CANopenNode Tutorial

Janez Paternoster
Cesta v Zajčjo Dobravo 14a
Ljubljana
Slovenia, EU
janez.paternoster@siol.net

1. December 2005

Contents

1. Introduction

With this tutorial you will make working CANopen network. CANopenNode, Open Source Library will be used. It will be shown how to: make a generic input/output device, use Process Data objects (transmission and mapping), use retentive variables in Object Dictionary, make own program for smart devices, use custom CAN message for NMT master, use emergency message for custom errors, etc.

In later chapter it will be shown, how to configure network with `panel_with_PIC+LCD+keypad` or `Web_interface_with_SC1x` examples (both are included with CANopenNode) or with standard commercial configuration tool.

In this tutorial are described most of CANopenNode features. It is quite a lot of them, so many information is in this tutorial. Source code is tested, so user should not have problems with it. If there are problems or questions, please contact the author.

1.1 Requirements

1. CANopenNode source v1.10, download from http://sourceforge.net/projects/canopennode/ ;
2. MPLAB IDE (download from http://www.microchip.com, free);
3. MPLAB C18 V3.00+ (download from http://www.microchip.com, free);
4. Three boards with microcontroller PIC18F458 (or other PIC18F with CAN) and CAN transceiver;
5. PIC programmer (MPLAB ICD2 from http://www.microchip.com works fine).
6. Knowledge of using MPLAB IDE and MPLAB C18.
7. Knowledge of CANopen protocol (online training at "http://www.can-cia.org/canopen/" or "http://www.esacademy.com/myacademy/", book: "http://www.canopenbook.com/").
8. Recommended: CANopen configuration tool like `panel_with_PIC+LCD+keypad` (part of CANopenNode) or `Web_interface_with_SC1x` (also part of CANopenNode) or any other commercial CANopen configuration tool.

1.2 Description

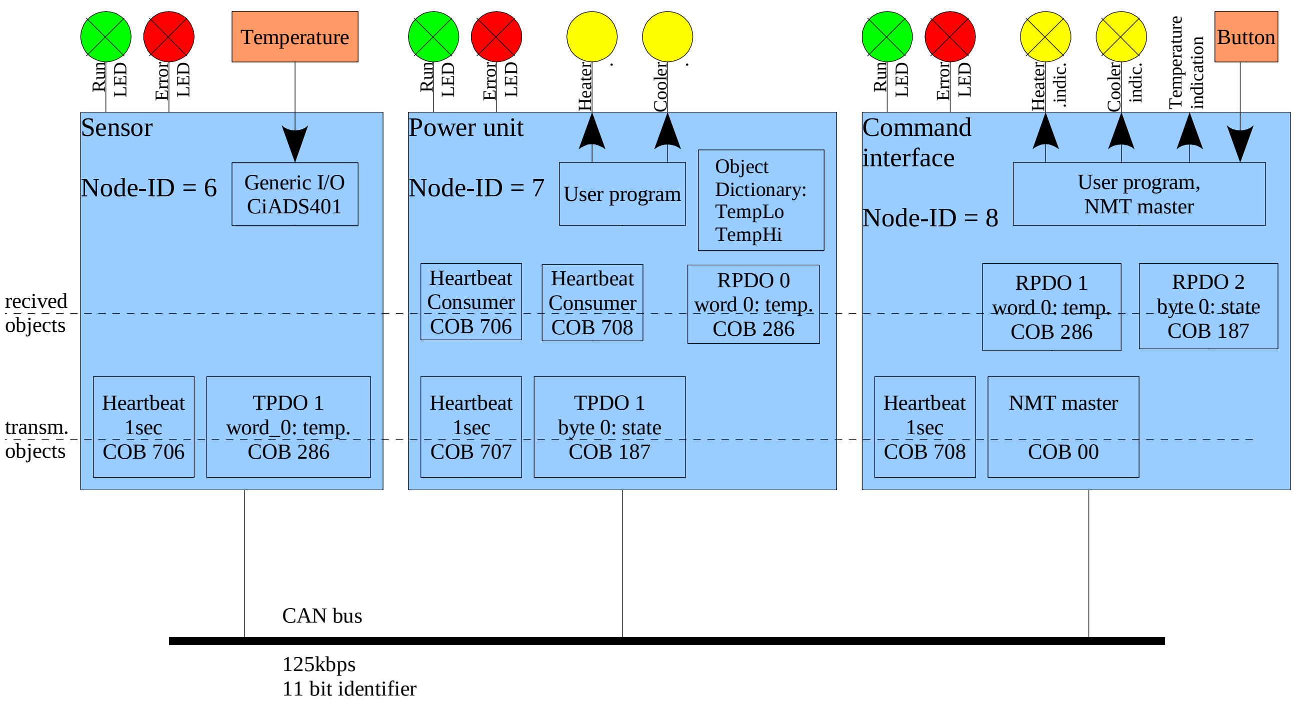
Network represents air-condition unit with remote sensor and command interface. It consists from three CANopen devices (Picture 1.1):

1. **Sensor:** stand alone temperature sensor:
    - Based on generic Input/Output device profile (CiADS401).
    - Transmit sensed temperature on every change of state and periodically with event timer (TPDO 1).
    - Produce Heartbeat every second.

2. **Power unit:** air-condition unit made from heater and cooler:
    - Receive temperature from Sensor (RPDO 0).

- TempLo and TempHi variables are in Object Dictionary. They can be accessed and changed with CANopen configuration tool via SDO communication objects. Variables are retentive (keep value after power off). Units are [° Celsius].
- Turn cooler or heater ON or OFF. Decision is based on Temperature from sensor, TempLo and TempHi. If device is not in operational state, cooler and heater will be powered off.
- Transmit state of cooler and heater on every change of state and periodically with event timer (TPDO 0).
- Produce Heartbeat every second.
- Monitor heartbeats from sensor and command interface.

3. **Command interface:** one button and two led diodes, optionally LCD display:
   - Receive temperature from Sensor (RPDO 0) and display it on LCD (optional).
   - Receive state of cooler and heater from Power unit (RPDO 1) and display it on LED diodes.
   - NMT master in connection with button. If button is pressed, all network will be set to pre-operational, and if button is pressed again, all network will be set to operational NMT state. If button is pressed for longer time (5 seconds), all nodes on network will reset.
   - Produce Heartbeat every second.

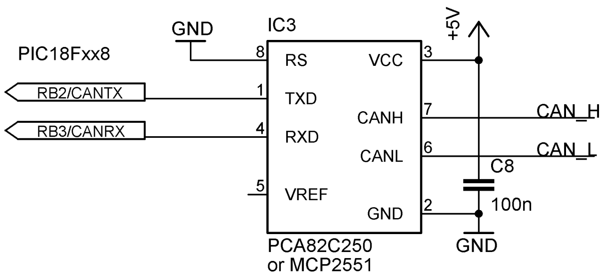


Picture 1.1 – Sample CANopen network

Besides communication objects described above, each node uses also SDO, Emergency and NMT (slave) communication objects.

2. Hardware

Hardware is made of Microchip PIC18F458 microcontroller, CAN transciever and some status led diodes or buttons. CAN transceiver Microchip MCP 2551 or Philips PCA82C250 can be used (see picture 2.1). Oscillator 8MHz with PLL x 4 mode will be used to get 32 MHz on PIC. If Other oscillator frequency or other pins for CAN diodes are used, *CO_driver.h* file has to be edited.



Picture 2.1 – CAN transceiver connected to PIC microcontroller

<u>Detailed description:</u>

1. **Sensor**:
   - Green LED diode between RB4 and GND (CAN Run led),
   - Red LED diode between RB5 and GND (CAN Error led),
   - 5kΩ potentiometer between +5V, GND and RA0 (instead of temperature sensor).

2. Power unit:

- Green LED diode between RB4 and GND (CAN Run led);
- Red LED diode between RB5 and GND (CAN Error led);
- Yellow LED diode between RC4 and GND (Cooler on);
- Yellow LED diode between RC5 and GND (Heater on).

3. Command interface:

- Green LED diode between RB4 and GND (CAN Run led);
- Red LED diode between RB5 and GND (CAN Error led);
- Yellow LED diode between RC4 and GND (Cooler state) ;
- Yellow LED diode between RC5 and GND (Heater state);
- Button between RC6 and VCC with pulldown resistor (NMT master);
- LCD to display temperature (not implemented here, only variable is available).

3. Program microcontrollers

Knowledge of using MPLAB IDE and MPLAB C18 is required. Both programs must be properly installed and configured.

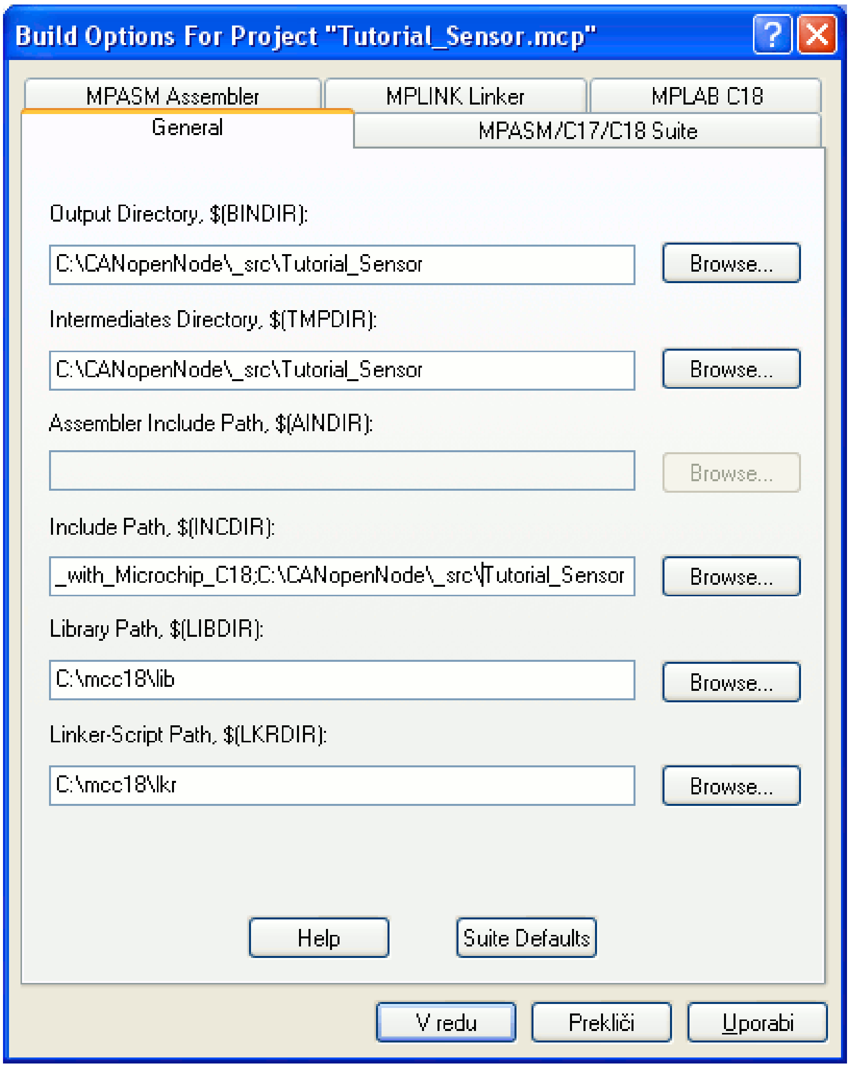
3.1 Source files

First download CANopenNode-V1.10.zip from http://sourceforge.net/projects/canopennode/ . Unzip the file on C: disk.

Three tutorial projects for Microchip MPLAB C18 are already made: *Tutorial_Sensor, Tutorial_Power* and *Tutorial_Command*. All three projects use the same library source files. Each can be opened with MPLAB IDE. Each project must be built without errors or warnings.

Folder settings in *Project>Build options>project>general* looks something like on picture 3.1. Complete Include Path, not visible from picture, is:

C:\mcc18\h;C:\CANopenNode_src\CANopen;C:\CANopenNode_src\CANopen\PIC18_with_Microchip_C18;C:\CANopenNode_src\Tutorial_Sensor



Picture 3.1 – Folder settings

Each project contains following files:

- src_\lesser.txt...license file
- src_\CANopen\CANopen.h..main header
- src_\CANopen\CO_errors.h...definitions for errors
- src_\CANopen\CO_stack.c..main code
- src_\CANopen\CO_OD.txt...Obj. Dict. description
- src_\CANopen\PIC18_with_Microchip_C18\CO_driver.h................Driver header
- src_\CANopen\PIC18_with_Microchip_C18\CO_driver.c................Processor specific code
- src_\CANopen\PIC18_with_Microchip_C18\main.c.........................main() and interrupts
- src_\CANopen\PIC18_with_Microchip_C18\memcpyram2flash.h....write to flash memory
- src_\CANopen\PIC18_with_Microchip_C18\memcpyram2flash.c....write to flash memory
- src_\CANopen\PIC18_with_Microchip_C18\CONFIG18f458.c........configuration - optional
- src_\CANopen\PIC18_with_Microchip_C18\18f458i.lkr..................default linker script
- src_\Tutorial_xxx\CO_OD.h..object dictionary header
- src_\Tutorial_xxx\CO_OD.c...object dictionary
- src_\Tutorial_xxx\user.c...user code
- src_\Tutorial_xxx\Tutorial_xxx.eds...electronic data sheet

3.2 Oscillator and CAN diodes settings

By default all microcontrollers are running on 32 MHz. This is achieved with 8MHz quartz and PLLx4 mode in PIC18fxxx microcontroller. If CONFIG18f458.c file is used, PLLx4 mode is set in line '*#pragma config OSC = HSPLL*', otherwise in MPLAB IDE go to *Configure>Configuration bits* and set oscillator: HS-PLL Enabled. If PLLx4 mode is set or reset (changed) voltage must be removed from microcontroller and change will take effect.

For different oscillator open *CO_driver.h* and change macro CO_OSCILATOR_FREQ. Alternative is adding a macro definition into each project file: *Project>Build options>project>MPLAB C18>Macro Definitions>Add...* and write '*CO_OSCILATOR_FREQ xx*' where xx is oscillator frequency and can be different for each project.

By default Green LED diode (CAN Run led) is on RB4 and Red LED diode (CAN Error led) is on RB5. Pins are defined in file *CO_driver.h* on same section as for oscillator. Pins can be changed there. If leds needs to be disabled, disable macros. Comment them for example:

```
#define PCB_RUN_LED_INIT()      //{TRISBbits.TRISB4 = 0; PORTBbits.RB4 = 0;}
#define PCB_RUN_LED(i)          //PORTBbits.RB4 = i
```

3.3 Programming Sensor

Sensor is based on Example_generic_IO. All differences from it will be shown. Example_generic_IO is based on _blank_project and CiADS401 – CANopen device profile for generic Input/Output modules. Sensor will use ADC converter and PDO object to send analog value over the the CAN.

Here is short extraction from CiADS401:

Object dictionary, index 0x6000: Read Digital Input – 64 bits mapped to TPDO 0.

Object dictionary, index 0x6200: Write Digital Output – 64 bits mapped to RPDO 0.

Object dictionary, index 0x6401: Read Analog Input – 12*16 bit value mapped to TPDO 1...3.

Object dictionary, index 0x6411: Write Analog Output – 12*16 bit value mapped to RPDO 1...3.

Here will be used only two bytes of TPDO 1, mapped from index 0x6401, subindex 0x01.

Open Project *Tutorial_Sensor.*

3.3.1 Setup Project – CO_OD.h file

Code is shown in fixed width fonts. Differences from Example_generic_IO are shown in green color. For better understanding comments in source code may be read for each line.

Code for section *Setup CANopen:*

```
#define CO_NO_SYNC           0
#define CO_NO_EMERGENCY      1
#define CO_NO_RPDO           0
#define CO_NO_TPDO           2
#define CO_NO_SDO_SERVER     1
#define CO_NO_SDO_CLIENT     0
#define CO_NO_CONS_HEARTBEAT 0
#define CO_NO_USR_CAN_RX     0
#define CO_NO_USR_CAN_TX     0
#define CO_MAX_OD_ENTRY_SIZE 20
#define CO_SDO_TIMEOUT_TIME  10
#define CO_NO_ERROR_FIELD    8
#define CO_PDO_PARAM_IN_OD
#define CO_PDO_MAPPING_IN_OD
#define CO_TPDO_INH_EV_TIMER
#define CO_VERIFY_OD_WRITE
#define CO_OD_IS_ORDERED
#define CO_SAVE_EEPROM
#define CO_SAVE_ROM
```

In this section can be defined number of specific objects used in CANopenNode. Same features may be disabled.

In our case SYNC object will not be used, PDOs will not be received, TPDO 0 will be unused (Digital inputs), TPDO 1 (two bytes) will be used for Analog value from temperature sensor, other nodes will not be monitored.

Code for section *Device profile for Generic I/O*:

```
//#define CO_IO_DIGITAL_INPUTS     //4 * 8 digital inputs
//#define CO_IO_DIGITAL_OUTPUTS    //4 * 8 digital outputs
#define CO_IO_ANALOG_INPUTS      //8 * 16bit analog inputs
//#define CO_IO_ANALOG_OUTPUTS     //2 * 16bit analog outputs
```

Code for section *Default values for object dictionary*:

```
#define ODD_PROD_HEARTBEAT    1000
...
#define ODD_ERROR_BEH_COMM    0x01
#define ODD_NMT_STARTUP       0x00000000L
```

Heartbeat object will be sent once per second.

If communication error bit in Error Register (index 1001) will be set and Device will be in Operational NMT state, device will stay Operational. Default behavior is that Communication error forces device to Pre-Operational NMT state.

On startup device will enter NMT Operational state.

Code for section *0x1800 Transmit PDO parameters*:

```
#define ODD_TPDO_PAR_COB_ID_0 0
#define ODD_TPDO_PAR_T_TYPE_0 255
#define ODD_TPDO_PAR_I_TIME_0 0
#define ODD_TPDO_PAR_E_TIME_0 0
#define ODD_TPDO_PAR_COB_ID_1 0
#define ODD_TPDO_PAR_T_TYPE_1 255
#define ODD_TPDO_PAR_I_TIME_1 1000
#define ODD_TPDO_PAR_E_TIME_1 60000
```

PDO 0 will newer be sent (because CO_IO_DIGITAL_INPUTS is disabled). PDO 1 COB-ID (11-bit CAN identifier) will be default – 0x280+Node-ID. Inhibit time for PDO 1 is 1000*100µs, so PDO 1 will newer be sent more often than 100 ms. PDO 1 will be send on change of state and every minute (60000 ms) – Transmission type is 255 – device profile specific.

Code for section *0x1A00 Transmit PDO mapping* for PDO 1:

```
#define ODD_TPDO_MAP_1_1      0x64010110L
#define ODD_TPDO_MAP_1_2      0x64010200L
#define ODD_TPDO_MAP_1_3      0x64010300L
#define ODD_TPDO_MAP_1_4      0x64010400L
#define ODD_TPDO_MAP_1_5      0x00000000L
#define ODD_TPDO_MAP_1_6      0x00000000L
#define ODD_TPDO_MAP_1_7      0x00000000L
#define ODD_TPDO_MAP_1_8      0x00000000L
```

Only one Analog value will be sent in TPDO 1, so on CAN bus it will have only two bytes of data. PDO length is calculated automatically from mapping in CANopenNode. Data are copied from variable *ODE_Read_Analog_Input[0]* (located in OD, index=0x6401, subindex=1, length=0x10 bytes) on every change of state. To see how it works, below is example code from file user.c, section *Write TPDOs*:

```
        if(CO_TPDO_InhibitTimer[1] == 0 && (
           CO_TPDO(1).WORD[0] != ODE_Read_Analog_Input[0])){

           CO_TPDO(1).WORD[0] = ODE_Read_Analog_Input[0];
           if(ODE_TPDO_Parameter[1].Transmission_type >= 254)
              CO_TPDOsend(1);
        }
```

Code for section *Default values for user Object Dictionary Entries*:

```
   #define ODD_CANnodeID    0x06
   #define ODD_CANbitRate   3
```

Node-ID for sensor is 6 and CAN bit-rate is 125 kbps.

3.3.2 Interface with application – User.c file

Code is shown in fixed width fonts. Differences from Example_generic_IO are shown in green color.

Code for head of file:

```
#include <adc.h>
```

Change *User_Init* function:

```
void User_Init(void){
   ...

   #ifdef CO_IO_ANALOG_INPUTS
      OpenADC(ADC_FOSC_RC & ADC_RIGHT_JUST & ADC_1ANA_0REF,
              ADC_CH0 & ADC_INT_OFF);
      ConvertADC();
      ODE_Read_Analog_Input[0] = 0;
      ODE_Read_Analog_Input[1] = 0;
      ODE_Read_Analog_Input[2] = 0;
      ODE_Read_Analog_Input[3] = 0;
      ODE_Read_Analog_Input[4] = 0;
      ODE_Read_Analog_Input[5] = 0;
      ODE_Read_Analog_Input[6] = 0;
      ODE_Read_Analog_Input[7] = 0;
   #endif

   ...
}
```

Change *User_Process1msIsr* function, section *Read from Hardware*:

```
   //CHANGE THIS LINE -> ODE_Read_Digital_Input.BYTE[0] = port_xxx
   //CHANGE THIS LINE -> ODE_Read_Digital_Input.BYTE[1] = port_xxx
   //CHANGE THIS LINE -> ODE_Read_Digital_Input.BYTE[2] = port_xxx
   //CHANGE THIS LINE -> ODE_Read_Digital_Input.BYTE[3] = port_xxx
```

```
if(BusyADC() == 0){
   ODE_Read_Analog_Input[0] = ReadADC() >> 2;   //8 bit value
   ConvertADC();
}
```

Function is executed every millisecond.

Eds file is a text file and can be used with CANopen monitors. It has now some overhead, because we didn't use digital inputs, nor outputs.

That's all for sensor. You may read user.c file, because it is the connection point between user code and CANopenNode stack. Next time, we will start from 'blank' file. You may also inspect the differences between files in *Example_generic_IO* and *_blank_project* folders. Very suitable tool for this can be found on http://www.winmerge.org/ .

3.3.3 Compile, program and test

Compile project and program PIC. Now disconnect and reconnect PIC to power supply to make sure PLLx4 will be enabled. Don't connect to network yet. After power on *Green CAN run led* should light. This means, NMT operating state is Operational. *Red CAN error led* should flash once. This means, CAN bus is passive. If you make short on CAN_LO and CAN_HI signals *Red CAN error led* will light. This means, CAN bus is Off. Sensor will not leave Operational state because of Communication errors. In our case this is OK.

3.4 Program Power unit

Power unit is based on _blank_project. All differences from it will be shown. It will be some kind of controller, which will control Heater and Cooler according to the state of network, operational state, state of other nodes, temperature from remote sensor and internal retentive variables for high and low limit of temperature. It will be quite advanced smart device.

Open Project *Tutorial_Power*

3.4.1 Add variables into Object Dictionary – CO_OD.c file

Following variables will be added:

Index, Subindex	Name	Type	Memory type	Access type
0x3000, 0x00	TempLo	UNSIGNED 16	Flash	Read/Write
0x3001, 0x00	TempHi	UNSIGNED 16	Flash	Read/Write
0x3100, 0x00	Status	UNSIGNED 8	RAM	Read only
0x3200, 0x00	RemoteTemperature	UNSIGNED 16	RAM	Read/Write

TempLo is temperature in °Celsius, below which heater will be turned on, TempHi is temperature in °Celsius, above which cooler will be turned on. TempLo and TempHi are both retentive and have no connection with PDOs. They can be written from network side (by SDO client), but they can not be changed directly by program. During write microcontroller is frozen for ~20 ms. As alternative for retentive variables it is possible to use EEPROM in combination with RAM. For example see variable *PowerOnCounter*.

Status and RemoteTemperature are two RAM variables, which will be mapped to PDOs. Here are discussed different approaches for connection between PDOs and variables.

Status contains information about state of cooler an heater. It has own variable. Later will be shown code, which will determine change of state and then copy contents of variable to TPDO data and trigger PDO.

RemoteTemperature is temperature in °Celsius received from remote sensor. It uses memory from first two bytes of RPDO 0 data. So when RPDO arrives, RemoteTemperature can be read immediately without extra copying.

It is not the rule, that Status uses own variable. It could use memory from TPDO, similar as RemoteTemperature. It can then be triggered by Event Timer. But it would not be possible to determine change of state.

Sometimes it is necessary to use separate variable for RPDO. It can be danger, if program is in the middle of read of RPDO data, then RPDO arrives and processes with high priority interrupt, and after that program reads second half of data. Now program has half data from old RPDO and half data from new RPDO !!! So during read it is good to disable interrupt. In our case this is not necessary, because we use only one byte.

Example of safe read of RPDO from low priority timer interrupt function is here:

```
if(CO_RPDO_New(0)){
   CO_DISABLE_CANRX_TMR();
   ODE_Write_Digital_Output.DWORD[0] = CO_RPDO(0).DWORD[0];
   CO_RPDO_New(0) = 0;
   CO_ENABLE_CANRX_TMR();
}
```

Definition of variables with their default values. Code added in section *Manufacturer specific variables*:

```
/*0x3000*/ ROM UNSIGNED16     TempLo = 18;
               //below this temperature heater will be turned on
/*0x3001*/ ROM UNSIGNED16     TempHi = 25;
               //above this temperature cooler will be turned on
/*0x3100*/     tData1byte     Status = 0;
/*0x3200*/
           #define RemoteTemperature CO_RPDO(0).WORD[0]
```

Verify values at Object Dictionary write. Code added in function *CO_OD_VerifyWrite*:

```
   case 0x3000://TempLo
               if((*((unsigned int*)data) > 35) ||
                  (*((unsigned int*)data) >= TempHi))
                  return 0x06090031L;  //Value of param. written too high
               break;
   case 0x3001://TempHi
               if((*((unsigned int*)data) < 5) ||
                  (*((unsigned int*)data) <= TempLo))
                  return 0x06090032L;  //Value of param. written too low
               break;
```

Add variables to Object Dictionary. Be careful, indexes and subindexes must be ordered in whole CO_OD array. Code added in section *Manufacturer specific*:

```
   OD_ENTRY(0x3000, 0x00, ATTR_RW|ATTR_ROM, TempLo),
   OD_ENTRY(0x3001, 0x00, ATTR_RW|ATTR_ROM, TempHi),
   OD_ENTRY(0x3100, 0x00, ATTR_RO, Status),
   OD_ENTRY(0x3200, 0x00, ATTR_RWW, RemoteTemperature),
```

That's all for CO_OD.c file. In CO_OD.h will be declaration of above variables and PDO mapping. In User.c variables will be used and TPDO will be triggered.

3.4.2 Setup Project – CO_OD.h file

Code is shown in fixed width fonts. Differences from _blank_project are shown in green color.

Code for section *Setup CANopen:*

```
   #define CO_NO_SYNC            0
   #define CO_NO_EMERGENCY       1
   #define CO_NO_RPDO            1
   #define CO_NO_TPDO            1
   #define CO_NO_SDO_SERVER      1
   #define CO_NO_SDO_CLIENT      0
   #define CO_NO_CONS_HEARTBEAT  2
   #define CO_NO_USR_CAN_RX      0
   #define CO_NO_USR_CAN_TX      0
   #define CO_MAX_OD_ENTRY_SIZE  20
   #define CO_SDO_TIMEOUT_TIME   10
   #define CO_NO_ERROR_FIELD     8
   #define CO_PDO_PARAM_IN_OD
   #define CO_PDO_MAPPING_IN_OD
   #define CO_TPDO_INH_EV_TIMER
   #define CO_VERIFY_OD_WRITE
   #define CO_OD_IS_ORDERED
   #define CO_SAVE_EEPROM
   #define CO_SAVE_ROM
```

SYNC object will not be used, RPDO 0 (two bytes) will be received from sensor, TPDO 0 (one byte) will be used for reporting status, other two nodes will be monitored.

Code for section *Default values for object dictionary* will stay default.

Code for section *0x1016 Heartbeat consumer*:

```
   #define ODD_CONS_HEARTBEAT_0  0x000605DCL
   #define ODD_CONS_HEARTBEAT_1  0x000805DCL
```

Nodes with ID 6 and 8 will be monitored. If Heartbeat signal is absent for more than 1500 ms, Power Unit will switch off to Pre-Operational.

Code for section *0x1400 Receive PDO parameters*:

```
#define ODD_RPDO_PAR_COB_ID_0 0x40000286L
#define ODD_RPDO_PAR_T_TYPE_0 255
```

Node will directly receive PDO from sensor – COB-ID = 0x286.

Code for section *0x1600 Receive PDO mapping*:

```
#define ODD_RPDO_MAP_0_1      0x32000010L
#define ODD_RPDO_MAP_0_2      0x00000000L
#define ODD_RPDO_MAP_0_3      0x00000000L
#define ODD_RPDO_MAP_0_4      0x00000000L
#define ODD_RPDO_MAP_0_5      0x00000000L
#define ODD_RPDO_MAP_0_6      0x00000000L
#define ODD_RPDO_MAP_0_7      0x00000000L
#define ODD_RPDO_MAP_0_8      0x00000000L
```

Length of RPDO 0 is two bytes. PDO received must have the same length as defined in mapping, or error will be generated. RPDO data can also be accessed through Object Dictionary index 0x3200, subindex 0 (location of 'variable' RemoteTemperature).

Code for section *0x1800 Transmit PDO parameters*:

```
#define ODD_TPDO_PAR_COB_ID_0 0
#define ODD_TPDO_PAR_T_TYPE_0 254
#define ODD_TPDO_PAR_I_TIME_0 100
#define ODD_TPDO_PAR_E_TIME_0 1000
```

PDO COB-ID (this is 11-bit CAN identifier) will be default – 0x180+Node-ID. Inhibit time for PDO 0 is 100*100µs, so PDO will newer be sent more often than 10 ms. PDO 0 will be send on change of state and every second (1000 ms). Transmission type is 254 – manufacturer specific.

Code for section *0x1A00 Transmit PDO mapping*:

```
#define ODD_TPDO_MAP_0_1      0x31000008L
#define ODD_TPDO_MAP_0_2      0x00000000L
#define ODD_TPDO_MAP_0_3      0x00000000L
#define ODD_TPDO_MAP_0_4      0x00000000L
#define ODD_TPDO_MAP_0_5      0x00000000L
#define ODD_TPDO_MAP_0_6      0x00000000L
#define ODD_TPDO_MAP_0_7      0x00000000L
#define ODD_TPDO_MAP_0_8      0x00000000L
```

Length of TPDO 0 is one byte (8 bits). TPDO data can also be accessed through Object Dictionary index 0x3100, subindex 0 (location of 'variable' Status).

Code for section *Default values for user Object Dictionary Entries*:

```
#define ODD_CANnodeID     0x07
#define ODD_CANbitRate    3
```

Node-ID for sensor is 7 and CAN bit-rate is 125 kbps.

3.4.3 Interface with application – User.c file

Code is shown in fixed width fonts. Differences from Example_generic_IO are shown in green color.

Some definitions in the head of the file:

```
#include "CANopen.h"

/*0x3000*/ extern ROM   UNSIGNED16     TempLo;
/*0x3001*/ extern ROM   UNSIGNED16     TempHi;
/*0x3100*/ extern       tData1byte     Status;
/*0x3200*/
           #define RemoteTemperature CO_RPDO(0).WORD[0]
#define STATUS_BYTE     Status.BYTE[0]
#define STATUS_COOLER   Status.BYTEbits[0].bit0
#define STATUS_HEATER   Status.BYTEbits[0].bit1
#define DigOut_COOLER   PORTCbits.RC4
#define DigOut_HEATER   PORTCbits.RC5
#define ERROR_TEMP_LOW  ERROR_USER_4
```

Change *User_Init* function:

```
void User_Init(void){
   ODE_EEPROM.PowerOnCounter++;

   TRISCbits.TRISC4 = 0; PORTCbits.RC4 = 0;
   TRISCbits.TRISC5 = 0; PORTCbits.RC5 = 0;
}
```

Change *User_Process1msIsr* function:

```
void User_Process1msIsr(void){
   //Inhibit timer is necessary for Change of State transmission
   extern volatile unsigned int CO_TPDO_InhibitTimer[];

   if(CO_NMToperatingState == NMT_OPERATIONAL &&
      CO_HBcons_AllMonitoredOperational == NMT_OPERATIONAL){

      //control heater and cooler
      if(RemoteTemperature < TempLo){
         DigOut_COOLER = 0;
         DigOut_HEATER = 1;
      }
      else if(RemoteTemperature > TempHi){
         DigOut_COOLER = 1;
         DigOut_HEATER = 0;
      }
      else{
         DigOut_COOLER = 0;
         DigOut_HEATER = 0;
         }

      //send emergency message in case of low temperature
      if(RemoteTemperature < 5)
         ErrorReport(ERROR_TEMP_LOW, RemoteTemperature);
      else if(ERROR_BIT_READ(ERROR_TEMP_LOW))
         ErrorReset(ERROR_TEMP_LOW, RemoteTemperature);
```

```
      //Change of state PDO transmission
      if(CO_TPDO_InhibitTimer[0] == 0 && (
         CO_TPDO(0).BYTE[0] != STATUS_BYTE)){

         CO_TPDO(0).BYTE[0] = STATUS_BYTE;
         if(ODE_TPDO_Parameter[0].Transmission_type >= 254)
            CO_TPDOsend(0);
      }
   }

   else{
      //turn off everything
      DigOut_COOLER = 0;
      DigOut_HEATER = 0;
   }

   //write status
   STATUS_COOLER = DigOut_COOLER;
   STATUS_HEATER = DigOut_HEATER;
}
```

Function is executed every millisecond. As alternative *User_ProcessMain* function can be used. Anyway all code must be unblocking.

3.4.4 Electronic Data Sheet – Tutorial_Power.eds file

Eds file can be used with CANopen monitors for easy and clear setup of parameters in Object dictionary. It can be edited with commercial program. Because it is a text file, it can also be edited by hand, as is shown here. You may compare *Tutorial_Power.eds* with *_blank_project.eds* to see the difference.

3.4.5 Compile, program and test

Compile project and program PIC. Now disconnect and reconnect PIC to power supply to make sure PLLx4 will be enabled. Don't connect to network yet. After power on *Green CAN run led* should blink with 2,5 Hz frequency. This means, operating state is pre-operational. *Red CAN error led* should flash once. This means, CAN bus is passive. Heater and Cooler leds must not light.

Now you can connect CAN interfaces on Sensor and Power unit together (Picture 4.1).

After power on CAN run led on Sensor and on Power unit must light without blinking (operational state). Startup state is controlled with variable *NMT_Startup*, which is located on index 0x1F80. CAN error leds on Sensor and on Power unit must be off.

Heater and Cooler leds will not light, because Power Unit monitors also Command interface, which is not yet present.

If red led lights or flashes once, something is wrong with communication. This must not happen, if yes, check cables, termination resistors, oscillator frequencies, PLL setting, macros for oscillator frequency and CAN baud rates.

3.5 Program Command interface

Command interface is based on _blank_project. All differences from it will be shown. It will display state of heater and cooler and can display temperature. It will also have NMT master, which can change operational state of nodes.

Open Project *Tutorial_Command*

3.5.1 Setup Project – CO_OD.h file

Code is shown in fixed width fonts. Differences from _blank_project are shown in green color.

We will try to simplify this device as much as possible. Memory size will be reduced.

Code for section *Setup CANopen:*

```
#define CO_NO_SYNC              0
#define CO_NO_EMERGENCY         1
#define CO_NO_RPDO              2
#define CO_NO_TPDO              0
#define CO_NO_SDO_SERVER        1
#define CO_NO_SDO_CLIENT        0
#define CO_NO_CONS_HEARTBEAT    0
#define CO_NO_USR_CAN_RX        0
#define CO_NO_USR_CAN_TX        1
#define CO_MAX_OD_ENTRY_SIZE    4
#define CO_SDO_TIMEOUT_TIME     10
#define CO_NO_ERROR_FIELD       8
//#define CO_PDO_PARAM_IN_OD
//#define CO_PDO_MAPPING_IN_OD
//#define CO_TPDO_INH_EV_TIMER
//#define CO_VERIFY_OD_WRITE
#define CO_OD_IS_ORDERED
#define CO_SAVE_EEPROM
//#define CO_SAVE_ROM
```

SYNC object will not be used, two RPDOs will be used, one for reception of temperature from remote sensor and one for reception of status from Power unit. TPDOs and monitoring of other nodes will not be used.

One custom CANtx message will be used for creation of NMT master.

Maximum size of variable in Object Dictionary is limited to 4 bytes, so no segmented transfer for SDO will be used.

PDO parameters will not be in object dictionary. They will still exist as variables, but wont be accessible. Flash memory size will be reduced, because variables won't be added to CO_OD[] array.

No mapping will be used. Variables also won't exist. In this case, if we had TPDO, it's data length would be fixed to 8 bytes. Since we have RPDOs, any data length will be accepted.

Macro CO_SAVE_ROM is disabled, so SDO client won't be able to write to variables located in flash memory. For example *Producer_Heartbeat_Time* located at index 0x1017 won't be writable. For this reason we can also remove files *memcpyram2flash.h* and *memcpyram2flash.c* from project.

Code for section *Default values for object dictionary* will stay default.

Code for section *0x1400 Receive PDO parameters*:

```
#define ODD_RPDO_PAR_COB_ID_0 0x40000286L
#define ODD_RPDO_PAR_T_TYPE_0 255
#define ODD_RPDO_PAR_COB_ID_1 0x40000187L
#define ODD_RPDO_PAR_T_TYPE_1 255
```

Node will directly receive PDO from sensor – COB-ID = 0x286 and PDO from Power unit – COB-ID = 0x187.

Code for section *Default values for user Object Dictionary Entries*:

```
#define ODD_CANnodeID     0x08
#define ODD_CANbitRate    3
```

Node-ID for sensor is 8 and CAN bit-rate is 125 kbps.

3.5.2 Interface with application – User.c file

Code is shown in fixed width fonts. Differences from Example_generic_IO are shown in green color.

Some definitions in the head of the file:

```
#include "CANopen.h"

#define RemoteTemperature CO_RPDO(0).WORD[0]
#define STATUS_COOLER     CO_RPDO(1).BYTEbits[0].bit0
#define STATUS_HEATER     CO_RPDO(1).BYTEbits[0].bit1
#define DigOut_COOLER_IND PORTCbits.RC4
#define DigOut_HEATER_IND PORTCbits.RC5
#define DigInp_Button     PORTCbits.RC6
#define NMT_MASTER        CO_TXCAN_USER
```

Change *User_Init* function:

```
void User_Init(void){
   ODE_EEPROM.PowerOnCounter++;

   TRISCbits.TRISC4 = 0; PORTCbits.RC4 = 0;
   TRISCbits.TRISC5 = 0; PORTCbits.RC5 = 0;
}
```

Change *User_ResetComm* function:

```
void User_ResetComm(void){
   //Prepare variables for custom made NMT master message.
   //arguments to CO_IDENT_WRITE(CAN_ID_11bit, RTRbit/*usually 0*/)
   CO_TXCAN[NMT_MASTER].Ident.WORD[0] = CO_IDENT_WRITE(0, 0);
   CO_TXCAN[NMT_MASTER].NoOfBytes = 2;
   CO_TXCAN[NMT_MASTER].NewMsg = 0;
```

```
   CO_TXCAN[NMT_MASTER].Inhibit = 0;
}
```

Change *User_Process1msIsr* function:

```
unsigned int Temperature = 0;

void User_Process1msIsr(void){
   static unsigned int DeboucigTimer = 0;

   if(DigInp_Button) DeboucigTimer++;
   else DeboucigTimer = 0;

   switch(DeboucigTimer){
      case 1000:  //button is pressed for one seccond
         if(CO_NMToperatingState == NMT_OPERATIONAL){
            CO_TXCAN[NMT_MASTER].Data.BYTE[0] = NMT_ENTER_PRE_OPERATIONAL;
            CO_NMToperatingState = NMT_PRE_OPERATIONAL;
         }
         else{
            CO_TXCAN[NMT_MASTER].Data.BYTE[0] = NMT_ENTER_OPERATIONAL;
            CO_NMToperatingState = NMT_OPERATIONAL;
         }
         CO_TXCAN[NMT_MASTER].Data.BYTE[1] = 0;  //all nodes
         CO_TXCANsend(NMT_MASTER);
         break;
      case 5000:  //button is pressed for 5 seconds
         //reset all remote nodes
         CO_TXCAN[NMT_MASTER].Data.BYTE[0] = NMT_RESET_NODE;
         CO_TXCAN[NMT_MASTER].Data.BYTE[1] = 0;  //all nodes
         CO_TXCANsend(NMT_MASTER);
         break;
      case 5010: //button is pressed for 5 seconds + 10 milliseconds
         CO_Reset(); //reset this node
   }

   //display variables
   DigOut_COOLER_IND = STATUS_COOLER;
   DigOut_HEATER_IND = STATUS_HEATER;
   Temperature = RemoteTemperature;
}
```

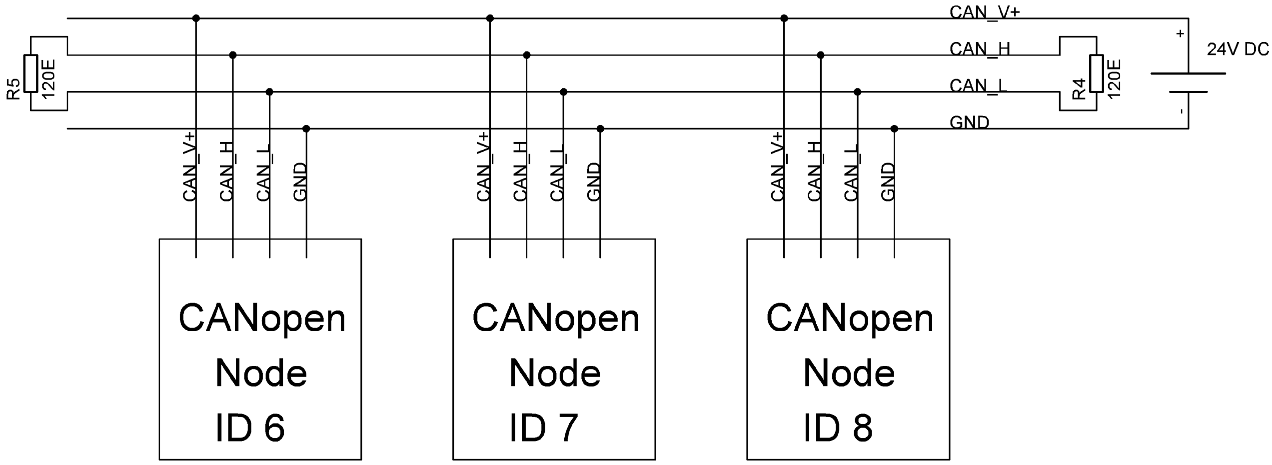
Function is executed every millisecond.

3.3.3 Compile, program and test

Compile project and program PIC. Now disconnect and reconnect PIC to power supply to make sure PLLx4 will be enabled. Don't connect to the network yet. After power on *Green CAN run led* should blink with 2,5 Hz frequency. This means, operating state is pre-operational. *Red CAN error led* should flash once. This means, CAN bus is passive.

4. Connect and test the network.

Now connect all three nodes to CANopen network as seen on picture 4.1. Power supply is optional. You should use twisted pair cables.



Picture 4.1 – Connection of three nodes to CANopen network.

After power on CAN run led on all nodes must light. CAN error leds must be off.

Network is now operational. You may turn potentiometer on sensor left and right. Cooler and Heater Leds must turn On and Off on both: Power unit and Command. If you press button for one second, whole network will switch between operational (green led lights) and pre-operational (green led is blinking). If you press button for five seconds, all nodes on network will reset.

Heater and Cooler on Power Unit works only, if all three nodes are in Operational state.

You may see, that led diode for heater or cooler lights on Command Interface, but on Power Unit do not and the network is in Pre-Operational state. In Command Interface was made no extra checking. You may see, how this problem is solved in Power Unit, where Heartbeat consumer is used.

Now try to power off sensor. Power unit enters pre-operational and turn off Cooler and Heater. Red led flashes twice – Heartbeat consumer error. Emergency massage 8130 was sent. For information about emergency message read *CO_errors.h* file. History of errors can be read from Object Dictionary, Index 1003.

User Emergency message is also sent, if temperature goes below 5°C. When this 'error' is solved, another Emergency is sent, which indicates No Error.

Below is sample trace from operational network. First row is time stamp in [s], second row is 11-bit CAN-ID in hex format, Third is data length, next rows are CAN data bytes in hex format .

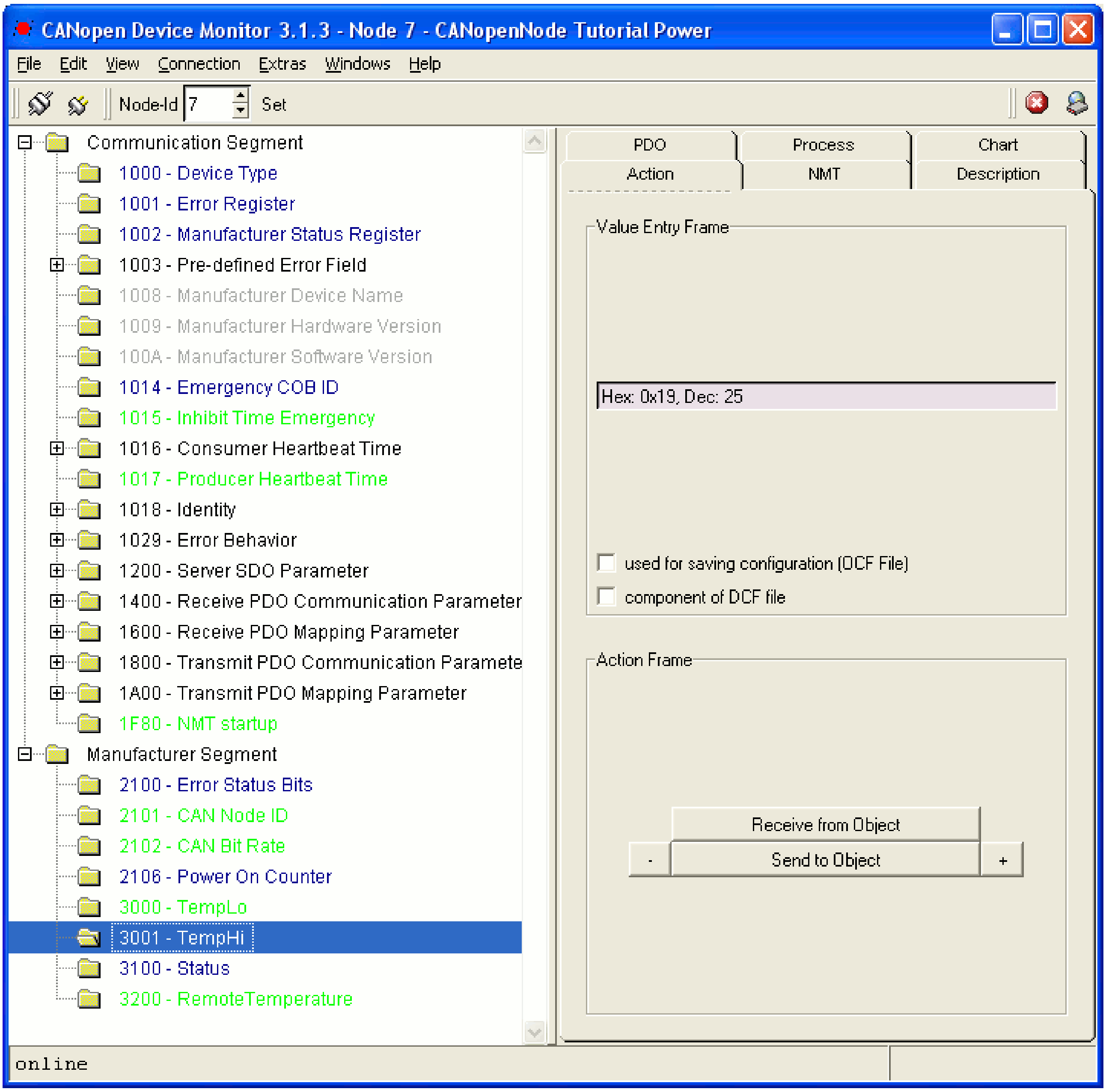
```
195.878   0x708   0x01     0x00         //Command Interface Bootup
195.879   0x707   0x01     0x00         //Power Unit Bootup
195.880   0x706   0x01     0x00         //Sensor Bootup
195.881   0x286   0x02     0xFF 0x00    //Temperature from sensor
196.879   0x708   0x01     0x05         //Heartbeat from Comm (operational)
196.880   0x187   0x01     0x00         //Status from Power Unit
196.881   0x707   0x01     0x05
196.881   0x706   0x01     0x05
196.891   0x187   0x01     0x01         //Status from Power Unit (Change of state)
197.879   0x708   0x01     0x05
197.881   0x187   0x01     0x01         //Status from Power Unit (Event Timer)
```

```
197.881   0x707  0x01     0x05
197.882   0x706  0x01     0x05
198.880   0x708  0x01     0x05
198.882   0x187  0x01     0x01
...
201.270   0x286  0x02     0x54 0x00    //Temperature from sensor (Change of state)
201.371   0x286  0x02     0x46 0x00    //Temperature from sensor (Inhibit time)
201.471   0x286  0x02     0x37 0x00
201.571   0x286  0x02     0x26 0x00
201.671   0x286  0x02     0x14 0x00
201.673   0x187  0x01     0x00
201.771   0x286  0x02     0x01 0x00
201.772   0x087  0x08     0x05 0x10 0x00 0x14 0x01 0x00 0x00 0x00 //Emergency - Low temp.
201.773   0x187  0x01     0x02
201.871   0x286  0x02     0x00 0x00
201.883   0x708  0x01     0x05
201.884   0x187  0x01     0x02
201.884   0x707  0x01     0x05
201.885   0x706  0x01     0x05
202.645   0x286  0x02     0x01 0x00
202.745   0x286  0x02     0x07 0x00
202.746   0x087  0x08     0x00 0x00 0x00 0x14 0x07 0x00 0x00 0x00 //Emerg. – Error Reset
202.845   0x286  0x02     0x0F 0x00
202.883   0x708  0x01     0x05
202.885   0x187  0x01     0x02
202.885   0x707  0x01     0x05
202.886   0x706  0x01     0x05
202.945   0x286  0x02     0x17 0x00
202.947   0x187  0x01     0x00
203.045   0x286  0x02     0x1D 0x00
...
208.888   0x708  0x01     0x05
208.889   0x187  0x01     0x01
208.890   0x707  0x01     0x05
208.890   0x706  0x01     0x05
209.060   0x000  0x02     0x80 0x00           //NMT master - All Enter Pre-Operational
209.889   0x708  0x01     0x7F
209.891   0x707  0x01     0x7F
209.891   0x706  0x01     0x7F
...
216.834   0x000  0x02     0x01 0x00           //NMT master - All Enter Operational
216.894   0x708  0x01     0x05
216.895   0x187  0x01     0x01
216.896   0x707  0x01     0x05
216.896   0x706  0x01     0x05
...
221.900   0x706  0x01     0x05
222.613   0x706  0x01     0x00                //Sensor was reset
222.614   0x286  0x02     0xFF 0x00
222.899   0x708  0x01     0x05
222.900   0x187  0x01     0x01
222.900   0x707  0x01     0x05
223.403   0x087  0x08     0x30 0x81 0x10 0x0E 0x00 0x00 0x00 0x40 //Emergency from Power
223.614   0x706  0x01     0x05                                //unit, Heartbeat Consumer
223.900   0x708  0x01     0x05
223.901   0x707  0x01     0x7F
224.615   0x706  0x01     0x05
224.901   0x708  0x01     0x05
224.902   0x707  0x01     0x7F
...
231.998   0x000  0x02     0x81 0x00           //NMT master - All Reset
231.999   0x706  0x01     0x00
232.000   0x707  0x01     0x00
232.000   0x286  0x02     0xFF 0x00
232.009   0x708  0x01     0x00
233.000   0x706  0x01     0x05
233.001   0x187  0x01     0x00
233.001   0x707  0x01     0x05
233.010   0x708  0x01     0x05
233.011   0x187  0x01     0x01
234.001   0x706  0x01     0x05
```

With CANopen configuration tool network can be configured. For example, it is possible to change the variable TempHi. Picture 4.2 shows CANopen Device Monitor from Port. Below is shown reading and writing TempHi with such tool. Network was in operational state. Change remains also after power off. For protocol description see standards.

```
//Configuration tool requests reading from node-id=7, index 3001, sub0
---.---       0x607  0x08    0x40 0x01 0x30 0x00 0x00 0x00 0x00 0x00
// Power unit  responses: node-id=7, index 3001, sub0, data 2 bytes long = 0x19
---.---       0x587  0x08    0x4b 0x01 0x30 0x00 0x19 0x00 0x00 0x00

//Configuration tool request write to node-id=7, index 3001, sub0, value 1E, 2 bytes
---.---       0x607  0x08    0x2b 0x01 0x30 0x00 0x1e 0x00 0x00 0x00
// Power unit  responses: write successful
---.---       0x587  0x08    0x60 0x01 0x30 0x00 0x00 0x00 0x00 0x00
```



Picture 4.2 – CANopen Device Monitor

5. User interface with PIC + LCD + Keypad

You may try the Example_panel_with_PIC+LCD+keypad. It is like command panel with LCD display and numeric keypad. It has SDO client integrated. It is prepared for this tutorial. It has menus for changing TempLo and TempHi, password protection, some system menus and a menu for accessing Object Dictionary on any node on the network.

Hex file is available for PIC18F458 and oscillator 8 MHz (PPLx4 is enabled). For building the sources full optimization must be used. Detailed description is in manual.

6. Ethernet User interface

You may try the Example_web_interface_with_SC1x. It is advanced user interface through web browser. It is possible to make very professional and user friendly interface with CANopen network. Simple CANopen monitor and SDO client is made.

Detailed description is in manual.

7. Conclusion

Now you can explore CANopenNode on your own. Almost all features was shown in this tutorial. You may also read the Manual, where you will find detailed description of CANopenNode source code. You may write your code with minimal programming, as in this tutorial. You may also change CANopenNode to fit your needs, because it is flexible enough.

Hopefully CANopenNode will be useful for many projects. Quality of the project mainly depends on experience of programmer. Quality project will also have reliable hardware, documentation included and system for finding errors, which may occur in device or on network. (Errors always occur and usually there is some banal reason for it). CANopenNode should not be the source of errors, but it can be the base for tracking them. In case of any questions you may always contact the author.

CANopenNode is currently written for 8-bit PIC18F458 with Microchip C18 compiler and for 16-bit SC1x with Borland C5.02 compiler. It should not be hard to port the code to other systems. Many examples are written, including this tutorial, device profile for generic I/O, user interface, etc. Users on other systems are invited to translate this code for other Microcontrollers.