A6GPRS Modem Services

Author: David Henry <mailto:mgadriver@gmail.com>

# Introduction

The A6GPRS is a low cost GSM modem with 2G capabilities.

This library aims to make general services such as telephony, SMS and TCP client communication easily available to the Arduino user.

The library is intended to be used on any hardware Serial port and will NOT work via SoftwareSerial.

The library is divided into a number of classes:

1. GPRSA6Device. This class handles low level tasks such as handling UART activity to and from the modem hardware. Broadly speaking the Arduino programmer does not need to be concerned with the methods and variables of this class. However some customization must be performed before use.
2. GPRSA6. This class encapsulated the modem AT command language to perform a variety of basic services such as basic telephony, SMS, GPRS connections and TCP client connections.
3. MQTT. This class implements the MQTT Messaging Protocol Version 3.1.1.
4. HTTP. This class … (TBD)
5. SMTP. This class … (TBD)
6. Call. This class will implement basic telephony, initiating calls, accepting calls,Caller ID etc. (TBD)
7. SMS. This class … (TBD)

# Design Constraints

The documentation available for the A6GPRS is incomplete and sometimes just incorrect. The code developed so far is a mix of reading the book and simple observation.

A particular problem is missing descriptions of unsolicited result codes. When connected as a TCP client to a server, data from the server may be mixed up with various modem result codes and it is up to the various classes to separate the 2 streams.

For example, the MQTT class does not expect to have to deal with incoming calls or SMS messages. You may wish to add that yourselves.

The modem works, by default, at a baud rate of 115200 bps. No other baud rate is supported.

# Customization

## GPRSA6Device

Edit the A6Modem.h file to determine which hardware serial port is used and the maximum size of the circular buffer used to hold incoming data.

1. Edit the macro HW\_SERIAL to define which UART is used i.e Serial, Serial1, Serial2 or Serial3.
2. Edit the macro HW\_SERIAL\_EVENT to define the matching serial event to HW\_SERIAL, e.g. Serial2 is handled by serialEvent2.
3. Edit the macro COMM\_BUF\_LEN to define the size of the circular buffer holding incoming data.

## MTTT

Edit the A6MQTT.h file to determine the maximum length of an incoming publish event (topic + message parts). Publish events that are too long are discarded.

1. Edit the macro MAX\_MESSAGE\_LENGTH to be large enough to accommodate and topic + message.

# Design Philosophy – MQTT

While the class handles the formatting and parsing of all interaction between the client and the broker, it raises events that should be handled by the application.

While the application may choose not to handle events raised by the MQTT class, it makes the application a lot simpler to do so.

The provided examples show how this is done.

# Design Philosophy – GPRSA6Device

Various debugging messages are present in the code. If the application wishes to see the messages it must implement ‘helper’ methods to display the message. For example the line

Gsm.DebugWrite(“got a reply”);

may appear anywhere in the library.

To see the message a method GPRSA6Device:: DebugWrite(char \*) must be implemented. If the message goes to a serial port or LCD screen or whatever is up to the developer.

# API

### GPRSA6:

Add the following line to your code

#include “A6Services.h”

An instance of GPRSA6 called gsm is automatically created.

### Methods

1. bool getIMEI(char\* buffer);
   1. Retrieve the modems IMIE value and write it to buffer. Buffer must be large enough to hold the text.
   2. Returns true if completed successfully, else false.
2. bool getCIMI(char\* buffer);
   1. Retrieve the SIM card CIMI value and write it to buffer. Buffer must be large enough to hold the text.
   2. Returns true if completed successfully, else false.
3. bool getRTC(char\* buffer);
   1. Retrieve the modems Real Time Clock value and write it to buffer. Buffer must be large enough to hold the text.
   2. The value is in the format yy/mo/dd,hh:mm:ss,+tz where
      1. yy is year since 2000
      2. mo month number
      3. dd day number
      4. hh hour
      5. mm minute
      6. ss second
      7. tx – timezone expressed in ½ hours before/after GMT
      8. Example “17/01/08,21:33:19,+02” is 8th January 2017 21 hours 33 minutes 19 secs GMT. Local timezone 1 hour before GMT.
   3. Returns true if completed successfully, else false.
4. bool setRTC(char\* buffer);
   1. Set the modems Real Time Clock value and from buffer.
   2. The format of buffer is as described in getRTC.
   3. Returns true if completed successfully, else false.
5. enum eCIPstatus getCIPstatus();
6. char \*getCIPstatusString(enum eCIPstatus);
7. char \*getCIPstatusString(); // current value
8. bool startIP(char \*apn);
   1. Start a GPRS connection with the Access Point named apn. No userid or password needed.
   2. Returns true if completed successfully, else false.
9. bool startIP(char \*apn,char\*userid,char \*password);
   1. Start a GPRS connection with the Access Point named apn using the credentials userid and password.
   2. Returns true if completed successfully, else false.
10. ePSstate getPSstate();
11. bool setPSstate(ePSstate);
12. bool stopIP();
13. bool getLocalIP(char \*);
14. bool connectTCPserver(char\*,int);
15. bool sendToServer(char\*);
16. bool sendToServer(char\*,int); // for ascii strings
17. bool sendToServer(byte\*,int); // for byte arrays