Программа для ЭВМ

«СМ Комитас»
Депонируемые материалы,
идентифицирующие программу для ЭВМ

Правообладатель:
Общество с ограниченной ответственностью «КОМИТАС»
125212, Российская Федерация, Москва, Головинское шоссе, д. 5, корп. 1, офис 10015/1

Авторы: отказывались быть упомянутыми в качестве таковых

Оглавление

Низкоуровневое ПО

1. Инициализация, конфигурация, основной цикл

```
#include <sys/types.h>
#include <sys/socket.h>
```

```cpp
#include <sys/time.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <algorithm>
#include <set>

#include "CanManager.hpp"
#include "MotorManager.hpp"
#include "Motor.hpp"
#include <gpiod.h>
#include "json11.hpp"
#include "modbus.h"
#include "Config.hpp"
#include "TCPServer.h"

#define SERVER_PORT 50007
#define MSERVER_PORT 502
#define HR_REGS 2500
modbus_mapping_t temp;
bool mounted;
bool sdPresent;
int num_message = 0;
TCPServer tcp;
pthread_t msg1[MAX_CLIENT];
MotorManager *MotorsBus[2];
NodeOutputLine NodeLine[2];
static sig_atomic_t sigval;
std::ifstream fileSD;
std::ofstream ofileSD;
Config cfg;

typedef struct {
  u_int16_t xConfig;
  u_int16_t ucMotorType;        /* Mapped at index 0x2000, subindex 0x01 */
  u_int16_t ucDriveDirection;       /* Mapped at index 0x2000, subindex 0x02 */
  float rSpeed;     /* Mapped at index 0x2000, subindex 0x04 */
  u_int16_t ucRampUp;       /* Mapped at index 0x2000, subindex 0x05 */
  u_int16_t ucRampDown;     /* Mapped at index 0x2000, subindex 0x06 */
  float OvercurrentLimit;       /* Mapped at index 0x2000, subindex 0x07 */
} MotorConfig_t;

typedef struct {
    std::string ip;
    std::string mask;
} Config_t;
```

```
Config_t stConfig;

static void onsig(int val)
{
    sigval = (sig_atomic_t)val;
}


int data[10];
enum {
    TCP,
    RTU
};


void GetMemoryTable();

char MNum[32][4]={"M01","M02","M03","M04","M05","M06","M07","M08","M09","M10",
                          "M11","M12","M13","M14","M15","M16","M17","M18","M19","M2
0",
                          "M21","M22","M23","M24","M25","M26","M27","M28","M29","M3
0","M31","M32"};

void Save_Cfg(void)
{
    std::cout << "Save_Cfg"<< std::endl;
    cfg.clear();
    for(int i=0;i<32;i++){
            MotorsBus[0]->_motors[i]->xConfig= MotorsBus[0]->_motors[i]-
>stStatus.xLink;
            cfg["CAN0"][MNum[i]]["isConfig"] = MotorsBus[0]->_motors[i]-
>stStatus.xLink;
            cfg["CAN0"][MNum[i]]["MotorType"] =
hr_param.N1_MotorParam[i].ucMotorType;
            cfg["CAN0"][MNum[i]]["Speed"] = hr_param.N1_MotorParam[i].rSpeed;
            cfg["CAN0"][MNum[i]]["RampUp"] = hr_param.N1_MotorParam[i].ucRampUp;
            cfg["CAN0"][MNum[i]]["RampDown"] =
hr_param.N1_MotorParam[i].ucRampDown;
            cfg["CAN0"][MNum[i]]["OvercurrentLimit"] =
hr_param.N1_MotorParam[i].OvercurrentLimit;

            MotorsBus[1]->_motors[i]->xConfig= MotorsBus[1]->_motors[i]-
>stStatus.xLink;
            cfg["CAN1"][MNum[i]]["isConfig"] = MotorsBus[1]->_motors[i]-
>stStatus.xLink;
            cfg["CAN1"][MNum[i]]["MotorType"] =
hr_param.N2_MotorParam[i].ucMotorType;
            cfg["CAN1"][MNum[i]]["Speed"] = hr_param.N2_MotorParam[i].rSpeed;
            cfg["CAN1"][MNum[i]]["RampUp"] = hr_param.N2_MotorParam[i].ucRampUp;
            cfg["CAN1"][MNum[i]]["RampDown"] =
hr_param.N2_MotorParam[i].ucRampDown;
```

```
            cfg["CAN1"][MNum[i]]["OvercurrentLimit"] =
hr_param.N2_MotorParam[i].OvercurrentLimit;
    }
    cfg["Net"]["IP1"] = hr_param.ip[0];
    cfg["Net"]["IP2"] = hr_param.ip[1];
    cfg["Net"]["IP3"] = hr_param.ip[2];
    cfg["Net"]["IP4"] = hr_param.ip[3];
    cfg["Net"]["Mask1"] = hr_param.mask[0];
    cfg["Net"]["Mask2"] = hr_param.mask[1];
    cfg["Net"]["Mask3"] = hr_param.mask[2];
    cfg["Net"]["Mask4"] = hr_param.mask[3];
    if(!cfg.write("/mnt/Comitas.cfg")){
        std::cerr<<"Could not write config file!"<<std::endl;
    }
}
void Default_Cfg(void)
{

}
void Read_Cfg(void)
{
    std::cout << "Read_Cfg"<< std::endl;
    if(mounted){
        try{
            cfg.clear();
            if(!cfg.read("/mnt/Comitas.cfg")){
                std::cout<<"Could not open config file!"<<std::endl;
                hr_param.ip[0] = 192;
                hr_param.ip[1] = 168;
                hr_param.ip[2] = 0;
                hr_param.ip[3] = 10;
                hr_param.mask[0] = 255;
                hr_param.mask[1] = 255;
                hr_param.mask[2] = 255;
                hr_param.mask[3] = 0;
                for (int i = 0; i < 32; i++)
                {
                    hr_param.N1_MotorParam[i].rSpeed = 500; //Float param
                    hr_param.N1_MotorParam[i].ucRampDown = 100; //Float param
                    hr_param.N1_MotorParam[i].ucRampUp = 60; //Float param
                    hr_param.N1_MotorParam[i].ucMotorType = 1; //Float param
                    hr_param.N2_MotorParam[i].rSpeed = 500; //Float param
                    hr_param.N2_MotorParam[i].ucRampDown = 100; //Float param
                    hr_param.N2_MotorParam[i].ucRampUp = 60; //Float param
                    hr_param.N2_MotorParam[i].ucMotorType = 1; //Float param
                }
            }else{
                hr_param.ip[0] = cfg["Net"]["IP1"].getInt();
                hr_param.ip[1] = cfg["Net"]["IP2"].getInt();
                hr_param.ip[2] = cfg["Net"]["IP3"].getInt();
                hr_param.ip[3] = cfg["Net"]["IP4"].getInt();
```

```
                hr_param.mask[0] = cfg["Net"]["Mask1"].getInt();
                hr_param.mask[1] = cfg["Net"]["Mask2"].getInt();
                hr_param.mask[2] = cfg["Net"]["Mask3"].getInt();
                hr_param.mask[3] = cfg["Net"]["Mask4"].getInt();
                for(int i=0;i<32;i++){
                    MotorsBus[0]->_motors[i]->xConfig =
(u_int16_t)cfg["CAN0"][MNum[i]]["isConfig"].getInt();
                    hr_param.N1_MotorParam[i].ucMotorType =
(u_int8_t)cfg["CAN0"][MNum[i]]["MotorType"].getInt();
                    hr_param.N1_MotorParam[i].rSpeed =
(u_int16_t)cfg["CAN0"][MNum[i]]["Speed"].getFloat();
                    hr_param.N1_MotorParam[i].ucRampUp =
(u_int8_t)cfg["CAN0"][MNum[i]]["RampUp"].getInt();
                    hr_param.N1_MotorParam[i].ucRampDown =
(u_int8_t)cfg["CAN0"][MNum[i]]["RampDown"].getInt();
                    hr_param.N1_MotorParam[i].OvercurrentLimit =
(u_int8_t)cfg["CAN0"][MNum[i]]["OvercurrentLimit"].getFloat();

                    MotorsBus[1]->_motors[i]->xConfig =
(u_int16_t)cfg["CAN1"][MNum[i]]["isConfig"].getInt();
                    hr_param.N2_MotorParam[i].ucMotorType =
(u_int8_t)cfg["CAN1"][MNum[i]]["MotorType"].getInt();
                    hr_param.N2_MotorParam[i].rSpeed =
(u_int16_t)cfg["CAN1"][MNum[i]]["Speed"].getFloat();
                    hr_param.N2_MotorParam[i].ucRampUp =
(u_int8_t)cfg["CAN1"][MNum[i]]["RampUp"].getInt();
                    hr_param.N2_MotorParam[i].ucRampDown =
(u_int8_t)cfg["CAN1"][MNum[i]]["RampDown"].getInt();
                    hr_param.N2_MotorParam[i].OvercurrentLimit =
(u_int8_t)cfg["CAN1"][MNum[i]]["OvercurrentLimit"].getFloat();
                }
            }
        }
        catch(std::invalid_argument& e){
            std::cout<<"Error reading config file!"<<std::endl;
            std::cout<<e.what()<<std::endl;
        }
    }else{
         std::cout<<"SD no found!"<<std::endl;
                hr_param.ip[0] = 192;
                hr_param.ip[1] = 168;
                hr_param.ip[2] = 0;
                hr_param.ip[3] = 10;
                hr_param.mask[0] = 255;
                hr_param.mask[1] = 255;
                hr_param.mask[2] = 255;
                hr_param.mask[3] = 0;
                for (int i = 0; i < 32; i++)
                {
                    hr_param.N1_MotorParam[i].rSpeed = 500; //Float param
                    hr_param.N1_MotorParam[i].ucRampDown = 100; //Float param
```

```
                    hr_param.N1_MotorParam[i].ucRampUp = 60; //Float param
                    hr_param.N1_MotorParam[i].ucMotorType = 1; //Float param
                    hr_param.N2_MotorParam[i].rSpeed = 500; //Float param
                    hr_param.N2_MotorParam[i].ucRampDown = 100; //Float param
                    hr_param.N2_MotorParam[i].ucRampUp = 60; //Float param
                    hr_param.N2_MotorParam[i].ucMotorType = 1; //Float param
                }
    }
}



void Set_IP(const std::string &new_ip, const std::string &new_mask)
{
    std::cout << "Set_IP"<< std::endl;
    std::cout << stConfig.ip << " <- " << new_ip << std::endl;
    std::cout << stConfig.mask << " <- " << new_mask << std::endl;
    stConfig.ip = new_ip;
    stConfig.mask = new_mask;
}

void close_app(int s) {
    tcp.closed();
    exit(0);
}
void checkSD(void) {
    fileSD.close();
    fileSD.open("/dev/mmcblk0p1", std::ios::in);
    if (fileSD.is_open()) {
        fileSD.close();
        sdPresent = true;
        if (!mounted)
        {
            //system("umount /dev/mmcblk0p1");
            int rez = system("mount -a /dev/mmcblk0p1 /mnt");
            if(rez == 0)
                mounted = true;
            else
                mounted = false;

        }
    }else{
            fileSD.close();
            sdPresent = false;
            mounted = false;

    }
}

const char *dev[]={"/sys/bus/iio/devices/iio:device0/in_voltage0_raw",
                   "/sys/bus/iio/devices/iio:device0/in_voltage1_raw",
```

```
                    "/sys/bus/iio/devices/iio:device0/in_voltage2_raw",
                    "/sys/bus/iio/devices/iio:device0/in_voltage3_raw",
                    "/sys/bus/iio/devices/iio:device0/in_voltage4_raw",
                    "/sys/bus/iio/devices/iio:device0/in_voltage5_raw",
                    "/sys/bus/iio/devices/iio:device0/in_voltage6_raw",
                    "/sys/bus/iio/devices/iio:device0/in_voltage7_raw",
                    "/sys/bus/iio/devices/iio:device0/in_voltage8_raw",
                    "/sys/bus/iio/devices/iio:device0/in_voltage9_raw"};


int main()
{
    struct gpiod_chip *chipLed;
    struct gpiod_chip *chipReset;
    chipLed = gpiod_chip_open("/dev/gpiochip2");
    chipReset = gpiod_chip_open("/dev/gpiochip0");

    NodeLine[0].Reset = gpiod_chip_get_line(chipReset, 30);
    NodeLine[1].Reset = gpiod_chip_get_line(chipReset, 31);
    NodeLine[0].Red = gpiod_chip_get_line(chipLed, 26);
    NodeLine[0].Blue = gpiod_chip_get_line(chipLed, 28);
    NodeLine[0].Green = gpiod_chip_get_line(chipLed, 27);
    NodeLine[1].Red = gpiod_chip_get_line(chipLed, 22);
    NodeLine[1].Blue = gpiod_chip_get_line(chipLed, 24);
    NodeLine[1].Green = gpiod_chip_get_line(chipLed, 23);

    /* Register signal handlers */
    /*
    if (signal(SIGINT, onsig)    == SIG_ERR ||
        signal(SIGTERM, onsig)   == SIG_ERR ||
        signal(SIGCHLD, SIG_IGN) == SIG_ERR)
    {
        return errno;
    }
    signal(SIGINT, close_app); */
    vector<int> opts = { SO_REUSEPORT, SO_REUSEADDR, TCP_NODELAY  };

    MotorsBus[0] =  new MotorManager("can0iobus",
&NodeLine[0],hr_param.N1_MotorADC1,hr_param.N1_MotorADC2,
                                     hr_param.N1_MotorStatus,hr_param.N1_MotorCMD
,hr_param.N1_HMIMotorCMD,
                                     hr_param.N1_MotorParam,hr_param.N1_SelectADC
);
    MotorsBus[1] =  new MotorManager("can1iobus",
&NodeLine[1],hr_param.N2_MotorADC1,hr_param.N2_MotorADC2,
                                     hr_param.N2_MotorStatus,hr_param.N2_MotorCMD
,hr_param.N2_HMIMotorCMD,
                                     hr_param.N2_MotorParam,hr_param.N2_SelectADC
);

    MotorsBus[0]->Init();
```

```
    MotorsBus[1]->Init();
    checkSD();
    Read_Cfg();

    if(( tcp.setup(MSERVER_PORT,opts) == 0)
        //&&( tcp.setup(MSERVER_PORT,opts) == 0)
        ) {
        /*if((pthread_create(&Wthread, NULL, received, (void *)0) == 0)
            ){*/
            temp.nb_registers = (sizeof(hr_param)/2);
            temp.start_registers = 0;
            temp.tab_registers = (unsigned short*)&hr_param;
            mb_mapping = &temp;
            /*mb_mapping = modbus_mapping_new(NULL,                          // no
coils
                                            NULL,                          // no
coils
                                            HR_REGS,    // holding reg
                                            NULL);                         // no
ir's*/

            if (mb_mapping != NULL){
                while(1) {
                    tcp.accepted(true);
                    MotorsBus[0]->Update();
                    MotorsBus[1]->Update();
                    checkSD();
                    if(hr_param.xHMICmd==1){
                        printf("Control off\n");
                    }
                    if(hr_param.xHMICmd==2){
                        printf("Control on\n");
                    }
                    if(hr_param.xHMICmd==4){
                        printf("Write cfg\n");
                        Save_Cfg();
                    }
                    if(hr_param.xHMICmd==8){
                        printf("Read cfg\n");
                        Read_Cfg();
                    }
                    if(hr_param.xHMICmd==16){
                        printf("Reset Addr\n");
                        MotorsBus[0]->xAddrInit = 0;
                        MotorsBus[1]->xAddrInit = 0;
                    }
                    if(hr_param.xHMICmd==32){
                        printf("Set IP\n");
                    }
                    hr_param.xHMICmd=0;
                    std::this_thread::sleep_for(std::chrono::milliseconds(5));
```

```
                }
            }
        //}
    }
    else
        cerr << "Errore apertura socket" << endl;

    tcp.closed();
    modbus_mapping_free(mb_mapping);
    delete MotorsBus[0];
    delete MotorsBus[1];
    gpiod_chip_close(chipLed);
    gpiod_chip_close(chipReset);

}
```

1.1 API конфигурации системы

```
#include "Config.hpp"

const unsigned int Config::TABS = 4;
const std::set<char> Config::delimeter = {'=', ';', '[', ']', '{', '}', ','};

// Element

Config::Element::Element(const std::string& name){
    this->name = name;
}

Config::Element::Element(){

}

void Config::Element::set(const std::string& value, const std::size_t i){
    if(i >= values.size()){
        values.resize(i+1);
    }

    values[i] = value;
}

void Config::Element::set(const int value, const std::size_t i){
    set(std::to_string(value), i);
}

void Config::Element::set(const float value, const std::size_t i){
    set(std::to_string(value), i);
}
```

```
void Config::Element::operator=(const std::string& value){
    set(value);
}

void Config::Element::operator=(const int value){
    set(value);
}

void Config::Element::operator=(const float value){
    set(value);
}

void Config::Element::operator=(const double value){
    set(float(value));
}

const std::string& Config::Element::getName(){
    return name;
}

const std::string& Config::Element::getString(const std::size_t i){
    if(i >= values.size()){
        values.resize(i+1);
        values[i] = "";
    }

    return values[i];
}

const int Config::Element::getInt(const std::size_t i){
    return std::stoi(getString(i));
}

const float Config::Element::getFloat(const std::size_t i){
    return std::stof(getString(i));
}

Config::Element& Config::Element::operator[](const std::string& name){
    for(auto i = children.begin(); i != children.end(); i++){
        if((*i).name == name){
            return *i;
        }
    }

    children.push_back(Element(name));

    return children.back();
}

bool Config::Element::exists(const std::string& name){
```

```
    for(auto e : children){
        if(e.name == name){
            return true;
        }
    }

    return false;
}

void Config::Element::remove(const std::string& name){
    for(auto i = children.begin(); i != children.end(); i++){
        if((*i).name == name){
            children.erase(i);
            return;
        }
    }
}

void Config::Element::clear(){
    children.clear();
}

const std::vector<Config::Element>& Config::Element::getAll(){
    return children;
}

// Config

bool Config::testString(const char c){
    if(isstring && c == '\\'){ // allow mask in strings
        mask = !mask;

        if(mask){
            return true;
        }
    }

    if(!mask && (c == '"' || c == '\'')){ // strings " or '
        isstring = !isstring;
    }

    mask = false; // reset mask

    return false;
}

bool Config::accept(const std::string& token){
    return token == get();
}

void Config::expect(const std::string& token){
```

```
    if(token != get()){
        throw std::invalid_argument("Unexpected token, was \""+get()+"\" expected
\""+token+"\" in line "+std::to_string(tokens[index].line)+" at character
"+std::to_string(tokens[index].character)+".");
    }

    next();
}

void Config::next(){
    if(index < tokens.size()-1){
        index++;
    }
}

const std::string& Config::get(){
    return tokens[index].token;
}

void Config::parseVar(std::vector<Element>& parent){
    Element e(get());
    next();
    expect("=");

    if(accept("[")){
        parseArray(e);
    }
    else if(accept("{")){
        parseObject(e);
    }
    else{
        e.values.push_back(get());
        next();
        expect(";");
    }

    parent.push_back(e);
}

void Config::parseArray(Element& e){
    expect("[");

    while(!accept("]")){
        e.values.push_back(get());
        next();

        if(accept(",")){
            next();
        }

        while(accept(",")){
```

```
            e.values.push_back("");
            next();
        }
    }

    expect("]");

    if(accept(";")){ // optional
        next();
    }
}

void Config::parseObject(Element& e){
    expect("{");

    while(!accept("}")){
        parseVar(e.children);
    }

    expect("}");

    if(accept(";")){ // optional
        next();
    }
}

void Config::preprocess(std::ifstream& file){
    input = "";

    std::string line;
    isstring = false;
    mask = false;

    while(std::getline(file, line)){
        for(auto c : line){
            if(!isstring && (c == ' ' || c == '\t')){ // skip white space
                continue;
            }

            if(!isstring && c == '#'){ // skip comments
                break;
            }

            testString(c);

            input += c;
        }

        if(isstring){
            input += '\n';
        }
```

```
    }
}

void Config::tokenize(){
    tokens.clear();

    auto end = delimeter.end();
    std::string token, deli;
    isstring = false;
    mask = false;
    std::size_t line = 1;
    std::size_t character = 0;

    for(auto c : input){
        if(testString(c)){
            continue;
        }

        // count lines and character but skip new line
        if(c == '\n'){
            line++;
            character = 0;

            continue;
        }

        if(!isstring && delimeter.find(c) != end){
            // cut " if string
            if(token.length() && token.at(0) == '"' && token.at(token.length()-1)
== '"'){
                token = token.substr(1, token.length()-2);
            }

            // store token
            if(!token.empty()){
                tokens.push_back(Token(token, line, character));
                token = "";
            }

            // store delimeter
            deli = c;
            tokens.push_back(Token(deli, line, character));

            continue;
        }

        token += c;
        character = 0;
    }
}
```

```
void Config::parse(){
    index = 0;
    root.clear();

    while(index != tokens.size()-1){
        parseVar(root.children);
    }
}

bool Config::isNumeric(const std::string& value){
    for(auto c : value){
        if(!std::isdigit(c, std::locale()) && c != '.' && c != '-'){
            return false;
        }
    }

    return true;
}

void Config::writeElement(std::ofstream& file, const Element& element, const
unsigned int tabs){
    if(!element.values.size() && !element.children.size()){
        return;
    }

    for(unsigned int i = 0; i < tabs; i++){
        file<<" ";
    }

    bool isnumeric = false;

    file<<element.name<<" = ";

    if(element.children.size()){
        file<<"{\n";

        for(auto e : element.children){
            writeElement(file, e, tabs+TABS);
        }

        for(unsigned int i = 0; i < tabs; i++){
            file<<" ";
        }

        file<<"}";
    }
    else if(element.values.size() == 1){
        if(isNumeric(element.values[0])){
            file<<element.values[0]<<";";
        }
        else{
```

```
            file<<"\""<<element.values[0]<<"\";";
        }
    }
    else{
        file<<"[";

        for(std::size_t i = 0; i < element.values.size(); i++){
            isnumeric = isNumeric(element.values[i]);

            if(!isnumeric){
                file<<"\"";
            }

            file<<element.values[i];

            if(!isnumeric){
                file<<"\"";
            }

            if(i != element.values.size()-1){
                file<<", ";
            }
        }

        file<<"]";
    }

    file<<"\n";
}

bool Config::read(const std::string& path){
    std::ifstream file(path);

    if(!file.good()){
        file.close();
        return false;
    }

    preprocess(file);
    file.close();
    tokenize();
    parse();

    return true;
}

bool Config::write(const std::string& path){
    std::ofstream file(path);

    if(!file.good()){
        file.close();
```

```
        return false;
    }

    for(auto e : root.children){
        writeElement(file, e);
    }

    file.close();

    return true;
}

Config::Element& Config::operator[](const std::string& name){
    return root[name];
}

bool Config::exists(const std::string& name){
    return root.exists(name);
}

void Config::remove(const std::string& name){
    root.remove(name);
}

void Config::clear(){
    root.clear();
}

const std::vector<Config::Element>& Config::getAll(){
    return root.children;
}
```

2. Класс конфигурации системы

```
#pragma once

#include <string>
#include <fstream>
#include <vector>
#include <set>
#include <stdexcept>
#include <locale>

class Config{
    public:
        class Element{
```

```
            friend Config;

            private:
                std::string name;
                std::vector<std::string> values;
                std::vector<Element> children;

            public:
                Element(const std::string& name);
                Element();

                void set(const std::string& value, const std::size_t i = 0);
                void set(const int value, const std::size_t i = 0);
                void set(const float value, const std::size_t i = 0);
                void operator=(const std::string& value);
                void operator=(const int value);
                void operator=(const float value);
                void operator=(const double value);

                const std::string& getName();
                const std::string& getString(const std::size_t i = 0);
                const int getInt(const std::size_t i = 0);
                const float getFloat(const std::size_t i = 0);

                Element& operator[](const std::string& name);
                bool exists(const std::string& name);
                void remove(const std::string& name);
                void clear();

                const std::vector<Element>& getAll();
        };

    private:
        struct Token{
            std::string token;
            std::size_t line, character;

            Token(const std::string& token, const std::size_t line, const
std::size_t character):
                token(token),
                line(line),
                character(character){

            }
        };

        static const unsigned int TABS;
        static const std::set<char> delimeter;

        // preprocessor, tokenizer and parser
        std::string input;
```

```
        std::size_t index;
        std::vector<Token> tokens;
        bool isstring, mask;

        bool testString(const char c);

        bool accept(const std::string& token);
        void expect(const std::string& token);
        void next();
        const std::string& get();

        void parseVar(std::vector<Element>& parent);
        void parseArray(Element& e);
        void parseObject(Element& e);

        void preprocess(std::ifstream& file);
        void tokenize();
        void parse();

        // output
        bool isNumeric(const std::string& value);
        void writeElement(std::ofstream& file, const Element& element, const
unsigned int tabs = 0);

        // top level element
        Element root;

    public:
        bool read(const std::string& path);
        bool write(const std::string& path);

        Element& operator[](const std::string& name);
        bool exists(const std::string& name);
        void remove(const std::string& name);
        void clear();

        const std::vector<Element>& getAll();
};
```

3. API для использования JSON формата

```
#include "json11.hpp"
#include <cassert>
#include <cmath>
#include <cstdlib>
```

```cpp
#include <cstdio>
#include <limits>

namespace json11 {

static const int max_depth = 200;

using std::string;
using std::vector;
using std::map;
using std::make_shared;
using std::initializer_list;
using std::move;

/* Helper for representing null - just a do-nothing struct, plus comparison
 * operators so the helpers in JsonValue work. We can't use nullptr_t because
 * it may not be orderable.
 */
struct NullStruct {
    bool operator==(NullStruct) const { return true; }
    bool operator<(NullStruct) const { return false; }
};

/* * * * * * * * * * * * * * * * * * * *
 * Serialization
 */

static void dump(NullStruct, string &out) {
    out += "null";
}
static void dumpStruct(NullStruct, string &out) {
    out += "null";
}

static void dump(double value, string &out) {
    if (std::isfinite(value)) {
        char buf[32];
        snprintf(buf, sizeof buf, "%.17g", value);
        out += buf;
    } else {
        out += "null";
    }
}
static void dumpStruct(double value, string &out) {
    if (std::isfinite(value)) {
        char buf[32];
        snprintf(buf, sizeof buf, "%.17g", value);
        out += buf;
    } else {
        out += "null";
    }
```

```
}

static void dump(int value, string &out) {
    char buf[32];
    snprintf(buf, sizeof buf, "%d", value);
    out += buf;
}
static void dumpStruct(int value, string &out) {
    char buf[32];
    snprintf(buf, sizeof buf, "%d", value);
    out += buf;
}

static void dump(bool value, string &out) {
    out += value ? "true" : "false";
}
static void dumpStruct(bool value, string &out) {
    out += value ? "true" : "false";
}

static void dumpStruct(const string &value, string &out) {
    out += '"';
    for (size_t i = 0; i < value.length(); i++) {
        const char ch = value[i];
        /*if (ch == '\\') {
            out += "\\\\";
        } else if (ch == '"') {
            out += "\\\"";
        } else if (ch == '\b') {
            out += "\\b";
        } else if (ch == '\f') {
            out += "\\f";
        } else if (ch == '\n') {
            out += "\\n";
        } else if (ch == '\r') {
            out += "\\r";
        } else if (ch == '\t') {
            out += "\\t";
        } else if (static_cast<uint8_t>(ch) <= 0x1f) {
            char buf[8];
            snprintf(buf, sizeof buf, "\\u%04x", ch);
            out += buf;
        } else if (static_cast<uint8_t>(ch) == 0xe2 &&
static_cast<uint8_t>(value[i+1]) == 0x80
                   && static_cast<uint8_t>(value[i+2]) == 0xa8) {
            out += "\\u2028";
            i += 2;
        } else if (static_cast<uint8_t>(ch) == 0xe2 &&
static_cast<uint8_t>(value[i+1]) == 0x80
                   && static_cast<uint8_t>(value[i+2]) == 0xa9) {
            out += "\\u2029";
```

```
            i += 2;
        } else */{
            out += ch;
        }
    }
    out += '"';
}

static void dump(const string &value, string &out) {
    out += '"';
    for (size_t i = 0; i < value.length(); i++) {
        const char ch = value[i];
        if (ch == '\\') {
            out += "\\\\";
        } else if (ch == '"') {
            out += "\\\"";
        } else if (ch == '\b') {
            out += "\\b";
        } else if (ch == '\f') {
            out += "\\f";
        } else if (ch == '\n') {
            out += "\\n";
        } else if (ch == '\r') {
            out += "\\r";
        } else if (ch == '\t') {
            out += "\\t";
        } else if (static_cast<uint8_t>(ch) <= 0x1f) {
            char buf[8];
            snprintf(buf, sizeof buf, "\\u%04x", ch);
            out += buf;
        } else if (static_cast<uint8_t>(ch) == 0xe2 &&
static_cast<uint8_t>(value[i+1]) == 0x80
                   && static_cast<uint8_t>(value[i+2]) == 0xa8) {
            out += "\\u2028";
            i += 2;
        } else if (static_cast<uint8_t>(ch) == 0xe2 &&
static_cast<uint8_t>(value[i+1]) == 0x80
                   && static_cast<uint8_t>(value[i+2]) == 0xa9) {
            out += "\\u2029";
            i += 2;
        } else {
            out += ch;
        }
    }
    out += '"';
}

static void dump(const Json::array &values, string &out) {
    bool first = true;
    out += "[";
    for (const auto &value : values) {
```

```
        if (!first)
            out += ", ";
        value.dump(out);
        first = false;
    }
    out += "]";
}
static void dumpStruct(const Json::array &values, string &out) {
    bool first = true;
    out += "[\n  ";
    for (const auto &value : values) {
        if (!first)
            out += ",\n  ";
        value.dumpStruct(out);
        first = false;
    }
    out += "\n ]";
}

static void dump(const Json::object &values, string &out) {
    bool first = true;
    out += "{";
    for (const auto &kv : values) {
        if (!first)
            out += ", ";
        dump(kv.first, out);
        out += ": ";
        kv.second.dump(out);
        first = false;
    }
    out += "}";
}

static void dumpStruct(const Json::object &values, string &out) {
    bool first = true;
    out += "{\n ";
    for (const auto &kv : values) {
        if (!first)
            out += ",\n ";
        dump(kv.first, out);
        out += ": ";
        kv.second.dumpStruct(out);
        first = false;
    }
    out += "\n}";
}

void Json::dumpStruct(string &out) const {
    m_ptr->dumpStruct(out);
}
```

```
void Json::dump(string &out) const {
    m_ptr->dump(out);
}

/* * * * * * * * * * * * * * * * * * * *
 * Value wrappers
 */

template <Json::Type tag, typename T>
class Value : public JsonValue {
protected:

    // Constructors
    explicit Value(const T &value) : m_value(value) {}
    explicit Value(T &&value)      : m_value(move(value)) {}

    // Get type tag
    Json::Type type() const override {
        return tag;
    }

    // Comparisons
    bool equals(const JsonValue * other) const override {
        return m_value == static_cast<const Value<tag, T> *>(other)->m_value;
    }
    bool less(const JsonValue * other) const override {
        return m_value < static_cast<const Value<tag, T> *>(other)->m_value;
    }

    const T m_value;
    void dump(string &out) const override { json11::dump(m_value, out); }
    void dumpStruct(string &out) const override { json11::dumpStruct(m_value,
out); }
};

class JsonDouble final : public Value<Json::NUMBER, double> {
    double number_value() const override { return m_value; }
    int int_value() const override { return static_cast<int>(m_value); }
    bool equals(const JsonValue * other) const override { return m_value ==
other->number_value(); }
    bool less(const JsonValue * other)   const override { return m_value
<  other->number_value(); }
public:
    explicit JsonDouble(double value) : Value(value) {}
};

class JsonInt final : public Value<Json::NUMBER, int> {
    double number_value() const override { return m_value; }
    int int_value() const override { return m_value; }
    bool equals(const JsonValue * other) const override { return m_value ==
other->number_value(); }
```

```
    bool less(const JsonValue * other)   const override { return m_value
<  other->number_value(); }
public:
    explicit JsonInt(int value) : Value(value) {}
};

class JsonBoolean final : public Value<Json::BOOL, bool> {
    bool bool_value() const override { return m_value; }
public:
    explicit JsonBoolean(bool value) : Value(value) {}
};

class JsonString final : public Value<Json::STRING, string> {
    const string &string_value() const override { return m_value; }
public:
    explicit JsonString(const string &value) : Value(value) {}
    explicit JsonString(string &&value)      : Value(move(value)) {}
};

class JsonArray final : public Value<Json::ARRAY, Json::array> {
    const Json::array &array_items() const override { return m_value; }
    const Json & operator[](size_t i) const override;
public:
    explicit JsonArray(const Json::array &value) : Value(value) {}
    explicit JsonArray(Json::array &&value)      : Value(move(value)) {}
};

class JsonObject final : public Value<Json::OBJECT, Json::object> {
    const Json::object &object_items() const override { return m_value; }
    const Json & operator[](const string &key) const override;
public:
    explicit JsonObject(const Json::object &value) : Value(value) {}
    explicit JsonObject(Json::object &&value)      : Value(move(value)) {}
};

class JsonNull final : public Value<Json::NUL, NullStruct> {
public:
    JsonNull() : Value({}) {}
};

/* * * * * * * * * * * * * * * * * * * *
 * Static globals - static-init-safe
 */
struct Statics {
    const std::shared_ptr<JsonValue> null = make_shared<JsonNull>();
    const std::shared_ptr<JsonValue> t = make_shared<JsonBoolean>(true);
    const std::shared_ptr<JsonValue> f = make_shared<JsonBoolean>(false);
    const string empty_string;
    const vector<Json> empty_vector;
    const map<string, Json> empty_map;
    Statics() {}
```

```
};

static const Statics & statics() {
    static const Statics s {};
    return s;
}

static const Json & static_null() {
    // This has to be separate, not in Statics, because Json() accesses
statics().null.
    static const Json json_null;
    return json_null;
}

/* * * * * * * * * * * * * * * * * * * *
 * Constructors
 */

Json::Json() noexcept                  : m_ptr(statics().null) {}
Json::Json(std::nullptr_t) noexcept    : m_ptr(statics().null) {}
Json::Json(double value)               : m_ptr(make_shared<JsonDouble>(value)) {}
Json::Json(int value)                  : m_ptr(make_shared<JsonInt>(value)) {}
Json::Json(bool value)                 : m_ptr(value ? statics().t : statics().f)
{}
Json::Json(const string &value)        : m_ptr(make_shared<JsonString>(value)) {}
Json::Json(string &&value)             :
m_ptr(make_shared<JsonString>(move(value))) {}
Json::Json(const char * value)         : m_ptr(make_shared<JsonString>(value)) {}
Json::Json(const Json::array &values)  : m_ptr(make_shared<JsonArray>(values)) {}
Json::Json(Json::array &&values)       :
m_ptr(make_shared<JsonArray>(move(values))) {}
Json::Json(const Json::object &values) : m_ptr(make_shared<JsonObject>(values))
{}
Json::Json(Json::object &&values)      :
m_ptr(make_shared<JsonObject>(move(values))) {}

/* * * * * * * * * * * * * * * * * * * *
 * Accessors
 */

Json::Type Json::type()                           const { return m_ptr-
>type();         }
double Json::number_value()                       const { return m_ptr-
>number_value(); }
int Json::int_value()                             const { return m_ptr-
>int_value();    }
bool Json::bool_value()                           const { return m_ptr-
>bool_value();   }
const string & Json::string_value()               const { return m_ptr-
>string_value(); }
```

```
const vector<Json> & Json::array_items()          const { return m_ptr-
>array_items();  }
const map<string, Json> & Json::object_items()    const { return m_ptr-
>object_items(); }
const Json & Json::operator[] (size_t i)          const { return
(*m_ptr)[i];           }
const Json & Json::operator[] (const string &key) const { return
(*m_ptr)[key];         }

double                    JsonValue::number_value()              const { return
0; }
int                       JsonValue::int_value()                 const { return
0; }
bool                      JsonValue::bool_value()                const { return
false; }
const string &            JsonValue::string_value()              const { return
statics().empty_string; }
const vector<Json> &      JsonValue::array_items()               const { return
statics().empty_vector; }
const map<string, Json> & JsonValue::object_items()              const { return
statics().empty_map; }
const Json &              JsonValue::operator[] (size_t)         const { return
static_null(); }
const Json &              JsonValue::operator[] (const string &) const { return
static_null(); }

const Json & JsonObject::operator[] (const string &key) const {
    auto iter = m_value.find(key);
    return (iter == m_value.end()) ? static_null() : iter->second;
}
const Json & JsonArray::operator[] (size_t i) const {
    if (i >= m_value.size()) return static_null();
    else return m_value[i];
}

/* * * * * * * * * * * * * * * * * * * *
 * Comparison
 */

bool Json::operator== (const Json &other) const {
    if (m_ptr == other.m_ptr)
        return true;
    if (m_ptr->type() != other.m_ptr->type())
        return false;

    return m_ptr->equals(other.m_ptr.get());
}

bool Json::operator< (const Json &other) const {
    if (m_ptr == other.m_ptr)
        return false;
```

```
    if (m_ptr->type() != other.m_ptr->type())
        return m_ptr->type() < other.m_ptr->type();

    return m_ptr->less(other.m_ptr.get());
}

/* * * * * * * * * * * * * * * * * * * *
 * Parsing
 */

/* esc(c)
 *
 * Format char c suitable for printing in an error message.
 */
static inline string esc(char c) {
    char buf[12];
    if (static_cast<uint8_t>(c) >= 0x20 && static_cast<uint8_t>(c) <= 0x7f) {
        snprintf(buf, sizeof buf, "'%c' (%d)", c, c);
    } else {
        snprintf(buf, sizeof buf, "(%d)", c);
    }
    return string(buf);
}

static inline bool in_range(long x, long lower, long upper) {
    return (x >= lower && x <= upper);
}

namespace {
/* JsonParser
 *
 * Object that tracks all state of an in-progress parse.
 */
struct JsonParser final {

    /* State
     */
    const string &str;
    size_t i;
    string &err;
    bool failed;
    const JsonParse strategy;

    /* fail(msg, err_ret = Json())
     *
     * Mark this parse as failed.
     */
    Json fail(string &&msg) {
        return fail(move(msg), Json());
    }
```

```
    template <typename T>
    T fail(string &&msg, const T err_ret) {
        if (!failed)
            err = std::move(msg);
        failed = true;
        return err_ret;
    }

    /* consume_whitespace()
     *
     * Advance until the current character is non-whitespace.
     */
    void consume_whitespace() {
        while (str[i] == 0x0A || str[i] == ' ' || str[i] == '\r' || str[i] ==
'\n' || str[i] == '\t')
            i++;
    }

    /* consume_comment()
     *
     * Advance comments (c-style inline and multiline).
     */
    bool consume_comment() {
      bool comment_found = false;
      if (str[i] == '/') {
        i++;
        if (i == str.size())
          return fail("unexpected end of input after start of comment", false);
        if (str[i] == '/') { // inline comment
          i++;
          // advance until next line, or end of input
          while (i < str.size() && str[i] != '\n') {
            i++;
          }
          comment_found = true;
        }
        else if (str[i] == '*') { // multiline comment
          i++;
          if (i > str.size()-2)
            return fail("unexpected end of input inside multi-line comment",
false);
          // advance until closing tokens
          while (!(str[i] == '*' && str[i+1] == '/')) {
            i++;
            if (i > str.size()-2)
              return fail(
                "unexpected end of input inside multi-line comment", false);
          }
          i += 2;
          comment_found = true;
        }
```

```
        else
          return fail("malformed comment", false);
      }
      return comment_found;
    }

    /* consume_garbage()
     *
     * Advance until the current character is non-whitespace and non-comment.
     */
    void consume_garbage() {
      consume_whitespace();
      if(strategy == JsonParse::COMMENTS) {
        bool comment_found = false;
        do {
          comment_found = consume_comment();
          if (failed) return;
          consume_whitespace();
        }
        while(comment_found);
      }
    }

    /* get_next_token()
     *
     * Return the next non-whitespace character. If the end of the input is
reached,
     * flag an error and return 0.
     */
    char get_next_token() {
        consume_garbage();
        if (failed) return static_cast<char>(0);
        if (i == str.size())
            return fail("unexpected end of input", static_cast<char>(0));

        return str[i++];
    }

    /* encode_utf8(pt, out)
     *
     * Encode pt as UTF-8 and add it to out.
     */
    void encode_utf8(long pt, string & out) {
        if (pt < 0)
            return;

        if (pt < 0x80) {
            out += static_cast<char>(pt);
        } else if (pt < 0x800) {
            out += static_cast<char>((pt >> 6) | 0xC0);
            out += static_cast<char>((pt & 0x3F) | 0x80);
```

```
    } else if (pt < 0x10000) {
        out += static_cast<char>((pt >> 12) | 0xE0);
        out += static_cast<char>(((pt >> 6) & 0x3F) | 0x80);
        out += static_cast<char>((pt & 0x3F) | 0x80);
    } else {
        out += static_cast<char>((pt >> 18) | 0xF0);
        out += static_cast<char>(((pt >> 12) & 0x3F) | 0x80);
        out += static_cast<char>(((pt >> 6) & 0x3F) | 0x80);
        out += static_cast<char>((pt & 0x3F) | 0x80);
    }
}

/* parse_string()
 *
 * Parse a string, starting at the current position.
 */
string parse_string() {
    string out;
    long last_escaped_codepoint = -1;
    while (true) {
        if (i == str.size())
            return fail("unexpected end of input in string", "");

        char ch = str[i++];

        if (ch == '"') {
            encode_utf8(last_escaped_codepoint, out);
            return out;
        }

        if (in_range(ch, 0, 0x1f))
            return fail("unescaped " + esc(ch) + " in string", "");

        // The usual case: non-escaped characters
        if (ch != '\\') {
            encode_utf8(last_escaped_codepoint, out);
            last_escaped_codepoint = -1;
            out += ch;
            continue;
        }

        // Handle escapes
        if (i == str.size())
            return fail("unexpected end of input in string", "");

        ch = str[i++];

        if (ch == 'u') {
            // Extract 4-byte escape sequence
            string esc = str.substr(i, 4);
            // Explicitly check length of the substring. The following loop
```

```
                // relies on std::string returning the terminating NUL when
                // accessing str[length]. Checking here reduces brittleness.
                if (esc.length() < 4) {
                    return fail("bad \\u escape: " + esc, "");
                }
                for (size_t j = 0; j < 4; j++) {
                    if (!in_range(esc[j], 'a', 'f') && !in_range(esc[j], 'A',
'F')
                            && !in_range(esc[j], '0', '9'))
                        return fail("bad \\u escape: " + esc, "");
                }

                long codepoint = strtol(esc.data(), nullptr, 16);

                // JSON specifies that characters outside the BMP shall be
encoded as a pair
                // of 4-hex-digit \u escapes encoding their surrogate pair
components. Check
                // whether we're in the middle of such a beast: the previous
codepoint was an
                // escaped lead (high) surrogate, and this is a trail (low)
surrogate.
                if (in_range(last_escaped_codepoint, 0xD800, 0xDBFF)
                        && in_range(codepoint, 0xDC00, 0xDFFF)) {
                    // Reassemble the two surrogate pairs into one astral-plane
character, per
                    // the UTF-16 algorithm.
                    encode_utf8((((last_escaped_codepoint - 0xD800) << 10)
                                | (codepoint - 0xDC00)) + 0x10000, out);
                    last_escaped_codepoint = -1;
                } else {
                    encode_utf8(last_escaped_codepoint, out);
                    last_escaped_codepoint = codepoint;
                }

                i += 4;
                continue;
            }

            encode_utf8(last_escaped_codepoint, out);
            last_escaped_codepoint = -1;

            if (ch == 'b') {
                out += '\b';
            } else if (ch == 'f') {
                out += '\f';
            } else if (ch == 'n') {
                out += '\n';
            } else if (ch == 'r') {
                out += '\r';
            } else if (ch == 't') {
```

```
                out += '\t';
            } else if (ch == '"' || ch == '\\' || ch == '/') {
                out += ch;
            } else {
                return fail("invalid escape character " + esc(ch), "");
            }
        }
    }

    /* parse_number()
     *
     * Parse a double.
     */
    Json parse_number() {
        size_t start_pos = i;

        if (str[i] == '-')
            i++;

        // Integer part
        if (str[i] == '0') {
            i++;
            if (in_range(str[i], '0', '9'))
                return fail("leading 0s not permitted in numbers");
        } else if (in_range(str[i], '1', '9')) {
            i++;
            while (in_range(str[i], '0', '9'))
                i++;
        } else {
            return fail("invalid " + esc(str[i]) + " in number");
        }

        if (str[i] != '.' && str[i] != 'e' && str[i] != 'E'
                && (i - start_pos) <=
static_cast<size_t>(std::numeric_limits<int>::digits10)) {
            return std::atoi(str.c_str() + start_pos);
        }

        // Decimal part
        if (str[i] == '.') {
            i++;
            if (!in_range(str[i], '0', '9'))
                return fail("at least one digit required in fractional part");

            while (in_range(str[i], '0', '9'))
                i++;
        }

        // Exponent part
        if (str[i] == 'e' || str[i] == 'E') {
            i++;
```

```
            if (str[i] == '+' || str[i] == '-')
                i++;

            if (!in_range(str[i], '0', '9'))
                return fail("at least one digit required in exponent");

            while (in_range(str[i], '0', '9'))
                i++;
        }

        return std::strtod(str.c_str() + start_pos, nullptr);
    }

    /* expect(str, res)
     *
     * Expect that 'str' starts at the character that was just read. If it does,
advance
     * the input and return res. If not, flag an error.
     */
    Json expect(const string &expected, Json res) {
        assert(i != 0);
        i--;
        if (str.compare(i, expected.length(), expected) == 0) {
            i += expected.length();
            return res;
        } else {
            return fail("parse error: expected " + expected + ", got " +
str.substr(i, expected.length()));
        }
    }

    /* parse_json()
     *
     * Parse a JSON object.
     */
    Json parse_json(int depth) {
        if (depth > max_depth) {
            return fail("exceeded maximum nesting depth");
        }

        char ch = get_next_token();
        if (failed)
            return Json();

        if (ch == '-' || (ch >= '0' && ch <= '9')) {
            i--;
            return parse_number();
        }

        if (ch == 't')
```

```
        return expect("true", true);

    if (ch == 'f')
        return expect("false", false);

    if (ch == 'n')
        return expect("null", Json());

    if (ch == '"')
        return parse_string();

    if (ch == '{') {
        map<string, Json> data;
        ch = get_next_token();
        if (ch == '}')
            return data;

        while (1) {
            if (ch != '"')
                return fail("expected '\"' in object, got " + esc(ch));

            string key = parse_string();
            if (failed)
                return Json();

            ch = get_next_token();
            if (ch != ':')
                return fail("expected ':' in object, got " + esc(ch));

            data[std::move(key)] = parse_json(depth + 1);
            if (failed)
                return Json();

            ch = get_next_token();
            if (ch == '}')
                break;
            if (ch != ',')
                return fail("expected ',' in object, got " + esc(ch));

            ch = get_next_token();
        }
        return data;
    }

    if (ch == '[') {
        vector<Json> data;
        ch = get_next_token();
        if (ch == ']')
            return data;

        while (1) {
```

```
                i--;
                data.push_back(parse_json(depth + 1));
                if (failed)
                    return Json();

                ch = get_next_token();
                if (ch == ']')
                    break;
                if (ch != ',')
                    return fail("expected ',' in list, got " + esc(ch));

                ch = get_next_token();
                (void)ch;
            }
            return data;
        }

        return fail("expected value, got " + esc(ch));
    }
};
}//namespace {

Json Json::parse(const string &in, string &err, JsonParse strategy) {
    JsonParser parser { in, 0, err, false, strategy };
    Json result = parser.parse_json(0);

    // Check for any trailing garbage
    parser.consume_garbage();
    if (parser.failed)
        return Json();
    if (parser.i != in.size())
        return parser.fail("unexpected trailing " + esc(in[parser.i]));

    return result;
}

// Documented in json11.hpp
vector<Json> Json::parse_multi(const string &in,
                               std::string::size_type &parser_stop_pos,
                               string &err,
                               JsonParse strategy) {
    JsonParser parser { in, 0, err, false, strategy };
    parser_stop_pos = 0;
    vector<Json> json_vec;
    while (parser.i != in.size() && !parser.failed) {
        json_vec.push_back(parser.parse_json(0));
        if (parser.failed)
            break;

        // Check for another object
        parser.consume_garbage();
```

```
        if (parser.failed)
            break;
        parser_stop_pos = parser.i;
    }
    return json_vec;
}

/* * * * * * * * * * * * * * * * * * * *
 * Shape-checking
 */

bool Json::has_shape(const shape & types, string & err) const {
    if (!is_object()) {
        err = "expected JSON object, got " + dump();
        return false;
    }

    const auto& obj_items = object_items();
    for (auto & item : types) {
        const auto it = obj_items.find(item.first);
        if (it == obj_items.cend() || it->second.type() != item.second) {
            err = "bad type for " + item.first + " in " + dump();
            return false;
        }
    }

    return true;
}

} // namespace json11
```

4. Класс для работы с JSON

```
#pragma once

#include <string>
#include <vector>
#include <map>
#include <memory>
#include <initializer_list>

#ifdef _MSC_VER
    #if _MSC_VER <= 1800 // VS 2013
        #ifndef noexcept
            #define noexcept throw()
        #endif
```

```
        #ifndef snprintf
            #define snprintf _snprintf_s
        #endif
    #endif
#endif

namespace json11 {

enum JsonParse {
    STANDARD, COMMENTS
};

class JsonValue;

class Json final {
public:
    // Types
    enum Type {
        NUL, NUMBER, BOOL, STRING, ARRAY, OBJECT
    };

    // Array and object typedefs
    typedef std::vector<Json> array;
    typedef std::map<std::string, Json> object;

    // Constructors for the various types of JSON value.
    Json() noexcept;                // NUL
    Json(std::nullptr_t) noexcept;  // NUL
    Json(double value);             // NUMBER
    Json(int value);                // NUMBER
    Json(bool value);               // BOOL
    Json(const std::string &value); // STRING
    Json(std::string &&value);      // STRING
    Json(const char * value);       // STRING
    Json(const array &values);      // ARRAY
    Json(array &&values);           // ARRAY
    Json(const object &values);     // OBJECT
    Json(object &&values);          // OBJECT

    // Implicit constructor: anything with a to_json() function.
    template <class T, class = decltype(&T::to_json)>
    Json(const T & t) : Json(t.to_json()) {}

    // Implicit constructor: map-like objects (std::map, std::unordered_map, etc)
    template <class M, typename std::enable_if<
        std::is_constructible<std::string, decltype(std::declval<M>().begin()-
>first)>::value
        && std::is_constructible<Json, decltype(std::declval<M>().begin()-
>second)>::value,
            int>::type = 0>
    Json(const M & m) : Json(object(m.begin(), m.end())) {}
```

```
    // Implicit constructor: vector-like objects (std::list, std::vector,
std::set, etc)
    template <class V, typename std::enable_if<
        std::is_constructible<Json, decltype(*std::declval<V>().begin())>::value,
            int>::type = 0>
    Json(const V & v) : Json(array(v.begin(), v.end())) {}

    // This prevents Json(some_pointer) from accidentally producing a bool. Use
    // Json(bool(some_pointer)) if that behavior is desired.
    Json(void *) = delete;

    // Accessors
    Type type() const;

    bool is_null()   const { return type() == NUL; }
    bool is_number() const { return type() == NUMBER; }
    bool is_bool()   const { return type() == BOOL; }
    bool is_string() const { return type() == STRING; }
    bool is_array()  const { return type() == ARRAY; }
    bool is_object() const { return type() == OBJECT; }

    // Return the enclosed value if this is a number, 0 otherwise. Note that
json11 does not
    // distinguish between integer and non-integer numbers - number_value() and
int_value()
    // can both be applied to a NUMBER-typed object.
    double number_value() const;
    int int_value() const;

    // Return the enclosed value if this is a boolean, false otherwise.
    bool bool_value() const;
    // Return the enclosed string if this is a string, "" otherwise.
    const std::string &string_value() const;
    // Return the enclosed std::vector if this is an array, or an empty vector
otherwise.
    const array &array_items() const;
    // Return the enclosed std::map if this is an object, or an empty map
otherwise.
    const object &object_items() const;

    // Return a reference to arr[i] if this is an array, Json() otherwise.
    const Json & operator[](size_t i) const;
    // Return a reference to obj[key] if this is an object, Json() otherwise.
    const Json & operator[](const std::string &key) const;

    // Serialize.
    void dump(std::string &out) const;
    std::string dump() const {
        std::string out;
        dump(out);
```

```
        return out;
    }
    // Serialize.
    void dumpStruct(std::string &out) const;
    std::string dumpStruct() const {
        std::string out;
        dumpStruct(out);
        return out;
    }

    // Parse. If parse fails, return Json() and assign an error message to err.
    static Json parse(const std::string & in,
                      std::string & err,
                      JsonParse strategy = JsonParse::STANDARD);
    static Json parse(const char * in,
                      std::string & err,
                      JsonParse strategy = JsonParse::STANDARD) {
        if (in) {
            return parse(std::string(in), err, strategy);
        } else {
            err = "null input";
            return nullptr;
        }
    }
    // Parse multiple objects, concatenated or separated by whitespace
    static std::vector<Json> parse_multi(
        const std::string & in,
        std::string::size_type & parser_stop_pos,
        std::string & err,
        JsonParse strategy = JsonParse::STANDARD);

    static inline std::vector<Json> parse_multi(
        const std::string & in,
        std::string & err,
        JsonParse strategy = JsonParse::STANDARD) {
        std::string::size_type parser_stop_pos;
        return parse_multi(in, parser_stop_pos, err, strategy);
    }

    bool operator== (const Json &rhs) const;
    bool operator<  (const Json &rhs) const;
    bool operator!= (const Json &rhs) const { return !(*this == rhs); }
    bool operator<= (const Json &rhs) const { return !(rhs < *this); }
    bool operator>  (const Json &rhs) const { return  (rhs < *this); }
    bool operator>= (const Json &rhs) const { return !(*this < rhs); }

    /* has_shape(types, err)
     *
     * Return true if this is a JSON object and, for each item in types, has a
field of
     * the given type. If not, return false and set err to a descriptive message.
```

```
     */
    typedef std::initializer_list<std::pair<std::string, Type>> shape;
    bool has_shape(const shape & types, std::string & err) const;

private:
    std::shared_ptr<JsonValue> m_ptr;
};

// Internal class hierarchy - JsonValue objects are not exposed to users of this
API.
class JsonValue {
protected:
    friend class Json;
    friend class JsonInt;
    friend class JsonDouble;
    virtual Json::Type type() const = 0;
    virtual bool equals(const JsonValue * other) const = 0;
    virtual bool less(const JsonValue * other) const = 0;
    virtual void dump(std::string &out) const = 0;
    virtual void dumpStruct(std::string &out) const = 0;
    virtual double number_value() const;
    virtual int int_value() const;
    virtual bool bool_value() const;
    virtual const std::string &string_value() const;
    virtual const Json::array &array_items() const;
    virtual const Json &operator[](size_t i) const;
    virtual const Json::object &object_items() const;
    virtual const Json &operator[](const std::string &key) const;
    virtual ~JsonValue() {}
};

} // namespace json11
```

5. Класс для математических операций

```
#include <stdint.h>

class Math
{
public:
    static const float Pi;
    static const float PiX2;

    static float LoopFloatConstrain(float input, float minValue, float maxValue)
    {
        if (maxValue < minValue)
        {
            return input;
        }
```

```
    if (input > maxValue)
    {
        float len = maxValue - minValue;
        while (input > maxValue)
        {
            input -= len;
        }
    }
    else if (input < minValue)
    {
        float len = maxValue - minValue;
        while (input < minValue)
        {
            input += len;
        }
    }
    return input;
}

static float LimitMax(float _input, float _max)
{
    if(_input > _max)
    {
        return _max;
    }
    else if(_input < -_max)
    {
        return -_max;
    }
    return _input;
}

static float FloatConstrain(float Input, float minValue, float maxValue)
{
    if (maxValue < minValue)
    {
        return Input;
    }

    if (Input > maxValue)
    {
        Input = maxValue;
    }
    else if (Input < minValue)
    {
        Input = minValue;
    }
    return Input;
}
```

```
    static uint32_t ConvertToFixed(float _inNum, float _inMin, float
_inPrecision)
    {
        return (uint32_t)((_inNum - _inMin) / _inPrecision);
    }

    static float ConvertFromFixed(uint32_t _inNum, float _inMin, float
_inPrecision)
    {
        return (float)(_inNum) * _inPrecision + _inMin;
    }
};
```

6. Математические константы

```
#include "Math.hpp"

const float Math::Pi = 3.14159265358979f;
const float Math::PiX2 = 6.283185307f;
```

7. API по работе с протоколом MODBUS

```
/*
 * Copyright © 2001-2011 Stéphane Raimbault <stephane.raimbault@gmail.com>
 *
 * SPDX-License-Identifier: LGPL-2.1+
 *
 * This library implements the Modbus protocol.
 * http://libmodbus.org/
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdarg.h>
```

```c
#include <errno.h>
#include <limits.h>
#include <time.h>
#ifndef _MSC_VER
#include <unistd.h>
#endif
#include "modbus.h"

/* Internal use */
#define MSG_LENGTH_UNDEFINED -1
modbus_mapping_t *mb_mapping;
/* Exported version */
const unsigned int libmodbus_version_major = 1;
const unsigned int libmodbus_version_minor = 0;
const unsigned int libmodbus_version_micro = 0;
HR_t hr_param;

/* 3 steps are used to parse the query */
typedef enum {
    _STEP_FUNCTION,
    _STEP_META,
    _STEP_DATA
} _step_t;

/* Builds a TCP response header */
static int modbus_tcp_build_response_basis(sft_t *sft, uint8_t *rsp)
{
    /* Extract from MODBUS Messaging on TCP/IP Implementation
       Guide V1.0b (page 23/46):
       The transaction identifier is used to associate the future
       response with the request. */
    rsp[0] = sft->t_id >> 8;
    rsp[1] = sft->t_id & 0x00ff;

    /* Protocol Modbus */
    rsp[2] = 0;
    rsp[3] = 0;

    /* Length will be set later by send_msg (4 and 5) */

    /* The slave ID is copied from the indication */
    rsp[6] = sft->slave;
    rsp[7] = sft->function;

    return _MODBUS_TCP_PRESET_RSP_LENGTH;
}

static int modbus_tcp_prepare_response_tid(const uint8_t *req, int *req_length)
{
    return (req[0] << 8) + req[1];
}
```

```c
/* Build the exception response */
static int response_exception( sft_t *sft,
                               int exception_code, uint8_t *rsp,
                               unsigned int to_flush,
                               const char* template, ...)
{
    int rsp_length;

    /* Build exception response */
    sft->function = sft->function + 0x80;
    rsp_length = modbus_tcp_build_response_basis(sft, rsp);
    rsp[rsp_length++] = exception_code;

    return rsp_length;
}

static int _modbus_tcp_send_msg_pre(uint8_t *req, int req_length)
{
    /* Substract the header length to the message length */
    int mbap_length = req_length - 6;

    req[4] = mbap_length >> 8;
    req[5] = mbap_length & 0x00FF;

    return req_length;
}


/* Send a response to the received request.
   Analyses the request and constructs a response.

   If an error occurs, this function construct the response
   accordingly.
*/
int modbus_reply(uint8_t *rsp, const uint8_t *req,
                 int req_length, modbus_mapping_t *mb_mapping)
{
    int offset;
    int slave;
    int function;
    uint16_t address;
    //uint8_t rsp[MAX_MESSAGE_LENGTH];
    int rsp_length = 0;
    sft_t sft;

    offset = _MODBUS_TCP_HEADER_LENGTH;
    slave = req[offset - 1];
    function = req[offset];
    address = (req[offset + 1] << 8) + req[offset + 2];
```

```
    sft.slave = slave;
    sft.function = function;
    sft.t_id = modbus_tcp_prepare_response_tid(req, &req_length);

    /* Data are flushed on illegal number of values errors. */
    switch (function) {
    case MODBUS_FC_READ_HOLDING_REGISTERS:
    case MODBUS_FC_READ_INPUT_REGISTERS: {
        unsigned int is_input = (function == MODBUS_FC_READ_INPUT_REGISTERS);
        int start_registers = is_input ? mb_mapping->start_input_registers :
mb_mapping->start_registers;
        int nb_registers = is_input ? mb_mapping->nb_input_registers :
mb_mapping->nb_registers;
        uint16_t *tab_registers = is_input ? mb_mapping->tab_input_registers :
mb_mapping->tab_registers;
        const char * const name = is_input ? "read_input_registers" :
"read_registers";
        int nb = (req[offset + 3] << 8) + req[offset + 4];
        /* The mapping can be shifted to reduce memory consumption and it
           doesn't always start at address zero. */
        int mapping_address = address - start_registers;

        if (nb < 1 || MODBUS_MAX_READ_REGISTERS < nb) {
            rsp_length = response_exception(
                &sft, MODBUS_EXCEPTION_ILLEGAL_DATA_VALUE, rsp, TRUE,
                "Illegal nb of values %d in %s (max %d)\n",
                nb, name, MODBUS_MAX_READ_REGISTERS);
        } else if (mapping_address < 0 || (mapping_address + nb) > nb_registers)
{
            rsp_length = response_exception(
                 &sft, MODBUS_EXCEPTION_ILLEGAL_DATA_ADDRESS, rsp, FALSE,
                "Illegal data address 0x%0X in %s\n",
                mapping_address < 0 ? address : address + nb, name);
        } else {
            int i;

            rsp_length = modbus_tcp_build_response_basis(&sft, rsp);
            rsp[rsp_length++] = nb << 1;
            for (i = mapping_address; i < mapping_address + nb; i++) {
                rsp[rsp_length++] = tab_registers[i] >> 8;
                rsp[rsp_length++] = tab_registers[i] & 0xFF;
            }
        }
    }
        break;
    case MODBUS_FC_WRITE_SINGLE_REGISTER: {
        int mapping_address = address - mb_mapping->start_registers;

        if (mapping_address < 0 || mapping_address >= mb_mapping->nb_registers) {
            rsp_length = response_exception(
                 &sft,
```

```
                    MODBUS_EXCEPTION_ILLEGAL_DATA_ADDRESS, rsp, FALSE,
                    "Illegal data address 0x%0X in write_register\n",
                    address);
            } else {
                int data = (req[offset + 3] << 8) + req[offset + 4];

                mb_mapping->tab_registers[mapping_address] = data;
                memcpy(rsp, req, req_length);
                rsp_length = req_length;
            }
        }
            break;
        case MODBUS_FC_WRITE_MULTIPLE_REGISTERS: {
            int nb = (req[offset + 3] << 8) + req[offset + 4];
            int mapping_address = address - mb_mapping->start_registers;

            if (nb < 1 || MODBUS_MAX_WRITE_REGISTERS < nb) {
                rsp_length = response_exception(
                     &sft, MODBUS_EXCEPTION_ILLEGAL_DATA_VALUE, rsp, TRUE,
                    "Illegal number of values %d in write_registers (max %d)\n",
                    nb, MODBUS_MAX_WRITE_REGISTERS);
            } else if (mapping_address < 0 ||
                       (mapping_address + nb) > mb_mapping->nb_registers) {
                rsp_length = response_exception(
                     &sft, MODBUS_EXCEPTION_ILLEGAL_DATA_ADDRESS, rsp, FALSE,
                    "Illegal data address 0x%0X in write_registers\n",
                    mapping_address < 0 ? address : address + nb);
            } else {
                int i, j;
                for (i = mapping_address, j = 6; i < mapping_address + nb; i++, j +=
2) {
                    /* 6 and 7 = first value */
                    mb_mapping->tab_registers[i] =
                        (req[offset + j] << 8) + req[offset + j + 1];
                }

                rsp_length = modbus_tcp_build_response_basis(&sft, rsp);
                /* 4 to copy the address (2) and the no. of registers */
                memcpy(rsp + rsp_length, req + rsp_length, 4);
                rsp_length += 4;
            }
        }
            break;
        case MODBUS_FC_WRITE_AND_READ_REGISTERS: {
            int nb = (req[offset + 3] << 8) + req[offset + 4];
            uint16_t address_write = (req[offset + 5] << 8) + req[offset + 6];
            int nb_write = (req[offset + 7] << 8) + req[offset + 8];
            int nb_write_bytes = req[offset + 9];
            int mapping_address = address - mb_mapping->start_registers;
            int mapping_address_write = address_write - mb_mapping->start_registers;
```

```
        if (nb_write < 1 || MODBUS_MAX_WR_WRITE_REGISTERS < nb_write ||
            nb < 1 || MODBUS_MAX_WR_READ_REGISTERS < nb ||
            nb_write_bytes != nb_write * 2) {
            rsp_length = response_exception(
                 &sft, MODBUS_EXCEPTION_ILLEGAL_DATA_VALUE, rsp, TRUE,
                "Illegal nb of values (W%d, R%d) in write_and_read_registers (max
W%d, R%d)\n",
                nb_write, nb, MODBUS_MAX_WR_WRITE_REGISTERS,
MODBUS_MAX_WR_READ_REGISTERS);
        } else if (mapping_address < 0 ||
                   (mapping_address + nb) > mb_mapping->nb_registers ||
                   mapping_address < 0 ||
                   (mapping_address_write + nb_write) > mb_mapping->nb_registers)
{
            rsp_length = response_exception(
                 &sft, MODBUS_EXCEPTION_ILLEGAL_DATA_ADDRESS, rsp, FALSE,
                "Illegal data read address 0x%0X or write address 0x%0X
write_and_read_registers\n",
                mapping_address < 0 ? address : address + nb,
                mapping_address_write < 0 ? address_write : address_write +
nb_write);
        } else {
            int i, j;
            rsp_length = modbus_tcp_build_response_basis(&sft, rsp);
            rsp[rsp_length++] = nb << 1;

            /* Write first.
               10 and 11 are the offset of the first values to write */
            for (i = mapping_address_write, j = 10;
                 i < mapping_address_write + nb_write; i++, j += 2) {
                mb_mapping->tab_registers[i] =
                    (req[offset + j] << 8) + req[offset + j + 1];
            }

            /* and read the data for the response */
            for (i = mapping_address; i < mapping_address + nb; i++) {
                rsp[rsp_length++] = mb_mapping->tab_registers[i] >> 8;
                rsp[rsp_length++] = mb_mapping->tab_registers[i] & 0xFF;
            }
        }
    }
        break;

    default:
        rsp_length = response_exception(
             &sft, MODBUS_EXCEPTION_ILLEGAL_FUNCTION, rsp, TRUE,
            "Unknown Modbus function code: 0x%0X\n", function);
        break;
    }
    _modbus_tcp_send_msg_pre(rsp,rsp_length);
    /* Suppress any responses when the request was a broadcast */
```

```
    return rsp_length;
}

/* Allocates 4 arrays to store bits, input bits, registers and inputs
   registers. The pointers are stored in modbus_mapping structure.

   The modbus_mapping_new_ranges() function shall return the new allocated
   structure if successful. Otherwise it shall return NULL and set errno to
   ENOMEM. */
modbus_mapping_t* modbus_mapping_new_start_address(
    unsigned int start_bits, unsigned int nb_bits,
    unsigned int start_input_bits, unsigned int nb_input_bits,
    unsigned int start_registers, unsigned int nb_registers,
    unsigned int start_input_registers, unsigned int nb_input_registers)
{
    modbus_mapping_t *mb_mapping;

    mb_mapping = (modbus_mapping_t *)malloc(sizeof(modbus_mapping_t));
    if (mb_mapping == NULL) {
        return NULL;
    }

    /* 0X */
    mb_mapping->nb_bits = nb_bits;
    mb_mapping->start_bits = start_bits;
    if (nb_bits == 0) {
        mb_mapping->tab_bits = NULL;
    } else {
        /* Negative number raises a POSIX error */
        mb_mapping->tab_bits =
            (uint8_t *) malloc(nb_bits * sizeof(uint8_t));
        if (mb_mapping->tab_bits == NULL) {
            free(mb_mapping);
            return NULL;
        }
        memset(mb_mapping->tab_bits, 0, nb_bits * sizeof(uint8_t));
    }

    /* 1X */
    mb_mapping->nb_input_bits = nb_input_bits;
    mb_mapping->start_input_bits = start_input_bits;
    if (nb_input_bits == 0) {
        mb_mapping->tab_input_bits = NULL;
    } else {
        mb_mapping->tab_input_bits =
            (uint8_t *) malloc(nb_input_bits * sizeof(uint8_t));
        if (mb_mapping->tab_input_bits == NULL) {
            free(mb_mapping->tab_bits);
            free(mb_mapping);
            return NULL;
        }
```

```
        memset(mb_mapping->tab_input_bits, 0, nb_input_bits * sizeof(uint8_t));
    }

    /* 4X */
    mb_mapping->nb_registers = nb_registers;
    mb_mapping->start_registers = start_registers;
    if (nb_registers == 0) {
        mb_mapping->tab_registers = NULL;
    } else {
        mb_mapping->tab_registers =
            (uint16_t *) malloc(nb_registers * sizeof(uint16_t));
        if (mb_mapping->tab_registers == NULL) {
            free(mb_mapping->tab_input_bits);
            free(mb_mapping->tab_bits);
            free(mb_mapping);
            return NULL;
        }
        memset(mb_mapping->tab_registers, 0, nb_registers * sizeof(uint16_t));
    }

    /* 3X */
    mb_mapping->nb_input_registers = nb_input_registers;
    mb_mapping->start_input_registers = start_input_registers;
    if (nb_input_registers == 0) {
        mb_mapping->tab_input_registers = NULL;
    } else {
        mb_mapping->tab_input_registers =
            (uint16_t *) malloc(nb_input_registers * sizeof(uint16_t));
        if (mb_mapping->tab_input_registers == NULL) {
            free(mb_mapping->tab_registers);
            free(mb_mapping->tab_input_bits);
            free(mb_mapping->tab_bits);
            free(mb_mapping);
            return NULL;
        }
        memset(mb_mapping->tab_input_registers, 0,
               nb_input_registers * sizeof(uint16_t));
    }

    return mb_mapping;
}

modbus_mapping_t* modbus_mapping_new(int nb_bits, int nb_input_bits,
                                     int nb_registers, int nb_input_registers)
{
    return modbus_mapping_new_start_address(
        0, nb_bits, 0, nb_input_bits, 0, nb_registers, 0, nb_input_registers);
}

/* Frees the 4 arrays */
void modbus_mapping_free(modbus_mapping_t *mb_mapping)
```

```
{
    if (mb_mapping == NULL) {
        return;
    }

    free(mb_mapping->tab_input_registers);
    free(mb_mapping->tab_registers);
    free(mb_mapping->tab_input_bits);
    free(mb_mapping->tab_bits);
    free(mb_mapping);
}
```

8. Структуры данных для API MODBUS

```
#ifndef MODBUS_H
#define MODBUS_H

/* Add this for macros that defined unix flavor */
#if (defined(__unix__) || defined(unix)) && !defined(USG)
#include <sys/param.h>
#endif

#ifndef _MSC_VER
#include <stdint.h>
#else
#include "stdint.h"
#endif


#if defined(_MSC_VER)
# if defined(DLLBUILD)
/* define DLLBUILD when building the DLL */
#  define MODBUS_API __declspec(dllexport)
# else
#  define MODBUS_API __declspec(dllimport)
# endif
#else
# define MODBUS_API
#endif

#ifdef  __cplusplus
# define MODBUS_BEGIN_DECLS  extern "C" {
# define MODBUS_END_DECLS    }
#else
# define MODBUS_BEGIN_DECLS
```

```
# define MODBUS_END_DECLS
#endif

MODBUS_BEGIN_DECLS

#ifndef FALSE
#define FALSE 0
#endif

#ifndef TRUE
#define TRUE 1
#endif

#ifndef OFF
#define OFF 0
#endif

#ifndef ON
#define ON 1
#endif

/* Modbus function codes */
#define MODBUS_FC_READ_COILS                0x01
#define MODBUS_FC_READ_DISCRETE_INPUTS      0x02
#define MODBUS_FC_READ_HOLDING_REGISTERS    0x03
#define MODBUS_FC_READ_INPUT_REGISTERS      0x04
#define MODBUS_FC_WRITE_SINGLE_COIL         0x05
#define MODBUS_FC_WRITE_SINGLE_REGISTER     0x06
#define MODBUS_FC_READ_EXCEPTION_STATUS     0x07
#define MODBUS_FC_WRITE_MULTIPLE_COILS      0x0F
#define MODBUS_FC_WRITE_MULTIPLE_REGISTERS  0x10
#define MODBUS_FC_REPORT_SLAVE_ID           0x11
#define MODBUS_FC_MASK_WRITE_REGISTER       0x16
#define MODBUS_FC_WRITE_AND_READ_REGISTERS  0x17

#define _MODBUS_TCP_HEADER_LENGTH      7
#define _MODBUS_TCP_PRESET_REQ_LENGTH 12
#define _MODBUS_TCP_PRESET_RSP_LENGTH  8

#define _MODBUS_TCP_CHECKSUM_LENGTH    0
#define MODBUS_BROADCAST_ADDRESS    0

/* Modbus_Application_Protocol_V1_1b.pdf (chapter 6 section 1 page 12)
 * Quantity of Coils to read (2 bytes): 1 to 2000 (0x7D0)
 * (chapter 6 section 11 page 29)
 * Quantity of Coils to write (2 bytes): 1 to 1968 (0x7B0)
 */
#define MODBUS_MAX_READ_BITS              2000
#define MODBUS_MAX_WRITE_BITS             1968

/* Modbus_Application_Protocol_V1_1b.pdf (chapter 6 section 3 page 15)
```

```
 * Quantity of Registers to read (2 bytes): 1 to 125 (0x7D)
 * (chapter 6 section 12 page 31)
 * Quantity of Registers to write (2 bytes) 1 to 123 (0x7B)
 * (chapter 6 section 17 page 38)
 * Quantity of Registers to write in R/W registers (2 bytes) 1 to 121 (0x79)
 */
#define MODBUS_MAX_READ_REGISTERS          125
#define MODBUS_MAX_WRITE_REGISTERS         123
#define MODBUS_MAX_WR_WRITE_REGISTERS      121
#define MODBUS_MAX_WR_READ_REGISTERS       125

/* The size of the MODBUS PDU is limited by the size constraint inherited from
 * the first MODBUS implementation on Serial Line network (max. RS485 ADU = 256
 * bytes). Therefore, MODBUS PDU for serial line communication = 256 - Server
 * address (1 byte) - CRC (2 bytes) = 253 bytes.
 */
#define MODBUS_MAX_PDU_LENGTH              253

/* Max between RTU and TCP max adu length (so TCP) */
#define MAX_MESSAGE_LENGTH 260
/* Consequently:
 * - RTU MODBUS ADU = 253 bytes + Server address (1 byte) + CRC (2 bytes) = 256
 *   bytes.
 * - TCP MODBUS ADU = 253 bytes + MBAP (7 bytes) = 260 bytes.
 * so the maximum of both backend in 260 bytes. This size can used to allocate
 * an array of bytes to store responses and it will be compatible with the two
 * backends.
 */
#define MODBUS_MAX_ADU_LENGTH              260

/* Random number to avoid errno conflicts */
#define MODBUS_ENOBASE 112345678

/* Protocol exceptions */
enum {
    MODBUS_EXCEPTION_ILLEGAL_FUNCTION = 0x01,
    MODBUS_EXCEPTION_ILLEGAL_DATA_ADDRESS,
    MODBUS_EXCEPTION_ILLEGAL_DATA_VALUE,
    MODBUS_EXCEPTION_SLAVE_OR_SERVER_FAILURE,
    MODBUS_EXCEPTION_ACKNOWLEDGE,
    MODBUS_EXCEPTION_SLAVE_OR_SERVER_BUSY,
    MODBUS_EXCEPTION_NEGATIVE_ACKNOWLEDGE,
    MODBUS_EXCEPTION_MEMORY_PARITY,
    MODBUS_EXCEPTION_NOT_DEFINED,
    MODBUS_EXCEPTION_GATEWAY_PATH,
    MODBUS_EXCEPTION_GATEWAY_TARGET,
    MODBUS_EXCEPTION_MAX
};

#define EMBXILFUN  (MODBUS_ENOBASE + MODBUS_EXCEPTION_ILLEGAL_FUNCTION)
#define EMBXILADD  (MODBUS_ENOBASE + MODBUS_EXCEPTION_ILLEGAL_DATA_ADDRESS)
```

```
#define EMBXILVAL  (MODBUS_ENOBASE + MODBUS_EXCEPTION_ILLEGAL_DATA_VALUE)
#define EMBXSFAIL  (MODBUS_ENOBASE + MODBUS_EXCEPTION_SLAVE_OR_SERVER_FAILURE)
#define EMBXACK    (MODBUS_ENOBASE + MODBUS_EXCEPTION_ACKNOWLEDGE)
#define EMBXSBUSY  (MODBUS_ENOBASE + MODBUS_EXCEPTION_SLAVE_OR_SERVER_BUSY)
#define EMBXNACK   (MODBUS_ENOBASE + MODBUS_EXCEPTION_NEGATIVE_ACKNOWLEDGE)
#define EMBXMEMPAR (MODBUS_ENOBASE + MODBUS_EXCEPTION_MEMORY_PARITY)
#define EMBXGPATH  (MODBUS_ENOBASE + MODBUS_EXCEPTION_GATEWAY_PATH)
#define EMBXGTAR   (MODBUS_ENOBASE + MODBUS_EXCEPTION_GATEWAY_TARGET)

/* Native libmodbus error codes */
#define EMBBADCRC  (EMBXGTAR + 1)
#define EMBBADDATA (EMBXGTAR + 2)
#define EMBBADEXC  (EMBXGTAR + 3)
#define EMBUNKEXC  (EMBXGTAR + 4)
#define EMBMDATA   (EMBXGTAR + 5)
#define EMBBADSLAVE (EMBXGTAR + 6)

extern const unsigned int libmodbus_version_major;
extern const unsigned int libmodbus_version_minor;
extern const unsigned int libmodbus_version_micro;

typedef struct _modbus modbus_t;

typedef struct {
    int nb_bits;
    int start_bits;
    int nb_input_bits;
    int start_input_bits;
    int nb_input_registers;
    int start_input_registers;
    int nb_registers;
    int start_registers;
    uint8_t *tab_bits;
    uint8_t *tab_input_bits;
    uint16_t *tab_input_registers;
    uint16_t *tab_registers;
} modbus_mapping_t;

typedef struct _sft {
    int slave;
    int function;
    int t_id;
} sft_t;

typedef enum
{
    MODBUS_ERROR_RECOVERY_NONE          = 0,
    MODBUS_ERROR_RECOVERY_LINK          = (1<<1),
    MODBUS_ERROR_RECOVERY_PROTOCOL      = (1<<2)
} modbus_error_recovery_mode;
```

```
typedef struct {
  int8_t Motor_Voltage;
  int8_t Logic_Voltage;
  int8_t Source_Voltage;
  int8_t Voltage_10V;
  int8_t Voltage_18V;
  int8_t Hall_Voltage;
}ADC1_t;

typedef struct {
  int8_t Sensor_Voltage;
  int8_t Motor_Current;
  int8_t CPU_Temperature;
  int8_t unused;
  uint16_t uiCustomADC;
} ADC2_t;

typedef struct {
  uint16_t rSpeed;
  uint8_t ucMotorType;
  uint8_t ucRampUp;
  uint8_t ucRampDown;
  uint8_t OvercurrentLimit;
} CDatasheet_t;


typedef struct {
    u_int16_t N1_MotorStatus[32];
    u_int16_t N2_MotorStatus[32];

    u_int16_t N1_MotorCMD[32];
    u_int16_t N2_MotorCMD[32];

    ADC1_t N1_MotorADC1[32];
    ADC1_t N2_MotorADC1[32];

    ADC2_t N1_MotorADC2[32];
    ADC2_t N2_MotorADC2[32];

    CDatasheet_t N1_MotorParam[32];
    CDatasheet_t N2_MotorParam[32];

    u_int16_t N1_HMIMotorCMD[32];
    u_int16_t N2_HMIMotorCMD[32];

    uint8_t N1_SelectADC[32];
    uint8_t N2_SelectADC[32];

    uint8_t ip[4];
    uint8_t mask[4];
    u_int16_t xHMICmd;
```

```
} HR_t;

extern HR_t hr_param;

MODBUS_API const char *modbus_strerror(int errnum);

MODBUS_API modbus_mapping_t* modbus_mapping_new_start_address(
    unsigned int start_bits, unsigned int nb_bits,
    unsigned int start_input_bits, unsigned int nb_input_bits,
    unsigned int start_registers, unsigned int nb_registers,
    unsigned int start_input_registers, unsigned int nb_input_registers);

MODBUS_API modbus_mapping_t* modbus_mapping_new(int nb_bits, int nb_input_bits,
                                                int nb_registers, int
nb_input_registers);
MODBUS_API void modbus_mapping_free(modbus_mapping_t *mb_mapping);

MODBUS_API int modbus_reply(uint8_t *rsp, const uint8_t *req,
                            int req_length, modbus_mapping_t *mb_mapping);
MODBUS_API int modbus_reply_exception(modbus_t *ctx, const uint8_t *req,
                                      unsigned int exception_code);


extern modbus_mapping_t *mb_mapping;
/**
 * UTILS FUNCTIONS
 **/

#define MODBUS_GET_HIGH_BYTE(data) (((data) >> 8) & 0xFF)
#define MODBUS_GET_LOW_BYTE(data) ((data) & 0xFF)
#define MODBUS_GET_INT64_FROM_INT16(tab_int16, index) \
    (((int64_t)tab_int16[(index)    ] << 48) + \
     ((int64_t)tab_int16[(index) + 1] << 32) + \
     ((int64_t)tab_int16[(index) + 2] << 16) + \
      (int64_t)tab_int16[(index) + 3])
#define MODBUS_GET_INT32_FROM_INT16(tab_int16, index) ((tab_int16[(index)] << 16)
+ tab_int16[(index) + 1])
#define MODBUS_GET_INT16_FROM_INT8(tab_int8, index) ((tab_int8[(index)] << 8) +
tab_int8[(index) + 1])
#define MODBUS_SET_INT16_TO_INT8(tab_int8, index, value) \
    do { \
        tab_int8[(index)] = (value) >> 8;  \
        tab_int8[(index) + 1] = (value) & 0xFF; \
    } while (0)
#define MODBUS_SET_INT32_TO_INT16(tab_int16, index, value) \
    do { \
        tab_int16[(index)    ] = (value) >> 16; \
        tab_int16[(index) + 1] = (value); \
    } while (0)
#define MODBUS_SET_INT64_TO_INT16(tab_int16, index, value) \
    do { \
```

```
        tab_int16[(index)    ] = (value) >> 48; \
        tab_int16[(index) + 1] = (value) >> 32; \
        tab_int16[(index) + 2] = (value) >> 16; \
        tab_int16[(index) + 3] = (value); \
    } while (0)

MODBUS_API void modbus_set_bits_from_byte(uint8_t *dest, int idx, const uint8_t
value);
MODBUS_API void modbus_set_bits_from_bytes(uint8_t *dest, int idx, unsigned int
nb_bits,
                                           const uint8_t *tab_byte);
MODBUS_API uint8_t modbus_get_byte_from_bits(const uint8_t *src, int idx,
unsigned int nb_bits);
MODBUS_API float modbus_get_float(const uint16_t *src);
MODBUS_API float modbus_get_float_abcd(const uint16_t *src);
MODBUS_API float modbus_get_float_dcba(const uint16_t *src);
MODBUS_API float modbus_get_float_badc(const uint16_t *src);
MODBUS_API float modbus_get_float_cdab(const uint16_t *src);

MODBUS_API void modbus_set_float(float f, uint16_t *dest);
MODBUS_API void modbus_set_float_abcd(float f, uint16_t *dest);
MODBUS_API void modbus_set_float_dcba(float f, uint16_t *dest);
MODBUS_API void modbus_set_float_badc(float f, uint16_t *dest);
MODBUS_API void modbus_set_float_cdab(float f, uint16_t *dest);

MODBUS_END_DECLS

#endif  /* MODBUS_H */
```

9. TCP/IP имплементация сервера

```
#include "TCPServer.h"
#include <netinet/tcp.h>
#include "modbus.h"

int TCPServer::num_client;
int TCPServer::last_closed;
bool TCPServer::isonline[MAX_PORT];
vector<descript_socket*> TCPServer::Message;
vector<descript_socket*> TCPServer::newsockfd;
std::mutex TCPServer::mt;

void* TCPServer::Task(void *arg)
```

```
{
    int n;
    struct descript_socket *desc = (struct descript_socket*) arg;
    //pthread_detach(pthread_self());

    cerr << "open client[ id:"<< desc->id <<" ip:"<< desc->ip <<" socket:"<<
desc->socket<<" send:"<< desc->enable_message_runtime <<" ]" << endl;
    while(1)
    {
        n = recv(desc->socket, desc->msg, MAXPACKETSIZE, 0);
        if(n != -1)
        {
            if(n==0)
            {
               isonline[desc->sockd] = false;
               cerr << "close client[ id:"<< desc->id <<" ip:"<< desc->ip <<"
socket:"<< desc->socket<<" ]" << endl;
               last_closed = desc->id;
               close(desc->socket);

               int id = desc->id;
               auto new_end = std::remove_if(newsockfd.begin(), newsockfd.end(),
                                                    [id](descript_socket *device)
                                                        { return device->id == id;
});
               newsockfd.erase(new_end, newsockfd.end());

               if(num_client>0) num_client--;
               break;
            }
            if(n > 0){
                desc->msg[n]=0;
                desc->byte_cnt = n;
                if (n >= 5) {
                    uint8_t rsp[MAX_MESSAGE_LENGTH];
                    uint16_t nn = modbus_reply(rsp, (uint8_t*)desc->msg, desc-
>byte_cnt, mb_mapping);
                    //GetMemoryTable();
                    if (nn > 0){
                        send(desc->socket,rsp,nn,0);
                        n=0;
                    }
                }
            }

        }
        std::this_thread::sleep_for(std::chrono::milliseconds(1));
    }
    if(desc != NULL)
    {
        delete desc;
```

```
        desc = NULL;
    }
    cerr << "exit thread: " << this_thread::get_id() << endl;
    pthread_exit(NULL);

    return 0;
}

int TCPServer::setnonblocking(int fd)
{
    int flags;
    if((flags = fcntl(fd, F_GETFD, 0)) < 0)
    {
        printf("get falg error\n");
        return -1;
    }

    flags |= O_NONBLOCK;
    if (fcntl(fd, F_SETFL, flags) < 0) {
        printf("set nonblock fail\n");
        return -1;
    }
    return 0;
}

int TCPServer::setup(int port, vector<int> opts)
{
    int cport=-1;
    for(int i = 0; i<MAX_PORT;i++)
        if(sockfd[i]==0){
            cport = i;
            break;}
    if(cport>=0){
        int opt = 1;
        isonline[cport] = false;
        last_closed = -1;

        bzero(&serverAddress, sizeof(struct sockaddr_in));
        serverAddress.sin_family = AF_INET;
        serverAddress.sin_addr.s_addr = INADDR_ANY;
        serverAddress.sin_port = htons(port);

        sockfd[cport] = socket(AF_INET,SOCK_STREAM,0);
        if (sockfd[cport] < 0) {
            perror("socket error");
            return -1;
        }
        if (setnonblocking(sockfd[cport]) < 0) {
            perror("setnonblock error");
        }
```

```
        setsockopt(sockfd[cport], SOL_SOCKET, SO_REUSEPORT, (char *) &opt,
sizeof(int));
        setsockopt(sockfd[cport], SOL_SOCKET, SO_REUSEADDR, (char *) &opt,
sizeof(int));
        setsockopt(sockfd[cport], SOL_SOCKET, TCP_NODELAY, (char *) &opt,
sizeof(int));
        /*fcntl(sockfd[cport], F_SETFL, O_NONBLOCK);

        serverAddress.sin_family      = AF_INET;
        serverAddress.sin_addr.s_addr = htonl(INADDR_ANY);
        serverAddress.sin_port        = htons(port);*/
        portfd[cport] = port;
        if((::bind(sockfd[cport],(struct sockaddr *)&serverAddress,
sizeof(serverAddress))) < 0){
            cerr << "Errore bind" << endl;
            return -1;
        }

        if(listen(sockfd[cport],5) < 0){
            cerr << "Errore listen" << endl;
            return -1;
        }
        num_client = 0;
        isonline[cport] = true;
    }else return -1;
    return 0;
}

void TCPServer::accepted(bool need_task)
{
    int mx=0;
    FD_ZERO(&readset);
    for(int i = 0; i<MAX_PORT;i++){
        FD_SET(sockfd[i], &readset);
        if(sockfd[i]>mx)mx=sockfd[i];
    }

    timeval timeout;
    timeout.tv_sec = 0;
    timeout.tv_usec = 0;
    if(select(mx+1, &readset, NULL, NULL, &timeout) > 0){
        for(int i = 0; i<MAX_PORT;i++){
            if(FD_ISSET(sockfd[i], &readset))
            {
                socklen_t sosize    = sizeof(clientAddress);
                descript_socket *so = new descript_socket;
                so->socket          = accept(sockfd[i],(struct
sockaddr*)&clientAddress,&sosize);
                if (so->socket < 0 ) {
                    std::cout << "socket err " <<sockfd[i]<<" "<< so->socket <<
endl;
```

```
                    //printf("socket err\n");
                }
                else{
                    std::cout << "socket ok " <<sockfd[i]<<" "<< so->socket <<
endl;
                    so->id              = num_client;
                    so->ip              = inet_ntoa(clientAddress.sin_addr);
                    so->port            = portfd[i];
                    so->sockd           = i;
                    newsockfd.push_back( so );
                    cerr << "accept client[ id:" << newsockfd[num_client]->id <<
                                        " ip:" << newsockfd[num_client]->ip <<
                                        " port:" << newsockfd[num_client]->port
<<
                                    " handle:" << newsockfd[num_client]->socket
<< " ]" << endl;
                    if(need_task)
                        pthread_create(&serverThread[num_client], NULL, &Task,
(void *)newsockfd[num_client]);
                    isonline[i]=true;
                    num_client++;
                }
            }
        }
    }
}

vector<descript_socket*> TCPServer::getMessage()
{
    std::lock_guard<std::mutex> guard(mt);
    return Message;
}

vector<descript_socket*> TCPServer::getSocket()
{
    std::lock_guard<std::mutex> guard(mt);
    return newsockfd;
}

void TCPServer::Send(string msg, int id)
{
    send(newsockfd[id]->socket,msg.c_str(),msg.length(),0);
}

void TCPServer::SendChar(char *msg,uint16_t len, int id)
{
    send(newsockfd[id]->socket,msg,len,0);
}

int TCPServer::get_last_closed_sockets()
{
```

```
    return last_closed;
}

void TCPServer::clean(int id)
{
    Message[id] = NULL;
    //memset(msg, 0, MAXPACKETSIZE);
}

string TCPServer::get_ip_addr(int id)
{
    return newsockfd[id]->ip;
}

bool TCPServer::is_online()
{
    return isonline;
}

void TCPServer::detach(int id)
{
    close(newsockfd[id]->socket);
    newsockfd[id]->ip = "";
    newsockfd[id]->id = -1;
    newsockfd[id]->port = -1;
    newsockfd[id]->message = "";
}

void TCPServer::closed()
{
    for(int i = 0; i<MAX_PORT;i++){
        close(sockfd[i]);
        portfd[i]=0;
    }
}
```

10. Структуры и классы для работы с TCP/IP

```
#ifndef TCP_SERVER_H
#define TCP_SERVER_H

#include <iostream>
#include <vector>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
```

```cpp
#include <netinet/in.h>
#include <string.h>
#include <arpa/inet.h>
#include <pthread.h>
#include <thread>
#include <algorithm>
#include <cctype>
#include <mutex>
#include <fcntl.h>

using namespace std;

#define MAXPACKETSIZE 20000
#define DESCPACKETSIZE 500
#define MAX_CLIENT 1000
#define MAX_PORT 1
//#define CODA_MSG 4

struct descript_socket{
    int socket     = -1;
    string ip      = "";
    int id         = -1;
    int port       = -1;
    int sockd      = -1;
    char msg[MAXPACKETSIZE];
    int byte_cnt   = -1;
    std::string message;
    bool enable_message_runtime = false;
};

class TCPServer
{
    public:
    int setup(int port, vector<int> opts = vector<int>());
    vector<descript_socket*> getMessage();
    vector<descript_socket*> getSocket();
    void accepted(bool need_task);
    void SendChar(char *msg,uint16_t len, int id);
    void Send(string msg, int id);
    void detach(int id);
    void clean(int id);
    bool is_online();
    string get_ip_addr(int id);
    int get_last_closed_sockets();
    void closed();
    int setnonblocking(int fd);
    fd_set readset;
    private:
    int sockfd[MAX_PORT], n, pid,portfd[MAX_PORT];
    struct sockaddr_in serverAddress;
    struct sockaddr_in clientAddress;
```

```
    pthread_t serverThread[ MAX_CLIENT ];

    static vector<descript_socket*> newsockfd;
    static vector<descript_socket*> Message;//[CODA_MSG];

    static bool isonline[MAX_PORT];
    static int last_closed;
    static int num_client;
    static std::mutex mt;
    static void * Task(void * argv);
};

#endif
```

11. API по работе с моторами

```
#include "MotorManager.hpp"
#include "Motor.hpp"
#include <time.h>

#define ST_ADDR_START 0
#define ST_ADDR_NULL 40
#define ST_LINK_RESET 80
#define ST_ADDR_SET 100
#define ST_ADDR_WAIT 120
#define ST_ADDR_OK 140
#define ST_MODE_OPER 160
#define ST_MODE_OPER_OK 180
#define ST_IDLE 200

void MotorManager::Init(void)
{
    gpiod_line_request_output(myLine->Reset, GPIOD_CONSUMER, 0);
    gpiod_line_request_output(myLine->Red, GPIOD_CONSUMER, 1);
    gpiod_line_request_output(myLine->Green, GPIOD_CONSUMER, 1);
    gpiod_line_request_output(myLine->Blue, GPIOD_CONSUMER, 1);

    //setting up port and bitrate
    for(int i = 0;i < 32;i++){
        _motors[i] = new Motor(&myADC1[i],&myADC2[i],&myStatus[i],&myParam[i]);
        _motors[i]->Registration(&CanBus, i + 1);
    }
    CanBus.SetPortName(m_PortName);
    CanBus.Init();
    //reAddressing();
    xAddrInit = false;
    addr_iter=ST_ADDR_START;
    addr=0;
```

```
}

void MotorManager::reAddressing(void){

    switch (addr_iter)
    {
    case ST_ADDR_START:
        //printf(" - Start Addr \n");
        std::cout << " - Start Addr "<< m_PortName << std::endl;
        addr = 0;
        addr_iter++;
        gpiod_line_set_value(myLine->Reset, 0);
        gpiod_line_set_value(myLine->Red, 1);
        gpiod_line_set_value(myLine->Green, 1);
        gpiod_line_set_value(myLine->Blue, 0);

        xStopSend = true;
        ChangeOperationalMode(STOPPED);
        break;
    case ST_ADDR_NULL:
        //printf(" - Set Addr NULL \n");
        std::cout << " - Set Addr NULL "<< m_PortName << std::endl;
        addr_iter++;
        SetAddress(0);
        break;
    case ST_LINK_RESET:
        std::cout << " - Reset Link "<< m_PortName << std::endl;
        //printf(" - Reset Link \n");
        addr_iter++;
        for(int i = 0;i < 32;i++){
            _motors[i]->stStatus.xLink = 0;
        }
        gpiod_line_set_value(myLine->Reset, 1);
        break;
    case ST_ADDR_SET:
        std::cout << " - Set Addr "<<addr <<" "<< m_PortName << std::endl;
        //printf(" - Set Addr %d \n",addr);
        addr_iter++;
        gpiod_line_set_value(myLine->Blue, addr&0x01);
        SetAddress(addr);
        addr++;
        if (addr>=32){
            addr_iter = ST_ADDR_OK;
        }
        break;
    case ST_ADDR_WAIT:
        addr_iter = ST_ADDR_SET;
        break;
    case ST_ADDR_OK:
        std::cout << " - Set Addr END "<< m_PortName << std::endl;
        //printf(" - Set Addr END \n");
```

```
        addr_iter++;
        gpiod_line_set_value(myLine->Blue, 0);
        gpiod_line_set_value(myLine->Reset, 0);
        break;
    case ST_MODE_OPER:
        std::cout << " - Set OPER MODE "<< m_PortName << std::endl;
        //printf(" - Set OPER MODE \n");
        addr_iter++;
        ChangeOperationalMode(OPERATIONAL);
        break;
    case ST_MODE_OPER_OK:
        std::cout << " - Addressing Ok "<< m_PortName << std::endl;
        //printf(" - Addressing Ok \n");
        addr_iter++;
        xStopSend = false;
        gpiod_line_set_value(myLine->Blue, 1);
        gpiod_line_set_value(myLine->Green, 0);
        gpiod_line_set_value(myLine->Red, 1);
        xAddrInit = true;
        xFindNotInit = false;

        addr_iter=ST_ADDR_START;
        addr=0;
        break;
    case ST_IDLE:
        break;
    default:
        addr_iter++;
        break;
    }

    /*gpiod_line_set_value(myLine->Reset, 0);
    gpiod_line_set_value(myLine->Red, 1);
    gpiod_line_set_value(myLine->Green, 1);
    gpiod_line_set_value(myLine->Blue, 0);

    xStopSend = true;
    ChangeOperationalMode(STOPPED);
    usleep(200000);
    SetAddress(0);
    usleep(200000);
    for(int i = 0;i < 32;i++){
        _motors[i]->stStatus.xLink = 0;
    }
    gpiod_line_set_value(myLine->Reset, 1);
    usleep(200000);
    for(int i = 1; i<32;i++) {
        gpiod_line_set_value(myLine->Blue, i&0x01);
        SetAddress(i);
        usleep(300000);
    }
```

```
    gpiod_line_set_value(myLine->Blue, 0);
    gpiod_line_set_value(myLine->Reset, 0);
    usleep(300000);

    ChangeOperationalMode(OPERATIONAL);
    usleep(200000);
    xStopSend = false;
    gpiod_line_set_value(myLine->Blue, 1);
    gpiod_line_set_value(myLine->Green, 0);
    gpiod_line_set_value(myLine->Red, 1);
    xAddrInit = 1;*/
}

void MotorManager::SetAddress(uint8_t addr){
    //can_frame frame;
    frame.can_id  = 0x600;
    frame.can_dlc = 8;
    frame.data[0] = 0x2F;
    frame.data[1] = 0x00;
    frame.data[2] = 0x30;
    frame.data[3] = 0x01;
    frame.data[4] = addr;
    frame.data[5] = 0x00;
    frame.data[6] = 0x00;
    frame.data[7] = 0x00;

    CanBus.SendMsg(frame);
}

void MotorManager::ChangeOperationalMode(NMT_STATES state){
    //can_frame frame;
    frame.can_id = 0 ;
    frame.can_dlc = 2;
    frame.data[0] = state;
    frame.data[1] = 0;
    frame.data[2] = 0;
    frame.data[3] = 0;
    frame.data[4] = 0;
    frame.data[5] = 0;
    frame.data[6] = 0;
    frame.data[7] = 0;

    CanBus.SendMsg(frame);

}


uint32_t MotorManager::getTick(void) {
    struct timespec ts;
    unsigned theTick = 0U;
    clock_gettime( CLOCK_MONOTONIC, &ts );
```

```
    theTick  = ts.tv_nsec / 1000000;
    theTick += ts.tv_sec * 1000;
    return theTick;
}

void MotorManager::SendParam(void){

    xHMICmd = 0;
    if((last_sendParam + 500)< getTick() ){
        std::unique_lock<std::mutex> qLock(CanBus.SocketWriteMtx);
        for(int i = 0;i < 32;i++){
            float fspeed=myParam[i].rSpeed*1.5;
            //can_frame frame;
            frame.can_id = 0x201+i;
            frame.can_dlc = 8;
            frame.data[0] = (unsigned char)(myParam[i].ucMotorType&0xFF);
            frame.data[1] = 0;
            memcpy(&frame.data[2],&fspeed,4);//&_motors[i]-
>stUstParams.rSpeed,4);
            frame.data[6] = (unsigned char)(myParam[i].ucRampUp&0xFF);
            frame.data[7] = (unsigned char)(myParam[i].ucRampDown&0xFF);
            CanBus.SendQueueMsg(frame);
        }
        last_sendParam = getTick();
    }
}

void MotorManager::SendCMD(void){
    for(int i = 0;i < 32;i++){
        unsigned char motor_byte = i / 8;
        unsigned char motor_bit = i % 8;
        //if ((_motors[i]->stStatus.xLink)&&(_motors[i]->xCmd))
        //if(hr_param.xHMICmd & 0x01)
        {
            if ((myHMICMD[i]&0x02))
                MotorDir[motor_byte] |= 0x01 << motor_bit;
            else
                MotorDir[motor_byte] &= ~(0x01 << motor_bit);
            if ((myHMICMD[i]&0x01))
                MotorCmd[motor_byte] |= 0x01 << motor_bit;
            else
                MotorCmd[motor_byte] &= ~(0x01 << motor_bit);
        }
        /*else{
            myHMICMD[i] = 0;
            if ((myCMD[i]&0x02))
                MotorDir[motor_byte] |= 0x01 << motor_bit;
            else
                MotorDir[motor_byte] &= ~(0x01 << motor_bit);
            if ((myCMD[i]&0x01))
                MotorCmd[motor_byte] |= 0x01 << motor_bit;
```

```
            else
                MotorCmd[motor_byte] &= ~(0x01 << motor_bit);
        } */
    }

    std::unique_lock<std::mutex> qLock(CanBus.SocketWriteMtx);
    if(!xStopSend)
    {

        //can_frame frame;

        frame.can_id = 0x300;
        frame.can_dlc = 8;


        frame.data[0] = MotorCmd[0];
        frame.data[1] = MotorCmd[1];
        frame.data[2] = MotorCmd[2];
        frame.data[3] = MotorCmd[3];
        frame.data[4] = MotorDir[0];
        frame.data[5] = MotorDir[1];
        frame.data[6] = MotorDir[2];
        frame.data[7] = MotorDir[3];
        //printf("%02X\n",MotorCmd[0]);
        CanBus.Msg5ms = frame;
    }else{

        //can_frame frame;

        frame.can_id = 0x000;
        frame.can_dlc = 0;
        CanBus.Msg5ms = frame;
    }
}

void MotorManager::Update(void)
{
    uint32_t dw=getTick();

    if((!CanBus.xFindNotInit)&&((dw-lastFindNotInit)>2000)){
        xFindNotInit = false;
    }

    if(CanBus.xFindNotInit){
        CanBus.xFindNotInit = false;
        lastFindNotInit = dw;
        if(!xFindNotInit){
            printf("Find not init!!!\n");
            xFindNotInit = true;
        }
    }
```

```
    /*if(xFindNotInit){
        gpiod_line_set_value(myLine->Red, 0);
        gpiod_line_set_value(myLine->Blue, 1);
        gpiod_line_set_value(myLine->Green, 1);
    }else{
        gpiod_line_set_value(myLine->Red, 1);
        gpiod_line_set_value(myLine->Blue, 1);
        gpiod_line_set_value(myLine->Green, 0);
    }
    if(xAddrInit){
        SendCMD();
        SendParam();
    }else{
        can_frame frame;
        frame.can_id = 0x000;
        frame.can_dlc = 0;
        CanBus.Msg5ms = frame;
        reAddressing();
    }*/
    if(!xAddrInit){
        can_frame frame;
        frame.can_id = 0x000;
        frame.can_dlc = 0;
        CanBus.Msg5ms = frame;
        reAddressing();
    }else{

        if(xFindNotInit){
            gpiod_line_set_value(myLine->Red, 0);
            gpiod_line_set_value(myLine->Blue, 1);
            gpiod_line_set_value(myLine->Green, 1);
        }else{
            gpiod_line_set_value(myLine->Red, 1);
            gpiod_line_set_value(myLine->Blue, 1);
            gpiod_line_set_value(myLine->Green, 0);
        }
            SendCMD();
            SendParam();
    }

    for(int i = 0;i < 32;i++){
        _motors[i]->Update();
    }

}

std::string MotorManager::GetJSON(void)
{
    stringStream.str("");

    return stringStream.str();
```

```
}
```

12. Структуры и классы для работы API мотора

```
#ifndef MOTORMANAGER
#define MOTORMANAGER

#include "CanManager.hpp"
#include "Motor.hpp"
#include <gpiod.h>
#include "modbus.h"


#define GPIOD_CONSUMER "COMITAS_CAN_GATE"

struct NodeOutputLine
{
    struct gpiod_line *Reset;
    struct gpiod_line *Red;
    struct gpiod_line *Green;
    struct gpiod_line *Blue;
} ;

enum NMT_STATES
{
    INITIALISING = 129,
    STOPPED = 2,
    OPERATIONAL = 1,
    SLEEP = 80,
    STANDBY = 96,
    PRE_OPERATIONAL = 128
};

class MotorManager
{

private:
    uint8_t MotorCmd[8] = {0}; //ID range from 201 to 204
    uint8_t MotorDir[8] = {0}; //ID range from 201 to 204
    bool xStopSend;
    bool xStopRead;
    std::string m_PortName;
    NodeOutputLine *myLine;
    ADC1_t *myADC1;
    ADC2_t *myADC2;
    u_int16_t *myStatus;
```

```
    u_int16_t *myCMD;
    u_int16_t *myHMICMD;
    CDatasheet_t *myParam;
    u_int8_t *mySelect;
public:
    unsigned short addr_iter;
    unsigned short addr;
    uint32_t last_sendParam;
    CDatasheet_t stUstParams;
    uint16_t xHMICmd;
    uint16_t xHMIControl;
    uint16_t xHMIMotorID;
    uint32_t lastFindNotInit;
    uint32_t lastSendParam;
    uint32_t getTick(void);
    std::ostringstream stringStream;
    bool xFindNotInit;
    CanManager CanBus;
    Motor *_motors[32];
    void SendParam(void);
    void SendCMD(void);
    can_frame frame;
    MotorManager(const std::string PortName,NodeOutputLine* Line,
                 ADC1_t *ADC1,
                 ADC2_t *ADC2,
                 u_int16_t *Status,
                 u_int16_t *CMD,
                 u_int16_t *HMICMD,
                 CDatasheet_t *Param,
                 u_int8_t *Select){
        xStopSend = false;
        xStopRead = false;
        last_sendParam = 0;
        xHMICmd = 0;
        xHMIControl = 0;
        xHMIMotorID = 0;
        lastFindNotInit = 0;
        lastSendParam = 0;
        m_PortName = PortName;
        myLine = Line;
        xAddrInit = 0;
        myADC1 = ADC1;
        myADC2 = ADC2;
        myStatus = Status;
        myCMD = CMD;
        myHMICMD = HMICMD;
        myParam = Param;
        mySelect = Select;
    }
    ~MotorManager(){
        for(int i = 0;i < 32;i++){
```

```
            delete _motors[i];
        }
    }
    void Init(void);

    void Add2Buffer(int ID, uint16_t CmdSet)
    {
        if(ID>0&&ID<32){
            MotorCmd[0] = CmdSet;
            MotorCmd[1] = CmdSet;
            MotorCmd[2] = CmdSet;
            MotorCmd[3] = CmdSet;
            MotorCmd[4] = CmdSet;
            MotorCmd[5] = CmdSet;
            MotorCmd[6] = CmdSet;
            MotorCmd[7] = CmdSet;
        }
    };

    void Update(void);
    std::string GetJSON(void);
    void reAddressing(void);
    bool xAddrInit;
    void SetAddress(uint8_t addr);
    void ChangeOperationalMode(NMT_STATES state);
};


#endif
```

13. API по работе с шиной CAN

```
#include "CanManager.hpp"
#include <fcntl.h>

void CanMsgHandler::Registration(CanManager *CanBus,int id)
{
    CanBus->HandlerMap.insert(std::make_pair(id,this));
}

void CanMsgHandler::HandleNewMsg(can_frame msg)
{
    std::cout << msg.can_id << std::endl;
}
```

```
void CanManager::Init()
{
    //setting up port and bitrate
    int rez = 0;
    rez = system(("sudo ip link set " + m_PortName + " type can bitrate
1000000").c_str());
    rez = system(("sudo ifconfig " + m_PortName + " up").c_str());
    rez = system(("sudo ip -details link show " + m_PortName).c_str());

    //setup socketcan
    struct sockaddr_can addr;
    struct ifreq ifr;
    SocketFD = socket(PF_CAN, SOCK_RAW|SOCK_NONBLOCK, CAN_RAW);
    strcpy(ifr.ifr_name, m_PortName.c_str());
    ioctl(SocketFD, SIOCGIFINDEX, &ifr);

    addr.can_family = AF_CAN;
    addr.can_ifindex = ifr.ifr_ifindex;
    bind(SocketFD, (struct sockaddr *)&addr, sizeof(addr));

    //Start
    StartThread();
}

void CanManager::SendMsg(can_frame msg)
{
    int nbytes = write(SocketFD, &msg, sizeof(struct can_frame));
    if(nbytes == -1)
    {
        //std::cerr << "send error" << std::endl;
        //exit(1);
    }
}

void CanManager::SendQueueMsg(can_frame msg)
{
    if(q.size() < 100)
        q.push(msg);
}

can_frame CanManager::GetQueueMsg(int* rez)
{
    can_frame msg;
    *rez = 0;
    if (!q.empty()){
        msg = q.front();
        q.pop();
        *rez = 1;
    }
    return msg;
```

```
}

void CanManager::StartThread(void)
{
    m_IsRunning = true;
    pthread_create(&RxThreadID,nullptr,&CanManager::RxThread, this);
    pthread_create(&TxThreadID,nullptr,&CanManager::TxThread, this);
}

void CanManager::EndThread(void)
{
    m_IsRunning = false;
}

void* CanManager::RxThread(void *argv)
{
    CanManager* ptr = ((CanManager*)argv);
    while(ptr->IsRunning())
    {
        /*
        //TODO: Use `select()` instead, non-block reading required here.
        //while(ptr -> TxRequestFlag){;}
        //Read Frame
        struct can_frame frame;
        //ptr -> SocketMtx.lock();
        int nbytes = read(ptr -> SocketFD, &frame, sizeof(struct can_frame));
        //ptr -> SocketMtx.unlock();

        //Distribute Msgs
        if (nbytes < 0) {
            std::cout << "read error" << std::endl;
            exit(1);
        }else{
            //Written like shit; Gonna Fix this
            if((frame.can_id&0x1F)==0x00){
                ptr->xFindNotInit = true;
            }else{
                auto iter = ptr->HandlerMap.find(frame.can_id&0x1F);
                if(iter==ptr->HandlerMap.end()) //Handler not found
                {
                    std::cerr << "Handler for Can ID " << frame.can_id  << " do
not exist" << std::endl;
                    exit(1);
                }

                ((CanMsgHandler *)(iter->second))->HandleNewMsg(frame);
            }
        }*/

        //TODO: Use `select()` instead, non-block reading required here.
        //while(ptr -> TxRequestFlag){;}
```

```
        //Read Frame
        struct can_frame frame;
        //ptr -> SocketMtx.lock();
        while(read(ptr -> SocketFD, &frame, sizeof(struct can_frame))>0)
        {
            if((frame.can_id&0x1F)==0x00){
                ptr->xFindNotInit = true;
            }else{
                auto iter = ptr->HandlerMap.find(frame.can_id&0x1F);
                if(iter==ptr->HandlerMap.end()) //Handler not found
                {
                    std::cerr << "Handler for Can ID " << frame.can_id  << " do
not exist" << std::endl;
                    exit(1);
                }

                ((CanMsgHandler *)(iter->second))->HandleNewMsg(frame);
            }
        }

        //control loop frequency
        std::this_thread::sleep_for(std::chrono::milliseconds(5));
    }

}

void* CanManager::TxThread(void *argv)
{
    struct can_frame frame;
    struct can_frame frame5ms;
    int NeedSendQueue = 0;
    CanManager* ptr = ((CanManager*)argv);
    while(ptr->IsRunning())
    {

        //TODO: Use `select()` instead, non-block reading required here.
        //while(((CanManager*)argv) -> TxRequestFlag){;}

        std::unique_lock<std::mutex> qLock(ptr -> SocketWriteMtx);
        frame = ptr -> GetQueueMsg(&NeedSendQueue);
        if(NeedSendQueue > 0){
            int nbytes = write(ptr -> SocketFD, &frame, sizeof(struct
can_frame));
            if(nbytes == -1)
            {
                //std::cerr << "send error" << std::endl;
                //exit(1);
            }
            NeedSendQueue = 0;
        }
        frame5ms = ptr -> Msg5ms;
```

```
        if(frame5ms.can_id > 0){
            int nbytes = write(ptr -> SocketFD, &frame5ms, sizeof(struct
can_frame));
            if(nbytes == -1)
            {
                //std::cerr << "send error" << std::endl;
                //exit(1);
            }
        }

        qLock.unlock();
        std::this_thread::sleep_for(std::chrono::milliseconds(5));
    }

}
```

14. Структуры и классы по работе с CAN

```
#ifndef CAN_MANAGER_HPP
#define CAN_MANAGER_HPP

#include <net/if.h>
#include <sys/ioctl.h>
#include <linux/can.h>
#include <linux/can/raw.h>
#include <iostream>
#include <signal.h>
#include <cstring>
#include <string>
#include <chrono>
#include <unistd.h>
#include <thread>
#include <mutex>
#include <pthread.h>
#include <queue>
#include <map>

class CanMsgHandler;

class CanManager
{
private:
    std::string m_PortName;
    bool m_IsRunning;

public:
    std::queue <can_frame> q;
```

```
    can_frame Msg5ms;
    CanManager(){
        xFindNotInit = false;
        m_PortName = "";
        m_IsRunning = false;
        SocketFD = 0;
        TxRequestFlag = false;
    };
    ~CanManager(){};
    int SocketFD;
    std::mutex SocketMtx;
    std::mutex SocketWriteMtx;
    bool TxRequestFlag;
    pthread_t RxThreadID{};
    pthread_t TxThreadID{};
    std::map<int,CanMsgHandler*> HandlerMap;

    void SetPortName(const std::string PortName){m_PortName = PortName;};
    void Init(void);
    bool IsRunning(){return m_IsRunning;}
    bool xFindNotInit;
    //Transmit
    void SendMsg(can_frame msg);
    void SendQueueMsg(can_frame msg);
    can_frame GetQueueMsg(int* rez);

    //Receive, with a single thread.
    void StartThread(void);
    void EndThread(void);
    static void* RxThread(void *argv);
    static void* TxThread(void *argv);

};

class CanMsgHandler
{
    private:
        int m_id;
    public:
        CanMsgHandler(){
            m_id = 0;
        };
        void Registration(CanManager *CanBus,int id);
        virtual void HandleNewMsg(can_frame msg) = 0 ;
};

#endif
```

1. Инициализация, конфигурация и запуск сервера

```
import os
import sys
import time
import socket
import logging
import argparse
import threading
import datetime
import pathlib
import time
import json
```

```
from . import APP_NAME, \
              APP_VERSION, \
              APP_DESCRIPTION, \
              CONFIG_DIR, \
              CACHE_DIR, \
              LOG_DIR


from threading import Thread, Lock
from pyModbusTCP.client import ModbusClient
from signal import signal, SIGINT
from flask import Flask, request
from flask import render_template, jsonify
from subprocess import check_output
#from gevent.pywsgi import WSGIServer

#HOST, PORT = "localhost", 50007
SERVER_HOST = "localhost"
SERVER_PORT = 502

PathToMain = str(pathlib.Path(__file__).parent.resolve())

#ctx = Context('local:')
#dev = ctx.find_device('2198000.adc')
#mt = None
#
#atr = [None]*20
#
regs_lock = Lock()
regs = []
stat_name = []
d1={}
d2={}

flapp = Flask("ComitasApp", template_folder="/usr/lib/python3.8/site-
packages/canopen_monitor/templates",
                            static_folder="/usr/lib/python3.8/site-
packages/canopen_monitor/static")
select_log = 0
select_dev = 0
ReAddr = 0
LinkCService = 0
send_cmd = 0
send_param = 0

@flapp.route('/data_json')
def data_json():
    global select_dev
    global selectADC
    global d1,LinkCService
```

```
    res_data = d1
    res_data["Link"] = LinkCService
    return jsonify(res_data)

@flapp.route('/DriveReset', methods=['POST'])
def drive_reset():
    global d2,send_cmd
    log_data = []
    d2["HMIControl"] = 16
    send_cmd = 4
    return jsonify(log_data)

@flapp.route('/ReadCfg', methods=['POST'])
def ReadCfg():
    global d2,send_cmd
    log_data = []
    d2["HMIControl"] = 8
    send_cmd = 4
    return jsonify(log_data)

@flapp.route('/WriteCfg', methods=['POST'])
def WriteCfg():
    global d2,send_cmd
    log_data = []
    d2["HMIControl"] = 4
    send_cmd = 4
    return jsonify(log_data)

@flapp.route('/SetHMIControl', methods=['POST'])
def SetHMIControl():
    global d2,send_cmd
    log_data = []
    input_json = request.get_json(force=True)
    if bool(input_json['hmi_ctrl']) == False:
        d2["HMIControl"] = 1
    else:
        d2["HMIControl"] = 2
    send_cmd = 4
    return jsonify(log_data)

@flapp.route('/SetIP', methods=['POST'])
def SetIP():
    global d1,d2,send_cmd
    log_data = []
    input_json = request.get_json(force=True)
    d2["IP"] = d1["IP"]
    d2["HMIControl"] = 32
    send_cmd = 4
    return jsonify(log_data)

@flapp.route('/DriveSave', methods=['POST'])
```

```
def drive_save():
    global d2,send_cmd
    log_data = []
    input_json = request.get_json(force=True)
    d2["MotorParam1"] = input_json["MotorParam1"]
    d2["MotorParam2"] = input_json["MotorParam2"]
    #d2["Select"] = input_json["Select"]
    send_cmd = 2
    return jsonify(log_data)

@flapp.route('/DriveOn', methods=['POST'])
def drive_ctrl():
    global d2
    global send_cmd
    log_data = []
    input_json = request.get_json(force=True)
    d2["HMICmd"] = d1["HMICmd"]

    d2["HMICmd"][int(input_json['dev_id'])] = int(input_json['drive_cmd'])
    send_cmd = 1
    return jsonify(log_data)

@flapp.route('/')
@flapp.route('/home')
def home_page():
    dummy_data = data_json()
    return render_template(
        'home.html',
        matches=dummy_data.json,
    )

def init_dirs():
    os.makedirs(CONFIG_DIR, exist_ok=True)
    os.makedirs(CACHE_DIR, exist_ok=True)
    os.makedirs(LOG_DIR, exist_ok=True)


def init_logs(log_level):
    logging.basicConfig(filename=f'{LOG_DIR}/latest.log', level=log_level)
    logging.info(f'{APP_NAME} v{APP_VERSION}', extra={'time': time.ctime()})

sockfd = None

def TCPGet(id,stop):
    print("  Exiting TCPGet.")

citer3 = 0
# init a thread lock
def RestartCService():
    print("restart C service")
    #pid = check_output(["pidof","Test"])
```

```
    os.system(f"pkill Test")
    #os.kill(pid, signal.SIGSTOP)
    time.sleep(0.050)
    os.system(f"Test")

#def TCPSend(id,stop):
def polling_thread():
    """Modbus polling thread."""
    global regs, regs_lock,d1,LinkCService,send_cmd,citer3
    c = ModbusClient(host=SERVER_HOST, port=SERVER_PORT,timeout=1.0,
auto_open=True)

    citer = 0
    citer2 = 0
    while True:
        if send_cmd == 1:
            reg_list = c.write_multiple_registers(704, d2["HMICmd"])
            send_cmd = 0
        if send_cmd == 2:
            reg_list = c.write_multiple_registers(512, d2["MotorParam1"])
            reg_list = c.write_multiple_registers(608, d2["MotorParam2"])
            #reg_list = c.write_multiple_registers(768, d2["Select"])
            send_cmd = 0
        if send_cmd ==3:
            reg_list = c.write_multiple_registers(800, d2["IP"])
            reg_list = c.write_single_register(804, d2["HMIControl"])
            send_cmd = 0
        if send_cmd == 4:
            reg_list = c.write_single_register(804, d2["HMIControl"])
            send_cmd = 0

        if citer == 0:
            # do modbus reading on socket
            reg_list = c.read_holding_registers(0, 64)
            if reg_list:
                LinkCService = True
                with regs_lock:
                    regs = list(reg_list)
                    d1["MotorStatus"]=regs
            else:
                citer3 = citer3 + 1
                LinkCService = False
        if citer == 1:
            # do modbus reading on socket
            reg_list = c.read_holding_registers(128, 96)
            if reg_list:
                with regs_lock:
                    regs = list(reg_list)
                    d1["N1ADC1"]=regs
        if citer == 2:
            # do modbus reading on socket
```

```
            reg_list = c.read_holding_registers(224, 96)
            if reg_list:
                with regs_lock:
                    regs = list(reg_list)
                    d1["N2ADC1"]=regs
        if citer == 3:
            # do modbus reading on socket
            reg_list = c.read_holding_registers(320, 96)
            if reg_list:
                with regs_lock:
                    regs = list(reg_list)
                    d1["N1ADC2"]=regs
        if citer == 4:
            # do modbus reading on socket
            reg_list = c.read_holding_registers(416, 96)
            if reg_list:
                with regs_lock:
                    regs = list(reg_list)
                    d1["N2ADC2"]=regs

        if citer == 5:
            # do modbus reading on socket
            reg_list = c.read_holding_registers(64, 64)
            if reg_list:
                with regs_lock:
                    regs = list(reg_list)
                    d1["MotorCMD"]=regs
        if citer == 6:
            # do modbus reading on socket
            reg_list = c.read_holding_registers(512, 96)
            if reg_list:
                with regs_lock:
                    regs = list(reg_list)
                    d1["MotorParam1"]=regs
        if citer == 7:
            # do modbus reading on socket
            reg_list = c.read_holding_registers(608, 96)
            if reg_list:
                with regs_lock:
                    regs = list(reg_list)
                    d1["MotorParam2"]=regs
        if citer == 8:
            # do modbus reading on socket
            reg_list = c.read_holding_registers(704, 64)
            if reg_list:
                with regs_lock:
                    regs = list(reg_list)
                    d1["HMICmd"]=regs
        if citer == 9:
            # do modbus reading on socket
            reg_list = c.read_holding_registers(768, 32)
```

```
            if reg_list:
                with regs_lock:
                    regs = list(reg_list)
                    d1["Select"]=regs
        if citer == 10:
            # do modbus reading on socket
            reg_list = c.read_holding_registers(800, 4)
            if reg_list:
                with regs_lock:
                    regs = list(reg_list)
                    d1["IP"]=regs

        citer = citer + 1
        if citer > 10 :
            citer = 0
        time.sleep(0.050)

    print("  Exiting TCPSend.")


def main():
    global regs, regs_lock,d1
    global mt
    global ReAddr
    global selectADC
    global LinkCService
    global sockfd,citer3
    try:
        #init_dirs()
        #init_logs('CRITICAL')
        #meta = Meta(CONFIG_DIR, CACHE_DIR)
        #features = meta.load_features()
        #eds_configs = load_eds_files(PathToMain, False)
        #mt = MessageTable(CANOpenParser(eds_configs))

        #signal(SIGINT, lambda h, f: (stopALL(), exit(0)))
        threading.Thread(target=lambda: flapp.run(host='0.0.0.0', port=5000,
debug=True, use_reloader=False)).start()
        #http_server = WSGIServer(('', 5000), app)
        #http_server.serve_forever()

        stop_threads = False
        # start polling thread
        tp = Thread(target=polling_thread)
        # set daemon: polling thread will exit if main thread exit
        tp.daemon = True
        tp.start()

        time.sleep(0.1)
        while True:
```

```
            #if citer3 > 10 :
            #    citer3 = 0
            #    RestartCService()
            #with regs_lock:
            #    print(dict(d1))
            #if sockfd is None:
            #    stop_threads = True
            #    try:
            #        sockfd = socket.socket()
            #        sockfd.setblocking(False)
            #    except OSError as msg:
            #        sockfd = None
#
            #    if sockfd is not None:
            #        try:
            #            sockfd.settimeout(1.0)
            #            sockfd.connect((HOST, PORT))
            #        except OSError as msg:
            #            LinkCService = False
            #            print("connect not open")
            #            sockfd.close()
            #            sockfd = None
#
            #    if sockfd is not None:
            #        stop_threads = False
            #        print("  Start Thread.")
            #        threading.Thread(target=TCPSend, args=(1,lambda:
stop_threads)).start()
            #        threading.Thread(target=TCPGet, args=(2,lambda:
stop_threads)).start()
#
            #if sockfd is None:
            #    LinkCService = False
            #    print("connect not open")

            time.sleep(0.500)

    except KeyboardInterrupt:
        stop_threads = True
        print('Goodbye!')

if __name__ == '__main__':
    main()
```

2. Описание веб интерфейса

```
import os
import sys
import time
import socket
import logging
import argparse
import threading
import datetime
import pathlib
import time
import json

from . import APP_NAME, \
              APP_VERSION, \
              APP_DESCRIPTION, \
              CONFIG_DIR, \
              CACHE_DIR, \
              LOG_DIR


from threading import Thread, Lock
from pyModbusTCP.client import ModbusClient
from signal import signal, SIGINT
from flask import Flask, request
from flask import render_template, jsonify
from subprocess import check_output
#from gevent.pywsgi import WSGIServer

#HOST, PORT = "localhost", 50007
SERVER_HOST = "localhost"
SERVER_PORT = 502

PathToMain = str(pathlib.Path(__file__).parent.resolve())

#ctx = Context('local:')
#dev = ctx.find_device('2198000.adc')
#mt = None
#
#atr = [None]*20
#
regs_lock = Lock()
regs = []
stat_name = []
d1={}
d2={}

flapp = Flask("ComitasApp", template_folder="/usr/lib/python3.8/site-
packages/canopen_monitor/templates",
                            static_folder="/usr/lib/python3.8/site-
packages/canopen_monitor/static")
select_log = 0
```

```
select_dev = 0
ReAddr = 0
LinkCService = 0
send_cmd = 0
send_param = 0

@flapp.route('/data_json')
def data_json():
    global select_dev
    global selectADC
    global d1,LinkCService
    res_data = d1
    res_data["Link"] = LinkCService
    return jsonify(res_data)

@flapp.route('/DriveReset', methods=['POST'])
def drive_reset():
    global d2,send_cmd
    log_data = []
    d2["HMIControl"] = 16
    send_cmd = 4
    return jsonify(log_data)

@flapp.route('/ReadCfg', methods=['POST'])
def ReadCfg():
    global d2,send_cmd
    log_data = []
    d2["HMIControl"] = 8
    send_cmd = 4
    return jsonify(log_data)

@flapp.route('/WriteCfg', methods=['POST'])
def WriteCfg():
    global d2,send_cmd
    log_data = []
    d2["HMIControl"] = 4
    send_cmd = 4
    return jsonify(log_data)

@flapp.route('/SetHMIControl', methods=['POST'])
def SetHMIControl():
    global d2,send_cmd
    log_data = []
    input_json = request.get_json(force=True)
    if bool(input_json['hmi_ctrl']) == False:
        d2["HMIControl"] = 1
    else:
        d2["HMIControl"] = 2
    send_cmd = 4
    return jsonify(log_data)
```

```python
@flapp.route('/SetIP', methods=['POST'])
def SetIP():
    global d1,d2,send_cmd
    log_data = []
    input_json = request.get_json(force=True)
    d2["IP"] = d1["IP"]
    d2["HMIControl"] = 32
    send_cmd = 4
    return jsonify(log_data)

@flapp.route('/DriveSave', methods=['POST'])
def drive_save():
    global d2,send_cmd
    log_data = []
    input_json = request.get_json(force=True)
    d2["MotorParam1"] = input_json["MotorParam1"]
    d2["MotorParam2"] = input_json["MotorParam2"]
    #d2["Select"] = input_json["Select"]
    send_cmd = 2
    return jsonify(log_data)

@flapp.route('/DriveOn', methods=['POST'])
def drive_ctrl():
    global d2
    global send_cmd
    log_data = []
    input_json = request.get_json(force=True)
    d2["HMICmd"] = d1["HMICmd"]

    d2["HMICmd"][int(input_json['dev_id'])] = int(input_json['drive_cmd'])
    send_cmd = 1
    return jsonify(log_data)

@flapp.route('/')
@flapp.route('/home')
def home_page():
    dummy_data = data_json()
    return render_template(
        'home.html',
        matches=dummy_data.json,
    )

def init_dirs():
    os.makedirs(CONFIG_DIR, exist_ok=True)
    os.makedirs(CACHE_DIR, exist_ok=True)
    os.makedirs(LOG_DIR, exist_ok=True)


def init_logs(log_level):
    logging.basicConfig(filename=f'{LOG_DIR}/latest.log', level=log_level)
    logging.info(f'{APP_NAME} v{APP_VERSION}', extra={'time': time.ctime()})
```

```
sockfd = None

def TCPGet(id,stop):
    print("  Exiting TCPGet.")

citer3 = 0
# init a thread lock
def RestartCService():
    print("restart C service")
    #pid = check_output(["pidof","Test"])
    os.system(f"pkill Test")
    #os.kill(pid, signal.SIGSTOP)
    time.sleep(0.050)
    os.system(f"Test")

#def TCPSend(id,stop):
def polling_thread():
    """Modbus polling thread."""
    global regs, regs_lock,d1,LinkCService,send_cmd,citer3
    c = ModbusClient(host=SERVER_HOST, port=SERVER_PORT,timeout=1.0,
auto_open=True)

    citer = 0
    citer2 = 0
    while True:
        if send_cmd == 1:
            reg_list = c.write_multiple_registers(704, d2["HMICmd"])
            send_cmd = 0
        if send_cmd == 2:
            reg_list = c.write_multiple_registers(512, d2["MotorParam1"])
            reg_list = c.write_multiple_registers(608, d2["MotorParam2"])
            #reg_list = c.write_multiple_registers(768, d2["Select"])
            send_cmd = 0
        if send_cmd ==3:
            reg_list = c.write_multiple_registers(800, d2["IP"])
            reg_list = c.write_single_register(804, d2["HMIControl"])
            send_cmd = 0
        if send_cmd == 4:
            reg_list = c.write_single_register(804, d2["HMIControl"])
            send_cmd = 0

        if citer == 0:
            # do modbus reading on socket
            reg_list = c.read_holding_registers(0, 64)
            if reg_list:
                LinkCService = True
                with regs_lock:
                    regs = list(reg_list)
                    d1["MotorStatus"]=regs
            else:
```

```
            citer3 = citer3 + 1
            LinkCService = False
    if citer == 1:
        # do modbus reading on socket
        reg_list = c.read_holding_registers(128, 96)
        if reg_list:
            with regs_lock:
                regs = list(reg_list)
                d1["N1ADC1"]=regs
    if citer == 2:
        # do modbus reading on socket
        reg_list = c.read_holding_registers(224, 96)
        if reg_list:
            with regs_lock:
                regs = list(reg_list)
                d1["N2ADC1"]=regs
    if citer == 3:
        # do modbus reading on socket
        reg_list = c.read_holding_registers(320, 96)
        if reg_list:
            with regs_lock:
                regs = list(reg_list)
                d1["N1ADC2"]=regs
    if citer == 4:
        # do modbus reading on socket
        reg_list = c.read_holding_registers(416, 96)
        if reg_list:
            with regs_lock:
                regs = list(reg_list)
                d1["N2ADC2"]=regs

    if citer == 5:
        # do modbus reading on socket
        reg_list = c.read_holding_registers(64, 64)
        if reg_list:
            with regs_lock:
                regs = list(reg_list)
                d1["MotorCMD"]=regs
    if citer == 6:
        # do modbus reading on socket
        reg_list = c.read_holding_registers(512, 96)
        if reg_list:
            with regs_lock:
                regs = list(reg_list)
                d1["MotorParam1"]=regs
    if citer == 7:
        # do modbus reading on socket
        reg_list = c.read_holding_registers(608, 96)
        if reg_list:
            with regs_lock:
                regs = list(reg_list)
```

```
                    d1["MotorParam2"]=regs
            if citer == 8:
                # do modbus reading on socket
                reg_list = c.read_holding_registers(704, 64)
                if reg_list:
                    with regs_lock:
                        regs = list(reg_list)
                        d1["HMICmd"]=regs
            if citer == 9:
                # do modbus reading on socket
                reg_list = c.read_holding_registers(768, 32)
                if reg_list:
                    with regs_lock:
                        regs = list(reg_list)
                        d1["Select"]=regs
            if citer == 10:
                # do modbus reading on socket
                reg_list = c.read_holding_registers(800, 4)
                if reg_list:
                    with regs_lock:
                        regs = list(reg_list)
                        d1["IP"]=regs

            citer = citer + 1
            if citer > 10 :
                citer = 0
            time.sleep(0.050)

        print("  Exiting TCPSend.")


    def main():
        global regs, regs_lock,d1
        global mt
        global ReAddr
        global selectADC
        global LinkCService
        global sockfd,citer3
        try:
            #init_dirs()
            #init_logs('CRITICAL')
            #meta = Meta(CONFIG_DIR, CACHE_DIR)
            #features = meta.load_features()
            #eds_configs = load_eds_files(PathToMain, False)
            #mt = MessageTable(CANOpenParser(eds_configs))

            #signal(SIGINT, lambda h, f: (stopALL(), exit(0)))
            threading.Thread(target=lambda: flapp.run(host='0.0.0.0', port=5000,
    debug=True, use_reloader=False)).start()
            #http_server = WSGIServer(('', 5000), app)
            #http_server.serve_forever()
```

```
        stop_threads = False
        # start polling thread
        tp = Thread(target=polling_thread)
        # set daemon: polling thread will exit if main thread exit
        tp.daemon = True
        tp.start()

        time.sleep(0.1)
        while True:

            #if citer3 > 10 :
            #    citer3 = 0
            #    RestartCService()
            #with regs_lock:
            #    print(dict(d1))
            #if sockfd is None:
            #    stop_threads = True
            #    try:
            #        sockfd = socket.socket()
            #        sockfd.setblocking(False)
            #    except OSError as msg:
            #        sockfd = None
#
            #    if sockfd is not None:
            #        try:
            #            sockfd.settimeout(1.0)
            #            sockfd.connect((HOST, PORT))
            #        except OSError as msg:
            #            LinkCService = False
            #            print("connect not open")
            #            sockfd.close()
            #            sockfd = None
#
            #    if sockfd is not None:
            #        stop_threads = False
            #        print("  Start Thread.")
            #        threading.Thread(target=TCPSend, args=(1,lambda:
stop_threads)).start()
            #        threading.Thread(target=TCPGet, args=(2,lambda:
stop_threads)).start()
#
            #if sockfd is None:
            #    LinkCService = False
            #    print("connect not open")

            time.sleep(0.500)

    except KeyboardInterrupt:
        stop_threads = True
        print('Goodbye!')
```

```
if __name__ == '__main__':
    main()
```

3. Описание блока логгирования

```
{% extends 'index.html' %}
{% block title %}
    Log Page
{% endblock %}

{% block content %}
<div class="content">
<table class="table table-dark table-striped table-hover" id="sel_table">
  <thead>
    <tr>
      <th scope="col" width="100px">Can</th>
      <th scope="col" width="100px">NodeID</th>
      <th scope="col" width="100px">CobID</th>
      <th scope="col" width="100px">Type</th>
      <th scope="col" width="100px">Age</th>
      <th scope="col">Message</th>
    </tr>
  </thead>
  <tbody id="table_body">
    {% for log in logs %}
      <tr>
        <td id="id_{{ log.id }}" style="display:none;">{{ log.id }}</td>
        <td id="canid_{{ log.id }}">{{ log.canid }}</td>
        <td id="nodeid_{{ log.id }}">{{ log.nodeid }}</td>
        <td id="cob_{{ log.id }}">{{ log.cob }}</td>
        <td id="type_{{ log.id }}">{{ log.type }}</td>
        <td id="age_{{ log.id }}">{{ log.age }}</td>
        <td id="message_{{ log.id }}">{{ log.message }}</td>
      </tr>
    {% endfor %}
  </tbody>
</table>
</div>

<div class="float_parent">
  <div class="float">
    <table class="table table-dark table-striped"  id="tip_table">
      <thead>
        <tr>
          <th scope="col">Name</th>
          <th scope="col">Val</th>
        </tr>
```

```
      </thead>
      <tbody id="table_body">
        {% for tip in tips %}
          <tr>
            <td id="tip_name_{{ tip.id }}">{{ tip.name }}</td>
            <td id="tip_val_{{ tip.id }}">{{ tip.val }}</td>
          </tr>
        {% endfor %}
      </tbody>
    </table>
  </div>
</div>

<script>
  var log_table =
document.getElementById('sel_table').getElementsByTagName('tbody')[0];
  var tip_table =
document.getElementById('tip_table').getElementsByTagName('tbody')[0];
  var selected = log_table.getElementsByClassName('selected');
  var sel_id = 0;
  log_table.onclick = highlight;

  function highlight(e) {
        if (selected[0]) selected[0].className = '';
        if (e.target.parentNode.id == "sel_row"){
          e.target.parentNode.className = 'selected';
          var coolVar = e.target.id;
          var partsArray = coolVar.split('_');
          sel_id = partsArray[1];
          var j = {
            "sel": sel_id
          };
          tip_table.innerHTML = '';
          fetch('/sel_log',{method: 'POST',body: JSON.stringify(j)})
        }
    }

    function fnselect(){
        var element = document.querySelectorAll('.selected');
        if(element[0]!== undefined){ //it must be selected
         alert(element[0].children[0].firstChild.data);
        }
    }

  function addRowLog(id) {
        var row = log_table.insertRow(0);
        row.id = "sel_row";
        row.innerHTML =
        '<td id="id_'+id+'" style="display:none;"></td>' +
        '<td id="canid_'+id+'"></td>'+
        '<td id="nodeid_'+id+'"></td>'+
```

```
      '<td id="cob_'+id+'"></td>'+
      '<td id="type_'+id+'"></td>'+
      '<td id="age_'+id+'"></td>'+
      '<td id="message_'+id+'"></td>';//+
}
function addRowTip(id) {
      var row = tip_table.insertRow(0);
      row.innerHTML =
      '<td id="tip_id_'+id+'" style="display:none;"></td>' +
      '<td id="tip_name_'+id+'"></td>'+
      '<td id="tip_val_'+id+'"></td>';
}

setInterval(function() {
    fetch('/log_json').then(
      response => response.json()
    ).then(
      data =>updateData(data)
    )
  }, 1000
);

function updateData(data) {
    data['list'].forEach(log =>
          updateElement(log)
        )
    data['param'].forEach(tip =>
          updateParam(tip)
        )
}

function updateElement(log) {
    Object.entries(log).forEach(([k,v]) => {
        element = document.getElementById(k + "_" + log.id);
        if ((element!== null)){
          // Get Previous value first
          previousValue = element.innerHTML;
          // If Previous value is not equal to the new value, change it!
          if (previousValue !== v.toString()) {
              blinkUpdate(element, v.toString());
          }
        }else{
          addRowLog(log.id);
        }
    })
}

function updateParam(tip) {
    Object.entries(tip).forEach(([k,v]) => {
        element = document.getElementById(k + "_" + tip.tip_id);
        if ((element!== null)){
```

```
            // Get Previous value first
            previousValue = element.innerHTML;
            // If Previous value is not equal to the new value, change it!
            if (previousValue !== v.toString()) {
                blinkUpdate(element, v.toString());
            }
          }else{
            addRowTip(tip.tip_id);
          }
      })
  }

  function blinkUpdate(element, newValue, newColor='#00ff00') {
      previousColor = element.style.color;
      element.innerHTML = newValue;
      element.style.color = newColor;
      // Set back to original color after timeout ms
      setTimeout(() => {
          element.style.color = previousColor
      }, 2000)
  }
</script>
{% endblock %}
```

4. Описание блоков веб интерфейса

```
{% extends 'index.html' %}
{% block title %}
    Home Page
{% endblock %}
{% block content %}

<div style="text-align: center;color: red;">
  <p1 id="Error_text" hidden></p1>
</div>
<div class="container-lg main">
<div class="content" style="display: flex;">
  <div style="display : block; padding: 0px 5px 5px 5px; width: 35%;">
    <table class="table table-dark table-striped table-hover" id="adc_table">
      <thead>
        <tr>
          <th scope="col">Name</th>
          <th scope="col">Val</th>
        </tr>
      </thead>
      <tbody id="table_body">
          <tr>
```

```
  <td id="Umain">Voltage main</td>
  <td id="val_Vmain"></td>
</tr>
<tr>
  <td id="5V">Voltage 5V</td>
  <td id="val_5V"></td>
</tr>
<tr>
  <td id="Uch1">Voltage ch1</td>
  <td id="val_Vch1"></td>
</tr>
<tr>
  <td id="Ich1">Current ch1</td>
  <td id="val_Ich1"></td>
</tr>
<tr>
  <td id="Uch2">Voltage ch2</td>
  <td id="val_Vch2"></td>
</tr>
<tr>
  <td id="Ich2">Current ch2</td>
  <td id="val_Ich2"></td>
</tr>
<tr>
  <td colspan="2" style="text-align: center;">Module</td>
</tr>
<tr>
  <td id="ULogic">Voltage logic</td>
  <td id="val_ULogic"></td>
</tr>
<tr>
  <td id="UMotor">Voltage motor</td>
  <td id="val_UMotor"></td>
</tr>
<tr>
  <td id="Usource">Voltage source</td>
  <td id="val_Usource"></td>
</tr>
<tr>
  <td id="U10V">Voltage 10V</td>
  <td id="val_U10V"></td>
</tr>
<tr>
  <td id="U18V">Voltage 18V</td>
  <td id="val_U18V"></td>
</tr>
<!--<tr>
  <td id="UphaseA">Voltage phase A</td>
  <td id="val_UphaseA"></td>
</tr>
<tr>
```

```
          <td id="UphaseB">Voltage phase B</td>
          <td id="val_UphaseB"></td>
        </tr>
        <tr>
          <td id="UphaseC">Voltage phase C</td>
          <td id="val_UphaseC"></td>
        </tr>
        <tr>
          <td id="Ucommon">Voltage common</td>
          <td id="val_Ucommon"></td>
        </tr>-->
        <tr>
          <td id="Usens">Voltage sens</td>
          <td id="val_Usens"></td>
        </tr>
        <tr>
          <td id="Uhall">Voltage hall</td>
          <td id="val_Uhall"></td>
        </tr>
        <tr>
          <td id="Icommon">Current common</td>
          <td id="val_Icommon"></td>
        </tr>
        <tr>
          <td id="T">T</td>
          <td id="val_T"></td>
        </tr>
        <!--<tr>
          <td id="IphaseB">Current phase B</td>
          <td id="val_IphaseB"></td>
        </tr>
        <tr>
          <td id="IphaseC">Current phase C</td>
          <td id="val_IphaseC"></td>
        </tr>-->
    </tbody>
  </table>
</div>
<div>
  <table class="table table-dark table-striped table-hover" id="sel_table">
    <thead>
      <tr>
        <th scope="col">CanID</th>
        <th scope="col">NodeID</th>
        <th scope="col">Type</th>
        <th scope="col">Sens</th>
        <th scope="col">Drive</th>
        <th scope="col">Current</th>
        <th scope="col">Ul</th>
        <th scope="col">Up</th>
        <th scope="col">Error</th>
```

```
          <th scope="col">Action</th>
        </tr>
      </thead>
      <tbody id="table_body">
        {% for i in range(1,33) %}
          <tr id = "sel_row" hidden>
            <td id="CanID">1</td>
            <td id="NodeID">{{ i }}</td>
            <td id="Type"></td>
            <td id="Sens"></td>
            <td id="Drive"></td>
            <td id="Current"></td>
            <td id="Ul"></td>
            <td id="Up"></td>
            <td id="Error"></td>
            <td id="Action">
              <button type="button" class="btn_drive1_on btn btn-outline-primary"
id="{{i}}" name="0">DriveOn</button>
              <button type="button" class="btn_drive1_off btn btn-outline-
primary" id="{{i}}" name="0">DriveOff</button>
            </td>
          </tr>
        {% endfor %}
        {% for i in range(1,33) %}
          <tr id = "sel_row" hidden>
            <td id="CanID">2</td>
            <td id="NodeID">{{ i }}</td>
            <td id="Type"></td>
            <td id="Sens"></td>
            <td id="Drive"></td>
            <td id="Current"></td>
            <td id="Ul"></td>
            <td id="Up"></td>
            <td id="Error"></td>
            <td id="Action">
              <button type="button" class="btn_drive1_on  btn btn-outline-
primary" value="DriveOn" id="{{32+i}}" name="0">DriveOn</button>
              <button type="button" class="btn_drive1_off  btn btn-outline-
primary" value="DriveOff" id="{{32+i}}" name="0">DriveOff</button>
            </td>
          </tr>
        {% endfor %}
      </tbody>
    </table>
  </div>
</div>

<div class="float_parent">
  <div class="float">
    <table class="table table-dark table-striped"  id="tip_table">
      <thead>
```

```
        <tr>
          <th scope="col">Name</th>
          <th scope="col">Val</th>
        </tr>
      </thead>
      <tbody id="table_body">
          <tr>
            <td id="tip_name_1">Type</td>
            <td id="tip_val_1">
              <select name="type" id="dev_type" style="width: 105px;">
                  <option value="0">ECO</option>
                  <option value="1">AI2</option>
                  <option value="2">Interroll</option>
              </select>
            </td>
          </tr>
          <tr>
            <td id="tip_name_2">Speed(0..1000)</td>
            <td id="tip_val_2"><input type="text" id="dev_speed"
size="5"></input></td>
          </tr>
          <tr>
            <td id="tip_name_2">Overcurrent</td>
            <td id="tip_val_2"><input type="text" id="dev_current"
size="5"></input></td>
          </tr>
<!--          <tr>
            <td id="tip_name_3">Dir</td>
            <td id="tip_val_3">
              <select name="dir" id="dev_dir" style="width: 105px;">
                  <option value="0">CW</option>
                  <option value="1">CCW</option>
              </select>
            </td>
          </tr>-->
          <tr>
            <td id="tip_name_4">RampOn(0..250)</td>
            <td id="tip_val_4"><input type="text" id="dev_rampup"
size="5"></input></td>
          </tr>
          <tr>
            <td id="tip_name_5">RampOff(0..250)</td>
            <td id="tip_val_5"><input type="text" id="dev_rampdown"
size="5"></input></td>
          </tr>
          <tr>
            <td id="tip_name_6">SelectADC</td>
            <td id="tip_val_6">
              <select name="selectADC" id="dev_selectADC" style="width: 105px;">
                  <option value="0">CURRENT_MOTOR</option>
                  <option value="1">VOLTAGE_MOTOR</option>
```

```
                    <option value="2">VOLTAGE_LOGIC</option>
                    <option value="3">VOLTAGE_SOURCE</option>
                    <option value="4">CURRENT_PHASEB</option>
                    <option value="5">VOLTAGE_10V</option>
                    <option value="6">CURRENT_PHASEC</option>
                    <option value="7">VOLTAGE_18V</option>
                    <option value="8">VOLTAGE_PHASEA</option>
                    <option value="9">VOLTAGE_PHASEB</option>
                    <option value="10">VOLTAGE_PHASEC</option>
                    <option value="11">VOLTAGE_COMMON</option>
                    <option value="12">VOLTAGE_SENS</option>
                    <option value="13">VOLTAGE_HALL</option>
                    <option value="14">VOLTAGE_T</option>
                </select>
              </td>
            </tr>
            <tr>
              <td id="tip_name_7">ADC</td>
              <td id="dev_customADC"></td>
            </tr>
        </tbody>
    </table>
    <div style="text-align: center;">
      <div><input type="checkbox" id="btn_for_all" name="btn_for_all"
class="btn_for_all" ><label for="btn_for_all" style="padding: 0px 0px 5px
5px;">For all</label></div>
      <button type="button" class="drive__save__btn btn btn-outline-primary"
style="width: 100%;">Save</button>
    </div>
  </div>
</div>
</div>


<script>
  let btns1_on = document.querySelectorAll('.btn_drive1_on');
  let btns1_off = document.querySelectorAll('.btn_drive1_off');
  let btns2_on = document.querySelectorAll('.btn_drive2_on');
  let btns2_off = document.querySelectorAll('.btn_drive2_off');
  const btns_save = document.querySelector('.drive__save__btn');
  var dev_type = document.getElementById('dev_type');
  var dev_speed = document.getElementById('dev_speed');
  var dev_dir = document.getElementById('dev_dir');
  var dev_rampup = document.getElementById('dev_rampup');
  var dev_rampdown = document.getElementById('dev_rampdown');
  var dev_selectADC = document.getElementById('dev_selectADC');
  var dev_customADC = document.getElementById('dev_customADC');
  var dev_customADC = document.getElementById('dev_customADC');
  var dev_for_all = document.getElementById('btn_for_all');
  var sel_id = 0;
  var sel_can_id = 0;
```

```
  var sel_node_id = 0;
  var param = new Object();;

  btns_save.addEventListener('click', function () {
      if(dev_for_all.checked){
        for (var i = 0; i < 32; i++) {
          param.MotorParam1[3*i] = dev_speed.value&0xFFFF;
          param.MotorParam1[3*i+1]=((dev_rampup.value&0xFF)<<8)|(dev_type.value&0
xFF)
          param.MotorParam1[3*i+2]=((dev_current.value&0xFF)<<8)|(dev_rampdown.va
lue&0xFF)
          param.MotorParam2[3*i] = dev_speed.value&0xFFFF;
          param.MotorParam2[3*i+1]=((dev_rampup.value&0xFF)<<8)|(dev_type.value&0
xFF)
          param.MotorParam2[3*i+2]=((dev_current.value&0xFF)<<8)|(dev_rampdown.va
lue&0xFF)
        }
      }else{
        if(sel_can_id==0){
          param.MotorParam1[3*sel_node_id] = dev_speed.value&0xFFFF;
          param.MotorParam1[3*sel_node_id+1]=((dev_rampup.value&0xFF)<<8)|(dev_ty
pe.value&0xFF);
          param.MotorParam1[3*sel_node_id+2]=((dev_current.value&0xFF)<<8)|(dev_r
ampdown.value&0xFF);
        }else{
          param.MotorParam2[3*sel_node_id] = dev_speed.value&0xFFFF;
          param.MotorParam2[3*sel_node_id+1]=((dev_rampup.value&0xFF)<<8)|(dev_ty
pe.value&0xFF);
          param.MotorParam2[3*sel_node_id+2]=((dev_current.value&0xFF)<<8)|(dev_r
ampdown.value&0xFF);
        }
      }
      fetch('/DriveSave', {
          headers : {
              'Content-Type' : 'application/json'
          },
          method : 'POST',
          body : JSON.stringify( param)
      })
      .then(function (response){

          if(response.ok) {
              response.json()
              .then(function(response) {
                  console.log(response);
              });
          }
          else {
              throw Error('Something went wrong');
          }
      })
```

```
    .catch(function(error) {
        console.log(error);
    });
});

function DriveOn(id,can) {
    fetch('/DriveOn', {
        headers : {
            'Content-Type' : 'application/json'
        },
        method : 'POST',
        body : JSON.stringify( {
            'can' : can,
            'dev_id' : id-1,
            'drive_cmd' : 1
        })
    })
    .then(function (response){

        if(response.ok) {
            response.json()
            .then(function(response) {
                console.log(response);
            });
        }
        else {
            throw Error('Something went wrong');
        }
    })
    .catch(function(error) {
        console.log(error);
    });
  }
function DriveOff(id,can) {
    fetch('/DriveOn', {
        headers : {
            'Content-Type' : 'application/json'
        },
        method : 'POST',
        body : JSON.stringify( {
            'can' : can,
            'dev_id' : id-1,
            'drive_cmd' : 0
        })
    })
    .then(function (response){

        if(response.ok) {
            response.json()
            .then(function(response) {
                console.log(response);
```

```
            });
        }
        else {
            throw Error('Something went wrong');
        }
    })
    .catch(function(error) {
        console.log(error);
    });
}


  var dev_table =
document.getElementById('sel_table').getElementsByTagName('tbody')[0];
  var tip_table =
document.getElementById('tip_table').getElementsByTagName('tbody')[0];
  var adc_table =
document.getElementById('adc_table').getElementsByTagName('tbody')[0];
  var err_line = document.getElementById('Error_text');
  var selected = dev_table.getElementsByClassName('selected');
  dev_table.onclick = highlight;

  var new_dev_id = 0
  function highlight(e) {
        if (selected[0]) selected[0].className = '';
        if (e.target.parentNode.id == "sel_row"){
          e.target.parentNode.className = 'selected';
          var row = e.target.parentNode;
          sel_can_id = row.cells[0].innerText - 1
          sel_node_id = row.cells[1].innerText - 1
          new_dev_id = 1;
        }
        var buttonOn;
        var buttonOff;
        if(e.target.textContent == "DriveOn"){
          buttonOn = e.target;
        }
        if(e.target.textContent == "DriveOff"){
          buttonOff = e.target;
        }
          if (buttonOn && this.contains(buttonOn)) {
              DriveOn(buttonOn.id,buttonOn.name);
          }
          if (buttonOff && this.contains(buttonOff)) {
              DriveOff(buttonOff.id,buttonOff.name);
          }

    }

    function fnselect(){
        var element = document.querySelectorAll('.selected');
```

```
        if(element[0]!== undefined){ //it must be selected
         alert(element[0].children[0].firstChild.data);
        }
    }

  setInterval(function() {
      fetch('/data_json').then(
        response => response.json()
      ).then(
        data =>updateData(data)
      ).catch(
        error => ShowError("Error connect to python service")
      )
    }, 500
  );

  function ShowError(str) {
      err_line.innerHTML = str;
      err_line.hidden = false;
  }

  function updateData(data) {

      if(data !== null){
        if(data.Link == true){
          updateElement(0,data)
          param.MotorParam1 = data.MotorParam1;
          param.MotorParam2 = data.MotorParam2;
          val_Vmain.innerText = data.adc0;
          val_5V.innerText = data.adc1;
          val_Vch1.innerText = data.adc2;
          val_Ich1.innerText = data.adc3;
          val_Vch2.innerText = data.adc4;
          val_Ich2.innerText = data.adc5;

          if(ip.value == ""){
            ip.value =
(data.IP[0]&0xFF)+"."+((data.IP[0]>>8)&0xFF)+"."+(data.IP[1]&0xFF)+"."+((data.IP[
1]>>8)&0xFF);
            mask.value =
(data.IP[2]&0xFF)+"."+((data.IP[2]>>8)&0xFF)+"."+(data.IP[3]&0xFF)+"."+((data.IP[
3]>>8)&0xFF);
          }
          if(data.sd == 0){
            ShowError("SD card not found")
            err_line.hidden = false;
          }else if (data.sd == 1){
            err_line.hidden = true;
          }else if (data.sd == 2){
            ShowError("Error mount SD card")
            err_line.hidden = false;
```

```
        }else if (data.sd == 3){
          ShowError("Config.xml not found")
          err_line.hidden = false;
        }
      }else{
        ShowError("Error connect to C++ service")
        err_line.hidden = false;
      }
    }else{
      ShowError("Error connect to C++ service")
      err_line.hidden = false;
    }
}

function updateElement(can,device) {
    var table = document.getElementById('sel_table');
    for (var i = 0; i < 64; i++) {
        if(hmi_ctrl.checked){
          btns1_on[i].disabled = false;
          //btns2_on[i].classList.remove("disable_but");
          btns1_off[i].disabled = false;
          //btns2_off[i].classList.remove("disable_but");
        }else{
          btns1_on[i].disabled = true;
          //btns2_on[i].classList.add("disable_but");
          btns1_off[i].disabled = true;
          //btns2_off[i].classList.add("disable_but");
        }

      row = table.rows[i+1]

      if(row!== undefined) {
        if(device !== null){
          if((device.MotorStatus[i]&0xC000)==0xC000){
            row.hidden=true;
          }
          if((device.MotorStatus[i]&0xC000)==0x8000){
            row.hidden=false;
            //red
            row.classList.add("MotorErr");
          }
          if((device.MotorStatus[i]&0xC000)==0x4000){
            row.hidden=false;
            row.classList.add("MotorNew");
            //blue
          }
          if((device.MotorStatus[i]&0xC000)==0){
            row.hidden=false;
            row.classList.remove("MotorErr");
            row.classList.remove("MotorNew");
            //grey
```

```
            }
            if(i<32){
              if(device.MotorParam1[3*i+1]&0xFF == 0){
                row.cells[2].innerText = "ECO"
              }else if(device.MotorParam1[3*i+1]&0xFF == 1){
                row.cells[2].innerText = "AI2"
              }else if(device.MotorParam1[3*i+1]&0xFF == 2){
                row.cells[2].innerText = "IR"
              }
              row.cells[3].innerText = device.MotorStatus[i]&0x03; //sens
              row.cells[4].innerText = device.MotorStatus[i]&0x04; //drive
              row.cells[5].innerText = ((device.N1ADC2[3*i]&0xFF00)>>8)/10;
//current
              row.cells[6].innerText = ((device.N1ADC1[3*i]&0xFF00)>>8)/10; //Ul
              row.cells[7].innerText = (device.N1ADC1[3*i]&0xFF)/10; //Up
              row.cells[8].innerText = device.MotorStatus[i]; //Error
            }else{
              if(device.MotorParam2[3*(i-32)+1]&0xFF == 0){
                row.cells[2].innerText = "ECO"
              }else if(device.MotorParam2[3*(i-32)+1]&0xFF == 1){
                row.cells[2].innerText = "AI2"
              }else if(device.MotorParam2[3*(i-32)+1]&0xFF == 2){
                row.cells[2].innerText = "IR"
              }
              row.cells[3].innerText = device.MotorStatus[i]&0x03; //sens
              row.cells[4].innerText = device.MotorStatus[i]&0x04; //drive
              row.cells[5].innerText = ((device.N2ADC2[3*(i-32)]&0xFF00)>>8)/10;
//current
              row.cells[6].innerText = ((device.N2ADC1[3*(i-32)]&0xFF00)>>8)/10;
//Ul
              row.cells[7].innerText = ((device.N2ADC1[3*(i-32)]&0xFF))/10; //Up
              row.cells[8].innerText = device.MotorStatus[i]; //Error

            }

          }
        /*}else{
          addRowDev(can,device);*/
        }
      }
      if(sel_can_id == 0){
        if((device.MotorParam1[sel_node_id] !== null)&&(new_dev_id>0)){
          dev_speed.value = device.MotorParam1[3*sel_node_id];
          dev_type.value = (device.MotorParam1[3*sel_node_id+1]&0xFF);
          dev_rampup.value = (device.MotorParam1[3*sel_node_id+1]&0xFF00)>>8;
          dev_rampdown.value = (device.MotorParam1[3*sel_node_id+2]&0xFF);
          dev_current.value = (device.MotorParam1[3*sel_node_id+2]&0xFF00)>>8;
          //dev_dir.value = device.MotorParam1[3*sel_node_id];
          dev_selectADC.value = 0;
          new_dev_id = 0;
        }
```

```
        val_UMotor.innerText = ((device.N1ADC1[3*(sel_node_id)+0]&0xFF))/10;
        val_ULogic.innerText = ((device.N1ADC1[3*(sel_node_id)+0]&0xFF00)>>8)/10;
        val_Usource.innerText = ((device.N1ADC1[3*(sel_node_id)+1]&0xFF))/10;
        val_U10V.innerText = ((device.N1ADC1[3*(sel_node_id)+1]&0xFF00)>>8)/10;
        val_U18V.innerText = ((device.N1ADC1[3*(sel_node_id)+2]&0xFF))/10;
        val_Uhall.innerText = ((device.N1ADC1[3*(sel_node_id)+2]&0xFF00)>>8)/10;

        val_Usens.innerText = ((device.N1ADC2[3*(sel_node_id)+0]&0xFF))/10;
        val_Icommon.innerText =
((device.N1ADC2[3*(sel_node_id)+0]&0xFF00)>>8)/10;
        val_T.innerText = ((device.N1ADC2[3*(sel_node_id)+1]&0xFF))/10;
        dev_customADC.innerText = (device.N1ADC2[3*(sel_node_id)+2]);
        /*val_UphaseA.innerText = device[sel_node_id].BEMFA;
        val_UphaseB.innerText = device[sel_node_id].BEMFB;
        val_UphaseC.innerText = device[sel_node_id].BEMFC;
        val_Ucommon.innerText = device[sel_node_id].BEMFN;
        val_IphaseB.innerText = device[sel_node_id].CPhaseB;
        val_IphaseC.innerText = device[sel_node_id].CPhaseC;
        */
      }else{
        if((device.MotorParam2[sel_node_id] !== null)&&(new_dev_id>0)){
          dev_speed.value = device.MotorParam2[3*sel_node_id];
          dev_type.value = (device.MotorParam2[3*sel_node_id+1]&0xFF);
          dev_rampup.value = (device.MotorParam2[3*sel_node_id+1]&0xFF00)>>8;
          dev_rampdown.value = (device.MotorParam2[3*sel_node_id+2]&0xFF);
          dev_current.value = (device.MotorParam2[3*sel_node_id+2]&0xFF00)>>8;
          //dev_dir.value = device.MotorParam1[3*sel_node_id];
          dev_selectADC.value = 0;
          new_dev_id = 0;
        }
        val_UMotor.innerText = ((device.N2ADC1[3*(sel_node_id)+0]&0xFF))/10;
        val_ULogic.innerText = ((device.N2ADC1[3*(sel_node_id)+0]&0xFF00)>>8)/10;
        val_Usource.innerText = ((device.N2ADC1[3*(sel_node_id)+1]&0xFF))/10;
        val_U10V.innerText = ((device.N2ADC1[3*(sel_node_id)+1]&0xFF00)>>8)/10;
        val_U18V.innerText = ((device.N2ADC1[3*(sel_node_id)+2]&0xFF))/10;
        val_Uhall.innerText = ((device.N2ADC1[3*(sel_node_id)+2]&0xFF00)>>8)/10;

        val_Usens.innerText = ((device.N2ADC2[3*(sel_node_id)+0]&0xFF))/10;
        val_Icommon.innerText =
((device.N2ADC2[3*(sel_node_id)+0]&0xFF00)>>8)/10;
        val_T.innerText = ((device.N2ADC2[3*(sel_node_id)+1]&0xFF))/10;
        dev_customADC.innerText = (device.N2ADC2[3*(sel_node_id)+2]);
        /*val_UphaseA.innerText = device[sel_node_id].BEMFA;
        val_UphaseB.innerText = device[sel_node_id].BEMFB;
        val_UphaseC.innerText = device[sel_node_id].BEMFC;
        val_Ucommon.innerText = device[sel_node_id].BEMFN;
        val_IphaseB.innerText = device[sel_node_id].CPhaseB;
        val_IphaseC.innerText = device[sel_node_id].CPhaseC;
        */
      }
  }
```

```
</script>
{% endblock %}
```

5. Описание обновления данных на странице

```
<table class="table table-dark table-striped">
  <thead>
    <tr>
      <th scope="col">ID</th>
      <th scope="col">Starting Time</th>
      <th scope="col">Team A</th>
      <th scope="col">Score</th>
      <th scope="col">Team B</th>
      <th scope="col">Minute</th>
    </tr>
  </thead>
  <tbody id="table_body">
    {% for match in matches %}
      <tr>
        <td id="id_{{ match.id }}">{{ match.id }}</td>
        <td id="starting_time_{{ match.id }}">{{ match.starting_time }}</td>
        <td id="team_a_{{ match.id }}">{{ match.team_a }}</td>
        <td id="score_{{ match.id }}">{{ match.score }}</td>
        <td id="team_b_{{ match.id }}">{{ match.team_b }}</td>
        <td id="minute_{{ match.id }}">{{ match.minute }}</td>
      </tr>
    {% endfor %}
  </tbody>
</table>
<script>
  setInterval(function() {
      fetch('/data_json').then(
        response => response.json()
      ).then(
        data =>
          data.forEach(match =>
            updateElement(match)
          )
      )
    }, 1000
  );

function updateElement(match) {
    Object.entries(match).forEach(([k,v]) => {
        element = document.getElementById(k + "_" + match.id);
```

```
        // Get Previous value first
        previousValue = element.innerHTML;

        // If Previous value is not equal to the new value, change it!
        if (previousValue !== v.toString()) {
            blinkUpdate(element, v.toString())
        }
    })
}

function blinkUpdate(element, newValue, newColor='#00ff00') {
    previousColor = element.style.color;
    element.innerHTML = newValue;
    element.style.color = newColor;
    // Set back to original color after timeout ms
    setTimeout(() => {
        element.style.color = previousColor
    }, 2000)
}
</script>
```

1. Описание основных модулей, старт всех систем, основной цикл работы P-NET

```
#include "sampleapp_common.h"

#include "app_utils.h"
#include "app_gsdml.h"
#include "app_data.h"
#include "app_log.h"
#include "app_can.h"
#include "app_web.h"
#include "config.h"
#include "modbus-private.h"
#include "modbus-tcp-private.h"
#include "modbus-tcp.h"
#include "modbus-version.h"
#include "modbus.h"
#include "app_ecc.h"
#include "osal.h"
#include "pnal.h"
#include <pnet_api.h>
```

```c
#include <pf_types.h>
#include <pf_cmina.h>

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gpiod.h>

/* Events handled by main task */
#define APP_EVENT_READY_FOR_DATA BIT (0)
#define APP_EVENT_TIMER          BIT (1)
#define APP_EVENT_ALARM          BIT (2)
#define APP_EVENT_SM_RELEASED    BIT (3)
#define APP_EVENT_ABORT          BIT (15)

/* Defines used for alarm demo functionality */
#define CHANNEL_ERRORTYPE_SHORT_CIRCUIT                         0x0001
#define CHANNEL_ERRORTYPE_LINE_BREAK                            0x0006
#define CHANNEL_ERRORTYPE_DATA_TRANSMISSION_IMPOSSIBLE          0x8000
#define CHANNEL_ERRORTYPE_NETWORK_COMPONENT_FUNCTION_MISMATCH 0x8008
#define EXTENDED_CHANNEL_ERRORTYPE_FRAME_DROPPED                0x8000
#define EXTENDED_CHANNEL_ERRORTYPE_MAUTYPE_MISMATCH             0x8001
#define EXTENDED_CHANNEL_ERRORTYPE_LINE_DELAY_MISMATCH          0x8002

#define APP_ALARM_USI                       0x0010
#define APP_DIAG_CHANNEL_NUMBER             4
#define APP_DIAG_CHANNEL_DIRECTION          PNET_DIAG_CH_PROP_DIR_INPUT
#define APP_DIAG_CHANNEL_NUMBER_OF_BITS     PNET_DIAG_CH_PROP_TYPE_1_BIT
#define APP_DIAG_CHANNEL_SEVERITY           PNET_DIAG_CH_PROP_MAINT_FAULT
#define APP_DIAG_CHANNEL_ERRORTYPE          CHANNEL_ERRORTYPE_SHORT_CIRCUIT
#define APP_DIAG_CHANNEL_ADDVALUE_A         0
#define APP_DIAG_CHANNEL_ADDVALUE_B         1234
#define APP_DIAG_CHANNEL_EXTENDED_ERRORTYPE 0
#define APP_DIAG_CHANNEL_QUAL_SEVERITY      0 /* Not used (Max one bit set) */

typedef enum app_demo_state
{
   APP_DEMO_STATE_ALARM_SEND = 0,
   APP_DEMO_STATE_LOGBOOK_ENTRY,
   APP_DEMO_STATE_ABORT_AR,
   APP_DEMO_STATE_CYCLIC_REDUNDANT,
   APP_DEMO_STATE_CYCLIC_NORMAL,
   APP_DEMO_STATE_DIAG_STD_ADD,
   APP_DEMO_STATE_DIAG_STD_UPDATE,
   APP_DEMO_STATE_DIAG_STD_REMOVE,
   APP_DEMO_STATE_DIAG_USI_ADD,
   APP_DEMO_STATE_DIAG_USI_UPDATE,
   APP_DEMO_STATE_DIAG_USI_REMOVE,
} app_demo_state_t;

typedef struct app_data_t
```

```
{
   pnet_t * net;

   /* P-Net configuration passed in app_init(). */
   const pnet_cfg_t * pnet_cfg;

   /* Application API for administration of plugged
    * (sub)modules and connection state. */
   app_api_t main_api;

   os_timer_t * main_timer;
   os_event_t * main_events;

   bool alarm_allowed;
   pnet_alarm_argument_t alarm_arg;
   app_demo_state_t alarm_demo_state;
   uint8_t alarm_payload[APP_GSDML_ALARM_PAYLOAD_SIZE];

   bool button1_pressed;
   bool button2_pressed;
   bool button2_pressed_previous;

   /* Counters used to control when buttons are checked
    * and process data is updated */
   uint32_t buttons_tick_counter;
   uint32_t process_data_tick_counter;

} app_data_t;

/* Forward declarations */
static void app_plug_dap (app_data_t * app, uint16_t number_of_ports);
static int app_set_initial_data_and_ioxs (app_data_t * app);
static void app_cyclic_data_callback (app_subslot_t * subslot, void * tag);

/** Static app data */
static app_data_t app_state;

pnet_t * app_get_pnet_instance (app_data_t * app)
{
   if (app == NULL)
   {
      return NULL;
   }

   return app->net;
}

/** Check if we are connected to the controller
 *
 * @param app             InOut:    Application handle
 * @return true if we are connected to the IO-controller
```

```
 */
static bool app_is_connected_to_controller (app_data_t * app)
{
   return app->main_api.ar[0].arep != UINT32_MAX;
}

app_data_t * app_init (const pnet_cfg_t * pnet_cfg, const app_args_t * app_args)
{
   app_data_t * app;
   uint16_t i;

   APP_LOG_INFO ("Init P-Net stack and sample application\n");

   app = &app_state;

   app->alarm_allowed = true;
   for (i = 0; i < PNET_MAX_AR; ++i)
   {
      app->main_api.ar[i].arep = UINT32_MAX;
      app->main_api.ar[i].events = 0;
   }
   app->pnet_cfg = pnet_cfg;

   app->net = pnet_init (app->pnet_cfg);

   if (app->net == NULL)
   {
      return NULL;
   }

   return app;
}

/**
 * Callback for timer tick.
 *
 * This is a callback for the timer defined in OSAL.
 * See \a os_timer_create() for details.
 *
 * @param timer   InOut: Timer instance
 * @param arg     InOut: User defined argument, app_data_t pointer
 */
static void main_timer_tick (os_timer_t * timer, void * arg)
{
   app_data_t * app = (app_data_t *)arg;

   os_event_set (app->main_events, APP_EVENT_TIMER);
}

int app_start (app_data_t * app, app_run_in_separate_task_t task_config)
{
```

```
   APP_LOG_INFO ("Start sample application main loop\n");
   if (app == NULL)
   {
      return -1;
   }

   app->main_events = os_event_create();
   if (app->main_events == NULL)
   {
      return -1;
   }

   app->main_timer = os_timer_create (
      APP_TICK_INTERVAL_US,
      main_timer_tick,
      (void *)app,
      false);

   if (app->main_timer == NULL)
   {
      os_event_destroy (app->main_events);
      return -1;
   }

   if (task_config == RUN_IN_SEPARATE_THREAD)
   {
      os_thread_create (
         "p-net_sample_app",
         APP_MAIN_THREAD_PRIORITY,
         APP_MAIN_THREAD_STACKSIZE,
         app_loop_forever,
         (void *)app);
   }

   os_timer_start (app->main_timer);

   return 0;
}

/**
 * Set outputs to default value
 */
static void app_set_outputs_default_value (void)
{
   APP_LOG_DEBUG ("Setting outputs to default values.\n");
   app_data_set_default_outputs();
}

/**
 * Set event flag(s) for one arep.
 */
```

```
static void app_event_set (
   app_data_t * app,
   uint32_t arep,
   uint32_t event,
   bool add_arep)
{
   app_ar_iterator_t iter;
   app_ar_t * ar;

   app_ar_iterator_init (&iter, &app->main_api);
   while (app_ar_iterator_next (&iter, &ar))
   {
      if (app_ar_arep (ar) == arep)
      {
         app_ar_event_set (ar, event);
         break;
      }
   }
   if (add_arep)
   {
      if (app_ar_iterator_done (&iter))
      {
         if (app_ar_add_arep (&app->main_api, arep, &ar))
         {
            app_ar_event_set (ar, event);
         }
      }
   }
   os_event_set (app->main_events, event);
}

/********************************** Callbacks ********************************/

static int app_connect_ind (
   pnet_t * net,
   void * arg,
   uint32_t arep,
   pnet_result_t * p_result)
{
   APP_LOG_DEBUG ("PLC connect indication. AREP: %u\n", arep);
   /*
    *  Handle the request on an application level.
    *  This is a very simple application which does not need to handle anything.
    *  All the needed information is in the AR data structure.
    */

   return 0;
}

static int app_release_ind (
   pnet_t * net,
```

```
   void * arg,
   uint32_t arep,
   pnet_result_t * p_result)
{
   APP_LOG_DEBUG ("PLC release (disconnect) indication. AREP: %u\n", arep);

   app_set_outputs_default_value();

   return 0;
}

static int app_dcontrol_ind (
   pnet_t * net,
   void * arg,
   uint32_t arep,
   pnet_control_command_t control_command,
   pnet_result_t * p_result)
{
   APP_LOG_DEBUG (
      "PLC dcontrol message (The PLC is done with parameter writing). "
      "AREP: %u  Command: %s\n",
      arep,
      app_utils_dcontrol_cmd_to_string (control_command));

   return 0;
}

static int app_sm_released_ind (
   pnet_t * net,
   void * arg,
   uint32_t arep,
   uint32_t api,
   uint16_t slot_number,
   uint16_t subslot_number,
   pnet_result_t * p_result)
{
   app_data_t * app = (app_data_t *)arg;

   APP_LOG_DEBUG (
      "SM released indication.\n"
      "  AREP: %u API: %u Slot: 0x%x Subslot: 0x%x\n",
      arep,
      api,
      slot_number,
      subslot_number);

   app_event_set (app, arep, APP_EVENT_SM_RELEASED, false);

   return 0;
}
```

```
static int app_ccontrol_cnf (
   pnet_t * net,
   void * arg,
   uint32_t arep,
   pnet_result_t * p_result)
{
   APP_LOG_DEBUG (
      "PLC ccontrol message confirmation (The PLC has received our Application "
      "Ready message). AREP: %u  Status codes: %d %d %d %d\n",
      arep,
      p_result->pnio_status.error_code,
      p_result->pnio_status.error_decode,
      p_result->pnio_status.error_code_1,
      p_result->pnio_status.error_code_2);

   return 0;
}

static int app_write_ind (
   pnet_t * net,
   void * arg,
   uint32_t arep,
   uint32_t api,
   uint16_t slot_nbr,
   uint16_t subslot_nbr,
   uint16_t idx,
   uint16_t sequence_number,
   uint16_t write_length,
   const uint8_t * p_write_data,
   pnet_result_t * p_result)
{
   int result = 0;
   app_data_t * app = (app_data_t *)arg;
   app_subslot_t * subslot;
   APP_LOG_DEBUG (
      "PLC write record indication.\n"
      "  AREP: %u API: %u Slot: %2u Subslot: %u Index: %u Sequence: %2u "
      "Length: %u\n",
      arep,
      api,
      slot_nbr,
      subslot_nbr,
      (unsigned)idx,
      sequence_number,
      write_length);

   subslot = app_utils_subslot_get (&app->main_api, slot_nbr, subslot_nbr);
   if (subslot == NULL)
   {
      APP_LOG_WARNING (
         "No submodule plugged in AREP: %u API: %u Slot: %2u Subslot: %u "
```

```
            "Index will not be written.\n",
            arep,
            api,
            slot_nbr,
            subslot_nbr);
         p_result->pnio_status.error_code = PNET_ERROR_CODE_WRITE;
         p_result->pnio_status.error_decode = PNET_ERROR_DECODE_PNIORW;
         p_result->pnio_status.error_code_1 = PNET_ERROR_CODE_1_APP_WRITE_ERROR;
         p_result->pnio_status.error_code_2 = 0; /* User specific */

         return -1;
      }

      result = app_data_write_parameter (
         slot_nbr,
         subslot_nbr,
         subslot->submodule_id,
         idx,
         p_write_data,
         write_length);
      if (result != 0)
      {
         APP_LOG_WARNING (
            "Failed to write index for AREP: %u API: %u Slot: %2u Subslot: %u "
            "index %u.\n",
            arep,
            api,
            slot_nbr,
            subslot_nbr,
            idx);
         p_result->pnio_status.error_code = PNET_ERROR_CODE_WRITE;
         p_result->pnio_status.error_decode = PNET_ERROR_DECODE_PNIORW;
         p_result->pnio_status.error_code_1 = PNET_ERROR_CODE_1_APP_WRITE_ERROR;
         p_result->pnio_status.error_code_2 = 0; /* User specific */
      }

      return result;
   }

   static int app_read_ind (
      pnet_t * net,
      void * arg,
      uint32_t arep,
      uint32_t api,
      uint16_t slot_nbr,
      uint16_t subslot_nbr,
      uint16_t idx,
      uint16_t sequence_number,
      uint8_t ** pp_read_data,
      uint16_t * p_read_length,
      pnet_result_t * p_result)
```

```
{
   int result = 0;
   app_data_t * app = (app_data_t *)arg;
   app_subslot_t * subslot;

   APP_LOG_DEBUG (
      "PLC read record indication.\n"
      "  AREP: %u API: %u Slot: %2u Subslot: %u Index: %u Sequence: %2u Max "
      "length: %u\n",
      arep,
      api,
      slot_nbr,
      subslot_nbr,
      (unsigned)idx,
      sequence_number,
      (unsigned)*p_read_length);

   subslot = app_utils_subslot_get (&app->main_api, slot_nbr, subslot_nbr);
   if (subslot == NULL)
   {
      APP_LOG_WARNING (
         "No submodule plugged in AREP: %u API: %u Slot: %2u Subslot: %u "
         "Index will not be read.\n",
         arep,
         api,
         slot_nbr,
         subslot_nbr);
      p_result->pnio_status.error_code = PNET_ERROR_CODE_READ;
      p_result->pnio_status.error_decode = PNET_ERROR_DECODE_PNIORW;
      p_result->pnio_status.error_code_1 = PNET_ERROR_CODE_1_APP_READ_ERROR;
      p_result->pnio_status.error_code_2 = 0; /* User specific */
      return -1;
   }

   result = app_data_read_parameter (
      slot_nbr,
      subslot_nbr,
      subslot->submodule_id,
      idx,
      pp_read_data,
      p_read_length);

   if (result != 0)
   {
      APP_LOG_WARNING (
         "Failed to read index for AREP: %u API: %u Slot: %2u Subslot: %u "
         "index %u.\n",
         arep,
         api,
         slot_nbr,
         subslot_nbr,
```

```
            idx);
         p_result->pnio_status.error_code = PNET_ERROR_CODE_READ;
         p_result->pnio_status.error_decode = PNET_ERROR_DECODE_PNIORW;
         p_result->pnio_status.error_code_1 = PNET_ERROR_CODE_1_APP_READ_ERROR;
         p_result->pnio_status.error_code_2 = 0; /* User specific */
      }

      return result;
   }

   static int app_state_ind (
      pnet_t * net,
      void * arg,
      uint32_t arep,
      pnet_event_values_t event)
   {
      uint16_t err_cls = 0;  /* Error code 1 */
      uint16_t err_code = 0; /* Error code 2 */
      const char * error_class_description = "";
      const char * error_code_description = "";

      app_data_t * app = (app_data_t *)arg;

      APP_LOG_DEBUG (
         "Event indication %s   AREP: %u\n",
         app_utils_event_to_string (event),
         arep);

      if (event == PNET_EVENT_ABORT)
      {
         if (pnet_get_ar_error_codes (net, arep, &err_cls, &err_code) == 0)
         {
            app_utils_get_error_code_strings (
               err_cls,
               err_code,
               &error_class_description,
               &error_code_description);
            APP_LOG_DEBUG (
               "    Error class: 0x%02x %s \n"
               "    Error code:  0x%02x %s \n",
               (unsigned)err_cls,
               error_class_description,
               (unsigned)err_code,
               error_code_description);
         }
         else
         {
            APP_LOG_DEBUG ("    No error status available\n");
         }
         /* Set output values */
         app_set_outputs_default_value();
```

```
      app_event_set (app, arep, APP_EVENT_ABORT, false);
   }
   else if (event == PNET_EVENT_PRMEND)
   {
      app_set_initial_data_and_ioxs (app);

      (void)pnet_set_provider_state (net, true);

      /* Send application ready at next tick
         Do not call pnet_application_ready() here as it will affect
         the internal stack states */
      app_event_set (app, arep, APP_EVENT_READY_FOR_DATA, true);
   }
   else if (event == PNET_EVENT_DATA)
   {
      APP_LOG_DEBUG ("Cyclic data transmission started\n\n");
   }

   return 0;
}

static int app_reset_ind (
   pnet_t * net,
   void * arg,
   bool should_reset_application,
   uint16_t reset_mode)
{
   APP_LOG_DEBUG (
      "PLC reset indication. Application reset mandatory: %u  Reset mode: %d\n",
      should_reset_application,
      reset_mode);

   return 0;
}

static int app_signal_led_ind (pnet_t * net, void * arg, bool led_state)
{
   APP_LOG_INFO ("Profinet signal LED indication. New state: %u\n", led_state);

   app_set_led (APP_PROFINET_SIGNAL_LED_ID, led_state);
   return 0;
}

static int app_exp_module_ind (
   pnet_t * net,
   void * arg,
   uint32_t api,
   uint16_t slot,
   uint32_t module_ident)
{
```

```
int ret = -1;
int result = 0;
app_data_t * app = (app_data_t *)arg;
const char * module_name = "unknown";
const app_gsdml_module_t * module_config;

APP_LOG_DEBUG ("Module plug indication\n");

if (slot >= PNET_MAX_SLOTS)
{
   APP_LOG_ERROR (
      "Wrong slot number received: %u  It should be less than %u\n",
      slot,
      PNET_MAX_SLOTS);
   return -1;
}

module_config = app_gsdml_get_module_cfg (module_ident);
if (module_config == NULL)
{
   APP_LOG_ERROR ("  Module ID %08x not found.\n", (unsigned)module_ident);
   /*
    * Needed to pass Behavior scenario 2
    */
   APP_LOG_DEBUG ("  Plug expected module anyway\n");
}
else
{
   module_name = module_config->name;
}

APP_LOG_DEBUG ("  Pull old module.    API: %u Slot: %2u\n", api, slot);
result = pnet_pull_module (net, api, slot);

if (result == 0)
{
   (void)app_utils_pull_module (&app->main_api, slot);
}

APP_LOG_DEBUG (
   "  Plug module.        API: %u Slot: %2u Module ID: 0x%x \"%s\"\n",
   api,
   slot,
   (unsigned)module_ident,
   module_name);

ret = pnet_plug_module (net, api, slot, module_ident);
if (ret == 0)
{
   (void)app_utils_plug_module (
      &app->main_api,
```

```
            slot,
            module_ident,
            module_name);
   }
   else
   {
      APP_LOG_ERROR (
         "Plug module failed. Ret: %u API: %u Slot: %2u Module ID: 0x%x\n",
         ret,
         api,
         slot,
         (unsigned)module_ident);
   }

   return ret;
}

static int app_exp_submodule_ind (
   pnet_t * net,
   void * arg,
   uint32_t api,
   uint16_t slot,
   uint16_t subslot,
   uint32_t module_id,
   uint32_t submodule_id,
   const pnet_data_cfg_t * p_exp_data)
{
   int ret = -1;
   int result = 0;
   pnet_data_cfg_t data_cfg = {0};
   app_data_t * app = (app_data_t *)arg;
   const app_gsdml_submodule_t * submod_cfg;
   const char * name = "Unsupported";
   app_utils_cyclic_callback cyclic_data_callback = NULL;

   APP_LOG_DEBUG ("Submodule plug indication.\n");

   submod_cfg = app_gsdml_get_submodule_cfg (submodule_id);
   if (submod_cfg != NULL)
   {
      data_cfg.data_dir = submod_cfg->data_dir;
      data_cfg.insize = submod_cfg->insize;
      data_cfg.outsize = submod_cfg->outsize;
      name = submod_cfg->name;

      if (data_cfg.insize > 0 || data_cfg.outsize > 0)
      {
         cyclic_data_callback = app_cyclic_data_callback;
      }
   }
   else
```

```
{
   APP_LOG_WARNING (
      "  Submodule ID 0x%x in module ID 0x%x not found. API: %u Slot: %2u "
      "Subslot %u \n",
      (unsigned)submodule_id,
      (unsigned)module_id,
      api,
      slot,
      subslot);

   /*
    * Needed for behavior scenario 2 to pass.
    * Iops will be set to bad for this subslot
    */
   APP_LOG_WARNING ("  Plug expected submodule anyway \n");

   data_cfg.data_dir = p_exp_data->data_dir;
   data_cfg.insize = p_exp_data->insize;
   data_cfg.outsize = p_exp_data->outsize;
}

APP_LOG_DEBUG (
   "  Pull old submodule. API: %u Slot: %2u Subslot: %u\n",
   api,
   slot,
   subslot);

result = pnet_pull_submodule (net, api, slot, subslot);
if (result == 0)
{
   (void)app_utils_pull_submodule (&app->main_api, slot, subslot);
}

APP_LOG_DEBUG (
   "  Plug submodule.      API: %u Slot: %2u Module ID: 0x%-4x\n"
   "                       Subslot: %u Submodule ID: 0x%x \"%s\"\n",
   api,
   slot,
   (unsigned)module_id,
   subslot,
   (unsigned)submodule_id,
   name);

APP_LOG_DEBUG (
   "                       Data Dir: %s  In: %u bytes  Out: %u bytes\n",
   app_utils_submod_dir_to_string (data_cfg.data_dir),
   data_cfg.insize,
   data_cfg.outsize);

if (
   data_cfg.data_dir != p_exp_data->data_dir ||
```

```
      data_cfg.insize != p_exp_data->insize ||
      data_cfg.outsize != p_exp_data->outsize)
   {
      APP_LOG_WARNING (
         "    Warning expected  Data Dir: %s  In: %u bytes  Out: %u bytes\n",
         app_utils_submod_dir_to_string (p_exp_data->data_dir),
         p_exp_data->insize,
         p_exp_data->outsize);
   }
   ret = pnet_plug_submodule (
      net,
      api,
      slot,
      subslot,
      module_id,
      submodule_id,
      data_cfg.data_dir,
      data_cfg.insize,
      data_cfg.outsize);

   if (ret == 0)
   {
      (void)app_utils_plug_submodule (
         &app->main_api,
         slot,
         subslot,
         submodule_id,
         &data_cfg,
         name,
         cyclic_data_callback,
         app);
   }
   else
   {
      APP_LOG_ERROR (
         "  Plug submodule failed. Ret: %u API: %u Slot: %2u Subslot %u "
         "Module ID: 0x%x Submodule ID: 0x%x \n",
         ret,
         api,
         slot,
         subslot,
         (unsigned)module_id,
         (unsigned)submodule_id);
   }

   return ret;
}

static int app_new_data_status_ind (
   pnet_t * net,
   void * arg,
```

```
   uint32_t arep,
   uint32_t crep,
   uint8_t changes,
   uint8_t data_status)
{
   bool is_running = data_status & BIT (PNET_DATA_STATUS_BIT_PROVIDER_STATE);
   bool is_valid = data_status & BIT (PNET_DATA_STATUS_BIT_DATA_VALID);

   APP_LOG_DEBUG (
      "Data status indication. AREP: %u  Data status changes: 0x%02x  "
      "Data status: 0x%02x\n",
      arep,
      changes,
      data_status);
   APP_LOG_DEBUG (
      "   %s, %s, %s, %s, %s\n",
      is_running ? "Run" : "Stop",
      is_valid ? "Valid" : "Invalid",
      (data_status & BIT (PNET_DATA_STATUS_BIT_STATE)) ? "Primary" : "Backup",
      (data_status & BIT (PNET_DATA_STATUS_BIT_STATION_PROBLEM_INDICATOR))
         ? "Normal operation"
         : "Problem",
      (data_status & BIT (PNET_DATA_STATUS_BIT_IGNORE))
         ? "Ignore data status"
         : "Evaluate data status");

   if (is_running == false || is_valid == false)
   {
      app_set_outputs_default_value();
   }

   return 0;
}

static int app_alarm_ind (
   pnet_t * net,
   void * arg,
   uint32_t arep,
   const pnet_alarm_argument_t * p_alarm_arg,
   uint16_t data_len,
   uint16_t data_usi,
   const uint8_t * p_data)
{
   app_data_t * app = (app_data_t *)arg;

   APP_LOG_DEBUG (
      "Alarm indication. AREP: %u API: %d Slot: %d Subslot: %d "
      "Type: %d Seq: %d Length: %d USI: %d\n",
      arep,
      p_alarm_arg->api_id,
      p_alarm_arg->slot_nbr,
```

```
      p_alarm_arg->subslot_nbr,
      p_alarm_arg->alarm_type,
      p_alarm_arg->sequence_number,
      data_len,
      data_usi);

   app->alarm_arg = *p_alarm_arg;

   app_event_set (app, arep, APP_EVENT_ALARM, false);

   return 0;
}

static int app_alarm_cnf (
   pnet_t * net,
   void * arg,
   uint32_t arep,
   const pnet_pnio_status_t * p_pnio_status)
{
   app_data_t * app = (app_data_t *)arg;

   APP_LOG_DEBUG (
      "PLC alarm confirmation. AREP: %u  Status code %u, "
      "%u, %u, %u\n",
      arep,
      p_pnio_status->error_code,
      p_pnio_status->error_decode,
      p_pnio_status->error_code_1,
      p_pnio_status->error_code_2);

   app->alarm_allowed = true;

   return 0;
}

static int app_alarm_ack_cnf (pnet_t * net, void * arg, uint32_t arep, int res)
{
   APP_LOG_DEBUG (
      "PLC alarm ACK confirmation. AREP: %u  Result: "
      "%d\n",
      arep,
      res);

   return 0;
}

/**********************************************************************/

/**
 * Plug all DAP (sub)modules
 * Use existing callback functions to plug the (sub-)modules
```

```
 * @param app              InOut:   Application handle
 * @param number_of_ports  In:      Number of active ports
 */
static void app_plug_dap (app_data_t * app, uint16_t number_of_ports)
{
   const pnet_data_cfg_t cfg_dap_data = {
      .data_dir = PNET_DIR_NO_IO,
      .insize = 0,
      .outsize = 0,
   };

   APP_LOG_DEBUG ("\nPlug DAP module and its submodules\n");

   app_exp_module_ind (
      app->net,
      app,
      APP_GSDML_API,
      PNET_SLOT_DAP_IDENT,
      PNET_MOD_DAP_IDENT);

   app_exp_submodule_ind (
      app->net,
      app,
      APP_GSDML_API,
      PNET_SLOT_DAP_IDENT,
      PNET_SUBSLOT_DAP_IDENT,
      PNET_MOD_DAP_IDENT,
      PNET_SUBMOD_DAP_IDENT,
      &cfg_dap_data);

   app_exp_submodule_ind (
      app->net,
      app,
      APP_GSDML_API,
      PNET_SLOT_DAP_IDENT,
      PNET_SUBSLOT_DAP_INTERFACE_1_IDENT,
      PNET_MOD_DAP_IDENT,
      PNET_SUBMOD_DAP_INTERFACE_1_IDENT,
      &cfg_dap_data);

   app_exp_submodule_ind (
      app->net,
      app,
      APP_GSDML_API,
      PNET_SLOT_DAP_IDENT,
      PNET_SUBSLOT_DAP_INTERFACE_1_PORT_1_IDENT,
      PNET_MOD_DAP_IDENT,
      PNET_SUBMOD_DAP_INTERFACE_1_PORT_1_IDENT,
      &cfg_dap_data);

   if (number_of_ports >= 2)
```

```
   {
      app_exp_submodule_ind (
         app->net,
         app,
         APP_GSDML_API,
         PNET_SLOT_DAP_IDENT,
         PNET_SUBSLOT_DAP_INTERFACE_1_PORT_2_IDENT,
         PNET_MOD_DAP_IDENT,
         PNET_SUBMOD_DAP_INTERFACE_1_PORT_2_IDENT,
         &cfg_dap_data);
   }

   if (number_of_ports >= 3)
   {
      app_exp_submodule_ind (
         app->net,
         app,
         APP_GSDML_API,
         PNET_SLOT_DAP_IDENT,
         PNET_SUBSLOT_DAP_INTERFACE_1_PORT_3_IDENT,
         PNET_MOD_DAP_IDENT,
         PNET_SUBMOD_DAP_INTERFACE_1_PORT_3_IDENT,
         &cfg_dap_data);
   }

   if (number_of_ports >= 4)
   {
      app_exp_submodule_ind (
         app->net,
         app,
         APP_GSDML_API,
         PNET_SLOT_DAP_IDENT,
         PNET_SUBSLOT_DAP_INTERFACE_1_PORT_4_IDENT,
         PNET_MOD_DAP_IDENT,
         PNET_SUBMOD_DAP_INTERFACE_1_PORT_4_IDENT,
         &cfg_dap_data);
   }

   APP_LOG_DEBUG ("Done plugging DAP\n\n");
}

/**
 * Handle cyclic input- and output data for a subslot.
 *
 * Data is read and written using functions in the .c file,
 * which handles the data and update the physical input and outputs.
 *
 * @param subslot    InOut: Subslot reference
 * @param tag        In:    Application handle, here \a app_data_t pointer
 */
static void app_cyclic_data_callback (app_subslot_t * subslot, void * tag)
```

```
{
   app_data_t * app = (app_data_t *)tag;
   uint8_t indata_iops = PNET_IOXS_BAD;
   uint8_t indata_iocs = PNET_IOXS_BAD;
   uint8_t * indata;
   uint16_t indata_size = 0;
   bool outdata_updated;
   uint16_t outdata_length;
   uint8_t outdata_iops;
   uint8_t outdata_buf[20]; /* Todo: Remove temporary buffer */

   if (app == NULL)
   {
      APP_LOG_ERROR ("Application tag not set in subslot?\n");
      return;
   }

   if (subslot->slot_nbr != PNET_SLOT_DAP_IDENT && subslot->data_cfg.outsize > 0)
   {
      outdata_length = subslot->data_cfg.outsize;
      CC_ASSERT (outdata_length < sizeof (outdata_buf));

      /* Get output data from the PLC */
      (void)pnet_output_get_data_and_iops (
         app->net,
         APP_GSDML_API,
         subslot->slot_nbr,
         subslot->subslot_nbr,
         &outdata_updated,
         outdata_buf,
         &outdata_length,
         &outdata_iops);

      app_utils_print_ioxs_change (
         subslot,
         "Provider Status (IOPS)",
         subslot->outdata_iops,
         outdata_iops);
      subslot->outdata_iops = outdata_iops;

      if (outdata_length != subslot->data_cfg.outsize)
      {
         APP_LOG_ERROR ("Wrong outputdata length: %u\n", outdata_length);
         app_set_outputs_default_value();
      }
      else if (outdata_iops == PNET_IOXS_GOOD)
      {
         /* Application specific handling of the output data to a submodule.
            For the sample application, the data sets a LED. */
         (void)app_data_set_output_data (
            subslot->slot_nbr,
```

```
            subslot->subslot_nbr,
            subslot->submodule_id,
            outdata_buf,
            outdata_length);
      }
      else
      {
         app_set_outputs_default_value();
      }
   }

   if (subslot->slot_nbr != PNET_SLOT_DAP_IDENT && subslot->data_cfg.insize > 0)
   {
      /* Get application specific input data from a submodule (not DAP)
       *
       * For the sample application, the data includes a button
       * state and a counter value. */
      indata = app_data_get_input_data (
         subslot->slot_nbr,
         subslot->subslot_nbr,
         subslot->submodule_id,
         app->button1_pressed,
         &indata_size,
         &indata_iops);

      /* Send input data to the PLC */
      (void)pnet_input_set_data_and_iops (
         app->net,
         APP_GSDML_API,
         subslot->slot_nbr,
         subslot->subslot_nbr,
         indata,
         indata_size,
         indata_iops);

      (void)pnet_input_get_iocs (
         app->net,
         APP_GSDML_API,
         subslot->slot_nbr,
         subslot->subslot_nbr,
         &indata_iocs);

      app_utils_print_ioxs_change (
         subslot,
         "Consumer Status (IOCS)",
         subslot->indata_iocs,
         indata_iocs);
      subslot->indata_iocs = indata_iocs;
   }
}
```

```c
/**
 * Set initial input data, provider and consumer status for a subslot.
 *
 * @param app              In:    Application handle
 */
static int app_set_initial_data_and_ioxs (app_data_t * app)
{
   int ret;
   uint16_t slot;
   uint16_t subslot_index;
   const app_subslot_t * p_subslot;
   uint8_t * indata;
   uint16_t indata_size;
   uint8_t indata_iops;

   for (slot = 0; slot < PNET_MAX_SLOTS; slot++)
   {
      for (subslot_index = 0; subslot_index < PNET_MAX_SUBSLOTS;
           subslot_index++)
      {
         p_subslot = &app->main_api.slots[slot].subslots[subslot_index];
         if (p_subslot->plugged)
         {
            indata = NULL;
            indata_size = 0;
            indata_iops = PNET_IOXS_BAD;

            if (
               p_subslot->data_cfg.insize > 0 ||
               p_subslot->data_cfg.data_dir == PNET_DIR_NO_IO)
            {

               /* Get input data for submodule
                *
                * For the sample application data includes
                * includes button state and a counter value
                */
               if (
                  p_subslot->slot_nbr != PNET_SLOT_DAP_IDENT &&
                  p_subslot->data_cfg.insize > 0)
               {
                  indata = app_data_get_input_data (
                     p_subslot->slot_nbr,
                     p_subslot->subslot_nbr,
                     p_subslot->submodule_id,
                     app->button1_pressed,
                     &indata_size,
                     &indata_iops);
               }
               else if (p_subslot->slot_nbr == PNET_SLOT_DAP_IDENT)
               {
```

```
      indata_iops = PNET_IOXS_GOOD;
   }

   ret = pnet_input_set_data_and_iops (
      app->net,
      app->main_api.api_id,
      p_subslot->slot_nbr,
      p_subslot->subslot_nbr,
      indata,
      indata_size,
      indata_iops);

   /*
    * If a submodule is still plugged but not used in current AR,
    * setting the data and IOPS will fail.
    * This is not a problem.
    * Log message below will only be printed for active submodules.
    */
   if (ret == 0)
   {
      APP_LOG_DEBUG (
         "  Set initial input data and IOPS for slot %2u subslot "
         "%5u %9s size %3d \"%s\" \n",
         p_subslot->slot_nbr,
         p_subslot->subslot_nbr,
         app_utils_ioxs_to_string (indata_iops),
         p_subslot->data_cfg.insize,
         p_subslot->submodule_name);
   }
}

if (p_subslot->data_cfg.outsize > 0)
{
   ret = pnet_output_set_iocs (
      app->net,
      app->main_api.api_id,
      p_subslot->slot_nbr,
      p_subslot->subslot_nbr,
      PNET_IOXS_GOOD);

   if (ret == 0)
   {
      APP_LOG_DEBUG (
         "  Set initial output         IOCS for slot %2u subslot "
         "%5u %9s          \"%s\"\n",
         p_subslot->slot_nbr,
         p_subslot->subslot_nbr,
         app_utils_ioxs_to_string (PNET_IOXS_GOOD),
         p_subslot->submodule_name);
   }
}
```

```
         }
      }
   }
   return 0;
}

/**
 * Send and receive cyclic/process data for all subslots.
 *
 * Updates the data only on every APP_TICKS_UPDATE_DATA invocation
 *
 * @param app        In:    Application handle
 */
static void app_handle_cyclic_data (app_data_t * app)
{
   /* For the sample application cyclic data is updated
    * with a period defined by APP_TICKS_UPDATE_DATA
    */
   app->process_data_tick_counter++;
   if (app->process_data_tick_counter < APP_TICKS_UPDATE_DATA)
   {
      return;
   }
   app->process_data_tick_counter = 0;

   app_utils_cyclic_data_poll (&app->main_api);
}

/**
 * Set alarm, diagnostics and logbook entries.
 *
 * Alternates between these functions each time the button2 is pressed:
 *  - pnet_alarm_send_process_alarm()
 *  - pnet_diag_std_add()
 *  - pnet_set_redundancy_state()
 *  - pnet_set_state()
 *  - pnet_diag_std_update()
 *  - pnet_diag_usi_add()
 *  - pnet_diag_usi_update()
 *  - pnet_diag_usi_remove()
 *  - pnet_diag_std_remove()
 *  - pnet_create_log_book_entry()
 *  - pnet_ar_abort()
 *
 * Uses first 8-bit digital input module, if available.
 *
 * @param app             InOut:    Application handle
 */
static void app_handle_demo_pnet_api (app_data_t * app)
{
   uint16_t slot = 0;
```

```
bool found_inputsubslot = false;
uint16_t subslot_ix = 0;
const app_subslot_t * p_subslot = NULL;
pnet_pnio_status_t pnio_status = {0};
pnet_diag_source_t diag_source = {
   .api = APP_GSDML_API,
   .slot = 0,
   .subslot = 0,
   .ch = APP_DIAG_CHANNEL_NUMBER,
   .ch_grouping = PNET_DIAG_CH_INDIVIDUAL_CHANNEL,
   .ch_direction = APP_DIAG_CHANNEL_DIRECTION};

/* Loop though all subslots to find first digital 8-bit input subslot */
while (!found_inputsubslot && (slot < PNET_MAX_SLOTS))
{
   for (subslot_ix = 0;
        !found_inputsubslot && (subslot_ix < PNET_MAX_SUBSLOTS);
        subslot_ix++)
   {
      p_subslot = &app->main_api.slots[slot].subslots[subslot_ix];
      if (
         app_utils_subslot_is_input (p_subslot) &&
         (p_subslot->submodule_id == APP_GSDML_SUBMOD_ID_DIGITAL_IN ||
          p_subslot->submodule_id == APP_GSDML_SUBMOD_ID_DIGITAL_IN_OUT))
      {
         found_inputsubslot = true;
         break;
      }
   }
   if (!found_inputsubslot)
   {
      slot++;
   }
}
if (!found_inputsubslot)
{
   APP_LOG_DEBUG ("Did not find any input module in the slots. Skipping.\n");
   return;
}

diag_source.slot = slot;
diag_source.subslot = p_subslot->subslot_nbr;

switch (app->alarm_demo_state)
{
case APP_DEMO_STATE_ALARM_SEND:
   if (app->alarm_allowed == true && app_is_connected_to_controller (app))
   {
      app->alarm_payload[0]++;
      APP_LOG_INFO (
         "Sending process alarm from slot %u subslot %u USI %u to "
```

```
            "PLC. Payload: 0x%x\n",
            slot,
            p_subslot->subslot_nbr,
            APP_ALARM_USI,
            app->alarm_payload[0]);
         pnet_alarm_send_process_alarm (
            app->net,
            app->main_api.ar[0].arep,
            APP_GSDML_API,
            slot,
            p_subslot->subslot_nbr,
            APP_ALARM_USI,
            APP_GSDML_ALARM_PAYLOAD_SIZE,
            app->alarm_payload);
         app->alarm_allowed = false; /* Not allowed until ACK received */

         /* todo handle return code on pnet_alarm_send_process_alarm */
      }
      else
      {
         APP_LOG_WARNING (
            "Could not send process alarm, as alarm_allowed == false or "
            "no connection available\n");
      }
      break;

   case APP_DEMO_STATE_CYCLIC_REDUNDANT:
      APP_LOG_INFO (
         "Setting cyclic data to backup and to redundant. See Wireshark.\n");
      if (pnet_set_primary_state (app->net, false) != 0)
      {
         APP_LOG_WARNING ("   Could not set cyclic data state to backup.\n");
      }
      if (pnet_set_redundancy_state (app->net, true) != 0)
      {
         APP_LOG_WARNING ("   Could not set cyclic data state to redundant.\n");
      }
      break;

   case APP_DEMO_STATE_CYCLIC_NORMAL:
      APP_LOG_INFO (
         "Setting cyclic data back to primary and non-redundant. See "
         "Wireshark.\n");
      if (pnet_set_primary_state (app->net, true) != 0)
      {
         APP_LOG_ERROR ("   Could not set cyclic data state to primary.\n");
      }
      if (pnet_set_redundancy_state (app->net, false) != 0)
      {
         APP_LOG_ERROR (
            "   Could not set cyclic data state to non-redundant.\n");
```

```
      }
      break;

   case APP_DEMO_STATE_DIAG_STD_ADD:
      APP_LOG_INFO (
         "Adding standard diagnosis. Slot %u subslot %u channel %u Errortype "
         "%u\n",
         diag_source.slot,
         diag_source.subslot,
         diag_source.ch,
         APP_DIAG_CHANNEL_ERRORTYPE);
      (void)pnet_diag_std_add (
         app->net,
         &diag_source,
         APP_DIAG_CHANNEL_NUMBER_OF_BITS,
         APP_DIAG_CHANNEL_SEVERITY,
         APP_DIAG_CHANNEL_ERRORTYPE,
         APP_DIAG_CHANNEL_EXTENDED_ERRORTYPE,
         APP_DIAG_CHANNEL_ADDVALUE_A,
         APP_DIAG_CHANNEL_QUAL_SEVERITY);
      break;

   case APP_DEMO_STATE_DIAG_STD_UPDATE:
      APP_LOG_INFO (
         "Updating standard diagnosis. Slot %u subslot %u channel %u\n",
         diag_source.slot,
         diag_source.subslot,
         diag_source.ch);
      pnet_diag_std_update (
         app->net,
         &diag_source,
         APP_DIAG_CHANNEL_ERRORTYPE,
         APP_DIAG_CHANNEL_EXTENDED_ERRORTYPE,
         APP_DIAG_CHANNEL_ADDVALUE_B);
      break;

   case APP_DEMO_STATE_DIAG_STD_REMOVE:
      APP_LOG_INFO (
         "Removing standard diagnosis. Slot %u subslot %u channel %u\n",
         diag_source.slot,
         diag_source.subslot,
         diag_source.ch);
      pnet_diag_std_remove (
         app->net,
         &diag_source,
         APP_DIAG_CHANNEL_ERRORTYPE,
         APP_DIAG_CHANNEL_EXTENDED_ERRORTYPE);
      break;

   case APP_DEMO_STATE_DIAG_USI_ADD:
      APP_LOG_INFO (
```

```
            "Adding USI diagnosis. Slot %u subslot %u\n",
            slot,
            p_subslot->subslot_nbr);
         pnet_diag_usi_add (
            app->net,
            APP_GSDML_API,
            slot,
            p_subslot->subslot_nbr,
            APP_GSDML_DIAG_CUSTOM_USI,
            11,
            (uint8_t *)"diagdata_1");
         break;

      case APP_DEMO_STATE_DIAG_USI_UPDATE:
         APP_LOG_INFO (
            "Updating USI diagnosis. Slot %u subslot %u\n",
            slot,
            p_subslot->subslot_nbr);
         pnet_diag_usi_update (
            app->net,
            APP_GSDML_API,
            slot,
            p_subslot->subslot_nbr,
            APP_GSDML_DIAG_CUSTOM_USI,
            13,
            (uint8_t *)"diagdata_123");
         break;

      case APP_DEMO_STATE_DIAG_USI_REMOVE:
         APP_LOG_INFO (
            "Removing USI diagnosis. Slot %u subslot %u\n",
            slot,
            p_subslot->subslot_nbr);
         pnet_diag_usi_remove (
            app->net,
            APP_GSDML_API,
            slot,
            p_subslot->subslot_nbr,
            APP_GSDML_DIAG_CUSTOM_USI);
         break;

      case APP_DEMO_STATE_LOGBOOK_ENTRY:
         if (app_is_connected_to_controller (app))
         {
            APP_LOG_INFO (
               "Writing to logbook. Error_code1: %02X Error_code2: %02X  Entry "
               "detail: 0x%08X\n",
               APP_GSDML_LOGBOOK_ERROR_CODE_1,
               APP_GSDML_LOGBOOK_ERROR_CODE_2,
               APP_GSDML_LOGBOOK_ENTRY_DETAIL);
            pnio_status.error_code = APP_GSDML_LOGBOOK_ERROR_CODE;
```

```
            pnio_status.error_decode = APP_GSDML_LOGBOOK_ERROR_DECODE;
            pnio_status.error_code_1 = APP_GSDML_LOGBOOK_ERROR_CODE_1;
            pnio_status.error_code_2 = APP_GSDML_LOGBOOK_ERROR_CODE_2;
            pnet_create_log_book_entry (
               app->net,
               app->main_api.ar[0].arep,
               &pnio_status,
               APP_GSDML_LOGBOOK_ENTRY_DETAIL);
         }
         else
         {
            APP_LOG_WARNING (
               "Could not add logbook entry as no connection is available\n");
         }
         break;

      case APP_DEMO_STATE_ABORT_AR:
         if (app_is_connected_to_controller (app))
         {
            APP_LOG_INFO (
               "Sample app will disconnect and reconnect. Executing "
               "pnet_ar_abort()  AREP: %u\n",
               app->main_api.ar[0].arep);
            (void)pnet_ar_abort (app->net, app->main_api.ar[0].arep);
         }
         else
         {
            APP_LOG_WARNING (
               "Could not execute pnet_ar_abort(), as no connection is "
               "available\n");
         }
         break;
      }

      switch (app->alarm_demo_state)
      {
      case APP_DEMO_STATE_ALARM_SEND:
         app->alarm_demo_state = APP_DEMO_STATE_CYCLIC_REDUNDANT;
         break;
      case APP_DEMO_STATE_CYCLIC_REDUNDANT:
         app->alarm_demo_state = APP_DEMO_STATE_CYCLIC_NORMAL;
         break;
      case APP_DEMO_STATE_CYCLIC_NORMAL:
         app->alarm_demo_state = APP_DEMO_STATE_DIAG_STD_ADD;
         break;
      case APP_DEMO_STATE_DIAG_STD_ADD:
         app->alarm_demo_state = APP_DEMO_STATE_DIAG_STD_UPDATE;
         break;
      case APP_DEMO_STATE_DIAG_STD_UPDATE:
         app->alarm_demo_state = APP_DEMO_STATE_DIAG_USI_ADD;
         break;
```

```
   case APP_DEMO_STATE_DIAG_USI_ADD:
      app->alarm_demo_state = APP_DEMO_STATE_DIAG_USI_UPDATE;
      break;
   case APP_DEMO_STATE_DIAG_USI_UPDATE:
      app->alarm_demo_state = APP_DEMO_STATE_DIAG_USI_REMOVE;
      break;
   case APP_DEMO_STATE_DIAG_USI_REMOVE:
      app->alarm_demo_state = APP_DEMO_STATE_DIAG_STD_REMOVE;
      break;
   case APP_DEMO_STATE_DIAG_STD_REMOVE:
      app->alarm_demo_state = APP_DEMO_STATE_LOGBOOK_ENTRY;
      break;
   case APP_DEMO_STATE_LOGBOOK_ENTRY:
      app->alarm_demo_state = APP_DEMO_STATE_ABORT_AR;
      break;
   default:
   case APP_DEMO_STATE_ABORT_AR:
      app->alarm_demo_state = APP_DEMO_STATE_ALARM_SEND;
      break;
   }
}

void app_pnet_cfg_init_default (pnet_cfg_t * pnet_cfg)
{
   app_utils_pnet_cfg_init_default (pnet_cfg);

   pnet_cfg->state_cb = app_state_ind;
   pnet_cfg->connect_cb = app_connect_ind;
   pnet_cfg->release_cb = app_release_ind;
   pnet_cfg->dcontrol_cb = app_dcontrol_ind;
   pnet_cfg->ccontrol_cb = app_ccontrol_cnf;
   pnet_cfg->read_cb = app_read_ind;
   pnet_cfg->write_cb = app_write_ind;
   pnet_cfg->exp_module_cb = app_exp_module_ind;
   pnet_cfg->exp_submodule_cb = app_exp_submodule_ind;
   pnet_cfg->new_data_status_cb = app_new_data_status_ind;
   pnet_cfg->alarm_ind_cb = app_alarm_ind;
   pnet_cfg->alarm_cnf_cb = app_alarm_cnf;
   pnet_cfg->alarm_ack_cnf_cb = app_alarm_ack_cnf;
   pnet_cfg->reset_cb = app_reset_ind;
   pnet_cfg->signal_led_cb = app_signal_led_ind;
   pnet_cfg->sm_released_cb = app_sm_released_ind;

   pnet_cfg->cb_arg = (void *)&app_state;
}

/**
 * Read button states from operating system
 *
 * Actual reading is done every APP_TICKS_READ_BUTTONS invocation
 *
```

```
 * @param app             InOut:    Application handle
 */
static void update_button_states (app_data_t * app)
{
   app->buttons_tick_counter++;
   if (app->buttons_tick_counter > APP_TICKS_READ_BUTTONS)
   {
      app->button1_pressed = app_get_button (0);
      app->button2_pressed = app_get_button (1);
      app->buttons_tick_counter = 0;
   }
}

void pnet_set_ip (pnet_t * net){
   uint32_t iip=0;
   uint32_t imask=0;
   if (net != NULL){
      //memcpy(&iip,stHR.ip,4);
      //memcpy(&imask,stHR.mask,4);
      iip=stHR.ip[0]<<24|stHR.ip[1]<<16|stHR.ip[2]<<8|stHR.ip[3];
      imask=stHR.mask[0]<<24|stHR.mask[1]<<16|stHR.mask[2]<<8|stHR.mask[3];
      net->cmina_nonvolatile_dcp_ase.full_ip_suite.ip_suite.ip_addr=iip;
      net->cmina_nonvolatile_dcp_ase.full_ip_suite.ip_suite.ip_mask=imask;
      net->cmina_current_dcp_ase.full_ip_suite.ip_suite.ip_addr=iip;
      net->cmina_current_dcp_ase.full_ip_suite.ip_suite.ip_mask=imask;
      net->cmina_commit_ip_suite = true;
      os_event_set (net->pf_bg_worker.events, APP_EVENT_TIMER);
      pf_cmina_dcp_set_commit(net);
   }
}

/* Event handlers for the main loop. */
static void app_handle_event_timer (app_data_t * app)
{
   os_event_clr (app->main_events, APP_EVENT_TIMER);

   if(stHR.setIP>0){
      stHR.setIP=0;
      pnet_set_ip (app_get_pnet_instance (app));
   }
   if (app_is_connected_to_controller (app))
   {
      app_handle_cyclic_data (app);
   }

   /* Run alarm demo function if button2 is pressed */
   if ((app->button2_pressed == true) && (app->button2_pressed_previous ==
false)&& false)
   {
      app_handle_data_led_state(false);
      update_button_states (app);
```

```
      app_handle_demo_pnet_api (app);
   }
   app->button2_pressed_previous = app->button2_pressed;

   /* Run p-net stack */
   pnet_handle_periodic (app->net);
}

/**
 * Handle AR specific events.
 *
 * Calls the \a handler for each AR that has a pending \a event.
 *
 * @param app              InOut: Application handle
 * @param event            In:    Event
 * @param handler          In:    Event handling function
 */
static void app_handle_event_ar (
   app_data_t * app,
   uint32_t event,
   app_ar_event_handler_t handler)
{
   app_ar_iterator_t iter;
   app_ar_t * ar;

   os_event_clr (app->main_events, event);
   app_ar_iterator_init (&iter, &app->main_api);
   while (app_ar_iterator_next (&iter, &ar))
   {
      if (app_ar_event_clr (ar, event))
      {
         if (handler (app, app_ar_arep (ar)))
         {
            app_ar_iterator_delete_current (&iter);
         }
      }
   }
}

/* AR specific event handlers */

/**
 * Handle an AR connection PrmEnd.
 *
 * @param app              InOut: Application handle
 * @param arep             In:    Arep
 *
 * @return 0 to indicate that the arep should not be removed.
 */
static int app_ar_ready_for_data_handler (app_data_t * app, uint32_t arep)
{
```

```
   int err;

   APP_LOG_DEBUG (
      "Application will signal that it is ready for data, for AREP %u.\n",
      arep);
   /* When the PLC sends a confirmation to this message, the
      pnet_ccontrol_cnf() callback will be triggered.  */
   err = pnet_application_ready (app->net, arep);
   if (err)
   {
      APP_LOG_ERROR (
         "Error returned when application telling that it is ready for "
         "data. Have you set IOCS or IOPS for all subslots?\n");
   }
   return 0;
}

/**
 * Handle an AR alarm.
 *
 * @param app              InOut: Application handle
 * @param arep             In:    Arep
 *
 * @return 0 to indicate that the arep should not be removed.
 */
static int app_ar_alarm_handler (app_data_t * app, uint32_t arep)
{
   pnet_pnio_status_t pnio_status = {0, 0, 0, 0};
   int err;

   err = pnet_alarm_send_ack (app->net, arep, &app->alarm_arg, &pnio_status);
   if (err)
   {
      APP_LOG_DEBUG ("Error when sending alarm ACK. Error: %d\n", err);
   }
   else
   {
      APP_LOG_DEBUG ("Alarm ACK sent\n");
   }
   return 0;
}

/**
 * Handle a released submodule.
 *
 * @param app              InOut: Application handle
 * @param arep             In:    Arep
 *
 * @return 0 to indicate that the arep should not be removed.
 */
static int app_ar_sm_released_handler (app_data_t * app, uint32_t arep)
```

```
{
   /* Ideally, we should only set data for the indicated submodule,
      but setting everything works. */
   app_set_initial_data_and_ioxs (app);
   pnet_sm_released_cnf (app->net, arep);
   return 0;
}

/**
 * Handle an AR abort.
 *
 * @param app              InOut: Application handle
 * @param arep             In:    Arep
 *
 * @return 1 to indicate that the arep should be removed.
 */
static int app_ar_abort_handler (app_data_t * app, uint32_t arep)
{
   APP_LOG_DEBUG ("Connection (AREP %u) closed\n", arep);
   return 1;
}

/**
 * Wait for events generated elsewhere in this program.
 *
 * @param app              InOut: Application handle
 * @param mask             In:    Bitmask of events to wait for
 * @param flags            Out:   Bitmask of pending events
 */
static void app_event_wait (app_data_t * app, uint32_t mask, uint32_t * flags)
{
   os_event_wait (app->main_events, mask, flags, OS_WAIT_FOREVER);
}



void app_loop_forever (void * arg)
{
   app_data_t * app = (app_data_t *)arg;
   uint32_t mask = APP_EVENT_READY_FOR_DATA | APP_EVENT_TIMER |
                   APP_EVENT_ALARM | APP_EVENT_SM_RELEASED | APP_EVENT_ABORT;
   uint32_t flags = 0;
   NodeOutputLine NodeLine[2];
   struct gpiod_chip *chipLed;
   struct gpiod_chip *chipReset;
   chipLed = gpiod_chip_open("/dev/gpiochip2");
   chipReset = gpiod_chip_open("/dev/gpiochip0");
   NodeLine[0].Reset = gpiod_chip_get_line(chipReset, 30);
   NodeLine[1].Reset = gpiod_chip_get_line(chipReset, 31);
   NodeLine[0].Red = gpiod_chip_get_line(chipLed, 26);
   NodeLine[0].Blue = gpiod_chip_get_line(chipLed, 28);
```

```
    NodeLine[0].Green = gpiod_chip_get_line(chipLed, 27);
    NodeLine[1].Red = gpiod_chip_get_line(chipLed, 22);
    NodeLine[1].Blue = gpiod_chip_get_line(chipLed, 24);
    NodeLine[1].Green = gpiod_chip_get_line(chipLed, 23);



    NodeInit(&stNode[0],&NodeLine[0],"can0iobus",0);
    NodeInit(&stNode[1],&NodeLine[1],"can1iobus",1);
    initModbusTCP(app);
    app_set_led (APP_DATA_LED_ID, false);
    app_plug_dap (app, app->pnet_cfg->num_physical_ports);
    APP_LOG_INFO ("Waiting for PLC connect request\n\n");
    /* Main event loop */
    for (;;)
    {
       app_event_wait (app, mask, &flags);
       if (flags & APP_EVENT_READY_FOR_DATA)
       {
          app_handle_event_ar (
             app,
             APP_EVENT_READY_FOR_DATA,
             app_ar_ready_for_data_handler);
       }
       if (flags & APP_EVENT_ALARM)
       {
          app_handle_event_ar (app, APP_EVENT_ALARM, app_ar_alarm_handler);
       }
       if (flags & APP_EVENT_TIMER)
       {
          app_handle_event_timer (app);
       }
       if (flags & APP_EVENT_SM_RELEASED)
       {
          app_handle_event_ar (
             app,
             APP_EVENT_SM_RELEASED,
             app_ar_sm_released_handler);
       }
       if (flags & APP_EVENT_ABORT)
       {
          app_handle_event_ar (app, APP_EVENT_ABORT, app_ar_abort_handler);
          app->alarm_allowed = true;
       }
    }
}
```

2. Определения функций P-NET и необходимых структур данных

```
#ifndef SAMPLEAPP_COMMON_H
#define SAMPLEAPP_COMMON_H

#include "osal.h"
#include "pnal.h"
#include <pnet_api.h>

#ifdef __cplusplus
extern "C" {
#endif

#define APP_TICK_INTERVAL_US 1000 /* 1 ms */

/* Thread configuration for targets where sample
 * event loop is run in a separate thread (not main).
 * This applies for linux sample app implementation.
 */
#define APP_MAIN_THREAD_PRIORITY  15
#define APP_MAIN_THREAD_STACKSIZE 2*4096 /* bytes */

#define APP_DATA_LED_ID            1
#define APP_PROFINET_SIGNAL_LED_ID 2

#define APP_TICKS_READ_BUTTONS 10
#define APP_TICKS_UPDATE_DATA  100

/** HW Offload configuration. */
typedef enum
{
   MODE_HW_OFFLOAD_NONE = 0,
   MODE_HW_OFFLOAD_CPU,
   MODE_HW_OFFLOAD_FULL,
} app_mode_t;

/** Command line arguments for sample application */
typedef struct app_args
{
   char path_button1[PNET_MAX_FILE_FULLPATH_SIZE]; /** Terminated string */
   char path_button2[PNET_MAX_FILE_FULLPATH_SIZE]; /** Terminated string */
   char path_storage_directory[PNET_MAX_DIRECTORYPATH_SIZE]; /** Terminated */
   char station_name[PNET_STATION_NAME_MAX_SIZE]; /** Terminated string */
   char eth_interfaces
      [PNET_INTERFACE_NAME_MAX_SIZE * (PNET_MAX_PHYSICAL_PORTS + 1) +
       PNET_MAX_PHYSICAL_PORTS]; /** Terminated string */
   int verbosity;
   int show;
   bool factory_reset;
```

```
   bool remove_files;
   app_mode_t mode;
} app_args_t;

typedef enum
{
   RUN_IN_SEPARATE_THREAD,
   RUN_IN_MAIN_THREAD
} app_run_in_separate_task_t;

typedef struct app_data_t app_data_t;

/**
 * AR specific event handler type.
 *
 * Handles an AR specific event.
 *
 * @param app          InOut: Application handle
 * @param arep         In:    Arep of the AR.
 *
 * @return 0 to indicate that the arep should be kept
 *         1 to indicate that the arep should be forgotten
 */
typedef int (*app_ar_event_handler_t) (app_data_t * app, uint32_t arep);

/** Partially initialise config values, and use proper callbacks
 *
 * @param pnet_cfg     Out:   Configuration to be updated
 */
void app_pnet_cfg_init_default (pnet_cfg_t * pnet_cfg);

/**
 * Initialize P-Net stack and application.
 *
 * The \a pnet_cfg argument shall have been initialized using
 * \a app_pnet_cfg_init_default() before this function is
 * called.
 *
 * @param pnet_cfg               In:    P-Net configuration
 * @param app_args               In:    Application arguments
 * @return Application handle, NULL on error
 */
app_data_t * app_init (const pnet_cfg_t * pnet_cfg, const app_args_t * app_args);

/**
 * Start application main loop
 *
 * Application must have been initialized using \a app_init() before
 * this function is called.
 *
 * If \a task_config parameters is set to RUN_IN_SEPARATE_THREAD a
```

```
 * thread execution the \a app_loop_forever() function is started.
 * If task_config is set to RUN_IN_MAIN_THREAD no such thread is
 * started and the caller must call the \a app_loop_forever() after
 * calling this function.
 *
 * RUN_IN_MAIN_THREAD is intended for rt-kernel targets.
 * RUN_IN_SEPARATE_THREAD is intended for linux targets.
 *
 * @param app                 In:    Application handle
 * @param task_config         In:    Defines if stack and application
 *                                   is run in main or separate task.
 * @return 0 on success, -1 on error
 */
int app_start (app_data_t * app, app_run_in_separate_task_t task_config);

/**
 * Application task definition. Handles events in eternal loop.
 *
 * @param arg                 In: Application handle
 */
void app_loop_forever (void * arg);

/**
 * Get P-Net instance from application
 *
 * @param app                 In:    Application handle
 * @return P-Net instance, NULL on failure
 */
pnet_t * app_get_pnet_instance (app_data_t * app);

/**
 * Set LED state
 * Hardware specific. Implemented in sample app main file for
 * each supported platform.
 *
 * @param id                In:    LED number, starting from 0.
 * @param led_state         In:    LED state. Use true for on and false for off.
 */
void app_set_led (uint16_t id, bool led_state);

/**
 * Read button state
 *
 * Hardware specific. Implemented in sample app main file for
 * each supported platform.
 *
 * @param id                In:    Button number, starting from 0.
 * @return  true if button is pressed, false if not
 */
bool app_get_button (uint16_t id);
```

```
#ifdef __cplusplus
}
#endif

#endif /* SAMPLEAPP_COMMON_H */
```

3. Управление CAN устройствами

```
#include "pnal.h"

#include "pnet_options.h"
#include "options.h"
#include "osal_log.h"

#include <net/ethernet.h>
#include <net/if.h>
#include <netpacket/packet.h>
#include <sys/ioctl.h>
#include <linux/can.h>
#include <linux/can/raw.h>

#include <stdlib.h>
#include <string.h>
#include "app_can.h"
#include "app_ecc.h"
#include "osal.h"
#include <time.h>
#include <gpiod.h>


HR_t stHR;
CanNode_t stNode[2];

#define ST_ADDR_START 0
#define ST_ADDR_NULL 40
#define ST_LINK_RESET 80
#define ST_ADDR_SET 100
#define ST_ADDR_WAIT 120
#define ST_ADDR_OK 140
#define ST_MODE_OPER 160
#define ST_MODE_OPER_OK 180
#define ST_IDLE 200

#define GPIOD_CONSUMER "COMITAS_CAN_GATE"
bool xAddrInit;

void NodeInit(CanNode_t* sNode,NodeOutputLine* Line, char sCan[10],unsigned short
num)
```

```
{
    sNode->myLine = Line;
    sNode->id = num;
    gpiod_line_request_output(sNode->myLine->Reset, GPIOD_CONSUMER, 0);
    gpiod_line_request_output(sNode->myLine->Red, GPIOD_CONSUMER, 1);
    gpiod_line_request_output(sNode->myLine->Green, GPIOD_CONSUMER, 1);
    gpiod_line_request_output(sNode->myLine->Blue, GPIOD_CONSUMER, 1);
    memcpy(sNode->cName,sCan,10);
    InitCAN(sNode);
    sNode->xAddrInit = false;
    sNode->addr_iter=ST_ADDR_START;
    sNode->addr=0;
}


uint32_t getTick(void) {
    struct timespec ts;
    unsigned theTick = 0U;
    clock_gettime( CLOCK_MONOTONIC, &ts );
    theTick  = ts.tv_nsec / 1000000;
    theTick += ts.tv_sec * 1000;
    return theTick;
}


void reAddressing(CanNode_t* sNode){

    switch (sNode->addr_iter)
    {
    case ST_ADDR_START:
        //printf(" - Start Addr %s\n",sNode->cName);
        LOG_FATAL (
         PNET_LOG,
         "MMNG(%d): %d - Start Addr %s\n",
         __LINE__, getTick(), sNode->cName);

        sNode->addr = 0;
        sNode->addr_iter++;
        gpiod_line_set_value(sNode->myLine->Reset, 0);
        gpiod_line_set_value(sNode->myLine->Red, 1);
        gpiod_line_set_value(sNode->myLine->Green, 1);
        gpiod_line_set_value(sNode->myLine->Blue, 0);

        sNode->xStopSend = true;
        ChangeOperationalMode(sNode,STOPPED);
        break;
    case ST_ADDR_NULL:
        LOG_FATAL (
         PNET_LOG,
         "MMNG(%d): %d - Start Addr NULL %s\n",
         __LINE__, getTick(), sNode->cName);
```

```
        sNode->addr_iter++;
        SetAddress(sNode,0);
        break;
    case ST_LINK_RESET:
        LOG_FATAL (
         PNET_LOG,
         "MMNG(%d): %d - Reset Link %s\n",
         __LINE__, getTick(), sNode->cName);

        sNode->addr_iter++;
        for(int i = 0;i < 32;i++){
            stHR.MotorStatus[sNode->id][i].st.xLink = 0;
        }
        gpiod_line_set_value(sNode->myLine->Reset, 1);
        break;
    case ST_ADDR_SET:
        LOG_FATAL (
         PNET_LOG,
         "MMNG(%d): %d - Set Addr %d %s\n",
         __LINE__, getTick(),sNode->addr, sNode->cName);

        sNode->addr_iter++;
        gpiod_line_set_value(sNode->myLine->Blue, sNode->addr&0x01);
        SetAddress(sNode,sNode->addr);
        sNode->addr++;
        if (sNode->addr>=32){
            sNode->addr_iter = ST_ADDR_OK;
        }
        break;
    case ST_ADDR_WAIT:
        sNode->addr_iter = ST_ADDR_SET;
        break;
    case ST_ADDR_OK:
        LOG_FATAL (
         PNET_LOG,
         "MMNG(%d): %d - Set Addr END %s\n",
         __LINE__, getTick(), sNode->cName);

        sNode->addr_iter++;
        gpiod_line_set_value(sNode->myLine->Blue, 0);
        gpiod_line_set_value(sNode->myLine->Reset, 0);
        break;
    case ST_MODE_OPER:
        LOG_FATAL (
         PNET_LOG,
         "MMNG(%d): %d - Set OPER MODE %s\n",
         __LINE__, getTick(), sNode->cName);

        sNode->addr_iter++;
        ChangeOperationalMode(sNode,OPERATIONAL);
        break;
```

```
    case ST_MODE_OPER_OK:
        LOG_FATAL (
         PNET_LOG,
         "MMNG(%d): %d - Addressing Ok %s\n",
         __LINE__, getTick(), sNode->cName);

        sNode->addr_iter++;
        sNode->xStopSend = false;
        gpiod_line_set_value(sNode->myLine->Blue, 1);
        gpiod_line_set_value(sNode->myLine->Green, 0);
        gpiod_line_set_value(sNode->myLine->Red, 1);
        sNode->xAddrInit = true;
        sNode->xFindNotInit = false;

        sNode->addr_iter=ST_ADDR_START;
        sNode->addr=0;
        break;
    case ST_IDLE:
        break;
    default:
        sNode->addr_iter++;
        break;
    }
}

void SetAddress(CanNode_t* sNode, uint8_t addr){
    struct can_frame frame;
    frame.can_id  = 0x600;
    frame.can_dlc = 8;
    frame.data[0] = 0x2F;
    frame.data[1] = 0x00;
    frame.data[2] = 0x30;
    frame.data[3] = 0x01;
    frame.data[4] = addr;
    frame.data[5] = 0x00;
    frame.data[6] = 0x00;
    frame.data[7] = 0x00;

    SendMsg(sNode, frame);
}

void ChangeOperationalMode(CanNode_t* sNode, enum NMT_STATES state){
    struct can_frame frame;
    frame.can_id = 0 ;
    frame.can_dlc = 2;
    frame.data[0] = state;
    frame.data[1] = 0;
    frame.data[2] = 0;
    frame.data[3] = 0;
    frame.data[4] = 0;
    frame.data[5] = 0;
```

```
    frame.data[6] = 0;
    frame.data[7] = 0;

    SendMsg(sNode, frame);

}


void SendParam(CanNode_t* sNode){
    struct can_frame frame;
    if((sNode->last_sendParam + 500)< getTick() ){
        for(int i = 0;i < 32;i++){
            float fspeed=stHR.MotorParam[sNode->id][i].rSpeed*1.5;
            frame.can_id = 0x201+i;
            frame.can_dlc = 8;
            frame.data[0] = (unsigned char)(stHR.MotorParam[sNode-
>id][i].ucMotorType&0xFF);
            frame.data[1] = 0;
            memcpy(&frame.data[2],&fspeed,4);//&_motors[i]-
>stUstParams.rSpeed,4);
            frame.data[6] = (unsigned char)(stHR.MotorParam[sNode-
>id][i].ucRampUp&0xFF);
            frame.data[7] = (unsigned char)(stHR.MotorParam[sNode-
>id][i].ucRampDown&0xFF);
            SendQueueMsg(sNode,frame);
        }
        sNode->last_sendParam = getTick();
    }
}

void SendCMD(CanNode_t* sNode){
    struct can_frame frame;
    uint8_t MotorCmd[8] = {0}; //ID range from 201 to 204
    uint8_t MotorDir[8] = {0}; //ID range from 201 to 204
    for(int i = 0;i < 32;i++){
        unsigned char motor_byte = i / 8;
        unsigned char motor_bit = i % 8;
        //if ((_motors[i]->stStatus.xLink)&&(_motors[i]->xCmd))
        //if(hr_param.xHMICmd & 0x01)
        {
            if ((stHR.HMIMotorCMD[sNode->id][i]&0x02))
                MotorDir[motor_byte] |= 0x01 << motor_bit;
            else
                MotorDir[motor_byte] &= ~(0x01 << motor_bit);
            if ((stHR.HMIMotorCMD[sNode->id][i]&0x01))
                MotorCmd[motor_byte] |= 0x01 << motor_bit;
            else
                MotorCmd[motor_byte] &= ~(0x01 << motor_bit);
        }
    }
```

```
    if(!sNode->xStopSend)
    {

        frame.can_id = 0x300;
        frame.can_dlc = 8;


        frame.data[0] = MotorCmd[0];
        frame.data[1] = MotorCmd[1];
        frame.data[2] = MotorCmd[2];
        frame.data[3] = MotorCmd[3];
        frame.data[4] = MotorDir[0];
        frame.data[5] = MotorDir[1];
        frame.data[6] = MotorDir[2];
        frame.data[7] = MotorDir[3];
        sNode->Msg5ms = frame;
    }else{

        frame.can_id = 0x000;
        frame.can_dlc = 0;
        sNode->Msg5ms = frame;
    }
}

void ECCUpdate(CanNode_t* sNode)
{
    uint32_t dw=getTick();

    if((!sNode->xFindNotInit)&&((dw-sNode->lastFindNotInit)>2000)){
        sNode->xFindNotInit = false;
    }

    if(!sNode->xAddrInit){
        struct can_frame frame;
        frame.can_id = 0x000;
        frame.can_dlc = 0;
        sNode->Msg5ms = frame;
        reAddressing(sNode);
    }else{

        if(sNode->xFindNotInit){
            gpiod_line_set_value(sNode->myLine->Red, 0);
            gpiod_line_set_value(sNode->myLine->Blue, 1);
            gpiod_line_set_value(sNode->myLine->Green, 1);
        }else{
            gpiod_line_set_value(sNode->myLine->Red, 1);
            gpiod_line_set_value(sNode->myLine->Blue, 1);
            gpiod_line_set_value(sNode->myLine->Green, 0);
        }
            SendCMD(sNode);
            SendParam(sNode);
```

```
    }
    for(int i = 0;i < 32;i++){
        if(stHR.MotorParam[sNode->id][i].rSpeed<50)stHR.MotorParam[sNode-
>id][i].rSpeed =50;
        if(stHR.MotorParam[sNode->id][i].rSpeed>1000)stHR.MotorParam[sNode-
>id][i].rSpeed =1000;
        if(stHR.MotorParam[sNode->id][i].ucRampUp<10)stHR.MotorParam[sNode-
>id][i].ucRampUp =10;
        if(stHR.MotorParam[sNode->id][i].ucRampDown<10)stHR.MotorParam[sNode-
>id][i].ucRampDown =10;

        if(( sNode->last_link[i] + 2000 ) < getTick() ){
            stHR.MotorStatus[sNode->id][i].st.xLink = 1;
        }else{
            stHR.MotorStatus[sNode->id][i].st.xLink = 0;
        }
    }
}
```

4. Определения структур и данных CAN

```
#ifndef APP_ECC_H
#define APP_ECC_H


#ifdef __cplusplus
extern "C" {
#endif

#include <stdint.h>
#include <linux/can.h>
#include <linux/can/raw.h>


enum NMT_STATES
{
    INITIALISING = 129,
    STOPPED = 2,
    OPERATIONAL = 1,
    SLEEP = 80,
    STANDBY = 96,
    PRE_OPERATIONAL = 128
```

```
};

typedef struct {
        struct can_frame f[50];  // указатель на данные
        int w_ptr;        // указатель на нижнюю границу
        int r_ptr;        // указатель на нижнюю границу
        int max;       // указатель на верхнюю границу
}queue ;

typedef struct {
  int8_t Motor_Voltage;
  int8_t Logic_Voltage;
  int8_t Source_Voltage;
  int8_t Voltage_10V;
  int8_t Voltage_18V;
  int8_t Hall_Voltage;
}ADC1_t;

typedef struct {
  int8_t Sensor_Voltage;
  int8_t Motor_Current;
  int8_t CPU_Temperature;
  int8_t unused;
  uint16_t uiCustomADC;
} ADC2_t;

typedef struct {
  uint16_t rSpeed;
  uint8_t ucMotorType;
  uint8_t ucRampUp;
  uint8_t ucRampDown;
  uint8_t OvercurrentLimit;
} CDatasheet_t;

typedef struct {
  uint16_t unused0:1;
  uint16_t unused1:1;
  uint16_t unused2:1;
  uint16_t unused3:1;
  uint16_t unused4:1;
  uint16_t unused5:1;
  uint16_t unused6:1;
  uint16_t unused7:1;
  uint16_t unused8:1;
  uint16_t unused9:1;
  uint16_t unused10:1;
  uint16_t unused11:1;
  uint16_t unused12:1;
  uint16_t unused13:1;
  uint16_t xConfig:1;
  uint16_t xLink:1;
```

```
} CStatus_t;

union uStatus_t{
  CStatus_t st;
  unsigned short b;
};

typedef struct {
  int16_t Vmain;
  int16_t V5v;
  int16_t VCh1;
  int16_t ICh1;
  int16_t VCh2;
  int16_t ICh2;
} ADCMain_t;

typedef struct {
    union uStatus_t MotorStatus[2][32];

    u_int16_t MotorCMD[2][32];

    ADC1_t MotorADC1[2][32];

    ADC2_t MotorADC2[2][32];

    CDatasheet_t MotorParam[2][32];

    u_int16_t HMIMotorCMD[2][32];

    uint8_t N1_SelectADC[2][32];

    uint8_t ip[4];
    uint8_t mask[4];
    ADCMain_t adc;
    uint16_t Status;
    u_int16_t xHMICmd;
    CDatasheet_t CurrentParam[2][32];
    uint8_t setIP;

} HR_t;


typedef struct NodeOutputLine
{
    struct gpiod_line *Reset;
    struct gpiod_line *Red;
    struct gpiod_line *Green;
    struct gpiod_line *Blue;
} NodeOutputLine;

typedef struct {
```

```
    NodeOutputLine *myLine;
    int SocketFD;
    bool xStopSend;
    bool xStopRead;
    bool m_IsRunning;
    bool xFindNotInit;
    bool xAddrInit;
    char cName[10];
    unsigned short id;
    unsigned short SendCnt;
    unsigned short RecvCnt;
    unsigned short Status;
    unsigned short addr_iter;
    unsigned short addr;
    unsigned short can_err;
    unsigned long lastFindNotInit;
    unsigned long last_sendParam;
    unsigned long last_link[32];
    struct can_frame Msg5ms;
    queue q;
} CanNode_t;

extern HR_t stHR;
extern CanNode_t stNode[2];
void ECCUpdate(CanNode_t* sNode);

void NodeInit(CanNode_t* sNode,NodeOutputLine* Line, char sCan[10],unsigned short
num);
void SetAddress(CanNode_t* sNode, uint8_t addr);
void ChangeOperationalMode(CanNode_t* sNode, enum NMT_STATES state);
#ifdef __cplusplus
}
#endif

#endif /* APP_LOG_H */
```

5. Функционал обмена информацией с веб интерфейсом и управления

```
#include "pnal.h"

#include "pnet_options.h"
#include "options.h"
#include "osal_log.h"

#include <net/ethernet.h>
```

```c
#include <net/if.h>
#include <netpacket/packet.h>
#include <sys/ioctl.h>
#include <errno.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include "app_can.h"
#include "app_web.h"
#include "config.h"
#include "modbus-private.h"
#include "modbus-tcp-private.h"
#include "modbus-tcp.h"
#include "modbus-version.h"
#include "modbus.h"
#include "app_ecc.h"
#include "config.h"
#include <mntent.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

//#include "memmgr.h"
//
//struct pnal_eth_handle
//{
//   pnal_eth_callback_t * callback;
//   void * arg;
//   int socket;
//   os_thread_t * thread;
//};
//
///**
// * @internal
// * Run a thread that listens to incoming raw Ethernet sockets.
// * Delegate the actual work to thread_arg->callback
// *
// * This is a function to be passed into os_thread_create()
// * Do not change the argument types.
// *
// * @param thread_arg     InOut: Will be converted to pnal_eth_handle_t
// */


#define APP_WEB_THREAD_PRIORITY  15
#define APP_WEB_THREAD_STACKSIZE 4096 /* bytes */

modbus_mapping_t temp;
FILE *file_ptr;
int adc_ptr;
```

```
const char *cfgCAN0Name[]={
    "CAN0Node1",
    "CAN0Node2",
    "CAN0Node3",
    "CAN0Node4",
    "CAN0Node5",
    "CAN0Node6",
    "CAN0Node7",
    "CAN0Node8",
    "CAN0Node9",
    "CAN0Node10",
    "CAN0Node11",
    "CAN0Node12",
    "CAN0Node13",
    "CAN0Node14",
    "CAN0Node15",
    "CAN0Node16",
    "CAN0Node17",
    "CAN0Node18",
    "CAN0Node19",
    "CAN0Node20",
    "CAN0Node21",
    "CAN0Node22",
    "CAN0Node23",
    "CAN0Node24",
    "CAN0Node25",
    "CAN0Node26",
    "CAN0Node27",
    "CAN0Node28",
    "CAN0Node29",
    "CAN0Node30",
    "CAN0Node31",
    "CAN0Node32"};

const char *cfgCAN1Name[]={
    "CAN1Node1",
    "CAN1Node2",
    "CAN1Node3",
    "CAN1Node4",
    "CAN1Node5",
    "CAN1Node6",
    "CAN1Node7",
    "CAN1Node8",
    "CAN1Node9",
    "CAN1Node10",
    "CAN1Node11",
    "CAN1Node12",
    "CAN1Node13",
    "CAN1Node14",
    "CAN1Node15",
```

```
    "CAN1Node16",
    "CAN1Node17",
    "CAN1Node18",
    "CAN1Node19",
    "CAN1Node20",
    "CAN1Node21",
    "CAN1Node22",
    "CAN1Node23",
    "CAN1Node24",
    "CAN1Node25",
    "CAN1Node26",
    "CAN1Node27",
    "CAN1Node28",
    "CAN1Node29",
    "CAN1Node30",
    "CAN1Node31",
    "CAN1Node32"};

bool mounted;
bool loaded;
bool init;
bool sdPresent;
void Save_Cfg(void)
{
    printf("Save_Cfg\n");
    CONFIG *cfg = config_open(null);
    if(config_load(cfg, "/mnt/Comitas.cfg")){

    }

    for(int i=0;i<32;i++){

            stHR.MotorStatus[0][i].st.xConfig = stHR.MotorStatus[0][i].st.xLink;
            config_set_value_int(cfg,cfgCAN0Name[i],"isConfig",stHR.MotorStatus[0
][i].st.xConfig^0x01);
            config_set_value_int(cfg,cfgCAN0Name[i],"MotorType",stHR.MotorParam[0
][i].ucMotorType);
            config_set_value_int(cfg,cfgCAN0Name[i],"Speed",stHR.MotorParam[0][i]
.rSpeed);
            config_set_value_int(cfg,cfgCAN0Name[i],"RampUp",stHR.MotorParam[0][i
].ucRampUp);
            config_set_value_int(cfg,cfgCAN0Name[i],"RampDown",stHR.MotorParam[0]
[i].ucRampDown);
            config_set_value_int(cfg,cfgCAN0Name[i],"OvercurrentLimit",stHR.Motor
Param[0][i].OvercurrentLimit);

            stHR.MotorStatus[1][i].st.xConfig = stHR.MotorStatus[1][i].st.xLink;
            config_set_value_int(cfg,cfgCAN1Name[i],"isConfig",stHR.MotorStatus[1
][i].st.xConfig^0x01);
            config_set_value_int(cfg,cfgCAN1Name[i],"MotorType",stHR.MotorParam[1
][i].ucMotorType);
```

```
            config_set_value_int(cfg,cfgCAN1Name[i],"Speed",stHR.MotorParam[1][i]
.rSpeed);
            config_set_value_int(cfg,cfgCAN1Name[i],"RampUp",stHR.MotorParam[1][i
].ucRampUp);
            config_set_value_int(cfg,cfgCAN1Name[i],"RampDown",stHR.MotorParam[1]
[i].ucRampDown);
            config_set_value_int(cfg,cfgCAN1Name[i],"OvercurrentLimit",stHR.Motor
Param[1][i].OvercurrentLimit);
    }

    config_set_value_int(cfg, "Net", "IP1", stHR.ip[0]);
    config_set_value_int(cfg, "Net", "IP2", stHR.ip[1]);
    config_set_value_int(cfg, "Net", "IP3", stHR.ip[2]);
    config_set_value_int(cfg, "Net", "IP4", stHR.ip[3]);
    config_set_value_int(cfg, "Net", "Mask1",stHR.mask[0]);
    config_set_value_int(cfg, "Net", "Mask2",stHR.mask[1]);
    config_set_value_int(cfg, "Net", "Mask3",stHR.mask[2]);
    config_set_value_int(cfg, "Net", "Mask4",stHR.mask[3]);

    config_save(cfg, "/mnt/Comitas.cfg");
    config_close(cfg);
}
void Default_Cfg(void)
{

}
void Read_Cfg(void)
{
    bool ReadOk = false;
    if(mounted){
        CONFIG *cfg = config_open(null);
        if(config_load(cfg, "/mnt/Comitas.cfg")){
            printf("Read_Cfg\n");
            //int nbr_sections = config_get_nbr_sections(cfg);
            //const char **section_names = config_get_sections(cfg);
            int i;
            /*const char *comments = config_get_comment(cfg);
            if(comments != null)
            {
                printf("#%s\n", comments);
            }*/
            stHR.ip[0] = config_get_value_int(cfg, "Net", "IP1", 192);
            stHR.ip[1] = config_get_value_int(cfg, "Net", "IP2", 168);
            stHR.ip[2] = config_get_value_int(cfg, "Net", "IP3", 0);
            stHR.ip[3] = config_get_value_int(cfg, "Net", "IP4", 10);
            stHR.mask[0] = config_get_value_int(cfg, "Net", "Mask1",255);
            stHR.mask[1] = config_get_value_int(cfg, "Net", "Mask2",255);
            stHR.mask[2] = config_get_value_int(cfg, "Net", "Mask3",255);
            stHR.mask[3] = config_get_value_int(cfg, "Net", "Mask4",0);
            for(i=0; i<32; i++)
            {
```

```
                stHR.MotorStatus[0][i].st.xConfig =
(config_get_value_int(cfg,cfgCAN0Name[i],"isConfig",0))^0x01;
                stHR.MotorParam[0][i].ucMotorType =
(u_int8_t)config_get_value_int(cfg,cfgCAN0Name[i],"MotorType",1);
                stHR.MotorParam[0][i].rSpeed =
(u_int16_t)config_get_value_int(cfg,cfgCAN0Name[i],"Speed",500);
                stHR.MotorParam[0][i].ucRampUp =
(u_int8_t)config_get_value_int(cfg,cfgCAN0Name[i],"RampUp",100);
                stHR.MotorParam[0][i].ucRampDown =
(u_int8_t)config_get_value_int(cfg,cfgCAN0Name[i],"RampDown",60);
                stHR.MotorParam[0][i].OvercurrentLimit =
(u_int8_t)config_get_value_int(cfg,cfgCAN0Name[i],"OvercurrentLimit",3);

                stHR.MotorStatus[1][i].st.xConfig =
(config_get_value_int(cfg,cfgCAN1Name[i],"isConfig",0))^0x01;
                stHR.MotorParam[1][i].ucMotorType =
(u_int8_t)config_get_value_int(cfg,cfgCAN1Name[i],"MotorType",1);
                stHR.MotorParam[1][i].rSpeed =
(u_int16_t)config_get_value_int(cfg,cfgCAN1Name[i],"Speed",500);
                stHR.MotorParam[1][i].ucRampUp =
(u_int8_t)config_get_value_int(cfg,cfgCAN1Name[i],"RampUp",100);
                stHR.MotorParam[1][i].ucRampDown =
(u_int8_t)config_get_value_int(cfg,cfgCAN1Name[i],"RampDown",60);
                stHR.MotorParam[1][i].OvercurrentLimit =
(u_int8_t)config_get_value_int(cfg,cfgCAN1Name[i],"OvercurrentLimit",3);
            }
            ReadOk=true;
            stHR.setIP = 1;
            stHR.Status &= ~0x02;
        }else{

            stHR.Status |= 0x02;
        }

        config_close(cfg);
    }
    if((!ReadOk)&&(!init)){
        init=true;
        printf("cfg no found!\n");
        stHR.ip[0] = 192;
        stHR.ip[1] = 168;
        stHR.ip[2] = 0;
        stHR.ip[3] = 10;
        stHR.mask[0] = 255;
        stHR.mask[1] = 255;
        stHR.mask[2] = 255;
        stHR.mask[3] = 0;
        for (int i = 0; i < 32; i++)
        {
            stHR.MotorParam[0][i].rSpeed = 500; //Float param
            stHR.MotorParam[0][i].ucRampDown = 100; //Float param
```

```
            stHR.MotorParam[0][i].ucRampUp = 60; //Float param
            stHR.MotorParam[0][i].ucMotorType = 1; //Float param
            stHR.MotorParam[1][i].rSpeed = 500; //Float param
            stHR.MotorParam[1][i].ucRampDown = 100; //Float param
            stHR.MotorParam[1][i].ucRampUp = 60; //Float param
            stHR.MotorParam[1][i].ucMotorType = 1; //Float param
        }
    }
    loaded = true;

}



void Set_IP()
{
    stHR.setIP = 1;
}
int is_mounted (char * dev_path) {
    FILE * mtab = NULL;
    struct mntent * part = NULL;
    int is_mounted = 0;
    if ( ( mtab = setmntent ("/etc/mtab", "r") ) != NULL) {
        while ( ( part = getmntent ( mtab) ) != NULL) {
            if ( ( part->mnt_fsname != NULL )
                && ( strcmp ( part->mnt_fsname, dev_path ) ) == 0 ) {
                is_mounted = 1;
            }
        }
        endmntent ( mtab);
    }
    return is_mounted;
}
void adc_get(void){


    char read_buf[512];
    memset(read_buf,0,sizeof(read_buf));

    if (adc_ptr<=0) {
        adc_ptr = open("/dev/iio:device0",O_RDWR|O_NOCTTY|O_NDELAY);
        printf("open adc err \n");
    }else{
        if((read(adc_ptr,read_buf,5))<=0)
        {
            printf("adc read err \n");
            close(adc_ptr);
            adc_ptr=0;
        }else
        {
            printf("adc value is :%s \n",read_buf);
```

```
        }

    }
}
void checkSD(void) {
    file_ptr = fopen("/dev/mmcblk0p1", "r");

    if (file_ptr) {
        fclose(file_ptr);
        if(!sdPresent){
            printf("SD Insert!\n");
            mounted = false;
        }
        if (is_mounted("/dev/mmcblk0p1")) {
            if(!sdPresent){
                int rez = system("umount /dev/mmcblk0p1");
                if(rez){

                }
            }else{
                stHR.Status |= 1;
                mounted = true;
            }
        }else{
            loaded = false;
            mounted = false;
            int rez = system("mount /dev/mmcblk0p1 /mnt");
            if(rez){

            }
            printf("Try mount SD!\n");
            stHR.Status = 2;
        }
        sdPresent = true;
    }else{
        if(sdPresent){
            int rez = system("umount -l /mnt/");
            if(rez){

            }
            printf("SD Extract!\n");
        }
        loaded = false;
        stHR.Status = 0;
        sdPresent = false;
        mounted = false;
    }
}

static void os_web_task (void * thread_arg)
```

```
{
    //app_data_t * app = (app_data_t *)thread_arg;

    modbus_t *ctx;
    modbus_mapping_t *mb_mapping;
    int rc;
    int s = -1;
    uint8_t *query;
    //int header_length;
    ctx = modbus_new_tcp(NULL, 502);
    query = malloc(MODBUS_TCP_MAX_ADU_LENGTH);
    //header_length = modbus_get_header_length(ctx);
    modbus_set_debug(ctx, FALSE);
    temp.nb_registers = (sizeof(stHR)/2);
    temp.start_registers = 0;
    temp.tab_registers = (unsigned short*)&stHR;
    mb_mapping = &temp;
    unsigned char SD_time=0;
    while (1)
    {

        s = modbus_tcp_listen(ctx, 1);
        modbus_tcp_accept(ctx, &s);
        while (1)
        {
            do {
                rc = modbus_receive(ctx, query);
                /* Filtered queries return 0 */
            } while (rc == 0);
            if (rc == -1 && errno != EMBBADCRC) {
                /* Quit */
                break;
            }
            rc = modbus_reply(ctx, query, rc, mb_mapping);
            if(SD_time>50){
                SD_time=0;
                adc_get();
                checkSD();
            }else
                SD_time ++;
            if((!loaded)&&(mounted))
                Read_Cfg();
            if(stHR.xHMICmd==1){
                printf("Control off\n");
            }
            if(stHR.xHMICmd==2){
                printf("Control on\n");
            }
            if(stHR.xHMICmd==4){
                printf("Write cfg\n");
                Save_Cfg();
```

```
            }
            if(stHR.xHMICmd==8){
                printf("Read cfg\n");
                Read_Cfg();
            }
            if(stHR.xHMICmd==16){
                printf("Reset Addr\n");
                stNode[0].xAddrInit = 0;
                stNode[1].xAddrInit = 0;
            }
            if(stHR.xHMICmd==32){
                printf("Set IP\n");
                Save_Cfg();
                Set_IP();
            }
            stHR.xHMICmd=0;
            os_usleep (5000);
        }
        os_usleep (5000);
    }
}

static void os_ecc_task (void * thread_arg)
{
    while (1)
    {
        ECCUpdate(&stNode[0]);
        ECCUpdate(&stNode[1]);
        os_usleep (1000);
    }
}

void initModbusTCP(app_data_t * app){
    os_thread_create (
         "os_web_task",
         APP_WEB_THREAD_PRIORITY,
         APP_WEB_THREAD_STACKSIZE,
         os_web_task,
         (void *)app);

    os_thread_create (
         "os_ecc_task",
         APP_WEB_THREAD_PRIORITY,
         APP_WEB_THREAD_STACKSIZE,
         os_ecc_task,
         (void *)app);
   //ctx->debug=1;
}
```

6. API для работы с CAN

```
#include "pnal.h"

#include "pnet_options.h"
#include "options.h"
#include "osal_log.h"

#include <net/ethernet.h>
#include <net/if.h>
#include <netpacket/packet.h>
#include <sys/ioctl.h>
#include <linux/can.h>
#include <linux/can/raw.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "app_can.h"
#include "app_ecc.h"
#include "osal.h"

#define APP_CAN_THREAD_PRIORITY  15
#define APP_CAN_THREAD_STACKSIZE 4096 /* bytes */


void InitCAN(CanNode_t *Node)
{
    int res=0;
    char cmd[60];
    //setting up port and bitrate
    sprintf(cmd,"sudo ifconfig %s down", Node->cName);
    res=(unsigned int)system(cmd);
    sprintf(cmd,"sudo ip link set %s type can bitrate 1000000", Node->cName);
    res=(unsigned int)system(cmd);
    sprintf(cmd,"sudo ifconfig %s up", Node->cName);
    res=(unsigned int)system(cmd);
    sprintf(cmd,"sudo ip -details link show %s", Node->cName);
    res=(unsigned int)system(cmd);
    /*res=(unsigned int)system(("sudo ifconfig can0iobus down"));
    res=(unsigned int)system(("sudo ip link set can0iobus type can bitrate
250000"));
    res=(unsigned int)system(("sudo ifconfig can0iobus up"));
    res=(unsigned int)system(("sudo ip -details link show can0iobus"));*/
    if(res<0){
        printf("CAN init failed\n");
    }
    //setup socketcan
    struct sockaddr_can addr;
    struct ifreq ifr;
```

```
    Node->SocketFD = socket(PF_CAN, SOCK_RAW|SOCK_NONBLOCK, CAN_RAW);
    strcpy(ifr.ifr_name, Node->cName);
    ioctl(Node->SocketFD, SIOCGIFINDEX, &ifr);

    addr.can_family = AF_CAN;
    addr.can_ifindex = ifr.ifr_ifindex;
    bind(Node->SocketFD, (struct sockaddr *)&addr, sizeof(addr));
    Node->q.max = 40;
    //Start
    StartThread();
}

uint32_t getCTick(void) {
    struct timespec ts;
    unsigned theTick = 0U;
    clock_gettime( CLOCK_MONOTONIC, &ts );
    theTick  = ts.tv_nsec / 1000000;
    theTick += ts.tv_sec * 1000;
    return theTick;
}

void SendMsg(CanNode_t *Node,struct can_frame msg)
{
    int nbytes = write(Node->SocketFD, &msg, sizeof(struct can_frame));
    if(nbytes == -1)
    {
        printf("send error\n");
        //exit(1);
    }
}

void SendQueueMsg(CanNode_t *Node,struct can_frame msg)
{
   Node->q.w_ptr++;
   if(Node->q.w_ptr>Node->q.max)Node->q.w_ptr=0;
   Node->q.f[Node->q.w_ptr] = msg;
}

int GetQueueMsg(CanNode_t *Node,struct can_frame* msg)
{
    if(Node->q.r_ptr!=Node->q.w_ptr){
        Node->q.r_ptr++;
        if(Node->q.r_ptr>Node->q.max)Node->q.r_ptr=0;
        *msg = Node->q.f[Node->q.r_ptr];
        Node->q.f[Node->q.r_ptr].can_id = 0;
        return Node->q.w_ptr - Node->q.r_ptr;
    }else{
        return 0;
    }
}
```

```
void StartThread(void)
{
    stNode[0].m_IsRunning = true;
    stNode[1].m_IsRunning = true;
    os_thread_create (
         "CanRx",
         APP_CAN_THREAD_PRIORITY,
         APP_CAN_THREAD_STACKSIZE,
         RxThread,
         0);
    os_thread_create (
         "CanTX",
         APP_CAN_THREAD_PRIORITY,
         APP_CAN_THREAD_STACKSIZE,
         TxThread,
         0);
}

void EndThread(void)
{
    stNode[0].m_IsRunning = true;
    stNode[1].m_IsRunning = true;
}

void RxThread(void *argv)
{
    struct can_frame frame;
    while(stNode[0].m_IsRunning|stNode[1].m_IsRunning)
    {
        //Read Frame
        for(int i=0;i<2;i++){
            while(read(stNode[i].SocketFD, &frame, sizeof(struct can_frame))>0){
                unsigned short id = frame.can_id&0x1F;
                if (id == 0x00){
                    stNode[i].xFindNotInit = true;
                } else {
                    if( id < 32 ){
                        id = id - 1;
                        switch ( frame.can_id & 0xFFC0 ) {
                            case 0x180:
                                stHR.MotorStatus[i][id].b &=~0x3FFF;
                                stHR.MotorStatus[i][id].b |=
frame.data[1]|((frame.data[2]<<8)&0x3F00);
                                break;
                            case 0x280:
                                stHR.MotorADC1[i][id].Motor_Voltage =
frame.data[0]+100;
                                stHR.MotorADC1[i][id].Logic_Voltage =
frame.data[1]+100;
                                stHR.MotorADC1[i][id].Source_Voltage =
frame.data[2]+100;
```

```
                                stHR.MotorADC1[i][id].Voltage_10V =
frame.data[3]+100;
                                stHR.MotorADC1[i][id].Voltage_18V =
frame.data[4]+100;
                                //stRecalVal2.BEMF_PhaseA = frame.data[5]+100;
                                //stRecalVal2.BEMF_PhaseB = frame.data[6]+100;
                                //stRecalVal2.BEMF_PhaseC = frame.data[7]+100;
                                break;
                            case 0x380:
                                //stRecalVal2.BEMF_Central = frame.data[0]+100;
                                stHR.MotorADC2[i][id].Sensor_Voltage =
frame.data[1]+100;
                                stHR.MotorADC1[i][id].Hall_Voltage =
frame.data[2]+100;
                                stHR.MotorADC2[i][id].Motor_Current =
frame.data[3];
                                //stRecalVal2.Current_PhaseB = frame.data[4]+100;
                                //stRecalVal2.Current_PhaseC = frame.data[5]+100;
                                stHR.MotorADC2[i][id].uiCustomADC =
(uint16_t)(frame.data[7] << 8 | frame.data[6]);
                                break;
                            case 0x480:
                                stHR.CurrentParam[i][id].ucMotorType =
frame.data[0];
                                //stCurrentParams.ucDriveDirection =
frame.data[1];
                                memcpy(&(stHR.CurrentParam[i][id].rSpeed),&(frame
.data[2]),4);
                                stHR.CurrentParam[i][id].ucRampUp =
frame.data[6];
                                stHR.CurrentParam[i][id].ucRampDown =
frame.data[7];
                                break;
                            default:
                                break;
                        }
                        stNode[i].last_link[id] = getCTick();
                    }
                }
            }
        }
        os_usleep (5000);
    }

}

void TxThread(void *argv)
{
    struct can_frame frame;
    struct can_frame frame5ms;
    while(stNode[0].m_IsRunning|stNode[1].m_IsRunning)
```

```
    {

        //TODO: Use `select()` instead, non-block reading required here.
        //while(((CanManager*)argv) -> TxRequestFlag){;}
        for(int i=0;i<2;i++){
            while(GetQueueMsg(&stNode[i],&frame)){
                int nbytes = write(stNode[i].SocketFD, &frame, sizeof(struct
can_frame));
                stNode[i].SendCnt++;
                if(nbytes == -1)
                {
                    //printf("send error\n");
                    //exit(1);
                    if(stNode[i].can_err<100)
                        stNode[i].can_err++;
                    else
                        stNode[i].Status|=0x02;
                }else{
                    stNode[i].Status&=~0x02;
                    stNode[i].can_err=0;
                }

            }
            frame5ms = stNode[i].Msg5ms;
            if(frame5ms.can_id > 0){
                int nbytes = send(stNode[i].SocketFD, &frame5ms, sizeof(struct
can_frame), 0);
                if(nbytes == -1)
                {
                    //printf("send error\n");
                    //exit(1);
                    if(stNode[i].can_err<100)
                        stNode[i].can_err++;
                    else
                        stNode[i].Status|=0x02;
                }else{
                    stNode[i].Status&=~0x02;
                    stNode[i].can_err=0;
                }
            }
        }
        os_usleep (5000);
    }

}
```

7. Определения функций API CAN

```
#ifndef APP_CAN_H
#define APP_CAN_H


#ifdef __cplusplus
extern "C" {
#endif

#include <stdint.h>
#include <linux/can.h>
#include <linux/can/raw.h>
#include "app_ecc.h"



void InitCAN(CanNode_t *Node);

void SendMsg(CanNode_t *Node,struct can_frame msg);
void SendQueueMsg(CanNode_t *Node,struct can_frame msg);
int GetQueueMsg(CanNode_t *Node,struct can_frame* msg);

//Receive, with a single thread.
void StartThread(void);
void EndThread(void);
void RxThread(void *arg);
void TxThread(void *arg);

#ifdef __cplusplus
}
#endif

#endif /* APP_LOG_H */
```