

IoT-GW30 | PROtop | PRO COM CAN

Quick Start Guide - CANopen communication with PROtop

Abstract:

This document describes how to communicate between an IoT-GW30 (Client) and a PROtop power supply (Server) via CANopen. The IoT-GW30 provides socketcan and Node-RED is used to implement partial functionality of the CANopen protocol. This enables to configure the PROtop (via PRO COM CAN module) and to read process data.

Hardware reference

No.	Component name	Article No.	Hardware / Firmware version
1	IOT-GW30	2682620000	Revision 6 / 1.14.0
2	PRO TOP1 120W 24V 5A EX	2466980000	2.2.0 / 2.0.1
3	PRO COM CAN OPEN	2467320000	1.0.0 / 1.0.3

Software reference

No.	Software name	Article No.	Software version
1	Node-RED	-	v. 1.1.2
2	node-red-contrib-socketcan	-	v. 1.2.5

File reference

No.	Name	Description	Version
1	QSG0041-IOT-GW30 Quick Start Guide for CANopen communication with PROtop.zip	The file contains a Node-RED flow related to this quick start guide	-

Contact

Weidmüller Interface GmbH & Co. KG
Klingenbergstraße 26
32758 Detmold, Germany
www.weidmueller.com

For any further support please contact your local sales representative:
<https://www.weidmueller.com/countries>

Content

1	Warning and Disclaimer.....	4
2	Preparation.....	5
2.1	Prerequisites.....	5
2.2	Hardware Setup.....	5
2.3	Software Setup.....	6
3	Application.....	7
3.1	Introduction.....	7
3.2	Init	8
3.3	Network Management (NMT).....	8
3.4	Service Data Objects (SDOs)	9
3.5	Process Data Objects (PDOs)	14

1 Warning and Disclaimer

Warning

Controls may fail in unsafe operating conditions, causing uncontrolled operation of the controlled devices. Such hazardous events can result in death and / or serious injury and / or property damage. Therefore, there must be safety equipment provided / electrical safety design or other redundant safety features that are independent from the automation system.

Disclaimer

This Application Note / Quick Start Guide / Example Program does not relieve you of the obligation to handle it safely during use, installation, operation and maintenance. Each user is responsible for the correct operation of his control system. By using this Application Note / Quick Start Guide / Example Program prepared by Weidmüller, you accept that Weidmüller cannot be held liable for any damage to property and / or personal injury that may occur because of the use.

Note

The given descriptions and examples do not represent any customer-specific solutions, they are simply intended to help for typical tasks. The user is responsible for the proper operation of the described products. Application notes / Quick Start Guides / Example Programs are not binding and do not claim to be complete in terms of configuration as well as any contingencies. By using this Application Note / Quick Start Guide / Example Program, you acknowledge that we cannot be held liable for any damages beyond the described liability regime. We reserve the right to make changes to this application note / quick start guide / example at any time without notice. In case of discrepancies between the proposals Application Notes / Quick Start Guides / Program Examples and other Weidmüller publications, like manuals, such contents have always more priority to the examples. We assume no liability for the information contained in this document. Our liability, for whatever legal reason, for damages caused using the examples, instructions, programs, project planning and performance data, etc. described in this Application Note / Quick Start Guide / Example is excluded.

Security notes

In order to protect equipment, systems, machines and networks against cyber threats, it is necessary to implement (and maintain) a complete state-of-the-art industrial security concept. The customer is responsible for preventing unauthorized access to his equipment, systems, machines and networks. Systems, machines and components should only be connected to the corporate network or the Internet if necessary and appropriate safeguards (such as firewalls and network segmentation) have been taken.

2 Preparation

2.1 Prerequisites

IoT-GW30:



The firmware version prior to 1.14.0 does not support the CAN interface. The default baud rate is 125kBd. Currently, the baud rate cannot be changed via Node-RED.

PRO COM CAN:



The firmware version prior to 1.0.3 has a fixed baud rate of 250kBd and is not compatible to the IoT-GW30. Subsequent versions provide an automatic detection for the baud rate.

2.2 Hardware Setup

The CAN-Bus is (mostly) wired in line structure. As mentioned before this document only describes the communication between two CAN participants to keep it as simple as possible. But the PRO COM module has two RJ45 sockets which allows to easily add multiple servers if necessary. The figure below shows a possible topology.

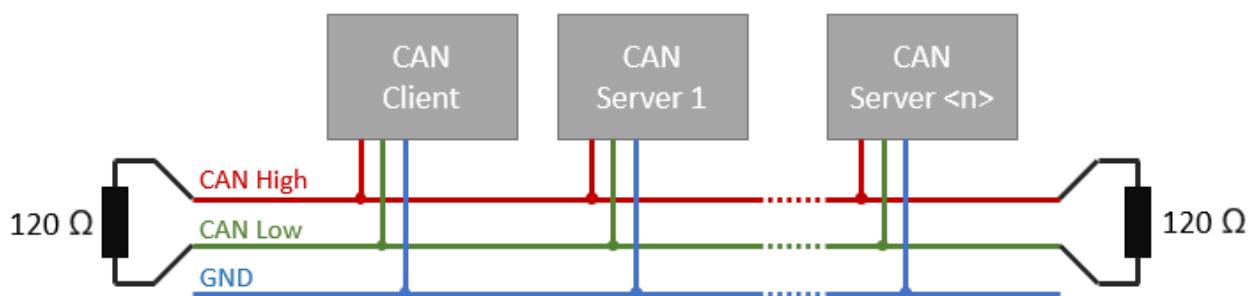


Figure 1 CAN wiring

The PRO COM module has RJ45 sockets whereas the IoT-GW30 has a terminal block for the CAN signals. The RJ45 pinout of the PRO COM module must be assigned as follows:

Table 1 PRO COM RJ45 pinout

Pin	Signal
1	CAN High
2	CAN Low
3	GND

The IoT-Gateway does not have an internal terminating resistor, so it must be added manually if needed. Whereas the terminating resistor of the PRO COM module can be activated with the DIP switch "T" set to "ON" position (Figure 2).

As mentioned before the default baud rates of the devices differ. The DIP switch "A" in the "ON" position (Figure 2) enables the automatic baud rate detection.

The CANopen servers are identified by unique so called "node IDs" between 1 and 127. The hexadecimal rotary switches "H" (upper address byte) and "L" (lower address byte) are used to set the desired id (Figure 2).

For further details (e.g., LED flashing behavior) see the operating instruction¹ of the PRO COM module.

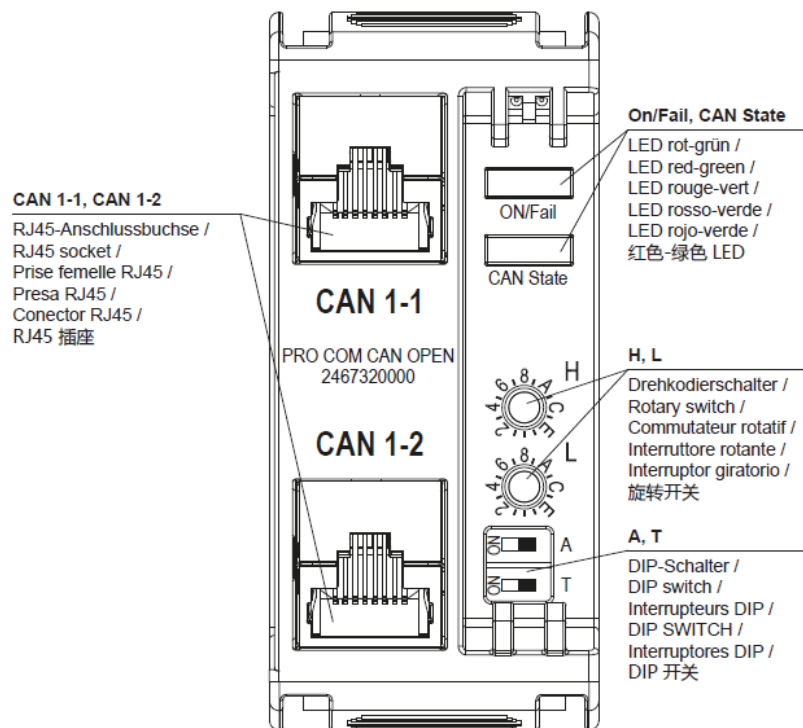


Figure 2 PRO COM CAN OPEN - top view

2.3 Software Setup

Download and install "node-red-contrib-socketcan" via Node-RED Editor ("manage palette", internet connection required). Version 1.2.5 is used for this QSG, but later versions should work properly, too.

Import attached Node-RED flows via Node-RED Editor and deploy the application.

¹ Can be found in the Downloads section of the corresponding catalogue [page](#).

3 Application

3.1 Introduction

The application example is split into different flows, which are described in the following subchapters. In order to increase the readability of the application, different sections are grouped and linked between each other. This way you can easily move the different parts for testing.

The node “socketcan in” accepts a few input parameters as described in the help section of that node. The parameters “canfd”, “ext” and “rtr” are not used in this application and always set to <false>. This document is not a CANopen tutorial, and it does not describe in detail how to setup the properties for each communication object. This can be found out by reading the source code and the comments of the corresponding nodes. But there is always a reference to the corresponding section in the CANopen specification which contains the full information.

Known issues / General hints:



Receiving messages on “socketcan out” node does not mean that the communication is established. The CAN specification requires that each CAN frame must be acknowledged by another device. If the communication is not possible (e.g., due to bad wiring or a missing server) the message is resent. In this setup this leads to a lot of messages on the “socketcan out” node. If these messages are permanently displayed on the debug output, Node-RED will crash (and restart) in a short time.



Figure 3 Warning debug output

So be careful about activating a debug-node directly behind the “socketcan out” node. The same applies to warning messages (node.warn).



Node-RED sometimes crashes during the deploying process. Avoid using multiple instances of the “socketcan in” node as well as “socketcan out” node.



Standard operation seems stable, but keep in mind that the described CANopen functionality is based on an open-source node, provided by a third party without any affiliation to Weidmüller.

A CANopen node has an object dictionary which defines a list of objects that are available on the device. The “Electronic Data Sheet” (EDS-file) of the COM CAN module contains the information that are required to communicate with the PROtop.

3.2 Init

An initialization flow is triggered once on startup. It contains information and functions which are used several times in other nodes.

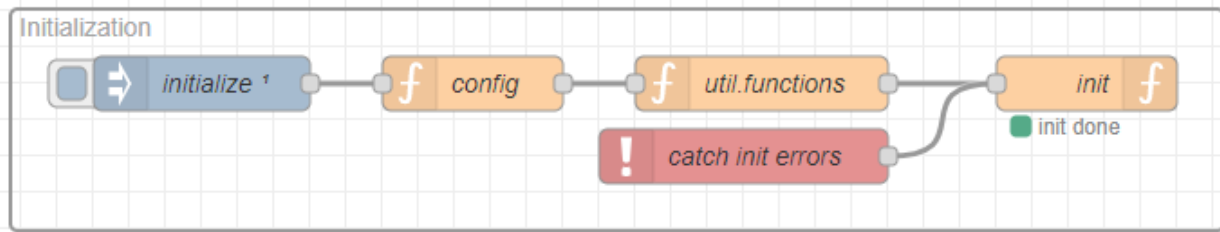


Figure 4 Initialization flow

The “config” node mainly contains information about the CANopen Server. The property “config.nodeId” is important and must match the node id of your “Hardware Setup”. If “config.debug” is equal to <true>, you will get some additional information while the program is running or in case of errors. The rest of the configuration content is referenced where it is needed in the following chapters.

The “util.functions” node contains some useful functions. In this application, only two conversion functions are defined, but the general structure allows to expand the content in a quite structured way.

3.3 Network Management (NMT)

The network management objects are used to control the state machine of the CANopen device. More detailed information can be found in the subsection Network management (Network Management Objects for previous versions) of the specification CiA 301². The “NMT” node takes the commands (msg.command) as an input parameter and creates the corresponding message for the “socketcan in” node.

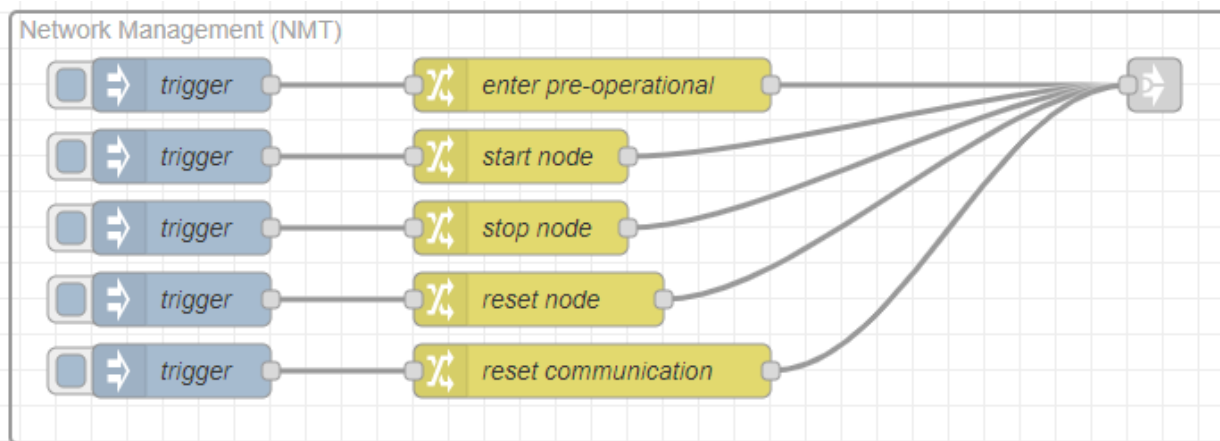


Figure 5 NMT commands

² Download link: [CiA 301 - CANopen application layer and communication profile](#).

As soon as the first message is sent to the PRO COM module, the status of the PRO COM module should change (see LED behavior in PRO COM operating instruction). Sending the command “enter pre-operational” enables the PRO COM module to process “Service Data Objects (SDOs)”-requests.

3.4 Service Data Objects (SDOs)

The SDO protocol provides access to the objects which are described in the EDS-File³. Some of these objects are predefined in the “config” node (which of course can be expanded). The global variable “config.sdos” contains these objects, so they can be used in the whole application.

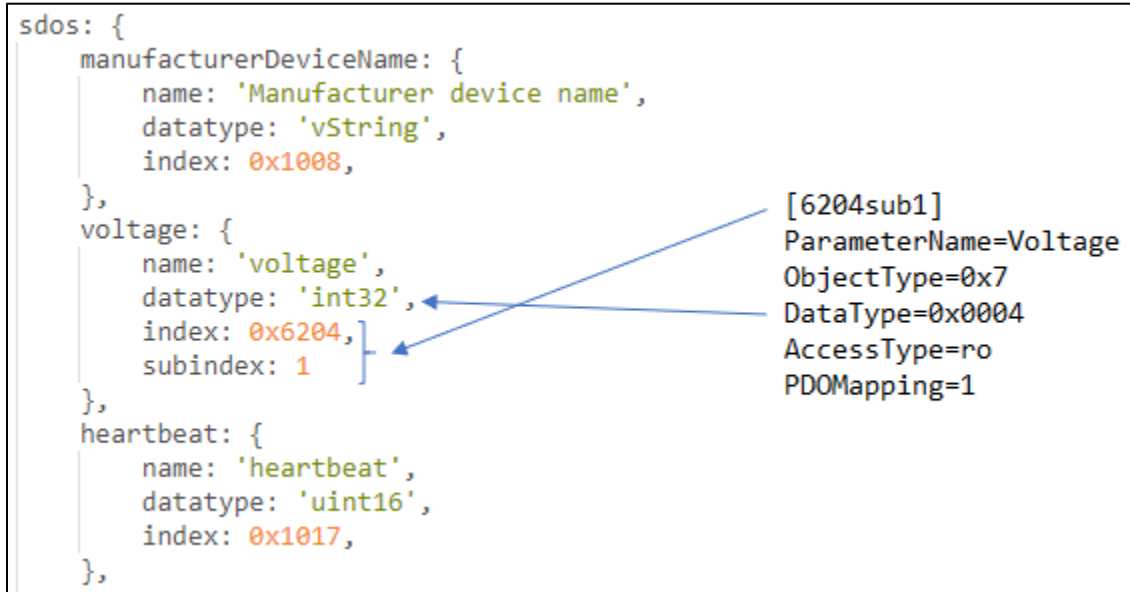


Figure 6 Excerpt of SDO config

The datatype is required to process the values. As mentioned before there are already predefined functions (util.functions.conversion) which convert the value to a byte array and vice versa (Figure 7). The functions can handle either the datatype names or the corresponding number (which can be found in the specification). However, not all data type conversions are predefined, only the basic data types and those which are required for this application example. But these functions can also be easily expanded.

³ Can be found in the Downloads section of the corresponding catalogue [page](#).

```

conversion: {
  valueToRaw: function(value, datatype = '-'){ //convert 'value' to byte-array

    datatype = typeof datatype === 'string' ? datatype.toLowerCase() : datatype;
    const buffer = new ArrayBuffer(4);
    const view = new DataView(buffer);
    let raw = [];

    switch(datatype){ //datatype specification (CiA 301 - chapter 9.5.3)
      case 'vstring':
      case 0x0009:
        for (let i=0; i<value.length; i++){
          raw.push(value.charCodeAt(i));
        }
        break;
      case 'uint8':
      case 0x0005:
        raw[0] = value;
        break;
      case 'int8':
      case 0x0002:
        view.setInt8(0,value);
        raw[0] = view.getUint8(0);
        break;
    }
  }
}

```

Figure 7 Excerpt of conversion functions

An SDO download can be used to write data to an SDO. Each example node contains an identifier from the SDO config (e.g. msg.identifier = "heartbeat") and a value to be set (e.g. msg.value = 1000). This information is forwarded to the "SDO - download (initiate)" which takes this information in combination with the config and the conversion functions to create a corresponding request.

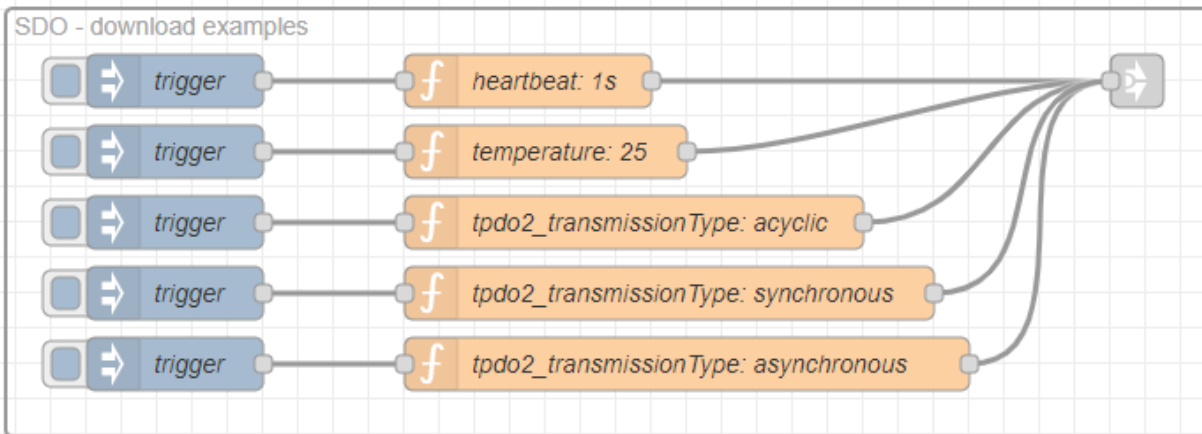


Figure 8 SDO download examples

The "heartbeat" object is the recommended way to receive the Server operation state cyclically. Triggering the "heartbeat" example results in a response from the PROtop which is received on

the “socketcan out” node. The message is handed over to the “assign response” node which forward it individually to different functions for further processing (Figure 9).

Each SDO request is evaluated individually by the “SDO response” node depending on the request type and the result is indicated on the node status. Afterwards the CANopen state of the PROtop becomes updated and displayed in the “heartbeat” node every second (Figure 9).

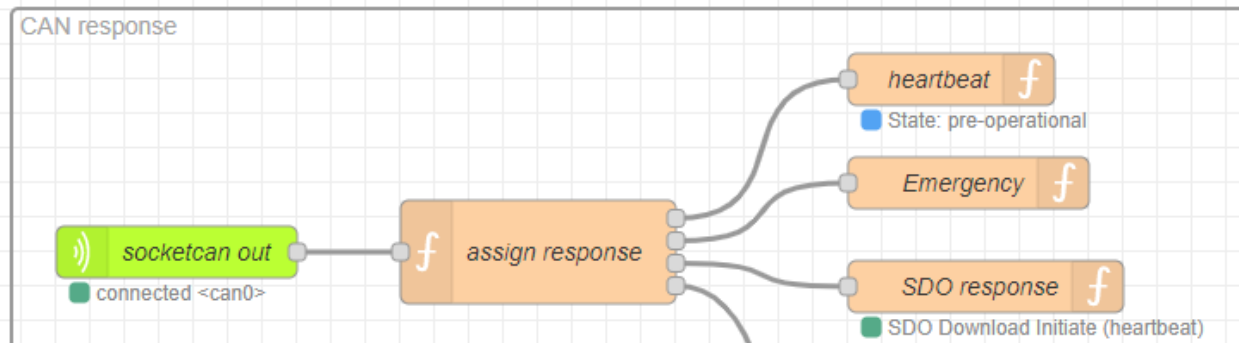


Figure 9 SDO response - heartbeat enabled

The temperature (download) example (Figure 8) shows an invalid request. As expected, a temperature value cannot be set by the Client.



Figure 10 SDO response - error

The PROtop replies with error data. This data is also interpreted by the “SDO response” node using mapping information which are predefined on the “config” node (config.sdoAbortCodes).

```
sdoAbortCodes: { //SDO abort codes (CiA 301 - chapter 9.2.2.2.7)
  0x05030000: {
    id: 0, //id can be used to evaluate certain errors (not part of the specification)
    description: 'Toggle bit not alternated'
  },
  0x05040000: {
    id: 1,
    description: 'SDO protocol timed out'
  },
  0x05040001: {
    id: 2,
    description: 'Client/server command specifier not valid or unknown'
  },
  0x05040002: {
    id: 3,
    description: 'Invalid block size (block mode only)'
  },
  0x05040003: {
    id: 4,
    description: 'Invalid sequence number (block mode only)'
  },
}
```

Figure 11 Excerpt of SDO abort codes

A SDO upload can be used to read data from an SDO. Each example node contains an identifier from the SDO config (e.g. msg.identifier = "temperature") to be read. This information is forwarded to the "SDO - upload (initiate)" which takes this information in combination with the config and the conversion functions to create a corresponding request.

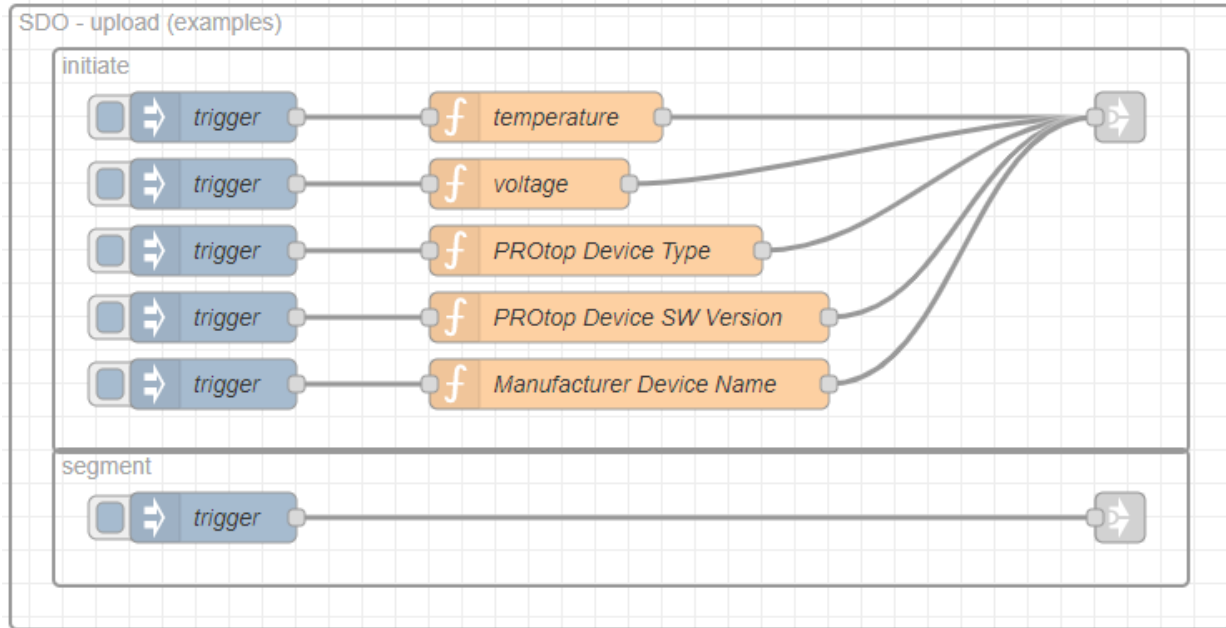


Figure 12 SDO upload examples

The Server response is evaluated again by the "SDO response" node. The displayed temperature value (Figure 13) is obviously incorrect.



Figure 13 SDO response - read temperature

Object 6000h to 9FFFh contains some profile specific information. In the context of the PROtop it is about standardized objects of power supplies (CiA 453). Particularly the object 6000h contains information about conversion and units. These information are provided in the "config" node (config.profileSpecific) and helps to interpret the data of the EDS-file. For the temperature, the value must be divided by 100 and the unit is "°C".

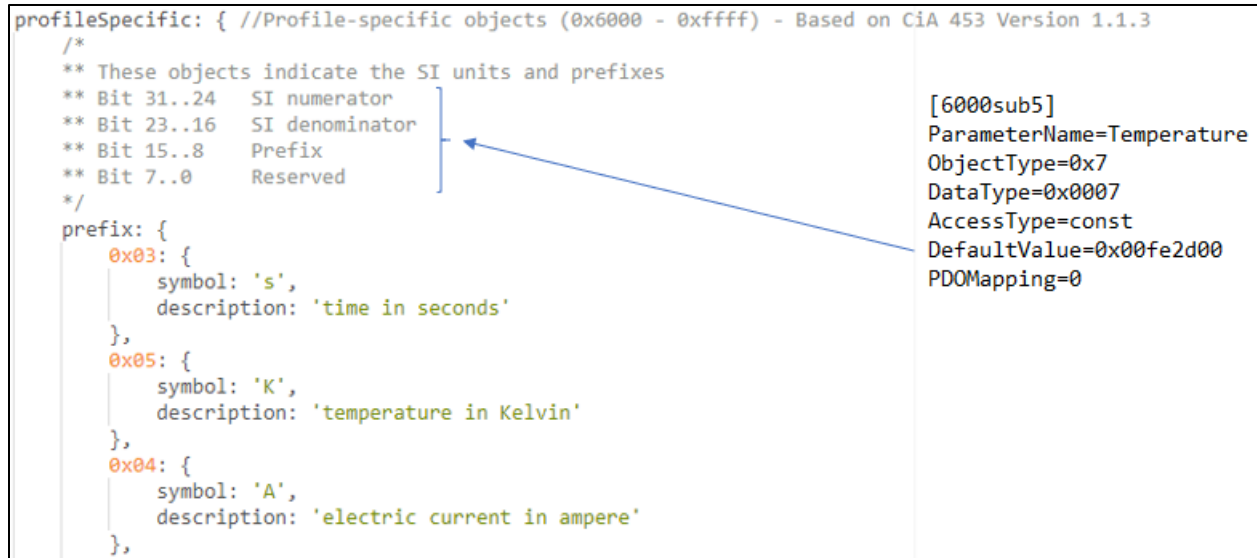


Figure 14 Excerpt of profile specific information

Some SDOs contain data which is greater than 4 bytes and cannot be transmitted in one message. This is often the case with string variables (e.g., “PROtop Device SW Version”). After an SDO upload (initiate) the “SDO response” nodes shows that more bytes need to be uploaded.



Figure 15 SDO response - SW Version: more bytes to be uploaded

Afterwards the SDO upload (segment) request (Figure 12) must be triggered until all bytes have been transmitted and the “SDO response” node indicates the result.



Figure 16 SDO response - SW Version: complete

Hint: An SDO download segment would work similarly but is not part of this sample application.

3.5 Process Data Objects (PDOs)

The PDO protocol is used for broadcasting high-priority data. A distinction is made between RPDOs (receive) and TPDOs (transmit). This application example only contains the TPDOs in order to receive the process data from the PROtop.

The mapping information of the TPDOs can be found in object 1A00h to 1A03h. The TPDOs of the PROtop are predefined and cannot be changed. Figure 17 shows an example for the relationship between the mapping object and the TPDO content.

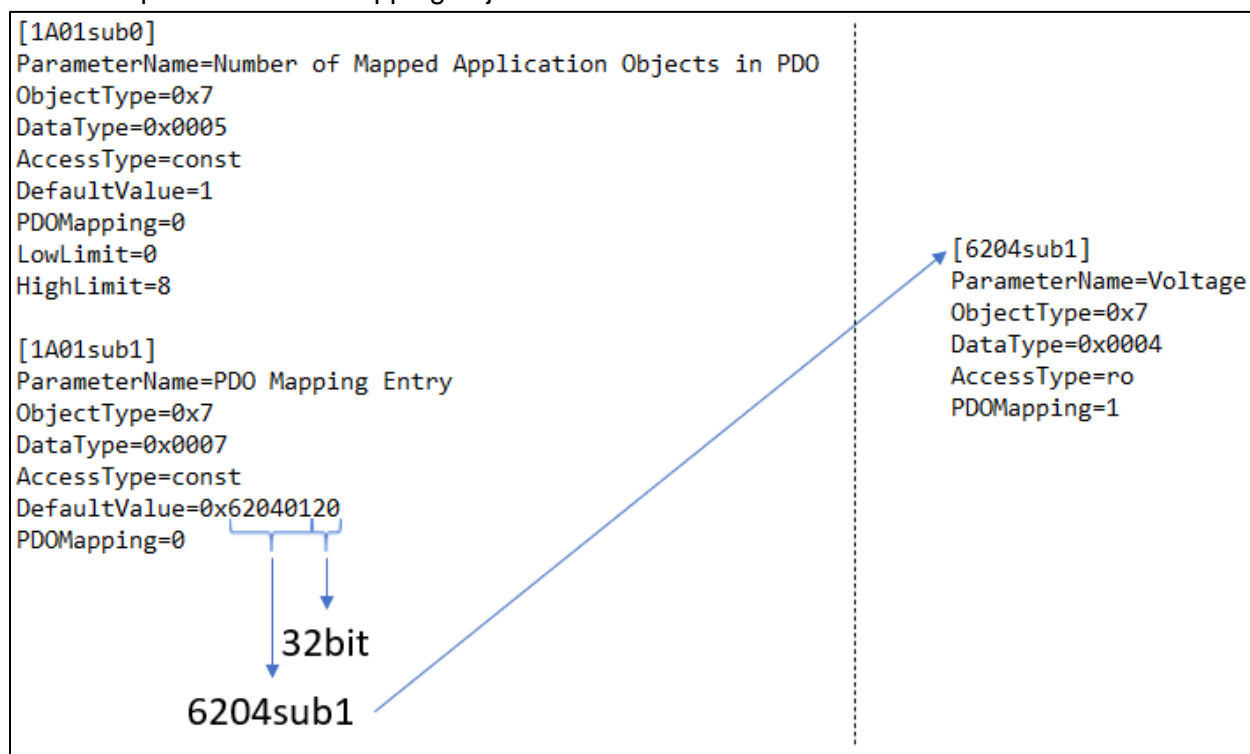


Figure 17 TPDO 2 - mapping example

The other TPDOs are mapped as in shown in Table 2 . This information is also provided in the “config” node (config.tpdos).

Table 2 TPDO mapping

TPDO	Mapping Object	Mapped Object	Info
1	1A00h	6011sub1	Condition
2	1A01h	6204sub1	Voltage
3	1A02h	6204sub2	Current
4	1A03h	6204sub3	Power

The PROtop must be in the “operational” state for PDO communication. This can be achieved by sending the “start node” command (Figure 5).

CANopen supports different transmission types. The PROtop sends the data “asynchronous” (or event-driven) when the value altered. “Synchronous” (cyclic) interval cannot be configured so it will not work. The transmission type can be set to “acyclic” in order to disable the automatic transmission. In this case the data can be fetched manually by a SDO upload request. The “SDO download examples” contain the information how to change the transmission type.

In the application example the TPDO data is separated into different “function” nodes which display the latest value (Figure 18). The conversion factors and unit information are hardcoded according to the Figure 14.

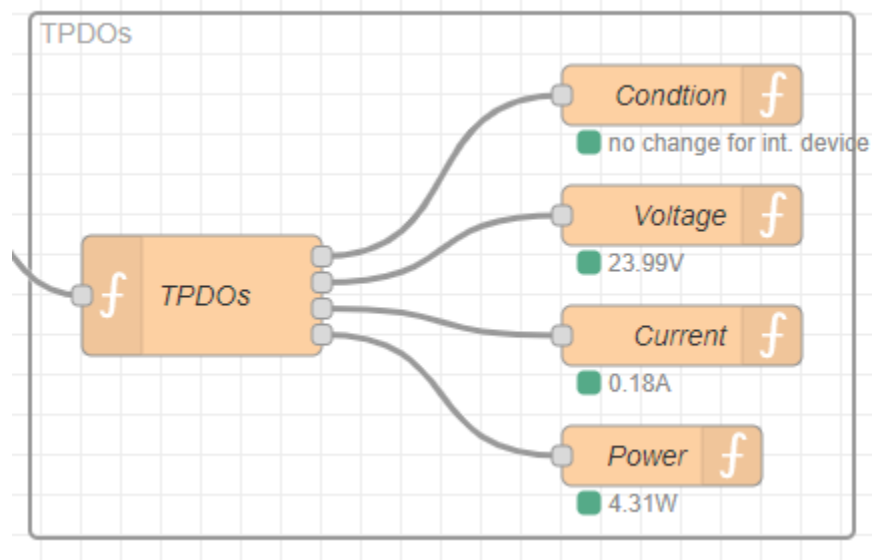


Figure 18 TPDO response