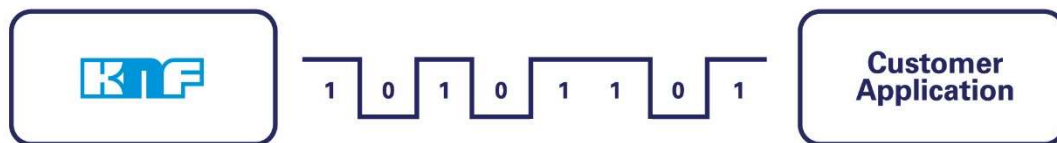


INTERFACE DESCRIPTION

KNF INTELLIGENT PUMP UART (TTL/RS232)



Index

1	Introduction.....	3
1.1	UART Specification	3
2	Communication Layers	4
2.1	HDLC (High-level-Data-Link-Control)	4
2.2	CanOpen Message	4
3	HDLC (High-level-Data-Link-Control).....	5
3.1	Start and End-Flag	5
3.2	Escaping	5
3.3	CRC	6
3.4	Sequence Handling (Invalid Frame detection)	6
4	CanOpen Message.....	7
4.1	Introduction	7
4.2	The Frame (SDO Expedited transfer)	8
4.2.1	Descriptor 1 + 2	8
4.2.2	Specifier	9
4.2.3	Index & SubIndex	9
4.2.4	Data.....	9
5	Examples.....	10
5.1	Set Motor Speed to 1000rpm.....	10
5.2	Read actual Motor Speed	13

1 Introduction

This Document describes the Communication Protocol used for Communication with KNF Intelligent devices which provide a UART (TTL/RS232) Interface.

1.1 UART Specification

Baud rate: 115200

Data Bits: 8

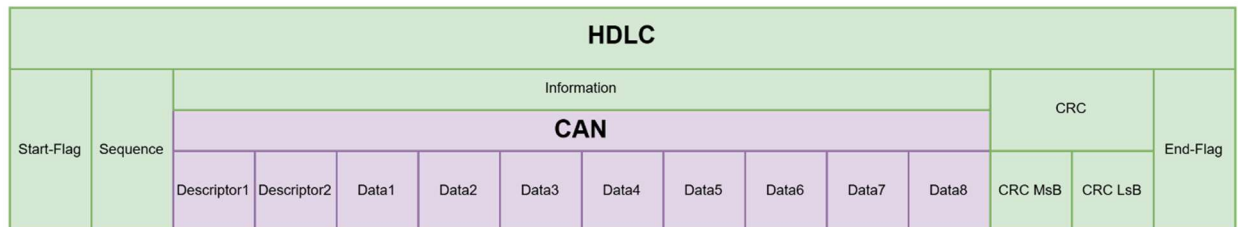
Parity: None

Stop Bits: 1

Flow Control: None

2 Communication Layers

The Communication Frame can be split into 2 Layers:
 Communication Layer which is HDLC (High-level-Data-Link-Control)
 Information Layer which is a Can Open Message.



2.1 HDLC (High-level-Data-Link-Control)

The HDLC Layer is responsible for frame handling. The main topics of the layer are:

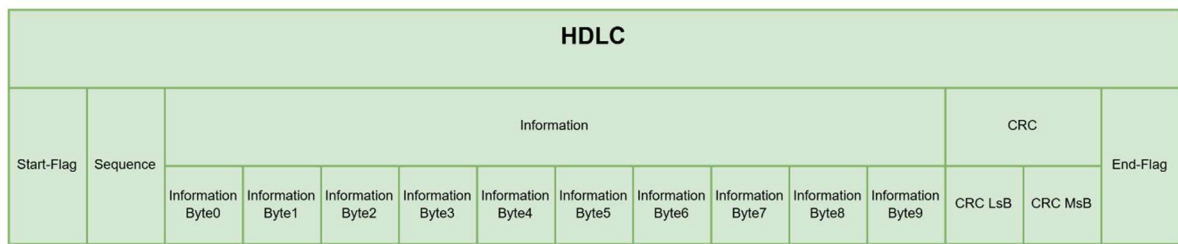
- Control Characters (Start/Endflag)
- Bytestuffing/Escaping (Escaped Characters)
- Sequence Handling
- CRC

2.2 CanOpen Message

The Information Layer represents a CanOpen Frame which contains:

- Cob ID (Message Identifier)
- Datalength
- Message Specifier
- Index
- Subindex
- Data

3 HDLC (High-level-Data-Link-Control)



3.1 Start and End-Flag

The Start and End Flag Byte is: **0x7e (~ , Tilde)**



3.2 Escaping

The Character to indicate an escape sequence is: **0x7d (}, closing brace)**

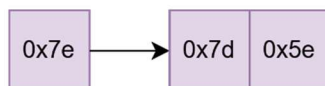


Escaping is necessary to ensure that data bytes identical to the start/end flag or the escape character can be transmitted without being misinterpreted.

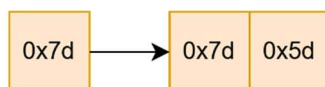
All characters from the sequence byte through the two CRC bytes may need to be escaped.

If a 0x7e or a 0x7d occurs in the original Data-Stream, it will be escaped (i.e. replaced) with two bytes

- 0x7e will be escaped to: 0x7d (0x7e XOR 0x20) = 0x7d 0x5e

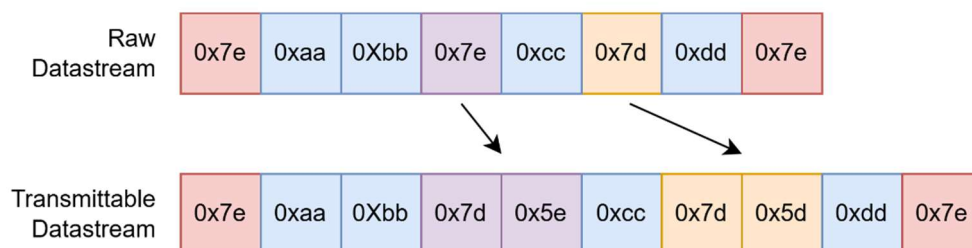


- 0x7d will be escaped to: 0x7d (0x7d XOR 0x20) = 0x7d 0x5d.



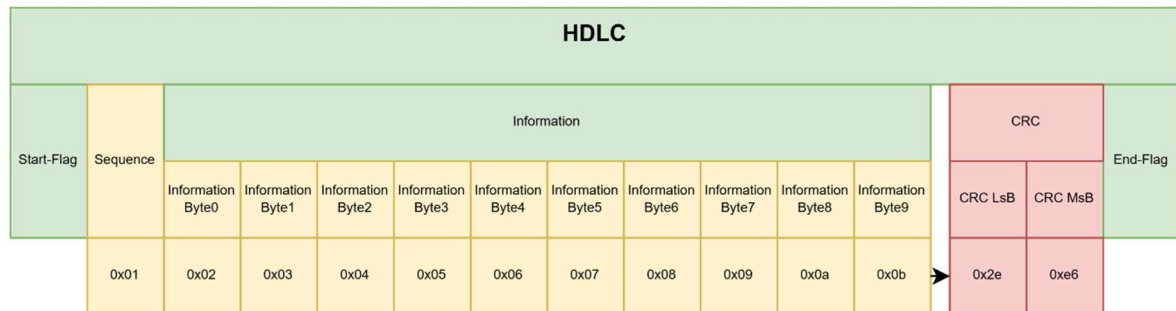
This means that 0x7E is used only to mark the beginning and end of a frame, and 0x7D is used only to show that the next byte was escaped and must be converted back.

Example:



3.3 CRC

The CRC is calculated over the whole Information data including the sequence byte



The CRC is a two Byte (16-Bit) Kermit algorithm

- Polynom: 0x1021
- [Online CRC-8 CRC-16 CRC-32 Calculator](#)

3.4 Sequence Handling (Invalid Frame detection)

The sequence number indicates the order of transmission of the frames

Allowed Sequence numbers are 1-254.

The next frame after the frame with sequence number 254 is a frame with sequence number 1.

Sequence number 0 is invalid.

A new communication sequence starts always with sequence number 1.

If the receiver detects an invalid frame, the receiver answers with a special frame.

The special frame has always sequence number 0xFF

Directly after the sequence number, the sequence number of the last correctly received frame

Special Frame			
Start-Flag	Special Sequence	Last correct received Sequence	End-Flag
0x7e	0xff	0x05	0x7e

Example:

1. Last Previous correct received frame had sequence number **0x53**
2. Next Frame is detected as Invalid from the receiver
3. Receiver answers with: 0x7e 0xFF **0x53** 0x7e
4. Server Resends frame if possible

4 CanOpen Message

4.1 Introduction

The Communication Protocol used is CANOpen. There can be found many Information about the CanOpen Protocol in the Internet.

A good explanation can be found here: <https://www.csselectronics.com/pages/canopen-tutorial-simple-intro>

For the implementation of the CAN protocol it's recommended to use a can-stack library. There are a few open-source libraries available on the internet.

C/C++: <https://canopen-stack.org/v4.4/>

Python: <https://pypi.org/project/canopen/>

CanOpen Standard Messages can have a couple of different communication objects, while the **Bold** written are supported by KNF Devices:

1. **NMT**
2. *SYNC*
3. *EMCY*
4. *TIME*
5. **PDO**
6. **SDO**
7. *Heartbeat*

In this document, we focus on SDO (Service Data Object).

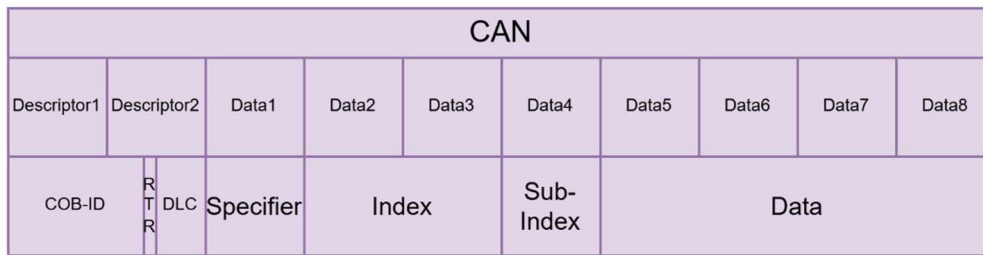
For SDOs are 3 variants for the communication:

1. **Expedited transfer**
This transfers the full payload in a single initialization/confirmation flow between the client and server, ideal for small 1-4 byte data payloads
2. **Segmented transfer**
This involves an initialization/confirmation step after which segments of up to 7 data bytes can be transferred (each segment requiring a confirmation message)
3. **Block transfer**
This is useful for large data transfers. Here, a confirmation is only required after each 'block' of messages (up to 127 segments per block), reducing the overhead

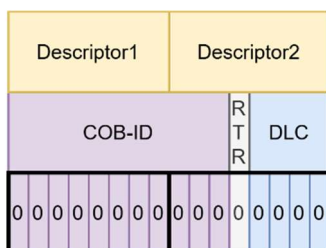
For most of the data objects, Expedited SDO transfer is sufficient.

Therefore the following documentation explains the Expedited SDO transfer.

4.2 The Frame (SDO Expedited transfer)



4.2.1 Descriptor 1 + 2



COB-ID

The 11-bit CAN ID is referred to as the Communication Object Identifier (**COB-ID**) and is split in two parts:

- **Function code:** 4 bits reflect the 'functionality' of the message
 - The Function code used for SDO Transfer: **Transmit: 0b1011 / Receive: 0b1100**
- **Node ID:** 7 bits reflect the node ID (between 1 and 127)
 - The Node Id of a KNF End-To-End Device is always **0x01**

Therefore, the COB-IDs to use for a SDO communication with a KNF device:

- **Send Request to KNF Device: 0x601**
- **Receive Answer from KNF Device: 0x581**

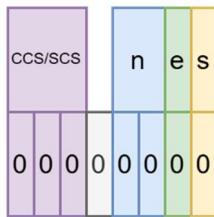
RTR

- 1 Bit → not used

DLC

- 4bits: Data Length Specifier, specifies the length of the Data which is following (max 8 = 0b1000)

4.2.2 Specifier



- The **CCS** (client command specifier) or **SCS** (server command specifier) is the transfer type
 - 1 = Download request
 - 2 = Upload request/response
 - 3 = Download response
 - 4 = Abort
- **n** shows the number of data bytes without data, maximum is 3 (0b11)
- If set, **e** indicates an 'expedited transfer' (all data is in a single CAN frame)
- If set, **s** indicates that data size is shown in n

4.2.3 Index & SubIndex

- Specifies the object to access
- See object dictionary of KNF Devices in separate Document
- **Index: Least significant Byte first**

4.2.4 Data

- Data to Transmit/Receive
- **Least significant Byte first**

5 Examples

5.1 Set Motor Speed to 1000rpm

1. Get object, datatype and unit from the device object documentation

Index-SubIndex	Object Name	Sub-Object Name	Key
0x68ff-0x00	Target speed	Speed Target	SpeedTargetCia402

SpeedTargetCia402	32	Speed Target (nTgt)	Target Speed. This value will be passed to the trajectory generator.	int32	mHz	Def: 0 Min: -100000 Max: 100000	rpm	Def: 0 Min: -6000 Max: 6000	1	1
-------------------	----	---------------------	--	-------	-----	---------------------------------------	-----	-----------------------------------	---	---

- The object to write is Index: 0x68ff Subindex 0x00
- Datatype is int32 → 4Bytes
- Unit is mHz → 1000rpm = 16.666Hz = **16666mHz**

2. Prepare the can frame

COB-ID	RTR	DLC	Specifier	Index	Sub-Index	Data			
0x601	0	8	CCS: 001 n: 00 e: 1 s: 1 0b00100011	0x68ff	0x00	16666 mHz = 0x0000411A			
0xc0	0x28	0x23	0xff	0x68	0x00	0x1a	0x41	0x00	0x00

- ccs: 001 --> initiate download request (write data to the server)
 n: 00 -> no omitted data bytes, thus 4 valid data bytes are to be transmitted
 e: 1 --> Expedited transfer (only one frame is used)
 s: 1 --> size indicator --> n contains valid data size information

3. Add Sequence Number

Sequence Number	Can Data									
0x01	0xc0	0x28	0x23	0xff	0x68	0x00	0x1a	0x41	0x00	0x00

Sequence number 1 indicates the initial frame of communication

4. Calculate CRC

Frame Data										CRC 0x7926		
0x01	0xc0	0x28	0x23	0xff	0x68	0x00	0x1a	0x41	0x00	0x00	0x26	0x79

CRC computation starts with the sequence number byte and ends with the last data byte of the frame.

5. Escape Character if needed

There are no 0x7e or 0x7d in the Stream, therefore no escaping is needed.

6. Add Start and End Flag

Start Flag	Frame Data											CRC 0x7926		End Flag
0x7e	0x01	0xc0	0x28	0x23	0xff	0x68	0x00	0x1a	0x41	0x00	0x00	0x26	0x79	0x7e

7. Transmit Frame

Transmit (hex): 7e01c02823ff68001a41000026797e

8. Receive Response

Receive (hex): 7e10b02860ff680000000000eade7e

Start Flag	Frame Data											CRC		End Flag	
0x7e	0x10	0xb0	0x28	0x60	0xff	0x68	0x00	0x00	0x00	0x00	0x00	0x00	0xea	0xde	0x7e

9. Remove Start- and End-Flag

Frame Data											CRC	
0x10	0xb0	0x28	0x60	0xff	0x68	0x00	0x00	0x00	0x00	0x00	0xea	0xde

10. Decode Escaped Character

There are no 0x7e or 0x7d in the Stream, therefore no decoding is needed.

11. Check CRC

CRC over the whole stream, including the received CRC should result in 0x0000

Frame Data											CRC		→ CRC Check 0x0000
0x10	0xb0	0x28	0x60	0xff	0x68	0x00	0x00	0x00	0x00	0xea	0xde		

12. Check Sequence Number

If sequence handling is enabled, verify that the newly received sequence number equals “lastReceivedSequence + 1”

If it does not, send a special frame containing the most recently received sequence number. In this example, the received sequence number 0xb0(176) is valid because the previous one was 0xaf(175).

13. Decode CAN Frame

COB-ID	RTR	DLC	Specifier	Index	Sub-Index	Data			
0x581	0	8	SCS: 011 n: - e: - s: - 0b01100000	0x68ff	0x00	Dont Care			
0xb0	0x28	0x60	0x68	0xff	0x00	0x00	0x00	0x00	0x00

COB-ID = 0x581 → correct for receiving frame from node 1 (0x580 means the transmission of a tx-SDO from Server to Client)

DLC = 8 → 8 Databytes to look at

SCS = 011 → Download response if a successful transfer of the data took place

Index = 0x68ff → Index that we wrote at

SubIndex = 0x00 → SubIndex that we wrote at

Data = Don't care because it was a write operation

5.2 Read actual Motor Speed

1. Get the object and datatype and unit from the device object documentation

Index-Subindex	Object Name	Sub-Object Name	Key
0x686c-0x00	Speed actual value	Speed Actual	SpeedActualCia402
SpeedActualCia402	39 Speed Actual (nm)	Actual Speed measured by the motor.	int32 mHz Def: 0 Min: -100000 Max: 100000 rpm Def: 0 Min: -6000 Max: 6000 1 0

The object to read is Index: **0x686c** Subindex **0x00**

Datatype is **int32** à 4Bytes

Unit is **mHz**

2. Prepare the can frame

COB-ID	RTR	DLC	Specifier	Index	Sub-Index
0x601	0	4	CCS: 010 n: 00 e: 0 s: 0 0b01000000	0x686c	0x00
0xc0	0x24	0x40	0x6c	0x68	0x00

COB-ID = 0x601 --> correct for transmitting a frame to node 1 (0x600 means the transmission of a rx-SDO from client to server)

DLC = 4 --> 4 Databytes to provide (Specifier, index and subindex)

ccs: 010 --> initiate upload request (request read data from the server)

n: 00 -> don't care

e: 0 --> don't care

s: 0 --> don't care

3. Add Sequence Number

Sequence Number	Can Data					
0x02	0xc0	0x24	0x40	0x6c	0x68	0x00

Sequence number 2 as this is the second frame we send

4. Calculate CRC



CRC computation starts with the sequence number byte and ends with the last data byte of the frame.

5. Escape Character if needed

There are no 0x7e or 0x7d in the Stream, therefore no escaping is needed.

6. Add Start- and End-Flag

Start Flag	Frame Data							CRC 0x6085		End Flag
0x7e	0x02	0xc0	0x24	0x40	0x6c	0x68	0x00	0x85	0x60	0x7e

7. Transmit Frame

Transmit (hex): 7e02c024406c680085607e

8. Receive Answer

Receive (hex): 7e55b028436c6800c83b0000b8ce7e

Start Flag	Frame Data											CRC		End Flag
0x7e	0x55	0xb0	0x28	0x43	0x6c	0x68	0x00	0xc8	0x3b	0x00	0x00	0xb8	0xce	0x7e

9. Remove Start- and End-Flag

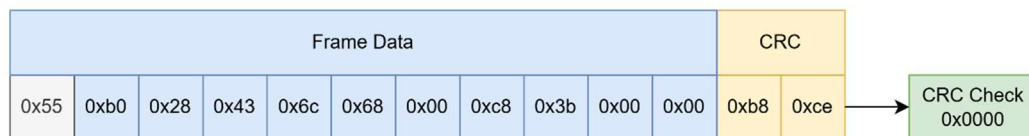
Frame Data											CRC	
0x55	0xb0	0x28	0x43	0x6c	0x68	0x00	0xc8	0x3b	0x00	0x00	0xb8	0xce

10. Decode Escaped Character

There are no 0x7e or 0x7d in the Stream, therefore no decoding needed.

11. Check CRC

CRC over whole Stream, including the received CRC should result in 0x0000



12. Check Sequence Number

If sequence handling is enabled, verify that the newly received sequence number equals "lastReceivedSequence + 1"

If it does not, send a special frame containing the most recently received sequence number. In this example, the received sequence number 0x55(85) is valid because the previous one was 0x54(84).

13. Decode CAN Frame

COB-ID	R T R	DLC	Specifier	Index	Sub-Index	Data							
0x581	0	8	SCS: 010 r- e- s- 0b01100000	0x686c	0x00	0x00003bc8 = 15304mHz							
0xb0	0x28	0x43	0x6c	0x68	0x00	0xc8	0x3b	0x00	0x00				

COB-ID = 0x581 → correct for receiving frame from node 1 (0x580 means the transmission of a tx-SDO from Server to Client)

DLC = 8 → 8 Databytes to look at

SCS = 010 → Upload Request response if a successful read of the object took place

Index = 0x686c → Index that we read from

SubIndex = 0x00 → SubIndex that we read from

Data = 0x00003bc8 = 15304mHz → **918.24rpm**