Open source implementation of ECSS-CAN Bus Extension Protocol for CubeSats

CAN in Space workshop
14-16 June 2017 at SITAEIL, Moli di Bari, Italy
Artur Scholz, Visionspace Technologies GmbH
Motivation

CubeSat Mission Status, 2000-present

- **Unknown**: 8.7%
- **Mission Achieved**: 18.9%
- **Mission In Progress**: 24.8%
- **Stowed**: 3.9%
- **Launch Fail**: 21.1%
- **DOA**: 15.2%
- **Early Loss**: 7.1%

Chart created Oct 2016 using data from M. Swartwout
https://sites.google.com/a/slu.edu/swartwout/home/cubesat-database
Overview

- CubeSats are lacking a space-grade command and control bus
- Traffic is moderate but most be utmost reliable and real-time
- Implementable in low-cost, constrained microcontroller
- Bus interaction should be simple and practical
Bus candidates

- MIL-STD-1553
- UART
- SPI
- I2C
- USB
- SpaceWire
- CANBus Extension minimal implementation
ECSS-E-ST-50-15C minimal implementation for CubeSats

Space engineering
CANbus extension protocol

ECSS-Secretariat
ESA ESTEC
Requirements & Standards Division
Noordwijk, The Netherlands
4.5 Communication model

The communication model is based on a CAN Network master connected to up to 126 slave devices.

7 bit ID => 128 (0..127)
0 = broadcast (not used)
1 = master
2-127 = slave
Communication model - CubeSat

- OBC
- EPS
- COM
- P/L
- Sensor
5.1.1.1 General

A spacecraft system using CAN Network shall use either of the following physical topologies:

1. A Linear multi-drop topology…

2. A Daisy chain topology...

Figure 5-1: Linear multi-drop topology

Figure 5-2: Daisy chain topology.
Physical layer – topology - CubeSat

No cables
No connectors
Medium is copper tracks and pins

120 Ohm terminating resistor
Data link layer

- 11-bit CAN ID, no ID extension
- No RTR (remote transmission request)
- No remote and overload frame
- Only data frame and error frame
CANopen higher layer protocol

- Service data objects
- Process data objects
- Synchronization object
- Emergency object
- Network management objects
  - Module control services
  - Error control services
  - Bootup service
  - Node state diagram
- Electronic data sheets
- Device and application profiles
- Object dictionary
CANopen higher layer protocol

- Service data objects
- Process data objects - PDO
- Synchronization object - SYNC
- Emergency object
- Network management objects
  - Module control services
  - Error control services - HB
  - Bootup service
  - Node state diagram
- Electronic data sheets
- Device and application profiles
- Object dictionary - OD

Minimal implementation
4.8.1 Overview

The selective bus access architecture allows communication on one bus at a time, whereas the parallel bus access architecture allows simultaneous communication on both a nominal and a redundant bus.
4.8.2 Node Monitoring via … Heartbeat Messages

...a node automatically transmits its communication state…

9.4.6.1 Module control services

Autonomous operations of slave nodes shall not be used.

6.5.2 Error control service

All slave nodes shall consume the redundancy master Heartbeat message.

4.8.3 Bus monitoring and reconfiguration management

The Redundancy Master defines the bus to be considered active by periodic transmission of CANopen Heartbeat messages on the active bus. The slave nodes monitor the presence of the Heartbeat message from the master to determine the active bus.
Redundancy management and monitoring

8.3.2 Start-up procedure
After a node power-on or after hardware reset, the node shall use the bus defined by the Bdefault parameter as the active bus.

9.4.9.2 Bootup service
Nodes shall not produce Bootup Service messages.
Redundancy management and monitoring

8.3.3 Bus monitoring protocol

The Redundancy Master shall periodically produce CANopen Master Heartbeat messages on the active bus.

The RM shall switch over and operate on the alternate bus by…:

1. Stopping transmission of HB messages on the active bus, and
2. Starting transmission of HB messages on the alternate bus

Each slave node shall be a consumer of the Master HB messages

Each slave node shall periodically transmit CANopen HB messages on the bus it considers being the active.
Redundancy management and monitoring

Figure 8-2: Bus monitoring logic

Figure 8-3: Slave bus selection process, toggling mechanism
# Redundancy management and monitoring

<table>
<thead>
<tr>
<th><strong>Table 8-1: BUS redundancy management parameters for slaves</strong></th>
<th><strong>Table 8-2: BUS redundancy management parameters for master</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>Consumer Heartbeat Time</td>
<td>Consumer Heartbeat Time</td>
</tr>
<tr>
<td>Index: 1016h</td>
<td>Index: 1016h</td>
</tr>
<tr>
<td>Subindex: 01h</td>
<td>Subindex: one per slave node</td>
</tr>
<tr>
<td>Data Type: Unsigned16 + Unsigned16</td>
<td>Data Type: Unsigned16 + Unsigned16</td>
</tr>
<tr>
<td>NodeID, HB Time in milliseconds</td>
<td>NodeID, HB Time in milliseconds</td>
</tr>
<tr>
<td></td>
<td>The Consumer Heartbeat Time parameter is specified by CANopen. The parameter specifies the maximum time allowed between two subsequent Heartbeat messages linked to that heartbeat consumer.</td>
</tr>
<tr>
<td></td>
<td>The Consumer Heartbeat Time parameter is specified by CANopen. The parameter specifies the maximum time allowed between two subsequent Heartbeat messages linked to that heartbeat consumer.</td>
</tr>
<tr>
<td>Producer Heartbeat Time</td>
<td>Producer Heartbeat Time</td>
</tr>
<tr>
<td>Index: 1017h</td>
<td>Index: 1017h</td>
</tr>
<tr>
<td>Subindex: 00h</td>
<td>Subindex: 01h</td>
</tr>
<tr>
<td>Data Type: Unsigned16</td>
<td>Data Type: Unsigned16</td>
</tr>
<tr>
<td>Unit: millisecond</td>
<td>Unit: millisecond</td>
</tr>
<tr>
<td></td>
<td>The Producer Heartbeat Time parameter is specified by CANopen. The parameter specifies the maximum time allowed between two subsequent Heartbeat message transmissions.</td>
</tr>
<tr>
<td></td>
<td>The Producer Heartbeat Time parameter is specified by CANopen. The parameter specifies the maximum time allowed between two subsequent Heartbeat message transmissions.</td>
</tr>
<tr>
<td>Bdefault</td>
<td>Bdefault</td>
</tr>
<tr>
<td>Index: 2000h</td>
<td>Index: 2000h</td>
</tr>
<tr>
<td>Subindex: 01h</td>
<td>Subindex: 01h</td>
</tr>
<tr>
<td>Data Type: Unsigned8</td>
<td>Data Type: Unsigned8</td>
</tr>
<tr>
<td></td>
<td>Bdefault specifies the bus to be considered active after a node power-on, node hardware reset.</td>
</tr>
<tr>
<td></td>
<td>Bdefault specifies the bus to be considered active after a master power-on or master hardware reset.</td>
</tr>
<tr>
<td>Trogate</td>
<td>Trogate</td>
</tr>
<tr>
<td>Index: 2000h</td>
<td>Index: 2000h</td>
</tr>
<tr>
<td>Subindex: 02h</td>
<td>Subindex: 02h</td>
</tr>
<tr>
<td>Data Type: Unsigned8</td>
<td>Data Type: Unsigned8</td>
</tr>
<tr>
<td></td>
<td>Trogate specifies the number of Consumer Heartbeat times during which the node is required to be listening for an NMT HB message on a particular bus before switching to the other bus.</td>
</tr>
<tr>
<td></td>
<td>Trogate specifies the number of Consumer Heartbeat times during which the node is required to be listening for an NMT HB message on a particular bus before switching to the other bus.</td>
</tr>
<tr>
<td>Ntrogate</td>
<td>Ntrogate</td>
</tr>
<tr>
<td>Index: 2000h</td>
<td>Index: 2000h</td>
</tr>
<tr>
<td>Subindex: 03h</td>
<td>Subindex: 03h</td>
</tr>
<tr>
<td>Data Type: Unsigned8</td>
<td>Data Type: Unsigned8</td>
</tr>
<tr>
<td></td>
<td>Ntrogate specifies the number of toggles between the Nominal and Redundant bus in case of no HB message being detected. If an even number is used the last toggle puts the system into Bdefault.</td>
</tr>
<tr>
<td></td>
<td>Ntrogate specifies the number of toggles between the Nominal and Redundant bus in case of no HB message being detected. If an even number is used the last toggle puts the system into Bdefault.</td>
</tr>
<tr>
<td>Ctrogate</td>
<td>Ctrogate</td>
</tr>
<tr>
<td>Index: 2000h</td>
<td>Index: 2000h</td>
</tr>
<tr>
<td>Subindex: 04h</td>
<td>Subindex: 04h</td>
</tr>
<tr>
<td>Data Type: Unsigned8</td>
<td>Data Type: Unsigned8</td>
</tr>
<tr>
<td></td>
<td>The counter of Ctoggles (bus toggles) shows the count of the number of toggles that have already been performed by the device.</td>
</tr>
<tr>
<td></td>
<td>The counter of Ctoggles (bus toggles) shows the count of the number of toggles that have already been performed by the device.</td>
</tr>
</tbody>
</table>
7.1.1 Time code formats

Each device … that maintains time information shall use Spacecraft Elapsed Time (SCET)… The time code format .. is the CCSDS Unsegmented Time Code (CUC).

![Figure 7-1: Format for objects containing the SCET]

8bit fine time→ 100 ms = 100 * 256/1000 = 25.6 → 25
7.1.2 Spacecraft elapsed time objects

Each device (that maintains time information) shall implement one *Local SCET Set* and one *Local SCET Get* object in the Object Dictionary.

7.2.2 Time distribution protocol

The Time Producer shall map the Local SCET Get object to a dedicated *Spacecraft Time PDO* transmit PDO...to convey its local time to the time consumers… There shall be only one Spacecraft Time PDO in a particular system. The Time Consumers shall map the Local SCET Set objects to the Spacecraft Time PDO receive PDO.

Each time consumer shall map the *Local SCET Get* object … to a dedicated Local Time PDO...to convey its local time on the CAN Network.
Minimal implementation

9.2 Object dictionary

...it is acceptable that the CANopen objects...are hardcoded...

9.3 Minimal set CANopen objects

...only 4 Transmit and 4 Receive PDOs are implemented.

<table>
<thead>
<tr>
<th>Object</th>
<th>Function code (ID-bits 3:0)</th>
<th>COB-ID</th>
<th>Communication parameters at OD index (hexa)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDO 1 (transmit)</td>
<td>0011</td>
<td>180 + Node ID</td>
<td>181 - 1FF</td>
</tr>
<tr>
<td>PDO 1 (receive)</td>
<td>0100</td>
<td>200 + Node ID</td>
<td>201 - 27F</td>
</tr>
<tr>
<td>PDO 2 (transmit)</td>
<td>0101</td>
<td>280 + Node ID</td>
<td>281 - 2FF</td>
</tr>
<tr>
<td>PDO 2 (receive)</td>
<td>0110</td>
<td>300 + Node ID</td>
<td>301 - 37F</td>
</tr>
<tr>
<td>PDO 3 (transmit)</td>
<td>0111</td>
<td>380 + Node ID</td>
<td>381 - 3FF</td>
</tr>
<tr>
<td>PDO 3 (receive)</td>
<td>1000</td>
<td>400 + Node ID</td>
<td>401 - 47F</td>
</tr>
<tr>
<td>PDO 4 (transmit)</td>
<td>1001</td>
<td>480 + Node ID</td>
<td>481 - 4FF</td>
</tr>
<tr>
<td>PDO 4 (receive)</td>
<td>1010</td>
<td>500 + Node ID</td>
<td>501 - 57F</td>
</tr>
<tr>
<td>NMT Error Control</td>
<td>1110</td>
<td>700 + Node ID</td>
<td>701 - 77F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object</th>
<th>Function code (ID-bits 3:0)</th>
<th>Resulting COB-IDs (hexa)</th>
<th>Communication parameters at OD index (hexa)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNC</td>
<td>0001</td>
<td>80</td>
<td>1005, 1006, 1007</td>
</tr>
</tbody>
</table>
Minimal implementation

9.4.1 Minimal set protocol definitions

Communication between master and slave nodes shall be…

1. Transmission of configuration data or commands to a slave, is called unconfirmed command.

2. Start of a data transmission from slave is called telemetry request.

… data bytes shall contain the command itself… to identify the process to be performed by the slave.
Minimal implementation

1400h = RPDO1
281h = TPDO2 + id 1

1801h = TPDO2
281h = TPDO2 + id 1

Figure 9-1: Unconfirmed Command exchange overview (example with PDO1)
9.4.3 Minimal set protocol data transmission

Data transmission exchange shall be triggered by either:

1. A Telemetry Request message
2. Or a SYNC message

When telemetry request data bytes are used...(they) shall contain the telemetry register(s) to be returned...
Minimal implementation

1401h = RPDO2
302h = RPDO2 + id 2

1801h = TPDO2
302h = TPDO2 + id 2

0110 = RPDO2
0101 = TPDO2

1801h = TPDO2
282h = TPDO2 + id 2

1401h = RPDO2
282h = TPDO2 + id 2

Figure 9-2: Telemetry request exchange overview (example with PDO2)
## Prototype implementation – CANopen objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Function</th>
<th>COB-ID</th>
<th>Function</th>
<th>COB-ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPDO1</td>
<td>Local SCET Get</td>
<td>TPDO1 + slave id</td>
<td>Spacecraft SCET Get</td>
<td>TPDO1 + master id</td>
</tr>
<tr>
<td>RPDO1</td>
<td>Local SCET Set</td>
<td>TPDO1 + master id</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPDO2</td>
<td>-</td>
<td>-</td>
<td>Send TC</td>
<td>RPDO2 + slave id</td>
</tr>
<tr>
<td>RPDO2</td>
<td>Receive TC</td>
<td>RPDO2 + slave id</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TPDO3</td>
<td>Send TM</td>
<td>TPDO3 + slave id</td>
<td>Send TM_REQ</td>
<td>RPDO3 + slave id</td>
</tr>
<tr>
<td>RPDO3</td>
<td>Receive TM_REQ</td>
<td>RPDO3 + slave id</td>
<td>Receive TM</td>
<td>TPDO3 + slave id</td>
</tr>
<tr>
<td>SYNC</td>
<td></td>
<td></td>
<td>Send SYNC</td>
<td>80h</td>
</tr>
<tr>
<td>HB</td>
<td>Send HB</td>
<td>700h + slave id</td>
<td>Send HB</td>
<td>700h + master id</td>
</tr>
</tbody>
</table>
## Prototype implementation – PDO data field

<table>
<thead>
<tr>
<th>PDO data field</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function code</td>
<td>Data</td>
</tr>
<tr>
<td>2 Bytes</td>
<td>0 to 6 Bytes</td>
</tr>
</tbody>
</table>

65536 TCs requests + 6 bytes of data each
65536 TM REQUESTS
Prototype implementation - scenario

Master sends HB message every 250ms, fixed
Master sends SYNC every 5 sec, fixed

Master loops:
  - sends SCET PDO every ~10 sec
  - sends dummy TC every ~0.1 sec
  - sends dummy TM REQ every ~0.1 sec

Slave toggles LED on HB
Slave toggles LED on SYNC
Slave toggles LED on SCET
Slave responds to TM REQ with dummy TM
Prototype implementation - transceiver

- Texas Instruments SN65HVD23x 3.3-V CAN Bus Transceivers
- Fully ISO11898-2 compliant, supports 1 Mbps
- 3.3V power supply
- In high-impedance when unpowered
Prototype 1 - overview

• 8-bit Atmel AVR
• Atmega16m1
• C language
• Selective bus architecture
Prototype 1 – code snippets

```c
int8_t canopen_sendpdo(const uint8_t pdo_index)
{
    struct can_frame_t can_frame;
    uint16_t tpdo_map;
    uint32_t map;
    uint8_t index;
    uint8_t subindex;
    uint8_t i;
    uint8_t j;

    /* Check that pdo_index is within valid range for TPDO*/
    assert ((pdo_index >= CANOPEN_TPDO_INDEX) &&
            (pdo_index < CANOPEN_TPDO_MAP_INDEX));

    /* Check that current canopen state allows sending of tpdo */
    if (canopen_get_state() != canopen_state_OPERATIONAL)
    {
        return -1;
    }

    i = canopen_find_index(pdo_index);
    /* Set COB-ID */
    can_frame.id = (struct canopenpdo_t*)canopen_od[i].object->cob_id;
    /* Set DLC */
    tpdo_map = canopen_find_index(pdo_index + 0x200);
    can_frame.dlc = (struct canopenpdo_map_t *)
        canopen_od[tpdo_map].object->number_of_entries;

    for (i = 0; i < can_frame.dlc; i++)
    {
        map = (struct canopenpdo_map_t *)canopen_od[tpdo_map].object->map[i];
        index = (map >> 16);
        subindex = (map >> 8) & 0xFF;

        j = canopen_find_index(index);
        can_frame.data[i] = (struct canopenpdo_t *)
            canopen_od[j].object->var[subindex];
    }

    can_send(CAN_MCB_TX, &can_frame);
    return 0;
}

void can_configure_receive(const uint8_t mcb,
                           const struct can_frame_t* can_frame)
{
    uint8_t page_saved;

    assert (mcb < CAN_MCB_MAX);

    /* Enable reception of CAN frames from the network master */
    page_saved = CAMPAGE;
    CAMPAGE = mcb << MOENB0; /* use MCB for Rx and auto-increment data index */
    CANSTM = 0x00; /* clear all MCB status flags */
    CANIDM = (i << RIIMSK) | (i << IDEMSK); /* set mask for rtr and ide */
    CANIDM = 0x00;
    CANIDM = (can_frame->id_mask & 0x07) << 3;
    CANIDM = can_frame->id_mask >> 3;
    CANIDT = 0x00; /* clear rtr and ide bits */
    CANIDT = 0x00;
    CANIDT = (can_frame->id & 0x07) << 5;
    CANIDT = can_frame->id >> 3;
    CANCMOB = (i << CONCMOB1) | (can_frame->dlc << DLC0); /* enable reception */
    CAMPAGE = page_saved;
    CANIE1 = 0; /* compatibility with future chips */
    CANIE2 = (i << mcb); /* Enable receive MCB interrupts */
    CANGIE = (i << ENIT) | (i << ENRX); /* Enable interrupts: global, receive */
}
```

ECSS-CAN for CubeSats

2017-06-16
Prototype 2 - overview

- 32-bit ARM
- STM32F405RGT6
- Micropython language
- Parallel bus architecture
from pyb import LED
from pyb import delay
led = LED(1)
while True:
    led.toggle()
    delay(1000)
Prototype 2 – code snippets

```python
from .frame import DataFrame

class Fdo:
    def __init__(self, node,
                 tpdo_index, tpdo_map_index, rpdo_index, rpdo_map_index):
        self.node = node
        self.tpdo_index = tpdo_index
        self.tpdo_map_index = tpdo_map_index
        self,rpdo_index = rpdo_index
        self,rpdo_map_index = rpdo_map_index

    def transmit(self, node_id):
        """Send a TPDO."""
        cob_id = self.node.od[self.tpdo_index][1].value + node_id
        map_record = self.node.od[self.tpdo_map_index]
        no_of_mappings = map_record[0].value

        data = []
        # go through all mapping entries
        for i in range(1, no_of_mappings + 1):
            s = map_record[i].value  # mapping is recorded as string
            od_index = int(s[0:4], 16)  # extract od index from hex string
            od_subindex = int(s[4:8], 16)
            length = int(s[8:12], 16)
            octets = int(length / 8)
            # read the value as stored in the od at given location
            val = self.node.od[od_index][od_subindex].value
            # append the value byte-wise to data list
            for octet in range(octets):
                # append data with LSB first
                data.append((val >> octet * 8) & 0xff)

        self.frame = DataFrame(frame_id=cob_id, data=data)
        self.node.network.send_frame(self.frame)
```
Prototype 2 – code snippets

```python
import time
import micropython
from canopen import Bus, Network, Node, MntSlave
import slave_defs
micropython.alloc_emergency_exception_buf(100)

# define main and redundant bus
bus_a = Bus(1, mode=pyb.CAN.NORMAL)
buses = Bus(2, mode=pyb.CAN.NORMAL)

# create network
network = Network()
network.connect(bus_a, bus_b)
network.set_active_bus(bus_a)

# create slave node from object dictionary
slave_node = Node(slave_defs.eds.cfg)
network.add_node(slave_node)

# create SMB slave and start HB reception
mmt_slave = MntSlave(slave_node)
mmt_slave.heartbeat.start()

# create SYNC object and associate process function
mmt_slave.sync.start()

def sync_function():
    print("info: sync received")

mmt_slave.sync.callback = sync_function

# define message reception filter
FIFO_0 = 0
FILTERBANK_0 = 0
FILTERBANK_1 = 1
network.active_bus.can.setfilter(
    FILTERBANK_0, pyb.CAN.LIST16, FIFO_0, (0x00, 0x701, 0x131, 0x000))
network.active_bus.can.setfilter(
    FILTERBANK_1, pyb.CAN.LIST16, FIFO_0, (0x282, 0x302, 0x000, 0x000))

try:
    while True:
        if network.active_bus.can.any(FIFO_0):
            message_id, _, _, can_data = network.active_bus.can.recv(0)

            if message_id == 0x701:
                mmt_slave.heartbeat.received()
            elif message_id == 0x080:
                mmt_slave.sync.received()
            elif message_id == 0x181:
                SCET = int.from_bytes(can_data[8:8], 'little')
                SCET_ms = int.from_bytes(can_data[0:3], 'little')
                print("info: received SCET {} sec + 1/256 sec".format(SCET, SCET_ms))
            elif message_id == 0x252:
                # react upon TC
                # ...

                # statistics
                to_received += 1
                if to_received % 100 == 0:
                    to_code = int.from_bytes(can_data[0:2], 'little')
                    to_data = int.from_bytes(can_data[2:8], 'little')
                    print("info: received 100 TCs, last with code {} and data {}".format(to_code, to_data))
            elif message_id == 0x252:
                tm_req_code = int.from_bytes(can_data[0:2], 'little')
                # send out TM reply
                slave_nodepdo[3].transmit(slave_node pdo, 1)
                print("info: send TM reply")
                print(TM_reply)
                # statistics
                tm_req_received += 1
                if tm_req_received % 500 == 0:
                    print("info: received and replied 500 TM REQs, last with code {}").format(tm_req_code)

except KeyboardInterrupt:
    mmt_slave.heartbeat.stop()
mmt_slave.sync.stop()
network.active_bus.can.rxcallback(0, None)
```

ECSS-CAN for CubeSats

2017-06-16
Prototype 2 – code snippets

```python
# create object dictionary
od = canopen.ObjectDictionary()

# define node id
od.node_id = 2

# define standard entries
od.add_object(0x1008, canopen.Variable(5000000, "Communc. cycle period (usec)"))
od.add_object(0x1016, canopen.Variable(2500, "Consumer heartbeat time (msec)"))

# Redundancy Management
rec = canopen.Record("Redundancy management")
rec.add_member(1, canopen.Variable(1, "Befault"))
rec.add_member(2, canopen.Variable(5, "Ttoggle"))
rec.add_member(3, canopen.Variable(10, "Ntoggle"))
rec.add_member(4, canopen.Variable(0, "Coggle"))
od.add_object(0x2000, rec)

# RPDO3 for TMREQUEST reply
rec = canopen.Record("RPDO3 parameter")
rec.add_member(0, canopen.Variable(2))
rec.add_member(1, canopen.Variable(canopen.COB_ID_RPDO_3))
rec.add_member(2, canopen.Variable(254, "transmission type"))
od.add_object(0x1802, rec)
rec = canopen.Record("RPDO3 mapping")
rec.add_member(0, canopen.Variable(2))
rec.add_member(1, canopen.Variable("61000110", "TM request mapping"))
rec.add_member(2, canopen.Variable("61000250", "TM data mapping"))
od.add_object(0x1A02, rec)
tm_request = canopen.Variable(0x0001)
tm_data = canopen.Variable(0x000000000000)
rec = canopen.Record("Reply TM REQUEST")
rec.add_member(0, canopen.Variable(2))
rec.add_member(1, tm_request)
rec.add_member(2, tm_data)
od.add_object(0x6100, rec)
```
Conclusion

• Both implementation work satisfactorily
• Needs to be tested with several slave nodes and data processing

• All code is made available open source and free of charge
• Python is great for prototyping
• Embedded C is needed for constraint environments

• ECSS minimal implementation needs further clarification, e.g. Master Redundancy Management, PDO exchange
librecube.net → Contribute → Work Packages → code repository

Feedback in the forum, mailing list, or in repository