```
/*
 * Program to answer requests on the celestron AUX serial bus like the
 * GPS module from Celstron.
 *
 * Time is taken from a battery backed clock Module e.g. DS3231
 * A GPS position is taken from  EEPROM, so the device mimics a GPS Module
 *
 * Pin 8 and 9 are used for AUX bus Communication.
 *
 * RTC is connected via I2C interface (SCL and SDA pins)
 *
 * Via the serial port you can send commands directly to the device.
 * Try H for a list of commands
 *
 * Device can also be set through tge AUX bus.
 * It behaves like the RTC for set commands and
 * responds like the GPS if time or position is requested.
 *
 *
 */
#include "define.h"
#include <SoftwareSerial.h>
#include "RTClib.h"

// with this "define" we inhibit most of the serial line output
#define SILENT
#undef SILENT

#include <EEPROM.h>
#define EE_year_0 0 // EEPROM time of the last update
#define EE_year_1 1
#define EE_month  2
#define EE_day    3
#define EE_hour   4
#define EE_minute 5
#define EE_second 6

#define EE_lat_2  8 // EEPROM postion of the last GPS postion LAT
#define EE_lat_1  9
#define EE_lat_0  10

#define EE_lon_2  11 // EEPROM postion of the last GPS postion LON
#define EE_lon_1  12
#define EE_lon_0  13

#define EE_I_AM_GPS 16 // EEPROM positiom to store how device behaves
#define EE_I_AM_RTC 17

SoftwareSerial   celestron_aux(8,9);

const double GPS_MULT_FACTOR = 46603.37778;  // = 2^24 / 360

/* these switches determine how the devic ebehaves */
```

```
bool i_am_gps = false; // behave like the GPS device
bool i_am_rtc = false; // behave like the TZC device
bool talk = false; // verbose mode OFF as start

// Packet structure on Celetron AUX port
#define PK_MAX_LEN 12
unsigned char packet[PK_MAX_LEN];   // Is 12 enough?
enum pk_state { PREAMBLE_WAIT, LENGTH_WAIT, DATA, CKSUM, DONE, VALID };
enum pk_state pkstate;
int pklen;
int pkidx;
int16_t cksum_accumulator;

RTC_DS3231 rtc;

void setup()
{
  // serial set-up

  celestron_aux .begin(19200); // AUX bus runs at 19200

  Serial        .begin(9600);
  Serial.println(F("\n\n\nSnoop Celestron with RTC Ready!"));

  if (! rtc.begin()) {
    Serial.println(F("Couldn't find RTC"));
    Serial.flush();
    abort();
  };
  if (rtc.lostPower()) {
    Serial.println(F("RTC lost power, let's set the time!"));
    // When time needs to be set on a new device, or after a power loss,
the
    // following line sets the RTC to the date & time this sketch was
compiled
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
  };

// Celestron Code set-up
  pkstate = PREAMBLE_WAIT;
  pklen = 0;
  pkidx = 0;


#ifndef SILENT
  print_rtc();
  print_gps();
#endif

/* set the state of hte Device */
  i_am_gps = true;
//  i_am_gps = EEPROM.read(EE_I_AM_GPS);
  i_am_rtc = true;
```

```
//  i_am_rtc = EEPROM.read(EE_I_AM_RTC);

};


void loop()
{
  uint8_t c;
  char cc;
  int32_t lat, lng;
  uint8_t *latBytePtr, *lngBytePtr;

  if (Serial.available()) {
    String in = "B51.22774110000000";

    in=Serial.readStringUntil('\n');
    in[0]=toupper(in[0]);
#ifndef SILENT
        if(talk) {
          Serial.print(F("Received Command from Serial Port\n"));
          Serial.println(in);
        };
#endif
    if(in[0]=='H') print_help();
    if(in[0]=='V') {
      talk = not(talk);
      if (talk) Serial.println(F("Verbose mode ON"));
      else Serial.println(F("Verbose mode OFF"));
    }
    if(in[0]=='G') {
      i_am_gps = not(i_am_gps);
      if (i_am_gps) Serial.println(F("GPS mode ON"));
      else Serial.println(F("GPS mode OFF"));
    }
//    if(in[0]=='R') {
//      i_am_rtc = not(i_am_rtc);
//      if (i_am_rtc) Serial.println(F("RTC mode ON"));
//      else Serial.println(F("RTC mode OFF"));
//    }
    if(parse_lon(in)) Serial.print(F("Set Longitude\n"));
    if(parse_lat(in)) Serial.print(F("Set Latitude\n"));
    if(parse_tim(in)) Serial.print(F("Set Time: "));
    if(parse_dat(in)) Serial.print(F("Set Date: "));
    print_rtc();
    print_gps();
  }
  // Read and echo chars from celestron_aux
  while (celestron_aux.available()) {
    c = celestron_aux.read();
#ifndef SILENT
    if(talk) {
     if (c==0x3b) Serial.write('\n');
      Serial.print( c, HEX ); // for debugging purposes
```

```
      Serial.write(' ');
    };
#endif
    packet_decode(c);
  };

  // check if the packet is valid
  if (pkstate != VALID) return;

  if (packet[FROM]==DEV_GPS) { // packet was sent from GPS
    // check if packet is from GPS, then i will store the coordinates in
the EEPROM
#ifndef SILENT
    if(talk) {
    Serial.print(F("Found GPS Device on the AUX bus! "));
    };
#endif
    i_am_gps = false;

    switch (packet[TYP])
    {
      case GPS_GET_VER:
#ifndef SILENT
        if(talk) {
          Serial.print(F(" GPS sent: gps_get_ver - "));
          Serial.print(packet[4], HEX);
          Serial.println(packet[5], HEX);
        };
#endif
      break;
      case GPS_GET_LAT:
        EEPROM.update(EE_lat_0, packet[6]);
        EEPROM.update(EE_lat_1, packet[5]);
        EEPROM.update(EE_lat_2, packet[4]);
        i_am_gps = false;
        rtc_stamp();
      break;
      case GPS_GET_LONG:
        EEPROM.update(EE_lon_0, packet[6]);
        EEPROM.update(EE_lon_1, packet[5]);
        EEPROM.update(EE_lon_2, packet[4]);
        i_am_gps = false;
        rtc_stamp();
      break;
      case GPS_GET_TIME:
        i_am_gps = false;
      break;
      case GPS_GET_DATE:
        i_am_gps = false;
      break;
      case GPS_GET_YEAR:
        i_am_gps = false;
      break;
```

```
    }
  };


  // check if the packet is for me, then I will respond!
  if (packet[TO] != DEV_GPS && packet[TO] != DEV_RTC) { // only look for
GPS and RTC packets
    pkstate = 0;
    pkidx = 0;
    pklen = 0;
    return;
  }

  // YES - it's for me
#ifndef SILENT
        if(talk) {
          if(packet[TO] == DEV_GPS)
            Serial.print( F("- GPS - " )); // for debugging purposes
          if(packet[TO] == DEV_RTC)
            Serial.print(F( "- RTC - " )); // for debugging purposes
        };
#endif

// here we respond to the request
  DateTime now =rtc.now(); // read latest time from RTC

  uint8_t dest = packet[FROM], src = packet[TO];

  switch(packet[TYP])
  {
    case GPS_LINKED:
    case GPS_TIME_VALID:
      if(i_am_gps) {
        pk_send(dest, src, packet[TYP], 1); // I always have a fix, if RTC
behavs like GPS
#ifndef SILENT
        if(talk) {
          Serial.println(F(" RTC sent: gps_LINKED/Time_valid"));
        };
#endif
      }
      else {
#ifndef SILENT
        if(talk) {
          Serial.print(F(" GPS sent: gps_LINKED/Time_valid - "));
          Serial.println(packet[4], HEX);
        };
#endif
      };
    break;

    case GPS_GET_TIME:
      if(i_am_gps) {
```

```
        pk_send(dest, src, GPS_GET_TIME, now.hour()-1, now.minute(), now.
second());
//        pk_send(dest, src, GPS_GET_TIME, EEPROM.read(EE_hour), EEPROM.
read(EE_minute), EEPROM.read(EE_second));
#ifndef SILENT
        if(talk) {
          Serial.println(F(" RTC sent: gps_get_time"));
        };
#endif
      }
      else {
#ifndef SILENT
        if(talk) {
          Serial.print(F(" GPS sent: gps_get_time - "));
          Serial.print(packet[4], HEX);
          Serial.print(packet[5], HEX);
          Serial.println(packet[6], HEX);
        };
#endif
      };
    break;

    case GPS_GET_HW_VER:
      if(i_am_gps){
        pk_send(dest, src, GPS_GET_HW_VER, GPS_HW_VER);
#ifndef SILENT
      if (talk) {
        Serial.println(F(" RTC sent: gps_get_hw_ver"));
      };
#endif
      }
      else {
#ifndef SILENT
        if (talk) {
          Serial.print(F(" GPS sent: gps_get_hw_ver - "));
          Serial.println(packet[4], HEX);
        };
#endif
      }
    break;

    case GPS_GET_YEAR:
      if(i_am_gps){
        pk_send(dest, src, GPS_GET_YEAR, now.year()>>8, now.year()&0xff);
#ifndef SILENT
        if(talk) {
          Serial.println(F(" RTC sent: gps_get_year"));
        };
#endif
      }
      else {
#ifndef SILENT
        if(talk) {
```

```
          Serial.print(F(" GPS sent: gps_get_year - "));
          Serial.print(packet[4], HEX);
          Serial.println(packet[5], HEX);
        };
#endif
      };
    break;

    case GPS_GET_DATE:
      if (i_am_gps){
        pk_send(dest, src, GPS_GET_DATE, now.month(), now.day());
#ifndef SILENT
        if(talk) {
          Serial.println(F(" RTC sent: gps_get_date"));
        };
#endif
        }
      else {
#ifndef SILENT
        if(talk) {
          Serial.print(F(" GPS sent: gps_get_date - "));
          Serial.print(packet[4], HEX);
          Serial.println(packet[5], HEX);
        };
#endif
        };
    break;

    case GPS_GET_LAT:
      if(i_am_gps){
#ifndef SILENT
        if(talk) {
          Serial.println(F(" RTC sent: gps_get_lat"));
        };
#endif
        pk_send(dest, src, GPS_GET_LAT, EEPROM.read(EE_lat_2), EEPROM.
read(EE_lat_1), EEPROM.read(EE_lat_0));
      }
      else{
#ifndef SILENT
        if(talk) {
          Serial.print(F(" GPS sent: gps_get_lat - "));
          Serial.print(packet[4], HEX);
          Serial.print(packet[5], HEX);
          Serial.println(packet[6], HEX);
        };
#endif
      };
    break;

    case GPS_GET_LONG:
      if(i_am_gps){
#ifndef SILENT
```

```
        if(talk) {
          Serial.println(F(" RTC sent: gps_get_long"));
        };
#endif
        pk_send(dest, src, GPS_GET_LONG, EEPROM.read(EE_lon_2), EEPROM.
read(EE_lon_1), EEPROM.read(EE_lon_0));
      }
      else{
#ifndef SILENT
        if(talk) {
          Serial.print(F(" GPS sent: gps_get_long - "));
          Serial.print(packet[4], HEX);
          Serial.print(packet[5], HEX);
          Serial.println(packet[6], HEX);
        };
#endif
      };
    break;

    case GPS_GET_VER:
      if(i_am_gps){
        pk_send(dest, src, GPS_GET_VER, 0, 2);  // Version 0.2 instead of
0.1 for the real GPS
#ifndef SILENT
        if(talk) {
          Serial.println(F(" RTC sent: gps_get_ver"));
        };
#endif
      }
      else{
#ifndef SILENT
        if(talk) {
          Serial.print(F(" GPS sent: gps_get_ver - "));
          Serial.print(packet[4], HEX);
          Serial.println(packet[5], HEX);
        };
#endif
      }
    break;
    case RTC_SET_LAT:
#ifndef SILENT
        if(talk) {
          Serial.println(F(" RTC sent: rtc_set_lat - "));
        };
#endif
        EEPROM.update(EE_lat_0, packet[4]);
        EEPROM.update(EE_lat_1, packet[5]);
        EEPROM.update(EE_lat_2, packet[6]);
        rtc_stamp();
    break;
    case RTC_SET_LONG:
#ifndef SILENT
        if(talk) {
```

```
          Serial.println(F(" RTC sent: rtc_set_long - "));
        };
#endif
        EEPROM.update(EE_lon_0, packet[4]);
        EEPROM.update(EE_lon_1, packet[5]);
        EEPROM.update(EE_lon_2, packet[6]);
        rtc_stamp();
    break;
    case RTC_SET_DATE:
#ifndef SILENT
        if(talk) {
          Serial.println(F(" RTC sent: rtc_set_date - "));
        };
#endif
        rtc.adjust(DateTime(now.year(), packet[4], packet[5], now.hour(),
now.minute(), now.second()));
    break;
    case RTC_SET_YEAR:
#ifndef SILENT
        if(talk) {
          Serial.println(F(" RTC sent: rtc_set_year - "));
        };
#endif
        rtc.adjust(DateTime(packet[4]*256 + packet[5], now.month(), now.
day(), now.hour(), now.minute(), now.second()));
    break;
    case RTC_SET_TIME:
#ifndef SILENT
        if(talk) {
          Serial.println(F(" RTC sent: rtc_set_time - "));
        };
#endif
        rtc.adjust(DateTime(now.year(), now.month(), now.day(),
(int8_t)packet[4], (uint8_t)packet[5], (uint8_t)packet[6]));
    break;
  }

  pkstate = PREAMBLE_WAIT;
  pklen = 0;
  pkidx = 0;
}

void packet_decode(int8_t c)
{
  switch (pkstate)
  {
    case PREAMBLE_WAIT:
    if (c == 0x3b || c==0x76) {
      pkstate = LENGTH_WAIT;
    }
    break;

    case LENGTH_WAIT:
```

```
    if (c < PK_MAX_LEN) {
      pklen = c;
      packet[0] = c;
      pkidx = 1;
      pkstate = DATA;
    }
    else
      pkstate = PREAMBLE_WAIT;
    break;

    case DATA:
    packet[pkidx] = c;
    pkidx++;
    if (pkidx == pklen + 1)
      pkstate = CKSUM;
    break;

    case CKSUM:
    if (pk_checksum(c))
      pkstate = VALID;
    else
      pkstate = PREAMBLE_WAIT;
    break;
  }
}

bool pk_checksum(int8_t target)
{
  int sum = 0;
  for (int i = 0; i <= pklen; i++) sum += packet[i];
  int8_t chk = (-sum) & 0xff;
  return (target == chk);
}

inline void cksum_init()
{
  cksum_accumulator = 0;
}

inline void cksum_update(uint8_t b)
{
  cksum_accumulator += b;
}

inline int8_t cksum_final()
{
  return (-cksum_accumulator) & 0xff;
}

// Send a 1-byte response
void pk_send(uint8_t dest, uint8_t src, uint8_t id, uint8_t byte0)
{
  cksum_init();
```

```
  // Send preamble
  celestron_aux.write(0x3b);
  // Send length 4
  cksum_update(0x04);
  celestron_aux.write(0x04);
  // Send src
  cksum_update(src);
  celestron_aux.write((uint8_t)src);
  // Send dest
  cksum_update(dest);
  celestron_aux.write(dest);
  // Send message id
  cksum_update(id);
  celestron_aux.write(id);
  // Send byte0
  cksum_update(byte0);
  celestron_aux.write(byte0);
  // Send checksum
  celestron_aux.write(cksum_final());
}

// Send a 2-byte response
void pk_send(uint8_t dest, uint8_t src, uint8_t id, uint8_t byte0, uint8_t
byte1)
{
  cksum_init();
  // Send preamble
  celestron_aux.write(0x3b);
  // Send length 5
  cksum_update(0x05);
  celestron_aux.write(0x05);
  // Send src
  cksum_update(src);
  celestron_aux.write((uint8_t)src);
  // Send dest
  cksum_update(dest);
  celestron_aux.write(dest);
  // Send message id
  cksum_update(id);
  celestron_aux.write(id);
  // Send byte0
  cksum_update(byte0);
  celestron_aux.write(byte0);
  // Send byte1
  cksum_update(byte1);
  celestron_aux.write(byte1);
  // Send checksum
  celestron_aux.write(cksum_final());
}

// Send a 3-byte response
void pk_send(uint8_t dest, uint8_t src, uint8_t id, uint8_t byte0, uint8_t
byte1, uint8_t byte2)
```

```
{
  cksum_init();
  // Send preamble
  celestron_aux.write(0x3b);
  // Send length 6
  cksum_update(0x06);
  celestron_aux.write(0x06);
  // Send src
  cksum_update(src);
  celestron_aux.write((uint8_t)src);
  // Send dest
  cksum_update(dest);
  celestron_aux.write(dest);
  // Send message id
  cksum_update(id);
  celestron_aux.write(id);
  // Send byte0
  cksum_update(byte0);
  celestron_aux.write(byte0);
  // Send byte1
  cksum_update(byte1);
  celestron_aux.write(byte1);
  // Send byte2
  cksum_update(byte2);
  celestron_aux.write(byte2);
  // Send checksum
  celestron_aux.write(cksum_final());
}

int parse_lon(String in)
{
  float longi=0.0;
  long int lo;
#ifndef SILENT
  printf("%s\n", &in[1]);
#endif
  if (in[0] != 'L') return (0);
  longi = atof(&in[1]);
  if (abs(longi)>90.0 || longi<0.0) return(0);

/* here we set the EEPROM  */
  lo = longi * GPS_MULT_FACTOR;
  EEPROM.update(EE_lon_0, lo& 0xff);
  EEPROM.update(EE_lon_1, (lo>>8) & 0xff);
  EEPROM.update(EE_lon_2, (lo>>16) & 0xff);
  return(1);
};

int parse_lat(String in)
{
  float lati=0.0;
  long int la;
#ifndef SILENT
```

```
  printf("%s\n", &in[1]);
#endif
  if (in[0] != 'B') return (0);
  lati=atof(&in[1]);
  if (abs(lati)>360.0 || lati<0.0) return(0);

/* here we set the EEPROM  */
  la = lati * GPS_MULT_FACTOR;
  EEPROM.update(EE_lat_0, la& 0xff);
  EEPROM.update(EE_lat_1, (la>>8) & 0xff);
  EEPROM.update(EE_lat_2, (la>>16) & 0xff);

  return(1);
};

int parse_tim( String in)
{
  int hr, mi, sec;
  DateTime now =rtc.now(); // read latest time from RTC

  if (in[0] != 'T') return (0);
  sscanf(&in[1], "%d:%d:%d", &hr, &mi, &sec);
  if(hr>23||mi>59||sec>59) return(0);
  rtc.adjust(DateTime(now.year(), now.month(), now.day(), hr, mi, sec));
  return(1);
}

int parse_dat( String in)
{
  int dd, mm, yy;
  DateTime now =rtc.now(); // read latest time from RTC

  if (in[0] != 'D') return (0);
  sscanf(&in[1], "%d/%d/%d", &dd, &mm, &yy);
  if(yy < 2020||mm>12||dd>31)return(0);

  rtc.adjust(DateTime(yy, mm, dd, now.hour(), now.minute(), now.second()));
  return(1);
}

void print_help()
{
  Serial.print(F("\nGPS/RTC Version 1.0 compiled on ")); Serial.
println((__DATE__));
  Serial.println(F("Possible Commands are:"));
  Serial.println(F("Set time: \t\tT08:23:21"));
  Serial.println(F("Set date: \t\tD22/12/2020"));
  Serial.println(F("Set Longitude: \t\tL6.7734556")); delay(100);
  Serial.println(F("Set Latitude: \t\tB51.2277411"));
  Serial.println(F("Toggle \t Verbose: \tV"));
  Serial.println(F("Toggle \t GPS-mode: \tG"));
//  Serial.println(F("Toggle \t RTC-mode: \tR"));
  Serial.println(F("Show Help: \t\tH"));
```

```
}

// print RTC time
void print_rtc()
{
  DateTime now = rtc.now();
  Serial.print(F("\nDate/Time: "));
  Serial.print(now.year(), DEC);
  Serial.print('/');
  Serial.print(now.month(), DEC);
  Serial.print('/');
  Serial.print(now.day(), DEC);
  Serial.print(" ");
  Serial.print(now.hour(), DEC);
  Serial.print(':');
  Serial.print(now.minute(), DEC);
  Serial.print(':');
  Serial.print(now.second(), DEC);
  Serial.println();
}

// print RTC postion
void print_gps()
{
  double f_pos;
  unsigned long i_pos;
  char out[14];
//  unsigned long int lo, la;
//  la=EEPROM.read(EE_lat_2)*256*256|EEPROM.read(EE_lat_1)*256|EEPROM.
read(EE_lat_0);
//  lo=EEPROM.read(EE_lon_2)*256*256|EEPROM.read(EE_lon_1)*256|EEPROM.
read(EE_lon_0);
  Serial.print(F("Latitude:  0x"));
  Serial.print(EEPROM.read(EE_lat_2), HEX);
  Serial.print(EEPROM.read(EE_lat_1), HEX);
  Serial.print(EEPROM.read(EE_lat_0), HEX);

  i_pos = EEPROM.read(EE_lat_0) + (256*EEPROM.read(EE_lat_1)) +
(65536*EEPROM.read(EE_lat_2));
  f_pos = i_pos / GPS_MULT_FACTOR;
  dtostrf(f_pos, 12, 6, out);
  Serial.print(F(" = "));
  Serial.println(out);

  Serial.print(F("Longitude: 0x"));
  Serial.print(EEPROM.read(EE_lon_2), HEX);
  Serial.print(EEPROM.read(EE_lon_1), HEX);
  Serial.print(EEPROM.read(EE_lon_0), HEX);

  i_pos = EEPROM.read(EE_lon_0) + (256*EEPROM.read(EE_lon_1)) +
(65536*EEPROM.read(EE_lon_2));
  f_pos = i_pos / GPS_MULT_FACTOR;
  dtostrf(f_pos, 12, 6, out);
```

```
  Serial.print(F(" = "));
  Serial.println(out);
}
// put timestap into EEPROM
void rtc_stamp()
{
  DateTime now = rtc.now();
  EEPROM.update(EE_second, now.second());
  EEPROM.update(EE_minute, now.minute());
  EEPROM.update(EE_hour, now.hour());
  EEPROM.update(EE_day, now.day());
  EEPROM.update(EE_month, now.month());
  EEPROM.update(EE_year_0, (now.year()>>8));
  EEPROM.update(EE_year_1, (now.year()&0xff));
}
```