



WITTENSTEIN

alpha

Getting started

**cynapse<sup>®</sup>**  
Beckhoff PLC – Beckhoff IO-Link master

## WITTENSTEIN alpha GmbH

Walter-Wittenstein-Straße 1  
D-97999 Igersheim  
Germany

### Cybertronic support

If you have questions about this implementation example, please contact:

[cybertronic-support@wittenstein.de](mailto:cybertronic-support@wittenstein.de)

### Customer service

		✉	☎
Deutschland	WITTENSTEIN alpha GmbH	service@wittenstein-alpha.de	+49 7931 493-12900
Benelux	WITTENSTEIN BVBA	service@wittenstein.biz	+32 9 326 73 80
Brasil	WITTENSTEIN do Brasil	vendas@wittenstein.com.br	+55 15 3411 6454
中国	威腾斯坦（杭州）实业有限公司	service@wittenstein.cn	+86 571 8869 5856
Österreich	WITTENSTEIN GmbH	office@wittenstein.at	+43 2256 65632-0
Danmark	WITTENSTEIN AB	info@wittenstein.dk	+45 4027 4151
France	WITTENSTEIN sarl	info@wittenstein.fr	+33 134 17 90 95
Great Britain	WITTENSTEIN Ltd.	sales.uk@wittenstein.co.uk	+44 1782 286 427
Italia	WITTENSTEIN S.P.A.	info@wittenstein.it	+39 02 241357-1
日本	ヴィッテンシュタイン株式会社	sales@wittenstein.jp	+81-3-6680-2835
North America	WITTENSTEIN holding Corp.	technicalsupport@wittenstein-us.com	+1 630-540-5300
España	WITTENSTEIN S.L.U.	info@wittenstein.es	+34 93 479 1305
Sverige	WITTENSTEIN AB	info@wittenstein.se	+46 40-26 50 10
Schweiz	WITTENSTEIN AG Schweiz	sales@wittenstein.ch	+41 81 300 10 30
台湾	威騰斯坦有限公司	info@wittenstein.tw	+886 3 287 0191
Türkiye	WITTENSTEIN Güç Aktarma Sistemleri Tic. Ltd. Şti.	info@wittenstein.com.tr	+90 216 709 21 23

© WITTENSTEIN alpha GmbH 2022

Subject to technical and content changes without notice.

## Table of contents

<b>1</b>	<b>About this manual</b> .....	<b>2</b>
1.1	Information symbols and cross references .....	2
<b>2</b>	<b>Hardware structure</b> .....	<b>3</b>
<b>3</b>	<b>Commissioning in TwinCAT3</b> .....	<b>4</b>
3.1	Configuration .....	4
3.2	Inserting connected devices .....	5
3.3	Configuration of the IO-Link master EL6224 .....	6
3.4	Importing the device description IODD .....	7
3.5	Creating a PLC program.....	9
3.6	Activating the configuration .....	10
<b>4</b>	<b>Access to IO-Link data</b> .....	<b>11</b>
4.1	Reading process data.....	11
4.2	Configure the process data format .....	12
4.3	Writing and reading parameters .....	13
4.4	Sample project: Reading/writing parameters.....	14
4.5	Blob transfer .....	16
4.6	Events.....	22
4.6.1	Reading out events using “Diag History” .....	22
4.6.2	Read out events using the “Detailed Device Status” parameter .....	23

# 1 About this manual

This guide contains procedures for the exemplary use of the WITTENSTEIN sensor cynapse®. This guide uses example code. If you require any code examples, please contact: [cybertronic-support@wittenstein.de](mailto:cybertronic-support@wittenstein.de)

The original was prepared in German, all other language versions are translations of the original instructions.

## 1.1 Information symbols and cross references

The following information symbols are used:

- Indicates an action to be performed
- ➔ Indicates the results of an action
- ⓘ Provides additional handling information

A cross reference refers to the chapter number and the header of the target section (e. g. chapter 2 “Hardware structure”).

A cross reference on a table refers to the table number (e. g. Table “Tbl-1”).

## 2 Hardware structure

The following hardware components were used for the sample project:

Control:	Beckhoff C6930
IO-LINK master:	Beckhoff EL6224
IO-LINK device:	WITTENSTEIN cynapse®

Figure 1 shows the schematic representation of the structure. The IO-Link master EL6224 is connected to a bus coupler EK1100, which in turn is connected to the control unit C6930 by means of EtherCAT (green). WITTENSTEIN cynapse® is connected to port 1 of the IO-Link master (black).

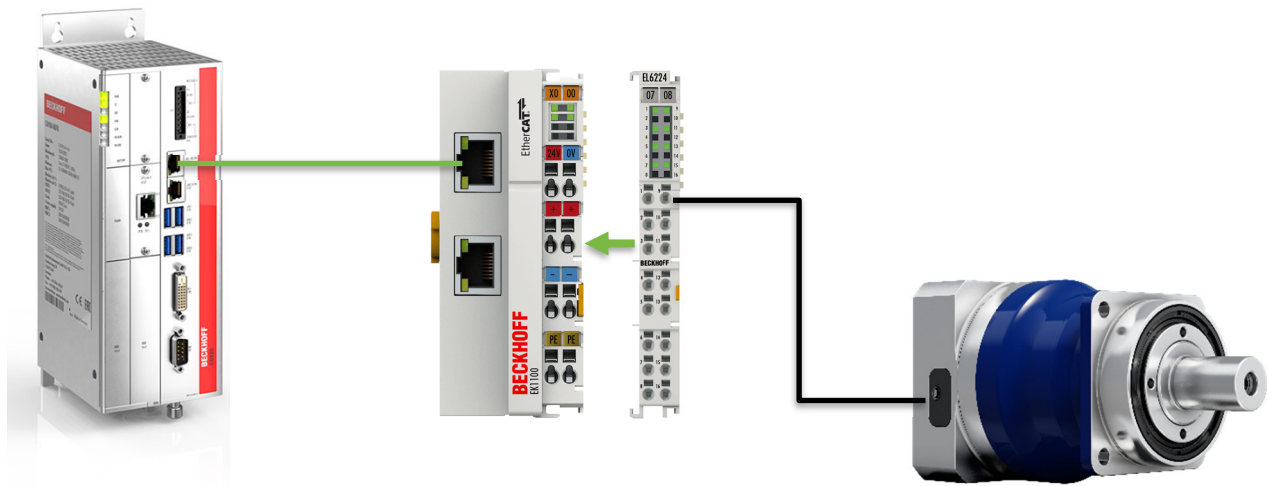


Figure 1: Schematic diagram

### 3 Commissioning in TwinCAT3

To commission cynapse® you need a new TwinCAT project. In addition, the following are required:

- A network port on the controller is configured as an EtherCAT port.
- TwinCAT development environment is installed.
- The hardware structure has been carried out according to Chapter 2.

#### 3.1 Configuration

Open the TwinCAT development environment (Visual Studio) and create a new TwinCAT project via “File” → “New” → “Project...”. The new project is visible in the project folder explorer. It is possible to use the TwinCAT “locally” on the C6390 or “remotely” from an engineering PC. In this implementation example, TwinCAT is used “locally”.

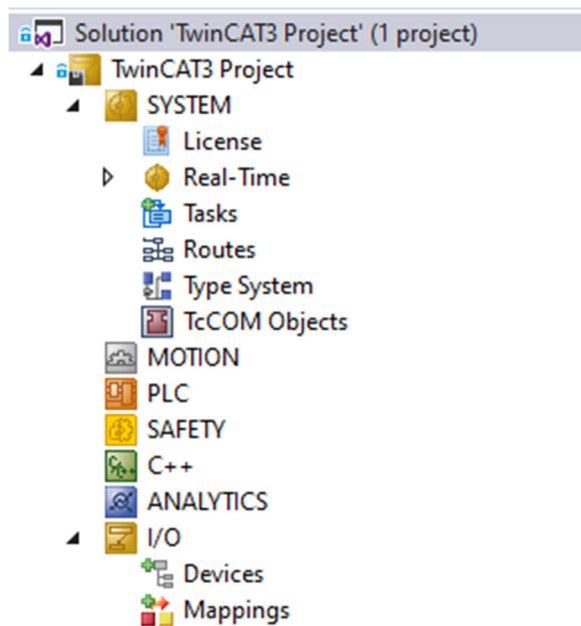




Figure 2: Project folder explorer

### 3.2 Inserting connected devices

In order to insert the connected devices, TwinCAT must be placed in “Conti Mode”, if it is not already in it. To do this, click on the symbol  or select “Restart TwinCAT (Config Mode)” from the “TWINCAT” menu →. Then, in the project folder explorer, you can select the “Devices” located within the “I/O” item and either right-click to open a context menu and select “Scan” or click on the icon  in the menu bar to start the action.

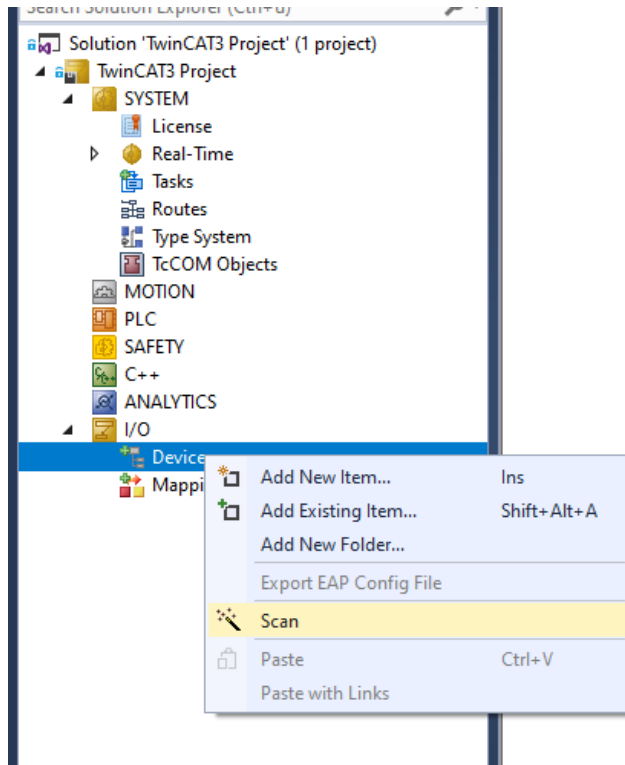


Figure 3: Select “Scan”

The following message must be confirmed and the “EtherCAT” devices must be selected in the dialog shown in Figure 4

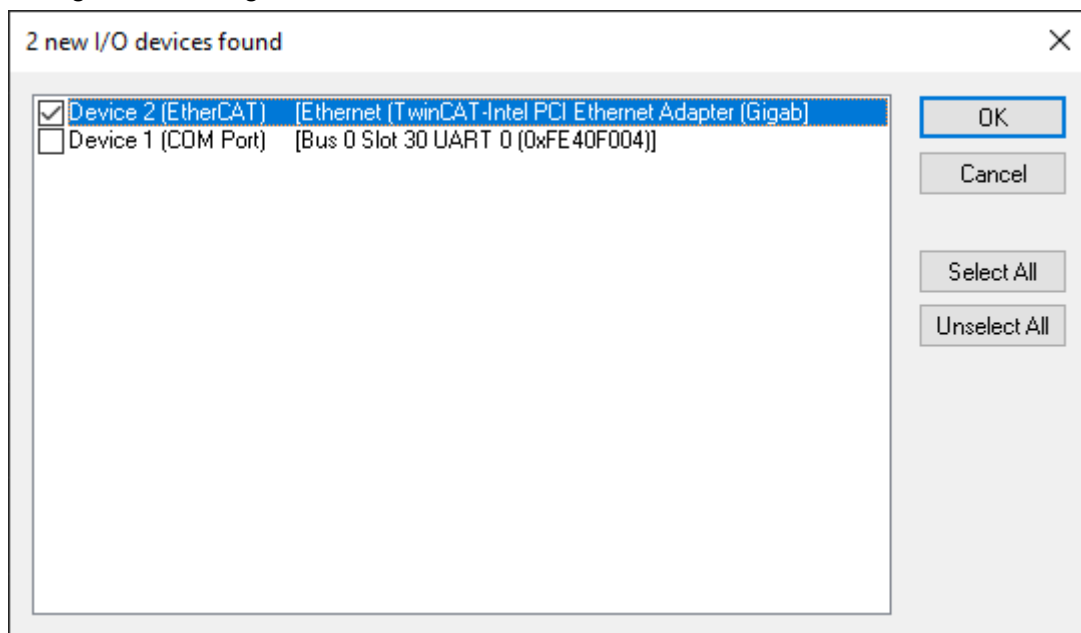


Figure 4: Selection of I/O devices to be integrated

The following message “Search for new boxes” must also be confirmed in order to search for the terminals connected to the devices.

Based on the configuration described here, the result is as follows:

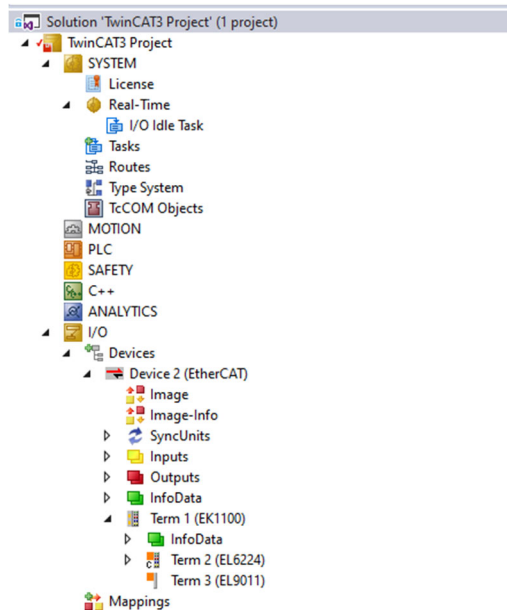


Figure 5: Configuring the TwinCAT 3 environment

- There is now an EtherCAT device in the “Devices” item (here: Device 2 (EtherCAT)). Again, the bus coupler EK1100 and the IO-Link master EL6224 are attached to this.

### 3.3 Configuration of the IO-Link master EL6224

Double-clicking on the EL6224 IO-Link master in the project folder explorer opens the terminal configuration menu. When creating the IO-Link master, an additional tab called “IO-Link” is created.

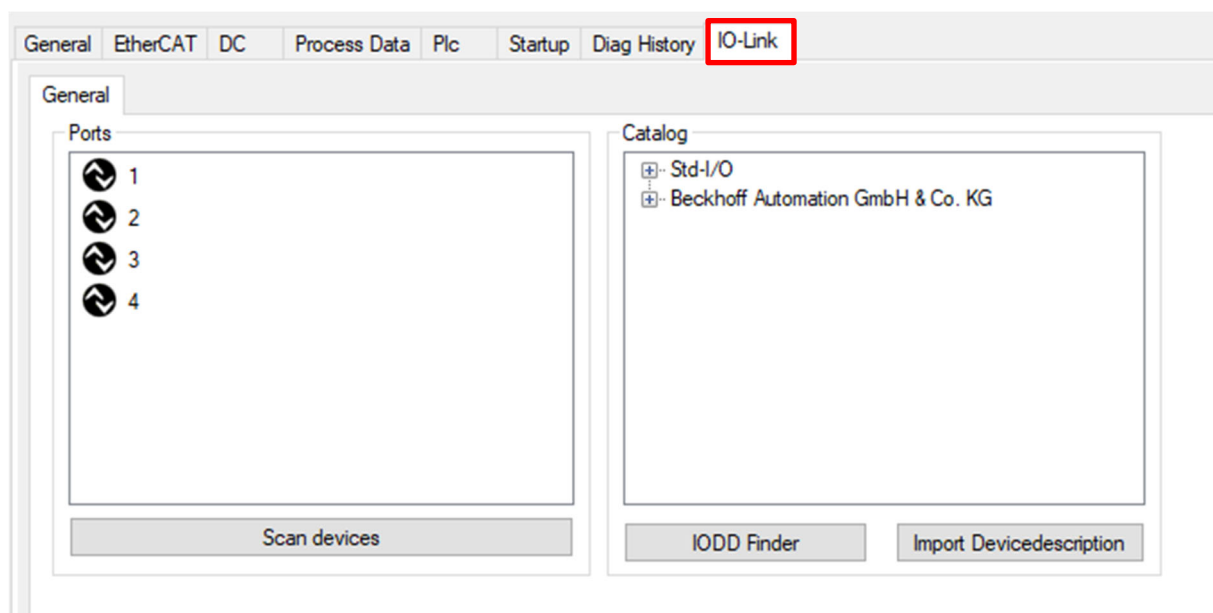


Figure 6: “IO-Link” tab

There are several possibilities for integrating the connected IO-Link device, here WITTENSTEIN cynapse®. At the beginning, the device description IODD is integrated.



### 3.4 Importing the device description IODD

Importing the device description simplifies the integration of the IO-Link device. With the help of the IODD (IO Device Description), the individual process data are broken down and a simple parameterization of the sensor is made possible. The IODD only has to be imported when a new IO-Link device is first commissioned. The import of the IODD is port-independent.

To insert the device description, there are the following two options:

#### Using the “Import DeviceDescription” function

This method requires that the required IODD is present. The current IODD of cynapse® can be obtained from:

➔ IODD Finder (<https://ioddfinder.io-link.com>)

When importing the IODD using the “Import DeviceDescription” button, you should do the following:

1. Press the “Import DeviceDescription” button in the “IO-Link” tab
2. Select and open the .xml file of the desired sensor
3. The imported files are stored in the \TwinCAT\3.X\Config\IO\IOLink folder. It is important not to copy the files directly into this folder, but to have them read in via “Import DeviceDescription”, otherwise important checks are bypassed!
4. Online configuration: If the IO-Link device is connected, it is created automatically with the corresponding parameters by clicking on “Scan devices”.
5. Offline configuration: After successful import, the IODD is displayed in the “Catalog” area. This can be assigned to a port by right-clicking on the corresponding device. Another way of assigning is to drag & drop the device from the catalog to the desired port.
6. Restart the EtherCAT system or reload the configuration in Config mode
7. The IO-Link devices are displayed and the process data is created.

## Using the IODD Finder

Another way to load the IODD is by clicking on the “IODD Finder” button. Clicking on it establishes a connection to the portal IODD Finder and displays all IO-Link devices stored there in a list. You can search for the desired sensor by filtering the list.

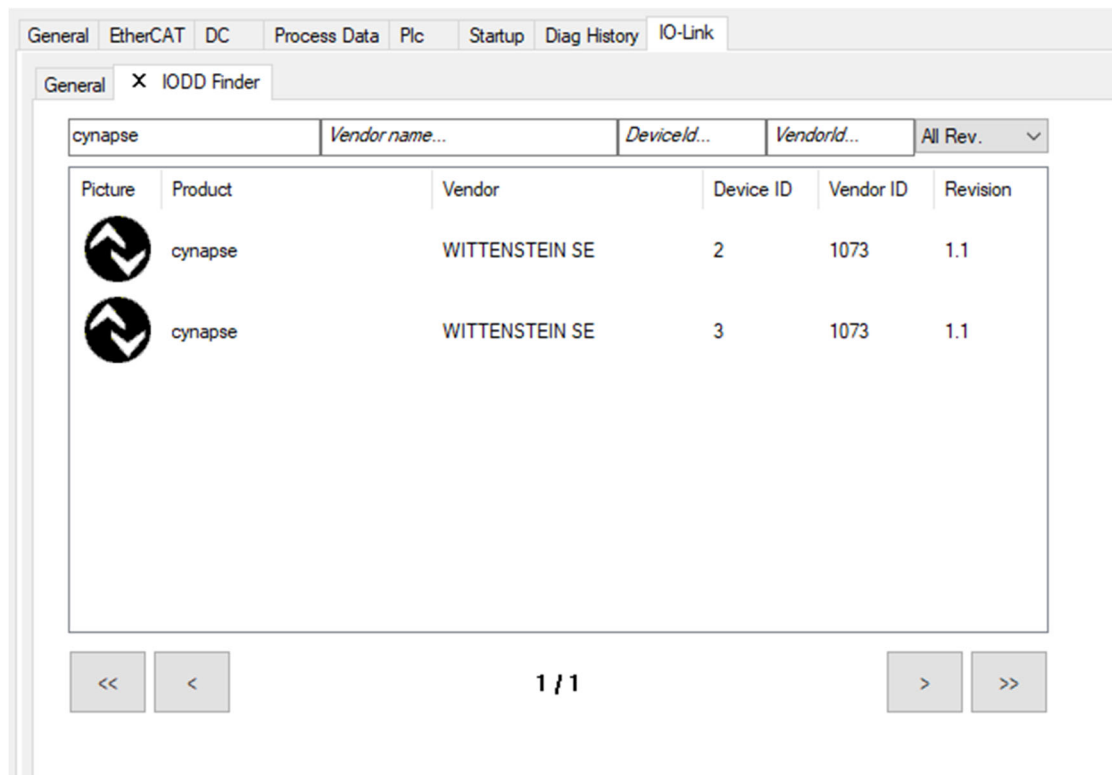



Figure 7: IODD Finder

Two entries for cynapse® are found. For new cynapse® sensors, please use the current version with the device ID “3”. Click on the  button to download the IODD. An internet connection is necessary for this. Subsequently, the IODD is in the catalog and can be assigned to a port as described in Chapter 3.4 from point 4.

### 3.5 Creating a PLC program

To create a programming environment, a new PLC subproject must be added from the context menu of “PLC” in the project folder explorer by selecting “Add New Item...”:

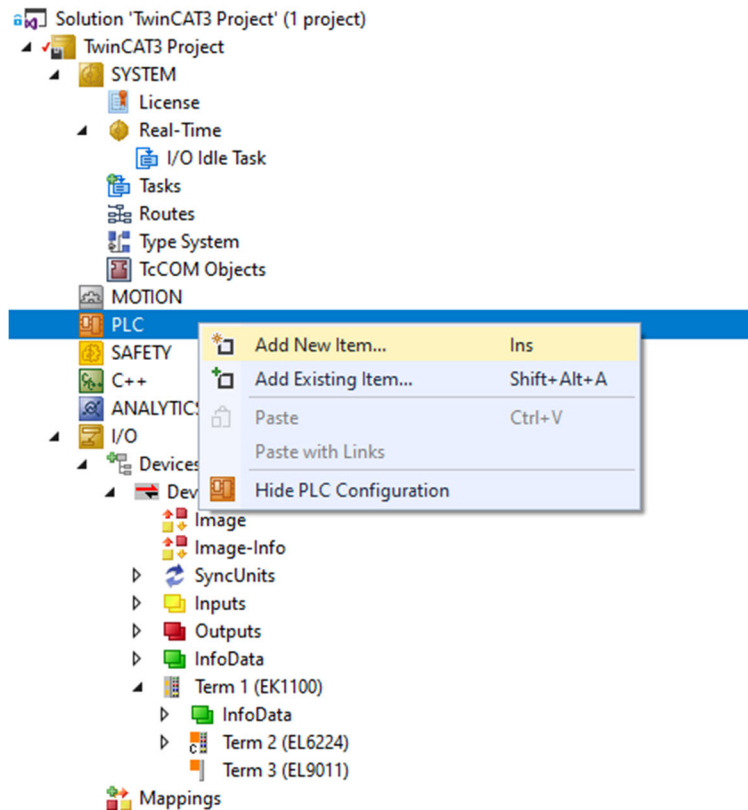


Figure 8: Creating a PLC program

In the subsequent dialog, a “Standard PLC Project” is selected and a project name (e.g. “cynapse\_example\_project”) is assigned. Selecting “Standard PLC Project” automatically creates the “Main” program. This can be opened by double-clicking on the PLC subproject “cynapse\_example\_project” in “POUs”. A global variable list (GVL) is also created. Here you have to create sample variables for the continued procedure, which can then be linked with the input variables of cynapse®:

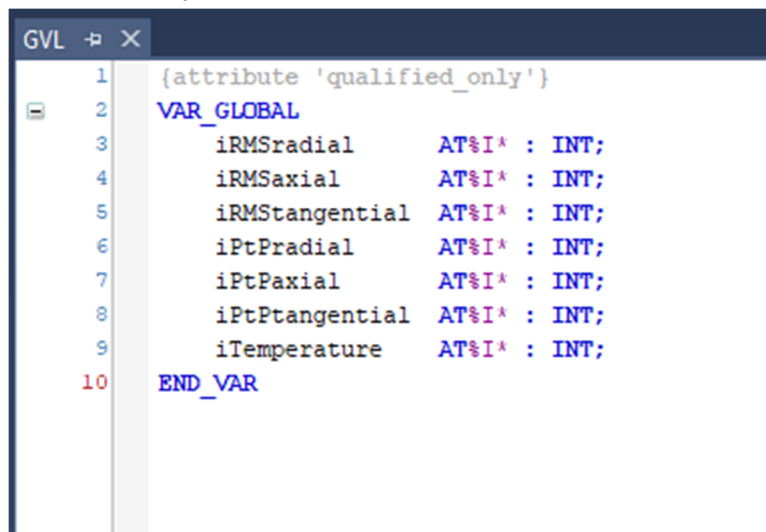
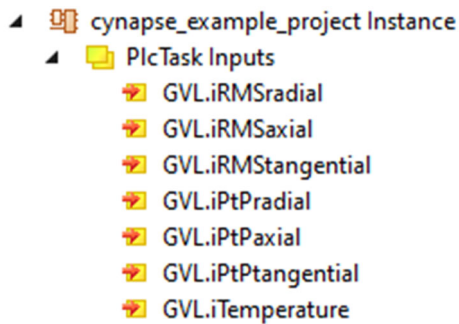


Figure 9: Creation of global variables

After the PLC project has been compiled via “Build”, the variables marked “AT%” are present in the “Assignments” and can be linked to the EtherCAT input variables.



Through right-clicking on the respective variable, a window for selecting its link is opened via “Change link...”.

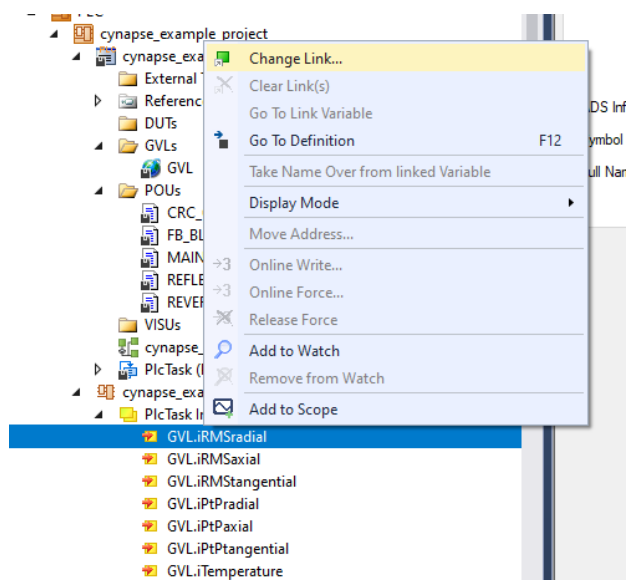



Figure 10: Create links between PLC variable and process objects

### 3.6 Activating the configuration

By activating the configuration via “TWINCAT” → “Activate configuration” or by clicking on the symbol , the TwinCAT control is set to run mode and the process data from cynapse® is interrogated cyclically.

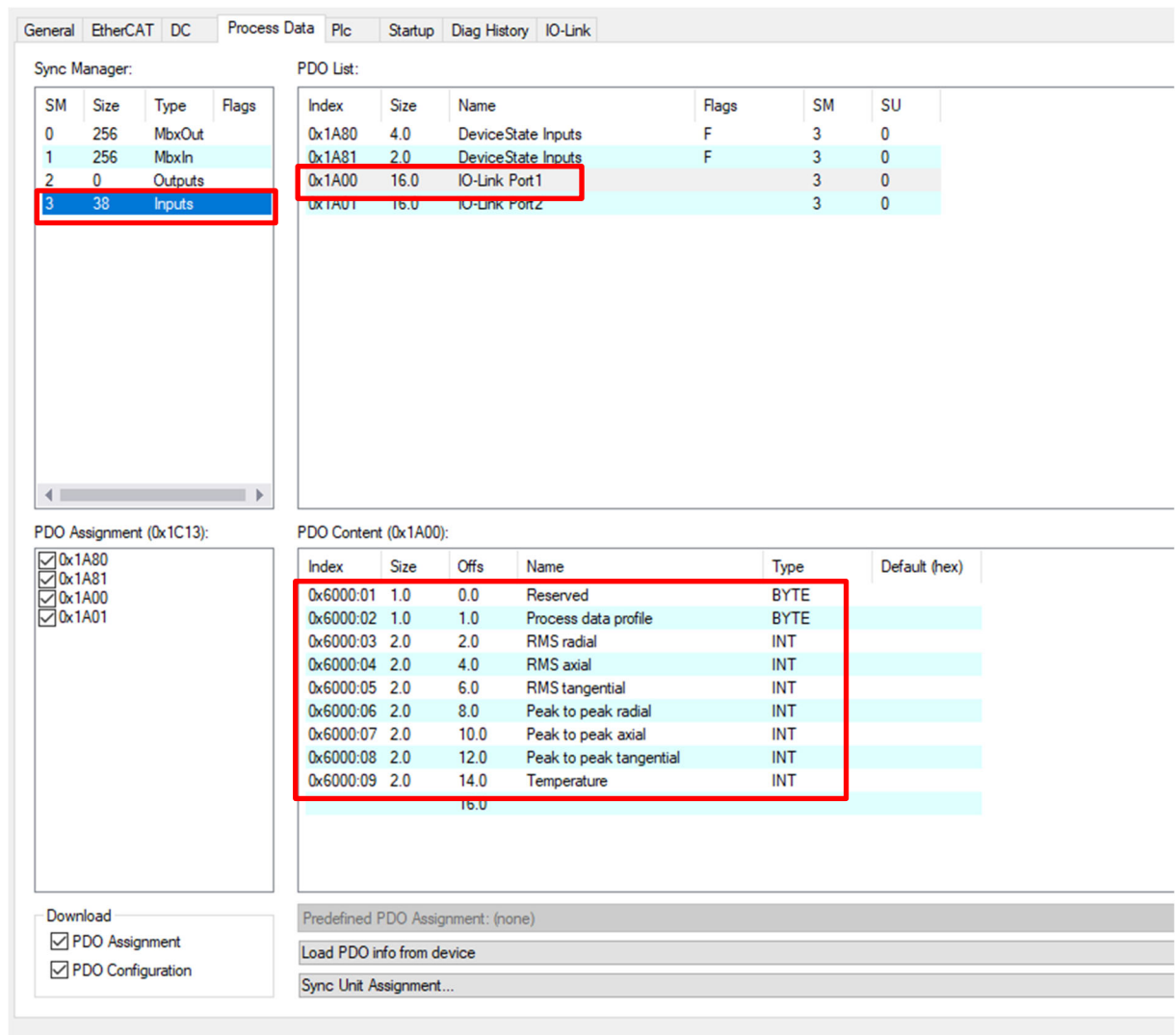
## 4 Access to IO-Link data

The Beckhoff IO-Link master terminal EL6224 is divided into two services. On the one hand, it provides an IO-Link master for the connected sensors, on the other hand, it is an EtherCAT slave of the higher-level TwinCAT master.

In principle, cyclic and acyclic data are exchanged between the IO-Link master and the IO-Link slave. From the perspective of the EtherCAT master, the cyclic process data can be accessed via the PDOs. The access to the acyclic data (blob, parameters and events) takes place via AoE.

### 4.1 Reading process data

After successful configuration of the respective IO-Link port, the process data is displayed in the “Process data” tab after a restart of the EtherCAT system. The process data are interrogated cyclically in the set system clock and can be linked to PLC variables for further use.



The screenshot shows the 'Process Data' configuration window with the following data:

**Sync Manager:**

SM	Size	Type	Flags
0	256	MbxOut	
1	256	MbxIn	
2	0	Outputs	
3	38	Inputs	

**PDO List:**

Index	Size	Name	Flags	SM	SU
0x1A80	4.0	DeviceState Inputs	F	3	0
0x1A81	2.0	DeviceState Inputs	F	3	0
0x1A00	16.0	IO-Link Port1		3	0
0x1A01	16.0	IO-Link Port2		3	0

**PDO Assignment (0x1C13):**

- 0x1A80
- 0x1A81
- 0x1A00
- 0x1A01

**PDO Content (0x1A00):**

Index	Size	Offs	Name	Type	Default (hex)
0x6000:01	1.0	0.0	Reserved	BYTE	
0x6000:02	1.0	1.0	Process data profile	BYTE	
0x6000:03	2.0	2.0	RMS radial	INT	
0x6000:04	2.0	4.0	RMS axial	INT	
0x6000:05	2.0	6.0	RMS tangential	INT	
0x6000:06	2.0	8.0	Peak to peak radial	INT	
0x6000:07	2.0	10.0	Peak to peak axial	INT	
0x6000:08	2.0	12.0	Peak to peak tangential	INT	
0x6000:09	2.0	14.0	Temperature	INT	

**Download:**

- PDO Assignment
- PDO Configuration

**Predefined PDO Assignment:** (none)

**Load PDO info from device:**

**Sync Unit Assignment...**

Figure 11: “Process data” tab

WITTENSTEIN cynapse® transmits process data on the current temperature, as well as various acceleration parameters. Various process data formats are provided in order to offer different characteristic data with a constant process data length of 16 bytes. The process data format is configured as described in Chapter 4.2.

cynapse® does not use any outgoing process data (from the point of view of the IO-Link master). For more information on the process data, please refer to the cynapse® operating manual.

## 4.2 Configure the process data format

cynapse® offers various process data formats. The process data length does not change. The process data format can be changed via the parameter “Settings” (index 0x60, subindex 0x09). The parameter can be changed via the PLC program (see chapter 4.4) or in the port configuration of the IO-Link master. This is done by right-clicking on the corresponding port in the “IO-Link” tab and then clicking on “Parameters”.

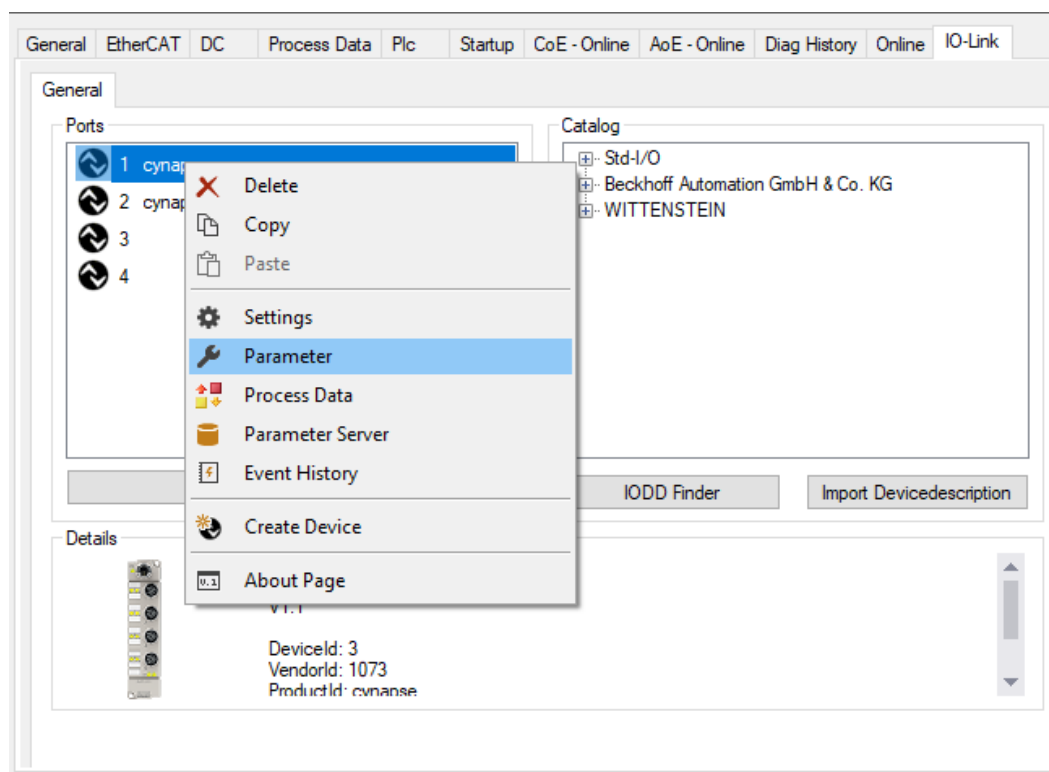


Figure 12: Open “Parameters” tab

There, all parameters of cynapse® can be read or written. When changing the process data format, note that the names of the process data are not changed. These are always the same regardless of the selected process data format. For more information on the available process data formats, see the cynapse® operating manual.

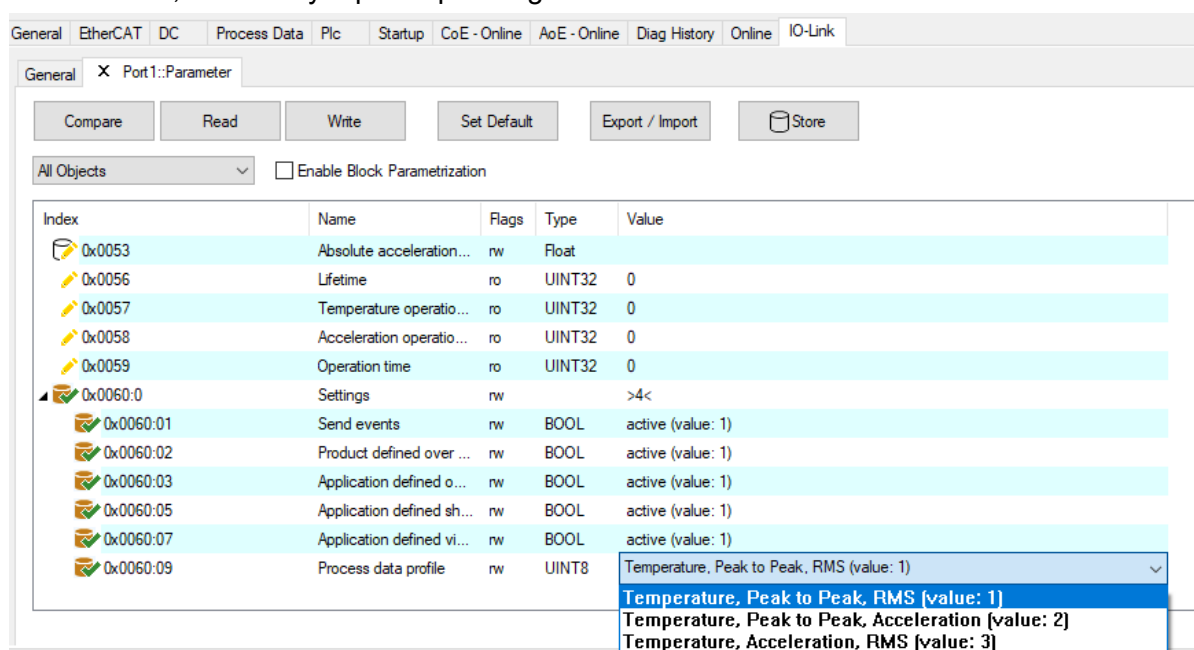


Figure 13: Changing the process data format

### 4.3 Writing and reading parameters

cynapse® supports parameterization by ISDU (Indexed Service Data Unit). These acyclic parameters must be explicitly requested or sent via the PLC. Access is via ADS or CoE. The following describes how to read and write parameters using ADS.

An ADS address always consists of NetID and port number. An ADS command is sent from TwinCAT via AoE (ADS over EtherCAT) to the IO-Link master AL1330 and from there forwarded to the demand data channel.

#### AoE NetID

The EtherCAT slave AL1330 has its own NetID for communication. This can be looked up in the terminal configuration in the “EtherCAT” tab under “Advanced settings” -> “Mailbox” -> “AoE”.

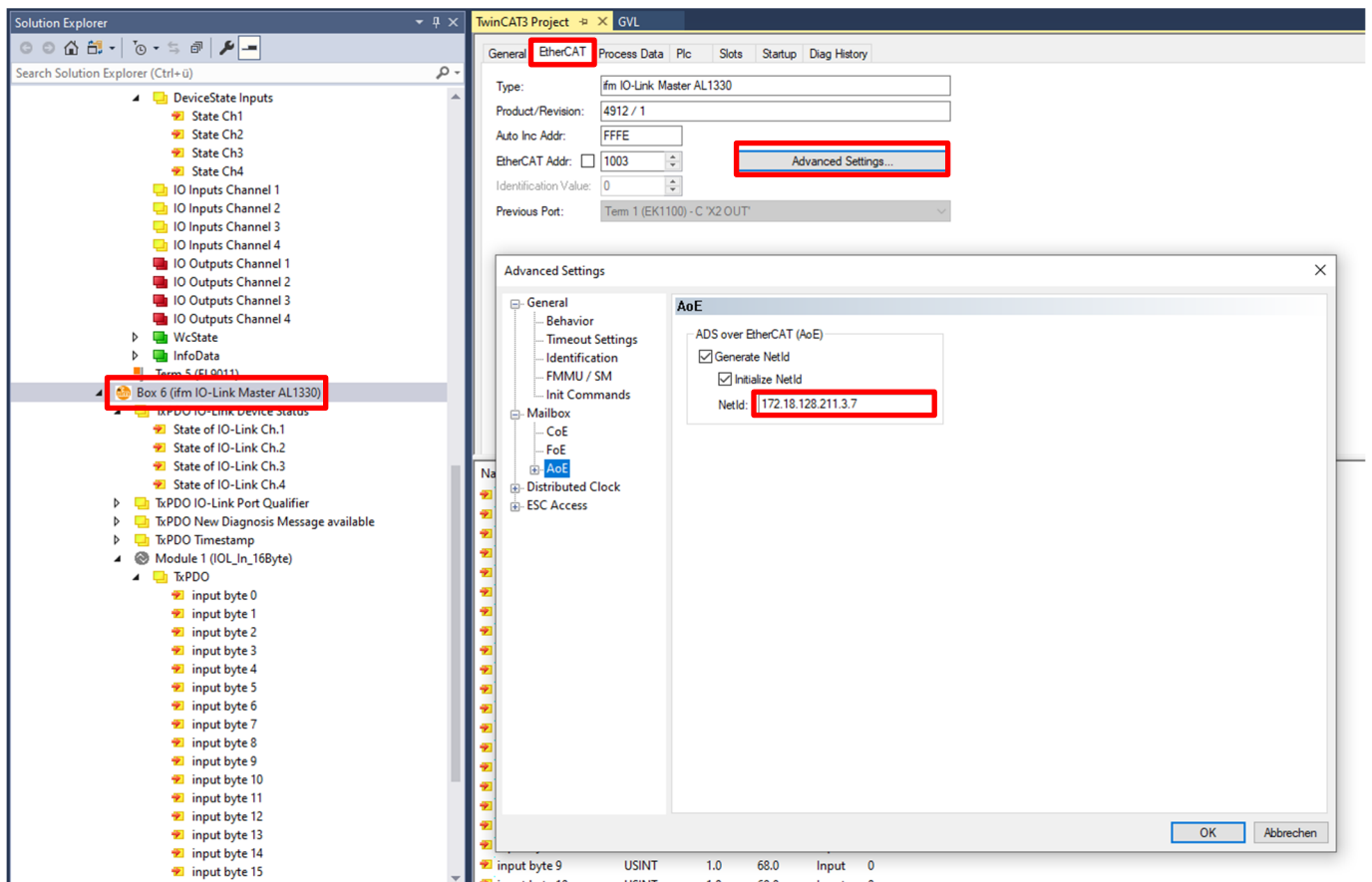


Figure 14: ADS address

#### Port number

The assignment of the query to a single IO-Link port takes place via the port number. The port number is assigned ascending from 0x1000. IO-Link port 1 thus corresponds to 0x1000 and IO-Link port n corresponds to port number 0x1000+n-1.

For the terminal EL6224 with 4 IO-Link ports used here, the following definition applies

- IO-Link Port 1 → port number 0x1000
- IO-Link Port 2 → port number 0x1001
- IO-Link Port 3 → port number 0x1002
- IO-Link Port 4 → port number 0x1003

## ADS index group

The index group of an ADS command is set to 0xF302 for the IO-Link demand data channel.

## ADS index offset

In index offset, the IO-Link addressing is encoded with index and subindex.

The index offset is 4-byte in size and is divided as follows:

- Bit 16-31: Index
- Bit 8-15: reserved
- Bit 0-7: Subindex

Example: Index 0x005D and subindex 0x02 equals index offset 0x005D0002

The available acyclic parameters can be read in the cynapse® operating manual.

## 4.4 Sample project: Reading/writing parameters

The following describes how parameters can be read or written via AoE using the function modules “ADSRead” and “ADSWrite”.

The two modules “ADSRead” and “ADSWrite” are part of the Beckhoff own library “TC2\_Standard”. By default, the library is loaded when a PLC project is rebuilt.

The following example demonstrates how to write and read the “Operating temperature threshold” parameter using the “ADSREAD” and “ADSWRITE” function blocks. First, a new threshold value is written with the instance “fbIOLWrite” of the “ADSWrite” module and then read for verification with the instance “fbIOLRead”.

- ① The complete sample project is available on request at:  
[cybertronic-support@wittenstein.de](mailto:cybertronic-support@wittenstein.de)

### Declaration part:

```

PROGRAM MAIN
VAR
    fbIOLRead      : ADSRead;
    fbIOLWrite     : ADSWrite;

    iState         : INT;
    bExecute       : BOOL;

    rTemperatureThresholdWrite : REAL := 40; //°C
    rTemperatureThresholdRead  : REAL;

    bBusy          : BOOL;
    bError         : BOOL;
    nErrID         : UDINT;
END_VAR

```



**Implementation:**

```

CASE iState OF
0 : IF bExecute THEN
    bBusy := TRUE;
    bError := FALSE;
    nErrID := 0;

    // Write new Process Data Profile to cynapse®

    fbIOLWrite(
        NETID:='172.18.128.211.3.3', //AoE-NetID EL6224
        PORT:= 16#1000, //PortNo IO-Link Port
        IDXGRP:= 16#F302, //Defined by Beckhoff
        IDXOFFS:= 16#00520000, //Index = 0x0052 and Subindex = 0x00
        LEN:= SIZEOF(rTemperatureThresholdWrite),
        SRCADDR:= ADR(rTemperatureThresholdWrite),
        WRITE := TRUE );

    iState := 1;
END_IF

1 : fbIOLWrite( WRITE := FALSE, BUSY=>bBusy, ERR=>bError, ERRID=>nErrID );

    IF NOT bBusy THEN
        IF NOT bError THEN
            iState := 2; //Success
        ELSE
            iState := 100; //Error
        END_IF
    END_IF

2 : //Read Process Data Profile
    fbIOLRead(
        NETID:='172.18.128.211.3.3', //AoE-NetID EL6224
        PORT:= 16#1000, //PortNo IO-Link Port
        IDXGRP:= 16#F302, //Defined by Beckhoff
        IDXOFFS:= 16#00520000, //Index = 0x0052 and Subindex = 0x00
        LEN:= SIZEOF(rTemperatureThresholdRead),
        DESTADDR:= ADR(rTemperatureThresholdRead),
        READ := TRUE );

    iState := 3;

3 : fbIOLRead( READ := FALSE, BUSY=>bBusy, ERR=>bError, ERRID=>nErrID );

    IF NOT bBusy THEN
        IF NOT bError THEN
            iState := 4; //Success
        ELSE
            iState := 100; //Error
        END_IF
    END_IF

4 : //Compare Read and Write Value
    IF rTemperatureThresholdRead = rTemperatureThresholdWrite THEN
        iState := 0; //Success
    ELSE
        iState := 100; //Error
    END_IF

```

```
bExecute := FALSE;

100 : //Implement Error Handler here

END_CASE
```

### 4.5 Blob transfer

IO-Link defines the exchange of larger amounts of data by the BLOB (Binary Large Object) Transfer Profile. cynapse® uses this to send collected data. The transfer is on request. Different data packages are provided by cynapse®, these can be found in the cynapse® operating manual.

The transmission of the acceleration data package is described in more detail below:

In addition to the raw data of the acceleration sensor, the data package contains the status of the four operating time counters and the temperature at the time of the measurement recording.

At the beginning, the data package must first be requested via the command “Request acceleration data package”. For this purpose, the value 0xA8 is sent via the IO-Link Index 0x02 by means of the “ADSWRITE”. Once this has been done, the data package can then be read out via blob with ID -4097. The sequence of the blob transfer is shown schematically in Figure 15.

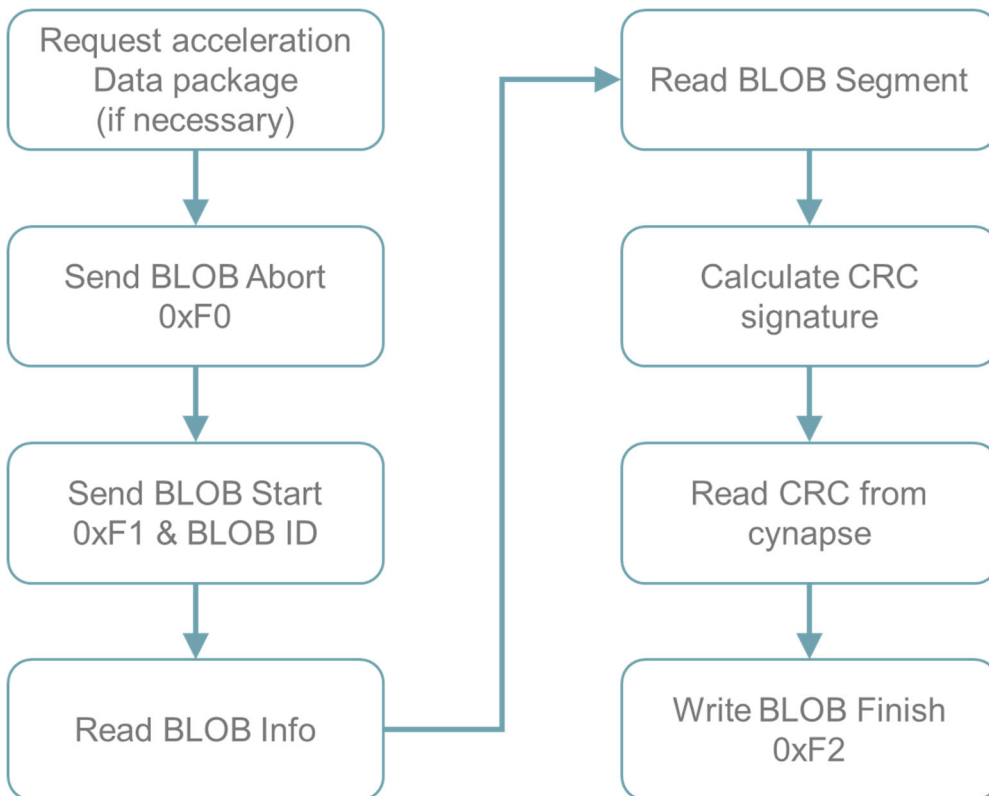


Figure 15: BLOB transfer flow

After requesting the acceleration data package, the command “BLOB Abort” is written. The communication always runs during the BLOB transfer via the IO-Link Index 0x32 “BLOB Channel”. The required transmission objects are listed in Figure 12. For more information, see the “IO-Link Profile BLBs & FW-Update Specification”. The “BLOB Abort” command terminates all active BLOB transfers and resets the sensor’s “BLOB state machine”. The command is not required and is used to avoid errors.

The “BLOB start” object is then sent together with the “BLOB ID”. The “BLOB IDs” supported by cynapse® can be found in the operating manual. cynapse® then returns the length of the blob to be read as “BLOB info”. The actual measurement data are stored in segments with a maximum length of 232 bytes, the first byte containing no measurement data, but the number of transmitted segments.

During segment transfer, a cyclic redundancy check (CRC: cyclic redundancy check) is performed and then matched to the CRC value from cynapse®. This allows a data loss during transmission to be detected. The BLOB transfer is finished with the command “BLOB finish”.

The following example program shows the sequence of the blob transfer in structured text (ST).

- ① The project with the two modules FB\_BLOB\_Read and CRC\_Gen (calculation of the CRC value) can be obtained on request at [cybertronic-support@wittenstein.de](mailto:cybertronic-support@wittenstein.de)

The program provides a buffer “ptBlobData” of 600000 bytes, which is the maximum length of a BLOB of cynapse®. Depending on the BLOB ID selected, this buffer can be made smaller. See the cynapse® operating manual.

### Declaration part

```

FUNCTION_BLOCK FB_BLOB_Read
VAR_INPUT
    IOLMasterNetID : T_AmsNetID;           // NetID IO-Link master
    uiIOLPort      : UINT;                 // IO-Link Port
    bExecute       : BOOL;

    ptBlobData : POINTER TO ARRAY[0..600000] OF BYTE;
END_VAR

VAR_OUTPUT
    bError : BOOL;
    bBusy  : BOOL;
    bDone  : BOOL;
    ErrID  : UDINT;
    RD_length : UDINT;
END_VAR

VAR
    BlobData : ARRAY[0..600000] OF BYTE;   //Blob Data
    iStep    : INT;

    rTrigExecute : R_TRIG;
    fbTrigDataPackage : ADSWRITE; //FB Request Acceleration Data Package
    DataPackage : BYTE := 16#A8; //Request Acceleration Data Package
    tWaitTimer : TON;

```

```

fbBlobWrite : ADSWRITE;
fbBLOBRead : ADSREADEX;

BlobAbort : BYTE := 16#F0;
BlobFinish : BYTE := 16#F2;
BlobStart : ARRAY[0..2] OF BYTE;
BlobRead : ARRAY[0..255] OF BYTE;
BlobCRC : ARRAY[0..4] OF BYTE;

l_Blob_ID : INT := -4097; //ID for BLOB Transfer

ptrBlobBuffer : POINTER TO BYTE;
udiBlobLength : UDINT;
i : INT;
iSgmtCnt : INT;
udiCnt: UDINT;
dwBlobLength : DWORD;
CRC : DWORD;
dwBlobCRC : DWORD;

END_VAR

```

### Implementation:

```

//Function Block ADSwrite: Request Acceleration Data Package 0xA8
fbTrigDataPackage(
    NETID:= IOLMasterNetID,
    PORT:= uiIOLPort,
    IDXGRP:= 16#F302, //Indexgrp of ADS command is specified as 0xF302 for the IO
link data channel
    IDXOFFS:= 16#020000, //IOLink-Index 0x02 Subindex 0x0 (System Command)
    LEN:= SIZEOF(DataPackage),
    SRCADDR:= ADR(DataPackage) );

//Function Block ADSwrite: Write Blobchannel
fbBlobWrite(
    NETID:= IOLMasterNetID,
    PORT:= uiIOLPort,
    IDXGRP:= 16#F302, //Defined by Beckhoff
    IDXOFFS := 16#320000 //IOLink-Index 0x32 Subindex 0x0 (Blob Channel)
);

//Function Block ADSRead: Read BlobChannel
fbBlobRead(
    NETID:= IOLMasterNetID,
    PORT:= uiIOLPort,
    IDXGRP:= 16#F302, //Defined by Beckhoff
    IDXOFFS := 16#320000 //IOLink-Index 0x32 Subindex 0x0 (Blob Channel)
);

//Error Handling
IF fbTrigDataPackage.ERR THEN
    iStep := 1000;
    ErrID := fbTrigDataPackage.ERRID;
END_IF

IF fbBlobRead.ERR THEN
    iStep := 1000;
    ErrID := fbBlobRead.ERRID;
END_IF

IF fbBlobWrite.ERR THEN
    iStep := 1000;

```

```

    ErrID := fbBlobWrite.ERRID;
  END_IF

  //*****
  // BLOB-Data Command
  //*****
  //Wait for rising Edge bExecute
  rTrigExecute(CLK:= bExecute);

  CASE iStep OF
  0 : //Wait for bExecute (rising Edge)
    IF rTrigExecute.Q THEN
      bBusy := TRUE;
      bError := FALSE;
      fbTrigDataPackage.WRITE := TRUE;
      iStep := 10;
    END_IF

    bDone := FALSE;

  10: //Set Trigger to request Acceleration Data Package
    IF NOT fbTrigDataPackage.BUSY THEN
      fbTrigDataPackage.WRITE := FALSE;
      tWaitTimer.IN := TRUE;
      iStep := 20;
    END_IF

  20: // Wait for 1s till Data Package is generated
    tWaitTimer(PT:= T#1000MS);
    IF tWaitTimer.Q THEN
      tWaitTimer.IN := FALSE;
      //Write BlobAbort to close transmission channel
      fbBlobWrite.LEN := SIZEOF(BlobAbort);
      fbBlobWrite.SRCADDR := ADR(BlobAbort);
      fbBlobWrite.WRITE := TRUE;
      iStep := 30;
    END_IF

  30: IF NOT fbBlobWrite.BUSY THEN
      fbBlobWrite.WRITE := FALSE;
      iStep := 40;
    END_IF

  40: //Write BlobStart to establish transmission channel for Blob ID -4097
    BlobStart[0] := 16#F1; //BLOBStart
    BlobStart[1] := INT_TO_BYTE(SHR(1_Blob_ID,8));
    BlobStart[2] := INT_TO_BYTE(1_Blob_ID);
    fbBlobWrite.LEN := SIZEOF(BlobStart);
    fbBlobWrite.SRCADDR := ADR(BlobStart);
    fbBlobWrite.WRITE := TRUE;
    iStep := 50;

  50: IF NOT fbBlobWrite.BUSY THEN
      fbBlobWrite.WRITE := FALSE;

      //Read Blob Info
      fbBlobRead.LEN := SIZEOF(BlobRead);
      fbBlobRead.DESTADDR := ADR(BlobRead);
      fbBlobRead.READ := TRUE;
      iStep := 60;
    END_IF
  
```

```

60: IF NOT fbBlobRead.BUSY THEN
    fbBlobRead.READ := FALSE;

    //The Blob Read Info contains the BLOB Length in 4 Octets (32Bit)
    dwBlobLength := SHL(BlobRead[1], 24) OR SHL(BlobRead[2], 16) OR
        SHL(BlobRead[3], 8) OR BlobRead[4];

        //Set BlobLength Counter to ZERO
        udiBlobLength := 0;
        //Set Segment Counter to ZERO
        iSgmtCnt := 0;
        //Initialize Pointer to BlobData
        ptrBlobBuffer := ADR(BlobData);
        iStep := 70;
    END_IF

70: //Read Blob Segment
    fbBlobRead.READ := TRUE;
    iStep := 80;

80: IF NOT fbBlobRead.BUSY THEN
    fbBlobRead.READ := FALSE;

    //Write BlobSegment to Array BlobData
    FOR udiCnt := 1 TO fbBlobRead.COUNT_R - 1 DO
        ptrBlobBuffer := ADR(BlobData)+udiBlobLength+SIZEOF(BYTE)*(udiCnt - 1);
        ptrBlobBuffer^ := BlobRead[udiCnt];
    END_FOR

    //Calculate overall length of BLOB Data
    udiBlobLength := udiBlobLength + (fbBlobRead.COUNT_R - 1);

    iStep := 81;
    END_IF

81: //CRC Check
    IF iSgmtCnt = 0 AND BlobRead[0] <> 16#30 THEN
        //calculate CRC signature by Function CRC_GEN for Initial Segment
        CRC := CRC_GEN(PT:=ADR(BlobRead[1]),
            SIZE:=UDINT_TO_INT(fbBlobRead.COUNT_R-1), PL := 32,
            PN := 16#741B8CD7, INIT:=16#7FFFFFFF, REV_IN:= TRUE,
            REV_OUT:= FALSE, XOR_OUT:= 16#0);
        iStep := 70;
    ELSIF BlobRead[0] = 16#30 AND iSgmtCnt > 0 THEN
        //calculate CRC signature by Function CRC_GEN for Last Segment
        CRC := CRC_GEN(PT:=ADR(BlobRead[1]),
            SIZE:=UDINT_TO_INT(fbBlobRead.COUNT_R-1), PL := 32,
            PN := 16#741B8CD7, INIT:=CRC, REV_IN:= TRUE, REV_OUT:= TRUE,
            XOR_OUT:= 16#FFFFFFF );
        iStep := 85;
    ELSIF BlobRead[0] = 16#30 AND iSgmtCnt = 0 THEN
        //calculate CRC signature by Function CRC_GEN
        CRC := CRC_GEN(PT:=ADR(BlobRead[1]),
            SIZE:=UDINT_TO_INT(fbBlobRead.COUNT_R-1), PL := 32,
            PN := 16#741B8CD7, INIT:=16#7FFFFFFF, REV_IN:= TRUE,
            REV_OUT:= TRUE, XOR_OUT:= 16#FFFFFFF );
        iStep := 85;
    ELSE
        //calculate CRC signature by Function CRC_GEN
        CRC := CRC_GEN(PT:=ADR(BlobRead[1]),

```

```

        SIZE:=UDINT_TO_INT(fbBlobRead.COUNT_R-1), PL := 32,
        PN := 16#741B8CD7, INIT:=CRC, REV_IN:= TRUE, REV_OUT:= FALSE,
        XOR_OUT:= 16#0 );
    iStep := 70;
END_IF

//increment Segment Counter
iSgmtCnt := iSgmtCnt + 1;

85: //CRC CHECK
//calculate CRC signature by Function CRC_GEN
// Read CRC signature from cynapse® 0x40
fbBlobRead.LEN := SIZEOF(BlobCRC);
fbBlobRead.DESTADDR := ADR(BlobCRC);
fbBlobRead.READ := TRUE;
iStep := 86;

86: IF NOT fbBlobRead.BUSY THEN
    fbBlobRead.READ := FALSE;

    // The Blob CRC contains the BLOB signature in 4 Octets (32Bit)
    dwBlobCRC := SHL(BlobCRC[1], 24) OR SHL(BlobCRC[2], 16) OR
                SHL(BlobCRC[3], 8) OR BlobCRC[4];

    IF CRC = dwBlobCRC THEN
        iStep := 90;
    ELSE
        // if not CRC = dwBlobCRC then generate Error Message
        iStep := 1000;
    END_IF
END_IF

90: //Write BLOB Finish 0xF2
fbBlobWrite.LEN := SIZEOF(BlobFinish);
fbBlobWrite.SRCADDR := ADR(BlobFinish);
fbBlobWrite.WRITE := TRUE;
iStep := 100;

100:IF NOT fbBlobWrite.BUSY THEN
    fbBlobWrite.WRITE := FALSE;
    bBusy := FALSE;
    bDone := TRUE;
    iStep := 0;
    MEMCPY(ptBlobData,ADR(BlobData),udiBlobLength);
    RD_Length := udiBlobLength;
END_IF

1000://Errorhandling
bError := TRUE;
iStep := 0;
END_CASE

```

## 4.6 Events

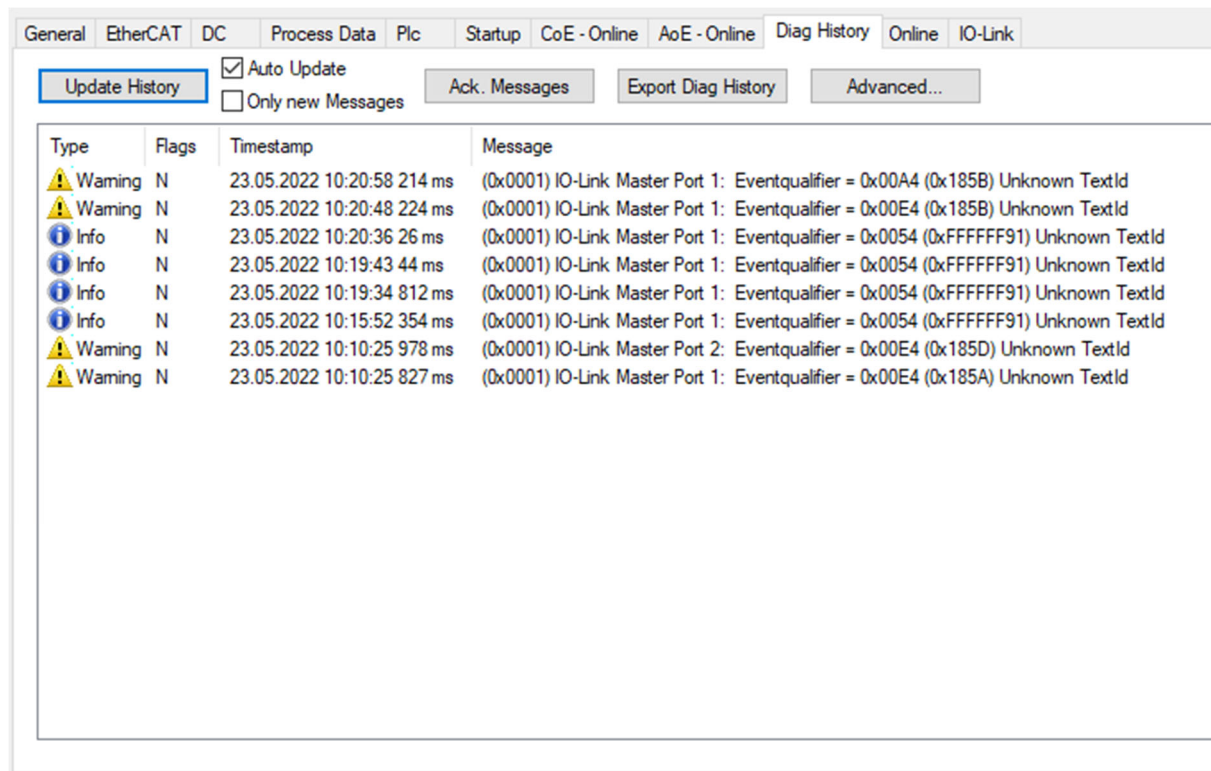
cynapse® supplies so-called IO-Link events for selected operating conditions, for example when vibration or temperature threshold values are exceeded. This can be evaluated by the higher-level control system.

In IO-Link there are 3 different types of events (Error, Warning, Information). Error and warning events always have a start (Appear) and an end (Disappear). Thus, event types are two time-shifted events that are sent by the IO-Link device. Information events are so-called singleshoot events. There is only one event here.

- ① The events supported by cynapse® are listed in the cynapse® operating manual.
- ① In order to send events, these must be activated in cynapse®. This release is made via index 0x60. A general event release (subindex 0x01) is necessary and a parameter dependent release (subindex 0x02 - 0x07) is possible.

### 4.6.1 Reading out events using “Diag History”

cynapse® forwards events to the IO-Link master. This signals this by setting the status bit “Device Diag”. Further information about the events can be found in the Diag History tab.



Type	Flags	Timestamp	Message
Warning	N	23.05.2022 10:20:58 214 ms	(0x0001) IO-Link Master Port 1: Eventqualifier = 0x00A4 (0x185B) Unknown TextId
Warning	N	23.05.2022 10:20:48 224 ms	(0x0001) IO-Link Master Port 1: Eventqualifier = 0x00E4 (0x185B) Unknown TextId
Info	N	23.05.2022 10:20:36 26 ms	(0x0001) IO-Link Master Port 1: Eventqualifier = 0x0054 (0xFFFFFFFF91) Unknown TextId
Info	N	23.05.2022 10:19:43 44 ms	(0x0001) IO-Link Master Port 1: Eventqualifier = 0x0054 (0xFFFFFFFF91) Unknown TextId
Info	N	23.05.2022 10:19:34 812 ms	(0x0001) IO-Link Master Port 1: Eventqualifier = 0x0054 (0xFFFFFFFF91) Unknown TextId
Info	N	23.05.2022 10:15:52 354 ms	(0x0001) IO-Link Master Port 1: Eventqualifier = 0x0054 (0xFFFFFFFF91) Unknown TextId
Warning	N	23.05.2022 10:10:25 978 ms	(0x0001) IO-Link Master Port 2: Eventqualifier = 0x00E4 (0x185D) Unknown TextId
Warning	N	23.05.2022 10:10:25 827 ms	(0x0001) IO-Link Master Port 1: Eventqualifier = 0x00E4 (0x185A) Unknown TextId

Figure 16: Diag History tab

The events that occur are determined by type (information, warning, error), flag, occurrence of the event (timestamp) and message (port number & event code) (see Figure 16). The IO-Link device can be uniquely assigned based on the port number.



#### 4.6.2 Read out events using the “Detailed Device Status” parameter

Events of the type Error or Warning can additionally be read out using the index 0x25 “Detailed Device Status”. The parameter contains only events that occur (Appear). The parameter consists of a series of data packages, each with a length of 3 bytes.

cynapse® provides a list of 11 entries. If the values are NULL, no event is active. The first empty entry can be canceled because the active events are at the top of the list.

Each 3 byte entry is divided into Event Qualifier (byte 1) and Event Code (bytes 2 and 3). The interpretation of the event codes can be found in the cynapse® operating manual.

#### Example

The cyclic query of the parameter “Detailed Device Status” index 0x25 yields the following result for the first 9 bytes:

➡ 0xE4185AE4185D000000

If you divide the answer into packages with a size of 3 bytes, you get the following result

➡ 0xE4185A 0xE4185D 0x000000

There are two events. The third data package is empty and does not supply an entry, so the search for events can be aborted here. The first two packages contain upcoming events. The first byte provides information about the EventQualifier. In both cases, this is 0xE4 and means that an occurring event (Appear) of the “Warning” type was sent by the device cynapse®.

① A detailed description of the EventQualifier can be found in the IO-Link specification.

The two subsequent bytes contain the event code described in the cynapse® operating manual.

➡ 0x185A → The user’s upper temperature threshold has been exceeded  
➡ 0x185D → The user’s vibration threshold has been exceeded



## Revision history

Revision	Date	Comment	Chapter
01	12/02/2019	New version	All
02	08/16/2022	cynapse® Trademark, Revision	All



WITTENSTEIN alpha GmbH · Walter-Wittenstein-Straße 1 · 97999 Igersheim · Germany  
Tel. +49 7931 493-0 · [info@wittenstein.de](mailto:info@wittenstein.de)

**WITTENSTEIN – one with the future**

**[www.wittenstein-alpha.de](http://www.wittenstein-alpha.de)**