Interfacing To Inverter using Python and Peak CAN Interface

Emsiso Inverters primarily use the CANOpen protocol to communicate with outside tools. It is possible to use public libraries to communicate with the inverter using Python to automate production or assembly processes.

In this article, we will show how to connect to an Emsiso inverter using a PEAK CAN Interface and set/get a parameter value using an SDO Read/Write operations.

The library that we use in this guide is:

canopen-python/canopen: CANopen for Python

- Note that all conditions or limitations still remain the same as if using the emDrive Configurator.
- The article is intended for programmers with working knowledge of Python.
- This article expects a basic understanding of CANOpen. It is suggested to get a cursory overview by reading this article on CANOpen basics: CANopen Explained - A Simple Intro [2025] – CSS Electronics

We will be using the publicly available Python CANOpen Library: canopen · PyPI

Note that most of the documentation on this page is lifted from the referenced pages, with minor modifications in order to show interfacing to the inverter.

> Exhaustive documentation for **canopen** is available on the official docu website:
> canopen 0.1.dev52+ge840449 documentation

Minimum Requirements:

- Python 3.8 or up **(except for Python 3.12.0 due to an unresolved issue with the canopen library)**
- PEAK CAN Interface Drivers

- Any Emsiso Inverter

Initial Setup:

First we need to install dependencies. In our case, the canopen library already installs the can-interface. It is strongly recommended to also install the *uptime* library in order to correctly show the frame times.

- pip install canopen
- pip install uptime

Step 1: Verify Connection:

In order to verify that the setup has been performed correctly, we can scan the nodes on the bus.

Take care to define proper channel and bitrate. The example below contains the values:

- PCAN_USBBUS1: Windows machine with 1 CAN interface connected
- 500000: 500 kbps bit rate

> Further information on connecting HW interfaces (esp. PEAK) is available here:
> PCAN Basic API - python-can 4.5.0 documentation

```
import canopen
import time

network = canopen.Network()

# We connect to the interface here. In our case it's the PCAN interface on bus 1.
network.connect(interface='pcan', channel='PCAN_USBBUS1', bitrate=500000)

time.sleep(0.05)
for node_id in network.scanner.nodes:
    # We expect to find 1 node if the inverter connection is successful
    print(f"Found node {node_id}!")

network.disconnect()
```

If the inverter is properly connected, is the only device on the bus, and all dependencies are correctly installed, you can expect the following output:

- Found node 1!

Step 2: Session Setup and End

In order to set up the CANOpen session, we first need to define the network - also containing the definition of the interface - and the node.

In order to properly use the CANOpen protocol the object definition file (.eds) needs to be provided. The EDS can either be downloaded from the device using the emDrive configurator or can be provided by request from Emsiso. The object dictionary defines the available objects on the devices, it's indexes, subindexes, descriptions and types.

For more information on how to interact or get values from the object dictionary, please refer to the documentation for Python canopen.

Session Setup:

```
import canopen
import time


network = canopen.Network()


# We connect to the interface here. In our case it's the PCAN interface on bus 1.
network.connect(interface='pcan', channel='PCAN_USBBUS1', bitrate=500000)


#Insert the node ID of the device you want to connect to and provide the EDS file.
#In this case, we are using node ID 1 and the EDS file for the emDRIVE COMPACT 450/60.
node = canopen.RemoteNode(1, 'emDRIVE_COMPACT_450_60_APP_V1.18.4.eds')


# We add the node to the network.
network.add_node(node)


# {... user code ... }
```

In order to properly disconnect and dispose of the resources, we need to perform some houskeeping on the session end.

Session End:

```
# { ... user code ... }


# If a network sync is active, stop the communication
# network.sync.stop()
```

```
# Disconnect from the network interface
network.disconnect()
```

Step 3: SDO Read/Write

Read Operation:

We will read from the [1018:1] - Vendor ID. The expected Vendor ID for Emsiso devices is: **966**.

```
# Here we perform an SDO read operation from Index 0x1018, Subindex 1 in order to get the vendor ID.
vendor_id = node.sdo[0x1018][1].raw

#Print the vendor ID.
print(f"Vendor ID: {vendor_id}")
```

Expected Output:

- Vendor ID: 966

Write Operation:

Writing to the object is very similar to reading. In this example we will be writing to object 2010:2 - Key Input. This object is normally used to raise the device access level, but in this case we will simply use it as an example. The device behaviour will not be changed by our write operation.

```
# Here we preform a SDO write operation into the Key Input parameter - Index 0x2010, Subindex 2.
# Writing a value of 1 to this parameter.
node.sdo[0x2010][2].raw = 1

#Print the vendor ID.
print(f"Write Successful.")
```

Expected Output:

- Write Successful

Step 4: NMT Operations & Device Reset

During the configuration of the inverter, it will sometimes be necessary to switch between different NMT states. In this case, we can use the NMT command.

An example on how to set the NMT state of the device (note that the network setup and disconnect must still be performed as in the above code examples):

```
# This will send a NMT command 0x01 (go to Operational) to a Node
node.nmt.send_command(0x1)
```

NMT command code	Meaning
0x01	Go to 'operational'
0x02	Go to 'stopped'
0x80	Go to 'pre-operational'
0x81	Go to 'reset node'
0x82	Go to 'reset communication'

Step 5: Saving Permanent Parameters

On Emsiso inverters, there is a set of permanent parameters. If you want to keep persistence across power cycle, these parameters must be stored after they are modified. This is done by writing a command to a specified object.

Saving & resetting the parameters to default values can be done by writing specific values into parameters. Please see the chapter about storing parameters in this KB article: Parameters | Emsiso KB

Revision #7
Created 9 June 2025 10:13:19 by Admin
Updated 7 November 2025 07:43:19 by Admin