DATA SHEET

CanLink-01

NEWSON NV

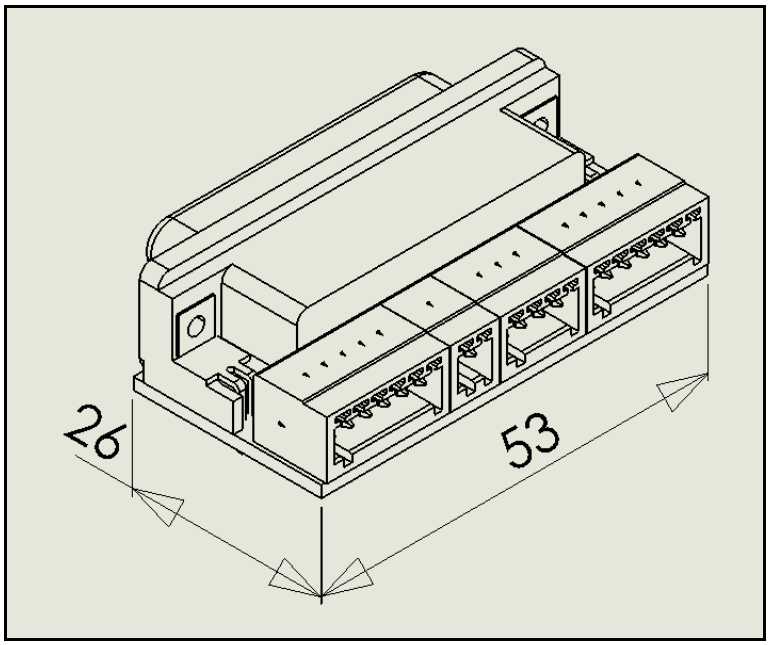
Table of Contents

1 CANLINK™ 3
1.1 DIMENSIONS 3
2 CONNECTOR IO-EXTENSION SIDE 4
2.1 CONNECTOR TYPE 4
2.2 INTERNAL SCHEME 4
2.3 PIN DESCRIPTIONS 4
3 USING THE CANLINK 5
3.1 CONNECTION TO CUA 5
3.2 CANOPEN 5
3.3 RHOTHOR DLL FUNCTIONS 7

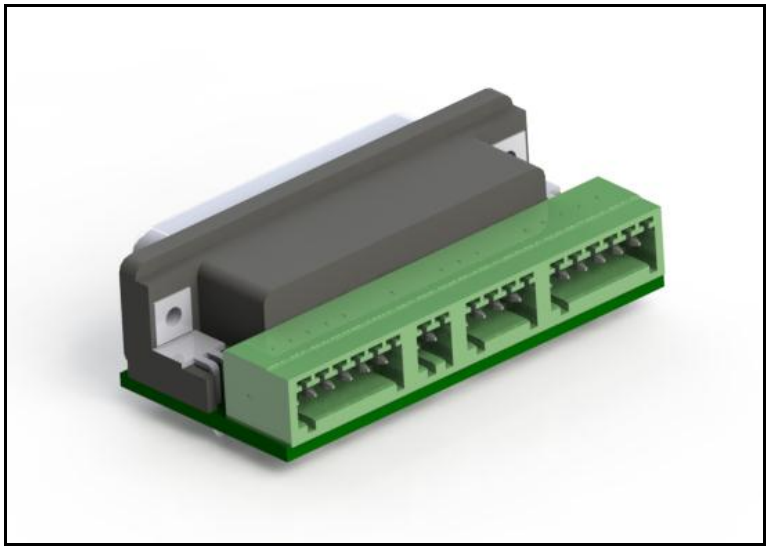
1 CANLINK™

The **CanLink™** hardware provides an interface to CANbus networks.

1.1 DIMENSIONS



All dimensions are in mm.

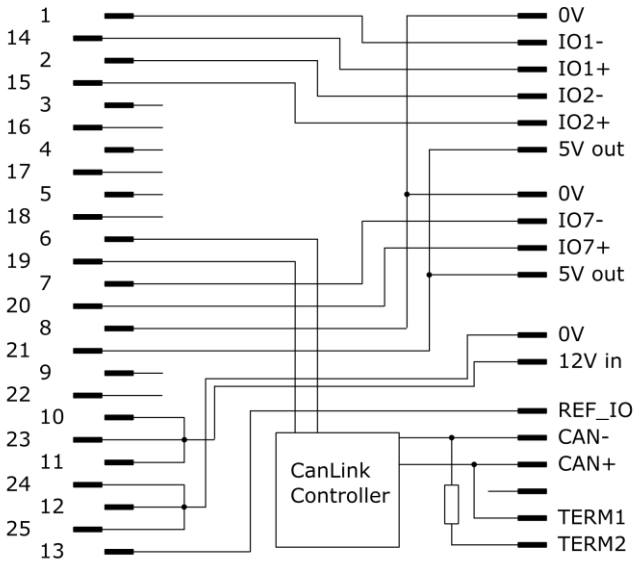


2 CONNECTOR IO-EXTENSION SIDE

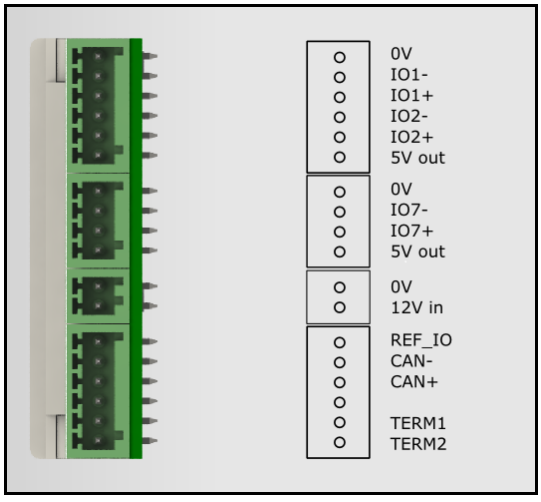
2.1 CONNECTOR TYPE

Phoenix MC 0,5/6-G-2,5
Phoenix MC 0,5/4-G-2,5
Phoenix MC 0,5/2-G-2,5
Phoenix MC 0,5/6-G-2,5

2.2 INTERNAL SCHEME



2.3 PIN DESCRIPTIONS



IO1-, IO1+, IO2-, IO2+, IO7-, IO7+: Differential I/O, RS485 compliant

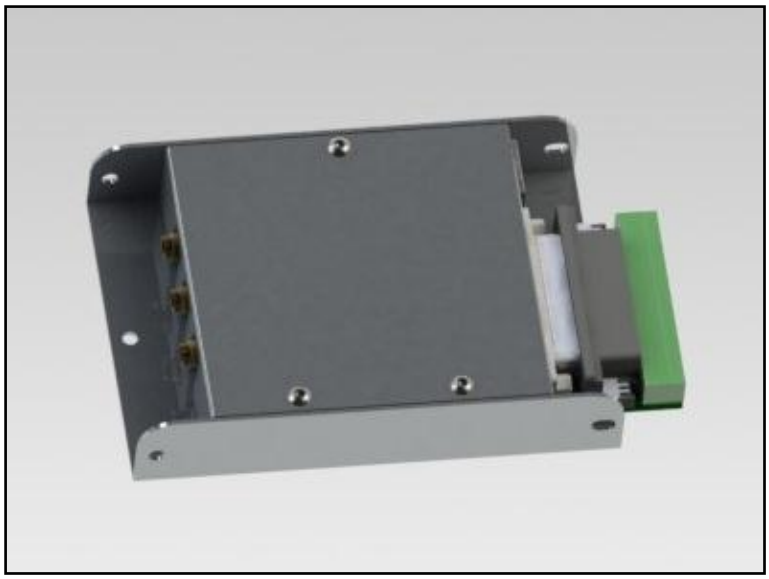
TERM1, TERM2: Termination inputs, to terminate bus close-wire TERM1 and TERM2

CAN-, CAN+: CAN bus network

3 USING THE CANLINK

3.1 CONNECTION TO CUA

Via sub-D 25p male, connect directly into the sub-D 25p female of the CUA-board.



3.2 CANOPEN

3.2.1 General

CANopen is a high level communications protocol, which operates on a Controller Area Network (CAN) bus. The standard defines exactly how information is exchanged between electronic controls. CANopen allows up to 127 different devices on a same bus.

Central concept in CANopen is the Object Dictionary (OD), which all nodes must contain. An OD contains all setup and configuration items for a node. The OD is implemented as an array of objects. Each object in the OD can be addressed through an Index (and SubIndex to address individual elements of structures of data)
This concept is also used in other fieldbus systems as well (Profibus, Interbus-S).

3.2.2 CAN bus

CAN bus is the physical layer of CANopen. It transmits short packets. An CANopen frame consists of a 11-bit ID, a remote transmission request (RTR) and 0 to 8 bytes of data

COB-ID	RTR	Data length	Data
11 bits	1 bit	4 bits	0-8 bytes

3.2.3 CANopen protocol

The CanOpen standard divides this 11-bit Communication Object Identifier (COB-ID) into a 4-bit functions code and a 7-bit CAN open node ID.

- The 4-bits function code determines the message type and specifies the priority on the bus (NMT, SYNC, EMCY, PDO, SDO...) This specifies the priority and type of the message.
- The CAN open node ID specifies the device's address on the network. The node identifier

There are 2 message types supported by CanLink which are meant for data transfer.

- A Process Data Object (PDO) protocol is used to transmit data. All PDOs are single frame, which means that a node can transfer up to 8 bytes of data per PDO. There are two kinds of PDOs: receive and transmit PDOs (RPDO and TPDO). A RPDO is used when a node needs to receive date and a TPDO is used when a node needs to transmit data. The CANopen standard defines eight identifiers, four for RPDOs and four for TPDOs.
- A Service Data Object (SDO) protocol is used to read and write values inside of an object dictionary. CANopen defines a write into an OD as an SDO download and a read from an OD as an SDO upload.

3.2.4 CANopen boot-up process

The CAN-network offers Network ManagemenT (NMT) services to the devices to initialize, start and stop nodes. This service is implemented as a master-slave concept. In a CAN-network the CanLink hardware will act as a **master**.
This means the CanLink hardware will be responsible to boot each node in the network.
To do so CanLink will issue NMT Module Broadcast message to the slaves. You can start this process by calling the rhothor library function rtOpenCanLink.

3.2.5 SDO Communication

The Service Data Object is used to access the Object Dictionary of a device. The requester of the OD access is called the client. This will always be the CANlink. The CAN devices whose OD is addressed is called the server. This will always be a CAN-slave. You can send an SDO packet to a CAN-slave by calling the rhothor library function rtSetCanLink.

3.2.6 PDO Communication

The Process Data Object is used to transfer real-time data. You can send a PDO packet to a CAN-slave by calling the rhothor library function rtSetCanLink.

3.2.7 Packet filtering and Synchronization

CAN-frames received by the CanLink hardware are automatically send to the CUA controller. These frames are filtered by the CUA controller and the filtered data is placed into registers on the CUA controller. A CanLink buffer (16bytes) is allocated on the CUA for this purpose. These registers are addressed using an address index (0 to 15)
By default, when there is no filter installed, all CAN-messages received by the CanLink are placed in this CanLink buffer starting on address 0. The method rtGetCanLink can be used to read out the content of the CanLink buffer on the CUA controller.

Use library function rtScanCanLink to install a filter on the CAN-message stream. A filter is defined by CAN open node ID, Index and SubIndex. Due to the CanLink buffer size you can install 2 filters at the same time (An CANopen frame can contain maximum 8 bytes of data). Once a filter is installed only CAN-messages that pass the filter are placed into the CanLink registers.

You can use the function rtWaitCanLink to make the CUA controller wait for a certain state in the CanLink registers.
To de-install a filter call rtScanCanLink with Index and SubIndex 0.

3.3 RHOTHOR DLL FUNCTIONS

The rhothor library contains several functions to control the devices on a CAN-network.
Rhothor.dll has following methods:

rtOpenCanLink,
rtGetCanLink,
rtSetCanLink,
rtWaitCanLink,
rtScanCanLink

long rtOpenCanLink(long Baudrate)

This method starts the CAN boot-up process. Baudrate defines the speed that will be used by CanLink to communicate with the CAN-slaves. Baudrate is expressed in kbps.

```
rtListOpen(1)
rtOpenCanLink(250)
rtListClose()
```

long rtSetCanLink(long Node, long Index, long SubIndex, char*Data)

This method sends a CAN message on the CAN-bus

Parameters

To send an SDO:
Node the CAN-slave Node ID on the CAN-bus.
Index defines which value of the Object Dictionary is accessed.
SubIndex defines which value of the Object Dictionary is accessed.
Data contains a hexadecimal string defining the data to send.

To send an PDO:
Node the CAN-slave Node ID on the CAN-bus.
Index should be 0 to define the CAN message request as a PDO request
SubIndex the endpoint number of the PDO. Should be 1,2,3 or 4.
Data contains a hexadecimal string defining the data to send.

long rtGetCanLink(long Address, long* Value)

This method retrieves the current value of the CANlink registers on the CUA-controller.

Parameters

Address Index indicating the register inside the CanLink buffer. Can be 0 to 15
Value value of the CanLink register

long rtScanCanLink(long Address, long Node, long Index, long SubIndex)

This method (de)installs a CanLink filter onto the CUA.

Parameters

Address	Index indicating start register inside the CanLink buffer where the filtered data of the CAN packets is stored.
Node	CAN-slave Node ID of the CAN packets which pass the filter
Index	the Index of the CAN packets which pass the filter
SubIndex	SubIndex of the CAN packets which pass the filter

long rtWaitCanLink(long Address, long Value, long Mask)

This method can be used for waiting. With this function the deflection control system goes idle till the CanLink register state defined by Value and Mask is reached.

Address	Index indicating the register inside the CanLink buffer which state is tested.
Mask	Defines bitwise which bits are to be checked
Value	Defines bitwise the condition of the register state defined by Mask