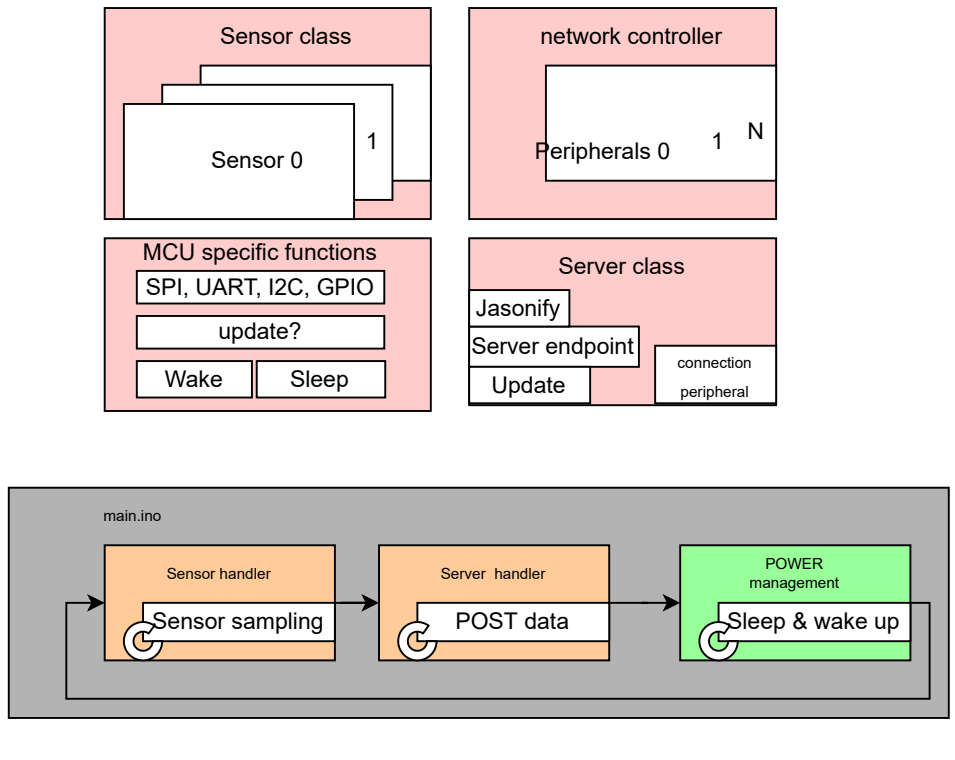
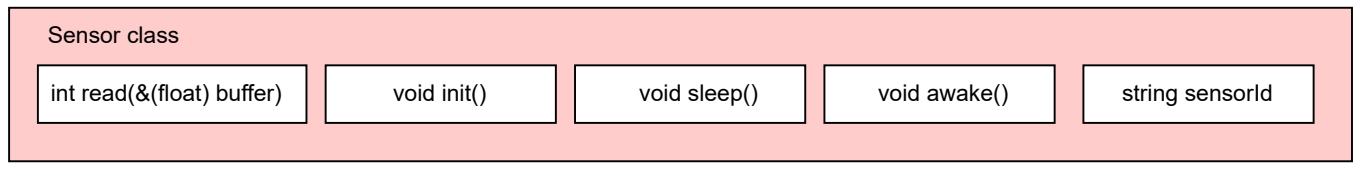
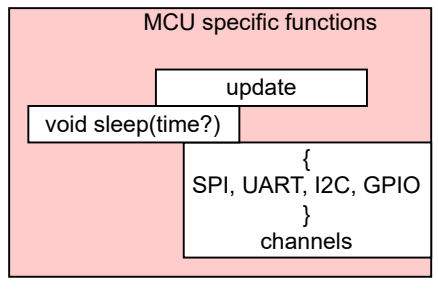

Sensor class
Sensor 0
1
network controller
Peripherals 0
1
N
MCU specific functions
SPI, UART, I2C, GPIO
update?
Wake
Sleep
Server class
Jasonify
Server endpoint
Update
connection
peripheral
main.ino
Sensor handler
Sensor sampling
Server handler
POST data
POWER
management
Sleep & wake up


Sensor class
int read(&(float) buffer)
void init()
void sleep()
void awake()
string sensorId


MCU specific functions
update
void sleep(time?)
{
SPI, UART, I2C, GPIO
}
channels

Sensor system declaration

sensor declaration

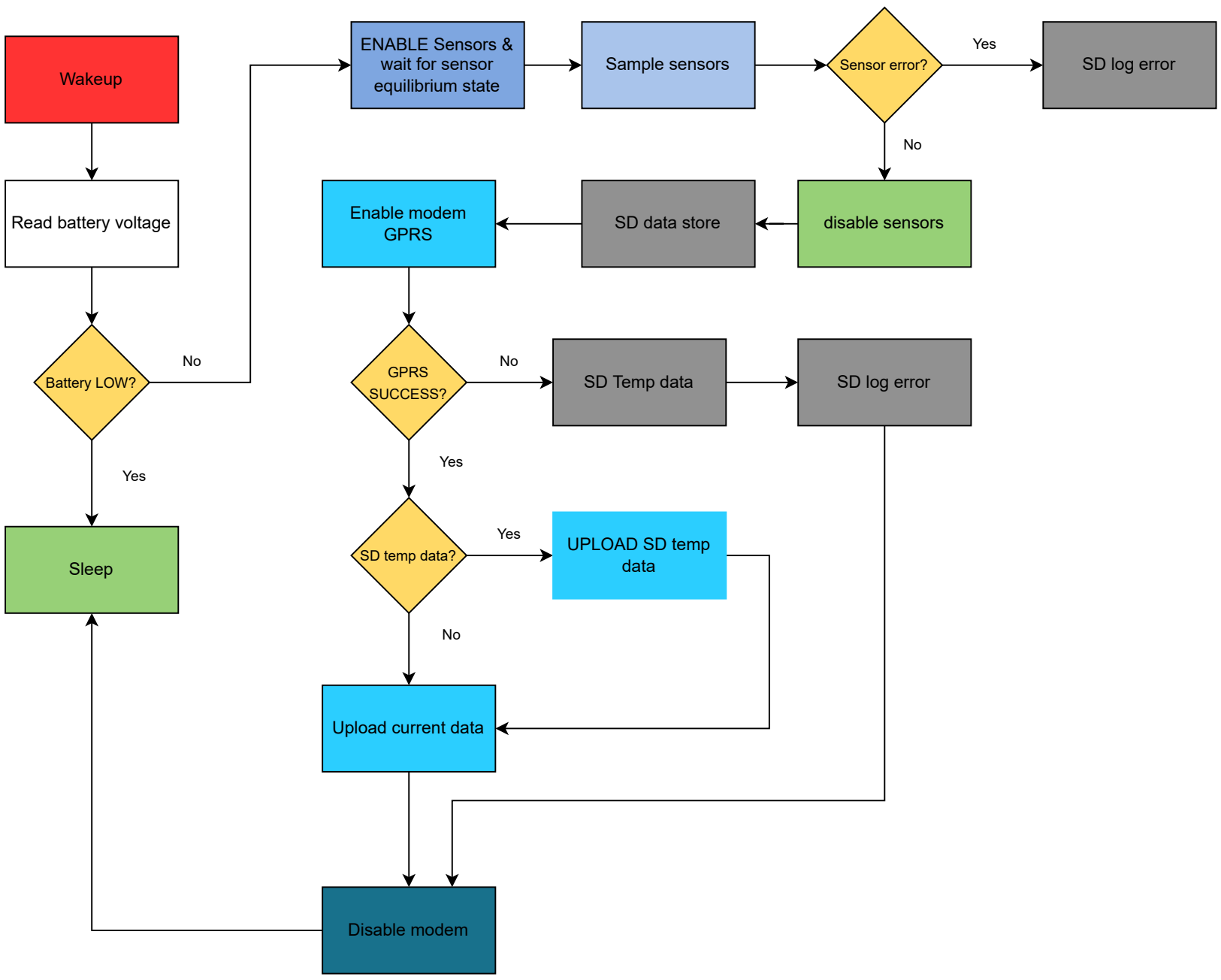
communication declaration

sleep time

server endpoint

1: {"moduleId": { "epoc": long,"sensorId(unit)": float, "sensorIdn(unit)": float, ...}}
2: {"module id": string, "epoc": long,"sensorId(unit)": float, "sensorIdn(unit)": float, ...}
3: {"moduleId": { "epoc":{ "val": long, "unit": epoc},"sensorId": { "val": float, "unit": Xunit}}}..}}

3: {"moduleId": [{"epoc":{ "val": long, "unit": epoc}},{"sensorId": { "val": float, "unit": Xunit},]}


Wakeup
Read battery voltage
Battery LOW?
No
Yes
Sleep
ENABLE Sensors & wait for sensor equilibrium state
Sample sensors
Sensor error?
Yes
SD log error
No
disable sensors
SD data store
Enable modem GPRS
GPRS SUCCESS?
No
SD Temp data
SD log error
Yes
SD temp data?
Yes
UPLOAD SD temp data
No
Upload current data
Disable modem

Sensor class attributes

enableSensor()
disableSensor()
(int) numberOfreadings
(String) sensorName[]
(String) samplesBuffer[]
getSamples()
calibrate()
(uLong) sensorStabilizeDelay
(String) unit[]
(int) status
(String) errorBuffer
(int)sensorStatus[]
begin()

Modem

enableModem()
disableModem()
getTime()
errorBuffer
status(error)
postData()
deviceName
begin()
establishConnection()
getTime()
dateTime

surfsideScienceStructs

success
error
connected

SurfsideScience

processSensors<>(T... x)
ErrorBuffer
enableSensors()
sensorStabilizeDelay
readSensor()
stopSensors()
postData(T com)
errorLog
payload
log
ReportErrorEndpoint()
generatePayload()

SD

begin()
sdStatus
errorBuffer
Storedata()
StoreErrorLog()
getTempData()

SleepFunction

Sleep

process sensors

begin sensors
Wait for sensr reach equilibrium

read sensors

Stop sensors

report_data(T modem, SDclass)

upload data

Wake up
begin connection
get time
upload data
Sleep

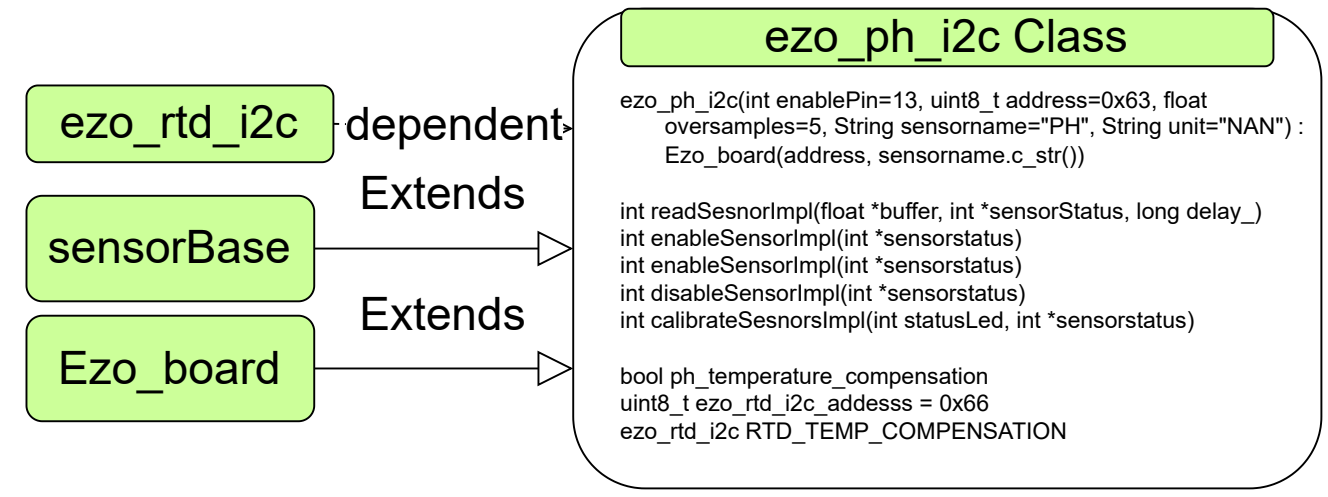
Log error if any
Set data to json array

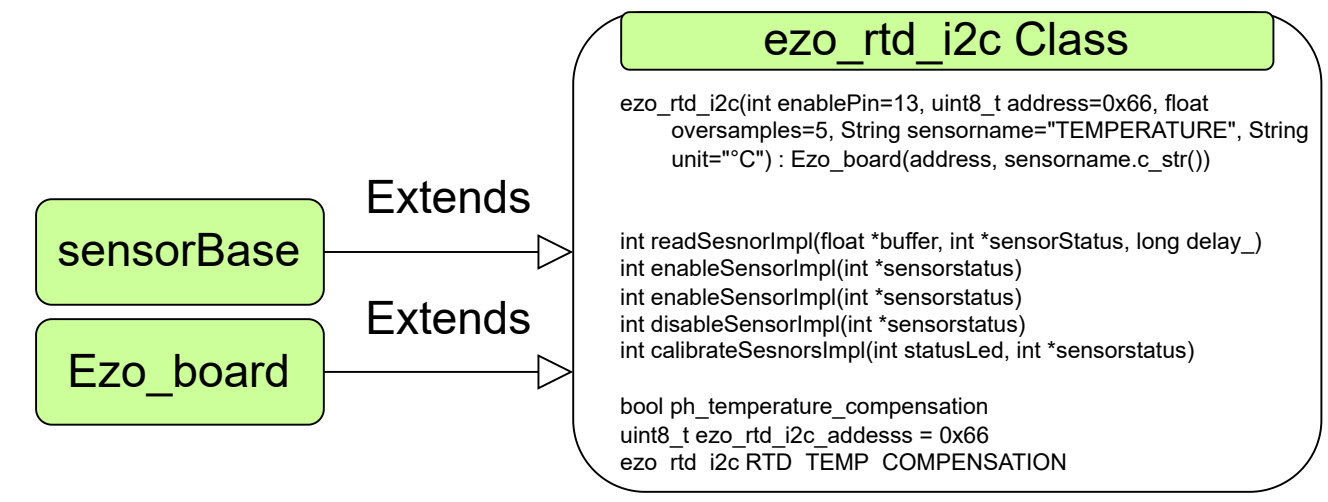
TinyGSM
ArduinoHttpClient
StreamDebugger

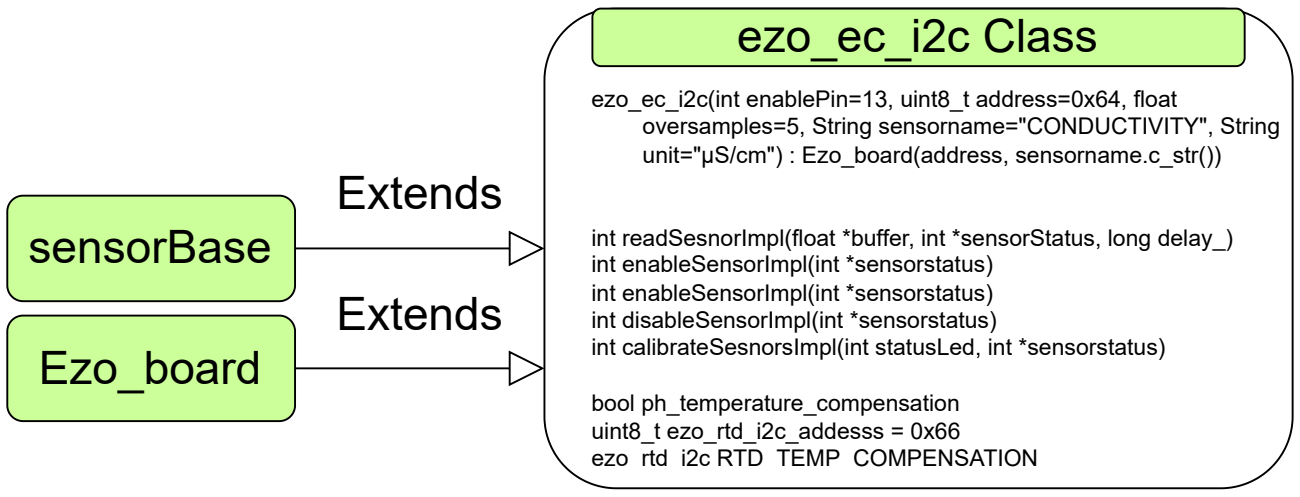
dependent

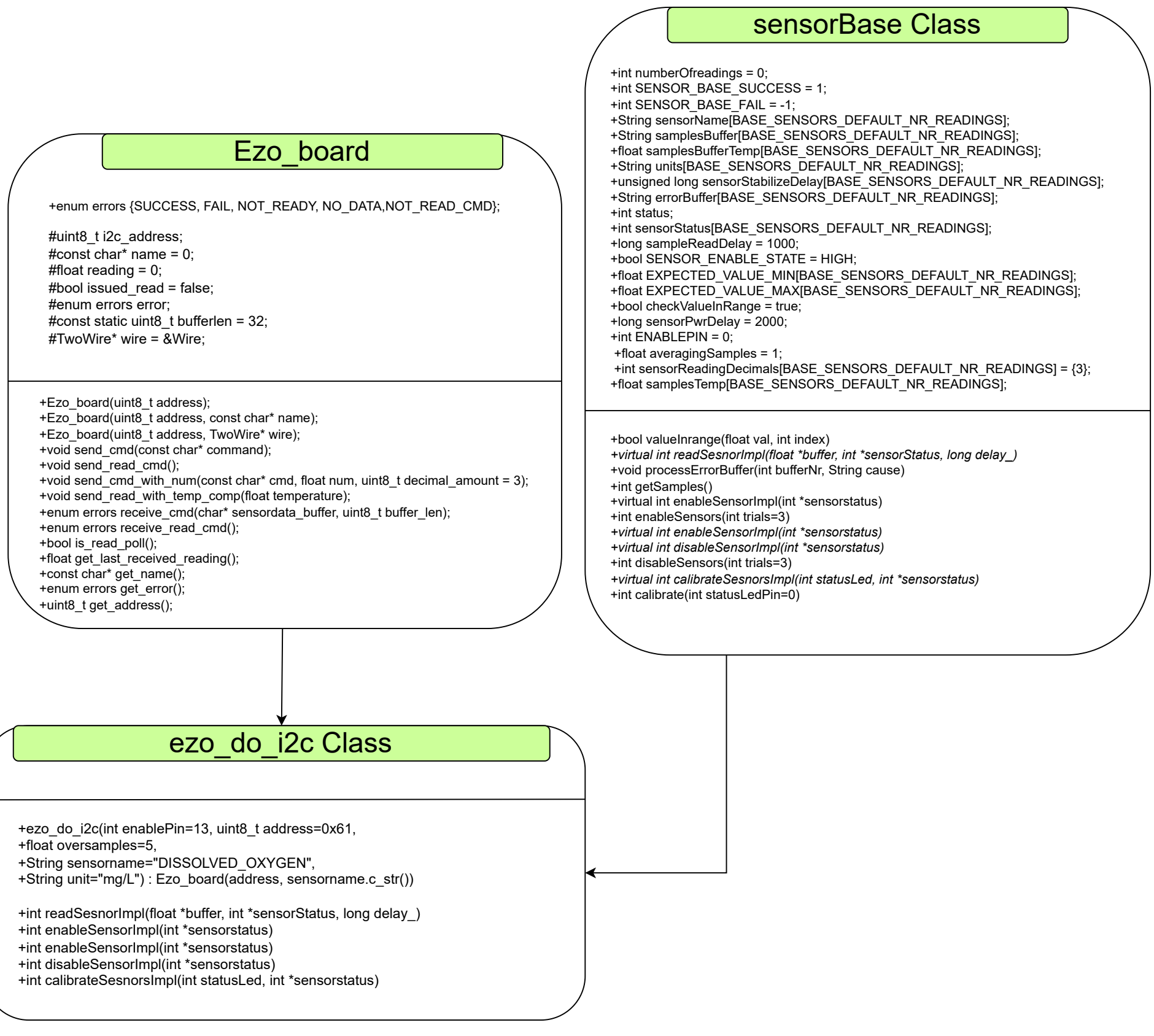
tinyGSMWrapper Class

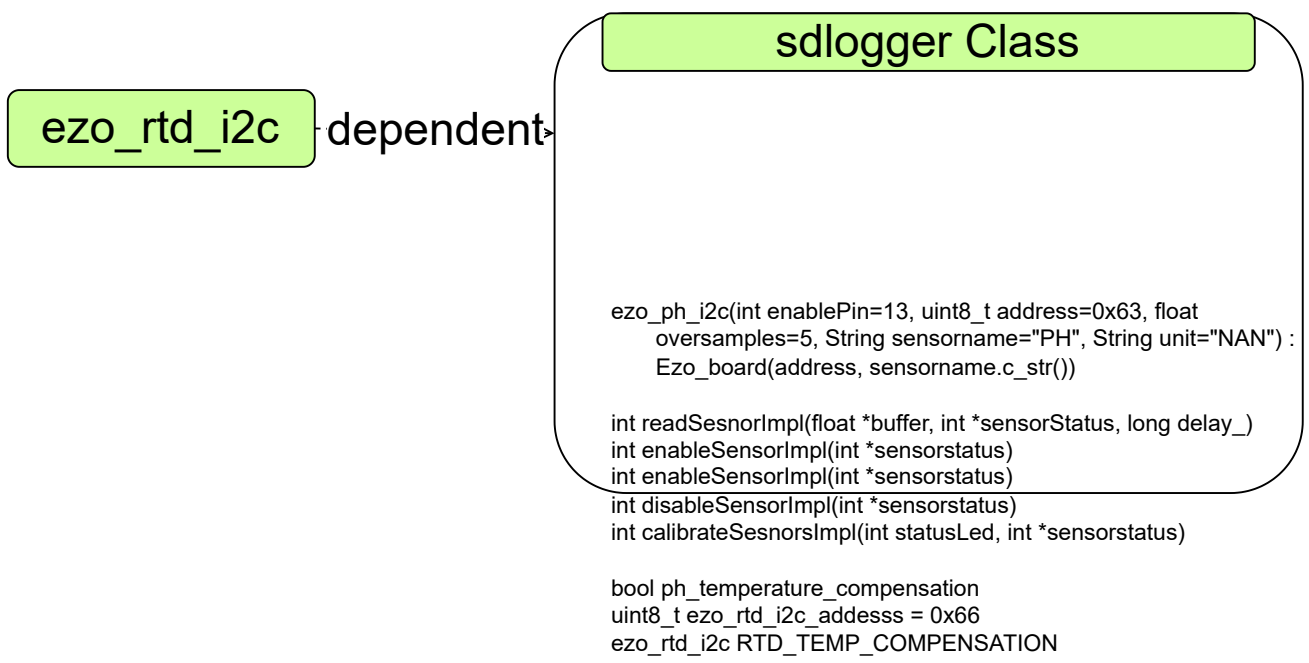
```
void begin(const char* apn="web.digicelaruba.com", const char* gprsuser="",
      const char* gprspass="", const char* server="surfside-db.brenchies.com",
      const char* postPath="/observations", long successCode=201,
      const char* contentType= "application/json", long uart_baud=115200,
      long pin_dtr=25, long pin_tx = 27,
      long pin_rx=26, long pin_pwr=4, String devicename="SIMCom SIM7000")

void processErrorBuffer(String cause)
int isModemAlive(bool response=1, int trials=5)
void sendPwrPulse(int delay_=1000, bool enable=true)
int enableModem(int trials=5)
int disableModem(int trials=20)
void getTime(int trials=3)
int getSignalQuality()
int establishConnection(int trials=3)
int postData(String payload, int trials=3)

String errorBuffer = "";
String deviceName = "";
int status = 1;
String dateTime = "";
long UART_BAUD = 115200;
long PIN_DTR  = 25;
long PIN_TX   = 27;
long PIN_RX   = 26;
long PIN_PWR  = 4;
long modemPwrdelay = 5000;
const char *APN   = "web.digicelaruba.com";
const char *GPRSUSER = "";
const char *GPRSPASS = "";
const char *GSMPIN = "";
bool gprsReady = false;
const char *SERVER = "surfside-db.brenchies.com";
const char *POSTPATH = "/observations";
const char *CONTENTTYPE = "application/json";
long PORT = 80;
long SUCCESSCODE = 201;
```


ezo_ph_i2c Class
ezo_rtd_i2c
dependent
Extends
sensorBase
Extends
Ezo_board
ezo_ph_i2c(int enablePin=13, uint8_t address=0x63, float oversamples=5, String sensorname="PH", String unit="NAN") : Ezo_board(address, sensorname.c_str())
int readSesnorImpl(float *buffer, int *sensorStatus, long delay_)
int enableSensorImpl(int *sensorstatus)
int enableSensorImpl(int *sensorstatus)
int disableSensorImpl(int *sensorstatus)
int calibrateSesnorsImpl(int statusLed, int *sensorstatus)
bool ph_temperature_compensation
uint8_t ezo_rtd_i2c_addesss = 0x66
ezo_rtd_i2c RTD_TEMP_COMPENSATION


ezo_rtd_i2c Class
sensorBase
Extends
Ezo_board
Extends
ezo_rtd_i2c(int enablePin=13, uint8_t address=0x66, float oversamples=5, String sensorname="TEMPERATURE", String unit="°C") : Ezo_board(address, sensorname.c_str())
int readSesnorImpl(float *buffer, int *sensorStatus, long delay_)
int enableSensorImpl(int *sensorstatus)
int enableSensorImpl(int *sensorstatus)
int disableSensorImpl(int *sensorstatus)
int calibrateSesnorsImpl(int statusLed, int *sensorstatus)
bool ph_temperature_compensation
uint8_t ezo_rtd_i2c_addesss = 0x66
ezo_rtd_i2c RTD_TEMP_COMPENSATION


ezo_ec_i2c Class
sensorBase
Extends
Ezo_board
Extends
ezo_ec_i2c(int enablePin=13, uint8_t address=0x64, float oversamples=5, String sensorname="CONDUCTIVITY", String unit="µS/cm") : Ezo_board(address, sensorname.c_str())
int readSesnorImpl(float *buffer, int *sensorStatus, long delay_)
int enableSensorImpl(int *sensorstatus)
int enableSensorImpl(int *sensorstatus)
int disableSensorImpl(int *sensorstatus)
int calibrateSesnorsImpl(int statusLed, int *sensorstatus)
bool ph_temperature_compensation
uint8_t ezo_rtd_i2c_addesss = 0x66
ezo_rtd_i2c RTD_TEMP_COMPENSATION


Ezo_board
+enum errors {SUCCESS, FAIL, NOT_READY, NO_DATA,NOT_READ_CMD};
#uint8_t i2c_address;
#const char* name = 0;
#float reading = 0;
#bool issued_read = false;
#enum errors error;
#const static uint8_t bufferlen = 32;
#TwoWire* wire = &Wire;
+Ezo_board(uint8_t address);
+Ezo_board(uint8_t address, const char* name);
+Ezo_board(uint8_t address, TwoWire* wire);
+void send_cmd(const char* command);
+void send_read_cmd();
+void send_cmd_with_num(const char* cmd, float num, uint8_t decimal_amount = 3);
+void send_read_with_temp_comp(float temperature);
+enum errors receive_cmd(char* sensordata_buffer, uint8_t buffer_len);
+enum errors receive_read_cmd();
+bool is_read_poll();
+float get_last_received_reading();
+const char* get_name();
+enum errors get_error();
+uint8_t get_address();
sensorBase Class
+int numberOfreadings = 0;
+int SENSOR_BASE_SUCCESS = 1;
+int SENSOR_BASE_FAIL = -1;
+String sensorName[BASE_SENSORS_DEFAULT_NR_READINGS];
+String samplesBuffer[BASE_SENSORS_DEFAULT_NR_READINGS];
+float samplesBufferTemp[BASE_SENSORS_DEFAULT_NR_READINGS];
+String units[BASE_SENSORS_DEFAULT_NR_READINGS];
+unsigned long sensorStabilizeDelay[BASE_SENSORS_DEFAULT_NR_READINGS];
+String errorBuffer[BASE_SENSORS_DEFAULT_NR_READINGS];
+int status;
+int sensorStatus[BASE_SENSORS_DEFAULT_NR_READINGS];
+long sampleReadDelay = 1000;
+bool SENSOR_ENABLE_STATE = HIGH;
+float EXPECTED_VALUE_MIN[BASE_SENSORS_DEFAULT_NR_READINGS];
+float EXPECTED_VALUE_MAX[BASE_SENSORS_DEFAULT_NR_READINGS];
+bool checkValueInRange = true;
+long sensorPwrDelay = 2000;
+int ENABLEPIN = 0;
+float averagingSamples = 1;
+int sensorReadingDecimals[BASE_SENSORS_DEFAULT_NR_READINGS] = {3};
+float samplesTemp[BASE_SENSORS_DEFAULT_NR_READINGS];
+bool valueInrange(float val, int index)
+virtual int readSesnorImpl(float *buffer, int *sensorStatus, long delay_)
+void processErrorBuffer(int bufferNr, String cause)
+int getSamples()
+virtual int enableSensorImpl(int *sensorstatus)
+int enableSensors(int trials=3)
+virtual int enableSensorImpl(int *sensorstatus)
+virtual int disableSensorImpl(int *sensorstatus)
+int disableSensors(int trials=3)
+virtual int calibrateSesnorsImpl(int statusLed, int *sensorstatus)
+int calibrate(int statusLedPin=0)
ezo_do_i2c Class
+ezo_do_i2c(int enablePin=13, uint8_t address=0x61,
+float oversamples=5,
+String sensorname="DISSOLVED_OXYGEN",
+String unit="mg/L") : Ezo_board(address, sensorname.c_str())
+int readSesnorImpl(float *buffer, int *sensorStatus, long delay_)
+int enableSensorImpl(int *sensorstatus)
+int enableSensorImpl(int *sensorstatus)
+int disableSensorImpl(int *sensorstatus)
+int calibrateSesnorsImpl(int statusLed, int *sensorstatus)


ezo_rtd_i2c
dependent
sdlogger Class
ezo_ph_i2c(int enablePin=13, uint8_t address=0x63, float
oversamples=5, String sensorname="PH", String unit="NAN") :
Ezo_board(address, sensorname.c_str())
int readSesnorImpl(float *buffer, int *sensorStatus, long delay_)
int enableSensorImpl(int *sensorstatus)
int enableSensorImpl(int *sensorstatus)
int disableSensorImpl(int *sensorstatus)
int calibrateSesnorsImpl(int statusLed, int *sensorstatus)
bool ph_temperature_compensation
uint8_t ezo_rtd_i2c_addesss = 0x66
ezo_rtd_i2c RTD_TEMP_COMPENSATION