CANopen
User’s Manual for
CLV6xx, RFH6xx, ICR6xx

Date	Version	Author	Changes
2009 12 01	1.0	Aschenbrenner	initial revision
2009 12 06	1.1	Aschenbrenner	minor corrections
2009 12 08	1.2	Aschenbrenner	corrections Jerry Finley included
2010 06 08	1.3	Adler	Additional infos for Result Output via PDO
2011 01 27	1.4	Thomas	Inclusion for ID^{pro} devices RFH6xx and Lector
2011 11 30	1.5	Kluemper	Correction of subindex at CAN inputs and outputs

Related Documents:

CiA Draft Standard 301
CANopen application layer and communication profile
(http://www.can-cia.de/)

Contents

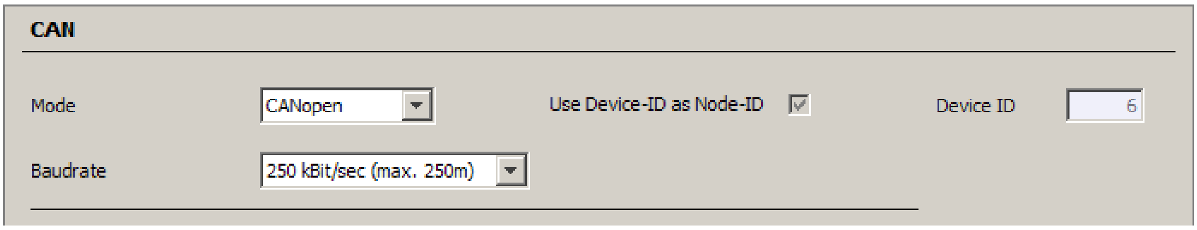
1 Overview CAN networking with SICK ID^{pro} devices

Each SICK ID^{pro} device has a CAN interface. It can be used in two different operation modes: SICK-Network or CANopen.

Using SICK-Network, the CAN interface is initialized for interconnection to different SICK AutoIdent products. This networking is also based on the CANopen protocol, but it is not intended to have other products besides AutoIdent sensors and controllers in the network.

It is not the subject of this document to describe the functions of the CAN SICK-Network.

If you select CANopen for The CAN Mode, the SICK ID^{pro} device will behave as a CANopen slave device which may work in any CANopen Network together with any other CANopen device.



If you want to run several devices in a CANopen network you must ensure that each device uses the same baudrate and that each device has its specific node ID.

For SICK ID^{pro} devices we use the device ID also as its CANopen Node ID.

Note: If you are using an external CMC600 parameter cloning module mounted in the CLVs Connection box, you can select the node ID by the address switches and you also can select the CAN baudrate and the CANopen mode by setting the mode switches to a specific position. (See CMC600 operating instructions)

2 CANopen Object Directory

Each CANopen slave device has a CANopen object directory (OBD). It describes all the data objects of the device which can be accessed (read or write) by data transfer on the CAN bus.

In case of SICK ID^{pro} devices we have a very huge object directory. This is because the CANopen protocol is also used for the CAN SICK-network. Most of the entries should not be used by customers. In the electronic data sheet, which describes the OBD, you will find comments for entries which should not be used. Unfortunately we cannot hide some entries, because the CANopen conformance test, which must be passed if the device is a certified CANopen product, checks if every data object which can be found is also entered into the electronic data sheet.

There are some entries which are valid for CANopen users:

Communication segment

Index	Subindex	Name	type	value
1000h	00h	Device type	unsigned 32	0x30191
1001h	00h	Error register	unsigned 8	
1002h	00h	Manufacturer status register	unsigned 32	
1003h	00h .. n	Predefined error field	unsigned 32	
1005h	00h	COP-ID Sync message	unsigned 32	
1008h	00h	Manufacturers Device Name	Visible string	
1009h	00h	Manufacturers Hardware Version	Visible string	
100Ah	00h	Manufacturers Software Version	Visible string	
1010h	01	p301_store_para	unsigned 32	
1011h	01	p301_restore_para	unsigned 32	
1014h	00	COB-ID emergency object	unsigned 32	
1015h	00	Inhibit time emergency	unsigned 16	
1017h	00	Producer Heartbeat time	unsigned 16	
1018h		Identity Object		
	00	number of entries	unsigned 8	
	01	Vendor ID	unsigned 32	0x0056
	02	Product Code	unsigned 32	
	03	Revision number	unsigned 32	
	04	Serial Number	unsigned 32	
1200h		Server SDO Parameter 1		
	00	Number of entries	unsigned 8	
	01	COB-ID Client → Server	unsigned 32	
	02	COB-ID Server → Client	unsigned 32	

Index	Subindex	Name	type	value
1400h	00h .. 02h	Receive PDO comm. param. 1	u8 / u32	
1401h	00h .. 02h	Receive PDO comm. param. 2	u8 / u32	
1402h	00h .. 02h	Receive PDO comm. param. 3	u8 / u32	
1403h	00h .. 02h	Receive PDO comm. param. 4	u8 / u32	
1600h	00h .. 08h	Receive PDO mapping param. 1	u8 / u32	
1601h	00h .. 08h	Receive PDO mapping param. 2	u8 / u32	
1602h	00h .. 08h	Receive PDO mapping param. 3	u8 / u32	
1603h	00h .. 08h	Receive PDO mapping param. 4	u8 / u32	
1800h	00h .. 05h	Transmit PDO comm. param. 1	u8 / u32	
1801h	00h .. 05h	Transmit PDO comm. param. 2	u8 / u32	
1802h	00h .. 05h	Transmit PDO comm. param. 3	u8 / u32	
1803h	00h .. 05h	Transmit PDO comm. param. 4	u8 / u32	
1A00h	00h .. 08h	Transmit PDO mapping param. 1	u8 / u32	
1A01h	00h .. 08h	Transmit PDO mapping param. 2	u8 / u32	
1A02h	00h .. 08h	Transmit PDO mapping param. 3	u8 / u32	
1A03h	00h .. 08h	Transmit PDO mapping param. 4	u8 / u32	

Manufacturer Segment (Used CLV6xx barcode data)

Index	Subindex	Name	type	access
2000h		Read result	unsigned 32	
2000h	00h	Number of entries (value = 4)	unsigned 8	RO
2000h	01h	counter (each read result)	unsigned 8	RO
2000h	02h	length of datastring	unsigned 16	RO
2000h	03h	data valid / free data on write	unsigned 8	R/WW
2000h	04h	Selected output format datastring	Visible string	RO
2001h		Successive single characters of read result	Visible string	
2001h	00h	Number of entries	unsigned 8	RO
2001h	01h .. 50h	One character on each subindex	unsigned 8	RO
2002h	00h	Counter for successive read results (used to see if result has changed)	unsigned 8	RO
2010h		reading diagnosis datastring (same as on serial AUX interface)	unsigned 32	
2010h	00h	Number of entries (value = 4)	unsigned 8	RO
2010h	01h	counter (each successive output)	unsigned 8	RO
2010h	02h	length of datastring	unsigned 16	RO
2010h	03h	data valid / free data on write	unsigned 8	R/WW
2010h	04h	reading diagnosis datastring	Visible string	RO
Index	Subindex	Name	type	access
2020h		command response datastring	unsigned 32	
2020h	00h	Number of entries (value = 4)	unsigned 8	RO
2020h	01h	counter (each successive output)	unsigned 8	RO
2020h	02h	length of datastring	unsigned 16	RO

2020h	03h	data valid / free data on write	unsigned 8	R/WW
2020h	04h	command response datastring	Visible string	RO
2200h		command datastring	domain (500)	WO
2300h		structure 'data available' (usually mapped to TPD=)		
2300h	00h	Number of entries (value = 7)	unsigned 8	Const
2300h	01h	Length of data to be uploaded	unsigned 16	RWR
2300h	02h	Type of data	unsigned 8	RWR
2300h	03h	counter	unsigned 8	RWR
2300h	04h	not used	unsigned 8	RWR
2300h	05h	Node ID (of source = Tx device)	unsigned 8	RWR
2300h	06h	dummy	unsigned 8	RWR
2300h	07h	dummy	unsigned 8	RWR
3000h	00h	Enable bits for device sending data strings via CAN: Bit0: Enable Read Result Datatstring (2000h / 4) Bit1: Enable Diagnosis Data (2010h / 4) Bit2: Enable Command Response (2020h / 4)	unsigned 8	RWW

Device Profile Segment (Digitial I/O)

Index	Subindex	Name	type	access
6000h		CANopen inputs (= Slave device output)		
6000h	00h	Number of entries (value = 2)	unsigned 8	RO
6000h	01h	Digital input byte 0	unsigned 8	RO
6000h	02h	Digital input byte 1	unsigned 8	RO
6200h		CANopen outputs (= slave device input)		
6200h	00h	Number of entries (value = 2)	unsigned 8	RO
6200h	01h	Digital output byte 0	unsigned 8	RWW
6200h	02h	Digital output byte 1	unsigned 8	RWW
6208h		Enable for CANopen outputs (for Slave device inputs)		
6208h	00h	Number of entries (value = 2)	unsigned 8	RO
6208h	01h	Digital output byte 0	unsigned 8	RW
6208h	02h	Digital output byte 1	unsigned 8	RW

3 Access to sensors read result

The CAN interface provides data transfer of either output format #1 or output format #2 like all the other data interfaces of the sensor. Output formats are data strings which are built by the senser at the end of each reading cycle. You can describe their format in the section 'Data Processing' \ ‘Output Format’ of the SOPAS engineering tool.

Like for all other data interfaces of the sensor you can also select for the CAN interface, which one of the data formats has to be communicated.

There are two different methods to transfer read results via CAN:

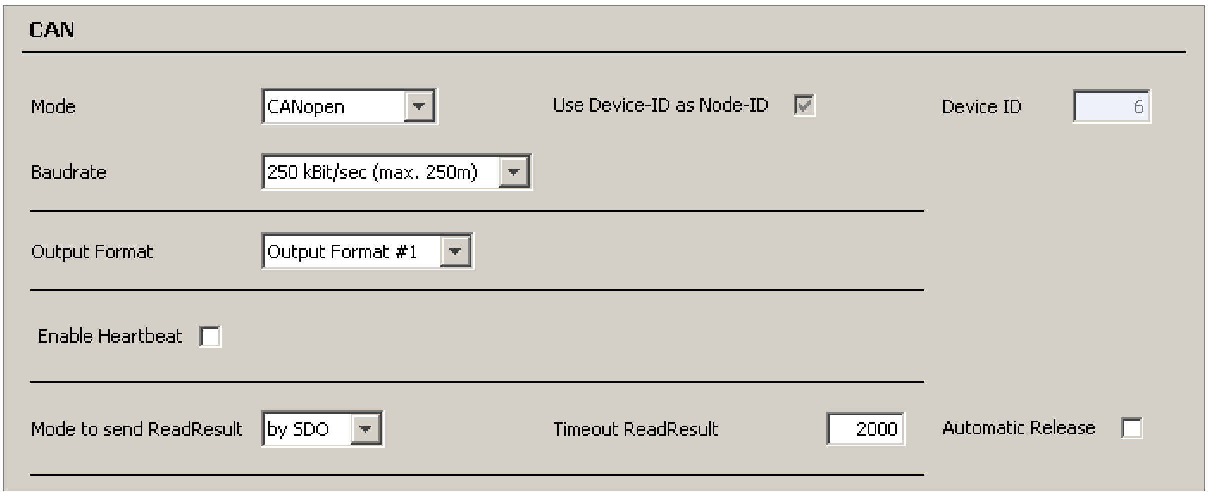
The first is uploading the data string to the PLC by initializing a SDO domain upload. This is the same implementation that was used for the CLV4xx barcode scanners. The process to do this, which is described below, is essentially unchanged from the existing process applying to the older barcode reader family.

The second method is to map the successive characters of the read result to one or several PDOs which will be sent each time a reading cycle has finished.

The selected data format will be entered into the object directory. The reading result is located at 0x2000. It may contain up to 500 characters (max.). At 2000h/01 we have a counter, which is incremented with each successive read result. It may be used to see new data has been sent. At 2000h/02 there is an entry which has the current length of the incoming data.

The object 0x2001 also contains the readresult. This is a data array including the first 50 characters of the read result data data frame. If the length is less than 50 bytes the last arrray elements will be set to 00h.

3.1 Read result Access by SDO upload



CANopen slave devices cannot initialize SDO data transfer themselves. They are always passive. It is a CANopen client's (e.g.PLC) job to initialize a SDO upload or download.

In our case the slave devices (reader devices) will send specific PDO data for signaling that new data is available. The first is called 'data available PDO' described below. A client which receives this PDO must start the SDO upload procedure to get the read result data domain. After the domain upload has finished, the client must signal to its server device that the read result data can be released. It can do this by writing 0x00h to the data valid byte, which is 2000h/03h for the read result domain.

The readers 'Data Available PDO' has a structure like described below: :

Data available PDO:		
Identifier:	11 bit	0x180 + Node id (like predefined connection set for TPDO1)
1st mapped object:	unsigned 16:	length of the read result data domain
2nd mapped object:	unsigend 8:	Type identifier
3rd mappded object:	unsigned 8:	counter, incremented for each read result
4th mapped object:	unsigned 8:	not used
5th mapped object	unsigned 8:	PDO source device Node ID

The type identifiers on the 2nd position of the mapped object, tells us which kind of data is available and at which position in the object directory we can find it:

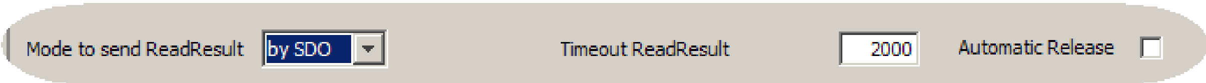
0x01	Read result data string	0x2000/4
0x04	Command response data string	0x2020/4
0x05	Diagnosis data String	0x2010/4
0x81	error: timeout 'Read result data string' reached	0x2000/4
0x84	error: timeout 'Command response data string' reached	0x2020/4
0x85	error: timeout 'Diagnosis data String' reached	0x2010/4

Example for data traffic on the CAN line

Scenario: a slave device (NodeID = 3) has read result data. Length = 10 bytes

CAN-Identifier	**CAN Object type / direction**	**CAN Data**	**comment**
Synchronisations-PDO			
183	PDO von CLV03	0A 00 01 00 03 03	Synchronisations-Pdo Type 01h: Read Result Datastring Length 0Ah: 10 Zeichen Counter: 00h Digout: 01h Knotennummer 03h
SDO Block upload			
603	CLI-->SRV03	A0 **00 20 04** 11 00 00 00	Initiate SDO Block upload request access to **2000h/4**
583	CLI<--SRV03	C2 00 20 04 0A 00 00 00	Response (size = **0000000A**)
603	CLI-->SRV03	A3 00 00 00 00 00 00 00	Initiate SDO Block upload start
583	CLI<--SRV03	01 **53 32 31 36 32 33 31**	Block1 upload, Data : "**S216231**"
583	CLI<--SRV03	82 **36 32 37** 0B FF 6D 35	Block2 upload (last), Data : "**627**"
603	CLI-->SRV03	A2 02 11 00 00 00 00 00	confirm
583	CLI<--SRV03	D1 00 00 00 00 00 00 00	End SDO Block Upload request
603	CLI-->SRV03	A1 00 00 00 00 00 00 00	confirm
SDO Release String			
603	CLI-->SRV03	2F **00 20 03 00** 9C 71 64	SDO download (expedited) Write **0** to **2000h / 3** : Release Read result datastring
583	CLI<--SRV03	60 00 20 03 00 00 00 00	confirm

When the server sends its 'data available PDO', it also starts a timeout to check if the client is starting the upload procedure. If the timer runs out, result data which was attached to the object directory will be released and a second PDO with an error type identifier (bit 7 = 1) will be sent.

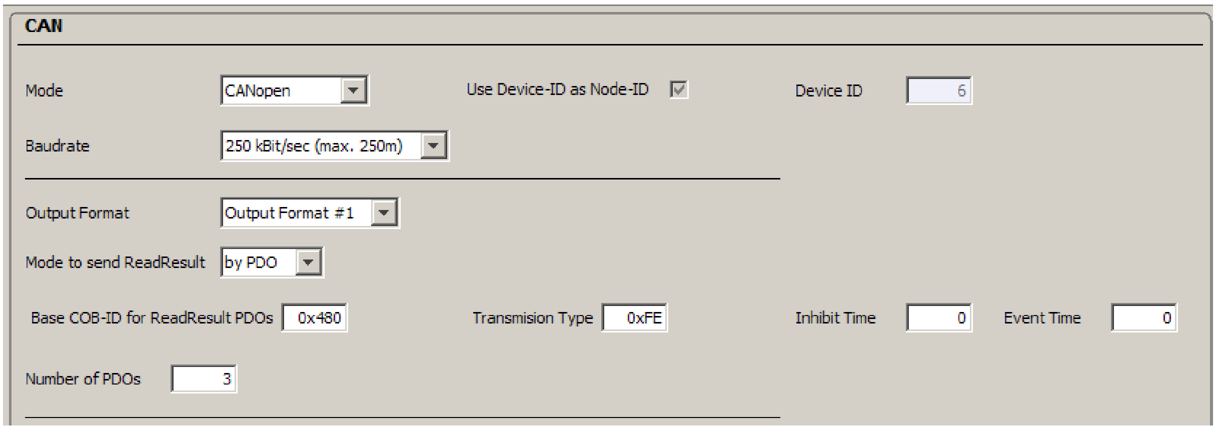


The timout time (in ms) can be selected by in the SOPAS ET.

There is also a checkbox 'automatic release'. If this is active, data will be automatically released after the sdo upload process.

If the client tries to upload read result data while the object directory is empty, it will get an abort message 0x08000022 (data cannot be transferred because of present device state).

3.2 Read result data transfer by PDO



It is also possible to transfer the read result data string within a set of one or several TPDOs.

Up to 50 characters of the read result data string can be sent. Each PDO carries 7 bytes of the string and an additional counter data byte, which is incremented for each successive read result. You need this counter byte to check if you have consistent readresult data within the set of different PDOs. The PDOs are sent with successive COB IDs (CAN object identifiers). In the example you will get 3 PDOs with Identifiers 0x480, 0x481 and 0x482.

COBID	data Byte1 = Counter	byte 2..8(ReadResult)	Datybates ASCII
480h	05h	52h 65h 61h 64h 52h 65h 73h	- R e a d R e s
481h	05h	75h 6Ch 74h 00h 00h 00h 00h	- u l t - - - -
482h	05h	00h 00h 00h 00h 00h 00h 00h	- - - - - - - -

COBID	data Byte1 = Counter	byte 2..8(ReadResult)	Datybates ASCII
480h	06h	31h 32h 33h 34h 35h 36h 37h	- 1 2 3 4 5 6 7
481h	06h	38h 39h 30h 41h 42h 43h 44h	- 8 9 0 A B C D
482h	06h	45h 46h 47h 48h 00h 00h 00h	- E FG H - - -

In the tables above you can see two successive read results. The first readresult consists of a 10 character datastring "Readresult". The second has 18 charcaters: "1234567890ABCDEFGH"

If the transmission type ID is 0xFE (see CANopen spec: asynchronous transmission), a PDO will be sent on change of data. This is at the end of each reading cycle, directly after the selected output format (#1 or #2) was built. If you want to have a cyclic transmission of PDOs you can enter the event time != 0.

It is also possible to map these 50 result characets (CANopen Object 0x2001 Subindex 1-50) to any Transmit PDO (see chapter 4.3.2 Editing PDOs with Sopas ET). The parameter “Mode to send ReadResult” has to be set to “by PDO”.

4 I/O Communication

The sensors fieldbus I/O data can of course be distributed via CAN. A CANopen master device has direct access via SDO communication to the objects 0x6000 (CANopen inputs = Slave device output) and 0x6200 (CANopen outputs = slave device input). There is also an object 0x6208 with enable bits, which define, which output bits (= sensor input bits) must be handled.

4.1 CANopen inputs (slave device outputs) (object 0x6000)

Bit	object	assignment	name	comment
Byte 0, Bit 0	0x6000/1	fixed	Device Ready	
Byte 0, Bit 1	0x6000/1	fixed	System Ready	Not for CANopen
Byte 0, Bit 2	0x6000/1	fixed	Good Read	
Byte 0, Bit 3	0x6000/1	fixed	No Read	
Byte 0, Bit 4	0x6000/1	fixed	Status External Output 1	Physical Output 1 of CDF600
Byte 0, Bit 5	0x6000/1	fixed	Status External Output 2	Physical Output 2 of CDF600
Byte 0, Bit 6	0x6000/1	fixed	Status Output 1 (Result 1)	
Byte 0, Bit 7	0x6000/1	fixed	Status Output 2 (Result 2)	
Byte 1, Bit 0	0x6000/2	fixed	External Input 1	Physical Input 1 of CDF600
Byte 1, Bit 1	0x6000/2	fixed	External Input 2	Physical Input 2 of CDF600
Byte 1, Bit 2	0x6000/2	fixed	Input 1 (Sensor 1)	
Byte 1, Bit 3	0x6000/2	fixed	Input 2 (Sensor 2)	
Byte 1, Bit 4	0x6000/2	soft	Defined by sensor configuration	Not yet implemented
Byte 1, Bit 5	0x6000/2	soft	Defined by sensor configuration	Not yet implemented
Byte 1, Bit 6	0x6000/2	soft	Defined by sensor configuration	Not yet implemented
Byte 1, Bit 7	0x6000/2	soft	Defined by sensor configuration	Not yet implemented

4.2 CANopen outputs (slave device inputs) (object 0x6200)

Bit	Object	assignment	name	comment
Byte 0, Bit 0	0x6200/1	fixed	Trigger	
Byte 0, Bit 1	0x6200/1	fixed	Sensor-Idle	
Byte 0, Bit 2	0x6200/1	fixed	TeachIn1	
Byte 0, Bit 3	0x6200/1	fixed	TeachIn2	
Byte 0, Bit 4	0x6200/1	fixed	External Output_1	Physical Output 1 of CDF600
Byte 0, Bit 5	0x6200/1	fixed	External Output_2	Physical Output 2 of CDF600
Byte 0, Bit 6	0x6200/1	fixed	Digital Output_1 (Result_1)	
Byte 0, Bit 7	0x6200/1	fixed	Digital Output_2 (Result_2)	
Byte 1, Bit 0	0x6200/2		PLC_Out_08	
Byte 1, Bit 1	0x6200/2	soft	PLC_Out_09	
Byte 1, Bit 2	0x6200/2	soft	PLC_Out_10	
Byte 1, Bit 3	0x6200/2	soft	PLC_Out_11	
Byte 1, Bit 4	0x6200/2	fixed	Distance_Config_0	LSB
Byte 1, Bit 5	0x6200/2	fixed	Distance_Config_1	
Byte 1, Bit 6	0x6200/2	fixed	Distance_Config_2	
Byte 1, Bit 7	0x6200/2	fixed	Distance_Config_3	MSB

4.3 PDO mapping for digital I/O

The I/O that can be found in the object directory can be mapped to the devices PDOs.

The device has 4 receive PDOs (RPDO1 .. RPDO4) and 4 transmit PDOs (TPDO1 .. TPDO4) which can be mapped by a user. Digital outputs (sensor inputs, Ox6200) can be mapped to RPDOs while digital inputs (0x6000) can be mapped to TPDOs. There are different methods how to do this:

4.3.1 Mapping by writing to OBD (standard CANopen method)

A CANopen user can enter a setup for TPD0 1..4 and for RPDO 1..4 like it is common for CANopen slave devices. By writing to the Objects 0x1400 .. 0x1403 he can set the RPDO communication parameters. 0x1600 .. 0x1603 is used for RPDo mapping parameters, 0x1800 .. 0x1803 for TPDO communication parameters and 0x1A00 .. 0x1A03 for TPDO mapping parameters. An experienced CANopen user knows how to do this.

These parameters can be stored permanently by writing to the Object 0x1010.

They are part of the sensors parameter set and so they will also be stored in an external parameter cloning device. (CMC600)

Note: If you want to use the 'Data Available PDO' which was described above, you should not use TPDO1 for your application.

4.3.2 Editing PDOs with Sopas ET

4.3.2.1 TPDOs

Communication and mapping parameters TPDO 1..4 and RPD0 1..4 can also be edited with the SOPAS engineering tool. For each PDO you can find a table in the SOPAS tool where communcation- and Mapping parameters can be entered:

CANopen Transmit PDOs 1 .. 4

Transmit PDOs

	TPDO1	TPDO2	TPDO3	TPDO4
Predef. conn.-	Yes	No	No	Yes
COB-ID	0x80000000	0x00000184	0x00000185	0x80000000
Transm. Type	0xFE	0xFE	0xFE	0xFE
Inhibit Time	0	0	0	0
Event Time	0	0	500	0
Num of map. o	7	2	2	0
Map Obj. 1	0x23000110	0x60000108	0x60000208	0x00000000
Map Obj. 2	0x23000208	0x60000208	0x60000108	0x00000000
Map Obj. 3	0x23000308	0x00000000	0x00000000	0x00000000
Map Obj. 4	0x23000408	0x00000000	0x00000000	0x00000000
Map Obj. 5	0x23000508	0x00000000	0x00000000	0x00000000
Map Obj. 6	0x23000608	0x00000000	0x00000000	0x00000000
Map Obj. 7	0x23000708	0x00000000	0x00000000	0x00000000
Map Obj. 8	0x00000000	0x00000000	0x00000000	0x00000000

Communication parameters:

If you use the predefined connection set, the CAN identifier for your PDO will be selected automatically as described in the CANopen Spec. There will be individual IDs for the PDO depending on the PDO type and number and also on the Node ID of the slave device.

The assigned Identifiers will be built from a base address (which is different for each PDO) plus the Node ID of the device (see Cia DS301 9.4.3).

PDO identifier base addresses:

PDO	Identifier base
TPDO1	0x180
TPDO2	0x280
TPDO3	0x380
TPDO4	0x480
RPDO1	0x200
RPDO2	0x300
RPDO3	0x400
RPDO4	0x500

Example: PDO identifier of TPD3 for a device with Node ID 9 will be 0x389

Especially for TPDOs you should use the predefined connection set identifiers if it is possible for your application. This will avoid conflicts in the assignment of identifiers.

Unfortunately the SOPAS tool cannot show, which identifier will be selected if you use the predifined connection set.

If the predefined connection set is disabled ('NO' selected) you can select your own identifier in the row 'COB-ID'. The entered value will be written to subindex 1 of the PDO communication parameter. Be aware, that we have an 11 bit Identifier, but a 32 bit value can be entered. (See CiA DS 301 9.6.3 object 1400h - 15ffh Table 55)

NOTE: The lowest value for a user defined PDO identifier is 0x180

The 'Tansmission Type' is the next parameter of the PDOs communcation object. The preselected value 0xFE selects asynchronous PDO transfer, which will be used in most of the CANopen systems. CANopen users who want to select another transmission type will be familiar with the CANopen spec, where you can find detailed description of the transmission type. (See CiA DS 301 6.3)

The inhibit time allows a guaranteed minimum pause to be configured between sending successive PDOs of the same type. Its unit is 100 us. The minimum pause is disabled if this value is 0. This parameter is normally not used. (See CiA DS 301 9.6.3 Table 55)

The Event Time allows sending a PDO cyclical. The value describes the cycle time in ms. If 0 the PDO will be sent on change of any of its mapped objects.

Mapping parameters:

Up to 8 objects from the OBD can be mapped to each PDO. First set the number of mapped objects to 0. Then enter the objects into the table. Each mapping object description is a 32 bit data value. You must enter the data format which is described in the CANopen spec for mapping objects:

MSB LSB

Index (16 bit)	Subindex (8 bit)	Object lengt (num of bits) (8 bit)

Example: '0x60000108' describes the 'Digital input byte 0':

Index = 0x6000, Subindex = 0x01, Number of Bits = 8

(See CiA DS 301 9.6.3 Object 1600h - 17ffh)

After entering all mapping objects needed, enter the number of objects which are used in your mapping.

In the sopas configuration example above there are three TPDOs mapped.

The first is the mapping for the 'Data available PDO' which is automatically entered if you select 'SDO' data transfer for the read result. The COB-ID is default and selects the predefined connection set identifier. It is 0x180 + Node ID for the first TPDO.

If the device ID (= CAN NodeID) of our slave device is 6 the data available PDO will be sent with COB-ID 0x186.

The second TPDO maps the two digital input data bytes 0x6000/01 and 0x6000/02 and assigns the Identifier 0x184. The event time is 0, so the PDO will be sent each time when any of the digitial input bits changes.

The third TPDO also maps also the two digital input data bytes but in different order. It assigns the Identifier 0x185. The event time is 500, so the PDO will be sent every 500 ms and also if any of the digitial input bits changes.

4.3.2.2 RPDOs

CANopen Receive PDOs 1 .. 4

Receive PDOs

	RPDO1	RPDO2	RPDO3	RPDO4
Predef. conn.-	No	No	Yes	Yes
COB-ID	0x00000220	0x00000230	0x80000000	0x80000000
Transm. Type	0xFE	0xFE	0xFE	0xFE
Num of map. o	1	1	0	0
Map Obj. 1	0x62000108	0x62000208	0x00000000	0x00000000
Map Obj. 2	0x00000000	0x00000000	0x00000000	0x00000000
Map Obj. 3	0x00000000	0x00000000	0x00000000	0x00000000
Map Obj. 4	0x00000000	0x00000000	0x00000000	0x00000000
Map Obj. 5	0x00000000	0x00000000	0x00000000	0x00000000
Map Obj. 6	0x00000000	0x00000000	0x00000000	0x00000000
Map Obj. 7	0x00000000	0x00000000	0x00000000	0x00000000
Map Obj. 8	0x00000000	0x00000000	0x00000000	0x00000000

In the configuration example above we entered 2 receive PDOs for the device. Each receives a single Byte. RPDO1 receives the low byte of the digital output object 0x6200 / 01. (Length = 08 bits)

The assigned COB-ID is 0x220. A CAN object with Identifier 0x220 can be used to trigger the SICK ID*pro* device. Bit 0 of 0x6200 / 01 is the trigger bit.

Bit	object	assignment	name
Byte 0, Bit 0	0x6200/0	fixed	Trigger
Byte 0, Bit 1	0x6200/0	fixed	Sensor-Idle
Byte 0, Bit 2	0x6200/0	fixed	TeachIn1
Byte 0, Bit 3	0x6200/0	fixed	TeachIn2
Byte 0, Bit 4	0x6200/0	fixed	External Output_1
Byte 0, Bit 5	0x6200/0	fixed	External Output_2

You need some other settings in the SICK ID*pro* device, to actually enable triggering by this PDO:

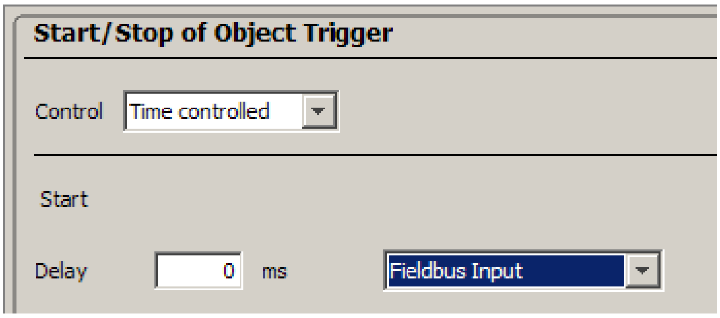
The digital input mask (which is the digital output mask from the PLCs point of view)

must be set:

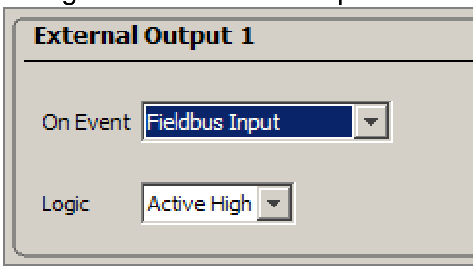
Mask for Dig. Input	0x11

In this example bit 5 and bit 1 is enabled. So the digital output of the PLC is enabled to access the the external output (Bit5) and the trigger bit.

Also, to enable this trigger functionality you must select the 'Fielbus Input' for the trigger configuration in SOPAS.



For setting the external Output directly by PDO the function oft the digital output bit must be configured as “Fieldbus Input”.



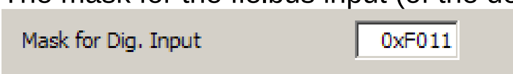
Digital output byte 1 of our device (0x6200/01) is mapped within the second RPDO.

It has its own identifier and so another source device may access those output bits.

Byte 1, Bit 0	0x6200/1		PLC_Out_08	
Byte 1, Bit 1	0x6200/1	soft	PLC_Out_09	
Byte 1, Bit 2	0x6200/1	soft	PLC_Out_10	
Byte 1, Bit 3	0x6200/1	soft	PLC_Out_11	
Byte 1, Bit 4	0x6200/1	fixed	Distance_Config_0	LSB
Byte 1, Bit 5	0x6200/1	fixed	Distance_Config_1	
Byte 1, Bit 6	0x6200/1	fixed	Distance_Config_2	
Byte 1, Bit 7	0x6200/1	fixed	Distance_Config_3	MSB

Bits 7 .. 4 of this byte can be used to change the distance configuraton of the CLV6xx. As shown above, you need to select some other switching parameters, to get the distance configuration changes by the fieldbus master device:

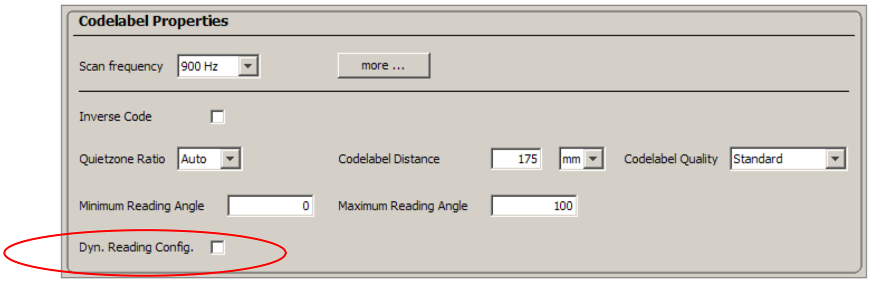
The mask for the fielbus input (of the device) must be enabled.



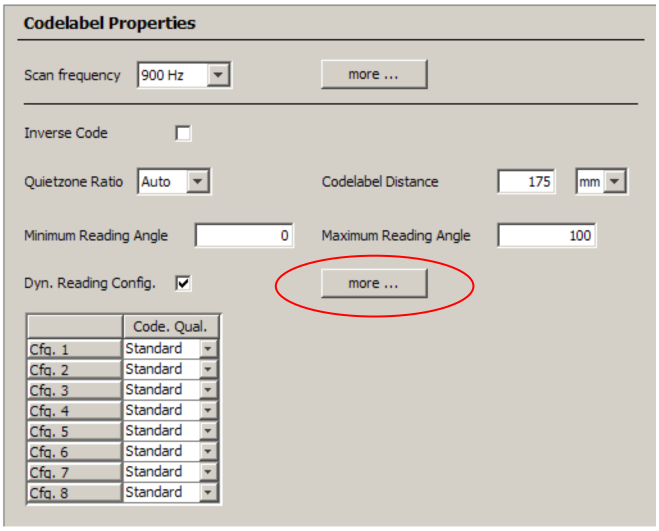
Also the distance configuration settings must be assigned to the fieldbus device. (If using a barcode scanner that has dynamic focus functionality as an option)

To get the menue below you must login to the barcode scanner at “Service Level” in SOPAS and select a specific menu path:

Menu: ReadingConfiguration - activate 'dynamic reading configuration'

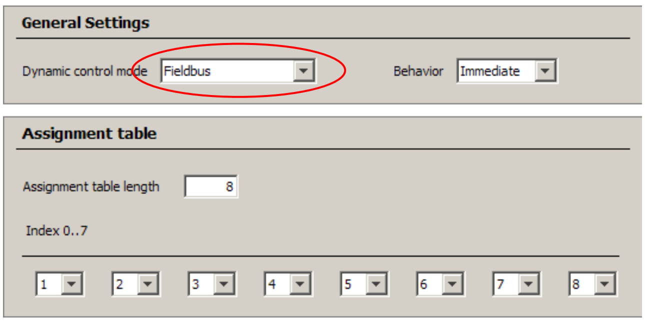


Select 'more' !



The 'Dynamic Control mode' must be set to 'Fieldbus'

The distance configuration bits 0..3 entered by our RPDO will be handled as index value (0..7) for the resulting distance configuration.

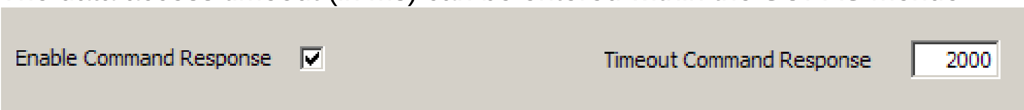


5 Sopas commands via CANopen

A CANopen client device can send SOPAS commands to a SICK ID*pro* device, just like commands can be sent by any other command interface. In case of CANopen the client must start a SDO download and write the command to the object directory of the accessing server device. You can find the command input object at index 0x2200/00 in the sensors OBD. The sensor will interpret each command, after the SDO download sequence has finished. It will put its command response in object 0x2020 and start a 'data available PDO' as described above for sending read result data strings. In case of a command response the Type identifier within the PDO is 0x04 (see table in chapter 3.1)

The procedure to get the command response is the same as getting read results. Except the OBD-entry to be accessed is 0x2020/04 and data release must be done by write access (data = 0) to 0x2020/03

The data access timeout (in ms) can be entered within the SOPAS menue.

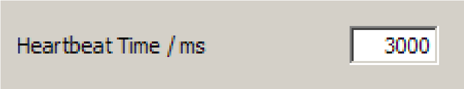


6 Reading diagnosis via CANopen (only CLV6xx)

Reading diagnosis data strings which normaly are sent on the AUX interface, can also be directed to the CANopen interface.

Getting the diagnosis data string is the same procedure as for read results and command response datastrings. The 'data available PDO will use a type identifier 0x05 (see table in chapter 3.1) . The diagnosis data strings can be uploaded from 0bject 0x2010/4

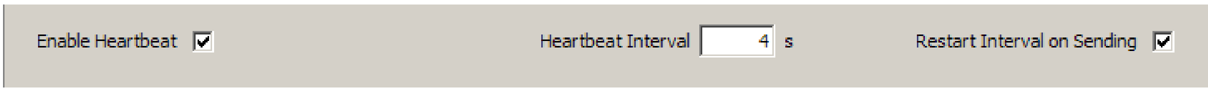
7 CANopen heartbeat objects



CANopen heartbeat objects (CAN objects with identifier 0x700 + NodeID) can be enabled by setting the heatbeat time to value != 0.

Of course heartbeating can also be enabled by writing to object 0x1017 = Producer Heartbeat Time same holds for any CANopen slave device.

8 Device specific heartbeat telegrams



This checkmark can be used to enable sending of heartbeat telegrams on the CAN bus. Heartbeat telegrams are specific datastrings which are defined in the data format section of the SOPAS menue. They can be sent like read result data strings, while there is no reading trigger for the device. It is just the same behavior as if a heartbeat telegram is sent on a serial interface instead of a read result telegram.

Our CANopen subsytem handles hearbeat telegrams in the same way as read result datastrings. It’s the same Object 0x2000/04.

9 The Emergency Object

9.1 Assigning the emergency object ID

The emergency object ID can be assigned using the SOPAS engineering tool.

You can also set the Emergency Object ID by writing to Object 0x1014.

9.2 Emergency Object Content

Several software instances of the sensor device can detect irregular states within the sensor. And there is a shared instance, called errorhandler, which collects information about status events which are detected all over the device.

Each warning can be identified by a 32 bit value which is divided in several bit groups.

Bit 31..24: Errorlevel:

Debug Error	0x01
Info	0x02
Warning	0x03
Error	0x04
Fatal Error	0x05

Bit 17 .. 8: Subsystem where error was detected

Genral errors	0x00
Error for test purpose	0x01
CAN-Network	0x02
Network general	0x03
Network monitor	0x04
External devices	0x07

Bit 7 .. 0: Error numbers within a subsystem

The errorhandler notifies each event to the CANopen system of the device, which enters entries into the object directory. This way sending of emergency objects is triggered.

All error events are put into the manufacturer status register 0x1002 formated in the way which is shown above. If there is an error reset event on an error type which was allready entered to object 0x012, then the entry will be deleted.

NOTE: If there was already a second error event put into the fifo, when the reset event happens, the error cannot be deleted within the fifo.

Unfortunately the CANopen error sytem cannot handle the 32 bit error states of our device.

So the errors are mapped to the CANopen error system as described below:

Entries to the error register (object 0x1001):

0x81 → generic error bit and manufacturer specific error bit will be set

Data of the emergency object

Emergency error Code (Bytes 0 and 1 of the Emergency Object)

Highbyte = 0xFF (→ DS301: 'Device specific error')

Lowbyte = Errorcode not an explicit entry, no subsystem information

Error register entry (Byte 2 of the emergency object)

0x81 for device specific errors.

Manufacturer specific error field (Byte 3 and 4 of the emergency object)
(= Byte 3 and 2 of the predefined error field)

Byte 3: error generating subsystem
Byte 4: error level