FIRMWARE MANUAL

UIO inputs / outputs firmware

Description

UIO firmware is the generic firmware for the whole range of UIO UniSwarm input output. Functionalities can vary depending on your hardware board reference and revision. Please check the specific hardware datasheet.

Features

- Compatibility with CANOpen protocol with CiA DS401 profile
- Low level Digital Signal Processing capable

Interfaces

- CAN Fd bus compatible with CANOpen
- RS485 / RS422 interface for protocols like Modbus, Profibus or DMX512...

Inputs

- Analog and digital inputs
- Configurable digital filters

Outputs

- Digital and PWM outputs
- Frequency configurable

.

Contents

Chapter 1

CiA 401

1.1 401 Profile for I/O devices

The CiA 401 specifies the CANopen interface for generic digital and analog input and output devices. The purpose of the I/O modules is to connect sensors and actors to the CANopen network. They can receive configuration information via the service data objects such as I/O configurations, conversion parameters for converting data into meaningful measurements and so on. At run time, data can be read from the sensor over the CAN bus by either a request or event-driven mechanism. The I/O modules also have a process data object mapping, which may be configured over a service data object for real-time operation.

1.1.1 Object

In this chapter, most of the objects described are indexed in the Standardized profile area (section 2.3) with the index range `0x6000` to `0x67FF`. Some objects are manufacturer specific with the index range `0x2000` to `0x5FFF`.

1.1.2 Board abilities

Introduction

If a device supports a specific type of I/O functionality (analog/digital I/O) it shall support the related default PDOs. However, the module may support additional manufacturer-specific PDOs. If variable PDO mapping is supported the PDO default settings may be changed by means of configuration. If a device does not support a specific I/O function, the related default PDOs remain unused.

`0x1000` **Device Type**

This object describe the type of device and the functionalities supported. Figure 1.1 shows the values of the structure. Table 1.1 defines the values for the I/O functionnalities and M (mapping of PDOs).

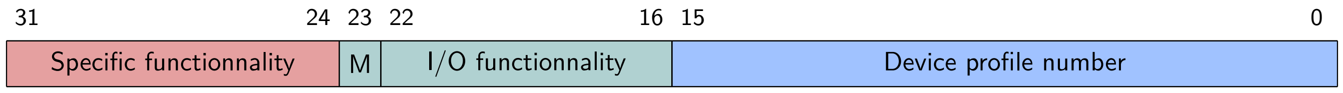


Figure 1.1: Value structure

Field name	**Definition**	
Device profile number	401_d	
I/O functionality - Bit 16	`b1` = digital input(s) implemented	`b0` = not implemented
I/O functionality - Bit 17	`b1` = digital output(s) implemented	`b0` = not implemented
I/O functionality - Bit 18	`b1` = analog input(s) implemented	`b0` = not implemented
I/O functionality - Bit 19	`b1` = analog output(s) implemented	`b0` = not implemented
I/O functionality - Bit 20 to Bit 22	Reserved	
M(apping of PDOs)	`b1` = device-specific PDO mapping is supported	`b0` = pre-defined, generic PDO mapping is supported

Table 1.1: Value definition for I/O functionalities and M

Table 1.2 defines the values of the specific functionnalities subfield.

Code	Function
0x00	No specific function
0x01	Joystick
0x02	Joystick
0x03	Joystick
0x04 to 0xFF	Reserved

Table 1.2: Value definition for specific functionalities

1.1.3 I/O channel to sub-index relation

For binary digital input or outputs functionalities, multiple channels may be regrouped in the same primary data. For exemple, an object storing 8-bit long data will contain a binary data for 8 channels on a sub-index.
In this case, the first sub-index regroups the channels number 1 to 8, the second sub-index the channels number 9 to 16 and so on.
The bit position is be calculated by the following formula:

$$Bit\ position = (I/O\ channel\ num. - 1) \mod data\ length$$

The sub-index, where a bit is located, is calculated by the following formula:

$$Sub\ index = (I/O\ channel\ num. - 1) \div data\ length + 1$$

Figure 1.2 shows an exemple:

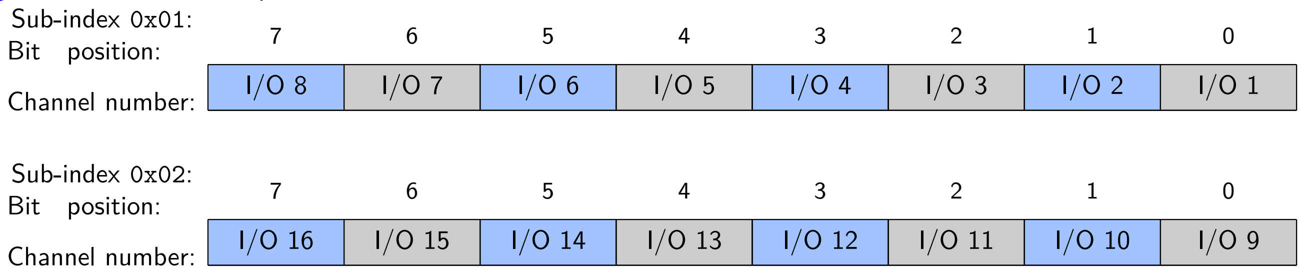


Figure 1.2: Exemple for an 8-bit access

1.1.4 Channel modes

0x4200 DO_Mode

This object controls the output state of the output channel 'n'. It allow the user to configure the channels as a digital output, an analog output, or disconnect the output stage from the channel. It also allow to choose the output mode of the board: push-pull, open-drain or open-source. On input and output capable channels, the input reading is not disconnected and can still be used.
Note: all channels may not support the full range of output modes. Please check your specific board hardware datasheet.
Each I/O channel is accessed with a dedicated sub-index in this object. Channel 1 is at the sub-index 1, channel 2 is at the sub-index 2 and so on:

Index	SubIndex	Name		
0x4200	1	Channel_0		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT16	RW	0	-	[;]

Figure 1.3: Object description 0x4200.1 *Channel_0*

Table 1.3 gives the values corresponding to the differents channel modes:

Channel mode	Value
No output	0x0000
Open-drain	0x0001
Open-source	0x0002
Push-Pull	0x0003
PWM Open-drain	0x0011
PWM Open-source	0x0012
PWM Push-Pull	0x0013

Table 1.3: Values for the differents channel modes

The output at the I/O is dependant on the selected I/O mode and the output logic signal (digital or PWM analog). Table 1.4 shows the produced output on an output channel according to the output mode and output logic state:

Output mode	Output logic state	Output channel state
No output	b0	Open[1]
	b1	Open[1]
Open-drain	b0	Open[1]
	b1	Connected to the drain
Open-source	b0	Open[1]
	b1	Connected to the source
Push-Pull	b0	Connected to the drain
	b1	Connected to the source

[1]The open state may not be equivalent to a disconnected state. Please check the specific board hardware manual for more informations.

Table 1.4: Output states according to the channel mode configuration

1.1.5 Digital Inputs

Data flow diagram

The following diagram shows how the data is handled inside the board and serves as a quick reference to understand how the different objects interact:

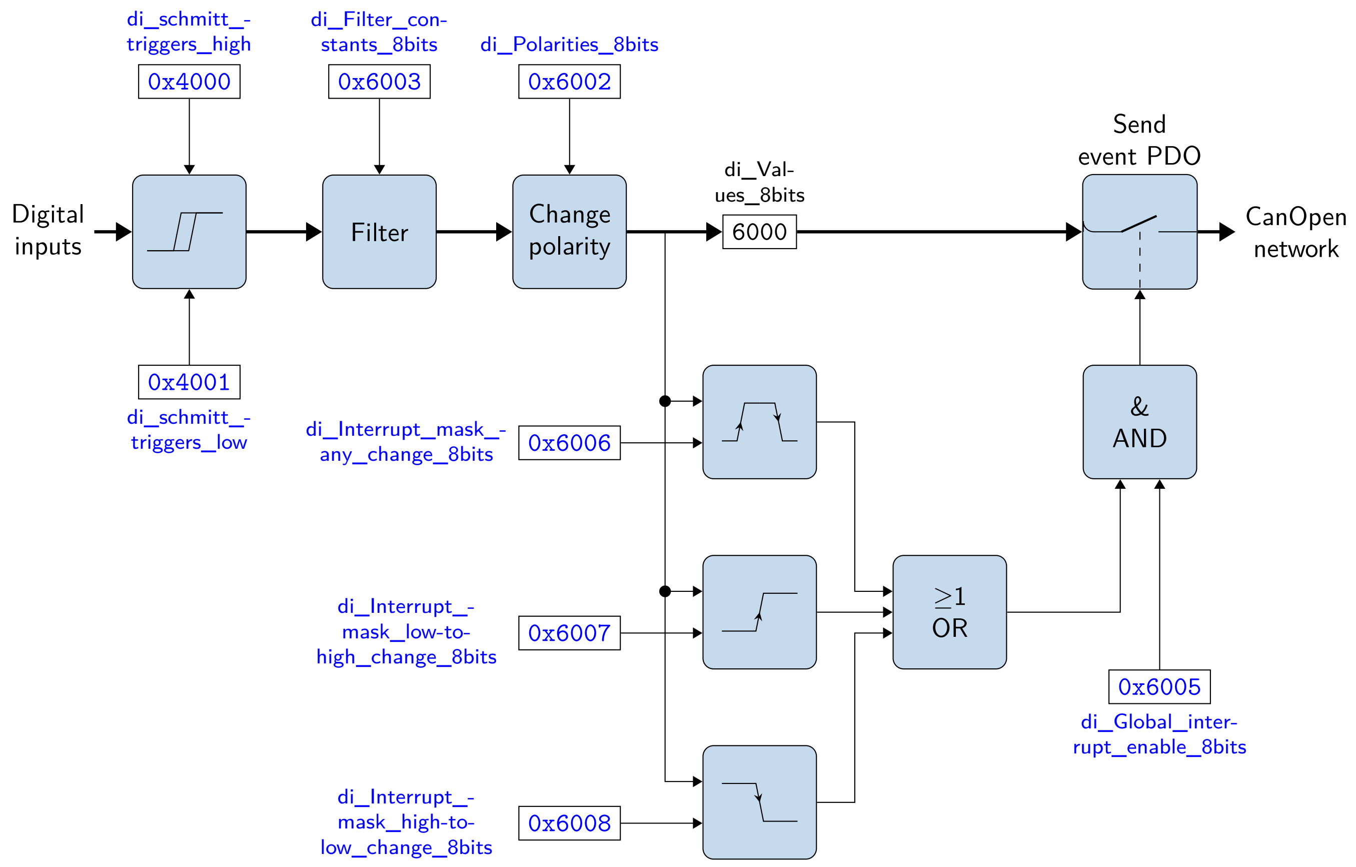

di_schmitt_-triggers_high
0x4000
di_Filter_con-stants_8bits
0x6003
di_Polarities_8bits
0x6002
Digital inputs
Filter
Change polarity
di_Val-ues_8bits
6000
Send event PDO
CanOpen network
0x4001
di_schmitt_-triggers_low
di_Interrupt_mask_-any_change_8bits
0x6006
& AND
di_Interrupt_-mask_low-to-high_change_8bits
0x6007
≥1 OR
0x6005
di_Global_inter-rupt_enable_8bits
di_Interrupt_-mask_high-to-low_change_8bits
0x6008

Figure 1.4: Data flow diagram for the digital inputs

0x6000 di_Values_8bits

This object read a group of 8 inputs and regroup them in a sub-index. A maximum of 254 x 8 inputs are adressable (2032 inputs), depending on your specific hardware.

The number of sub-index on a specific board is stored at the sub-index 0 of this object:

Index	SubIndex	Name		
0x6000	1	Channels_0-7		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT8	RO,TPDO	0	-	[;]

Figure 1.5: Object description 0x6000.1 *Channels_0-7*

0x4000 **di_schmitt_triggers_high**

This object sets the high voltage threshold for the nth channel digital input Shmitt trigger. The digital input logic signal will switch from b0 to b1 when the voltage on the input is greater than the one set at the corresponding sub-index. This value is set on 16 bits, and correspond to the value read by the object 0x6401 ai_Values_16bits.

Each digital input high level trigger is accessed with a dedicated sub-index in this object. Channel 1 is at the sub-index 1, channel 2 is at the sub-index 2 and so on:

Index	SubIndex	Name		
0x4000	1	Channel_00		
Data Type	**Acces**	**Default**	**Unit**	**Range**
INT16	RW	-	-	[;]

Figure 1.6: Object description 0x4000.1 *Channel_00*

0x4001 **di_schmitt_triggers_low**

This object sets the low voltage threshold for the nth channel digital input Shmitt trigger. The digital input logic signal will switch from b1 to b0 when the voltage on the input is less than the one set at the corresponding sub-index. This value is set on 16 bits, and correspond to the value read by the object 0x6401 ai_Values_16bits.

Each digital input low level trigger is accessed with a dedicated sub-index in this object. Channel 1 is at the sub-index 1, channel 2 is at the sub-index 2 and so on:

Index	SubIndex	Name		
0x4001	1	Channel_00		
Data Type	**Acces**	**Default**	**Unit**	**Range**
INT16	RW	-	-	[;]

Figure 1.7: Object description 0x4001.1 *Channel_00*

0x6002 **di_Polarities_8bits**

This object defines the polarity of a bank of 8 input channels. The input polarity of a channel invert the logic state of the input in the board. This step happen after the Schmitt-trigger process if your board support it. The inputs polarities can be inverted individually.
The input is inverted when the corresponding bit of this object is set to b1.

Index	SubIndex	Name		
0x6002	1	Channels_0-7		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT8	RW	0	-	[;]

Figure 1.8: Object description 0x6002.1 *Channels_0-7*

0x6003 **di_Filter_constants_8bits**

This object enables or disables an additional configurable filter. The type of the filter and its configuration may vary depending on your specific board, please check the corresponding hardware datasheet for more informations.
The filter is enabled when the corresponding bit of this object is set to b1.

Index	SubIndex	Name		
0x6003	1	Channels_0-7		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT8	RW	0	-	[;]

Figure 1.9: Object description 0x6003.1 *Channels_0-7*

0x6005 **di_Global_interrupt_enable_8bits**

This object enables or disables the interrupt behavior for all the digital inputs without changing the interrupt masks of the objects 0x6006 di_Interrupt_mask_any_change_8bits, 0x6007 di_Interrupt_mask_low-to-high_change_8bits and 0x6008 di_Interrupt_mask_high-to-low_change_8bits.
Digital input interrupts are enabled when this object is set to True (b1) and disabled when set to False (b0).

Index	SubIndex	Name		
0x6005	0	di_Global_interrupt_enable_8bits		
Data Type	**Acces**	**Default**	**Unit**	**Range**
BOOLEAN	RW	1	-	[;]

Figure 1.10: Object description 0x6005.0 *di_Global_interrupt_enable_8bits*

0x6006 **di_Interrupt_mask_any_change_8bits**

This object determines which digital input channels should produce an interruption on a rising or falling edge detection. Interruptions are enabled when the corresponding bit in this object is set to b1.

Index	SubIndex	Name		
0x6006	1	Channels_0-7		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT8	RW	255	-	[;]

Figure 1.11: Object description 0x6006.1 *Channels_0-7*

0x6007 **di_Interrupt_mask_low-to-high_change_8bits**

This object determines which digital input channels should produce an interruption on a rising edge detection. If the input is inverted by the 0x6002 di_Polarities_8bits object, the interrupt still happen on the rising logical edge, which corresponds to the falling physical edge.
Interruptions are enabled when the corresponding bit in this object is set to b1.

Index	SubIndex	Name		
0x6007	1	Channels_0-7		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT8	RW	0	-	[;]

Figure 1.12: Object description 0x6007.1 *Channels_0-7*

0x6008 **di_Interrupt_mask_high-to-low_change_8bits**

This object determines which digital input channels should produce an interruption on a falling edge detection. If the input is inverted by the 0x6002 di_Polarities_8bits object, the interrupt still happen on the falling logical edge, which corresponds to the rising physical edge.
Interruptions are enabled when the corresponding bit in this object is set to b1.

Index	SubIndex	Name		
0x6008	1	Channels_0-7		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT8	RW	0	-	[;]

Figure 1.13: Object description 0x6008.1 *Channels_0-7*

1.1.6 Digital Outputs

The following diagram shows how the data is handled inside the board and serves as a quick reference to understand how the different objects interact:

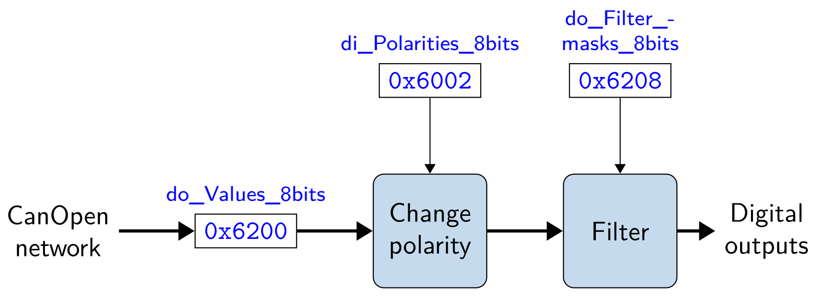


Figure 1.14: Data flow diagram for the digital outputs

0x6200 **do_Values_8bits**

This object read a group of 8 outputs and regroup them in a sub-index. A maximum of 254 x 8 outputs are adressable (2032 outputs), depending on your specific hardware.
The number of sub-index on a specific board is stored at the sub-index 0 of this object:

Index	SubIndex	Name		
0x6200	1	Channels_0-7		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT8	RW,RPDO	0	-	[;]

Figure 1.15: Object description 0x6200.1 *Channels_0-7*

0x6202 **do_Polarities_8bits**

This object defines the polarity of the digital output channels. When the polarity of a digital output channel is inverted, the physical output state is inverted from the logical output state in object 0x6200 do_Values_8bits.

Index	SubIndex	Name		
0x6202	1	Channels_0-7		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT8	RW	0	-	[;]

Figure 1.16: Object description 0x6202.1 *Channels_0-7*

0x6208 **do_Filter_masks_8bits**

This object enables or disables an output filter mask for the digital outputs. The behavior of the output channels logic state compared to the requested values in object 0x6000 di_Values_8bits depends from the corresponding bit in this object as follows:
b1: the output logic state is set to the requested output value.
b0: the requested output value is ignored, the output logic state is kept the same.

Index	SubIndex	Name		
0x6208	1	Channels_0-7		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT8	RW	255	-	[;]

Figure 1.17: Object description 0x6208.1 *Channels_0-7*

1.1.7 Analog Inputs

The following diagram shows how the data is handled inside the board and serves as a quick reference to understand how the different objects interact:

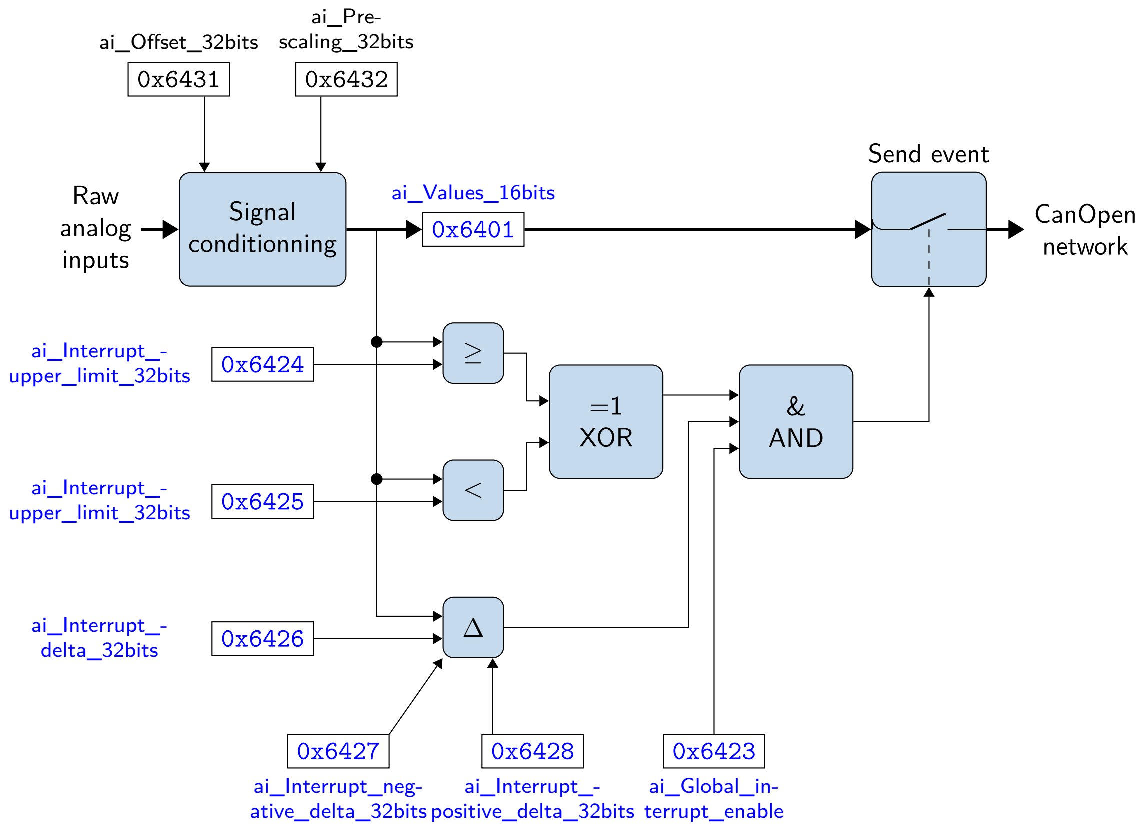


Figure 1.18: Data flow diagram for the analog inputs

`0x6401` ai_Values_16bits

This object read the value of the input channel 'n'. Value is 16-bit wide. Depending on the resolution of your specific hardware, there may be less than 16 significant bits, but the value is always left-ajusted to occupy the most significant bits. The remaining bits at the right side of the LSB are set to zero.
Note: The maximum mesurable voltage may vary, please refer to your board hardware datasheet.
Each analog input channel is accessed with a dedicated sub-index in this object. Channel 1 is at the sub-index 1, channel 2 is at the sub-index 2 and so on:

Index	SubIndex	Name		
0x6401	1	Channel_0		
Data Type	**Acces**	**Default**	**Unit**	**Range**
INT16	RO,TPDO	0	-	[;]

Figure 1.19: Object description `0x6401.1` *Channel_0*

`0x6421` ai_Interrupt_triggers

This object determines which event should produce an interrupt for a specific input channel. All bits set to `b1` will enable the corresponding event, all bits set to `b0` will output a False logical state for the corresponding event.
The following table defines how the object is structured:

Field	Definition
Bit 0	Upper limit exceeded
Bit 1	Lower limit subceeded
Bit 2	Input changed by more than delta
Bit 3	Input reduced by more than negative delta
Bit 4	Input increased by more than positive delta
r	Reserved for future use

Table 1.5: Analog input interrupt trigger selection definition

Index	SubIndex	Name		
0x6421	1	Channel_0		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT8	RW	7	-	[;]

Figure 1.20: Object description 0x6421.1 *Channel_0*

0x6422 **ai_Interrupt_source**

This object is used to determine which analog input channel produced an interrupt. The bits set relate to the channels number that produced an interrupt.
A bit set to 1 means that the corresponding channel produced an interrupt. **Note:** This object has a 32 bits format.

Index	SubIndex	Name		
0x6422	1	Channels 0-7		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RO,TPDO	0	-	[;]

Figure 1.21: Object description 0x6422.1 *Channels 0-7*

0x6423 **ai_Global_interrupt_enable**

This object enables or disables the interrupt behavior for all the analog inputs without changing the interrupt masks of the object 0x6421 *ai_Interrupt_triggers*.
Analog input interrupts are enabled when this object is set to True (b1) and disabled when set to False (b0).

Index	SubIndex	Name		
0x6423	0	ai_Global_interrupt_enable		
Data Type	**Acces**	**Default**	**Unit**	**Range**
BOOLEAN	RW	0	-	[;]

Figure 1.22: Object description 0x6423.0 *ai_Global_interrupt_enable*

0x6424 **ai_Interrupt_upper_limit_32bits**

If enabled (see object 0x6423 ai_Global_interrupt_enable) an interrupt is triggered when the analog input value is equal or greater than this objects value. As long as the condition is met, every change of the analog input value generates a new interrupt. This can be avoided with aditional trigger conditions (e.g. object 0x6426 ai_Interrupt_delta_32bits).

Note: The value of this object is compared to the scaled and offseted analog input value.

Index	SubIndex	Name		
0x6424	1	Channel_0		
Data Type	**Acces**	**Default**	**Unit**	**Range**
INT32	RW	0	-	[;]

Figure 1.23: Object description 0x6424.1 *Channel_0*

0x6425 **ai_Interrupt_lower_imit_32bits**

If enabled (see object 0x6423 ai_Global_interrupt_enable) an interrupt is triggered when the analog input value is less than this objects value. As long as the condition is met, every change of the analog input value generates a new interrupt. This can be avoided with aditional trigger conditions (e.g. object 0x6426 ai_Interrupt_delta_32bits).
Note: The value of this object is compared to the scaled and offseted analog input value.

Index	SubIndex	Name		
0x6425	1	Channel_0		
Data Type	**Acces**	**Default**	**Unit**	**Range**
INT32	RW	0	-	[;]

Figure 1.24: Object description 0x6425.1 *Channel_0*

0x6426 **ai_Interrupt_delta_32bits**

This object set the delta value before sending an interrupt trigger signal (difference between the current analog input value and the last interrupts analog input value).
Note: The value of this object is compared to the scaled and offseted analog input value.

Index	**SubIndex**	**Name**		
0x6426	1	Channel_0		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RW	0	-	[;]

Figure 1.25: Object description 0x6426.1 *Channel_0*

0x6427 **ai_Interrupt_negative_delta_32bits**

This object set the negative delta value on the analog input before sending an interrupt trigger signal (falling below the last communicated value).
Note: The value of this object is compared to the scaled and offseted analog input value.

Index	**SubIndex**	**Name**		
0x6427	1	Channel_0		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RW	0	-	[;]

Figure 1.26: Object description 0x6427.1 *Channel_0*

0x6428 **ai_Interrupt_positive_delta_32bits**

This object set the positive delta value on the analog input before sending an interrupt trigger signal (rising above the last communicated value).
Note: The value of this object is compared to the scaled and offseted analog input value.

Index	**SubIndex**	**Name**		
0x6428	1	Channel_0		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RW	0	-	[;]

Figure 1.27: Object description 0x6428.1 *Channel_0*

1.1.8 Analog Outputs

The following diagram shows how the data is handled inside the board and serves as a quick reference to understand how the different objects interact:

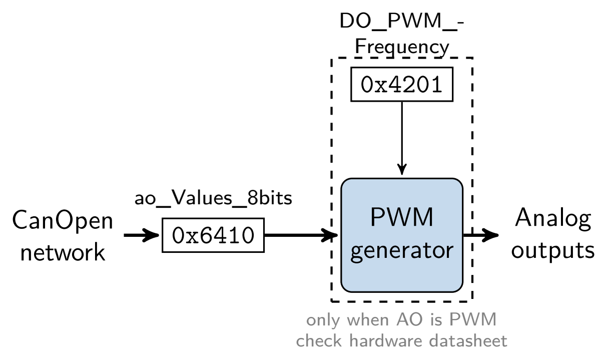


Figure 1.28: Data flow diagram for the analog outputs

0x6411 **ao_Values_16bits**

This object writes a 16 bits integer value on the output channel 'n'. The maximum outputed voltage may vary depending on your board, configuration and the input voltage. The output may be a PWM signal rather than a true analogic tension. Please refer to your board hardware datasheet.

Each analog output channel is accessed with a dedicated sub-index in this object. Channel 1 is at the sub-index 1, channel 2 is at the sub-index 2 and so on:

Index	**SubIndex**	**Name**		
0x6411	1	Channel_0		
Data Type	**Acces**	**Default**	**Unit**	**Range**
INT16	RW,RPDO	0	-	[;]

Figure 1.29: Object description 0x6411.1 *Channel_0*

1.1.9 Specifics error codes definition

Additional error codes specifictaion

In addition to the standard NMT error codes, this profile specify the following specific error codes:

Error codes	**Definition**
0x0080	Warning: analog inputs disabled
0x2310	Output current too high (overload)
0x2320	Short circuit at outputs
0x2330	Load dump at outputs
0x3110	Input voltage too high
0x3120	Input voltage too low
0x3210	Internal voltage too high
0x3220	Internal voltage too low
0x3310	Output voltage too high
0x3320	Output voltage too low

Table 1.6: Error codes

Analog inputs disabled warning

If the CANopen device transits to NMT operational state and the analog input global interrupt object (0x6423) is set to FALSE, it transmit an Emergency message with the error code 0x0080 . This Emergency message does not cause a transition into NMT pre-operational or NMT stopped state.

Chapter 2

CANOpen

2.1 CANOpen

CANOpen is based on bus CAN (Controller Area Network) which is ISO-11898 standardized. CANOpen protocol is standardized by the CAN In Automation (CIA) under the name of CiA301.

2.1.1 CAN - CANOpen

CANOpen uses standard CAN frames to communicate on the bus. The frames are identified with an Id : CAN-ID. It is coded on 11 bits. The CAN-ID with the lowest value has priority if multiple frames are sent at the same time from different devices.

The simplified CANOpen frame:

11 bit	0	1	2	3	4	5	6	7
CAN-ID	Data							

Table 2.1: Frame CANOpen

Note: A CAN message can contain a maximum of 8 bytes of usefull data.
Note: The data is always sent on the bus in **little-endian format**.

2.1.2 CANOpen Node

Each device, also called Node, is identified by a unique identifier in the network called: Node-Id. The Node-Id can take a value in the range [1-127].

All descriptions and functionalities of a Node are described in the EDS (Electronic Data Sheet) file.

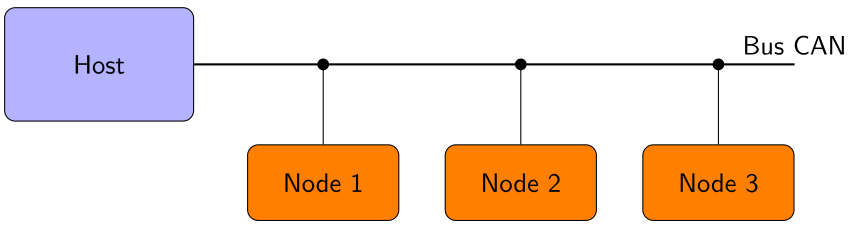


Table 2.2: Bus CAN

2.1.3 Communication

The CanOpen defines three communication protocol sequences :

- Master/Slave protocol
- Client/Server protocol
- Producer/Consumer protocol

Master/Slave protocol

This protocol works on a network where only one CanOpen master is present for a specific functionality. And therefore the other CanOpen devices are slaves. The master sends a request and the slave responds.

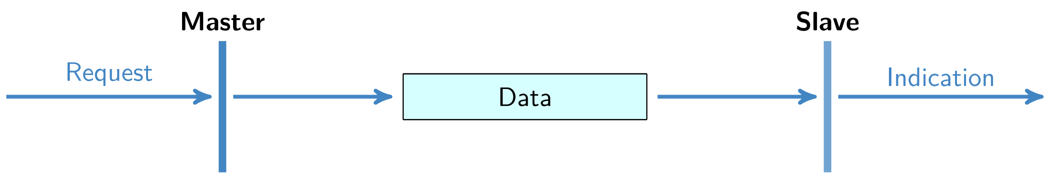


Figure 2.1: Unconfirmed Master/Slave protocol

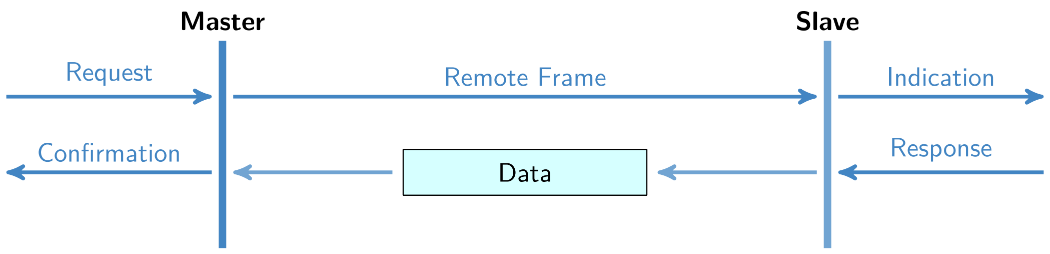


Figure 2.2: Confirmed Master/Slave protocol

Client/Server protocol

This protocol is used between a client and a server. When the client makes a request (download / upload), the server triggers the processing of the request. The server responds to the request when the task is completed.

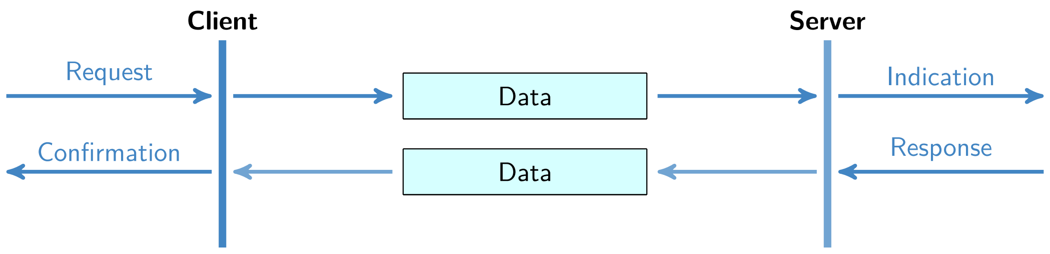


Figure 2.3: Client/Server protocol

Producer/Consumer protocol

This protocol works with a producer that sends a message that can be received by one or more devices on the network. The producer does not receive confirmation.

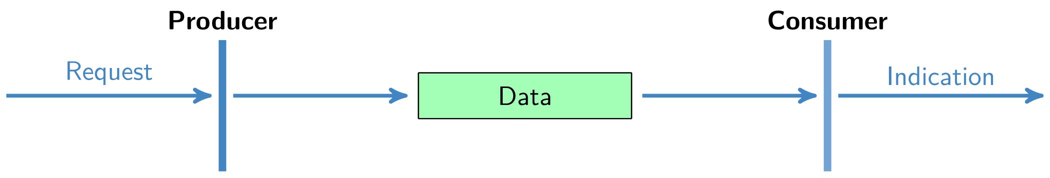


Figure 2.4: Push Producer/Consumer protocol

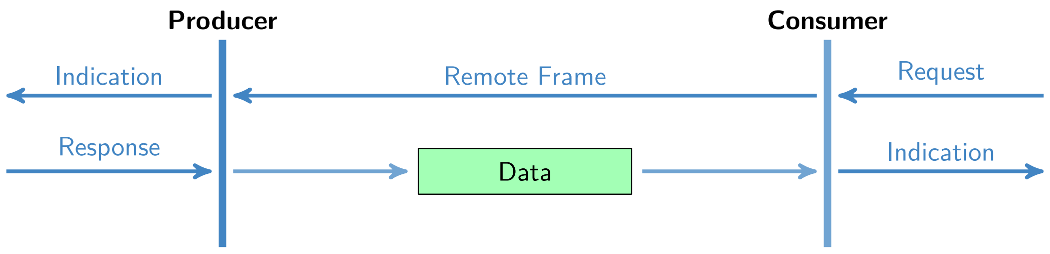


Figure 2.5: Pull Producer/Consumer protocol

2.2 CANOpen Services

The services provided by the CANOpen stack allow a standard communication between the devices on the network. These services allow communication object exchanges. There are 5 types of services:

- Network management services
  - Network management services (NMT)
  - Node guarding, network equipment monitoring
  - Heartbeat, network equipment monitoring
  - Boot-up
- Service data object (SDO), providing read and write access to the dictionary of objects
- Process data object (PDO), allowing to transmit data in real time process:
  - TPDO Transmit-PDO for use in data transmission
  - RPDO Receive-PDO for use in data reception
- SYNC, synchronization object used by PDO.
- EMCY, emergency

Each type of message is defined by the allocation of the following CAN-IDs:

Services	Default CAN-ID	CAN-ID configurable
NMT	0x000	
SYNC	0x080	yes
EMCY	0x080 + Node-Id	yes
TPDO1	0x180 + Node-Id	yes
RPDO1	0x200 + Node-Id	yes
TPDO2	0x280 + Node-Id	yes
RPDO2	0x300 + Node-Id	yes
TPDO3	0x380 + Node-Id	yes
RPDO3	0x400 + Node-Id	yes
TPDO4	0x480 + Node-Id	yes
RPDO4	0x500 + Node-Id	yes
TSDO	0x580 + Node-Id	
RSDO	0x600 + Node-Id	
Boot-Up	0x700 + Node-Id	
Nodeguarding and Heartbeat	0x700 + Node-Id	

Table 2.3: Index of CAN-ID Services

Some services can have a CAN-ID configurable by a communication object:

Services	Object
SYNC	0x1005
EMCY	0x1014
TPDOX	0x180X
RPDOX	0x140X

Table 2.4: Index of CAN-ID Services

But the changing CAN-ID should not interfere with the following reserved CAN-IDs:

CAN-ID	used by COB
0x000	NMT
0x00 0x07F	reserved
0x101 0x180	reserved
0x581 0x5FF	default SDO (tx)
0x601 0x67F	default SDO (rx)
0x6E0 0x6FF	reserved
0x701 0x77F	NMT Error Control
0x780 0x7FF	reserved

Table 2.5: Restricted CAN-ID

2.3 Object dictionary

The object dictionary is a collection of all the data items which have an influence on the behavior of the application objects, the communication objects and the state machine used on the device. Each device on the network has its own object dictionary.

The object dictionary is divided into several areas:

Index range	Description
0x0000	Reserved
0x0001 to 0x025F	Data types
0x0260 to 0x0FFF	Reserved
0x1000 to 0x1FFF	Communication profile area
0x2000 to 0x5FFF	Manufacturer-specific profile area
0x6000 to 0x9FFF	Standardized profile area

Table 2.6: Object dictionary area

The **Communication profile area** contain the communication specific parameters. These objects are common to all CANopen devices.

The **Standardized profile area** contain all data objects common to a profiles of CANopen devices that may be read or written via the network. The objects from 6000 h to 9FFF h describe parameters and functionality.

The **Manufacturer-specific** profile area contains the objects for specific UniSwarm features.

2.3.1 Description of the object dictionary

The objects of the dictionary are described by several parameters. This description is materialized by an EDS file: Electronic Data Sheet. ASCII format respecting a strict syntax that can be used by the bus configuration software.

Index and sub-index

These form the unique identifier of an object in the objects dictionary in hexadecimal notation.

Object code

The object code denotes what kind of object is at a particular index within the objects dictionary.
They can be one of the following:

Object name	Description	Codage
NULL	An object with no data fields	0x00
DOMAIN	A large variable amount of data	0x02
DEFTYPE	A type definition for simple data type such as a Boolean, Unsigned16	0x05
DEFSTRUCT	Defines a new record type	0x06
VAR	A single value	0x07
ARRAY	A data area in which each entry is of the same data type.	0x08
RECORD	A data area that contains entries that are a combination of simple data types.	0x09

Table 2.7: Object code

Data type

The data type information indicates the data type of the object.

Index	Name	Size in byte
0x0001	Boolean	1
0x0002	Integer8	1
0x0003	Integer16	2
0x0004	Integer32	4
0x0005	Unsigned8	1
0x0006	Unsigned16	2
0x0007	Unsigned32	4
0x0008	Real32	4
0x0009	VISIBLE STRING	...
0x000A	OCTET STRING	...
0x000B	UNICODE STRING	...
0x000C	TIME OF DAY	..
0x000D	TIME DIFFERENCE	...
0x000F	Domain	...
0x0010	Integer24	3
0x0011	Real64	8
0x0012	Integer40	5
0x0013	Integer48	6
0x0014	Integer56	7
0x0015	Integer64	8
0x0016	Unsigned24	3
0x0018	Unsigned40	5
0x0019	Unsigned48	6
0x001A	Unsigned56	7
0x001B	Unsigned64	8
0x0020	PDO COMMUNICATION PARAMETER	...
0x0021	PDO MAPPING	...
0x0022	SDO PARAMETER	...
0x0023	IDENTITY	...

Table 2.8: Data type

Note: Data type with indices from 0x0001 to 0x0007, 0x0010, from 0x0012 to 0x0016 , and from 0x0018 to 0x001B may be mapped in order to define the appropriate space in the RPDO.
Note: Data type with indices from 0x0008 to 0x000F, 0x0011, from 0x0020 to 0x0023 shall not be mapped into RPDOs

Access usage

Acces object:

- rw: read and write access
- wo: write only access
- ro: read only access
- const: read only access, value is constant

In addition, there are the access attributes for the PDOs:

- rww: read and write access and can be mapped on RPDO
- rwr: read and write access and can be mapped on TPDO

2.4 Network management services (NMT)

Network management (NMT) follows a master-slave structure. All devices are NMT slaves but the network must have a device master (device master, computer or other).

The service provides a tool to initiate, start, monitor, reset or stop the devices. Monitoring is made with the Node guarding and Heartbeat functionalities.

2.4.1 NMT Network management

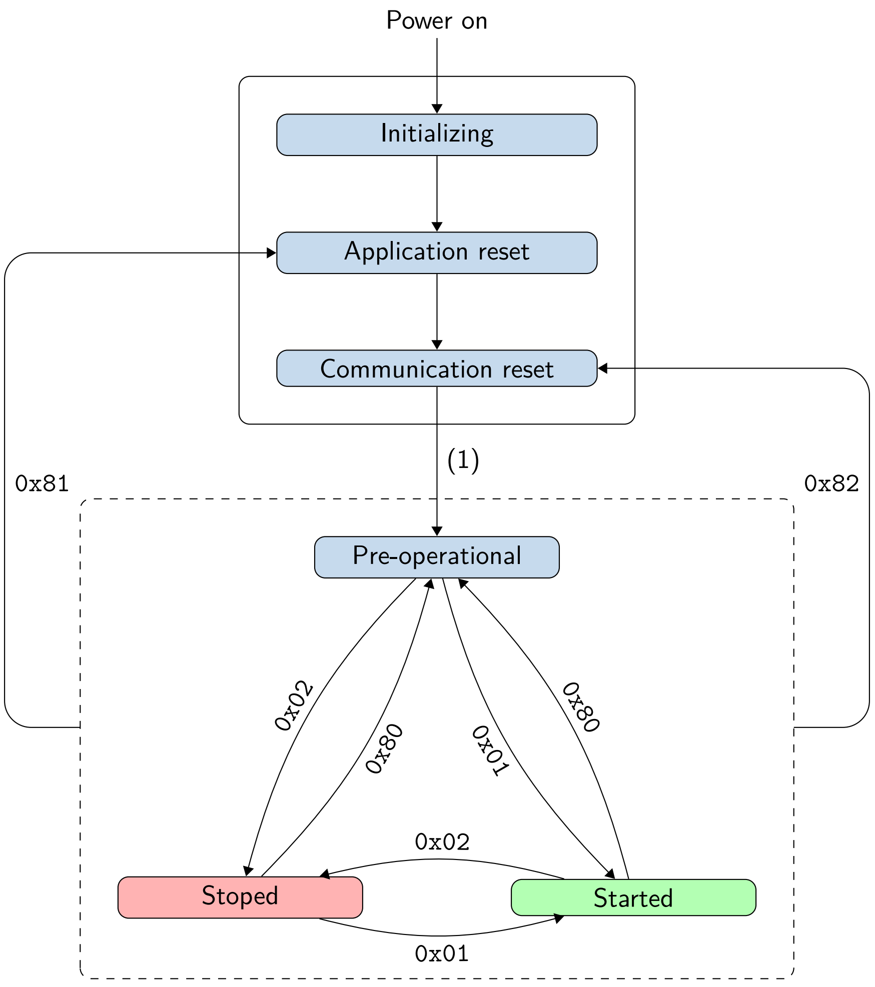
The NMT master controls the state of each NMT slave. The state can be chosen among the following ones: Stopped, Pre-operational, Started, Initialization.

NMT state machine

The NMT state machine determines the behavior of the communication function unit.

- state **Initialization**
  - **Initializing**: the device enters this state after a power-on or an hardware/software reset.
  - **Application reset**: The object dictionnary in Manufacturer-specific profile area, index range `0x2000` to `0xFFFF` is reset.
  - **Communication reset**: The object dictionnary in Communication profile area, index range `0x1000` to `0x1FFF` is reset.
- state **Pre-operational**
- state **Started**
- state **Stopped**

The following figure show the state diagram:



(1) Automatic switch to Pre-operational and send Boot-up message

Figure 2.6: Different status of a CAN Open node

NMT frame

The NMT frame allow to change the state of device. An NMT frame have the CAN-ID `0x000` and a payload of two bytes. The first one is the mode and the second one the node id.

CAN-ID	Byte[0]	Byte[1]
0x000	Mode	Node-Id

Figure 2.7: NMT frame

Mode value	Description
0x01	Start node
0x02	Stop node
0x80	Enter in Pre-operational mode
0x81	Application reset
0x82	Communications reset

Figure 2.8: MODE values in NMT frame

NMT States and Services

Authorized services according to the NMT state:

	Pre-operational	Started	Stopped
PDO		X	
SDO	X	X	
SYNC	X	X	
EMCY	X	X	
Node guarding	X	X	X
Heartbeat	X	X	X

Figure 2.9: NMT States and authorized Services

2.4.2 Node Guarding - Heartbeat

Two services are available to detect an error on the CAN network: the **Node guarding** service and the **Heartbeat** service.

- **Node guarding**: the master sends a message periodically and each slave has to respond within a time limit.
- **Heartbeat**: each slave sends a message with his state without prior request from the master.

Note: The **Heartbeat** service has priority over the **Node guarding** service. Activation of the **Heartbeat** service results in the deactivation of the **Node Guarding** service.

Node guarding

This service monitors the status of devices on the bus and makes it possible to detect remote errors on the network.
The master and the slave monitor each other: the master cyclically requests the NMT status of the slave. In each response from the slave, the Toggle-bit (bit 7) is toggled.

Two monitoring functions are implemented:

- **Node guarding**: The master can react accordingly if these responses are not sent or if the slave always responds with the same bit Toggle.
- **Life guarding**: The slave monitors the reception of messages from the master, if the message is not sent within the allotted time, the Life Time, the slave triggers an EMCY message (with ode 0x8130) see table 2.11.
  The Life time : *Life Time = 0x100C Guard Time x 0x100D Life Time Factor*.

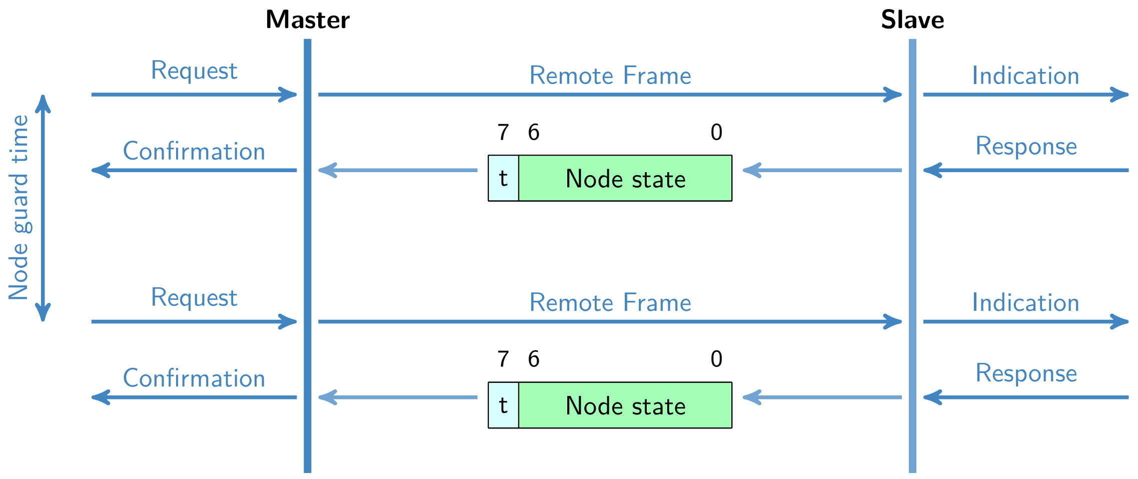
Note: This service is activated by setting value in object `0x100C` Guard Time and `0x100D` Life Time Factor other than zero.

Node guarding frame The node guarding service follows a master / slave architecture: the NMT master sends an RTR (Remote Transmission Request) message with CAN-ID 700 + node Id to a slave and the slave responds with an 8-bit message.

The response message is built with a Toggle-bit (bit 7) and the current NMT state of the slave in bits 6 to 0. For the first response and after a NMT reset, the toggle bit should be 0.

CAN-ID	Byte[0]
0x700 + Node-Id	0x00

Figure 2.10: Node guarding frame



- t: Toggle Bit
- Node state:
  - 4: Stopped
  - 5: Started
  - 27: Pre-operational

Figure 2.11: Node guarding protocol

Heartbeat

The Heartbeat works in Producer/Consumer mode with one producer and 0 minimum consumer.

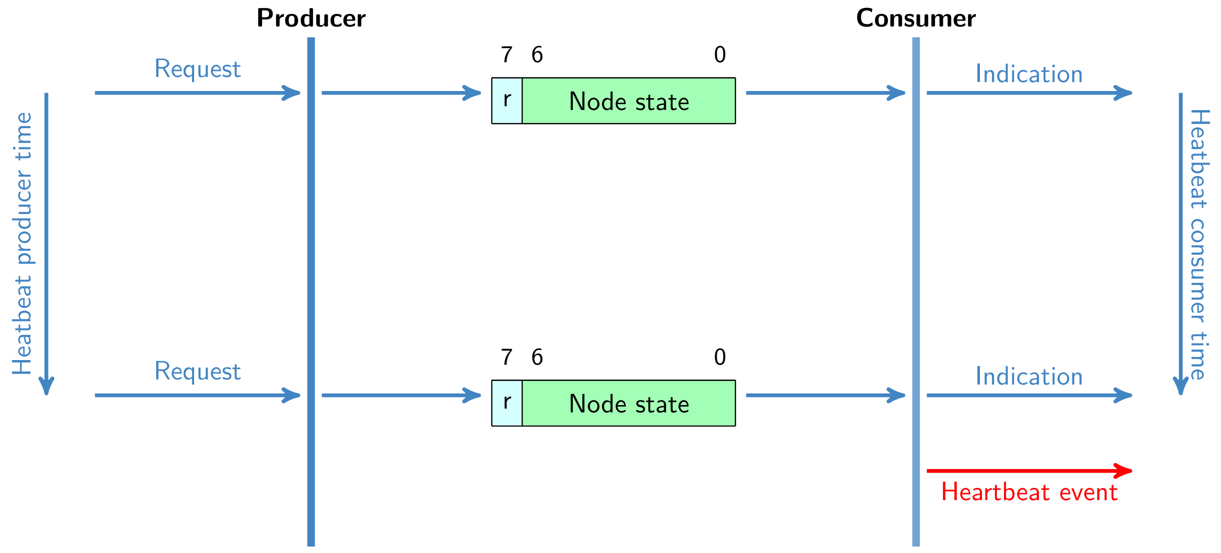
The producer heartbeat send a heartbeat message periodically, with the time between two messages defined by the "Producer heartbeat time" object.

The consumer check if it received a message in the time defined by the object "Consumer heartbeat time".

Note: This service is activated by setting the producer heartbeat time object in object 0x1017 Producer Heartbeat Time to a value other than zero.

CAN-ID	Byte[0]
0x700 + Node-Id	0x00

Figure 2.12: Heartbeat frame



- r: reserved (always 0)
- Node state:
  - 0: Boot-Up
  - 4: Stopped
  - 5: Operational
  - 127: Pre-operational

Figure 2.13: Heartbeat protocol

2.4.3 Boot-up

After power-up, the slave sends a Boot-up message to indicate that the Initializing phase is complete.

CAN-ID	Byte[0]	
0x700 + Node-Id	0x00	

Figure 2.14: Boot-up frame

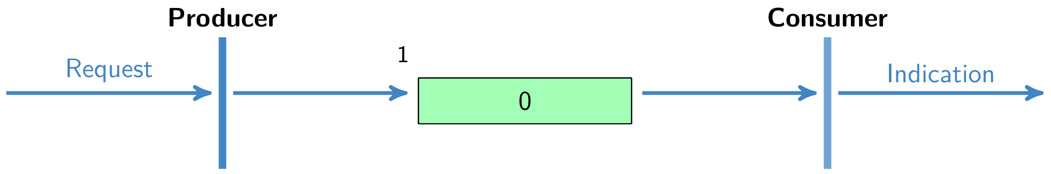


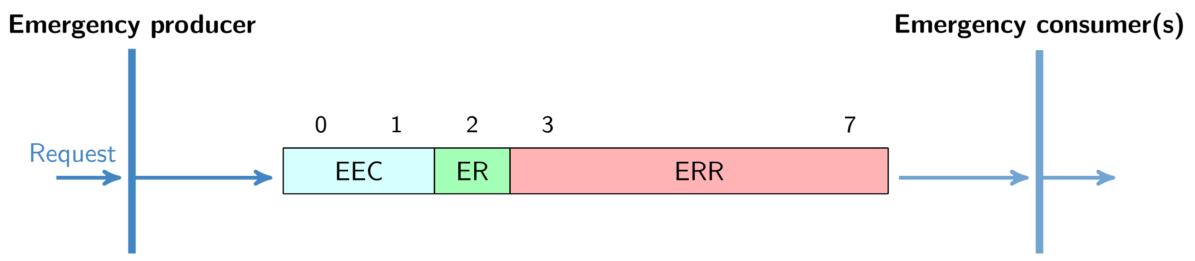
Figure 2.15: Boot-up protocol

2.5 EMCY

The EMCY service is used to transmit application faults associated with each station. When a fault is detected the device send a EMCY message with Emergency error code, Error register and error code (optional).

CAN-ID	Byte[0]	
0x080 + Node-Id	...	

Figure 2.16: EMCY frame



- EEC: Emergency error code
- ER: Error register (see object `0x1001` *Error Register*)
- ERR: Manufacturer-specific error code (Optional)

Figure 2.17: Emergency protocol with Byte[0]

EEC	Description
0x8130	Life guard error or heartbeat error

Table 2.9: Emergency error codes

2.5.1 Definition of parameters

`0x1001` Error Register

Index	SubIndex	Name		
0x1001	0	Error Register		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT8	RO,TPDO	-	-	[;]

Figure 2.18: Object description `0x1001.0` *Error Register*

Bit	Description
0	Generic error
1	Current
2	Voltage
3	Temperature
4	Communication error (overrun, error state)
5	Device profile specific
6	reserved (always 0 b)
7	manufacturer-specific

Table 2.10: Error code register of object `0x1001`

`0x1029` Error Behaviour

Sets the behavior in the event of a serious device failure in the Operational NMT state. By default, the device automatically enter the Pre-operational NMT state.

Failures include the following communication errors:

- CAN interface bus stop conditions
- NodeGuarding "time out"
- Heartbeat "time out"
- Serious errors can also be caused by internal device failures.

`0x1029.1` ***Communication Error***

Index	**SubIndex**	**Name**		
0x1029	1	Communication Error		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT8	RW	0	-	[0;2]

Figure 2.19: Object description `0x1029.1` *Communication Error*

Bit	**Description**
0	Change to NMT state Pre-operational (default)
1	No change of the NMT state
2	Change to NMT state Stopped

Table 2.11: Error class values of `0x0x1029.1`

2.6 Service data object (SDO)

This service provides access to the device object dictionary by an Index and Sub-index without time constraints in writing or reading.

Each network device is an SDO server, the one that holds the OD. The client refers to the node requesting to read or write an object value in the server's object dictionary

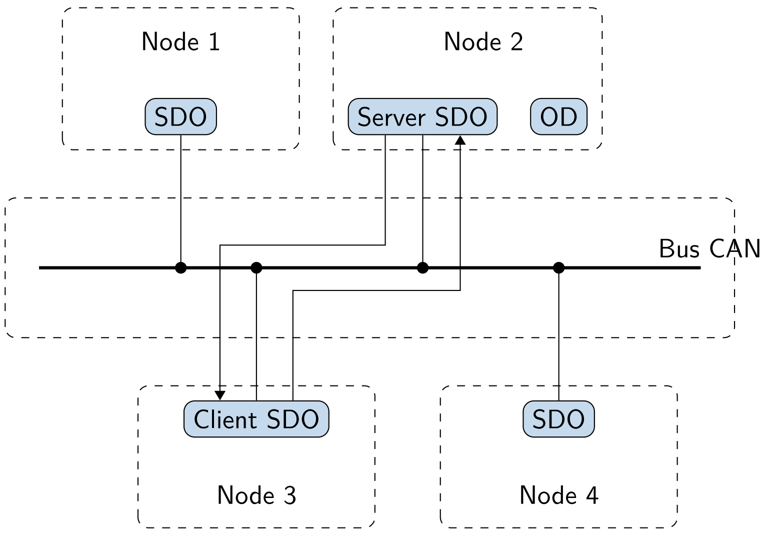
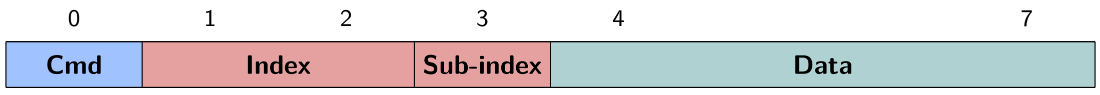


Table 2.12: Server/Client SDO

2.6.1 SDO message

CAN-ID	**Byte[0-7]**
CAN-ID+ Node-Id	...

Figure 2.20: SDO frame



- Cmd: Command
- Index: Index of object
- Sub-Index:

Figure 2.21: Expedited SDO Upload detail of frame

2.6.2 Expedited Transfer

This mode of communication is used to write or read data in object. The size of the object must be inferior or equal to 4 bytes. An answer is expected after each request, either with data, with a confirmation or with an error message.

SDO Reading

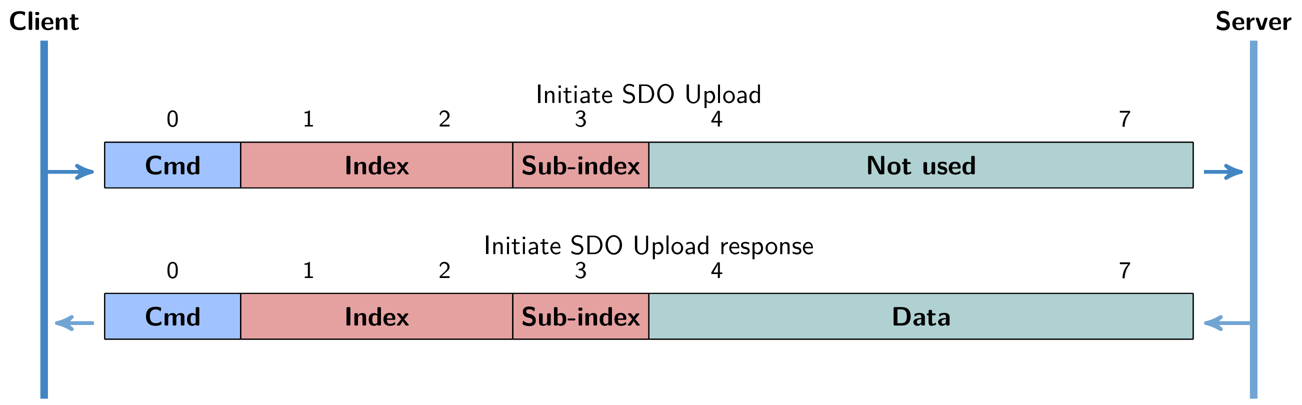


Figure 2.22: Expedited SDO Upload protocol

Cmd	Description
0x40	Upload/Reading request
0x4F	Upload/Reading response for data of size of 1 byte
0x4B	Upload/Reading response for data of size of 2 byte
0x47	Upload/Reading response for data of size of 3 byte
0x43	Upload/Reading response for data of size of 4 byte
0x2F	Download/Writing for data of size of 1 byte
0x2B	Download/Writing for data of size of 2 byte
0x27	Download/Writing for data of size of 3 byte
0x23	Download/Writing for data of size of 4 byte
0x60	Download/Writing response

Figure 2.23: List of Cmd

Initiate SDO Upload/Reading request

0	1	2	3	4			7
Cmd	**Index**		**Sub-index**	**Data**			
0x40	lsb	msb	Sub-index	0	0	0	0

Initiate SDO Upload/Reading response

Read response for a data size of 1 byte:

0	1	2	3	4			7
Cmd	**Index**		**Sub-index**	**Data**			
0x4F	lsb	msb	Sub-index	data	0	0	0

Read response for a data size of 2 byte:

0	1	2	3	4			7
Cmd	**Index**		**Sub-index**	**Data**			
0x4B	lsb	msb	Sub-index	lsb	msb	0	0

Read response for a data size of 3 byte:

0	1	2	3	4			7
Cmd	**Index**		**Sub-index**	**Data**			
0x47	lsb	msb	Sub-index	lsb	···	msb	0

Read response for a data size of 4 byte:

0	1	2	3	4			7
Cmd	**Index**		**Sub-index**	**Data**			
0x43	lsb	msb	Sub-index	lsb	···	···	msb

SDO Download/Writing

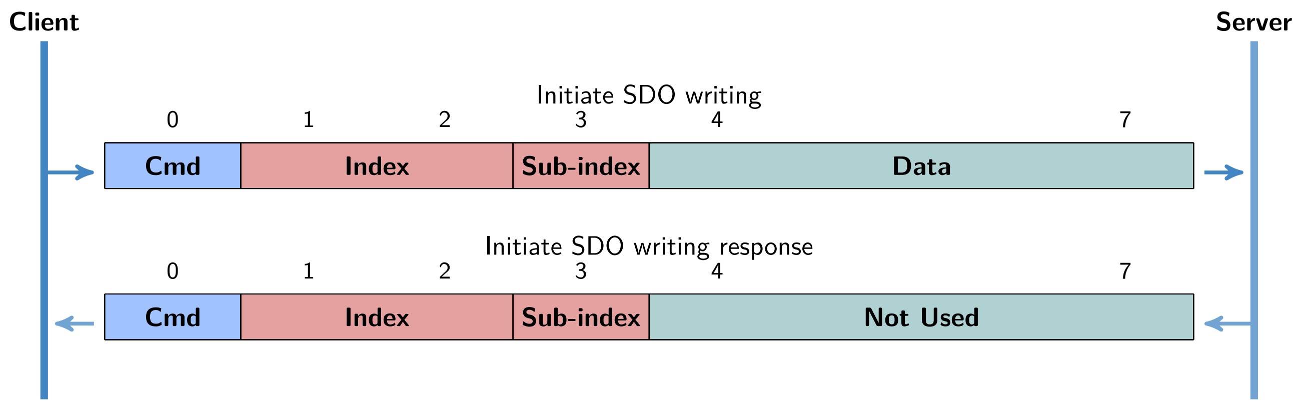


Figure 2.24: Expedited SDO Download protocol

Initiate SDO Download/Writing

Write request for data of size of 1 byte:

0	1	2	3	4			7
Cmd	**Index**		**Sub-index**	**Data**			
0x2F	lsb	msb	Sub-index	data	0	0	0

Write request for data of size of 2 byte:

0	1	2	3	4			7
Cmd	**Index**		**Sub-index**	**Data**			
0x2B	lsb	msb	Sub-index	lsb	msb	0	0

Write request for data of size of 3 byte:

0	1	2	3	4			7
Cmd	**Index**		**Sub-index**	**Data**			
0x27	lsb	msb	Sub-index	lsb	···	msb	0

Write request for data of size of 4 byte:

0	1	2	3	4			7
Cmd	**Index**		**Sub-index**	**Data**			
0x23	lsb	msb	Sub-index	lsb	···	···	msb

Initiate SDO Download/Writing response

0	1	2	3	4			7
Cmd	**Index**		**Sub-index**	**Data**			
0x60	lsb	msb	Sub-index	0	0	0	0

2.6.3 SDO abort transfer

Error response:

0	1	2	3	4			7
Cmd	**Index**		**Sub-index**	**SDO abort codes**			
0x80	lsb	msb	Sub-index	lsb	···	···	msb

2.6.4 SDO abort codes

Error codes	Description
0x05030000	Toggle bit not alternated
0x05040000	SDO protocol timed out
0x05040001	Client/server command specifier not valid or unknown
0x05040002	Invalid block size (block mode only)
0x05040003	Invalid sequence number (block mode only)
0x05040004	CRC error (block mode only)
0x05040005	Out of memory
0x06010000	Unsupported access to an object
0x06010001	Attempt to read a write only object
0x06010002	Attempt to write a read only object
0x06020000	Object does not exist in the object dictionary
0x06040041	Object cannot be mapped to the PDO
0x06040042	The number and length of the objects to be mapped would exceed PDO length
0x06040043	General parameter incompatibility reason
0x06040047	General internal incompatibility in the device
0x06060000	Access failed due to an hardware error
0x06070010	Data type does not match, length of service parameter does not match
0x06070012	Data type does not match, length of service parameter too high
0x06070013	Data type does not match, length of service parameter too low
0x06090011	Sub-index does not exist
0x06090030	Invalid value for parameter (download only)
0x06090031	Value of parameter written too high (download only)
0x06090032	Value of parameter written too low (download only)
0x06090036	Maximum value is less than minimum value
0x060A0023	Resource not available: SDO connection
0x08000000	General error
0x08000020	Data cannot be transferred or stored to the application
0x08000021	Data cannot be transferred or stored to the application because of local control
0x08000022	Data cannot be transferred or stored to the application because of the present device state
0x08000023	Object dictionary dynamic generation fails or no object dictionary is present
0x08000024	No data available

Table 2.13: SDO error codes

2.7 Process data object (PDO)

The purpose of the "Process Data Objects (PDO)" is to provide a transfer of data in real time during the operation of the controller. This service is performed without protocol overload or confirmation. The size of a PDO frame is variable and depends on the size of the object.

The PDO provides an interface to the application objects in the object dictionary. Data type and mapping of objects is determined by the corresponding PDO mapping structure in the object dictionary.

The PDO configuration process (PDO Mapping) allows to configure the number of objects in a PDO. This process uses the SDO service.
There are two types of PDO, the Transmit-PDO (TPDO) for use in data transmission and the Receive-PDO (RPDO) for use in data reception. The data transmission via the PDO service operates according to a producer / consumer relationship: RPDOs are frames received from the master or others nodes. TPDOs are frames transmitted to others.

PDOs are described by the PDO communication parameter and the PDO mapping parameter.

Note: A node can have a maximum of four TPDOs and four RPDOs.

There is an index couple for each PDO: the columns "Index RPDO communication" and "Index RPDO mapping" provide the indexes of specials objects used to read or modify the parameters of communication objects via an SDO object:

RPDO	**CAN-ID**	**Object Index**	
		Index RPDO communication	Index RPDO mapping
RPDO1	0x200 + Node-Id	0x1400	0x1600
RPDO2	0x300 + Node-Id	0x1401	0x1601
RPDO3	0x400 + Node-Id	0x1402	0x1602
RPDO4	0x500 + Node-Id	0x1403	0x1603

Table 2.14: Index used of RPDOs

TPDO	**CAN-ID**	**Object Index**	
		Index TPDO communication	Index TPDO mapping
TPDO1	0x180 + Node-Id	0x1800	0x1A00
TPDO2	0x280 + Node-Id	0x1801	0x1A01
TPDO3	0x380 + Node-Id	0x1802	0x1A02
TPDO4	0x480 + Node-Id	0x1803	0x1A03

Table 2.15: Index used of TPDOs

2.7.1 PDO message

There is two ways to transmit a PDO message:

- Synchronous transmission: the object are synchronized on SYNC.
- Event-driven transmission

This PDO is only activated if the status of CanOpen is "Started". It is necessary to activate the PDOs, for that the "valid" bit of CAN-ID must be to set to 0x0.

Example:
to activate or deactivate the TPDO1, the object with the index 0x1800 and sub-index 0x1 is used:

- Deactivate TPDO1: set sub-index 1 to 0x80000181
- Activate TPDO1: set sub-index 1 to 0x00000181

The value 0x181 is the id of TPDO1.

After each SYNC, two things happen in this order:

- for TPDOs: the slaves samples and copy the data into TPDOs which are then sent on the bus.
- for RPDOs: the previous received RPDO data from master is copied in the objects database and made available to the application.

2.7.2 SYNC

The master in a CAN Open network sends a unique sync frame for all the nodes. At reception, the nodes transmit theirs TPDOs and apply their RPDOs. A SYNC frame has the ID `0x080` and does not contain a payload.

CAN-ID
0x080

Figure 2.25: SYNC frame

The frequency of SYNC frame can be variable, but is always sent by the master.

2.7.3 PDO dynamic mapping

General procedure:

The following procedure shall be used for re-mapping, which may take place during the NMT state Pre-operational and during the NMT state Operational, if supported:

1. Deactivate PDO: setting the valid bit to `0x1` of sub-index `0x01` of the PDO communication parameter.
2. Disable mapping: setting sub-index `0x00` to 0.
3. Modify mapping: changing the values of the corresponding sub-indexes.
4. Enable mapping: setting sub-index `0x00` to the number of mapped objects.
5. Activate PDO: setting the valid bit to 0 of sub-index `0x01` of the PDO communication parameter.

2.7.4 PDO parameter objects

`0x140n` RPDO Parameter X

Note: 'n' : common instance of index (0,1,2,3) and 'X' : number of RPDO (1,2,3,4)
Objects involved : `0x1400` *RPDO Parameter 1*, `0x1401.0`, `0x1402` *RPDO Parameter 3*, `0x1403` *RPDO Parameter 4*

The PDO parameter describes the communication abilities of the PDO.

Index	Sub-index	Name	Data type
RPDO: 0x1400 to 0x1403	0x00	Highest sub-index supported	Unsigned8
	0x01	COB ID	Unsigned32
	0x02	Transmission type	Unsigned8
	0x03	Inhibit time	Unsigned16
	0x04	reserved	Unsigned8
	0x05	Event timer	Unsigned16

Table 2.16: PDO config objects

`0x140n.1` *COB ID*

Index	SubIndex	Name		
0x140n	1	COB ID		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RW	512	-	[0x00000080;0xFFFFFFFF]

Figure 2.26: Object description `0x140n.1` *COB ID*

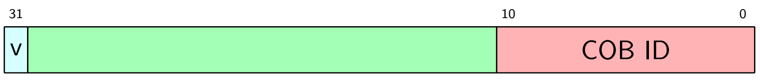


Table 2.17: COB ID

- **v**: valid

Value	Description
0x00	PDO exists / is valid
0x01	PDO does not exist / is not valid

Table 2.18: Description of bit 31: v

- **COB ID** of PDO

`0x140n.2` ***Transmission Type***

Index	SubIndex	Name		
0x140n	2	Transmission Type		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT8	RW	0	-	[;]

Figure 2.27: Object description `0x140n.2` *Transmission Type*

The transmission of the PDO depends on the configuration of the Transmission Types parameters which can be:

Value	Description
0x00 to 0xF0	Synchronous
0xFE	Event

Table 2.19: Description of RPDO transmission type

- **Synchronous**: the synchronous transmission type means that the data is transmitted immediately, but it is applied when SYNC is received. The SYNC service provides a data synchronization signal over the network.
- **Event**: the Event-driven transmission type means that the PDO may be received at any time and that the data is applied immediately after reception.

`0x160n` RPDO Mapping X

Note: 'n' : common instance of index (0,1,2,3) and 'X' : number of RPDO (1,2,3,4)
Objects involved : `0x1600` *RPDO Mapping 1*, `0x1601` *RPDO Mapping 2*, `0x1602` *RPDO Mapping 3*, `0x1603` *RPDO Mapping 4*

The PDO mapping parameter contains informations about the content of the PDO.

	Sub-index	Name	Data type
0x1600 to 0x1603	00	Number of mapped objects in PDO	Unsigned8
	01	1st object to be mapped	Unsigned32
	02	2 nd object to be mapped	Unsigned32
	...	...	Unsigned32
	64	64 th object to be mapped	Unsigned32

Figure 2.28: PDO mapping parameter

- **Sub-index 0x00**: the number of valid object entries within the mapping record. If it is equal to 0, the Mapping is disabled.
- **Sub-index 0x01 to 0x40**: the information of the mapped application objects. The object describes the content of the PDO by the index, sub-index and length of the mapped object.

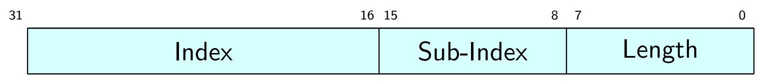


Figure 2.29: Structure of PDO mapping

Note: The MSB is first.
Note: The length is the length of object in bits.

`0x160n` ***RPDO Mapping X***

Index	SubIndex	Name		
0x160n	0	RPDO Mapping X		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT8	RW	2	-	[0;8]

Figure 2.30: Object description `0x160n.0` *RPDO Mapping X*

`0x180n` **TPDO Parameter X**

Note: 'n' : common instance of index (0,1,2,3) and 'X' : number of RPDO (1,2,3,4)
Objects involved : `0x1800` *TPDO Parameter 1*, `0x1801` *TPDO Parameter 2*, `0x1802` *TPDO Parameter 3*, `0x1803` *TPDO Parameter 4*

The PDO parameter describes the communication abilities of the PDO.

Index	Sub-index	Name	Data type
TPDO: 0x1800 to 0x1803	0x00	Highest sub-index supported	Unsigned8
	0x01	COB ID	Unsigned32
	0x02	Transmission type	Unsigned8
	0x03	Inhibit time	Unsigned16
	0x04	reserved	Unsigned8
	0x05	Event timer	Unsigned16

Table 2.20: PDO config objects

Index	SubIndex	Name		
0x180n	1	COB ID		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RW	384	-	[0x00000080;0xFFFFFFFF]

Figure 2.31: Object description `0x180n.1` *COB ID*

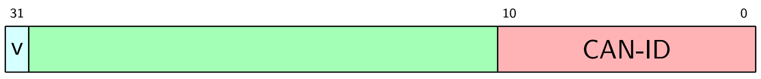


Table 2.21: COB ID

- **v**: valid

Value	Description
0x00	PDO exists / is valid
0x01	PDO does not exist / is not valid

Table 2.22: Description of bit 31: v

- **COB ID** of PDO

`0x180n.2` ***Transmission Type***

Index	SubIndex	Name		
0x180n	2	Transmission Type		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT8	RW	1	-	[;]

Figure 2.32: Object description `0x180n.2` *Transmission Type*

The transmission of the PDO depends on the configuration of the Transmission Types parameters which can be:

Value	Description
0x00	Synchronous (acyclic)
0x01 to 0xF0	Synchronous cyclic every N SYNC
0xFC	RTR-only (synchronous)
0xFD	RTR-only (event-driven)
0xFE	Event-driven

Table 2.23: Description of TPDO transmission type

- **Synchronous acyclic**: On internal event, the sampling will start and will transmit after the next SYNC.
- **Synchronous**: the TPDO is transmitted after each SYNC received. The sampling of the data will start and will transmit on reception of each the SYNC received.
- **RTR-only (synchronous)**: On RTR (Remote Transmission Request) received, the sampling will start and will transmit after the next SYNC.
- **RTR-only (event-driven)**: On RTR (Remote Transmission Request) received, the sampling start and transmit immediately.
- **Event-driven**: sampling may be transmitted at any time when the internal event happen.

`0x180n.3` ***Inhibit Time*** It's a minimum interval time for PDO transmission. This parameter is available only for the transmission type `0xFE` and `0xFF`. The value is defined as multiple of 100 μs. The value of 0 shall disable the inhibit time. The value shall not be changed while the PDO exists.

Index	SubIndex	Name		
0x180n	3	Inhibit Time		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT16	RW	0	-	[;]

Figure 2.33: Object description `0x180n.3` *Inhibit Time*

`0x180n.6` ***SYNC start value*** The SYNC message of which the counter value equals the SYNC Start value is be regarded as the first received SYNC message. The value of 0 shall disable the SYNC Start value.

Index	SubIndex	Name		
0x180n	6	SYNC start value		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT8	RW	0	-	[;]

Figure 2.34: Object description `0x180n.6` *SYNC start value*

`0x1A0n` TPDO Mapping X

Note: 'n' : common instance of index (0,1,2,3) and 'X' : number of RPDO (1,2,3,4)
Objects involved : `0x1A00` *TPDO Mapping 1*, `0x1A01` *TPDO Mapping 2*, `0x1A02` *TPDO Mapping 3*, `0x1A03` *TPDO Mapping 4*

The PDO mapping parameter contains information about the content of the PDO.

	Sub-index	Name	Data type
0x1A00 to 0x1A03	00	Number of mapped objects in PDO	Unsigned8
	01	1st object to be mapped	Unsigned32
	02	2 nd object to be mapped	Unsigned32
	...	...	Unsigned32
	64	64 th object to be mapped	Unsigned32

Figure 2.35: PDO mapping parameter

- **Sub-index** `0x00`: the number of valid object entries within the mapping record. If it is equal to 0, the Mapping is disabled.

- **Sub-index** 0x01 to 0x40: the information of the mapped application objects. The object describes the content of the PDO by the index, sub-index and length of the mapped object.

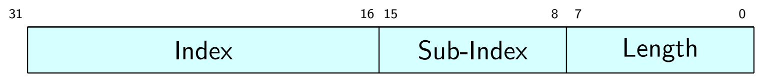


Figure 2.36: Structure of PDO mapping

Note: The MSB is first.
Note: The length is the length of object in bits.

Index	SubIndex	Name		
0x1A0n	0	TPDO Mapping X		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT8	RW	2	-	[0;8]

Figure 2.37: Object description 0x1A0n.0 *TPDO Mapping X*

2.8 Object description

2.8.1 Communication Profile Area object

0x1000 Device Type

This object provide informations about the device type.

Index	SubIndex	Name		
0x1000	0	Device Type		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RO	402	-	[;]

Figure 2.38: Object description 0x1000.0 *Device Type*

It is composed of two field, the device profile and additional information, both on 16 bits.

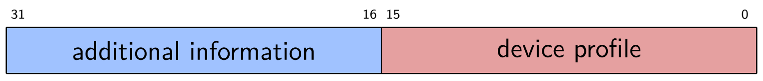


Table 2.24: Frame of Device Type

Device profile		Additional information	
Value	Description	Value	Description
0x191	CiA 401 standard is supported		
0x192	CiA 402 standard is supported		

Table 2.25: Description of device type

0x1008 Manufacturer Device Name

Index	SubIndex	Name		
0x1008	0	Manufacturer Device Name		
Data Type	**Acces**	**Default**	**Unit**	**Range**
VSTRING	RO	UMC1BDS32 motion controller	-	[;]

Figure 2.39: Object description 0x1008.0 *Manufacturer Device Name*

0x1009 Manufacturer Hardware Version

Index	SubIndex	Name		
0x1009	0	Manufacturer Hardware Version		
Data Type	**Acces**	**Default**	**Unit**	**Range**
VSTRING	RO	v1.1.0	-	[;]

Figure 2.40: Object description 0x1009.0 *Manufacturer Hardware Version*

`0x1010` Store Parameters

This object stores the value of the parameters according to the Communication profile , Manufacturer profile and Standardized profile or all profiles. The profile areas are defined to Object dictionary.

This functionnality can only work in State PreOp if this is not the case, the device respond with the SDO abort transfer service (SDO abort code: `0x08000021`).

Signature To start backup of parameters, a specific signature is required to avoiding wrong manipulation. Specific signature is written in appropriate subindex.

The signature is "save":

3	2	1	0
e	v	a	s
0x65	0x76	0x61	0x73

Figure 2.41: Signature of store

Upon receipt of the correct signature in the appropriate subindex, the device restore the default settings and then confirm the SDO transmission (SDO download initiation response).

If an erroneous signature is written, the device refuse to store the defaults and respond with the SDO abort transfer service (SDO abort code: `0x08000020`).

Different possibility to store:

Index	SubIndex	Name		
0x1010	1	Save all Parameters		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RW	0	-	[;]

Figure 2.42: Object description `0x1010.1` *Save all Parameters*

Index	SubIndex	Name		
0x1010	2	Save Communication Parameters		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RW	0	-	[;]

Figure 2.43: Object description `0x1010.2` *Save Communication Parameters*

Index	SubIndex	Name		
0x1010	3	Save Standardized Parameters		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RW	0	-	[;]

Figure 2.44: Object description `0x1010.3` *Save Standardized Parameters*

Index	SubIndex	Name		
0x1010	4	Save Manufacturer Parameters		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RW	0	-	[;]

Figure 2.45: Object description `0x1010.4` *Save Manufacturer Parameters*

`0x1011` Restore Default Parameters

This object restores the factory or saved values of the parameters according to the Communication profile , Manufacturer profile and Standardized profile or all profiles. The profile areas are defined to Object dictionary.

This functionnality can only work in State PreOp if this is not the case, the device respond with the SDO abort transfer service (SDO abort code: `0x08000021`).

Signature To start the restoration of parameters, a specific signature is required to avoiding wrong manipulation. Specific signature is written in appropriate subindex.

The signature is "load":

3	2	1	0
d	a	o	l
0x64	0x61	0x6F	0x6C

Figure 2.46: Signature of restore

Upon receipt of the correct signature in the appropriate subindex, the device restore the default settings and then confirm the SDO transmission (SDO download initiation response).

If an erroneous signature is written, the device refuse to restore the defaults and respond with the SDO abort transfer service (SDO abort code: 0x08000020).

Automatic restore This feature determines an automatic restore after a NMT service reset node, NMT service reset communication or power cycled. This functionality is configured with a command (table below) written in the appropriate subindex.

31 … 1	0
reserved	cmd

Table 2.26: Automatic restore

Value	Description
0x00	Device don't restore settings automatically
0x01	Device restore settings automatically

Table 2.27: Description of bit 0: cmd

Different possibility to restore:

Index	SubIndex	Name		
0x1011	1	Restore all Factory Parameters		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RW	0	-	[0;1]

Figure 2.47: Object description 0x1011.1 *Restore all Factory Parameters*

Index	SubIndex	Name		
0x1011	2	Restore Factory Communication Parameters		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RW	0	-	[0;1]

Figure 2.48: Object description 0x1011.2 *Restore Factory Communication Parameters*

Index	SubIndex	Name		
0x1011	3	Restore Factory Standardized Parameters		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RW	0	-	[0;1]

Figure 2.49: Object description 0x1011.3 *Restore Factory Standardized Parameters*

Index	SubIndex	Name		
0x1011	4	Restore Factory Manufacturer Parameters		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RW	0	-	[0;1]

Figure 2.50: Object description 0x1011.4 *Restore Factory Manufacturer Parameters*

Index	SubIndex	Name		
0x1011	5	Restore all saved Parameters		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RW	1	-	[0;1]

Figure 2.51: Object description 0x1011.5 *Restore all saved Parameters*

Index	SubIndex	Name		
0x1011	6	Restore saved Communication Parameters		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RW	1	-	[0;1]

Figure 2.52: Object description 0x1011.6 *Restore saved Communication Parameters*

Index	SubIndex	Name		
0x1011	7	Restore saved Standardized Parameters		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RW	1	-	[0;1]

Figure 2.53: Object description 0x1011.7 *Restore saved Standardized Parameters*

Index	SubIndex	Name		
0x1011	8	Restore saved Manufacturer Parameters		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RW	1	-	[0;1]

Figure 2.54: Object description 0x1011.8 *Restore saved Manufacturer Parameters*

0x100A **Manufacturer Software Version**

Index	SubIndex	Name		
0x100A	0	Manufacturer Software Version		
Data Type	**Acces**	**Default**	**Unit**	**Range**
VSTRING	RO	v1.0.2	-	[;]

Figure 2.55: Object description 0x100A.0 *Manufacturer Software Version*

0x100C **Guard Time**

Index	SubIndex	Name		
0x100C	0	Guard Time		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT16	RW	0	ms	[0;65535]

Figure 2.56: Object description 0x100C.0 *Guard Time*

0x100D **Life Time Factor**

Index	SubIndex	Name		
0x100D	0	Life Time Factor		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT8	RW	0	-	[0;255]

Figure 2.57: Object description 0x100D.0 *Life Time Factor*

0x1016 **Consumer Heartbeat Time**

Index	SubIndex	Name		
0x1016	1	Consumer Heartbeat Time		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RW	0	ms	[0x0;0x007FFFFF]

Figure 2.58: Object description 0x1016.1 *Consumer Heartbeat Time*

0x1017 Producer Heartbeat Time

Index	SubIndex	Name		
0x1017	0	Producer Heartbeat Time		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT16	RW	0	ms	[0;65535]

Figure 2.59: Object description 0x1017.0 *Producer Heartbeat Time*

0x1018 Identity Object

Index	SubIndex	Name		
0x1018	1	Vendor Id		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RO	1186	-	[;]

Figure 2.60: Object description 0x1018.1 *Vendor Id*

Index	SubIndex	Name		
0x1018	2	Product Code		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RO	4097	-	[;]

Figure 2.61: Object description 0x1018.2 *Product Code*

Index	SubIndex	Name		
0x1018	3	Revision number		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RO	1	-	[;]

Figure 2.62: Object description 0x1018.3 *Revision number*

Index	SubIndex	Name		
0x1018	4	Serial number		
Data Type	**Acces**	**Default**	**Unit**	**Range**
UINT32	RO	0	-	[;]

Figure 2.63: Object description 0x1018.4 *Serial number*

Appendix A

Firmware version history

Version	Date	Change
1.0.1	2021/10/23	Initial public version

Appendix B

Datasheet revision history

Revision	Date	Change
A	2020-03-20	Initial public revision
B	2021-11-08	Reviewed structure of document