Serial Logs:

Problem highlighted in Red colored. The device restarts here and then loses all connections.

It reboots, checks signal quality, try to connect GPS. After this restarts again.

The source code is copied at the end of logs

10:40:01.361 -> Open the GNSS Function Fali!

10:40:01.605 ->

10:40:01.605 -> -> AT+QGPS?

10:40:01.605 -> AT+QGPS?

10:40:01.605 -> <- +QGPS: 1

10:40:01.605 -> <-

10:40:01.605 -> <- OK

10:40:01.605 -> The GNSS Function is Opened!

10:40:01.697 ->

10:40:01.697 -> -> AT+QGPSEND

10:40:01.697 -> AT+QGPSEND

10:40:01.697 -> <- OK

10:40:01.809 -> -> AT+QGPS=1

10:40:01.809 -> AT+QGPS=1

10:40:01.809 -> <- OKmax\_employee\_number500

10:40:01.853 -> max\_job\_code500

10:40:01.853 -> max\_cost\_code500

10:40:01.853 -> max\_run\_hours1000

0:01.853 -> job\_code\_digits4

10:40:01.853 -> cost\_code\_digits3

10:40:01.853 -> run\_hours\_digits4

10:40:01.916 ->

10:40:01.916 -> -> AT+CPIN?

10:40:01.916 -> AT+CPIN?

10:40:01.916 -> <- +CPIN: READY

10:40:02.053 -> -> AT+CGREG?

10:40:02.053 -> AT+CGREG?

10:40:02.053 -> <- +CGREG: 0,1

10:40:02.053 -> <-

10:40:02.053 -> <- OK

10:40:02.145 -> -> AT+CSQ

10:40:02.145 -> AT+CSQ

10:40:02.145 -> <- +CSQ: 25,99

10:40:02.145 -> <-

10:40:02.145 -> <- OK

10:40:02.145 -> Device network registered and signal quality good...

10:40:02.145 ->

10:40:02.273 ->

10:40:02.273 -> -> ATE0

10:40:02.273 ->

10:40:02.273 -> -> ATE0

10:40:02.273 -> ATE0

10:40:02.273 -> <- OK

10:40:02.398 -> -> AT+QIDEACT=1

10:40:02.398 ->

10:40:02.398 -> <- OK

10:40:02.490 -> -> ATI

10:40:02.490 ->

10:40:02.490 -> <- Quectel

10:40:02.490 -> <- EC21

10:40:02.490 -> <- Revision: EC21AUFAR06A04M4G

10:40:02.490 -> <-

10:40:02.490 -> <- OK

10:40:02.490 -> Quectel

10:40:02.490 -> EC21

10:40:02.490 -> Revision: EC21AUFAR06A04M4G

10:40:02.598 ->

10:40:02.598 -> -> AT+QGMR

10:40:02.598 ->

10:40:02.598 -> <- EC21AUFAR06A04M4G\_01.001.01.001

10:40:02.598 -> <-

10:40:02.598 -> <- OK

10:40:02.598 -> EC21AUFAR06A04M4G\_01.001.01.001

10:40:02.714 ->

10:40:02.714 -> -> AT+CPIN?

10:40:02.714 ->

10:40:02.714 -> <- +CPIN: READY

10:40:02.791 -> -> AT+CGREG?

10:40:02.714 -> <- +CPIN: READY

10:40:02.791 -> -> AT+CGREG?

10:40:02.791 ->

10:40:02.791 -> <- +CGREG: 0,1

10:40:02.791 -> <-

10:40:02.791 -> <- OK

10:40:02.915 -> -> AT+QICSGP=1,1,"","","",3

10:40:02.915 ->

10:40:02.915 -> <- OK

10:40:03.012 -> -> AT+QIACT=1

10:40:03.012 ->

10:40:03.012 -> <- OK

10:40:03.128 -> -> AT+CGPADDR=1

10:40:03.128 ->

10:40:03.128 -> <- +CGPADDR: 1,"100.110.156.40"

10:40:03.128 -> <-

10:40:03.128 -> <- OK

10:40:03.128 -> Stack smashing protect failure!

10:40:03.128 ->

10:40:03.128 -> abort() was called at PC 0x400ecebc on core 1

10:40:03.128 ->

10:40:03.128 -> ELF file SHA256: 0000000000000000

10:40:03.128 ->

10:40:03.128 -> Backtrace: 0x400857f0:0x3ffb1e30 0x40085a65:0x3ffb1e50 0x400ecebc:0x3ffb1e70 0x400d7253:0x3ffb1e90 0x400d275d:0x3ffb1ef0 0x400d5328:0x3ffb1f80 0x400db221:0x3ffb1fb0 0x40086a75:0x3ffb1fd0

10:40:03.160 ->

10:40:03.160 -> Rebooting...

10:40:03.160 -> ets Jun 8 2016 00:22:57

10:40:03.160 ->

10:40:03.160 -> configsip: 0, SPIWP:0xee

10:40:03.160 -> clk\_drv:0x00,q\_drv:0x00,d\_drv:0x00,cs0\_drv:0x00,hd\_drv:0x00,wp\_drv:0x00

10:40:03.160 -> mode:DIO, clock div:1

10:40:03.160 -> load:0x3fff0018,len:4

10:40:03.204 -> load:0x3fff001c,len:1216

10:40:03.204 -> ho 0 tail 12 room 4

10:40:03.204 -> load:0x40078000,len:10944

10:40:03.204 -> load:0x40080400,len:6388

10:40:03.204 -> entry 0x400806b4

10:40:05.448 -> No memory detected. Freezing.

10:40:05.448 -> Memory detected!

10:40:05.448 -> size;

10:40:05.448 -> 80

10:40:05.448 -> Mem size in bytes: 65536

10:40:06.574 ->

10:40:06.574 -> -> AT+QGPS=1

10:40:06.574 ->

10:40:06.574 -> <-

10:40:06.574 -> <- +CME ERROR: 504

10:40:06.574 -> <-

10:40:06.697 -> Open the GNSS Function Fali!

10:40:06.913 ->

0:40:06.913 ->

10:40:06.913 -> <- +QGPS: 1

10:40:06.913 -> <-

10:40:06.913 -> <- OK

10:40:06.913 -> The GNSS Function is Opened!

10:40:07.025 ->

10:40:07.025 -> -> AT+QGPSEND

10:40:07.025 ->

10:40:07.025 -> <- OK

**Arduino Code:**

#include "WisLTEBG96MQTT.h"

#include <ArduinoJson.h>

#include <Adafruit\_Sensor.h>

#include <Adafruit\_ADXL345\_U.h>

#include <Keypad.h>

#include <LiquidCrystal\_I2C.h>

#include <EEPROM.h>

#include <SparkFun\_External\_EEPROM.h>

//#include "SoftwareSerial.h"

#define DSerial Serial

#define ATSerial Serial2

// #define AT\_TX\_PIN 11

// #define AT\_RX\_PIN 10

// SoftwareSerial DSerial(AT\_RX\_PIN, AT\_TX\_PIN);

//#ifndef CONFIG\_ARDUINO\_LOOP\_STACK\_SIZE

// default next line:

//#define CONFIG\_ARDUINO\_LOOP\_STACK\_SIZE 8192

// changed to:

//#define CONFIG\_ARDUINO\_LOOP\_STACK\_SIZE 8192\*4

//#endif

//#define USE\_CERT

#define USE\_CERT

/\* Fill your certificate.pem.root CA wiht LINE ENDING \*/

const char pem\_pkey[] PROGMEM = {"-----BEGIN PRIVATE KEY-----\r\n"

"MIIEwAIBADANBgkqhkiG9w0BAQEFAASCBKowggSmAgEAAoIBAQDq1FT42/MdGCfX\r\n"

"xZtECXQ2wkXPfMA9SGEbLtU+M8cFLkAlTcmt2gZiOchzItBjrkVo157wSG/8po6W\r\n"

"IfZksEy1HOIk1ApJ0zmODnIqUm541KUKxoUV5jYR+FAutDJxq4/rIgSINl7UsZxN\r\n"

"XnkmVkUX+C3WzoWT5ernX0tpuO0fuNbVwu0XoLfoS56tM0HqhQDomPMtJV9+utCH\r\n"

"TatSoPMrZ0mKXrzrr7kVEHka7oOBfriu4YvehFdSjw4cVug68s7FoeBNH+Y05eRj\r\n"

"X5QcbF3omumAMYW2pkT1WTiAcayF9U3KR9WfMhEzH1/GotEydEihIjvyhaIKcQRu\r\n"

"yhTRwQfTAgMBAAECggEBANN1xtqd/ctaQC6eJBXdSYgx4vlXhijpL4Bx6NZte68f\r\n"

"meW9qXnjFdKH5AhB9lN8z8X8PIR1RlrFhSqOhLjgxQJmcwHk6yrViUNxoL6Zoh8u\r\n"

"pTwRzjANMO13pISaAb64Vg3PgTNDepufIHfPNdse3nMaKfT/3etzo9cbyeX2/5iK\r\n"

"eVCHHnvGiBjvHZwa5Dds66hrHmr8ziD9Mm0lZA7DTcAQUewUP1yhovLdqr0G5T9I\r\n"

"+tZWaa0jmuX6kggKIbWd85WNaqavo2YvDjTdY16O8t1eC/r29rQzcNdn119SnD1I\r\n"

"blUpyuKwa3xws6+spxg8Sd6U1GRdfzLXO9Msrvvkx6ECgYEA+RS/SINaEyei+wOw\r\n"

"ijGLztXOW7mdd4Ay7J91mdsZBY69v9bO9vi0VaFb+iExuMNSeiYTFaDH800lJASJ\r\n"

"VhxDzifjeZ+pA1iKfF5sK11DVoakls+i4hdWUo8qdsMxNcXh8RDXQsYkejPHFc1j\r\n"

"U4SV+av9fLOcI2K8ZqwMr2BS37ECgYEA8Vo9QjEz+mx2wo9Wh9bzsMJ9UB+BUz2V\r\n"

"t7W3XTOIey+OOyLmuESbnluzbjuOxkhhjDCTx2r4WCnGsWYJNYQQz+L3t3Pr0VUI\r\n"

"7+vParC48MWy+dUd+QsK/MapriYQUdOkEjtR1c9D9/9Uv6rX/FvYmW0yGEaK+5+l\r\n"

"8VHL7FKX5MMCgYEA0iDmnk3xl2dA33JlZLAXug41YLxeU4oba/O6SmnD2iRpf4XA\r\n"

"ZCm8OOE/ruyB/dUJp3Kndv+Er4TNaG1LTxHZQfn0tY40D1aKvhEKQCsVu9Eq2O+i\r\n"

"AgqeEFgnArn4sdyWl8AVWYe/DjjQI1x6BuM7gr3Nw33ioxeQyS0sJmekgoECgYEA\r\n"

"mIAOaP6x+zABslnjsR1vCWc6yx+9FZ26NkRJwkhHmp6n3lHlOtWPBvamX6aeRkeB\r\n"

"65TeK4k5tmtfMsRoWTpDbTfakj5a6QA0D3UOsDlOAkTJG/c/YxdJMANPP8ypTyrN\r\n"

"Zv+4a8L1DYR2Rk1q//gA2qEWoCQiuW9c6ShoE8D0XQUCgYEA24G0kWv7S1WvSg0w\r\n"

"rFTwqj8ibMLYVc7bDPpk33+xkN903t55EKAQqsiMoWHjKMM+tJUXUBAzPmxwMDeW\r\n"

"VAM/XGLaztIe8SlbSaJCuAAlmG6cY2U/BoS2+bZpNjhz4kc97ffq1h/i2FCuZr46\r\n"

"xrtrOFDK1RCzlk7A0LIDL4dTTMw=\r\n"

"-----END PRIVATE KEY-----\r\n"

};

/\* Fill your certificate.pem.crt wiht LINE ENDING \*/

const char pem\_cert[] PROGMEM = {

"-----BEGIN CERTIFICATE-----\r\n"

"MIIGnTCCBYWgAwIBAgIIaU9QKMWHGRAwDQYJKoZIhvcNAQELBQAwgbQxCzAJBgNV\r\n"

"BAYTAlVTMRAwDgYDVQQIEwdBcml6b25hMRMwEQYDVQQHEwpTY290dHNkYWxlMRow\r\n"

"GAYDVQQKExFHb0RhZGR5LmNvbSwgSW5jLjEtMCsGA1UECxMkaHR0cDovL2NlcnRz\r\n"

"LmdvZGFkZHkuY29tL3JlcG9zaXRvcnkvMTMwMQYDVQQDEypHbyBEYWRkeSBTZWN1\r\n"

"cmUgQ2VydGlmaWNhdGUgQXV0aG9yaXR5IC0gRzIwHhcNMjIxMTA0MDcxNjIwWhcN\r\n"

"MjMxMTA0MDY1OTAwWjAeMRwwGgYDVQQDDBMqLnRyYWNjY2l2aWwuY29tLmF1MIIB\r\n"

"IjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA3jI32mfzH7yoitVm5R+KlpJX\r\n"

"BS6jjVJtjT70Za+ZZIh/6xc8puva+6lizdNJKt51oLt5Hax8xIXhAzs33g/M6PjV\r\n"

"IiCa0FANovcw/RNHt4uopJGLLJDaBlPLTViPuUUKwlI620+kZDtjsrlCTXbX7iPJ\r\n"

"l9j4Ho8DCpEfSp3G9H2cjHgWqfYGtvIVuzvHhnIbNwgsiYYqe6r79yvEKNHW2nyk\r\n"

"FV1G17FpTk9rIIuXPoACnEqXKV8WGdnXsUOq7gTcsIrZ3+8HmYeBTDZX4cGKfPX8\r\n"

"sM4oBAmwo+xmwkBGnwWIoSCSpahOA9tU9rQlvm61/QViArzNKyYHDWdhYVgU4QID\r\n"

"AQABo4IDRjCCA0IwDAYDVR0TAQH/BAIwADAdBgNVHSUEFjAUBggrBgEFBQcDAQYI\r\n"

"KwYBBQUHAwIwDgYDVR0PAQH/BAQDAgWgMDgGA1UdHwQxMC8wLaAroCmGJ2h0dHA6\r\n"

"Ly9jcmwuZ29kYWRkeS5jb20vZ2RpZzJzMS00NzI5LmNybDBdBgNVHSAEVjBUMEgG\r\n"

"C2CGSAGG/W0BBxcBMDkwNwYIKwYBBQUHAgEWK2h0dHA6Ly9jZXJ0aWZpY2F0ZXMu\r\n"

"Z29kYWRkeS5jb20vcmVwb3NpdG9yeS8wCAYGZ4EMAQIBMHYGCCsGAQUFBwEBBGow\r\n"

"aDAkBggrBgEFBQcwAYYYaHR0cDovL29jc3AuZ29kYWRkeS5jb20vMEAGCCsGAQUF\r\n"

"BzAChjRodHRwOi8vY2VydGlmaWNhdGVzLmdvZGFkZHkuY29tL3JlcG9zaXRvcnkv\r\n"

"Z2RpZzIuY3J0MB8GA1UdIwQYMBaAFEDCvSeOzDSDMKIz1/tss/C0LIDOMDEGA1Ud\r\n"

"EQQqMCiCEyoudHJhY2NjaXZpbC5jb20uYXWCEXRyYWNjY2l2aWwuY29tLmF1MB0G\r\n"

"A1UdDgQWBBQ8fuVBcQ2vemUWcDfrKq4bQhgwUTCCAX0GCisGAQQB1nkCBAIEggFt\r\n"

"BIIBaQFnAHYA6D7Q2j71BjUy51covIlryQPTy9ERa+zraeF3fW0GvW4AAAGEQX9v\r\n"

"BgAABAMARzBFAiAPAm21xgpIWj1zu0w8AYy8QhcP7H/TbfJWuVtHXH1ctgIhAJxP\r\n"

"R6Z+4s+U95SUReNtlvu1f8c07j9T3hrp+WifcHRxAHYAejKMVNi3LbYg6jjgUh7p\r\n"

"hBZwMhOFTTvSK8E6V6NS61IAAAGEQX9vrAAABAMARzBFAiEAzzz7cXQ1qEgONBRE\r\n"

"u8acs7NJAb/Et0m44pAVWaHL/+UCIGd5ScT54u3UL/yc/eK0tRD6rYHaeMmvtHbM\r\n"

"W8qLvihBAHUAs3N3B+GEUPhjhtYFqdwRCUp5LbFnDAuH3PADDnk2pZoAAAGEQX9w\r\n"

"qAAABAMARjBEAiBNkt31Xmv2AnZVegVSnBGu4C4Lfkah7UsMI3/Rl77AXwIgQEuE\r\n"

"ENCNW7uuVaYm8S/4BNfZV40UMICgIJChnXT/Ev4wDQYJKoZIhvcNAQELBQADggEB\r\n"

"ALCrcOH25pXZxlAWArXrreoVKti8gJjKUWKZ5nRdTbm+ODmoyu4CecEttGxrSCrH\r\n"

"vRlYeZfUHIQ5KR9kk2QIbsH86H1GlKclD9rvK/8EfnNL29wJO8kSITsZN55vauTd\r\n"

"4LakwT3bD/i4hc7CHraoHTAWeYEimbW6JrNNDjTYPzge5QnLLhA+xnE14J9rNIu\r\n"

"+732Vttud+lwLQpUUWJBMi8l+T16uhOMncQSQiJ8gEyYG5/nJL9vJpVkR47lUz5a\r\n"

"AA8RiwSsem+dQjMdssEDnvV2mqsFu3jKh/GUvE7bmNL+Kizs8SKfrYxvSdu6VMa0\r\n"

"sQiMwL0OiY0esCweAfvdvxc=\r\n"

"-----END CERTIFICATE-----\r\n"

};

/\* Fill your private.pem.key wiht LINE ENDING \*/

const char pem\_CA[] PROGMEM = {"-----BEGIN CERTIFICATE-----\r\n"

"MIIDdzCCAl+gAwIBAgIEAgAAuTANBgkqhkiG9w0BAQUFADBaMQswCQYDVQQGEwJJ\r\n"

"RTESMBAGA1UEChMJQmFsdGltb3JlMRMwEQYDVQQLEwpDeWJlclRydXN0MSIwIAYD\r\n"

"VQQDExlCYWx0aW1vcmUgQ3liZXJUcnVzdCBSb290MB4XDTAwMDUxMjE4NDYwMFoX\r\n"

"DTI1MDUxMjIzNTkwMFowWjELMAkGA1UEBhMCSUUxEjAQBgNVBAoTCUJhbHRpbW9y\r\n"

"ZTETMBEGA1UECxMKQ3liZXJUcnVzdDEiMCAGA1UEAxMZQmFsdGltb3JlIEN5YmVy\r\n"

"VHJ1c3QgUm9vdDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAKMEuyKr\r\n"

"mD1X6CZymrV51Cni4eiVgLGw41uOKymaZN+hXe2wCQVt2yguzmKiYv60iNoS6zjr\r\n"

"IZ3AQSsBUnuId9Mcj8e6uYi1agnnc+gRQKfRzMpijS3ljwumUNKoUMMo6vWrJYeK\r\n"

"mpYcqWe4PwzV9/lSEy/CG9VwcPCPwBLKBsua4dnKM3p31vjsufFoREJIE9LAwqSu\r\n"

"XmD+tqYF/LTdB1kC1FkYmGP1pWPgkAx9XbIGevOF6uvUA65ehD5f/xXtabz5OTZy\r\n"

"dc93Uk3zyZAsuT3lySNTPx8kmCFcB5kpvcY67Oduhjprl3RjM71oGDHweI12v/ye\r\n"

"jl0qhqdNkNwnGjkCAwEAAaNFMEMwHQYDVR0OBBYEFOWdWTCCR1jMrPoIVDaGezq1\r\n"

"BE3wMBIGA1UdEwEB/wQIMAYBAf8CAQMwDgYDVR0PAQH/BAQDAgEGMA0GCSqGSIb3\r\n"

"DQEBBQUAA4IBAQCFDF2O5G9RaEIFoN27TyclhAO992T9Ldcw46QQF+vaKSm2eT92\r\n"

"9hkTI7gQCvlYpNRhcL0EYWoSihfVCr3FvDB81ukMJY2GQE/szKN+OMY3EU/t3Wgx\r\n"

"jkzSswF07r51XgdIGn9w/xZchMB5hbgF/X++ZRGjD8ACtPhSNzkE1akxehi/oCr0\r\n"

"Epn3o0WC4zxe9Z2etciefC7IpJ5OCBRLbf1wbWsaY71k5h+3zvDyny67G7fyUIhz\r\n"

"ksLi4xaNmjICq44Y3ekQEe5+NauQrz4wlHrQMz2nZQ/1/I6eYs9HRCwBXbsdtTLS\r\n"

"R9I4LtD+gdwyah617jzV/OeBHRnDJELqYzmp\r\n"

"-----END CERTIFICATE-----\r\n"

};

#define MQTT\_RESP\_200 "res/200"

#define MQTT\_RESP\_202 "res/202"

#define MQTT\_RESP\_204 "res/204"

#define MQTT\_CONNECTION\_LOST "connectionlost"

#define ignitionPin 25

#define boostEN 13

bool isSignalQualityGood = false;

bool isSimNetworkWorking = false;

bool isMqttHubConnected = false;

bool isMqttProvConnected = false;

bool isDeviceProvisioned = false;

bool isDeviceUpdated = false;

bool isInputStarted = false;

bool isInputDone = false;

bool isGpsFixAcquired = false;

bool resetFlag = false;

char hub\_server[50];

char hub\_client\_id[50];

char hub\_username[130];

char provPollTopic[170];

String Lat = "";

String Long = "";

float prevLat;

float prevLong;

float disTravelled;

float battPercent;

unsigned int min\_rssi = 2;

unsigned long signalTimeoutStart = 0;

unsigned long timeoutPeriod = 0;

unsigned long updateTimeoutStart = 0;

unsigned long updateTimeoutPeriod = 0;

unsigned long inputStartTime = 0;

unsigned long inputTimeout = 60000;

unsigned int comm\_pdp\_index = 1; // The range is 1 ~ 16

unsigned int comm\_ssl\_index = 2; // The range is 0 ~ 5

unsigned int comm\_mqtt\_index = 5; // The range is 0 ~ 5

Mqtt\_Version\_t version = MQTT\_V4;

Mqtt\_Qos\_t mqtt\_qos = AT\_MOST\_ONCE;

String operationalTime = "";

GNSS\_Work\_Mode\_t mode = STAND\_ALONE;

JsonObject respJSON;

DynamicJsonDocument respJSONDoc(512);

struct employee\_input\_info {

int currentUpdateVer;

int max\_employee\_number;

int max\_job\_code;

int max\_cost\_code;

int max\_run\_hours;

int employee\_number\_digits;

int job\_code\_digits;

int cost\_code\_digits;

int run\_hours\_digits;

};

struct deviceData {

float Lat;

float Long;

int employee\_number;

int job\_code;

int cost\_code; // 3

int run\_hours;

// float disTravelled;

float battPercent;

char speedKmh[10] = "";

char accelState[15] = "";

char time[25];

};

unsigned int savedDataCount = 0;

unsigned int savedDataPostCount = 0;

int currentUpdateVer = 1;

int max\_employee\_number;

int max\_job\_code;

int max\_cost\_code;

int max\_run\_hours;

int employee\_number\_digits;

int job\_code\_digits;

int cost\_code\_digits;

int run\_hours\_digits;

String employee\_number = "";

String job\_code = "";

String cost\_code = ""; // 3

String run\_hours = "";

char input\_key;

const byte ROWS = 4; //four rows

const byte COLS = 4; //three columns

char keys[ROWS][COLS] = {

{'1', '2', '3', 'A'},

{'4', '5', '6', 'B'},

{'7', '8', '9', 'C'},

{'\*', '0', '#', 'D'}

};

byte rowPins[ROWS] = {13, 12, 14, 27}; //connect to the row pinouts of the keypad

byte colPins[COLS] = {26, 25, 33, 32}; //connect to the column pinouts of the keypad

WisLTEBG96MQTT WisLTE(ATSerial, DSerial);

LiquidCrystal\_I2C lcd(0x27, 20, 4);

ExternalEEPROM extEEPROM;

Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

Adafruit\_ADXL345\_Unified accel = Adafruit\_ADXL345\_Unified();

sensors\_event\_t event;

float px, py, pz;

char accelState[15];

char speedKmh[10];

//WisLTEBG96SSL ssl(ATSerial, DSerial);

//WisLTEBG96Serial serial(ATSerial, DSerial);

void setup() {

ATSerial.begin(115200);

DSerial.begin(115200);

while (ATSerial.read() >= 0);

pinMode(ignitionPin, INPUT);

pinMode(boostEN, OUTPUT);

digitalWrite(boostEN, HIGH);

delay(1000);

lcd.init();

lcd.backlight();

lcd.clear();

lcd.setCursor(1, 1);

lcd.print(F("Intializing"));

lcd.setCursor(1, 2);

lcd.print(F("Device..."));

Wire.begin();

if (extEEPROM.begin() == false)

{

Serial.println("No memory detected. Freezing.");

}

Serial.println("Memory detected!");

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

// Serial.println(sizeof(deviceData));

Serial.print("Mem size in bytes: ");

Serial.println(extEEPROM.length());

// while(!WisLTE.InitModule()){

//

// }

lcd.clear();

lcd.setCursor(1, 1);

lcd.print(F("Device"));

lcd.setCursor(1, 2);

lcd.print(F("intialized..."));

delay(1000);

TurnGpsOn();

EEPROM\_read\_data();

// if(!accel.begin()){

// Serial.println(F("No valid sensor found"));

// lcd.clear();

// lcd.setCursor(1,1);

// lcd.print(F("acceleratormeter"));

// lcd.setCursor(1,2);

// lcd.print(F("not connected!"));

// delay(1000);

// }

// else{

// Serial.println(F("G sensor found...\n"));

// }

// accel.setRange(ADXL345\_RANGE\_8\_G);

// accel.getEvent(&event);

// px = event.acceleration.x/9.8;

// py = event.acceleration.y/9.8;

// pz = event.acceleration.z/9.8;

keypad.addEventListener(startRide);

}

void loop() {

if ( millis() - signalTimeoutStart >= timeoutPeriod ) {

CheckSignalQuality();

if ( !isSignalQualityGood ) {

// lcd.clear();

// lcd.setCursor(1,0);

// lcd.print(F("Sginal Quality"));

// lcd.setCursor(1,1);

// lcd.print(F("Bad!"));

// lcd.setCursor(1,2);

// lcd.print(F("skipping posting"));

lcd.clear();

lcd.setCursor(1, 0);

lcd.print(F("please"));

lcd.setCursor(1, 1);

lcd.print(F("wait..."));

delay(1000);

}

}

if ( !isSimNetworkWorking & isSignalQualityGood ) {

lcd.clear();

lcd.setCursor(1, 0);

lcd.print(F("please"));

lcd.setCursor(1, 1);

lcd.print(F("wait..."));

configureBG96(); // configure APN and PDP index

if ( isSimNetworkWorking ) {

if ( !configureMQTTS() ) { // Configure SSL and MQTT

lcd.clear();

lcd.setCursor(1, 0);

lcd.print(F("please"));

lcd.setCursor(1, 1);

lcd.print(F("wait..."));

delay(1000);

}

}

else {

lcd.clear();

lcd.setCursor(1, 0);

lcd.print(F("please"));

lcd.setCursor(1, 1);

lcd.print(F("wait..."));

}

}

if ( !isDeviceProvisioned & !isMqttProvConnected & isSimNetworkWorking & isSignalQualityGood ) {

lcd.clear();

lcd.setCursor(1, 0);

lcd.print(F("please"));

lcd.setCursor(1, 1);

lcd.print(F("wait..."));

connectToProvService(); // Connect to Azure provisioning service

if (!isMqttProvConnected) {

lcd.clear();

lcd.setCursor(1, 0);

lcd.print(F("please"));

lcd.setCursor(1, 1);

lcd.print(F("wait..."));

delay(1000);

}

}

if ( !isDeviceProvisioned & isMqttProvConnected & isSimNetworkWorking & isSignalQualityGood ) {

getProvisioned(); // register device with provision service and save Hub info

if (!isDeviceProvisioned) {

lcd.clear();

lcd.setCursor(1, 0);

lcd.print(F("please"));

lcd.setCursor(1, 1);

lcd.print(F("wait..."));

delay(1000);

}

else {

lcd.clear();

lcd.setCursor(1, 0);

lcd.print(F("please"));

lcd.setCursor(1, 1);

lcd.print(F("wait..."));

delay(1000);

}

}

if ( !isMqttHubConnected & isDeviceProvisioned & isSimNetworkWorking & isSignalQualityGood ) {

lcd.clear();

lcd.setCursor(1, 0);

lcd.print(F("please"));

lcd.setCursor(1, 1);

lcd.print(F("wait..."));

connectToHub(); // Connect to IoT Hub

if (!isMqttHubConnected) {

lcd.clear();

lcd.setCursor(1, 0);

lcd.print(F("please"));

lcd.setCursor(1, 1);

lcd.print(F("wait..."));

delay(1000);

}

else {

lcd.clear();

lcd.setCursor(1, 0);

lcd.print(F("please"));

lcd.setCursor(1, 1);

lcd.print(F("wait..."));

delay(1000);

}

}

if ( millis() - updateTimeoutStart >= updateTimeoutPeriod ) {

if ( !isDeviceUpdated & isMqttHubConnected & isSimNetworkWorking & isSignalQualityGood) {

lcd.clear();

lcd.setCursor(1, 0);

lcd.print(F("please"));

lcd.setCursor(1, 1);

lcd.print(F("wait..."));

checkDeviceUpdate(); // Check for updates and apply them

if (!isDeviceUpdated) {

lcd.clear();

lcd.setCursor(1, 0);

lcd.print(F("please"));

lcd.setCursor(1, 1);

lcd.print(F("wait..."));

delay(1000);

}

else {

lcd.clear();

lcd.setCursor(1, 0);

lcd.print(F("please"));

lcd.setCursor(1, 1);

lcd.print(F("wait..."));

delay(1000);

}

}

}

inputStartTime = millis();

while ( (millis() - inputStartTime <= inputTimeout) & !isInputDone ) {

if ( !isInputStarted ) {

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Press any key to");

lcd.setCursor(1, 2);

lcd.print("start data input..");

delay(150);

}

if ( !isInputDone ) {

keypad.getKey();

}

// Serial.println(millis() - inputStartTime);

}

if ( !isInputDone ) {

employee\_number = "";

job\_code = "";

cost\_code = ""; // 3

run\_hours = "";

isInputStarted = false;

lcd.clear();

lcd.setCursor(1, 0);

lcd.print(F("please"));

lcd.setCursor(1, 1);

lcd.print(F("wait..."));

}

else {

inputStartTime = millis();

while (millis() - inputStartTime <= 10000) {

if ( !isInputStarted ) {

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Press any key to");

lcd.setCursor(1, 2);

lcd.print("end Ride...");

delay(150);

}

if ( isInputDone ) {

keypad.getKey();

}

}

if (isInputDone) {

isInputStarted = false;

}

}

if ( isMqttHubConnected & isSignalQualityGood ) {

getAndSendData();

}

else {

saveDataLocally();

}

}

void getBattPercent()

{

int analogValue = analogRead(A0);

float volt = analogValue \* (5.0 / 1023.0);

if (volt <= 3.5) {

battPercent = 0;

return;

}

battPercent = map(volt, 3.5, 4.2, 0, 100);

Serial.print("analogValue : ");

Serial.println(analogValue);

Serial.print("Battery volt : ");

Serial.println(volt);

Serial.print("Battery percentage : ");

Serial.println(battPercent);

}

void getTime()

{

char time1[25];

// Cmd\_Status\_t status = 0;

WisLTE.DevClock(time1, READ\_MODE);

char \*ptr = strstr((const char \*)time1, "+");

\*ptr = '\0';

ptr = strstr((const char \*)time1, ",");

operationalTime = String(ptr + 1);

}

void setDefaultTime()

{

char time1[25];

strcpy(time1, "80/01/06,00:00:00+20");

// Cmd\_Status\_t status = 1;

WisLTE.DevClock(time1, WRITE\_MODE);

}

void displayInfo()

{

lcd.clear();

lcd.setCursor(1, 0);

lcd.print("vehicle speed");

lcd.setCursor(6, 1);

lcd.print(String(speedKmh) + "kmh");

lcd.setCursor(1, 2);

lcd.print("Operational Time");

lcd.setCursor(5, 3);

lcd.print(operationalTime);

delay(1000);

}

void saveDataLocally()

{

lcd.clear();

lcd.setCursor(1, 0);

lcd.print(F("please"));

lcd.setCursor(1, 1);

lcd.print(F("wait..."));

getGpsLoc();

// getAccelInfo();

getDistance();

getTime();

getBattPercent();

displayInfo();

delay(1000);

deviceData data1;

data1.Lat = Lat.toFloat();

data1.Long = Long.toFloat();

data1.employee\_number = employee\_number.toInt();

data1.job\_code = job\_code.toInt();

data1.cost\_code = cost\_code.toInt();

data1.run\_hours = run\_hours.toInt();

// data1.disTravelled = disTravelled;

data1.battPercent = battPercent;

strcpy(data1.speedKmh, speedKmh);

strcpy(data1.accelState, accelState);

strcpy(data1.time, operationalTime.c\_str());

if (savedDataCount \* sizeof(data1) <= 64000) {

Serial.print("\n\nsaving data...");

Serial.println(operationalTime.c\_str());

extEEPROM.put(sizeof(data1)\*savedDataCount, data1);

savedDataCount++;

}

}

void getDistance()

{

float distance = 0;

if (prevLat == 0 & prevLong == 0) {

prevLat = Lat.toFloat();

prevLong = Long.toFloat();

// Serial.print("Lat = ");

// Serial.println(prevLat);

// Serial.print("Long = ");

// Serial.println(prevLong);

}

else {

if (Lat != "" & Long != "") {

distance = getDistanceBetween( prevLat, prevLong, Lat.toFloat(), Long.toFloat());

disTravelled += (distance / 1000);

// Serial.print("dis = ");

// Serial.println(distance);

prevLat = Lat.toFloat();

prevLong = Long.toFloat();

}

}

}

float getDistanceBetween(float lat1, float long1, float lat2, float long2)

{

float delta = radians(long1 - long2);

float sdlong = sin(delta);

float cdlong = cos(delta);

lat1 = radians(lat1);

lat2 = radians(lat2);

float slat1 = sin(lat1);

float clat1 = cos(lat1);

float slat2 = sin(lat2);

float clat2 = cos(lat2);

delta = (clat1 \* slat2) - (slat1 \* clat2 \* cdlong);

delta = sq(delta);

delta += sq(clat2 \* sdlong);

delta = sqrt(delta);

float denom = (slat1 \* slat2) + (clat1 \* clat2 \* cdlong);

delta = atan2(delta, denom);

return delta \* 6372795;

}

void getAndSendData()

{

char dataPubTopic[70];

char serializedData[256];

if (savedDataCount == 0) {

lcd.clear();

lcd.setCursor(1, 0);

lcd.print(F("please"));

lcd.setCursor(1, 1);

lcd.print(F("wait..."));

getGpsLoc();

// getAccelInfo();

// getDistance();

getBattPercent();

getTime();

displayInfo();

clearJSON();

respJSONDoc["GPS Lat"] = Lat;

respJSONDoc["GPS Long"] = Long;

// respJSONDoc["Distance traveled"] = disTravelled;

respJSONDoc["Vehicle speed"] = speedKmh;

respJSONDoc["Vehicle state"] = accelState;

respJSONDoc["Employee Num"] = employee\_number;

respJSONDoc["Job Code"] = job\_code;

respJSONDoc["Cost Code"] = cost\_code;

respJSONDoc["Run Hours"] = run\_hours;

respJSONDoc["Batt%"] = battPercent;

respJSONDoc["Operational Time"] = operationalTime;

sprintf(dataPubTopic, "%s%s%s", "devices/", hub\_client\_id, "/messages/events/");

serializeJson(respJSONDoc, serializedData);

if ( WisLTE.MQTTPublishMessagesEx(comm\_mqtt\_index, 1, AT\_LEAST\_ONCE, dataPubTopic, false, serializedData, strlen(serializedData) ) != 0) {

isMqttHubConnected = false;

Serial.println("/nisMqttHubConnected1 = false/n");

return;

}

}

else if (savedDataCount > savedDataPostCount) {

unsigned long startTime = millis();

while (millis() - startTime <= 60000) {

deviceData data2;

extEEPROM.get(savedDataPostCount \* sizeof(data2), data2);

clearJSON();

respJSONDoc["GPS Lat"] = String(data2.Lat, 4);

respJSONDoc["GPS Long"] = String(data2.Long, 4);

// respJSONDoc["Distance traveled"] = String(data2.disTravelled,4);

respJSONDoc["Vehicle speed"] = data2.speedKmh;

respJSONDoc["Vehicle state"] = data2.accelState;

respJSONDoc["Employee Num"] = String(data2.employee\_number);

respJSONDoc["Job Code"] = String(data2.job\_code);

respJSONDoc["Cost Code"] = String(data2.cost\_code);

respJSONDoc["Run Hours"] = String(data2.run\_hours);

respJSONDoc["Batt%"] = String(data2.battPercent);

respJSONDoc["Operational Time"] = data2.time;

sprintf(dataPubTopic, "%s%s%s", "devices/", hub\_client\_id, "/messages/events/");

serializeJson(respJSONDoc, serializedData);

if ( WisLTE.MQTTPublishMessagesEx(comm\_mqtt\_index, 1, AT\_LEAST\_ONCE, dataPubTopic, false, serializedData, strlen(serializedData) ) != 0) {

isMqttHubConnected = false;

Serial.println("/nisMqttHubConnected2 = false/n");

return;

}

savedDataPostCount++;

if (savedDataCount == savedDataPostCount) {

savedDataCount = 0;

savedDataPostCount = 0;

extEEPROM.erase();

break;

}

}

}

else {

getGpsLoc();

// getDistance();

getTime();

getBattPercent();

displayInfo();

clearJSON();

respJSONDoc["GPS Lat"] = Lat;

respJSONDoc["GPS Long"] = Long;

// respJSONDoc["Distance Traveled"] = disTravelled;

respJSONDoc["Vehicle speed"] = speedKmh;

respJSONDoc["Vehicle state"] = accelState;

respJSONDoc["Employee Num"] = employee\_number;

respJSONDoc["Job Code"] = job\_code;

respJSONDoc["Cost Code"] = cost\_code;

respJSONDoc["Run Hours"] = run\_hours;

respJSONDoc["Operational Time"] = operationalTime;

sprintf(dataPubTopic, "%s%s%s", "devices/", hub\_client\_id, "/messages/events/");

serializeJson(respJSONDoc, serializedData);

if ( WisLTE.MQTTPublishMessagesEx(comm\_mqtt\_index, 1, AT\_LEAST\_ONCE, dataPubTopic, false, serializedData, strlen(serializedData) ) != 0) {

Serial.println("/nisMqttHubConnected3 = false/n");

isMqttHubConnected = false;

return;

}

}

}

void getAccelInfo()

{

float sumX = 0, sumY = 0;

float avgX = 0, avgY = 0;

float i;

for (i = 0; i < 10; i++) {

accel.getEvent(&event);

sumX += abs(event.acceleration.x / 9.8 - px);

sumY += abs(event.acceleration.y / 9.8 - py);

}

avgX = sumX / i;

avgY = sumY / i;

// Serial.print("avgX = ");

// Serial.println(avgX);

// Serial.print("avgY = ");

// Serial.println(avgY);

if ( (avgX > 0.3 | avgY > 0.3) & digitalRead(ignitionPin)) {

sprintf(accelState, "Moving");

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

}

else if ((avgX > 0.3 | avgY > 0.3) & !digitalRead(ignitionPin)) {

sprintf(accelState, "Towed");

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

}

else {

sprintf(accelState, "Idle");

}

px = event.acceleration.x / 9.8;

py = event.acceleration.y / 9.8;

pz = event.acceleration.z / 9.8;

}

void TurnGpsOn()

{

// if(!WisLTE.TurnOnGNSS(mode, WRITE\_MODE)){

// lcd.clear();

// lcd.setCursor(1,1);

// lcd.print("GPS inactive...");

// lcd.setCursor(1,2);

// lcd.print("Please check network");

// delay(1000);

// }

// else{

// Serial.println("\r\nOpen the GNSS Function Fali!");

// if(WisLTE.TurnOnGNSS(mode, READ\_MODE)){

// Serial.println("\r\nThe GNSS Function is Opened!");

// lcd.clear();

// lcd.setCursor(1,2);

// lcd.print("GPS activated...");

// delay(1000);

// //WisLTE.TurnOffGNSS();

//

// }

// }

while (!WisLTE.TurnOnGNSS(mode, WRITE\_MODE)) {

Serial.println("\r\nOpen the GNSS Function Fali!");

lcd.clear();

lcd.setCursor(1, 0);

lcd.print(F("please"));

lcd.setCursor(1, 1);

lcd.print(F("wait..."));

delay(100);

if (WisLTE.TurnOnGNSS(mode, READ\_MODE)) {

Serial.println("\r\nThe GNSS Function is Opened!");

WisLTE.TurnOffGNSS();

}

}

lcd.clear();

lcd.setCursor(1, 0);

lcd.print(F("please"));

lcd.setCursor(1, 1);

lcd.print(F("wait..."));

// delay(1000);

// WisLTE.setGNSSPriority(0);

}

void getGpsLoc()

{

Lat = "";

Long = "";

char GpsPosition[128];

char \*token;

int i = 0;

unsigned long startTime = 0;

// WisLTE.setGNSSPriority(1);

// delay(500);

startTime = millis();

while (millis() - startTime <= 180000) {

if (!WisLTE.GetGNSSPositionInformation(GpsPosition)) {

Lat = "";

Long = "";

strcpy(accelState, "");

strcpy(speedKmh, "");

}

else {

Serial.print("\n\nNEMA senstence = ");

Serial.println(GpsPosition);

token = strtok(GpsPosition, ",");

while (i < 8)

{

token = strtok(NULL, ",");

if ( i == 0 )

Lat = String(token);

else if ( i == 1 ) {

Long = String(token);

}

else if ( i == 7 ) {

// Serial.print("Speed = ");

// Serial.println(token);

// Serial.print("\n\n");

sprintf(speedKmh, String(token).c\_str());

}

if (strcmp(speedKmh, "")) {

sprintf(accelState, "");

}

else if (digitalRead(ignitionPin) == 0 & atof(speedKmh) > 1) {

sprintf(accelState, "Towed");

}

else if (digitalRead(ignitionPin) == 1 & atof(speedKmh) > 1) {

sprintf(accelState, "Moving");

}

else {

sprintf(accelState, "idle");

}

i++;

}

// delay(500);

// WisLTE.setGNSSPriority(0);

return;

}

delay(1500);

}

// delay(500);

// WisLTE.setGNSSPriority(0);

}

void CheckSignalQuality()

{

char err\_code[60];

if ( WisLTE.RegisterAndCheckSignalQuality(err\_code, min\_rssi) ) {

isSignalQualityGood = true;

// Serial.println(err\_code);

timeoutPeriod = 0;

return;

}

else {

isSignalQualityGood = false;

isSimNetworkWorking = false;

Serial.println(err\_code);

signalTimeoutStart = millis();

timeoutPeriod = 60000;

return;

}

}

void saveHubInfo()

{

const char \*server = respJSON["registrationState"]["assignedHub"];

const char \*client\_id = respJSON["registrationState"]["deviceId"];

strcpy(hub\_server, server);

strcpy(hub\_client\_id, client\_id);

sprintf(hub\_username, "%s%s%s%s", hub\_server, "/", hub\_client\_id, "/?api-version=2018-06-30");

// Serial.println(hub\_server);

// Serial.println(hub\_client\_id);

// Serial.println(hub\_username);

}

bool connectToHub()

{

unsigned int mqtt\_port = 8883;

WisLTE.CloseMQTTClient(comm\_mqtt\_index);

WisLTE.CloseMQTTNetwork(comm\_mqtt\_index);

int ret = WisLTE.OpenMQTTNetwork(comm\_mqtt\_index, hub\_server, mqtt\_port);

if (ret == 2) {

isMqttHubConnected = false;

isSimNetworkWorking = false;

resetFlag = true;

return false;

}

else if ( ret == 0) {

Serial.println(F("\r\nSet the MQTT Service Address Success!"));

}

else {

Serial.println(F("\r\nSet the MQTT Service Address Fail!"));

isMqttHubConnected = false;

isSimNetworkWorking = false;

return false;

}

if ( WisLTE.CreateMQTTClient(comm\_mqtt\_index, hub\_client\_id, hub\_username, "") != 0) {

Serial.println(F("\r\nCreate a MQTT Client Fail!"));

// if(!WisLTE.CloseMQTTNetwork(comm\_mqtt\_index)){

// isSimNetworkWorking = false;

// return false;

// }

isMqttHubConnected = false;

return false;

}

else {

Serial.println(F("\r\nCreate a MQTT Client Success!"));

}

isMqttHubConnected = true;

return true;

}

bool connectToProvService()

{

char prov\_server[] = "global.azure-devices-provisioning.net";

unsigned int mqtt\_port = 8883;

char prov\_client\_id[] = "itechdpstest";

char prov\_username[] = "0ne001D3E80/registrations/itechdpstest/api-version=2019-03-31";

WisLTE.CloseMQTTClient(comm\_mqtt\_index);

WisLTE.CloseMQTTNetwork(comm\_mqtt\_index);

if ( WisLTE.OpenMQTTNetwork(comm\_mqtt\_index, prov\_server, mqtt\_port) != 0) {

Serial.println(F("\r\nSet the MQTT Service Address Fail!"));

isSimNetworkWorking = false;

isMqttProvConnected = false;

return false;

}

else {

Serial.println(F("\r\nSet the MQTT Service Address Success!"));

}

if ( WisLTE.CreateMQTTClient(comm\_mqtt\_index, prov\_client\_id, prov\_username, "") != 0) {

Serial.println(F("\r\nCreate a MQTT Client Fail!"));

// if(!WisLTE.CloseMQTTNetwork(comm\_mqtt\_index)){

// isSimNetworkWorking = false;

// return false;

// }

isMqttProvConnected = false;

return false;

}

else {

Serial.println(F("\r\nCreate a MQTT Client Success!"));

}

isMqttProvConnected = true;

return true;

}

bool configureMQTTS()

{

char ssl\_error[100];

if (!init\_ssl(comm\_ssl\_index, ssl\_error)) {

Serial.println(ssl\_error);

}

else {

Serial.println(ssl\_error);

Serial.println(F("SSL Configured..."));

}

if (!WisLTE.SetMQTTConfigureParameters(comm\_mqtt\_index, comm\_pdp\_index, version, 150, SERVER\_DISCARD\_INFORMATION)) {

Serial.println(F("\r\nConfig the MQTT Parameter Fail!"));

// int e\_code;

// if (WisLTE.returnErrorCode(e\_code)){

// Serial.print("\r\nERROR CODE: ");

// Serial.println(e\_code);

// Serial.println("Please check the documentation for error details.");

// }

return false;

}

else {

Serial.println(F("\r\nConfig the MQTT Parameter Success!"));

}

if (!WisLTE.SetMQTTEnableSSL(comm\_mqtt\_index, comm\_ssl\_index, true)) {

Serial.println(F("\r\nEnable the SSL Fail!"));

return false;

}

else {

Serial.println(F("\r\nEnable the SSL Success!"));

}

return true;

}

bool configureBG96()

{

char APN[] = "";

char inf[50];

char apn\_error[50];

if (resetFlag) {

if (WisLTE.SetDevFunctionality(MINIMUM\_FUNCTIONALITY) != 1) {

resetFlag = true;

return false;

}

delay(1000);

if (WisLTE.SetDevFunctionality(FULL\_FUNCTIONALITY) != 1) {

resetFlag = true;

return false;

}

resetFlag = false;

}

WisLTE.SetDevCommandEcho(false);

WisLTE.DeactivateDevAPN(comm\_pdp\_index);

if (WisLTE.GetDevInformation(inf)) {

Serial.println(inf);

}

else {

isSimNetworkWorking = false;

return false;

}

cleanBuffer(inf);

if (WisLTE.GetDevVersion(inf)) {

Serial.println(inf);

}

else {

isSimNetworkWorking = false;

return false;

}

if (!WisLTE.InitAPN(comm\_pdp\_index, APN, "", "", apn\_error)) {

Serial.println(apn\_error);

isSimNetworkWorking = false;

return false;

}

Serial.println(apn\_error);

TurnGpsOn();

// WisLTE.GetDevNetSignalQuality(rssi);

// Serial.print("rssi: ");

// Serial.println(rssi);

// if(rssi)

isSimNetworkWorking = true;

return true;

}

bool checkDeviceUpdate()

{

updateTimeoutStart = millis();

updateTimeoutPeriod = 3600000;

String resp = "";

char getPropSubTopic[] = "$iothub/twin/res/#";

char getPropPubTopic[] = "$iothub/twin/GET/?$rid=1";

char sendPropPubTopic[] = "$iothub/twin/PATCH/properties/reported/?$rid=1";

if (WisLTE.MQTTSubscribeTopic(comm\_mqtt\_index, 1, getPropSubTopic, mqtt\_qos) != 0) {

isMqttHubConnected = false;

return false;

}

if ( WisLTE.MQTTPublishMessagesEx(comm\_mqtt\_index, 1, AT\_LEAST\_ONCE, getPropPubTopic, false, "{}", strlen("{}") ) != 0) {

isMqttHubConnected = false;

return false;

}

resp = getResponse(5);

if (!checkResp(resp, MQTT\_RESP\_200, isDeviceUpdated)) {

return false;

}

if (respJSON["desired"]["$version"] == currentUpdateVer) {

Serial.println();

Serial.println(F("Device already updated..."));

isDeviceUpdated = true;

return true;

}

else {

Serial.println(F("Updating device..."));

currentUpdateVer = respJSON["desired"]["$version"].as<int>();

max\_employee\_number = respJSON["desired"]["settings"]["maxEN"].as<int>();

max\_job\_code = respJSON["desired"]["settings"]["maxJC"].as<int>();

max\_cost\_code = respJSON["desired"]["settings"]["maxCC"].as<int>();

max\_run\_hours = respJSON["desired"]["settings"]["maxRH"].as<int>();

employee\_number\_digits = respJSON["desired"]["settings"]["ENdigits"].as<int>();

job\_code\_digits = respJSON["desired"]["settings"]["JCdigits"].as<int>();

cost\_code\_digits = respJSON["desired"]["settings"]["CCdigits"].as<int>();

run\_hours\_digits = respJSON["desired"]["settings"]["RHdigits"].as<int>();

if (!EEPROM\_wirte\_data()) {

return false;

}

Serial.println(F("Upadate Done..."));

}

clearJSON();

respJSONDoc["updatestatus"] = "Updated";

char reportedProp[50];

serializeJson(respJSONDoc, reportedProp);

// Serial.println(reportedProp);

if ( WisLTE.MQTTPublishMessagesEx(comm\_mqtt\_index, 1, AT\_LEAST\_ONCE, sendPropPubTopic, false, reportedProp, strlen(reportedProp) ) != 0) {

isMqttHubConnected = false;

return false;

}

resp = getResponse(5);

if (!checkResp(resp, MQTT\_RESP\_204, isDeviceUpdated)) {

return false;

}

if ( WisLTE.MQTTUnsubscribeTopic(comm\_mqtt\_index, 1, getPropSubTopic) != 0) {

isMqttHubConnected = false;

return false;

}

Serial.println(F("Twin Property GET Topic unsubscribed"));

isDeviceUpdated = true;

return true;

}

bool getProvisioned()

{

String resp = "";

char provRespTopic[] = "$dps/registrations/res/#";

char provPubTopic[] = "$dps/registrations/PUT/iotdps-register/?$rid=1";

char provPollTopic[170];

if (WisLTE.MQTTSubscribeTopic(comm\_mqtt\_index, 1, provRespTopic, mqtt\_qos) != 0) {

isMqttProvConnected = false;

return false;

}

if ( WisLTE.MQTTPublishMessagesEx(comm\_mqtt\_index, 1, AT\_LEAST\_ONCE, provPubTopic, false, "{}", strlen("{}") ) != 0)

{

isMqttProvConnected = false;

return false;

}

resp = getResponse(5);

if (!checkResp(resp, MQTT\_RESP\_202, isDeviceProvisioned)) {

return false;

}

if (respJSON["operationId"] != "null" & respJSON["operationId"] != "" ) {

cleanBuffer(provPollTopic);

sprintf(provPollTopic, "%s%s%s", "$dps/registrations/GET/iotdps-get-operationstatus/?$rid=1", "&operationId=", respJSON["operationId"].as<const char \*>());

}

else {

isDeviceProvisioned = false;

return false;

}

resp = MQTT\_RESP\_202;

while (resp == MQTT\_RESP\_202)

{

if ( WisLTE.MQTTPublishMessagesEx(comm\_mqtt\_index, 1, AT\_LEAST\_ONCE, provPollTopic, false, "{}", strlen("{}") ) != 0) {

isMqttProvConnected = false;

return false;

}

resp = getResponse(5);

// if(respJSON["status"] == "failed"){

//

// }else

if (resp == MQTT\_RESP\_200) {

break;

}

delay(2000);

}

if (!checkResp(resp, MQTT\_RESP\_200, isDeviceProvisioned)) {

return false;

}

if ( WisLTE.MQTTUnsubscribeTopic(comm\_mqtt\_index, 1, provRespTopic) != 0) {

isMqttProvConnected = false;

return false;

}

Serial.println(F("Porvision Topic unsubscribed"));

saveHubInfo();

isDeviceProvisioned = true;

if (!WisLTE.CloseMQTTClient(comm\_mqtt\_index)) {

isMqttProvConnected = false;

return false;

}

isMqttProvConnected = false;

return true;

}

String getResponse(int timeout)

{

char mqtt\_recv[512];

String resp = "";

char \*resp\_pointer;

char \*obj\_pointer;

clearJSON();

Mqtt\_URC\_Event\_t ret = WisLTE.WaitCheckMQTTURCEvent(mqtt\_recv, timeout);

switch (ret)

{

case MQTT\_RECV\_DATA\_EVENT:

{

// Serial.println(mqtt\_recv);

resp\_pointer = strstr((const char \*)mqtt\_recv, "res/");

for (int i = 0; i < 7; i++) {

resp += \*resp\_pointer;

resp\_pointer++;

}

Serial.println();

Serial.println(resp);

int chr = '"';

obj\_pointer = strrchr((const char \*)mqtt\_recv, chr);

\*obj\_pointer = '\0';

obj\_pointer = strstr((const char \*)mqtt\_recv, ",");

// Serial.println(obj\_pointer+2);

deserializeJson(respJSONDoc, (const char \*)(obj\_pointer + 2));

respJSON = respJSONDoc.as<JsonObject>();

// Serial.println();

// Serial.print(F("memory usage: "));

// Serial.println(respJSONDoc.memoryUsage());

return String(resp);

break;

}

case MQTT\_STATUS\_EVENT:

{

resp = MQTT\_CONNECTION\_LOST;

return resp;

break;

}

case MQTT\_TIMEOUT:

{

Serial.println(F("\nMQTT unkown error"));

resp = MQTT\_CONNECTION\_LOST;

return resp;

break;

}

default:

{

Serial.println(F("\nMQTT unkown error"));

resp = MQTT\_CONNECTION\_LOST;

return resp;

break;

}

};

return MQTT\_CONNECTION\_LOST;

}

bool checkResp(String resp, String resp\_test, bool &flag)

{

if (resp == MQTT\_CONNECTION\_LOST) {

isMqttProvConnected = false;

isMqttHubConnected = false;

return false;

}

else if (resp == resp\_test) {

return true;

}

else {

flag = false;

return false;

}

}

void clearJSON()

{

respJSONDoc.clear();

respJSON.clear();

}

void cleanBuffer(char \*buf)

{

memset(buf, '\0', strlen(buf));

}

bool DeleteFiles(char \*filename)

{

char cmd[32], buf[32];

strcpy(cmd, FILE\_DELETE\_FILES);

sprintf(buf, "=\"%s\"", filename);

strcat(cmd, buf);

if (WisLTE.sendAndSearch(cmd, RESPONSE\_OK, RESPONSE\_ERROR, 2)) {

return true;

}

return false;

}

bool UploadFiles(String filename, String u\_file)

{

char cmd[32], buf[32],buf1[32],buf2[100];

strcpy(cmd, FILE\_UPLOAD\_FILES);

filename.toCharArray(buf1, filename.length());

u\_file.toCharArray(buf2, u\_file.length());

// sprintf(buf, "=\"%s\",%d", filename, strlen(u\_file));

sprintf(buf, "=\"%s\",%d", buf1, u\_file.length());

strcat(cmd, buf);

if (WisLTE.sendAndSearch(cmd, RESPONSE\_CONNECT, RESPONSE\_ERROR, 5)) {

//if (WisLTE.sendDataAndCheck(u\_file, RESPONSE\_OK, RESPONSE\_ERROR, 10))

if (WisLTE.sendDataAndCheck(buf2, RESPONSE\_OK, RESPONSE\_ERROR, 10))

{

return true;

}

}

return false;

}

bool init\_ssl(unsigned int ssl\_index, char \*err\_code) {

unsigned long start\_time = millis();

char \*e\_str;

int f\_err\_code;

char ch;

String cert = "";

if (!WisLTE.SetSSLParameters(ssl\_index, TLS\_1\_2, SUPPORT\_ALL\_ABOVE, 300)) {

e\_str = "\r\nSSL ERROR: An error occurred while setting the ssl parameter.\r\n";

strcpy(err\_code, e\_str);

return false;

}

for (unsigned int k = 0; k < strlen\_P(pem\_CA); k++) {

ch = pgm\_read\_byte\_near(pem\_CA + k);

cert.concat(ch);

}

// passing CA\_cert

while (!UploadFiles((char \*)ssl\_ca\_cert\_name, cert.c\_str())){

if(WisLTE.returnErrorCode(f\_err\_code)){

if (f\_err\_code == 407){

start\_time = millis();

while (!DeleteFiles((char \*)ssl\_ca\_cert\_name)){

if(millis() - start\_time >= 10\*1000UL){

e\_str = "\r\nSSL ERROR: The ssl ca cert file exists. An error occurred while deleting the original file during the re-upload process.\r\n";

strcpy(err\_code, e\_str);

return false;

}

}

}

}else if(millis() - start\_time >= 30\*1000UL){

sprintf(e\_str, "\r\nSSL ERROR: Error uploading file, error code: %d ,Please check the corresponding documentation for details.\r\n", f\_err\_code);

strcpy(err\_code, e\_str);

return false;

}

}

start\_time = millis();

cert="";

for (unsigned int k = 0; k < strlen\_P(pem\_cert); k++) {

ch = pgm\_read\_byte\_near(pem\_cert + k);

cert.concat(ch);

}

while (!UploadFiles((const char \*)ssl\_client\_cert\_name, cert.c\_str())){

if(WisLTE.returnErrorCode(f\_err\_code)){

if (f\_err\_code == 407){

start\_time = millis();

while (!DeleteFiles((char \*)ssl\_client\_cert\_name)){

if(millis() - start\_time >= 10\*1000UL){

e\_str = "\r\nSSL ERROR: The ssl ca cert file exists. An error occurred while deleting the original file during the re-upload process.\r\n";

strcpy(err\_code, e\_str);

return false;

}

}

}

}else if(millis() - start\_time >= 30\*1000UL){

sprintf(e\_str, "\r\nSSL ERROR: Error uploading file, error code: %d ,Please check the corresponding documentation for details.\r\n", f\_err\_code);

strcpy(err\_code, e\_str);

return false;

}

}

start\_time = millis();

cert="";

for (unsigned int k = 0; k < strlen\_P(pem\_pkey); k++) {

ch = pgm\_read\_byte\_near(pem\_pkey + k);

cert.concat(ch);

}

while (!UploadFiles((const char \*)ssl\_client\_key\_name, cert.c\_str())){

if(WisLTE.returnErrorCode(f\_err\_code)){

if (f\_err\_code == 407){

start\_time = millis();

while (!DeleteFiles((char \*)ssl\_client\_key\_name)){

if(millis() - start\_time >= 10\*1000UL){

e\_str = "\r\nSSL ERROR: The ssl ca cert file exists. An error occurred while deleting the original file during the re-upload process.\r\n";

strcpy(err\_code, e\_str);

return false;

}

}

}

}else if(millis() - start\_time >= 30\*1000UL){

sprintf(e\_str, "\r\nSSL ERROR: Error uploading file, error code: %d ,Please check the corresponding documentation for details.\r\n", f\_err\_code);

strcpy(err\_code, e\_str);

return false;

}

}

start\_time = millis();

while (!WisLTE.SetSSLCertificate(ssl\_index, (char \*) ssl\_ca\_cert\_name, (char \*) ssl\_client\_cert\_name, (char \*) ssl\_client\_key\_name , true)) { //ssl\_ca\_cert\_name, ssl\_client\_cert\_name, ssl\_client\_key\_name

if (millis() - start\_time >= 30 \* 1000UL) {

e\_str = "\r\nSSL ERROR: An error occurred while setting the ssl certificate.\r\n";

strcpy(err\_code, e\_str);

return false;

}

}

e\_str = "\r\nSSL OK: The ssl were successfully initialized.\r\n";

strcpy(err\_code, e\_str);

return true;

}

bool EEPROM\_wirte\_data()

{

if ( max\_employee\_number <= 0 | currentUpdateVer <= 1 | max\_job\_code <= 0 | max\_cost\_code <= 0 | max\_run\_hours <= 0) {

currentUpdateVer = 1;

max\_employee\_number = 500;

max\_job\_code = 500;

max\_cost\_code = 500;

max\_run\_hours = 1000;

return false;

}

else if (employee\_number\_digits < 4 | job\_code\_digits < 4 | cost\_code\_digits < 3 | run\_hours\_digits < 4) {

employee\_number\_digits = 4;

job\_code\_digits = 4;

cost\_code\_digits = 3;

run\_hours\_digits = 4;

return false;

}

employee\_input\_info data\_limits = {

currentUpdateVer,

max\_employee\_number,

max\_job\_code,

max\_cost\_code,

max\_run\_hours,

employee\_number\_digits,

job\_code\_digits,

cost\_code\_digits,

run\_hours\_digits

};

EEPROM.put(0, data\_limits);

return true;

}

void EEPROM\_read\_data()

{

employee\_input\_info data\_limits;

EEPROM.get(0, data\_limits);

currentUpdateVer = data\_limits.currentUpdateVer;

max\_employee\_number = data\_limits.max\_employee\_number;

max\_job\_code = data\_limits.max\_job\_code;

max\_cost\_code = data\_limits.max\_cost\_code;

max\_run\_hours = data\_limits.max\_run\_hours;

employee\_number\_digits = data\_limits.employee\_number\_digits;

job\_code\_digits = data\_limits.job\_code\_digits;

cost\_code\_digits = data\_limits.cost\_code\_digits;

run\_hours\_digits = data\_limits.run\_hours\_digits;

if ( max\_employee\_number <= 0 | currentUpdateVer <= 1 | max\_job\_code <= 0 | max\_cost\_code <= 0 | max\_run\_hours <= 0) {

currentUpdateVer = 1;

max\_employee\_number = 500;

max\_job\_code = 500;

max\_cost\_code = 500;

max\_run\_hours = 1000;

}

if (employee\_number\_digits < 4 | job\_code\_digits < 4 | cost\_code\_digits < 3 | run\_hours\_digits < 4) {

employee\_number\_digits = 4;

job\_code\_digits = 4;

cost\_code\_digits = 3;

run\_hours\_digits = 4;

}

Serial.print("max\_employee\_number");

Serial.println(max\_employee\_number);

Serial.print("max\_job\_code");

Serial.println(max\_job\_code);

Serial.print("max\_cost\_code");

Serial.println(max\_cost\_code);

Serial.print("max\_run\_hours");

Serial.println(max\_run\_hours);

Serial.print("employee\_number\_digits");

Serial.println(employee\_number\_digits);

Serial.print("job\_code\_digits");

Serial.println(job\_code\_digits);

Serial.print("cost\_code\_digits");

Serial.println(cost\_code\_digits);

Serial.print("run\_hours\_digits");

Serial.println(run\_hours\_digits);

}

void startRide(KeypadEvent input\_key) {

if (keypad.getState() == PRESSED)

{

if ( !isInputDone )

{

if ( !isInputStarted ) {

isInputStarted = true;

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Enter Employee Num");

return;

}

if (employee\_number.length() < employee\_number\_digits)

{

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Enter Employee Num");

lcd.setCursor(2, 2);

Serial.println();

Serial.print("input key is: ");

Serial.println(input\_key);

if (input\_key == 'A' | input\_key == 'B' | input\_key == 'D' | input\_key == '\*' | input\_key == '#')

{

Serial.println("Not Allowed!");

lcd.clear();

lcd.print("Input Not Allowed!");

delay(500);

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Enter Employee Num");

lcd.setCursor(2, 2);

lcd.print(employee\_number);

Serial.print("employee\_number1 is: ");

Serial.println(employee\_number);

}

else if (input\_key == 'C')

{

if (employee\_number.length() > 0)

{

employee\_number.remove(employee\_number.length() - 1);

}

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Enter Employee Num");

lcd.setCursor(2, 2);

lcd.print(employee\_number);

Serial.print("employee\_number2 is: ");

Serial.println(employee\_number);

}

else

{

employee\_number.concat(input\_key);

lcd.print(employee\_number);

Serial.print("employee\_number3 is: ");

Serial.println(employee\_number);

inputStartTime = millis();

}

if (employee\_number.toInt() < 0 | employee\_number.toInt() > max\_employee\_number)

{

lcd.clear();

lcd.print("Invalid Input!");

Serial.print(" Invalid Input!");

employee\_number = "";

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Enter Employee Num");

return;

}

if (employee\_number.length() == employee\_number\_digits) {

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Enter Job Code");

lcd.setCursor(2, 2);

return;

}

return;

}

if (job\_code.length() < job\_code\_digits)

{

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Enter Job Code");

lcd.setCursor(2, 2);

Serial.println(input\_key);

if (input\_key == 'A' | input\_key == 'B' | input\_key == 'D' | input\_key == '\*' | input\_key == '#')

{

Serial.println("Not Allowed!");

lcd.clear();

lcd.print("Input Not Allowed!");

delay(500);

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Enter Job Code");

lcd.setCursor(2, 2);

lcd.print(job\_code);

Serial.println(job\_code);

}

else if (input\_key == 'C')

{

if (job\_code.length() > 0)

{

job\_code.remove(job\_code.length() - 1);

}

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Enter Job Code");

lcd.setCursor(2, 2);

lcd.print(job\_code);

Serial.println(job\_code);

}

else

{

job\_code.concat(input\_key);

lcd.print(job\_code);

}

if (job\_code.toInt() < 0 | job\_code.toInt() > max\_job\_code)

{

lcd.clear();

lcd.print("Invalid Input!");

job\_code = "";

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Enter Job Code");

return;

}

if (job\_code.length() == job\_code\_digits) {

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Enter Cost Code");

lcd.setCursor(2, 2);

return;

}

return;

}

if (cost\_code.length() < cost\_code\_digits)

{

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Enter Cost Code");

lcd.setCursor(2, 2);

Serial.println(input\_key);

if (input\_key == 'A' | input\_key == 'B' | input\_key == 'D' | input\_key == '\*' | input\_key == '#')

{

Serial.println("Not Allowed!");

lcd.clear();

lcd.print("Input Not Allowed!");

delay(500);

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Enter Cost Code");

lcd.setCursor(2, 2);

lcd.print(cost\_code);

Serial.println(cost\_code);

}

else if (input\_key == 'C')

{

if (cost\_code.length() > 0)

{

cost\_code.remove(cost\_code.length() - 1);

}

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Enter Cost Code");

lcd.setCursor(2, 2);

lcd.print(cost\_code);

Serial.println(cost\_code);

}

else

{

cost\_code.concat(input\_key);

lcd.print(cost\_code);

}

if (cost\_code.toInt() < 0 | cost\_code.toInt() > max\_cost\_code)

{

lcd.clear();

lcd.print("Invalid Input!");

cost\_code = "";

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Enter Cost Code");

return;

}

if (cost\_code.length() == cost\_code\_digits) {

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Enter Run Hours");

lcd.setCursor(2, 2);

return;

}

return;

}

if (run\_hours.length() < run\_hours\_digits)

{

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Enter Run Hours");

lcd.setCursor(2, 2);

Serial.println(input\_key);

if (input\_key == 'A' | input\_key == 'B' | input\_key == 'D' | input\_key == '\*' | input\_key == '#')

{

Serial.println("Not Allowed!");

lcd.clear();

lcd.print("Input Not Allowed!");

delay(500);

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Enter Run Hours");

lcd.setCursor(2, 2);

lcd.print(run\_hours);

Serial.println(run\_hours);

}

else if (input\_key == 'C')

{

if (run\_hours.length() > 0)

{

run\_hours.remove(run\_hours.length() - 1);

}

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Enter Run Hours");

lcd.setCursor(2, 2);

lcd.print(run\_hours);

Serial.println(run\_hours);

}

else

{

run\_hours.concat(input\_key);

lcd.print(run\_hours);

}

if (run\_hours.toInt() < 0 | run\_hours.toInt() > max\_run\_hours)

{

lcd.clear();

lcd.print("Invalid Input!");

run\_hours = "";

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Enter Run Hours");

return;

}

if (run\_hours.length() == run\_hours\_digits) {

lcd.clear();

lcd.setCursor(1, 0);

lcd.print("EN: ");

lcd.print(employee\_number);

lcd.setCursor(11, 0);

lcd.print("JC: ");

lcd.print(job\_code);

lcd.setCursor(1, 1);

lcd.print("CC: ");

lcd.print(cost\_code);

lcd.setCursor(11, 1);

lcd.print("RH: ");

lcd.print(run\_hours);

lcd.setCursor(1, 2);

lcd.print("A: Save");

lcd.setCursor(1, 3);

lcd.print("B: Re-Enter");

Serial.println("at the end: ");

Serial.println(employee\_number);

Serial.println(job\_code);

Serial.println(cost\_code);

Serial.println(run\_hours);

return;

}

return;

}

if (input\_key == 'A') {

Serial.println("pressed A");

isInputDone = true;

isInputStarted = false;

setDefaultTime();

return;

}

else if (input\_key == 'B') {

employee\_number = "";

job\_code = "";

cost\_code = "";

run\_hours = "";

isInputStarted = false;

return;

}

else {

return;

}

}

else

{

if ( !isInputStarted ) {

isInputStarted = true;

lcd.clear();

lcd.setCursor(1, 1);

lcd.print("Are you Sure?");

lcd.setCursor(1, 2);

lcd.print("A: Yes B: No");

inputStartTime = millis();

return;

}

if (input\_key == 'B') {

isInputDone = true;

isInputStarted = false;

inputStartTime = 0;

return;

}

else if (input\_key == 'A') {

employee\_number = "";

job\_code = "";

cost\_code = "";

run\_hours = "";

prevLat = 0;

prevLong = 0;

disTravelled = 0;

isInputStarted = false;

isInputDone = false;

inputStartTime = 0;

return;

}

else {

return;

}

}

}

return;

}