**SOURCE CODE:**

//A7\_Client.cpp

//Adapted from https://github.com/adafruit/Adafruit\_GPS/blob/master/Adafruit\_GPS.cpp

//Adafruit GPS library

//16 Nov 2016 Capstone Project Water Quality Monitoring System

#include "A7Client.h"

#define A7Client\_ENABLE\_DEBUG

#define A7Client\_ENABLE\_RESP\_DEBUG

SoftwareSerial\* serialA7 = NULL;

// how long are max NMEA lines to parse?

#define MAXLINELENGTH 100

// we double buffer: read one line in and leave one for the main program

volatile char line1[MAXLINELENGTH];

volatile char line2[MAXLINELENGTH];

// our index into filling the current line

volatile uint8\_t lineidx=0;

// pointers to the double buffers

volatile char\* currentline = line1;

volatile char\* lastline;

volatile boolean recvdflag = false;

volatile boolean inStandbyMode;

A7Client::A7Client(uint8\_t rx, uint8\_t tx, uint32\_t baudRate){

serialA7 = new SoftwareSerial(rx, tx);

serialA7->begin(baudRate);

}

bool A7Client::init(){

}

bool A7Client::checkResponse(const char\* resp, unsigned int timeout, unsigned int

chartimeout)

{

int len = strlen(resp);

int sum = 0;

unsigned long timerStart, prevChar; //prevChar is the time when the previous Char has

been read.

timerStart = millis();

prevChar = 0;

while (1) {

if (serialA7->available()) {

char c = serialA7->read();

//debug purpose

#ifdef A7Client\_ENABLE\_RESP\_DEBUG

Serial.print(c);

#endif

////

prevChar = millis();

sum = (c == resp[sum]) ? sum + 1 : 0;

if (sum == len)break;

} if ((unsigned long) (

millis() -

timerStart) >

timeout \*

1000UL) {

Serial.println(F("Timeout"));

return false;

}

//If interchar Timeout => return FALSE. So we can return sooner from this function.

if (((unsigned long) (millis() - prevChar) > chartimeout) && (prevChar != 0)) {

Serial.println(F("InterChar Timeout"));

return false;

}

}

serialA7->flush();

return true;

}

void A7Client::changeBaud(){

while (1) {

Serial.println(F("Trying to change baud rate at 115200"));//change baud rate at 115200

serialA7->write("AT+IPR=9600\r\n"); //need to change baud rate at every startup, default

is 115200kbps

serialA7->write("AT+IPR=9600\r\n");

serialA7->write("AT+IPR=9600\r\n");

serialA7->write("AT+IPR=9600\r\n");

serialA7->write("AT+IPR=9600\r\n");

serialA7->write("AT+IPR=9600\r\n");

serialA7->write("AT+IPR=9600\r\n");

serialA7->write("AT+IPR=9600\r\n");

if (checkResponse("\r\n", DEFAULT\_TIMEOUT, DEFAULT\_INTERCHAR\_TIMEOUT )) {

//we got a response!

serialA7->end();

serialA7->begin(9600);

break;

}

else {

//try changing the baud to 9600...

serialA7->end();

serialA7->begin(9600);

Serial.println(F("Trying to check baud rate at 9600"));

serialA7->write("AT\r\n"); //The first resposnd is almost always +CME ERROR:58

something due to the try at 115200 kbps.

if (checkResponse("\r\n", DEFAULT\_TIMEOUT, DEFAULT\_INTERCHAR\_TIMEOUT )) {

//we got a response!

break;

} else {

serialA7->write("AT\r\n");

if (checkResponse("OK\r\n", DEFAULT\_TIMEOUT, DEFAULT\_INTERCHAR\_TIMEOUT ))

{

break;

} else {

serialA7->flush();

serialA7->write("AT\r\n");

if (checkResponse("OK\r\n", DEFAULT\_TIMEOUT,

DEFAULT\_INTERCHAR\_TIMEOUT )) {

break;

}

}

}

//If there is no response, change it back to 115200 baud and repeat...

serialA7->end();

serialA7->begin(115200);

}

}

Serial.println(F("Baud changed to 9600"));

}

bool A7Client::startGPS(){

serialA7->write("AT+GPS=1\r\n");

if(checkResponse("OK\r\n", 15, DEFAULT\_INTERCHAR\_TIMEOUT )){ //15 seconds,

serialA7->write("AT+GPSRD=1\r\n");

if(checkResponse("OK\r\n", DEFAULT\_TIMEOUT, DEFAULT\_INTERCHAR\_TIMEOUT

)){

Serial.println(F("GPS Started"));

return true;

}else {

Serial.println(F("GPS Start Failed 1"));

return false;

}

}else {

Serial.println(F("GPS Start Failed 2"));

return false;

}

}

bool A7Client::readGPS(){

Serial.println(F("reading from GPS..."));

if (checkResponse("+GPSRD:$", DEFAULT\_TIMEOUT, DEFAULT\_INTERCHAR\_TIMEOUT

)){

Serial.println(F("After checkresponse?"));

lineidx = 0;

char c = 0;

// char buffer[20];

// while(serialA7->available()){

while(!recvdflag){

while(serialA7->available()){

// Serial.println(F("Reading..."));

c = serialA7->read();

Serial.print(c);

if (c == '\n') {

currentline[lineidx] = 0;

if (currentline == line1) {

currentline = line2;

lastline = line1;

} else {

currentline = line1;

lastline = line2;

}

lineidx = 0;

recvdflag = true;

Serial.println(F("Looks like array is READY!"));

// Serial.println("----");

// Serial.println((char \*)lastline);

// Serial.println("----");

return true;

}

// Serial.println(F(""));

currentline[lineidx++] = c;

if (lineidx >= MAXLINELENGTH){

lineidx = MAXLINELENGTH-1;

return false;

}

}

}

}

}

char\* A7Client::lastNMEA(void){

recvdflag = false;

return (char\*) lastline;

}

uint8\_t A7Client::parseHex(char c) {

if (c < '0')

return 0;

if (c <= '9')

return c - '0';

if (c < 'A')

return 0;

if (c <= 'F')

return (c - 'A')+10;

// if (c > 'F')

return 0;

}

bool A7Client::parse(char \*nmea) {

// do checksum check

// first look if we even have one

if (nmea[strlen(nmea)-4] == '\*') {

uint16\_t sum = parseHex(nmea[strlen(nmea)-3]) \* 16;

sum += parseHex(nmea[strlen(nmea)-2]);

// check checksum

//shouldn't we start from i=0?????

for (uint8\_t i=0; i < (strlen(nmea)-4); i++) {

// Serial.println(nmea[i]);

sum ^= nmea[i];

} if (

sum != 0) {

// bad checksum :(

Serial.print(F("Checksum = "));

// Serial.println(sum);

return false;

}

}else {

Serial.println(F("No \* delimiter"));

return false;

} int32\_t

degree;

long minutes;

char degreebuff[10];

// look for a few common sentences

if (strstr(nmea, "GPGGA")) {

// found GGA

char \*p = nmea;

// get time

p = strchr(p, ',')+1;

float timef = atof(p);

uint32\_t time = timef;

hour = time / 10000;

minute = (time % 10000) / 100;

seconds = (time % 100);

milliseconds = fmod(timef, 1.0) \* 1000;

// parse out latitude

p = strchr(p, ',')+1;

if (',' != \*p)

{

strncpy(degreebuff, p, 2);

p += 2;

degreebuff[2] = '\0';

degree = atol(degreebuff) \* 10000000;

strncpy(degreebuff, p, 2); // minutes

p += 3; // skip decimal point

strncpy(degreebuff + 2, p, 4);

degreebuff[6] = '\0';

minutes = 50 \* atol(degreebuff) / 3;

latitude\_fixed = degree + minutes;

latitude = degree / 100000 + minutes \* 0.000006F;

latitudeDegrees = (latitude-100\*int(latitude/100))/60.0;

latitudeDegrees += int(latitude/100);

}

p = strchr(p, ',')+1;

if (',' != \*p)

{

if (p[0] == 'S') latitudeDegrees \*= -1.0;

if (p[0] == 'N') lat = 'N';

else if (p[0] == 'S') lat = 'S';

else if (p[0] == ',') lat = 0;

else return false;

}

// parse out longitude

p = strchr(p, ',')+1;

if (',' != \*p)

{

strncpy(degreebuff, p, 3);

p += 3;

degreebuff[3] = '\0';

degree = atol(degreebuff) \* 10000000;

strncpy(degreebuff, p, 2); // minutes

p += 3; // skip decimal point

strncpy(degreebuff + 2, p, 4);

degreebuff[6] = '\0';

minutes = 50 \* atol(degreebuff) / 3;

longitude\_fixed = degree + minutes;

longitude = degree / 100000 + minutes \* 0.000006F;

longitudeDegrees = (longitude-100\*int(longitude/100))/60.0;

longitudeDegrees += int(longitude/100);

}

p = strchr(p, ',')+1;

if (',' != \*p)

{

if (p[0] == 'W') longitudeDegrees \*= -1.0;

if (p[0] == 'W') lon = 'W';

else if (p[0] == 'E') lon = 'E';

else if (p[0] == ',') lon = 0;

else return false;

}

p = strchr(p, ',')+1;

if (',' != \*p)

{

fixquality = atoi(p);

}

p = strchr(p, ',')+1;

if (',' != \*p)

{

satellites = atoi(p);

}

p = strchr(p, ',')+1;

if (',' != \*p)

{

HDOP = atof(p);

}

p = strchr(p, ',')+1;

if (',' != \*p)

{

altitude = atof(p);

}

p = strchr(p, ',')+1;

p = strchr(p, ',')+1;

if (',' != \*p)

{

geoidheight = atof(p);

}

Serial.println(F("Completed conversion!"));

return true;

}

return false;

}

int A7Client::connect(IPAddress ip, uint16\_t port){

char host[16] = {0};

snprintf(host, 15, "%d.%d.%d.%d", ip[3], ip[2], ip[1], ip[0]);

return connect(host, port);

}

int A7Client::connect(const char \*host, uint16\_t port){

serialA7->write("AT+CIPSTART=\"TCP\",\"");

serialA7->write(host);

serialA7->write("\",");

serialA7->print(port); //Too lazy to solve this uint16\_t into uint8\_t problem

serialA7->write("\r\n");

Serial.println(F("Opening TCP via AT"));

if(checkResponse("CONNECT OK\r\n", DEFAULT\_TIMEOUT,

DEFAULT\_INTERCHAR\_TIMEOUT )){

serialA7->flush();

A7\_DEBUG("Connected");

return true;

}else {

A7\_DEBUG("Connect failed");

return false;

}

}

void A7Client::A7\_DEBUG(char \*msg){

#ifdef A7Client\_ENABLE\_DEBUG

Serial.print(F("Debug: "));

Serial.println(msg);

#endif

}

size\_t A7Client::write(uint8\_t b){

return write(&b, 1);

}

size\_t A7Client::write(char \*buf){

if (buf == NULL) return 0;

return write((const uint8\_t \*)buf, strlen(buf));

}

size\_t A7Client::write(const uint8\_t \*buf, size\_t size){

serialA7->write("AT+CIPSEND\r\n");

if(checkResponse(">", DEFAULT\_TIMEOUT, DEFAULT\_INTERCHAR\_TIMEOUT )){

A7\_DEBUG(">shown, sending bytes");

serialA7->write(buf, size);

//serialA7->write(0x0D);

//serialA7->write(0x0A);

//serialA7->write(0x0D);

//serialA7->write(0x0A);

serialA7->write(0x1A); //required on A7 module to signal the end of command

if(checkResponse("OK\r\n", DEFAULT\_TIMEOUT, DEFAULT\_INTERCHAR\_TIMEOUT

)){

A7\_DEBUG("Send succeeded");

return size;

}

}

}

size\_t A7Client::writeSerial(uint8\_t b){ //send command directly thorugh serial

return writeSerial(&b, 1);

}

size\_t A7Client::writeSerial(char \*buf){

if (buf == NULL) return 0;

return writeSerial((const uint8\_t \*)buf, strlen(buf));

}

size\_t A7Client::writeSerial(const uint8\_t \*buf, size\_t size){

return serialA7->write(buf, size);

}

int A7Client::available(){

return serialA7->available();

}

int A7Client::read(){

return serialA7->read();

}

int A7Client::read(uint8\_t \*buf, size\_t size){

return size;

}

void A7Client::flush(){

serialA7->flush();

}

int A7Client::peek(){

serialA7->peek();

}

void A7Client::stop(){

if (connected()){

serialA7->write("AT+CIPCLOSE\r\n");

if(checkResponse("OK\r\n", DEFAULT\_TIMEOUT, DEFAULT\_INTERCHAR\_TIMEOUT

)){

A7\_DEBUG("Close succeeded");

}else{ A7\_DEBUG("Close failed");

}

}else{ A7\_DEBUG("Nothing to be closed");

}

}

uint8\_t A7Client::connected(){

serialA7->write("AT+CIPSTATUS\r\n");

if(checkResponse("CONNECT OK \r\n", DEFAULT\_TIMEOUT,

DEFAULT\_INTERCHAR\_TIMEOUT )){

serialA7->flush();

A7\_DEBUG("Connected");

return 1;

}else{ A7\_DEBUG("Not connected");

return 0;

}

}

A7Client::operator bool(){

return (connected()==1);

}