```
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
#include <SoftwareSerial.h>
#include "collectiveproje.h"

SoftwareSerial nmeaSourceA(4, 5);
  NMEA nmeaDecoder(ALL);
//uint8_t datasend[20];
static uint8_t mydata[10];//= {0x00, 0x00, 0x00};


float flat, flon,falt;

static const PROGMEM u1_t NWKSKEY[16] = { 0xEA, 0xF8, 0x44, 0x7E,
0x34, 0xAA, 0xEB, 0x97, 0x58, 0xCF, 0x11, 0x1F, 0xB0, 0xE8, 0x25,
0x14 };
//// LoRaWAN AppSKey, application session key
//// This is the default Semtech key, which is used by the
prototype TTN
//// network initially.
////ttn
static const u1_t PROGMEM APPSKEY[16] = { 0xA5, 0xA6, 0x5C, 0xE4,
0xDD, 0x77, 0x12, 0xF8, 0xA8, 0xB1, 0x23, 0xD6, 0x1B, 0x0B, 0xB0,
0xFB  };
////
//// LoRaWAN end-device address (DevAddr)
//// See http://thethingsnetwork.org/wiki/AddressSpace
//// ttn
static const u4_t DEVADDR = 0x26011B16;


// These callbacks are only used in over-the-air activation, so
they are
// left empty here (we cannot leave them out completely unless
// DISABLE_JOIN is set in config.h, otherwise the linker will
complain).
```

```
void Data()
{

  float windspeed,temperature,winddirection;

 while (nmeaSourceA.available())  {

    while(nmeaDecoder.decode(nmeaSourceA.read()))
    {
      char* t0 = nmeaDecoder.term(0);
      char* t1 = nmeaDecoder.term(1);
      char* t2 = nmeaDecoder.term(2);
      char* t3 = nmeaDecoder.term(3);
      char* t4 = nmeaDecoder.term(4);
      char* t5 = nmeaDecoder.term(5);
      char* t6 = nmeaDecoder.term(6);
      char* t7 = nmeaDecoder.term(7);
      char* t8 = nmeaDecoder.term(8);
      char* t9 = nmeaDecoder.term(9);
      Serial.println(nmeaDecoder.sentence());
      if(strcmp(t0,"WIXDR")){// Check if measure is available
        windspeed = atof(t3)*0.5144;  // knot to m/s
        winddirection=atof(t1);
        Serial.print("wind speed: ");
        Serial.println(windspeed);

        Serial.print("wind direction: ");
        Serial.println(winddirection);
        Serial.println(t3);
       }

       if(strcmp(t0,"IIMWV")){            // Check if measure is
available
          temperature = atof(t2);  // Tem oC
```

```
        Serial.print("temperature: ");
        Serial.println(temperature);
       Serial.println(t3);
      }
    Serial.println(nmeaDecoder.sentence());
     Serial.println(".......");
       delay(5000);
  }

  //Serial.println("[INFO]: End Decoding...");
 }



  // convert float to int
    int16_t Temp1=round(temperature*100);
    int16_t Wdir1=round(winddirection*100);
    int16_t w1=round(windspeed*100);
     //int16_t Ipv1=round(Ipv*100);
 // int16_t Vpv1=round(Vpv*100);

  // convert int to byte
  mydata[0]=highByte(Temp1);
  mydata[1]=lowByte(Temp1);
  mydata[2]=highByte(Wdir1);
  mydata[3]=lowByte(Wdir1);
  mydata[4]=highByte(w1);
  mydata[5]=lowByte(w1);

//
//     conversion float to string
//    affiche data
//
    Serial.print("temperature: ");
    Serial.println(Temp1);
    Serial.print("wind speed: ");
    Serial.println(w1);
```

```
     Serial.print("wind direction: ");
     Serial.println(Wdir1);


   // Serial.println(Temp1);
   // Serial.println(Wdir1);
    //Serial.println(w1);
   // Serial.println(Ipv1);
   //Serial.println(Vpv1);

 delay(1000);
}

void os_getArtEui (u1_t* buf) { }
void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }


static osjob_t initjob,sendjob,blinkjob;

/* Schedule TX every this many seconds (might become longer due to
duty
 cycle limitations).*/
const unsigned TX_INTERVAL = 1;

// Pin mapping
const lmic_pinmap lmic_pins = {
    .nss = 10,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 9,
    .dio = {2, 6, 7},
};

void do_send(osjob_t* j){
    // Check if there is not a current TX/RX job running
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println("OP_TXRXPEND, not sending");
```

```
    } else {
        //GPSRead();
        //GPSWrite();
       Data();
        // Prepare upstream data transmission at the next possible
time.
          //LMIC_setTxData2(1,datasend,sizeof(datasend)-1,0);
        LMIC_setTxData2(1, mydata, sizeof(mydata)-1, 0);
        Serial.println("Packet queued");
        //Serial.println("GPS");
//        Serial.print("LMIC.freq:");
//        Serial.println(LMIC.freq);
//        Serial.println("");
//        Serial.println("");
        Serial.println("Receive data:");

    }
    // Next TX is scheduled after TX_COMPLETE event.
}

void onEvent (ev_t ev) {
    Serial.print(os_getTime());
    Serial.print(": ");
    switch(ev) {
        case EV_SCAN_TIMEOUT:
            Serial.println(F("EV_SCAN_TIMEOUT"));
            break;
        case EV_BEACON_FOUND:
            Serial.println(F("EV_BEACON_FOUND"));
            break;
        case EV_BEACON_MISSED:
            Serial.println(F("EV_BEACON_MISSED"));
            break;
        case EV_BEACON_TRACKED:
            Serial.println(F("EV_BEACON_TRACKED"));
            break;
        case EV_JOINING:
```

```
            Serial.println(F("EV_JOINING"));
            break;
        case EV_JOINED:
            Serial.println(F("EV_JOINED"));
            break;
        case EV_RFU1:
            Serial.println(F("EV_RFU1"));
            break;
        case EV_JOIN_FAILED:
            Serial.println(F("EV_JOIN_FAILED"));
            break;
        case EV_REJOIN_FAILED:
            Serial.println(F("EV_REJOIN_FAILED"));
            break;
        case EV_TXCOMPLETE:
            Serial.println(F("EV_TXCOMPLETE (includes waiting for
RX windows)"));
            if (LMIC.txrxFlags & TXRX_ACK)
              Serial.println(F("Received ack"));
            if (LMIC.dataLen) {
              Serial.println(F("Received "));
              Serial.println(LMIC.dataLen);
              Serial.println(F(" bytes of payload"));
            }
            // Schedule next transmission
            os_setTimedCallback(&sendjob,
os_getTime()+sec2osticks(TX_INTERVAL), do_send);
            break;
        case EV_LOST_TSYNC:
            Serial.println(F("EV_LOST_TSYNC"));
            break;
        case EV_RESET:
            Serial.println(F("EV_RESET"));
            break;
        case EV_RXCOMPLETE:
            // data received in ping slot
            Serial.println(F("EV_RXCOMPLETE"));
```

```
            break;
        case EV_LINK_DEAD:
            Serial.println(F("EV_LINK_DEAD"));
            break;
        case EV_LINK_ALIVE:
            Serial.println(F("EV_LINK_ALIVE"));
            break;
         default:
            Serial.println(F("Unknown event"));
            break;
    }
}

void setup() {
     // initialize digital pin  as an output.
    //Serial.begin(57600);
   // Serial.println("Start");
  //  nmeaSourceA.begin(4800);

    Serial.begin(115200);
    // ss.begin(9600);
    nmeaSourceA.begin(4800);
   // while(!Serial);
    //Serial.println("---- ");
    Serial.println("Connect to TTN");
    #ifdef VCC_ENABLE
    // For Pinoccio Scout boards
    pinMode(VCC_ENABLE, OUTPUT);
    digitalWrite(VCC_ENABLE, HIGH);
    delay(1000);
    #endif

    // LMIC init
    os_init();
    // Reset the MAC state. Session and pending data transfers will
be discarded.
    LMIC_reset();
```

```
    /*LMIC_setClockError(MAX_CLOCK_ERROR * 1/100);
     Set static session parameters. Instead of dynamically
establishing a session
     by joining the network, precomputed session parameters are be
provided.*/
    #ifdef PROGMEM
    /* On AVR, these values are stored in flash and only copied to
RAM
       once. Copy them to a temporary buffer here, LMIC_setSession
will
       copy them into a buffer of its own again.*/
    uint8_t appskey[sizeof(APPSKEY)];
    uint8_t nwkskey[sizeof(NWKSKEY)];
    memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
    memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
    LMIC_setSession (0x1, DEVADDR, nwkskey, appskey);
    #else
    // If not running an AVR with PROGMEM, just use the arrays
directly
    LMIC_setSession (0x1, DEVADDR, NWKSKEY, APPSKEY);
    #endif

    // Disable link check validation
    LMIC_setLinkCheckMode(0);

    // TTN uses SF9 for its RX2 window.
    LMIC.dn2Dr = DR_SF9;



    // Set data rate and transmit power (note: txpow seems to be
ignored by the library)
    LMIC_setDrTxpow(DR_SF7,14);

    // Start job
    do_send(&sendjob);
}
```

```
//void NMEAWrite()
//{
//  /*Convert GPS data to format*/
//  //datastring1 +=dtostrf(flat, 0, 4, gps_lat);
//  //datastring2 +=dtostrf(flon, 0, 4, gps_lon);
//  //datastring3 +=dtostrf(falt, 0, 2, gps_alt);
//
//
//


  //strcpy(datasend,gps_lon); //the format of datasend is
longtitude,latitude,altitude
//  Serial.print("###########    ");
//  Serial.print("NO.");
//  Serial.print(count);
//  Serial.println("    ###########");
//  Serial.println("----");
//  Serial.print("[");
//  // Serial.print((char*)datasend);
//  Serial.print("]");
//  Serial.print("");
//  /*
//  for(int k = 0; k < 20;k++)
//  {
//  Serial.print("[");
//  Serial.print(datasend[k], HEX);
//  Serial.print("]");
//  }
//  Serial.println("");
//  Serial.println("");*/
//  count++;
 // }

  //int32_t lng = flat * 10000;
```

```
  //int32_t lat = flon * 10000;

   // datasend[0] = windspeed;
 // datasend[1] = lat >> 8;
  //datasend[2] = lat >> 16;

  //datasend[3] = lng;
  //datasend[4] = lng >> 8;
  //datasend[5] = lng >> 16;
  //smartdelay(1000);



//static void smartdelay(unsigned long ms)
//{
//  unsigned long start = millis();
// // do
//  //{
////    while (nmeaSourceA.available())
////    {
////      gps.encode(nmeaSourceA.read());
////    }
//  //}
//  while (millis() - start < ms);
//}


void loop() {                //  I HAVE CHANGED HERE
    os_runloop_once();
}
```