```
/*  ESP-12 Toolbox
 *  Chris Gimson
 *  chris@gimson.co.uk
 18th December 2016 */

#include <DHT.h>   // for DHT temp/humidity sensor
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include "Gsender.h"  // for email function
#include <WiFiUdp.h>  // for NTP Time server

// Extra notes
// Highlighted text = parameters that you will need to change to suit your configuration

   // ************ Wifi NETWORK  ******************************
const char* ssid     = "HUAWEI-E5186-C0E1";   // WiFi network details
const char* password = "7GJGD5GA9AB";
const boolean WiFi_Connect_On_Demand_Global = true;  // = true = Wifi only connected when required, = false = Wifi on all of the time
int status = WL_IDLE_STATUS;     // the Wifi radio's status
WiFiClient client;

   // ************ THINGSPEAK   *********************************
const char* server = "api.thingspeak.com";
String API = "87Q3RB2EXNUUBY90";    // Your Thingspeak.com twitter account API key
String apiKey = "TMPHWJI2Q2QNMXXJ";   // Your thingspeak Data channel Write API key,

   // ************ GMAIL EMAIL ACCOUNT) ******************
// IPAddress server2( 1, 2, 3, 4 );
char server2[] = "smtpcorp.com";
int port = 2525;
// remember to set up your personal email server details in the Gsender.h tab


   // ************ NTP TIME SERVER   ******************
unsigned int localPort = 2390;      // local port to listen for UDP packets

IPAddress timeServerIP(132,163,4,101); // time-a.timefreq.bldrdoc.gov
          // Alternative time servers
// IPAddress timeServerIP(132,163,4,102); // time-b.timefreq.bldrdoc.gov
// IPAddress timeServerIP(132,163,4,103); // time-c.timefreq.bldrdoc.gov

const int NTP_PACKET_SIZE = 48; // NTP time stamp is in the first 48 bytes of the message
byte packetBuffer[ NTP_PACKET_SIZE]; //buffer to hold incoming and outgoing packets
const byte TimeZoneOffset = 2;      // This value corrects NTP time for your time zone. South Africa = +2
WiFiUDP udp;       // A UDP instance for sending and receiving packets over UDP

   // ************ DHT TEMP/HUMIDITY  ***********************
float Humidity;
float Temperature;
const byte DHTPIN = 2;    // The IO pin the DHT is connected to
DHT dht(DHTPIN, DHT11,16);   // Use this line if DHT11 used
// DHT dht(DHTPIN, DHT22,16);// Use this line if DHT22 used
```

```
// ************ POWER CONSUMPTION MONITOR **************************
float KiloWattHr = 0;      // Total power consumed Kilowatt hours
int RMSPower;              // Instantaqneous power Watts
float RMSCurrent;          // Instantaneous current Amps
const float CT_Scaler_Coefficient = 32.404;  // Calibration scaling factor for Current transformer
unsigned long PowerSampleInterval;  // Used for totalised power calculation
const byte LineVoltage = 220;     // set to the nominal power voltage in your country

   // ************ GENERAL IO  *********************************
const byte Heartbeat_LED = 13;    // Heartbeat LED IO pin number
const boolean LED_On = true;      // status of LED output pin when LED is ON

  // ************ ALARMS **********************************
const byte Alarm_Enable = 12;   // Alarm enable switch IO pin number
const byte Alarm_Input = 14;  // Input from PIR sensor IO pin number
const boolean AlarmOn = LOW;   // state of input pin when alarm is ON
const boolean AlarmEnabled = LOW;  // state of alarm enable pin when the alarm is enabled
const String Email_Alarm_To_EmailAddress = "chrisgimson@yahoo.co.uk";
const String Email_Alarm_Message_Subject_Line = "URGENT ALARM";
const String Email_Alarm_Message_Body = "The house alarm has been activated";
const String Tweet_Alarm_Message = "URGENT ALARM, The house alarm has been activated";
boolean Block_Alarm;      // true = block alarm sensing, false = respond to alarm contact
unsigned long Time_Alarm_detected;      // keeps track of time since alarm was triggered
const unsigned long Block_Alarm_Period = 180000;     // length that the alarm is blocked after initiation (mS)
```

```
// ********** Main Sketch ****************************
void setup()
{
  Serial.begin(115200);
  delay(10);
  dht.begin();  // remove if DHT sensor not used
  pinMode(Heartbeat_LED,OUTPUT);  // Heart beat LED
  digitalWrite(Heartbeat_LED,!LED_On);
  pinMode(Alarm_Input,INPUT_PULLUP);  // PIR or alarm actuator input - remove if not used
  pinMode(Alarm_Enable,INPUT_PULLUP); // Alarm Enable switch - remove if not used
  ConnectToWiFi();      // Connect to WiFi network
  udp.begin(localPort); // For NTP timer server
  printWifiData();      // Print the WiFi connection details
  if (WiFi_Connect_On_Demand_Global) WiFi.disconnect();;       // Disconnect from Wifi if the WiFi dieconnect on demand flag is
set
  PowerSampleInterval = millis();  // Take initial time stamp for KwH integration calculation
  Block_Alarm = false;         // enable alarms
}

void loop()     // loop to demonstrate the individual functions
{
  // Set up a test message for emailing & tweeting the measured parameters
  String Message =   "Current=" + String(RMSCurrent)
                   + " power=" + String(RMSPower)
                   + " kWh=" + String(KiloWattHr)
                   + " humidity=" + String(Humidity)
                   + " temp=" + String(Temperature);// time stamp is added later
  byte inChar;            // check for keyboard instruction
  inChar = Serial.read();
  if(inChar == 'e') SendEmail("chris@gimson.co.uk","Data packet from ESP-12", Message,WiFi_Connect_On_Demand_Global);
  if(inChar == 't') Send_Tweet(Message,WiFi_Connect_On_Demand_Global);  // Send a Twitter message
  if(inChar == 'd') Read_Sensor_Data_And_Upload_To_Thingspeak(WiFi_Connect_On_Demand_Global);  // Send data to
Thingspeak account
  if(inChar == 'h') GetTempHumidityReadings();   // Get temp & Humity readings only
   if(inChar == 'p') Read_CT_And_Calculate_Current_And_Power();  // Get electrical parameters
  if(inChar == 'T') Serial.println (Get_NTP_Time(WiFi_Connect_On_Demand_Global));
  digitalWrite(Heartbeat_LED,LED_On);   // Flash the heartbeat LED
  delay(150);
  digitalWrite(Heartbeat_LED,!LED_On);
  delay(150);
  Check_Alarm_And_Email_Tweet_If_Active(WiFi_Connect_On_Demand_Global);    // Simple check for alarm activation
}
    // ********** WiFi NETWORK FUNCTIONS  ***************

void ConnectToWiFi()   // Connect ESP8266 to local WiFi network
{
  Serial.println(); Serial.println("Connecting to Local WiFi Network");
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
  {             // wait for connection
    delay(500);
    Serial.print(".");
  }
  Serial.println("Connected");
  Serial.println("");
}
```

```
void printWifiData()   // Print all of the relevant details of your Wifi connection
{
  Serial.print(F("SSID: "));     // SSID
  Serial.println(WiFi.SSID());
  long rssi = WiFi.RSSI();       // print the received signal strength:
  Serial.print(F("signal strength (RSSI):"));
  Serial.println(rssi);
  IPAddress ip = WiFi.localIP();
    Serial.print(F("IP Address: ")); // IP address given to the connection
  Serial.println(ip);
  byte mac[6];                   // MAC address
  WiFi.macAddress(mac);
  Serial.print(F("MAC address: "));
  Serial.print(mac[5],HEX);
  Serial.print(F(":"));
  Serial.print(mac[4],HEX);
  Serial.print(F(":"));
  Serial.print(mac[3],HEX);
  Serial.print(F(":"));
  Serial.print(mac[2],HEX);
  Serial.print(F(":"));
  Serial.print(mac[1],HEX);
  Serial.print(F(":"));
  Serial.println(mac[0],HEX);
  IPAddress subnet = WiFi.subnetMask();   // Subnet mask
  Serial.print(F("NetMask: "));
  Serial.println(subnet);
  IPAddress gateway = WiFi.gatewayIP();   // Gateway address
  Serial.print(F("Gateway: "));
  Serial.println(gateway);
  Serial.println("");
}

   // ********** DHT / TEMPERATURE Functions  *********
void GetTempHumidityReadings()   // Single read of Temp & Humidity DHT sensor.
{
  if (isnan(Humidity) || isnan(Temperature))
  {
    Serial.println("Failed to read DHT sensor!");
    return;
  }
  Humidity = dht.readHumidity();
  Temperature = dht.readTemperature();
  Serial.print ("Temp 'C = "); Serial.print(Temperature); Serial.print ("  Humidity % = "); Serial.println(Humidity);
}       // end of GetTempHumidityReadings()
```

```
// ********** EMAIL FUNCTIONS ***************

void SendEmail(String EmailAddress, String subject, String Message, boolean WiFi_Connect_On_Demand)
        // Send email to specified address with specified subject line and message.
        // If WiFi_Connect_On_Demand = true, connect to Wifi first and disconnect at end
        // If WiFi_Connect_On_Demand = false, do not connect to Wifi, assumes that connection is already made
{
  boolean Status = false;
  digitalWrite(Heartbeat_LED,LED_On);
  if (WiFi_Connect_On_Demand) ConnectToWiFi();

  Message = Message + " Time=" + Get_NTP_Time(false);   // Add time stamp to the message

  if (client.connect(server,80))
  {
    Serial.println();
    Serial.print("Sending email to "); Serial.println(EmailAddress);
    Gsender *gsender = Gsender::Instance();    // Getting pointer to class instance
    if(gsender->Subject(subject)->Send(EmailAddress, Message))
    {
      Serial.println("Email successfully sent");
      Serial.println(subject);
      Serial.println(Message);
      Status = true;
    }
    else
    {
      Serial.print("Error sending message: ");
      Serial.println(gsender->getError());
    }
  }
  if (Status == false) Serial.println("failed to find server!");
  if (WiFi_Connect_On_Demand)
  {
    WiFi.disconnect();
    Serial.println("Disconnected from Network");  Serial.println();
  }
  digitalWrite(Heartbeat_LED,!LED_On);
}       // end of SendEmail
```

```
// ********** THINGSPEAK FUNCTIONS ***************
void Send_Tweet(String MessageBody, boolean WiFi_Connect_On_Demand)
        // Tweet the specified message to a Twitter account linked to ThingSpeak account
        // If WiFi_Connect_On_Demand = true, connect to Wifi first and disconnect at end
        // If WiFi_Connect_On_Demand = false, do not connect to Wifi (assumes that connection is already made)
{
  digitalWrite(Heartbeat_LED,LED_On);     // Set the Heartbeat LED
  if (WiFi_Connect_On_Demand) ConnectToWiFi();   // If enabled, switch WiFi ON
  boolean Status = false;
  delay(10);
  MessageBody = MessageBody + " Time=" + Get_NTP_Time(false);   // Add time stamp to the message
  if (client.connect(server,80))   // if the Wifi is connected
  {
    delay(10);
    Serial.println("Sending tweet -  ");
    Serial.print(MessageBody); Serial.println("  ");    // Print out the mesage being sent
    client.print("GET /apps/thingtweet/1/statuses/update?key="
         + API + "&status="
         + MessageBody
         + " HTTP/1.1\r\n");
    client.print("Host: api.thingspeak.com\r\n");
    client.print("Accept: */*\r\n");
    client.print("User-Agent: Mozilla/4.0 (compatible; esp8266 Lua; Windows NT 5.1)\r\n");
    client.print("\r\n");
    Serial.println("Message Sent");
    Status = true;
  }
  if (Status == false) Serial.println("failed to find server!");
  if (WiFi_Connect_On_Demand)
  {
    WiFi.disconnect();
    Serial.println("Disconnected from WiFi Network");
    Serial.println();
  }
  digitalWrite(Heartbeat_LED,!LED_On);    // Reset the heartbeat LED
}

void Read_Sensor_Data_And_Upload_To_Thingspeak(boolean WiFi_Connect_On_Demand)
        // Get & send upto 8 data parameters to ThingSpeak account
        // If WiFi_Connect_On_Demand = true, connect to Wifi first and disconnect at end
        // If WiFi_Connect_On_Demand = false, do not connect to Wifi, assumes that connection is already made
{
  digitalWrite(Heartbeat_LED,LED_On);     // SEt the Heartbeat LED
  GetTempHumidityReadings();               // Get the DHT data
  Read_CT_And_Calculate_Current_And_Power();  // Get power data
  boolean Status = false;
  if (WiFi_Connect_On_Demand) ConnectToWiFi();
  Serial.println("Sending Data to Thingspeak -  ");
  if (client.connect(server,80))
  { // "184.106.153.149" or api.thingspeak.com
    // Thingspeak data channel write API key, Upto 8 parameters per Thingspeak data channel
    String postStr = apiKey;
    postStr +="&field1=";           // Send Data Channel 1
    postStr += String(RMSCurrent);     // send relevant data parameter - change to suit

     //  ------------------------- Send Data Channel 2. Comment out if not required
```

```
  postStr +="&field2=";
  postStr += String(RMSPower);      // send relevant data parameter - change to suit

   //   ------------------------- Send Data Channel 3. Comment out if not required
  postStr +="&field3=";
  postStr += String(KiloWattHr);    // send relevant data parameter - change to suit

   //   ------------------------- Send Data Channel 4. Comment out if not required
  postStr +="&field4=";
  postStr += String(Temperature);  // send relevant data parameter - change to suit

   //   ------------------------- Send Data Channel 5. Comment out if not required
  postStr +="&field5=";
  postStr += String(Humidity);    // send relevant data parameter - change to suit

  //   ------------------------- Send Data Channel 6. Comment out if not required
  // postStr +="&field6=";
  // postStr += String(XXXXXXXX);    // send relevant data parameter - change to suit

  //   ------------------------- Send Data Channel 7. Comment out if not required
  // postStr +="&field7=";
  // postStr += String(XXXXXXXX);    // send relevant data parameter - change to suit

  //   ------------------------- Send Data Channel 8. Comment out if not required
  // postStr +="&field8=";
  // postStr += String(XXXXXXXX);    // send relevant data parameter - change to suit

  postStr += "\r\n\r\n";
  client.print("POST /update HTTP/1.1\n");
  client.print("Host: api.thingspeak.com\n");
  client.print("Connection: close\n");
  client.print("X-THINGSPEAKAPIKEY: "+apiKey +"\n");
  client.print("Content-Type: application/x-www-form-urlencoded\n");
  client.print("Content-Length: ");
  client.print(postStr.length());
  client.print("\n\n");
  client.print(postStr);
  Status = true;
 }
 if (Status == false) Serial.println("failed to find server!");
 Serial.print ("Data sent  ");
 PrintDataSet();
 if (WiFi_Connect_On_Demand)
 {
  WiFi.disconnect();
  Serial.println("Disconnected from Network");
 }
 digitalWrite(Heartbeat_LED,!LED_On);
}
```

```
// ********** ALARM FUNCTIONS **************

void Check_Alarm_And_Email_Tweet_If_Active(boolean WiFi_Connect_On_Demand)   // Check if alarm status and send email & Tweet if activated.
{       // If WiFi_Connect_On_Demand = true, connect to Wifi first and disconnect at end
        // If WiFi_Connect_On_Demand = false, do not connect to Wifi, assumes that connection is already made
 if (Check_For_Active_Alarm_1_Sec_Debounce())   // check to see if an alarm is present for 1 second
 {              // Alarm is activated
  if (WiFi_Connect_On_Demand) ConnectToWiFi();
  SendEmail(Email_Alarm_To_EmailAddress, Email_Alarm_Message_Subject_Line,Email_Alarm_Message_Body, false);
  Send_Tweet(Tweet_Alarm_Message,false);
  if (WiFi_Connect_On_Demand) WiFi.disconnect();
 }
}

boolean Check_For_Active_Alarm_1_Sec_Debounce()  // returns true if an enabled alarm is active for > 1 sec
{                                   // further alarm notifications are blocked for 3 minutes if alarm is detected
 byte Debounce = 0;
 boolean Status = false;
 if (Block_Alarm)
 {                  // Block alarm timer is running so alarm is blocked .... check for time out
  unsigned long Block_Interval = millis() - Time_Alarm_detected;  // re-calculate the alarm block timer
  if (Block_Interval > Block_Alarm_Period)  Block_Alarm = false; // If timeout, re-enable the alarm
 }
 else
 {          // Alarm block timer not active so check for alarm
  if(digitalRead(Alarm_Input) == AlarmOn && digitalRead(Alarm_Enable) == AlarmEnabled)
  {          // is alarm enabled and the alarm input active?
   Status = true;
   while (Debounce < 100)  // if the alarm is active, check that it is on for 1 second
   {
    if (digitalRead(Alarm_Input) == !AlarmOn) // checkif alarm is on or off
    {         // alarm is OFF
     Status = false;
     break;    // alarm not active for debounce period
    }
    delay(10);
    Debounce ++;  // alarm is ON therefore contine with debounce timer
   } // end of while(debounce .....
  }  // end of  if(digitalRead(Alarm_Input) ....
  if (Status) // check if alarm status is being returned as ON (true)
  {         // alarm is ON
   Block_Alarm = true;   // alarm is debounced and on - start block alarm timer
   Time_Alarm_detected = millis();  // reload block alarm timer for count down
  }
 }    // end of else
 return Status;
}    // end of Check_For_Active_Alarm_1_Sec_Debounce()
```

```
// ********** UTILITY FUNCTIONS **************
void PrintDataSet()     // Print out all measured parameters
{
  Serial.print("Current = "); Serial.print(RMSCurrent,1);
  Serial.print(" Power="); Serial.print(RMSPower,1);
  Serial.print(" kWh="); Serial.print(KiloWattHr,1);
  Serial.print(" Temp="); Serial.print(Temperature,1);
  Serial.print(" Hum="); Serial.println(Humidity,1);
  Serial.println();
}

 // *****************  POWER CALCULATIONS  *****************
void Read_CT_And_Calculate_Current_And_Power()
{
  int Current = 0;
  int MaxCurrent = 0;
  int MinCurrent = 1023;

  for (int i=0 ; i<=1000 ; i++)  //Monitors and logs the current input for 1000 samples to determine max and min current
  {
    Current = analogRead(A0);    //Reads current input and records maximum and minimum current
    if(Current > MaxCurrent) MaxCurrent = Current;
    else
    if(Current < MinCurrent) MinCurrent = Current;
  }
  unsigned int PeakCurrent = ((MaxCurrent - MinCurrent)/2.0);  // Peak = half of peak to peak
  // Serial.print ("Peak = "); Serial.println(PeakCurrent);

  // Serial.print("A/D = "); Serial.print(Current); Serial.print("   Peak I = "); Serial.println(PeakCurrent);
  // Serial.print("Maximum I = "); Serial.print(MaxCurrent), Serial.print(" :  Mininimum I = "); Serial.println(MinCurrent);
  // Serial.println();

  RMSCurrent = ((PeakCurrent)*0.707)/CT_Scaler_Coefficient;    //Calculates RMS current based on Peak value
  RMSPower = LineVoltage*RMSCurrent;    //Calculates RMS Power Assuming Voltage 220VAC, change to 110VAC accordingly

  PowerSampleInterval = millis()-PowerSampleInterval;   // Calculate time interval since last KwH calculation in mS
  float Time = PowerSampleInterval/3600000;         // integrating time in hours
  KiloWattHr = KiloWattHr + (RMSPower*Time/1000);    //Calculate kilowatt hours used
  PowerSampleInterval = millis();                 // save timestamp for next calculation
  PrintDataSet();
}
```

```
// ***************************** NTP TIMER SERVER FUNCTIONS ********************
String Get_NTP_Time(boolean WiFi_Connect_On_Demand)  // returns the NTP time as a string
                        // WiFi_Connect_On_Demand = true = connnect before getting data & disconnect afterwards
{
  String NTP_Time = "T_Error";
  if (WiFi_Connect_On_Demand) ConnectToWiFi();
  sendNTPpacket(timeServerIP); // send an NTP packet to the time server
                        // wait to see if a reply is available
  uint32_t beginWait = millis();
  int cb;
  delay(1500);
  while (millis() - beginWait < 2000)
  {
    cb = udp.parsePacket();
    delay(100);
    if (cb >= NTP_PACKET_SIZE);
    {
      Serial.println("Packet received");
            // Read the received data packet
      udp.read(packetBuffer, NTP_PACKET_SIZE); // read the packet into the buffer

    //the timestamp starts at byte 40 of the received packet and is four bytes,
    // or two words, long. First, extract the two words:

      unsigned long highWord = word(packetBuffer[40], packetBuffer[41]);
      unsigned long lowWord = word(packetBuffer[42], packetBuffer[43]);
    // combine the four bytes (two words) into a long integer
    // NTP time (seconds since Jan 1 1900):
      unsigned long secsSince1900 = highWord << 16 | lowWord;
    // Convert NTP time into GMT time & correct for time zone
      const unsigned long seventyYears = 2208988800UL;
      unsigned long epoch = secsSince1900 - seventyYears;    // subtract seventy years:

    NTP_Time = String(((epoch  % 86400L)/3600)+ TimeZoneOffset) + ":" ;
    if ( ((epoch % 3600) / 60) < 10 )
    {
      NTP_Time = NTP_Time + "0";   // The first 10 minutes of each hour needs a leading '0'
    }
    NTP_Time = NTP_Time + String((epoch  % 3600) / 60) + ":";
    if ( (epoch % 60) < 10 )      // The first 10 minutes of each hour needs a leading '0'
    {
      NTP_Time = NTP_Time + "0";
    }
    NTP_Time = NTP_Time + String((epoch % 60));
    // Serial.print("String value of time = "); Serial.println(NTP_Time);
    break;
    }     // end of if (cb >= NTP_Packet_Size);
  }
  if (WiFi_Connect_On_Demand)
  {
    WiFi.disconnect();
    Serial.println("Disconnected");  Serial.println();
  }
  return NTP_Time;
}
```

```
// send an NTP request to the time server at the given address
unsigned long sendNTPpacket(IPAddress& address)
{
  Serial.println("Requesting NTP Time data ...");
  memset(packetBuffer, 0, NTP_PACKET_SIZE);  // set all bytes in the buffer to 0
                            // Initialize values needed to form NTP request
  packetBuffer[0] = 0b11100011;   // LI, Version, Mode
  packetBuffer[1] = 0;     // Stratum, or type of clock
  packetBuffer[2] = 6;     // Polling Interval
  packetBuffer[3] = 0xEC;  // Peer Clock Precision
                        // 8 bytes of zero for Root Delay & Root Dispersion
  packetBuffer[12]  = 49;
  packetBuffer[13]  = 0x4E;
  packetBuffer[14]  = 49;
  packetBuffer[15]  = 52;
       // all NTP fields have been given values, now send a packet requesting a timestamp:
  udp.beginPacket(address, 123); //NTP requests are to port 123
  udp.write(packetBuffer, NTP_PACKET_SIZE);
  udp.endPacket();
}
```

```
// In the Gsender.cpp tab of the main sketch

#include "Gsender.h"
Gsender* Gsender::_instance = 0;
Gsender::Gsender(){}
Gsender* Gsender::Instance()
{
   if (_instance == 0)
      _instance = new Gsender;
   return _instance;
}

Gsender* Gsender::Subject(const char* subject)
{
 delete [] _subject;
 _subject = new char[strlen(subject)+1];
 strcpy(_subject, subject);
 return _instance;
}
Gsender* Gsender::Subject(const String &subject)
{
 return Subject(subject.c_str());
}

bool Gsender::AwaitSMTPResponse(WiFiClientSecure &client, const String &resp, uint16_t timeOut)
{
 uint32_t ts = millis();
 while (!client.available())
 {
  if(millis() > (ts + timeOut))
  {
   _error = "SMTP Response TIMEOUT!";
   return false;
  }
 }
 _serverResponce = client.readStringUntil('\n');
       // ***************************************************************************************
//Serial.println(_serverResponce);  // comment out if the full server responses is not needed to be printed
       // ***************************************************************************************
 if (resp && _serverResponce.indexOf(resp) == -1) return false;
 return true;
}

String Gsender::getLastResponce()
{
 return _serverResponce;
}

const char* Gsender::getError()
{
 return _error;
}
```

```
bool Gsender::Send(const String &to, const String &message)
{
  WiFiClientSecure client;

  if(!client.connect(SMTP_SERVER, SMTP_PORT))
  {
    _error = "Could not connect to mail server";
    return false;
  }
  if(!AwaitSMTPResponse(client, "220"))
  {
    _error = "Connection Error";
    return false;
  }

  client.println("HELO friend");
  if(!AwaitSMTPResponse(client, "250"))
  {
    _error = "identification error";
    return false;
  }

  client.println("AUTH LOGIN");
  AwaitSMTPResponse(client);
  client.println(EMAILBASE64_LOGIN);
  AwaitSMTPResponse(client);

  client.println(EMAILBASE64_PASSWORD);
  if (!AwaitSMTPResponse(client, "235"))
  {
    _error = "SMTP AUTH error";
    return false;
  }
  String mailFrom = "MAIL FROM: <" + String(FROM) + '>';
  client.println(mailFrom);
  AwaitSMTPResponse(client);
  String rcpt = "RCPT TO: <" + to + '>';
  client.println(rcpt);
  AwaitSMTPResponse(client);
  client.println("DATA");
  if(!AwaitSMTPResponse(client, "354"))
  {
    _error = "SMTP DATA error";
    return false;
  }

  client.println("From: <" + String(FROM) + '>');
  client.println("To: <" + to + '>');

  client.print("Subject: ");
  client.println(_subject);

  client.println("Mime-Version: 1.0");
  client.println("Content-Type: text/html; charset=\"UTF-8\"");
  client.println("Content-Transfer-Encoding: 7bit");
  client.println();
```

```
 String body = "<!DOCTYPE html><html lang=\"en\">" + message + "</html>";
 client.println(body);
 client.println(".");
 if (!AwaitSMTPResponse(client, "250"))
 {
  _error = "Sending message error";
  return false;
 }
 client.println("QUIT");
 if (!AwaitSMTPResponse(client, "221"))
 {
  _error = "SMTP QUIT error";
  return false;
 }
 return true;
}

// In the Gsender.h tab of the main sketch
//  Originally created by Boris Shobat

#include <WiFiClientSecure.h>
class Gsender
{
  protected:
    Gsender();
  private:
    const int SMTP_PORT = 465;
    const char* SMTP_SERVER = "smtp.gmail.com";

    // ***********  THE DETAILS OF YOUR SENDING GMAIL EMAIL ACCOUNT **************

    // go to https://www.base64encode.org/
    // in the top box type in exactly the gmail login username for your account & press "Encode" button
    // Cut or copy the result from the second box and past it below for the EMAILBASE64_LOGIN parameter
    // Repeat for the account password and past the result to the EMAILBASE64_PASSWORD

    const char* EMAILBASE64_LOGIN = "Y2hyaXNmbW1zb25AZ21hbWwuY29t";
    const char* EMAILBASE64_PASSWORD = "Y2hygXM2OYEy";

    // Enter the email address that you want to appear in the From field of the email (it does not need to be a gmail address)
    const char* FROM = "chrisgimson@gmail.com";
     // ********************************************************
    const char* _error = nullptr;
    char* _subject = nullptr;
    String _serverResponce;
    static Gsender* _instance;
    bool AwaitSMTPResponse(WiFiClientSecure &client, const String &resp = "", uint16_t timeOut = 10000);

  public:
    static Gsender* Instance();
    Gsender* Subject(const char* subject);
    Gsender* Subject(const String &subject);
    bool Send(const String &to, const String &message);
    String getLastResponce();
    const char* getError();
};
```