

Arduino bioreaktor Herbion

v0.9.0

Generated by Doxygen 1.12.0

1 main issue	1
1.0.1 this project working by using platformio extension for visual studio code	1
1.0.2 Code is not fully tested	1
1.0.3 all code on develop branch is still in progress it is not recommend to copy/clone any part of it	1
1.1 Hardware Documentation	1
1.1.1 Check Hardware_configuration for more info	1
1.2 Hardware and software Configuration	1
1.2.1 Check Configuration for more info	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Class Documentation	9
5.1 arduino_task< T > Class Template Reference	9
5.2 Config_var Class Reference	9
5.3 ConfigurationVariable Class Reference	9
5.4 DataHMS Class Reference	10
5.5 MeasureArray Class Reference	10
5.6 MemoryManager Class Reference	10
5.7 my_rotary_encoder Class Reference	11
5.8 MyLCD Class Reference	11
5.9 SdMemoryManager Class Reference	11
5.9.1 Detailed Description	12
5.9.2 Member Function Documentation	12
5.9.2.1 write_data_frame_to_st()	12
5.10 Sensor Class Reference	12
5.11 TimerLowPriority Class Reference	13
6 File Documentation	15
6.1 arduino_os.hpp	15
6.2 bioreactor_defined_const.hpp	16
6.3 components.hpp	17
6.4 eeprom_menager.hpp	17
6.5 lcd_display.hpp	18
6.6 my_encoder.hpp	18
6.7 sd_memory.hpp	19
6.8 sensor_config.hpp	20
6.9 Unit_tests.hpp	21

6.10 utility.hpp	21
Index	23

Chapter 1

main issue

1.0.1 this project working by using platformio extension for visual studio code

1.0.2 Code is not fully tested

1.0.3 all code on develop branch is still in progress it is not recommend to copy/clone any part of it

1.1 Hardware Documentation

1.1.1 Check Hardware_configuration for more info

note that this documentation do not contain information about components connection

1.2 Hardware and software Configuration

1.2.1 Check Configuration for more info

how to setup environment and connect pin to make project work with base setup

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

arduino_task< T >	9
Config_var	9
ConfigurationVariable	9
DataHMS	10
LiquidCrystal_I2C	
MyLCD	11
MeasureArray	10
MemoryManager	10
my_rotary_encoder	11
SdMemoryManager	11
Sensor	12
TimerLowPriority	13

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

arduino_task< T >	9
Config_var	9
ConfigurationVariable	9
DataHMS	10
MeasureArray	10
MemoryManager	10
my_rotary_encoder	11
MyLCD	11
SdMemoryManager	
Main purpose is write data to SD card	11
Sensor	12
TimerLowPriority	13

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

arduino_os.hpp	15
bioreactor_defined_const.hpp	16
components.hpp	17
eeprom_menager.hpp	17
lcd_display.hpp	18
my_encoder.hpp	18
sd_memory.hpp	19
sensor_config.hpp	20
Unit_tests.hpp	21
utility.hpp	21

Chapter 5

Class Documentation

5.1 `arduino_task< T >` Class Template Reference

Public Member Functions

- **`arduino_task`** (`uint8_t` priority, unsigned long repeat_time, `T *object`, `void(T::*method)()`)
- void **`examine_task_state`** (unsigned long time)
- void **`run_task`** ()

The documentation for this class was generated from the following file:

- `arduino_os.hpp`

5.2 `Config_var` Class Reference

Public Member Functions

- **`Config_var`** (`float` desire_ph, `float` max_ph_acceptable_deviation, `float` desire_temp, `float` max_temp_acceptable_deviation)

The documentation for this class was generated from the following files:

- `utility.hpp`
- `utility.cpp`

5.3 `ConfigurationVariable` Class Reference

Public Member Functions

- **`ConfigurationVariable`** (`MemoryManager` &mem_manager)
- int **`get_addr`** () const
- float **`return_config_value`** () const
- void **`retrieve_config_values_from_eeprom`** ()
- void **`change_config_value`** (`float` value)

The documentation for this class was generated from the following files:

- `eeeprom_menager.hpp`
- `eeeprom_menager.cpp`

5.4 DataHMS Class Reference

Public Member Functions

- **DataHMS** (long hour, long minute, long second)
- String **return_data** ()

Public Attributes

- long **m_offset**

The documentation for this class was generated from the following files:

- utility.hpp
- utility.cpp

5.5 MeasureArray Class Reference

Public Member Functions

- **MeasureArray** (int size)
- void **init** (float initial_value)
- void **add_measure** (float value)
- float **read_measure** (char index)
- float **get_average** ()

The documentation for this class was generated from the following files:

- utility.hpp
- utility.cpp

5.6 MemoryManager Class Reference

Public Member Functions

- **MemoryManager** (int memory_start, int memory_length)
- int **give_memory** (int require_memory)

The documentation for this class was generated from the following files:

- eeprom_menager.hpp
- eeprom_menager.cpp

5.7 my_rotary_encoder Class Reference

Public Member Functions

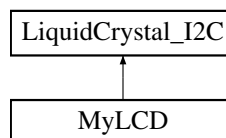
- **my_rotary_encoder** (uint8_t pin_a, uint8_t pin_b, uint8_t button, long sensitivity)
- void **init** ()
- int **get_encoder_pos** ()
- int **get_button_state** ()
- long **return_button_inactivate_state_time** ()
- void **check_encoder_pos** ()
- void **reset_encoder_pos** ()
- int **get_encoder_move** ()
- float **set_value** (float initial_value, float step, [MyLCD](#) lcd)
- int **get_button_depth** ()
- void **reset__button_depth** ()

The documentation for this class was generated from the following files:

- my_encoder.hpp
- my_encoder.cpp

5.8 MyLCD Class Reference

Inheritance diagram for MyLCD:



Public Member Functions

- **MyLCD** (uint8_t addr, uint8_t column_num, uint8_t row_num)
- void **initialize** ()
- void **send_float_value** (String text, float value, int lcd_row)
- void **send_string** (String text, String value, int lcd_row)

The documentation for this class was generated from the following files:

- lcd_display.hpp
- lcd_display.cpp

5.9 SdMemoryManager Class Reference

Main purpose is write data to SD card.

```
#include <sd_memory.hpp>
```

Public Member Functions

- **SdMemoryManager** (uint8_t mosi_pin, uint8_t miso_pin, uint8_t sck_pin, uint8_t cs_pin)
- void **init** ()
- void **write_to_st** ()
- void **write_data_frame_to_st** ([Sensor](#) &thermometer, [Sensor](#) &ph_meter, [Sensor](#) &oxygen_meter, [DataHMS](#) &data)
- String **DEBUG_write_data_frame** ([Sensor](#) &thermometer, [Sensor](#) &ph_meter, [Sensor](#) &oxygen_meter, [DataHMS](#) &data)
- void **save** ()
- void **close_file** ()

5.9.1 Detailed Description

Main purpose is write data to SD card.

this class mostly use function in SD.h library main idea is to create file and write data to it

5.9.2 Member Function Documentation

5.9.2.1 write_data_frame_to_st()

```
void SdMemoryManager::write_data_frame_to_st (
    Sensor & thermometer,
    Sensor & ph_meter,
    Sensor & oxygen_meter,
    DataHMS & data)
```

Parameters

<i>thermometer</i>	
<i>ph_meter</i>	
<i>oxygen_meter</i>	
<i>data</i>	

The documentation for this class was generated from the following files:

- sd_memory.hpp
- sd_memory.cpp

5.10 Sensor Class Reference

Public Member Functions

- **Sensor** (MeasuringDevice *measuring_dev, [ConfigurationVariable](#) &zero_shift, [ConfigurationVariable](#) &linear_factor)
- float **get_value_from_measurement** ()
- float **get_value** ()
- void **init** ()

Public Attributes

- [ConfigurationVariable](#) **m_zero_shift**
- [ConfigurationVariable](#) **m_linear_factor**
- [MeasuringDevice](#) * **m_measuring_device**

The documentation for this class was generated from the following files:

- [eeprom_menager.hpp](#)
- [eeprom_menager.cpp](#)

5.11 TimerLowPriority Class Reference

Public Member Functions

- bool **activate** (int time_to_activate)
- void **reset** ()

The documentation for this class was generated from the following files:

- [utility.hpp](#)
- [utility.cpp](#)

Chapter 6

File Documentation

6.1 arduino_os.hpp

```
00001 //some kind basic job schedule optimized to minimal size and ram usage
00002
00003
00004 //general idea - adding task then planer will check
00005
00006 #include <Arduino.h>
00007
00008 #ifndef ARDUINO_OS
00009 #define ARDUINO_OS
00010
00011 template <typename T>
00012 class arduino_task {
00013 private:
00014
00015     const uint8_t m_priority;
00016     const unsigned long m_repeat_time;
00017
00018
00019     T* m_object;
00020     void (T::*m_task_method)();
00021
00022
00023     uint8_t m_state;
00024     unsigned long m_last_call;
00025
00026 public:
00027
00028     arduino_task(uint8_t priority, unsigned long repeat_time, T* object, void (T::*method)());
00029     void examine_task_state(unsigned long time);
00030     void run_task();
00031 };
00032
00033 template <typename T>
00034 arduino_task<T>::arduino_task(uint8_t priority, unsigned long repeat_time, T* object, void
(T::*method)())
00035     : m_priority(priority),
00036       m_repeat_time(repeat_time),
00037       m_object(object),
00038       m_task_method(method),
00039       m_state(0),
00040       m_last_call(0)
00041 {}
00042
00043 template <typename T>
00044 void arduino_task<T>::examine_task_state(unsigned long time) {
00045     if (time >= m_last_call + m_repeat_time) {
00046         m_state = 1;
00047         m_last_call = time;
00048     } else {
00049         m_state = 0;
00050     }
00051 }
00052
00053 template <typename T>
00054 void arduino_task<T>::run_task() {
00055
00056     if(m_state==1){(m_object->*m_task_method)();}
00057     m_state = 0;
```

```

00058
00059 }
00060
00061
00062
00063
00064
00065 #endif //ARDUINO_OS

```

6.2 bioreactor_defined_const.hpp

```

00001 #ifndef BIOREACTOR_CONST
00002 #define BIOREACTOR_CONST
00003
00004 #include<Arduino.h>
00005
00006 //BUTTON SETTING - CAN BE CONFIGURABLE!
00007 const char BUTTON_DEFAULT_STATE = 1; //mean pin state is HIGH
00008 const long int BUTTON_STAY_ON_STATE = 2000; //[ms] time button dont change its state
00009 const long int STAY_IN_MENU_TIME = 3000; //[ms] time after you can exit menu
00010
00011
00012 //ROTARY ENCODER SETTINGS - CAN BE CONFIGURABLE!
00013
00014 const unsigned long SENSITIVITY = 150; // [ms] time after re will not change it state after did it
    before //TODO use better world :)
00015
00016
00017 //LCD SETTINGS
00018 // for now not available
00019
00020 //SENSOR DESIRE PARAMETER - CAN BE CONFIGURABLE!
00021 const float DESIRE_PH = 7;
00022 const float MAX_PH_ACCEPTABLE_DEVIATION = 0.5;
00023 const float DESIRE_TEMP = 21.37;
00024 const float MAX_TEMP_ACCEPTABLE_DEVIATION = 4.2069;
00025
00026
00027 //LCD PHYSICAL CONNECTION
00028 // TODO
00029
00030 //SD CARD READER PHYSICAL CONNECTION
00031 constexpr uint8_t CS_PIN = 10;
00032 constexpr uint8_t DEF_MOSI_PIN = 11;
00033 constexpr uint8_t DEF_MISO_PIN = 12;
00034 constexpr uint8_t DEF_SCK_PIN = 13;
00035 #define FILENAME "m.txt"
00036
00037 //ROTARY ENCODER PHYSICAL CONNECTION
00038 constexpr uint8_t RE_PIN_A = 3;
00039 constexpr uint8_t RE_PIN_B = 4;
00040 constexpr uint8_t RE_BUTTON_PIN = 5;
00041
00042 //SENSORS PHYSICAL CONNECTION
00043 constexpr uint8_t PH_METER_PIN = A1;
00044 constexpr uint8_t THERMOMETER_PIN = A2;
00045 constexpr uint8_t OXYGEN_METER_PIN = A3;
00046
00047 //SAMPLE PUMP PARAMETER (this pump is peristaltic and work only in on/off mode)
00048 constexpr int PUMP_FLOW_SPEED = 60; //[ml/min]
00049 constexpr int SAMPLE_SIZE = 2; //[ml]
00050 constexpr int PUMP_SAMPLE_TAKING_TIME = int(float(SAMPLE_SIZE)/float(PUMP_FLOW_SPEED)*60*1000);
    //[ms]; should be 2000ms
00051
00052 //PH KEEPERS PUMP _PH_CORRECTION - CAN BE CONFIGURABLE!
00053 const long int TIME_BETWEEN_PH_CORRECTION = 10000; //[ms] time between ph correction [TODO change
    name]
00054 const long int CORRECTION_TIME = 100; // [ms] initial time of correction
00055 const float MULT_PER_DEGREE = 3; // it mean if desire ph is 6 and we have 8 time of pump activate
    will be 100*3*3 = 900 ms
00056 const long int MAX_REACTION_TIME = 3000; //max time of open the pump
00057
00058
00059
00060 // thermometer parameter (it is hardware parameter)
00061 const long int THERMISTOR_NOMINAL = 10000; // [ohm] resistance at 25 degrees C
00062 const uint8_t TEMPERATURE_NOMINAL = 25; // temp. for nominal resistance (almost always 25 C)
00063 const long int B_COEFFICIENT = 3950; // [UNIT!] The beta coefficient of the thermistor
    (usually 3000-4000)
00064 const long int SERIES_RESISTOR = 10000; // [ohm] the value of the 'other' resistor
00065
00066
00067
00068 #endif // BIOREACTOR_CONST

```

6.3 components.hpp

```

00001 #include <Arduino.h>
00002 #include <math.h>
00003 #include "bioreactor_defined_const.hpp"
00004 #include "utility.hpp"
00005
00006 #ifndef BIOREACTOR_COMPONENTS
00007 #define BIOREACTOR_COMPONENTS
00008
00009
00010 class SimplePeristalticPump
00011 {
00012 private:
00013     uint8_t m_pin_forward;
00014
00015 public:
00016     SimplePeristalticPump(uint8_t pin_forward);
00017     void init();
00018     void stabilize_ph(float current_ph, float desire_ph);
00019 };
00020
00021
00022 class PeristalticPump
00023 {
00024 private:
00025     const uint8_t m_pwm_pin; //TODO snake case
00026     const uint8_t m_dir1_pin;
00027     const uint8_t m_dir2_pin;
00028     int m_current_pwm_value;
00029
00030 public:
00031     PeristalticPump(uint8_t pwm_pin, uint8_t dir1_pin, uint8_t dir2_pin); //TODO rename variable
00032     void init();
00033     void set_pump_speed(int value); // from -100 (max reverse speed) to 100 (max forward speed)
00034     long get_current_speed() const;
00035     void take_sample();
00036 };
00037
00038 class MeasuringDevice
00039 {
00040     //TODO: add virtual destructor even empty
00041 protected:
00042     const int m_read_pin;
00043     float m_value; // check what sensors return TODO for now int ; but probably will be change in
inheritance
00044
00045 public:
00046     MeasuringDevice(int read_pin);
00047     void init();
00048     virtual float get_value();
00049 };
00050
00051 class Thermometer : public MeasuringDevice
00052 {
00053 public:
00054     Thermometer(int read_pin);
00055     float get_value() override;
00056 };
00057
00058 class PhMeter : public MeasuringDevice
00059 {
00060 public:
00061     PhMeter(int read_pin);
00062     float get_value() override;
00063 };
00064
00065 class OxygenMeter : public MeasuringDevice
00066 {
00067 public:
00068     OxygenMeter(int read_pin);
00069     float get_value() override;
00070 };
00071
00072 #endif // BIOREACTOR_COMPONENTS

```

6.4 eeprom_menager.hpp

```

00001 #include "components.hpp"
00002 #include <EEPROM.h>
00003
00004 #ifndef TEST
00005 #define TEST

```

```

00006
00007 class MemoryManager
00008 {
00009     private:
00010         const int m_memory_start;
00011         const int m_memory_size;
00012         int m_memory_pointer;
00013
00014     public:
00015         MemoryManager(int memory_start, int memory_length);
00016         int give_memory(int require_memory);
00017 };
00018
00019 class ConfigurationVariable
00020 {
00021     private:
00022         int m_memory_addr;
00023         float m_value = -21.37;
00024
00025     public:
00026         ConfigurationVariable(MemoryManager &mem_manager);
00027         int get_addr() const;
00028         float return_config_value() const;
00029         void retrieve_config_values_from_eeprom();
00030         void change_config_value(float value);
00031 };
00032
00033 class Sensor
00034 {
00035     private:
00036         float m_value = 10;
00037
00038     public: // bad practice TODO in free time try to change to private
00039         ConfigurationVariable m_zero_shift, m_linear_factor; //TODO - make reference to this object not
copy
00040         MeasuringDevice *m_measuring_device;
00041
00042     public:
00043         Sensor(MeasuringDevice *measuring_dev, ConfigurationVariable &zero_shift, ConfigurationVariable
&linear_factor);
00044         float get_value_from_measurement();
00045         float get_value();
00046         void init();
00047 };
00048
00049 #endif // TEST

```

6.5 lcd_display.hpp

```

00001 #ifndef LCD_DISPLAY_BR
00002 #define LCD_DISPLAY_BR
00003
00004 #include<LiquidCrystal_I2C.h>
00005 // not refactored
00006
00007 class MyLCD : public LiquidCrystal_I2C
00008 {
00009
00010     public:
00011         MyLCD(uint8_t addr, uint8_t column_num, uint8_t row_num);
00012         void initialize(); //TODO: better name
00013         void send_float_value(String text, float value, int lcd_row); //TODO: change to template
00014         void send_string(String text, String value, int lcd_row);
00015 };
00016
00017 #endif // LCD_DISPLAY_BR

```

6.6 my_encoder.hpp

```

00001 #ifndef MY_ENCODER
00002 #define MY_ENCODER
00003
00004 #include <Arduino.h>
00005 #include "lcd_display.hpp"
00006
00007
00013 class my_rotary_encoder
00014 {
00015     private:

```

```

00016     const uint8_t m_pin_a;
00017     const uint8_t m_pin_b;
00018     const uint8_t m_pin_button;
00019
00020     uint8_t m_button_depth;
00021     unsigned long m_sensitivity;
00022     unsigned long m_button_inactivate_state_time; //TODO rename
00023     unsigned long m_last_change;
00024
00025     uint8_t m_a_val = 0;
00026     uint8_t m_pinALast = 0;
00027     uint8_t m_button_state = 1;
00028     int m_encoderPosCount;
00029
00030 public:
00031     my_rotary_encoder(uint8_t pin_a, uint8_t pin_b, uint8_t button, long sensitivity);
00032     void init();
00033     int get_encoder_pos();
00034     int get_button_state();
00035     long return_button_inactivate_state_time();
00036     void check_encoder_pos();
00037     void reset_encoder_pos();
00038     int get_encoder_move();
00039     float set_value(float initial_value, float step, MyLCD lcd);
00040     int get_button_depth();
00041     void reset__button_depth();
00042 };
00043
00044 #endif

```

6.7 sd_memory.hpp

```

00001 //TODO REFACTOR WHOLE CLASS in way which Define is not used in CPP file
00002
00003 #ifndef SM_MEMORY
00004 #define SM_MEMORY
00005
00006 #include <SD.h>
00007 #include "eeprom_menager.hpp"
00008 #include "utility.hpp"
00009
00010 class SdMemoryManager
00011 {
00012 private:
00013     // pins
00014     const uint8_t m_mosi_pin;
00015     const uint8_t m_miso_pin;
00016     const uint8_t m_sck_pin;
00017     const uint8_t m_cs_pin;
00018
00019     uint8_t m_write_number=0;
00020
00021     File m_file;
00022     String m_filename;
00023     String m_dataBuffer;
00024     unsigned long m_last_write = 0;
00025
00026 public:
00027     SdMemoryManager(uint8_t mosi_pin, uint8_t miso_pin, uint8_t sck_pin, uint8_t cs_pin);
00028
00029     void init();
00030     void write_to_st();
00031     void write_data_frame_to_st(Sensor& thermometer, Sensor& ph_meter, Sensor& oxygen_meter, DataHMS&
00032 data);
00033     String DEBUG_write_data_frame(Sensor& thermometer, Sensor& ph_meter, Sensor& oxygen_meter,
00034 DataHMS& data);
00035     void save();
00036     void close_file();
00037 };
00038
00039 #endif
00040
00041 /*
00042 #include <SD.h>
00043
00044 const int chipSelect = 10;
00045
00046 // file name to use for writing
00047 const char filename[] = "datalog.txt";
00048
00049 // File object to represent file

```

```

00074 File myFile;
00075 // string to buffer output
00076 String dataBuffer;
00077 // last time data was written to card:
00078 unsigned long lastMillis = 0;
00079
00080 void setup() {
00081     // Open serial communications and wait for port to open:
00082     Serial.begin(9600);
00083     // reserve 1 kB for String used as a dataBuffer
00084     dataBuffer.reserve(1024);
00085
00086     // set LED pin to output, used to blink when writing
00087     pinMode(LED_BUILTIN, OUTPUT);
00088
00089     // wait for Serial Monitor to connect. Needed for native USB port boards only:
00090     while (!Serial);
00091
00092     Serial.print("Initializing SD card...");
00093
00094     if (!SD.begin(chipSelect)) {
00095         Serial.println("Initialization failed. Things to check:");
00096         Serial.println("1. is a card inserted?");
00097         Serial.println("2. is your wiring correct?");
00098         Serial.println("3. did you change the chipSelect pin to match your shield or module?");
00099         Serial.println("Note: press reset button on the board and reopen this Serial Monitor after fixing
your issue!");
00100         while (true);
00101     }
00102
00103     Serial.println("Initialization done.");
00104
00105     // If you want to start from an empty file,
00106     // uncomment the next line:
00107     // SD.remove(filename);
00108     // try to open the file for writing
00109
00110     myFile = SD.open(filename, FILE_WRITE);
00111     if (!myFile) {
00112         Serial.print("error opening ");
00113         Serial.println(filename);
00114         while (true);
00115     }
00116
00117     // add some new lines to start
00118     myFile.println();
00119     myFile.println("Hello World!");
00120     Serial.println("Starting to write to file...");
00121 }
00122
00123 void loop() {
00124     // check if it's been over 10 ms since the last line added
00125     unsigned long now = millis();
00126     if ((now - lastMillis) >= 10) {
00127         // add a new line to the dataBuffer
00128         dataBuffer += "Hello ";
00129         dataBuffer += now;
00130         dataBuffer += "\r\n";
00131         // print the buffer length. This will change depending on when
00132         // data is actually written to the SD card file:
00133         Serial.print("Unsaved data buffer length (in bytes): ");
00134         Serial.println(dataBuffer.length());
00135         // note the time that the last line was added to the string
00136         lastMillis = now;
00137     }
00138
00139     // check if the SD card is available to write data without blocking
00140     // and if the dataBuffered data is enough for the full chunk size
00141     unsigned int chunkSize = myFile.availableForWrite();
00142     if (chunkSize && dataBuffer.length() >= chunkSize) {
00143         // write to file and blink LED
00144         digitalWrite(LED_BUILTIN, HIGH);
00145         myFile.write(dataBuffer.c_str(), chunkSize);
00146         digitalWrite(LED_BUILTIN, LOW);
00147         // remove written data from dataBuffer
00148         dataBuffer.remove(0, chunkSize);
00149     }
00150 }
00151
00152 */

```

6.8 sensor_config.hpp

```
00001 #include "bioreactor_defined_const.hpp"
```



```

00002 #include "components.hpp"
00003 #include "eeprom_menager.hpp"
00004 #include <Arduino.h>
00005
00006 #ifndef MY_SC
00007 #define MY_SC
00008
00009 Sensor setup_thermometer_sensors(MemoryManager &manager);
00010 Sensor setup_ph_sensors(MemoryManager &manager);
00011 Sensor setup_oxygen_sensors(MemoryManager &manager);
00012 void test_sensor(Sensor &sensor, float value_new, String sensor_name = "Sensor");
00013
00014 #endif

```

6.9 Unit_tests.hpp

```

00001 #include "utility.hpp"
00002 #include <AUnit.h>
00003
00004 #ifndef MY_UNIT_TEST
00005 #define MY_UNIT_TEST
00006
00007 // not refactored
00008
00009 int dumb_func()
00010 {
00011     return 1;
00012 }
00013
00014 MeasureArray m1(10), m2(3); // 10 ; 3
00015
00016 float test_mes_array(MeasureArray &ma)
00017 {
00018     ma.add_measure(20);
00019     ma.add_measure(10);
00020     ma.add_measure(5);
00021     ma.add_measure(5);
00022     ma.add_measure(60);
00023     return (ma.get_average());
00024 }
00025
00026 float test_mes_array2(MeasureArray &ma)
00027 {
00028     ma.add_measure(20);
00029     ma.add_measure(10);
00030     ma.add_measure(2137);
00031
00032     return ma.read_measure(2);
00033 }
00034
00035 test(test_mes_arrayTest)
00036 {
00037     assertEquals(test_mes_array(m1), 10.0);
00038 }
00039
00040 test(test_mes_array2Test)
00041 {
00042     assertEquals(test_mes_array2(m2), 2137.0);
00043 }
00044
00045 test(dumb_funcTest)
00046 {
00047     assertEquals(dumb_func(), 1);
00048 }
00049
00050 #endif // MY_UNIT_TEST

```

6.10 utility.hpp

```

00001 #ifndef MY_UTILITY
00002 #define MY_UTILITY
00003
00004 #include<string.h>
00005 #include<Arduino.h>
00006
00007 #include "my_encoder.hpp"
00008 #include "eeprom_menager.hpp"
00009 #include "lcd_display.hpp"
00010

```

```

00011 class Config_var    //more like struct but... TODO check if we can change it to real struct
00012 {
00013     private:
00014         float m_desire_ph;
00015         float m_max_ph_acceptable_deviation;
00016
00017         float m_desire_temp;
00018         float m_max_temp_acceptable_deviation;
00019
00020     public:
00021         Config_var(float desire_ph, float max_ph_acceptable_deviation, float desire_temp, float
max_temp_acceptable_deviation);
00022 };
00023
00024
00025
00026
00027 class MeasureArray // TODO is not good enough make it better
00028 {
00029     private: //TODO: inheritance after arduino vector
00030         //TODO: is not good enough make it better
00031         const char m_array_size; // max 256 (but not recommended to use more than 100)
00032         float *measurement; // Pointer array, will be initialized in constructor
00033         int m_memory_cursor = 0;
00034         float m_oldest_measure;
00035
00036     public:
00037         MeasureArray(int size);
00038         void init(float initial_value); //TODO make it vector
00039         void add_measure(float value);
00040         float read_measure(char index);
00041         float get_average(); //TODO: add const
00042 };
00043
00044 class TimerLowPriority
00045 {
00046     private:
00047         unsigned long m_start_time;
00048         unsigned long m_end_time;
00049
00050     public:
00051         TimerLowPriority();
00052         bool activate(int time_to_activate);
00053         void reset();
00054 };
00055
00056 // TODO high priority timer
00057
00058 class DataHMS
00059 {
00060     public:
00061         long m_offset;
00062
00063     public:
00064         DataHMS(long hour, long minute, long second);
00065         String return_data(); //TODO const
00066 };
00067
00068 void print_config_menu(my_rotary_encoder &encoder, MyLCD &lcd, // main sensors
00069                      Sensor &term, Sensor &ph, Sensor &oxygen, PeristalticPump& pump);
00070
00071 #endif

```

Index

arduino_os.hpp, [15](#)
arduino_task< T >, [9](#)

bioreactor_defined_const.hpp, [16](#)

components.hpp, [17](#)
Config_var, [9](#)
ConfigurationVariable, [9](#)

DataHMS, [10](#)

eeprom_menager.hpp, [17](#)

lcd_display.hpp, [18](#)

main issue, [1](#)
MeasureArray, [10](#)
MemoryManager, [10](#)
my_encoder.hpp, [18](#)
my_rotary_encoder, [11](#)
MyLCD, [11](#)

sd_memory.hpp, [19](#)
SdMemoryManager, [11](#)
 write_data_frame_to_st, [12](#)
Sensor, [12](#)
sensor_config.hpp, [20](#)

TimerLowPriority, [13](#)

Unit_tests.hpp, [21](#)
utility.hpp, [21](#)

write_data_frame_to_st
 SdMemoryManager, [12](#)