Appendices

Arduino code

```
#include <LTask.h>
#include <LWiFi.h>
#include <LWiFiServer.h>
#include <LWiFiClient.h>
#include <LFlash.h>
#include <LSD.h>
#include <LStorage.h>

#include "Arduino.h"
#include "Wire.h"
#include <LGPS.h>
#include <math.h>
#include "LBattery.h"
#define Drv LFlash
LFile webFile;
#define WIFI_AP "mhmd"            // replace your WiFi AP SSID
//#define WIFI_PASSWORD ""  // replace your WiFi AP password
//#define WIFI_AUTH LWIFI_WEP
// choose from LWIFI_OPEN, LWIFI_WPA, or LWIFI_WEP according to your AP
```

```
#define REQ_BUF_SZ   20

char HTTP_req[REQ_BUF_SZ] = {0}; // buffered HTTP request stored as null terminated string

char req_index = 0;


#include <Wire.h>


const int buttonPin  = 5;

float temperature;

float pressure;

float atm;

float altitude;

float temper;

float hum;


int buttonState = 0;


LWiFiServer server(80);

int current_quality=-1;

int visibleLux;

int sensorValue;

float rSensor;

char buff[256];

char buff2[256]; // for gps

unsigned long starttime;

int ledPin = 4;

int callCount = 0;
```

```
void setup()
{

  LTask.begin();
  LWiFi.begin();
Drv.begin();
  LGPS.powerOn();
  Serial.begin(115200);
  delay(10000);



  // keep retrying until connected to AP
  Serial.println("Connecting to AP");
   while (0 == LWiFi.connect(WIFI_AP))
  {
    delay(1000);
  }
 pinMode(buttonPin, INPUT_PULLUP);
  printWifiStatus();
  Serial.println("Start Server");
  server.begin();
  Serial.println("Server Started");
}

int loopCount = 0;
String receivedInfo;
```

```
String gpsInformation;
void loop()
{
  // put your main code here, to run repeatedly:
  delay(500);
  loopCount++;
  LWiFiClient client = server.available();
  if (client)
  {
    // an http request ends with a blank line
    boolean currentLineIsBlank = true;
    while (client.connected())
    {
      if (client.available())
      {
        // we basically ignores client request, but wait for HTTP request end
        int c = client.read();
        Serial.print((char)c);
        receivedInfo+=(char)c;
              if (req_index < (REQ_BUF_SZ - 1)) {
                HTTP_req[req_index] = c;          // save HTTP request
character
                req_index++;
             }
        if (c == '\n' && currentLineIsBlank)
        {
          if (StrContains(HTTP_req, "GET /pic2.png")) {
```

```
            webFile = Drv.open("pic2.png");
            if (webFile) {
               client.println("HTTP/1.1 200 OK");
               client.println();
                while(webFile.available()) {
               client.write(webFile.read()); // send web page to client
            }
            webFile.close();
            }
         }
Serial.print("Incoming request ");
Serial.println(c);
Serial.println("Reading GPS Data");
gpsSentenceInfoStruct info;
LGPS.getData(&info);
Serial.println("Reading Battery Data");
int batteryLevel = LBattery.level();
bool batteryCharging = LBattery.isCharging()==1;
Serial.println("Reading Sensory  Data");

current_quality=analogRead(0);
//visibleLux = TSL2561.readVisibleLux();
sensorValue = analogRead(1);
rSensor=(float)(1023-sensorValue)*10/sensorValue;



Serial.print("QueryString contains simple");
```

```
        Serial.println(receivedInfo.indexOf("?simple")>0);


        if (receivedInfo.indexOf("?simple") >0){
          ShowSimpleStatusPage(batteryLevel, batteryCharging, info,
client);


        }
        else {
           ShowAdvancedStatusPage(batteryLevel, batteryCharging,info,
client );
        }
          req_index = 0;
                StrClear(HTTP_req, REQ_BUF_SZ);
        break;
      }
      if (c == '\n')
      {
        // you're starting a new line
        currentLineIsBlank = true;
      }
      else if (c != '\r')
      {
        // you've gotten a character on the current line
        currentLineIsBlank = false;
      }
    }
    else
```

```
    {
        // close the connection:
        Serial.println("close connection");
        client.stop();
        Serial.println("client disconnected");
    }
  }
  // give the web browser time to receive the data
  delay(500);
  callCount++;
  // close the connection:
  Serial.println("close connection");
  client.stop();
  Serial.println("client disconnected");
  receivedInfo = "";
 }
}


void ShowSimpleStatusPage(int batteryLevel, bool batteryCharging,
gpsSentenceInfoStruct info, LWiFiClient client)
{
 Serial.println("Showing Simple Page");
 Serial.println("send response");
 // send a standard http response header
 client.println("HTTP/1.1 200 OK");
 client.println("Content-Type: text/html");
```

```
  client.println("Connection: close");  // the connection will be closed after completion of the response
  client.println();
  client.println("<!DOCTYPE HTML>");
  client.println("<HTML>");
  client.println("<BODY>");
  client.print(temper);
  client.print(";");
  client.print(hum);
  client.print(";");
  client.print(temperature, 2); //display 2 decimal places
  client.print(";");
  client.print(pressure, 0); //whole number only.
  client.print(";");
  client.print(atm, 4); //display 4 decimal places
  client.print(";");
  client.print(altitude, 2); //display 2 decimal places
  client.print(";");
  client.print(String(batteryCharging));
  client.print(";");
  client.print(String(batteryLevel));
  client.print(";");
  client.print(callCount);
  client.print(";");
  client.print(loopCount);
  client.print(";");
  client.print((char*)info.GPGGA);
```

```
  client.print(";");
  //client.println(parseGPGGA((const char*)info.GPGGA));
  //,client));
  client.print(gpsInformation);
  client.print(";");
  client.print(current_quality);
  client.print(";");
    client.print(sensorValue);
  client.print(";");
  //parseGPGGA((const char*)info.GPGGA)),client)
  // output the value of each analog input pin
  client.println("Temperature1, Humidity1, Temperature2, Pressure, Atmosphere, Altitude, Charging, BatteryLevel, CallCount, LoopCount, GPS");
    client.println("</body>");
  client.println("</html>");
}

void ShowAdvancedStatusPage(int batteryLevel, bool batteryCharging, gpsSentenceInfoStruct info, LWiFiClient client)
{
  Serial.println("Showing Advanced Page");
  Serial.println("send response");
  // send a standard http response header
  client.println("HTTP/1.1 200 OK");
  client.println("Content-Type: text/html");
  client.println("Connection: close");  // the connection will be closed after completion of the response
```

```
  client.println("Refresh: 30");  // refresh the page automatically every 30 sec

  client.println();

  client.println("<!DOCTYPE HTML>");

  client.println("<HTML>");

  client.println("<HEAD>");


  client.println("<style> body { background-color: #b0c4de;}</style>");

  client.println("<meta name='apple-mobile-web-app-capable' content='yes' />");

  client.println("<meta name='apple-mobile-web-app-status-bar-style' content='black-translucent' />");

  client.println("<link rel='stylesheet' type='text/css' href='http://randomnerdtutorials.com/ethernetcss.css' />");

  client.println("<TITLE>fire figting system</TITLE>");

  client.println("</HEAD>");

  client.println("<BODY>");

  client.print("<H1>fire figting system</H1>");

  client.println("<HR />");

  client.println("<br />");

  client.println("<img src='http://www.ibiblio.org/hyperwar/USA/USA-A-Utah/maps/USA-A-Utah-20.jpg' alt='W3Schools.com' width='104' height='124' />");


  buttonState = digitalRead(buttonPin);


  Serial.println("buttonState is");

  Serial.println(buttonState);

  // output the value of each analog input pin
```

```
 if (buttonState == HIGH) {
  // turn LED on:


client.println("<br />");Serial.println("Start Server");
  client.print("<img src='http://www.ibiblio.org/hyperwar/USA/USA-A-Utah/maps/USA-A-Utah-20.jpg' alt='W3Schools.com' width='90' height='90' />");


}
else {
  // turn LED off:
client.println("<br />");


client.println("there is no fire in location 1");
Serial.println("Start Server");
}
client.print("Current Temperature is ");
client.print(temper);
client.print(" C ");
client.println("<br />");
client.print("Current Humidity is ");
client.print(hum);
client.print("%");
client.println("<br />");


client.println("<br />");
client.println("Trying Humidity Sensor Data");
```

```
client.println("<hr />");
client.print("Temperature: ");
client.print(temperature, 2); //display 2 decimal places
client.print("deg C");
client.println("<br />");
client.print("Pressure: ");
client.print(pressure, 0); //whole number only.
client.print(" Pa");
client.println("<br />");
client.print("Related Atmosphere: ");
client.print(atm, 4); //display 4 decimal places
client.println("<br />");
client.print("Altitude: ");
client.print(altitude, 2); //display 2 decimal places
client.print(" m");
client.println("<hr />");

client.println("<br />");
client.println("System data");
client.println("<hr />");
client.println("<br />");
client.print("Battery Charging: ");
client.print(String(batteryCharging));
client.println("<br />");
client.print("Battery: ");
client.print(String(batteryLevel));
client.print("Page has been requested ");
```

```
  client.print(callCount);
  client.print(" times with a loopcount of ");
  client.print(loopCount);
  client.println("<br />");


  client.println("<br />");
  client.println("GPS data");
  client.println("<hr />");
  client.println("<br />");
  client.print("GPS : ");
  client.print((char*)info.GPGGA);
  //parseGPGGA((const char*)info.GPGGA)),client)
  client.println("<br />");
  client.println("<br />");
  client.println("<h6>This page will auto refresh in 30 seconds, too
impatient? Refresh below.</h6>");
  client.println("<a href=\"\"\">Refresh</a>");
  client.println("</body>");
  client.println("</html>");
}



void parseGPGGA(const char* GPGGAstr)
{
  Serial.println("Processing GPS Data");
  /* Refer to http://www.gpsinformation.org/dale/nmea.htm#GGA
```

```
  * Sample data:
$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*4
7

  * Where:

  *  GGA          Global Positioning System Fix Data

  *  123519       Fix taken at 12:35:19 UTC

  *  4807.038,N   Latitude 48 deg 07.038' N

  *  01131.000,E  Longitude 11 deg 31.000' E

  *  1            Fix quality: 0 = invalid

  *                      1 = GPS fix (SPS)

  *                      2 = DGPS fix

  *                      3 = PPS fix

  *                      4 = Real Time Kinematic

  *                      5 = Float RTK

  *                      6 = estimated (dead reckoning) (2.3 feature)

  *                      7 = Manual input mode

  *                      8 = Simulation mode

  *  08           Number of satellites being tracked

  *  0.9          Horizontal dilution of position

  *  545.4,M      Altitude, Meters, above mean sea level

  *  46.9,M       Height of geoid (mean sea level) above WGS84

  *               ellipsoid

  *  (empty field) time in seconds since last DGPS update

  *  (empty field) DGPS station ID number

  *  *47          the checksum data, always begins with *

  */

 double latitude;
```

```
  double longitude;
  int tmp, hour, minute, second, num ;
  if(GPGGAstr[0] == '$')
  {
    tmp = getComma(1, GPGGAstr);
    hour     = (GPGGAstr[tmp + 0] - '0') * 10 + (GPGGAstr[tmp + 1] - '0');
    minute   = (GPGGAstr[tmp + 2] - '0') * 10 + (GPGGAstr[tmp + 3] - '0');
    second    = (GPGGAstr[tmp + 4] - '0') * 10 + (GPGGAstr[tmp + 5] - '0');

    sprintf(buff2, "UTC timer %2d-%2d-%2d", hour, minute, second);
    Serial.println(buff2);
    gpsInformation+= buff2;

    tmp = getComma(2, GPGGAstr);
    latitude = getDoubleNumber(&GPGGAstr[tmp]);
    tmp = getComma(4, GPGGAstr);
    longitude = getDoubleNumber(&GPGGAstr[tmp]);
    sprintf(buff2, "latitude = %10.4f, longitude = %10.4f", latitude, longitude);
    Serial.println(buff2);
    gpsInformation+= buff2;

    tmp = getComma(7, GPGGAstr);
    num = getIntNumber(&GPGGAstr[tmp]);
    sprintf(buff2, "satellites number = %d", num);
```

```
    Serial.println(buff2);
    gpsInformation+= buff2;
  }
  else
  {
    Serial.println("Error receiving GPS Data");
  }
}

static double getDoubleNumber(const char *s)
{
  char buf[10];
  unsigned char i;
  double rev;

  i=getComma(1, s);
  i = i - 1;
  strncpy(buf, s, i);
  buf[i] = 0;
  rev=atof(buf);
  return rev;
}

static unsigned char getComma(unsigned char num,const char *str)
{
  unsigned char i,j = 0;
  int len=strlen(str);
```

```
  for(i = 0;i < len;i ++)
  {
     if(str[i] == ',')
      j++;
     if(j == num)
      return i + 1;
  }
  return 0;
}

static double getIntNumber(const char *s)
{
  char buf[10];
  unsigned char i;
  double rev;

  i=getComma(1, s);
  i = i - 1;
  strncpy(buf, s, i);
  buf[i] = 0;
  rev=atoi(buf);
  return rev;
}

void printWifiStatus()
{
  // print the SSID of the network you're attached to:
```

```
  Serial.print("SSID: ");
  Serial.println(LWiFi.SSID());

  // print your WiFi shield's IP address:
  IPAddress ip = LWiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  Serial.print("subnet mask: ");
  Serial.println(LWiFi.subnetMask());

  Serial.print("gateway IP: ");
  Serial.println(LWiFi.gatewayIP());

  // print the received signal strength:
  long rssi = LWiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}
void StrClear(char *str, char length)
{
    for (int i = 0; i < length; i++) {
        str[i] = 0;
    }
}
char StrContains(char *str, char *sfind)
```

```
{
    char found = 0;
    char index = 0;
    char len;

    len = strlen(str);

    if (strlen(sfind) > len) {
        return 0;
    }
    while (index < len) {
        if (str[index] == sfind[found]) {
            found++;
            if (strlen(sfind) == found) {
                return 1;
            }
        }
        else {
            found = 0;
        }
        index++;
    }

    return 0;
}
```