolCom.xActivate      := TRUE;
            IF udtExample.udtIolCom.xError  = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtIolCom.xActive  = TRUE THEN
                udtExample.iState   := 100;
            END_IF;

        100: (* Read *)
            udtExample.udtIolCom.wIndex     := wIndex;
            udtExample.udtIolCom.xTrigger   := TRUE;
            IF udtExample.udtIolCom.xError  = TRUE THEN
                udtExample.udtIolCom.xTrigger   := FALSE;
                (* Because this 2nd IOL device does not have the parameter "reset code"
                this read access causes an error *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtIolCom.xDone    = TRUE THEN
                udtExample.udtIolCom.xTrigger   := FALSE;
                (* You can find the reset code in udtExample.arrIOL_ReadData *)
                udtExample.iState               := 200;
            END_IF;

        9000: (* Error state *)
            (* Implement your error handling here.
            In this example we reset the IOL_COM with deactivation and continue *)
            udtExample.iState   := 9010;

        9010: (* Reset *)
            udtExample.udtIolCom.xActivate  := FALSE;
            IF udtExample.udtIolCom.xActive = FALSE THEN
                udtExample.iState   := 32000;
            END_IF;

        32000: (* Finished *)
            (* This part of the example is successfully finished *)
```

```
            udtExample.iState   := 32000;
            udtExample.iExample := 32000;
    END_CASE;
END_IF;
```

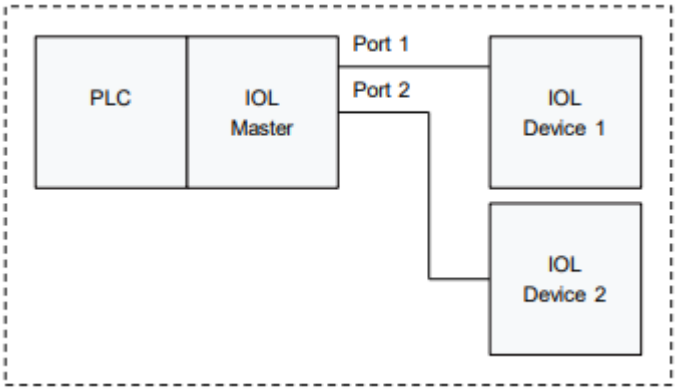
6.4 Example 4: IOL_X_EXA_AXL_SE_IOL4_LOC

Important: Only the AsynCom library is needed, the IOL_Com library is not needed.

6.4.1 Plant

- AXC F 2152 (2404267)
  - AXL SE IOL4 (1088132)

6.4.2 External components



IOL device 1 = AXL E IOL DO8 M12 6P (2702659)

IOL device 2 = AXL E IOL DI8 M12 6P (2702658)

6.4.3 Example description

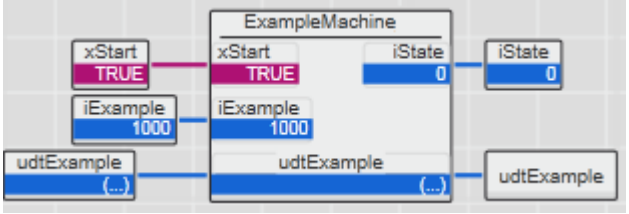
In the example project, we find the function block ExampleMachine. This function block contains a state machine for each example with a detailed description about what we have to do to use the function block correctly.

The following examples can be executed:

Function	iExample	Codesheet
Read / write to IOL master	1000	E1000
Read / write to IOL device 1	2000	E2000
Read / write to IOL device 2	3000	E3000

6.4.3.1 Example machine

For starting ExampleMachine function block, the requested example can be selected at iExample input and xStart input has set to TRUE.



6.4.3.2 State machine

6.4.3.2.1 E1000

```
(*
** IOL master
*)
IF
    xStart              = TRUE AND
    udtExample.iExample = 1000
THEN
    CASE udtExample.iState OF
        0: (* Init *)
            (* IOL master = AXL F IOL8 2H - 1027843 *)
            udtExample.wSlot                := WORD#16#0001;
            (* IOL master = AXL F IOL8 2H - 1027843 *)
            udtExample.bSubSlot             := BYTE#16#00;
            (* Reset code *)
            wIndex                          := WORD#16#0716;
            iLen                            := 8;
            (* Deactivate first to have a reliable start condition *)
            udtExample.udtAsynCom.xActivate := FALSE;
            IF udtExample.udtAsynCom.xActive    = FALSE THEN
                udtExample.iState   := 10;
            END_IF;

        10: (* Activate *)
            udtExample.udtAsynCom.xActivate := TRUE;
            IF udtExample.udtAsynCom.xError = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtAsynCom.xActive = TRUE THEN
                udtExample.iState   := 100;
            END_IF;

        100: (* Read *)
            udtExample.udtAsynCom.udtRead.wIndex    := wIndex;
            udtExample.udtAsynCom.udtRead.iMlen     := iLen;
            udtExample.udtAsynCom.udtRead.xReq      := TRUE;
            IF udtExample.udtAsynCom.xError = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtAsynCom.udtRead.xDone   = TRUE THEN
                udtExample.udtAsynCom.udtRead.xReq  := FALSE;
                (* You can find the reset code in udtExample.udtAsynCom.udtRead.arrData *)
                udtExample.iState   := 200;
            END_IF;

        200: (* Write *)
            udtExample.udtAsynCom.udtWrite.arrData[1]   := BYTE#16#00;
            udtExample.udtAsynCom.udtWrite.arrData[2]   := BYTE#16#02;
            udtExample.udtAsynCom.udtWrite.arrData[3]   := BYTE#16#00;
            udtExample.udtAsynCom.udtWrite.arrData[4]   := BYTE#16#02;
            udtExample.udtAsynCom.udtWrite.arrData[5]   := BYTE#16#00;
            udtExample.udtAsynCom.udtWrite.arrData[6]   := BYTE#16#02;
            udtExample.udtAsynCom.udtWrite.arrData[7]   := BYTE#16#00;
            udtExample.udtAsynCom.udtWrite.arrData[8]   := BYTE#16#02;
            udtExample.udtAsynCom.udtWrite.wIndex       := wIndex;
            udtExample.udtAsynCom.udtWrite.iLen         := iLen;
            udtExample.udtAsynCom.udtWrite.xReq         := TRUE;
            IF udtExample.udtAsynCom.xError = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtAsynCom.udtWrite.xDone  = TRUE THEN
```

```
                udtExample.udtAsynCom.udtWrite.xReq := FALSE;
                udtExample.iState                   := 300;
            END_IF;

        300: (* Read back *)
            udtExample.udtAsynCom.udtRead.wIndex    := wIndex;
            udtExample.udtAsynCom.udtRead.iMlen     := iLen;
            udtExample.udtAsynCom.udtRead.xReq      := TRUE;
            IF udtExample.udtAsynCom.xError = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF
                udtExample.udtAsynCom.udtRead.xDone         = TRUE AND
                (* Compare read data with write data *)
                udtExample.udtAsynCom.udtRead.arrData[1]    = BYTE#16#00 AND
                udtExample.udtAsynCom.udtRead.arrData[2]    = BYTE#16#02 AND
                udtExample.udtAsynCom.udtRead.arrData[3]    = BYTE#16#00 AND
                udtExample.udtAsynCom.udtRead.arrData[4]    = BYTE#16#02 AND
                udtExample.udtAsynCom.udtRead.arrData[5]    = BYTE#16#00 AND
                udtExample.udtAsynCom.udtRead.arrData[6]    = BYTE#16#02 AND
                udtExample.udtAsynCom.udtRead.arrData[7]    = BYTE#16#00 AND
                udtExample.udtAsynCom.udtRead.arrData[8]    = BYTE#16#02
            THEN
                udtExample.udtAsynCom.udtRead.xReq  := FALSE;
                udtExample.iState                   := 32000;
            END_IF;

        9000: (* Error state *)
            (* Implement your error handling here *)
            udtExample.iState   := udtExample.iState;

        32000: (* Finished *)
            (* This part of the example is successfully finished *)
            udtExample.iState   := 32000;
            udtExample.iExample := 32000;
    END_CASE;
END_IF;
```

6.4.3.2.2 E2000

```
(*
** 1st IOL device
*)
IF
    xStart              = TRUE AND
    udtExample.iExample = 2000
THEN
    CASE udtExample.iState OF
        0: (* Init *)
            (* IOL master *)
            udtExample.wSlot                := WORD#16#0001;
            (* 1st IOL device = AXL E IOL DO8 M12 6P - 2702659 *)
            udtExample.bSubSlot             := BYTE#16#01;
            (* Reset code *)
            wIndex                          := WORD#16#0101;
            iLen                            := 2;
            (* Deactivate first to have a reliable start condition *)
            udtExample.udtAsynCom.xActivate := FALSE;
            IF udtExample.udtAsynCom.xActive    = FALSE THEN
                udtExample.iState   := 10;
            END_IF;

        10: (* Activate *)
            udtExample.udtAsynCom.xActivate := TRUE;
```

```
            IF udtExample.udtAsynCom.xError = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtAsynCom.xActive = TRUE THEN
                udtExample.iState   := 100;
            END_IF;

        100: (* Read *)
            udtExample.udtAsynCom.udtRead.wIndex    := wIndex;
            udtExample.udtAsynCom.udtRead.iMlen     := iLen;
            udtExample.udtAsynCom.udtRead.xReq      := TRUE;
            IF udtExample.udtAsynCom.xError = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtAsynCom.udtRead.xDone   = TRUE THEN
                udtExample.udtAsynCom.udtRead.xReq  := FALSE;
                (* You can find the reset code in udtExample.udtAsynCom.udtRead.arrData *)
                udtExample.iState                   := 200;
            END_IF;

        200: (* Write *)
            udtExample.udtAsynCom.udtWrite.arrData[1]   := BYTE#16#00;
            udtExample.udtAsynCom.udtWrite.arrData[2]   := BYTE#16#02;
            udtExample.udtAsynCom.udtWrite.wIndex       := wIndex;
            udtExample.udtAsynCom.udtWrite.iLen         := iLen;
            udtExample.udtAsynCom.udtWrite.xReq         := TRUE;
            IF udtExample.udtAsynCom.xError = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtAsynCom.udtWrite.xDone  = TRUE THEN
                udtExample.udtAsynCom.udtWrite.xReq := FALSE;
                udtExample.iState                   := 300;
            END_IF;

        300: (* Read back *)
            udtExample.udtAsynCom.udtRead.wIndex    := wIndex;
            udtExample.udtAsynCom.udtRead.iMlen     := iLen;
            udtExample.udtAsynCom.udtRead.xReq      := TRUE;
            IF udtExample.udtAsynCom.xError = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF
                udtExample.udtAsynCom.udtRead.xDone         = TRUE AND
                (* Compare read data with write data *)
                udtExample.udtAsynCom.udtRead.arrData[1]    = BYTE#16#00 AND
                udtExample.udtAsynCom.udtRead.arrData[2]    = BYTE#16#02
            THEN
                udtExample.udtAsynCom.udtRead.xReq  := FALSE;
                udtExample.iState                   := 32000;
            END_IF;

        9000: (* Error state *)
            (* Implement your error handling here *)
            udtExample.iState   := udtExample.iState;

        32000: (* Finished *)
            (* This part of the example is successfully finished *)
            udtExample.iState   := 32000;
            udtExample.iExample := 32000;
    END_CASE;
END_IF;
```

6.4.3.2.3 E3000

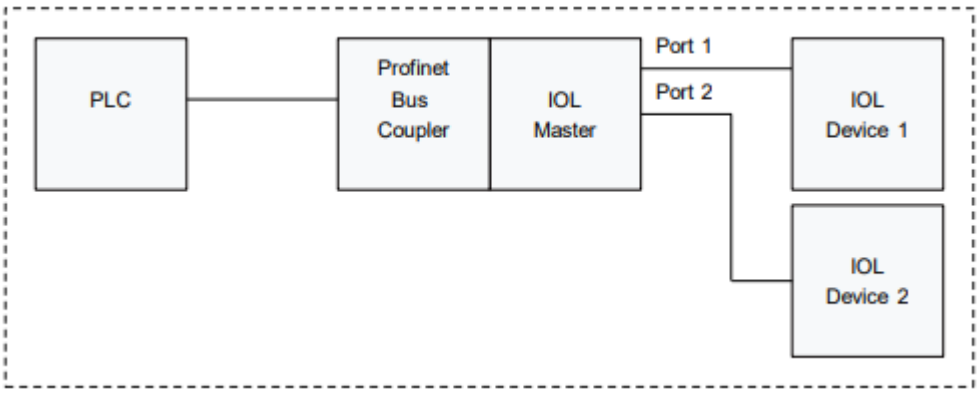
```
(*
** 2nd IOL device
*)
IF
    xStart              = TRUE AND
    udtExample.iExample = 3000
THEN
    CASE udtExample.iState OF
        0: (* Init *)
            (* IOL master = AXL F IOL8 2H - 1027843 *)
            udtExample.wSlot                := WORD#16#0001;
            (* 2nd IOL device = AXL E IOL DI8 M12 6P - 2702658 *)
            udtExample.bSubSlot             := BYTE#16#02;
            (* Reset code *)
            wIndex                          := WORD#16#0101;
            iLen                            := 2;
            (* Deactivate first to have a reliable start condition *)
            udtExample.udtAsynCom.xActivate := FALSE;
            IF udtExample.udtAsynCom.xActive    = FALSE THEN
                udtExample.iState   := 10;
            END_IF;

        10: (* Activate *)
            udtExample.udtAsynCom.xActivate := TRUE;
            IF udtExample.udtAsynCom.xError = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtAsynCom.xActive = TRUE THEN
                udtExample.iState   := 100;
            END_IF;

        100: (* Read *)
            udtExample.udtAsynCom.udtRead.wIndex    := wIndex;
            udtExample.udtAsynCom.udtRead.iMlen     := iLen;
            udtExample.udtAsynCom.udtRead.xReq      := TRUE;
            IF udtExample.udtAsynCom.xError = TRUE THEN
                udtExample.udtAsynCom.udtRead.xReq  := FALSE;
                (* The connected IOL device has no reset code => Error = TRUE *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtAsynCom.udtRead.xDone   = TRUE THEN
                udtExample.udtAsynCom.udtRead.xReq  := FALSE;
                (* You can find the reset code in udtExample.udtAsynCom.udtRead.arrData *)
                udtExample.iState                   := 200;
            END_IF;

        9000: (* Error state *)
            (* The connected IOL device has no reset code => Error = TRUE
            We continue with this example. *)
            udtExample.iState   := 32000;

        32000: (* Finished *)
            (* This part of the example is successfully finished *)
            udtExample.iState   := 32000;
            udtExample.iExample := 32000;
        END_CASE;
END_IF;
```

6.5 Example 5: IOL_X_EXA_AXL_SE_IOL4_PN_TPS

6.5.1 Plant

- AXC F 2152 (2404267)
  - AXL F BK PN TPS (2403869)
    - AXL SE IOL4 (1088132)

6.5.2 External components



IOL device 1 = AXL E IOL DO8 M12 6P (2702659)

IOL device 2 = AXL E IOL DI8 M12 6P (2702658)

6.5.3 Example description

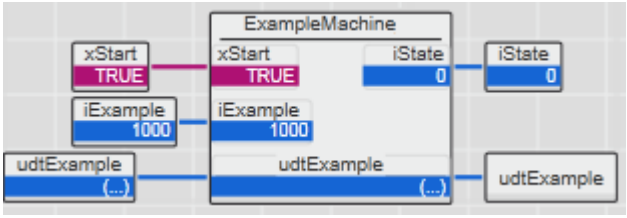
In the example project, we find the function block ExampleMachine. This function block contains a state machine for each example with a detailed description about what we have to do to use the function block correctly.

The following examples can be executed:

Function	iExample	Codesheet
Read / write to IOL master	1000	E1000
Read / write to IOL device 1	2000	E2000
Read / write to IOL device 2	3000	E3000

6.5.3.1 Example machine

For starting ExampleMachine function block, the requested example can be selected at iExample input and xStart input has set to TRUE.



6.5.3.2 State machine

6.5.3.2.1 E1000

```
(*
** IOL master
*)
IF
    xStart              = TRUE AND
    udtExample.iExample = 1000
THEN
    CASE udtExample.iState OF
        0: (* Init *)
            (* Reset Code *)
            wIndex          := WORD#16#0024;
            iWriteLength    := 8;

            (* IOL master *)
            udtExample.dwNodeID                 := WORD#37;
            (* IOL master *)
            udtExample.udtIolCom.iPort          := 0;
            udtExample.udtIolCom.iMode          := 1;
            udtExample.udtIolCom.xRead_Write    := FALSE;
            udtExample.udtIolCom.xTrigger       := FALSE;
            (* Deactivate to achieve a precise start condition *)
            udtExample.udtIolCom.xActivate      := FALSE;
            IF udtExample.udtIolCom.xActive = FALSE THEN
                udtExample.iState   := 10;
            END_IF;

        10: (* Activate *)
            udtExample.udtIolCom.xActivate  := TRUE;
            IF udtExample.udtIolCom.xError  = TRUE  THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtIolCom.xActive  = TRUE THEN
                udtExample.iState   := 100;
            END_IF;

        100: (* Read *)
            udtExample.udtIolCom.wIndex     := wIndex;
            udtExample.udtIolCom.xTrigger   := TRUE;
            IF udtExample.udtIolCom.xError  = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtIolCom.xDone    = TRUE THEN
                udtExample.udtIolCom.xTrigger   := FALSE;
                (* You can find the reset code in udtExample.arrIOL_ReadData *)
                udtExample.iState               := 200;
            END_IF;

        200: (* Write *)
            udtExample.arrIOL_WriteData[0]      := BYTE#16#00;
            udtExample.arrIOL_WriteData[1]      := BYTE#16#02;
            udtExample.arrIOL_WriteData[2]      := BYTE#16#00;
            udtExample.arrIOL_WriteData[3]      := BYTE#16#02;
```

```
            udtExample.arrIOL_WriteData[4]      := BYTE#16#00;
            udtExample.arrIOL_WriteData[5]      := BYTE#16#02;
            udtExample.arrIOL_WriteData[6]      := BYTE#16#00;
            udtExample.arrIOL_WriteData[7]      := BYTE#16#02;
            udtExample.udtIolCom.wIndex         := wIndex;
            udtExample.udtIolCom.xRead_Write    := TRUE;
            udtExample.udtIolCom.iWriteLength   := iWriteLength;
            udtExample.udtIolCom.xTrigger       := TRUE;
            IF udtExample.udtIolCom.xError  = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtIolCom.xDone    = TRUE THEN
                udtExample.udtIolCom.xTrigger   := FALSE;
                udtExample.iState               := 300;
            END_IF;

        300: (* Read back *)
            udtExample.udtIolCom.wIndex         := wIndex;
            udtExample.udtIolCom.xRead_Write    := FALSE;
            udtExample.udtIolCom.xTrigger       := TRUE;
            IF udtExample.udtIolCom.xError  = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF
                udtExample.udtIolCom.xDone      = TRUE AND
                (* Compare read data with write data *)
                udtExample.arrIOL_ReadData[0]   = BYTE#16#00 AND
                udtExample.arrIOL_ReadData[1]   = BYTE#16#02 AND
                udtExample.arrIOL_ReadData[2]   = BYTE#16#00 AND
                udtExample.arrIOL_ReadData[3]   = BYTE#16#02 AND
                udtExample.arrIOL_ReadData[4]   = BYTE#16#00 AND
                udtExample.arrIOL_ReadData[5]   = BYTE#16#02 AND
                udtExample.arrIOL_ReadData[6]   = BYTE#16#00 AND
                udtExample.arrIOL_ReadData[7]   = BYTE#16#02
            THEN
                udtExample.udtIolCom.xTrigger   := FALSE;
                udtExample.iState               := 32000;
            END_IF;

        9000: (* Error state *)
            (* Implement your error handling here *)
            udtExample.iState   := udtExample.iState;

        32000: (* Finished *)
            (* This part of the example is successfully finished *)
            udtExample.iState   := 32000;
            udtExample.iExample := 32000;
    END_CASE;
END_IF;
```

6.5.3.2.2 E2000

```
(*
** 1st IOL device
*)
IF
    xStart              = TRUE AND
    udtExample.iExample = 2000
THEN
    CASE udtExample.iState OF
        0: (* Init *)
            (* Reset code *)
            wIndex          := WORD#16#0101;
            iWriteLength    := 2;
```

```
    (* IOL master *)
    udtExample.dwNodeID                 := WORD#37;
    (* 1st IOL device *)
    udtExample.udtIolCom.iPort          := 1;
    udtExample.udtIolCom.iMode          := 1;
    udtExample.udtIolCom.xRead_Write    := FALSE;
    udtExample.udtIolCom.xTrigger       := FALSE;
    udtExample.udtIolCom.xActivate      := TRUE;
    IF udtExample.udtIolCom.xError  = TRUE THEN
        (* Goto error handling *)
        udtExample.iState   := 9000;
    ELSIF udtExample.udtIolCom.xActive  = TRUE THEN
        udtExample.iState   := 100;
    END_IF;

100: (* Read *)
    udtExample.udtIolCom.wIndex     := wIndex;
    udtExample.udtIolCom.xTrigger   := TRUE;
    IF udtExample.udtIolCom.xError  = TRUE THEN
        (* Goto error handling *)
        udtExample.iState   := 9000;
    ELSIF udtExample.udtIolCom.xDone    = TRUE THEN
        udtExample.udtIolCom.xTrigger   := FALSE;
        (* You can find the reset code in udtExample.arrIOL_ReadData *)
        udtExample.iState               := 200;
    END_IF;

200: (* Write *)
    udtExample.arrIOL_WriteData[0]      := BYTE#16#00;
    udtExample.arrIOL_WriteData[1]      := BYTE#16#02;
    udtExample.udtIolCom.wIndex         := wIndex;
    udtExample.udtIolCom.xRead_Write    := TRUE;
    udtExample.udtIolCom.iWriteLength   := iWriteLength;
    udtExample.udtIolCom.xTrigger       := TRUE;
    IF udtExample.udtIolCom.xError  = TRUE THEN
        (* Goto error handling *)
        udtExample.iState   := 9000;
    ELSIF udtExample.udtIolCom.xDone    = TRUE THEN
        udtExample.udtIolCom.xTrigger   := FALSE;
        udtExample.iState               := 300;
    END_IF;

300: (* Read back *)
    udtExample.udtIolCom.wIndex         := wIndex;
    udtExample.udtIolCom.xRead_Write    := FALSE;
    udtExample.udtIolCom.xTrigger       := TRUE;
    IF udtExample.udtIolCom.xError  = TRUE THEN
        (* Goto error handling *)
        udtExample.iState   := 9000;
    ELSIF
        udtExample.udtIolCom.xDone          = TRUE
        (* Compare read data with write data *)
        AND udtExample.arrIOL_ReadData[0]   = BYTE#16#00
        AND udtExample.arrIOL_ReadData[1]   = BYTE#16#02
    THEN
        udtExample.udtIolCom.xTrigger   := FALSE;
        udtExample.iState               := 32000;
    END_IF;

9000: (* Error state *)
    (* Implement your error handling here *)
    udtExample.iState   := udtExample.iState;
```

```
        32000: (* Finished *)
            (* This part of the example is successfully finished *)
            udtExample.iState   := 32000;
            udtExample.iExample := 32000;
    END_CASE;
END_IF;
```

6.5.3.2.3 E3000

```
(*
** 2nd IOL device
*)
IF
    xStart              = TRUE AND
    udtExample.iExample = 3000
THEN
    CASE udtExample.iState OF
        0: (* Init *)
            (* Reset code *)
            wIndex          := WORD#16#0101;
            iWriteLength    := 2;

            (* IOL master *)
            udtExample.dwNodeID                 := WORD#37;
            (* 2nd IOL device *)
            udtExample.udtIolCom.iPort          := 2;
            udtExample.udtIolCom.iMode          := 1;
            udtExample.udtIolCom.xRead_Write    := FALSE;
            udtExample.udtIolCom.xTrigger       := FALSE;
            udtExample.udtIolCom.xActivate      := TRUE;
            IF udtExample.udtIolCom.xError  = TRUE THEN
                (* Goto error handling *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtIolCom.xActive  = TRUE THEN
                udtExample.iState   := 100;
            END_IF;

        100: (* Read *)
            udtExample.udtIolCom.wIndex     := wIndex;
            udtExample.udtIolCom.xTrigger   := TRUE;
            IF udtExample.udtIolCom.xError  = TRUE THEN
                udtExample.udtIolCom.xTrigger   := FALSE;
                (* Because this 2nd IOL device does not have the parameter "reset code"
                this read access causes an error *)
                udtExample.iState   := 9000;
            ELSIF udtExample.udtIolCom.xDone    = TRUE THEN
                udtExample.udtIolCom.xTrigger   := FALSE;
                (* You can find the reset code in udtExample.arrIOL_ReadData *)
                udtExample.iState               := 200;
            END_IF;

        9000: (* Error state *)
            (* Implement your error handling here.
            In this example we reset the IOL_COM with deactivation and continue *)
            udtExample.iState   := 9010;

        9010: (* Reset *)
            udtExample.udtIolCom.xActivate  := FALSE;
            IF udtExample.udtIolCom.xActive = FALSE THEN
                udtExample.iState   := 32000;
            END_IF;

        32000: (* Finished *)
            (* This part of the example is successfully finished *)
```

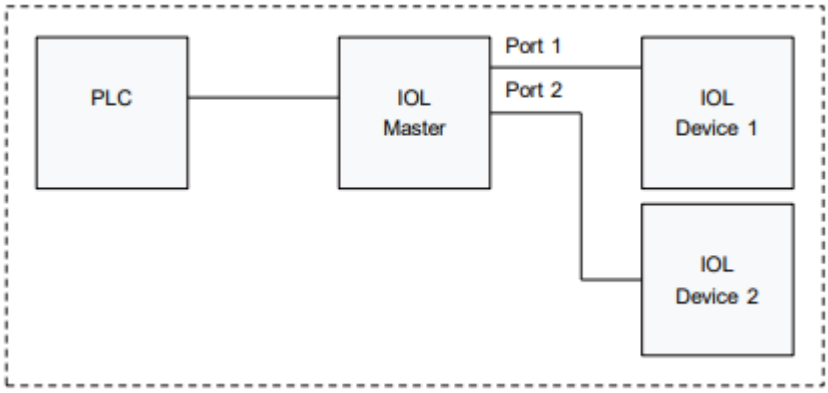
```
            udtExample.iState   := 32000;
            udtExample.iExample := 32000;
    END_CASE;
END_IF;
```

6.6 Example 6: IOL_X_EXA_MA8_PN_DI8

6.6.1 Plant

- AXC F 2152 (2404267)
  - IOL MA8 PN DI8 (1072838)

6.6.2 External components



IOL device 1 = AXL E IOL DO8 M12 6P (2702659)

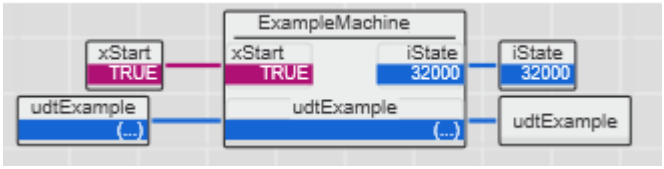
IOL device 2 = AXL E IOL DI8 M12 6P (2702658)

6.6.3 Example description

Read article numbers of IOL devices.

6.6.3.1 Example machine

For starting ExampleMachine function block, xStart input has set to TRUE.



6.6.3.2 State machine

```
CASE iState OF

    0: (* Wait for start of example program *)
        iIndex  := 0;
        IF xStart   = TRUE THEN
            iState  := 1;
        END_IF;

    1: (* Init *)
        iIteration                  := 0;
        iIterationMax               := 1;
```

```
    udtExample.xActivate        := FALSE;
    udtExample.xPNIO_Data_Valid := TRUE;
    udtExample.udtConfig.wIndex := WORD#0;
    udtExample.iMode            := 0;
    udtExample.xTrigger         := FALSE;
    udtExample.xRead_Write      := FALSE;
    udtExample.iPort            := 0;
    (* Article number *)
    udtExample.wIndex           := WORD#16#13;
    udtExample.bSubIndex        := BYTE#0;
    udtExample.iWriteLength     := 1;

    IF udtExample.xActive   = FALSE THEN
        iState  := 10;
    END_IF;

10: (* Loop *)
    IF iIteration   = 0 THEN
        (* AXL E IOL DO8 M12 6P - 2702659 *)
        udtExample.iPort    := 1;
        (* Reference value *)
        strRef              := '2702659';
    ELSE
        (* AXL E IOL DI8 M12 6P - 2702658 *)
        udtExample.iPort    := 2;
        (* Reference value *)
        strRef              := '2702658';
    END_IF;
    (* Activating of function block and starting read instruction *)
    udtExample.xActivate    := TRUE;
    udtExample.xTrigger     := TRUE;
    IF udtExample.xError    = TRUE THEN
        (* Goto error handling *)
        iState  := 9000;
    ELSIF udtExample.xDone  = TRUE THEN
        (* Copy elementary data types from a byte stream to a variable *)
        BUF_TO_STRING_X1.REQ        := TRUE;
        BUF_TO_STRING_X1.BUF_OFFS   := DINT#0;
        BUF_TO_STRING_X1.BUF_CNT    := DINT#7;
        iState                      := 20;
    END_IF;

20: (* When copy operation is complete *)
    IF
        BUF_TO_STRING_X1.DONE   = TRUE AND
        EQ_STRING(strTmp, strRef) = TRUE
    THEN
        udtExample.xTrigger     := FALSE;
        BUF_TO_STRING_X1.REQ    := FALSE;
        iState                  := 100;
    ELSE
        (* Goto error handling *)
        iState  := 9000;
    END_IF;

100: (* If iteration is finished then go to next state *)
    IF iIteration   = iIterationMax THEN
        iState      := 32000;
    ELSE
        iIteration  := iIteration + 1;
        iState      := 10;
    END_IF;

9000: (* Error state *)
```

```
        (* Implement your error handling here *)
        iState  := 9000;

    32000: (* End *)
        IF xStart   = FALSE THEN
            (* Restart *)
            iState  := 0;
        END_IF;

END_CASE;
```

7 Appendix

7.1 IO-Link diagnostics[1,2]

[1] IO-Link Community, IOL-Interface-Spec_10002_V112_Jul13, page 85
[2] IO-Link Community, IOL-Comm-Spec_10002_V10_090118.doc, pages 115-116

7.1.1 IOL-M Error_Codes

IOL-M Error_Code	Coding	Definitions
No error	16#0000	No IOL-M or IOL-Server errors.
-	16#0001 - 16#06FF	Reserved for future use.
IOL_Call conflict	16#7000	Unexpected write req instead of read req.
Wrong IOL_Call	16#7001	Decode error.
Port blocked	16#7002	Port occupied by another task.
-	16#7003 - 16#7FFF	Reserved for future use.
Timeout	16#8000	Timeout when IOL-Ds or IOL-M ports are busy.
Wrong index	16#8001	IOL_Index >32767 or <65535
Wrong port address	16#8002	Port address beyond defined maximum.
Wrong port function	16#8003	Port function not supported.
-	16#8004 - 16#FFFF	Reserved for future use.

7.1.2 Error codes

Type	Origin	Name	Category	Mode	Inst.	Value	Remark
PDU buffer overflow	remote	S_PDU_BUFFER	ERROR	SINGLE SHOT	DL	16#52	Device buffer is too small for storing the complete PDU.
PDU checksum error (master)	local	M_PDU_CHECK	ERROR	SINGLE SHOT	DL	16#56	Calculated PDU checksum in master does not match actual received SPDU.
PDU checksum error (device)	remote	S_PDU_CHECK	ERROR	SINGLE SHOT	DL	16#56	Calculated PDU checksum in device does not match actual received SPDU.
PDU flow control error	remote	S_PDU_FLOW	ERROR	SINGLE SHOT	DL	16#56	Violation of flow control rule during transfer of SPDU between master and device.
Illegal PDU service primitive (master)	local	M_PDU_ILLEGAL	ERROR	SINGLE SHOT	AL	16#57	Unknown service primitive or wrong response e.g. Read response or Write request
Illegal PDU service primitive (device)	remote	S_PDU_ILLEGAL	ERROR	SINGLE SHOT	AL	16#58	Unknown service primitive e.g. different protocol revision.
Communication error	local / remote	COM_ERR	ERROR	SINGLE SHOT	unknown	16#10	Negative service response initiated by a communication error, e.g. IO-Link connection interrupted
Device / application error	remote	S_APP_DEV	ERROR	SINGLE SHOT	APP	16#80	Service PDU transferred, but not processed due to device error. See error details in Additional Code.

7.1.3 Additional codes for error code S_APP_DEV

Type	Value	Remark
No details	16#00	-
Index not available	16#11	-
Subindex not available	16#12	-
Service temporarily not available	16#20	-
Service temporarily not available, local control	16#21	local operation at device inhibits access
Service temporarily not available, device control	16#22	device state does not allow access, e.g. during teach or calibration
Access denied	16#23	e.g. write on read-only parameters
Parameter value out of range	16#30	-
Parameter value above limit	16#31	-
Parameter value below limit	16#32	-
Interfering parameter	16#40	collision with other parameter values
Application failure	16#81	e.g. separate application and communication processors
Application not ready	16#82	e.g. separate application and communication processors

7.2 Diagnosis of used firmware function blocks

7.2.1 PDI_READ

for PLCnext Engineer

ERROR = TRUE

STATUS[0]	STATUS[1]	Meaning
16#09B0	16#000C	The variable connected to RD_1 is invalid (no array or invalid array type).
16#09B0	16#000B	The array connected to RD_1 is too small to save the requested receive data.
16#09B0	16#000E	Timeout. No response to the sent PDI READ request received.
16#09B0	16#000F	An internal error has occurred.

When receiving a negative confirmation as response to a PDI_READ request, the Axioline module directly copies the received error code (Error_Code and Add_Info) to STATUS[0] or STATUS[1]. These error codes are module-specific. For a description see the respective module documentation.

7.2.2 PDI_WRITE

for PLCnext Engineer

ERROR = TRUE

STATUS[0]	STATUS[1]	Meaning
16#09B0	16#000A	The variable connected to SD_1 is invalid (no array or invalid array type).
16#09B0	16#0009	Invalid value at DATA_CNT input. The value is either greater than the array connected to SD_1, greater than the maximum allowed length (245 bytes) or equal to zero.
16#09B0	16#000E	Timeout. No response to the sent PDI WRITE request received.
16#09B0	16#000F	An internal error has occurred.

When receiving a negative confirmation as response to a PDI_WRITE request, the Axioline module directly copies the received error code (Error_Code and Add_Info) to STATUS[0] or STATUS[1]. These error codes are module-specific. For a description see the respective module documentation.

7.2.3 RDREC

for PLCnext Engineer

Error code (hex)	Meaning
16#0000	No error occurred.
16#F001	Too many instances used.
16#F002	Error during initialization of the function block.
16#F003	Invalid ID.
16#F004	Invalid HANDLE/ID.
16#F005	Resources conflict.
16#F006	A function block internal task could not be generated.
16#F007	Too many instances used.
16#F008	Invalid type of a parameter.
16#F009	Invalid parameter value.
16#F00A	Unallowed parameter.
16#F00B	Invalid length specified.
16#F00C	ID could not be created (too many IDs).
16#F00D	No entry found that matches the specified ID.
16#F00F	No further entries found.
16#F010	Entry in use.
16#F011	Alarm acknowledgement could not be done.
16#F012	Error reading the AR parameters (1st time).
16#F013	Negative acknowledgement received for the execution of a PROFINET service.
16#F014	Invalid length for parameter LEN/MLEN or/and RECORD data record too short.
16#F015	The service used to read the RECORD data record could not be run.
16#F016	The service used to write the RECORD data record could not be run.
16#F017	Service acknowledgement not received.
16#F018	Invalid INDEX used to access the RECORD data record of the IO device, for example, INDEX greater than 16#7FFF.
16#F019	Unknown command code.
16#F01A	Error starting the Application Relation (AR).
16#F01B	Error stopping the Application Relation (AR).
16#F01C	Notification of stopped Application Relation (AR) failed.
16#F01D	Setting the "Drive BF" flag failed.
16#F01E	Error reading the AR parameters (2nd time).

7.2.4 WRREC

for PLCnext Engineer

Error code (hex)	Meaning
16#0000	No error occurred.
16#F001	Too many instances used.
16#F002	Error during initialization of the function block.
16#F003	Invalid ID.
16#F004	Invalid HANDLE/ID.
16#F005	Resources conflict.
16#F006	A function block internal task could not be generated.
16#F007	Too many instances used.
16#F008	Invalid type of a parameter.
16#F009	Invalid parameter value.
16#F00A	Unallowed parameter.
16#F00B	Invalid length specified.
16#F00C	ID could not be created (too many IDs).
16#F00D	No entry found that matches the specified ID.
16#F00F	No further entries found.
16#F010	Entry in use.
16#F011	Alarm acknowledgement could not be done.
16#F012	Error reading the AR parameters (1st time).
16#F013	Negative acknowledgement received for the execution of a PROFINET service.
16#F014	Invalid length for parameter LEN/MLEN or/and RECORD data record too short.
16#F015	The service used to read the RECORD data record could not be run.
16#F016	The service used to write the RECORD data record could not be run.
16#F017	Service acknowledgement not received.
16#F018	Invalid INDEX used to access the RECORD data record of the IO device, for example, INDEX greater than 16#7FFF.
16#F019	Unknown command code.
16#F01A	Error starting the Application Relation (AR).
16#F01B	Error stopping the Application Relation (AR).
16#F01C	Notification of stopped Application Relation (AR) failed.
16#F01D	Setting the "Drive BF" flag failed.
16#F01E	Error reading the AR parameters (2nd time).

7.3 Data types

```
TYPE
    (* Array of read / write data *)
    IOL_ARR_DATA    : ARRAY [0..231] OF BYTE;

    (* Config data *)
    IOL_UDT_CONF    : STRUCT
        wIndex      : WORD;
        tRetry      : TIME; (* Time between before retry *)
        tTimeout    : TIME; (* Global timeout *)
    END_STRUCT;

    (* Array witg diag data *)
    IOL_UDT_DIAG    : STRUCT
        iState  : INT;
    END_STRUCT;
END_TYPE
```

8 In case of problems

In case of problems you should check:

1. Is the latest version of the library used? You can download the latest version from the PLCnext Store (https://www.plcnextstore.com).
2. Is the library released for the PLCnext Engineer version you are using? Please refer to the chapter “Change notes” of this documentation.
3. Is the library released for the PLC / PLC firmware version you are using? Please refer to the chapter “Change notes” of this documentation.
4. Does every module in your system has the required firmware version? In some cases functionalities are supported for higher firmware versions only.
5. Is your system running? Check all LEDs of all modules. Check the status in the web based management of your PLC.
6. Is your IOL device connected on the correct port of your IOL master? There are IOL-A type and IOL-B type ports. Check what your IOL device needs.
7. Is the correct IOL cable used? There are IOL-A type and IOL-B type cables.
8. Is it needed to set IOL_COM.udtConfig.wIndex for your IOL master? Please refer to “IOL_COM / Input parameters”.
9. Is IOL_COM.iPort set to the IOL port that your are using?
10. Is IOL_COM.iMode set correctly for your IOL master? Please refer to “IOL_COM / Input parameters”.

9 Support

For technical support please contact your local PHOENIX CONTACT agency

at https://www.phoenixcontact.com

Owner:

PHOENIX CONTACT Electronics GmbH
Business Unit Automation Systems
System Services
Library Services

In case of a support request we need:

- Development system with
  - Name (e.g. PC Worx, PLCnext Engineer)
  - Version (e.g. PLCnext Engineer 2022.0.1 LTS)
- Bus structure / plant including all articles with
  - Name
  - Order number
  - Firmware version
- External components
- Used libraries with
  - Name
  - Version (e.g. IOL_Basic_7)
- Detailed problem description with
  - Diag codes of all function blocks in error state