A6Lib

Generated by Doxygen 1.8.14

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 A6lib Class Reference

A library for controlling Ai-Thinker A6 modem.

```
#include <A6lib.h>
```

Public Member Functions

- A6lib (HardwareSerial ∗port)
- A6lib (SoftwareSerial ∗port)
- A6lib (uint8_t rx_pin, uint8_t tx_pin)
- ∼A6lib ()
- void handle ()
- bool start (unsigned long baud, uint8_t max_retry)
- void powerUp (int pin)
- void hardReset (uint8_t pin)
- void softReset ()
- String getFirmWareVer ()
- int8_t getRSSI ()
- uint8_t getSignalQuality ()
- String getRealTimeClock ()
- String getIMEI ()
- String getSMSSca ()
- RegisterStatus getRegisterStatus ()
- bool setPreferedStorage (SMSStorageArea)
- bool setCharSet (const String &charset)
- bool sendSMS (const String &number, const String &text)
- bool sendPDU (const String &number, const String &content)
- bool sendPDU (const String &number, wchar_t ∗content, uint8_t len)
- SMSInfo readSMS (uint8_t index)
- bool deleteSMS (uint8_t index)
- int8_t getSMSList (int8_t ∗buff, uint8_t len, SMSRecordType record)
- void dial (String number)
- void redial ()
- void answer ()
- void hangUp ()

- callInfo checkCallStatus ()
- void setVol (byte level)
- void enableSpeaker (byte enable)
- void addHandler (void_cb_t)
- void onSMSSent (sms_tx_cb_t)
- void onSMSReceived (sms_rx_cb_t)
- void onSMSStorageFull (sms_full_cb_t)
- String sendCommand (const String &command, uint16_t reply_timeout=2000)

Static Public Member Functions

- static String registerStatusToString (RegisterStatus)

3.1.1 Detailed Description

A library for controlling Ai-Thinker A6 modem.

An Arduino library for communicating with the AI-Thinker A6 GSM module, It currently supports ESP8266 and A↩ VR architectures. This small lib mainly intended for Ai-Thinker A6 modem but may possiblly work with other GSM modems supporting standard AT command set (e.g SIM800,SIM900,...). Using this lib is straightforward, you can create an object of A6lib via HardwareSerial, SoftwareSerial or just two pin number for built in SoftwareSerial. Then you usually should power up your module (A6lib::powerUp()) and initlize A6lib object to start communicating with modem at desired baud rate. from now on, use public APIs to control your modem and get informations from it.

This lib has been modified to be asynchronous, so currently you can pass your functions to register APIs to catch these events:

1. SMS sent
2. SMS recevied
3. Storage area is full

Note

A note about A6lib::addHandler(): When you have some important tasks in your code for example reading keypad etc, you can add a main function for running those tasks and pass it to A6lib::addHandler(), when you pass a valid function, lib will call it whenever it's in waiting state (waiting for modem to reply) and thus it'll prevent locking in that precious time.

To get start you can check out examples directory.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 A6lib() [1/3]

```
A6lib::A6lib (
            HardwareSerial * port )
```

Constructs A6lib object with the given serial *port*.

Parameters

port	HardwareSerial object for use inside A6lib.

3.1.2.2 A6lib() [2/3]

```
A6lib::A6lib (
            SoftwareSerial * port )
```

Constructs A6lib object with the given serial port.

Parameters

port	SoftwareSerial object for use inside A6lib

3.1.2.3 A6lib() [3/3]

```
A6lib::A6lib (
            uint8_t rx_pin,
            uint8_t tx_pin )
```

Constructs A6lib object with the given pin numbers. this is done by creating new SoftwareSerial object.

Parameters

tx_pin	SoftwareSerial TX pin
rx_pin	SoftwareSerial RX pin

3.1.2.4 ∼A6lib()

```
A6lib::∼A6lib ( )
```

Destroys A6lib object.

3.1.3 Member Function Documentation

3.1.3.1 addHandler()

```
void A6lib::addHandler (
             void_cb_t cb )
```

Add the A6lib main handler callback. A6lib will call this handler when it is inside the waiting routine. it'll prevent lock in your code when you have some critical tasks to run. Note: The result of passing loop() to this function is undefined!

3.1.3.2 answer()

```
void A6lib::answer ( )
```

3.1.3.3 checkCallStatus()

```
callInfo A6lib::checkCallStatus ( )
```

3.1.3.4 deleteSMS()

```
bool A6lib::deleteSMS (
             uint8_t index )
```

Delete a SMS from modem prefered storage area.

Parameters

index	sms index in storage area

Returns

true on success

3.1.3.5 dial()

```
void A6lib::dial (
             String number )
```

3.1.3.6 enableSpeaker()

```
void A6lib::enableSpeaker (
             byte enable )
```

3.1.3.7 getFirmWareVer()

```
String A6lib::getFirmWareVer ( )
```

Get the revision identification or firmware version of modem.

Returns

If success a String contain firmware version, and if fail an empty string.

3.1.3.8 getIMEI()

```
String A6lib::getIMEI ( )
```

Get the modem IMEI.

Returns

if success a string contain IMEI number, if fail an empty string.

3.1.3.9 getRealTimeClock()

```
String A6lib::getRealTimeClock ( )
```

Get the real time from modem.

Returns

if success a string contain time in format yy/mm/dd,hh:mm:ss+zz, if fail an empty string.

3.1.3.10 getRegisterStatus()

```
RegisterStatus A6lib::getRegisterStatus ( )
```

Get the network registration status of modem.

Returns

on of the RegisterStatus value

3.1.3.11 getRSSI()

```
int8_t A6lib::getRSSI ( )
```

Get the modem signal strength based on RSSI(measured as dBm).

Returns

If success a value between -113dBm and -51dBm and if fail 0.

3.1.3.12 getSignalQuality()

```
uint8_t A6lib::getSignalQuality ( )
```

Get the modem signal quality level.

Returns

if success a value between 0-100 and if fail 255.

3.1.3.13 getSMSList()

```
int8_t A6lib::getSMSList (
             int8_t * buff,
             uint8_t len,
             SMSRecordType record )
```

Get the list of available SMS in prefered storage area.

Parameters

buff	input buffer to store SMS indexes.
len	size of buff
record	on of the SMSRecordType.

Returns

if fail -1, otherwise number of founded SMS.

3.1.3.14 getSMSSca()

```
String A6lib::getSMSSca ( )
```

Get the current SMS service center address from modem.

Returns

if success a string contain SCA, if fail an empty string

3.1.3.15 handle()

```
void A6lib::handle ( )
```

the main handler of A6lib object. this function needs to be called inside main loop regularly, for callbacks to work correctly.

3.1.3.16 hangUp()

```
void A6lib::hangUp ( )
```

3.1.3.17 hardReset()

```
void A6lib::hardReset (
             uint8_t pin )
```

This function will do a hard reset on module. It's recommended to do this via an NMOS. Note: it will take some time for module to start + register for network. You may also need to reinitilize module with A6lib::start().

Parameters

pin	the pin number which is connected to module reset(RST) pin.

3.1.3.18 onSMSReceived()

```
void A6lib::onSMSReceived (
             sms_rx_cb_t cb )
```

This function will register your callback and will call it when new SMS arrives.

Parameters

cb	pointer to callback function

3.1.3.19 onSMSSent()

```
void A6lib::onSMSSent (
             sms_tx_cb_t cb )
```

This function will register your callback and will call it when a SMS is sent.

Parameters

cb	pointer to callback function

3.1.3.20 onSMSStorageFull()

```
void A6lib::onSMSStorageFull (
             sms_full_cb_t cb )
```

This function will register your callback and will call it when modem prefered storage area is full.

Parameters

cb	pointer to callback function

3.1.3.21 powerUp()

```
void A6lib::powerUp (
             int pin )
```

this optional function will keep the PWR pin of modem in high TTL at start up to correctly powering module. Module needs this pin to be high TTL for about 2 sec.

Parameters

pin	the pin number which is connected to module PWR pin.

3.1.3.22 readSMS()

```
SMSInfo A6lib::readSMS (
             uint8_t index )
```

Read a SMS in modem prefered storage area

Parameters

index	sms index in storage area

Returns

a SMSInfo object contain SMS information(number+date+timestamp) on success, and if fail an empty SMSInfo object.

3.1.3.23 redial()

```
void A6lib::redial ( )
```

3.1.3.24 registerStatusToString()

```
String A6lib::registerStatusToString (
             RegisterStatus st )  [static]
```

3.1.3.25 sendCommand()

```
String A6lib::sendCommand (
             const String & command,
             uint16_t reply_timeout = 2000 )
```

Send new command to modem. command should be a valid AT command, otherwise modem will return error with corresponding error code. Note: you may want to check modem is busy or not with A6lib::isBusy().

Parameters

command	the command to be sent with AT prefix
reply_timeout	the timeout for modem to reply

Returns

if success an string contain modem reply, otherwise contain error code

3.1.3.26 sendPDU() [1/2]

```
bool A6lib::sendPDU (
             const String & number,
             const String & content )
```

Send an ASCII SMS in PDU mode.

Parameters

number	the detination phone number which should begin with international code
content	the SMS content in ASCII and up to 160 chars

Returns

true on success

3.1.3.27 sendPDU() [2/2]

```
bool A6lib::sendPDU (
            const String & number,
            wchar_t * content,
            uint8_t len )
```

Send a UCS2 SMS in PDU mode.

Parameters

number	the detination phone number which should begin with international code
content	the SMS content coded in UCS2 format and up to 70 chars.
len	the number of UCS2 chars in *content*

Returns

true on success

3.1.3.28 sendSMS()

```
bool A6lib::sendSMS (
            const String & number,
            const String & text )
```

Send SMS (in text mode) to specified number.

Parameters

number	valid destination number without +
text	SMS content in ascii encoding

Returns

true on success

3.1.3.29 setCharSet()

```
bool A6lib::setCharSet (
             const String & charset )
```

set the module charset.

Parameters

charset	the required charset. Could be on of the following: GSM UCS2 HEX PCCP936

Returns

true on success.

3.1.3.30 setPreferedStorage()

```
bool A6lib::setPreferedStorage (
             SMSStorageArea area )
```

Set the modem prefered storage area. It's set to SMSStorageArea::ME by defualt.

Parameters

area	could be on of the SMSStorageArea

Returns

true on success

3.1.3.31 setVol()

```
void A6lib::setVol (
             byte level )
```

3.1.3.32 softReset()

```
void A6lib::softReset ( )
```

This function implement a software restart on module(if suppoerted). Note: it will take some time for module to start + register for network. You may also need to reinitilize module with A6lib::start().

3.1.3.33 start()

```
bool A6lib::start (
            unsigned long baud,
            uint8_t max_retry )
```

This is the A6lib object initlizer routine. you must call this usually once in setup routine.

Parameters

baud	the desired baud rate to start with
max_retry	the maximum number of time A6lib object try to etablish connection.

Returns

true on success

The documentation for this class was generated from the following files:

- A6lib.h
- A6lib.cpp

3.2 callInfo Struct Reference

```
#include <A6lib.h>
```

Public Attributes

- int index
- call_direction direction
- call_state state
- call_mode mode
- int multiparty
- String number
- int type

3.2.1 Member Data Documentation

3.2.1.1 direction

```
call_direction callInfo::direction
```

3.2.1.2 index

```
int callInfo::index
```

3.2.1.3 mode

```
call_mode callInfo::mode
```

3.2.1.4 multiparty

```
int callInfo::multiparty
```

3.2.1.5 number

```
String callInfo::number
```

3.2.1.6 state

```
call_state callInfo::state
```

3.2.1.7 type

```
int callInfo::type
```

The documentation for this struct was generated from the following file:

- A6lib.h

3.3 SMSInfo Struct Reference

```
#include <A6lib.h>
```

Public Member Functions

- SMSInfo ()

Public Attributes

- String number
- String date
- String message

3.3.1 Constructor & Destructor Documentation

3.3.1.1 SMSInfo()

```
SMSInfo::SMSInfo ( )  [inline]
```

3.3.2 Member Data Documentation

3.3.2.1 date

```
String SMSInfo::date
```

3.3.2.2 message

```
String SMSInfo::message
```

3.3.2.3 number

```
String SMSInfo::number
```

The documentation for this struct was generated from the following file:

- A6lib.h

Chapter 4

File Documentation

4.1 A6lib.cpp File Reference

```
#include <stdarg.h>
#include "A6lib.h"
#include "pdu.h"
```

4.2 A6lib.h File Reference

```
#include <Arduino.h>
#include <SoftwareSerial.h>
#include <HardwareSerial.h>
```

Classes

- struct SMSInfo
- struct callInfo
- class A6lib

  A library for controlling Ai-Thinker A6 modem.

Typedefs

- typedef void(∗ void_cb_t) (void)
- typedef void(∗ sms_rx_cb_t) (uint8_t indx, const SMSInfo &)
- typedef void(∗ sms_tx_cb_t) (void)
- typedef void_cb_t sms_full_cb_t

Enumerations

- enum call_direction { DIR_OUTGOING = 0, DIR_INCOMING = 1 }
- enum call_state {
  CALL_ACTIVE = 0, CALL_HELD = 1, CALL_DIALING = 2, CALL_ALERTING = 3,
  CALL_INCOMING = 4, CALL_WAITING = 5, CALL_RELEASE = 7 }
- enum call_mode {
  MODE_VOICE = 0, MODE_DATA = 1, MODE_FAX = 2, MODE_VOICE_THEN_DATA_VMODE = 3,
  MODE_VOICE_AND_DATA_VMODE = 4, MODE_VOICE_AND_FAX_VMODE = 5, MODE_VOICE_THEN_DATA_DMODE
  = 6, MODE_VOICE_AND_DATA_DMODE = 7,
  MODE_VOICE_AND_FAX_FMODE = 8, MODE_UNKNOWN = 9 }
- enum RegisterStatus {
  NotRegistered = 0, Registered_HomeNetwork = 1, Searching_To_Register = 2, Register_Denied = 3,
  Unknow = 4, Registered_Roaming = 5 }
- enum SMSStorageArea { ME = 1, SM }
- enum SMSRecordType { All, Unread, Read }

4.2.1 Typedef Documentation

4.2.1.1 sms_full_cb_t

```
typedef void_cb_t sms_full_cb_t
```

4.2.1.2 sms_rx_cb_t

```
typedef void(* sms_rx_cb_t) (uint8_t indx, const SMSInfo &)
```

4.2.1.3 sms_tx_cb_t

```
typedef void(* sms_tx_cb_t) (void)
```

4.2.1.4 void_cb_t

```
typedef void(* void_cb_t) (void)
```

4.2.2 Enumeration Type Documentation

4.2.2.1 call_direction

```
enum call_direction
```

Enumerator

DIR_OUTGOING	
DIR_INCOMING	

4.2.2.2 call_mode

`enum call_mode`

Enumerator

MODE_VOICE	
MODE_DATA	
MODE_FAX	
MODE_VOICE_THEN_DATA_VMODE	
MODE_VOICE_AND_DATA_VMODE	
MODE_VOICE_AND_FAX_VMODE	
MODE_VOICE_THEN_DATA_DMODE	
MODE_VOICE_AND_DATA_DMODE	
MODE_VOICE_AND_FAX_FMODE	
MODE_UNKNOWN	

4.2.2.3 call_state

`enum call_state`

Enumerator

CALL_ACTIVE	
CALL_HELD	
CALL_DIALING	
CALL_ALERTING	
CALL_INCOMING	
CALL_WAITING	
CALL_RELEASE	

4.2.2.4 RegisterStatus

`enum RegisterStatus`

Enumerator

NotRegistered	
Registered_HomeNetwork	
Searching_To_Register	
Register_Denied	
Unknow	
Registered_Roaming	

4.2.2.5 SMSRecordType

```
enum SMSRecordType
```

Enumerator

All	
Unread	
Read	

4.2.2.6 SMSStorageArea

```
enum SMSStorageArea
```

Enumerator

ME	
SM	

4.3 pdu.h File Reference

```
#include <stdint.h>
#include <wchar.h>
```

Functions

- int pdu_encode (const char ∗sca, const char ∗phone, const char ∗text, uint8_t text_len, uint8_t ∗pdu, uint8_t pdu_size)

  Encode input SMS text (which is coded in ASCII) into a SMS-SUBMIT pdu.
- int pdu_encodew (const char ∗sca, const char ∗phone, const wchar_t ∗text, uint8_t text_len, uint8_t ∗pdu, uint8_t pdu_size)

  Encode input SMS text (which is coded in UCS2) into a SMS-SUBMIT pdu.

4.3.1 Function Documentation

4.3.1.1 pdu_encode()

```
int pdu_encode (
             const char * sca,
             const char * phone,
             const char * text,
             uint8_t text_len,
             uint8_t * pdu,
             uint8_t pdu_size )
```

Encode input SMS *text* (which is coded in ASCII) into a SMS-SUBMIT pdu.

Parameters

sca	a null terminated string contain SMS service center address
phone	a null terminated string contain destination phone number
text	the SMS content in ASCII
text_len	the number of chars in SMS content(could be up to 160 char long)
pdu	the input buffer which is going to hold the final pdu
pdu_size	the size of input pdu buffer

Returns

if success a positive value represent number of pdu octets written, if fail a negative value represent error code

4.3.1.2 pdu_encodew()

```
int pdu_encodew (
             const char * sca,
             const char * phone,
             const wchar_t * text,
             uint8_t text_len,
             uint8_t * pdu,
             uint8_t pdu_size )
```

Encode input SMS *text* (which is coded in UCS2) into a SMS-SUBMIT pdu.

Parameters

sca	a null terminated string contain SMS service center address
phone	a null terminated string contain destination phone number
text	the SMS content coded in UCS2 coding scheme
text_len	the number of UCS2 chars in SMS content(could be up to 70 char long)
pdu	the input buffer which is going to hold the final pdu
pdu_size	the size of input pdu buffer

Returns

if success a positive value represent number of pdu octets written, if fail a negative value represent error code

Index