```
00000000011111111112222222222333333333344444444445555555555666666666677777777781234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
//written for the Arduino uno or compatible
// Arduino default libraries
#include <Arduino.h> //GNU LGPL Arduino
#include <Wire.h> //GNU LGPL Arduino
#include <SPI.h> //needed for lorawan
//Altimiter Interface
#include <MS5611.h> //MIT License, Rob Tillaart
//Adafruit Sensor
#include <Adafruit_ICM20X.h> //BSD Liscence, Adafruit
#include <Adafruit_ICM20649.h> //BSD Liscence, Adafruit
#include <Adafruit_Sensor.h> //Apache Liscence, Adafruit
//Radio interfacing
#include <RH_RF95.h> //GPL Liscence, AirSpayce Ltd.
#include <RHEncryptedDriver.h> //security driver
#include <Speck.h> //encryption
#include <ArduinoJson.h>
//GPS
#include <SparkFun_u-blox_GNSS_Arduino_Library.h>
//project specific headers
#include "../include/megapins.h" //pin table
#include "../include/addr.h" //i2c addresses
#include "../include/v3.hpp" //vector library
#include "../include/altitude.h" //pressure -> altitude conversion
#include "../params.h" //flight parameter file
#include "../include/err.h" //error codes
#include "../include/utility.hpp" //util fxns

#define USBBAUD 9600
////////////////FUNCTIONS////////////////

//======== Debug functions
#if DEBUG == 1
#endif
////////////////CONTROL VARS////////////////
int doMode = 0;
/*
This allows for switching the operation preformed by the FC
*/
bool hasNotLaunched = true;
////////////////GPS OBJS////////////////
SFE_UBLOX_GNSS GPS;
uint32_t timeUNX;
long latitude;
long longitude;
long gpsAlt;
byte SIV;
////////////////RADIO OBJS & FXNS////////////////
RH_RF95 lora(rf_cs, rf_irq); //init of rfm95w
Speck cipher;
RHEncryptedDriver driver(lora,cipher);
const int msgLen = 251;
uint8_t data[msgLen+1];
uint8_t rbuf[RH_RF95_MAX_MESSAGE_LEN];
uint8_t rbuf_s;
StaticJsonDocument<msgLen> doc;
void forceSendError(uint8_t code) {
  doc.clear();
  doc["e"] = code;
  serializeJson(doc,data);
  Serial.println((char*)data);
  Serial.println();
  driver.send(data,256);
  #if DEBUG == 1
  Serial.print("ERROR: ");
  Serial.println(code);
  #endif
```

```
00000000011111111112222222222333333333344444444445555555555666666666677777777781
12345678901234567890123456789012345678901234567890123456789012345678901234567890
}
void sendMessage(StaticJsonDocument<msgLen> Document){
  uint8_t Dta[msgLen+1];
  serializeJson(Document,Dta);
  #if DEBUG == 1
  Serial.println((char*)Dta);
  Serial.println();
  #endif
  driver.send(Dta ,255);
  Document.clear();
}
///////////////SENSOR OBJS///////////////
MS5611 bar1(BAR1_AD);
MS5611 bar2(BAR2_AD);
Adafruit_ICM20649 icm;
uint16_t measurement_delay_us = measurementDel;
void readBarometers(float* pressure, float* temperature, float* altitude, float*
  float p1, t1, p2, t2, alt1, alt2;
  if(bar1.read() == 0){
    p1=bar1.getPressure();
    t1=bar1.getTemperature();
    alt1=barometric(p1,*p_orig,t2);
    if (isBetween(p1, 10, 1200) == false || isBetween(t1,-40, 85) == false) {
      forceSendError(ERR_BAR1_RANGE);
    }
  } else {
    forceSendError(ERR_BAR1_COMM);
  }
  if(bar2.read() == 0){
    p2=bar2.getPressure();
    t2=bar2.getTemperature();
    alt2=barometric(p2,*p_orig,t2);
    if (isBetween(p2, 10, 1200) == false || isBetween(t2,-40, 85) == false) {
      forceSendError(ERR_BAR2_RANGE);
    }
  } else {
    forceSendError(ERR_BAR2_COMM);
  }
  //averaging and assingment
  *pressure = (p1+p2)/2;
  *temperature = (t1+t2)/2;
  *altitude = (alt1+alt2)/2;
}

///////////////IMU CALC VARS///////////////
float imudel_s = 0.065535;
v3 lastacc, lastvel, pos, vel, acc = 0;
float p0, t0, alt0, alt, p, t = 0;
char tempMsg[msgLen];
//======== Application loop
void setup() {
  Wire.begin(); //start i2c
  #if DEBUG == 1
  Serial.begin(USBBAUD);
  Serial.println("Began Init");
  //while(!Serial) ; //wait for serial start
  #endif
  ///////////////IMU SETUP///////////////
  icm.begin_I2C(ADDR::ICM); //without this the chip wont start. lazy bastard
  icm.setAccelRange(ICM20649_ACCEL_RANGE_30_G); //set to 30G range
  icm.setGyroRange(ICM20649_GYRO_RANGE_2000_DPS); //2000 deg/sec range

  ///////////////LORWAN CONFIG///////////////
  /* This rather verbose method of resetting the radio comes from
  https://forum.arduino.cc/t/radiohead-library-and-rfm95-weird-init-issues/38905
  and was implemented here to combat some issues with the RH_RF::init().
```

```
00000000011111111112222222222333333333344444444445555555555666666666677777777781
12345678901234567890123456789012345678901234567890123456789012345678901234567890
  Turned out that the enable pin (EN) could not be connected to ground.
  Rather than remove this code, it was left here because there is no
  such thing as being too careful.
  */
  pinMode(4,OUTPUT);
  digitalWrite(4,LOW);
  delayMicroseconds(100);
  pinMode(4,INPUT);
  #if DEBUG==1
  Serial.println("LORA Reset");
  #endif
  delayMicroseconds(100);
  while(!lora.init()){
    delay(100);
    Serial.println("LORA has not initialized");
  }
  Serial.println("LORA Initialized, Coninuing");
  lora.setFrequency(freq); //freq from params.h
  lora.setTxPower(power); //power from params.h
  lora.setModemConfig(RH_RF95::ModemConfigChoice::Bw125Cr48Sf4096);
  cipher.setKey(key,16); //key from params.h
  ////////////////BAROMETER SETUP////////////////
  bool b1 = bar1.begin(); //start barometers
  bool b2 = bar2.begin();
  if (b1 == false) {forceSendError(ERR_BAR1_COMM);}
  if (b2 == false) {forceSendError(ERR_BAR2_COMM);}

  //Average averageCycles of baro readings to set p0, t0, and alt0
  for(size_t i = 0 ; i < averageCycles ; i++) {
    float *tempP, *tempT, *tempAlt;
    float *pAcc, *tAcc, *altAcc;
    readBarometers(tempP, tempT, tempAlt, &p0); //read
    *pAcc += *tempP;//accumulate
    *tAcc += *tempT;
    *altAcc += *tempAlt;
    p0 = *pAcc / averageCycles; //average
    t0 = *tAcc / averageCycles;
    alt0 = *altAcc / averageCycles;
  }
  //GPS Config
  bool g = GPS.begin();
  if(g == false){
    forceSendError(ERR_GPS_COMM);
  }
  GPS.setI2COutput(COM_TYPE_UBX);
  GPS.saveConfigSelective(VAL_CFG_SUBSEC_IOPORT);
  //INITIALIZATION MESSAGE
}
void loop() {
  if (hasNotLaunched == true) {
    float * lastAlt;
    while (1) {
      readBarometers(&p, &t, &alt, &p0);
      if ((alt - *lastAlt) > 5) {
        hasNotLaunched = false;
      }
      *lastAlt = alt;
    }
  }
  //Inertial/Barometric
  if (doMode == 1){
    //BAROMETERS
    readBarometers(&p, &t, &alt, &p0);
    doc["alt"] = alt; //! Add to paper
    doc["p"] = p;
    doc["t"] = t;
```

```
00000000011111111112222222222333333333344444444445555555555666666666677777777781
12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

```
    //IMU
    sensors_event_t accel, gyro, temp;
    icm.getEvent(&accel, &gyro, &temp);
    acc = v3(accel);
    vel = integrate(lastacc,acc,imudel_s);
    pos = integrate(lastvel, vel,imudel_s);
    lastacc = acc;
    lastvel = vel;
  }
  //gps
  if (doMode == 2) {
    latitude = GPS.getLatitude();
    longitude = GPS.getLongitude();
    timeUNX = GPS.getUnixEpoch();
    gpsAlt = GPS.getAltitudeMSL();
    doc["unx"] = timeUNX;
    doc["lat"] = latitude;
    doc["lon"] = longitude;
    doc["alt"] = gpsAlt;
    sendMessage(doc);
  }
  //////// RUN MODE SWITCHING ////////
  if(doMode == 2) {
    doMode = 1;
  } else {
    doMode += 1;
  } //*doMode can only be 0 at startup.


  #if DEBUG == 1
  #if DEBUGALT == 1
  if (0 != MS5611_READ_OK)
  {
    Serial.print("Error in read: ");
    Serial.println(0);
  }
  else
  {
    Serial.print("T:");
    Serial.print(t);
    Serial.print("\nP:");
    Serial.println(p);
  }
  #endif
  #if DEBUGIMU == 1
  acc.to_str("m/s2\n");
  vel.to_str("m/s\n");
  pos.to_str("m\n");
  #endif
  #endif
  digitalWrite(LED_BUILTIN, LOW);
}
```