

A6Lib

Generated by Doxygen 1.8.14



# Contents

<b>1</b>	<b>Class Index</b>	<b>1</b>
1.1	Class List . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
3.1	A6lib Class Reference . . . . .	5
3.1.1	Detailed Description . . . . .	6
3.1.2	Constructor & Destructor Documentation . . . . .	6
3.1.2.1	A6lib() [1/3] . . . . .	6
3.1.2.2	A6lib() [2/3] . . . . .	7
3.1.2.3	A6lib() [3/3] . . . . .	7
3.1.2.4	~A6lib() . . . . .	7
3.1.3	Member Function Documentation . . . . .	7
3.1.3.1	addHandler() . . . . .	8
3.1.3.2	answer() . . . . .	8
3.1.3.3	checkCallStatus() . . . . .	8
3.1.3.4	deleteSMS() . . . . .	8
3.1.3.5	dial() . . . . .	8
3.1.3.6	enableSpeaker() . . . . .	9
3.1.3.7	getFirmWareVer() . . . . .	9
3.1.3.8	getIMEI() . . . . .	9
3.1.3.9	getRealTimeClock() . . . . .	9

3.1.3.10	getRegisterStatus()	9
3.1.3.11	getRSSI()	10
3.1.3.12	getSignalQuality()	10
3.1.3.13	getSMSList()	10
3.1.3.14	getSMSSca()	10
3.1.3.15	handle()	11
3.1.3.16	hangUp()	11
3.1.3.17	hardReset()	11
3.1.3.18	onSMSReceived()	11
3.1.3.19	onSMSSent()	12
3.1.3.20	onSMSSStorageFull()	12
3.1.3.21	powerUp()	12
3.1.3.22	readSMS()	12
3.1.3.23	redial()	13
3.1.3.24	registerStatusToString()	13
3.1.3.25	sendCommand()	13
3.1.3.26	sendPDU() [1/2]	13
3.1.3.27	sendPDU() [2/2]	14
3.1.3.28	sendSMS()	14
3.1.3.29	setCharSet()	15
3.1.3.30	setPreferredStorage()	15
3.1.3.31	setVol()	15
3.1.3.32	softReset()	16
3.1.3.33	start()	16
3.2	callInfo Struct Reference	16
3.2.1	Member Data Documentation	16
3.2.1.1	direction	17
3.2.1.2	index	17
3.2.1.3	mode	17
3.2.1.4	multiparty	17
3.2.1.5	number	17
3.2.1.6	state	17
3.2.1.7	type	17
3.3	SMSInfo Struct Reference	18
3.3.1	Constructor & Destructor Documentation	18
3.3.1.1	SMSInfo()	18
3.3.2	Member Data Documentation	18
3.3.2.1	date	18
3.3.2.2	message	18
3.3.2.3	number	18

<b>4 File Documentation</b>	<b>19</b>
4.1 A6lib.cpp File Reference	19
4.2 A6lib.h File Reference	19
4.2.1 Typedef Documentation	20
4.2.1.1 sms_full_cb_t	20
4.2.1.2 sms_rx_cb_t	20
4.2.1.3 sms_tx_cb_t	20
4.2.1.4 void_cb_t	20
4.2.2 Enumeration Type Documentation	20
4.2.2.1 call_direction	20
4.2.2.2 call_mode	21
4.2.2.3 call_state	21
4.2.2.4 RegisterStatus	21
4.2.2.5 SMSRecordType	22
4.2.2.6 SMSStorageArea	22
4.3 pdu.h File Reference	22
4.3.1 Function Documentation	23
4.3.1.1 pdu_encode()	23
4.3.1.2 pdu_encodew()	23
<b>Index</b>	<b>25</b>



# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">A6lib</a>	A library for controlling Ai-Thinker A6 modem . . . . .	<a href="#">5</a>
<a href="#">callInfo</a>	. . . . .	<a href="#">16</a>
<a href="#">SMSInfo</a>	. . . . .	<a href="#">18</a>





## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">A6lib.cpp</a>	19
<a href="#">A6lib.h</a>	19
<a href="#">pdu.h</a>	22



## 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 AVR architectures. This small lib mainly intended for Ai-Thinker A6 modem but may possibly 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 initialize [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 received
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

<i>port</i>	HardwareSerial object for use inside <a href="#">A6lib</a> .
-------------	--

## 3.1.2.2 A6lib() [2/3]

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

Constructs [A6lib](#) object with the given serial port.

## Parameters

<i>port</i>	SoftwareSerial object for use inside <a href="#">A6lib</a>
-------------	--

## 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

<i>tx_pin</i>	SoftwareSerial TX pin
<i>rx_pin</i>	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 preferred storage area.

##### Parameters

<i>index</i>	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 preferred storage area.

##### Parameters

<i>buff</i>	input buffer to store SMS indexes.
<i>len</i>	size of buff
<i>record</i>	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**

<i>pin</i>	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**

<i>cb</i>	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

<i>cb</i>	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 preferred storage area is full.

#### Parameters

<i>cb</i>	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

<i>pin</i>	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 preferred storage area

## Parameters

<i>index</i>	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

<i>command</i>	the command to be sent with AT prefix
<i>reply_timeout</i>	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

<i>number</i>	the detination phone number which should begin with international code
<i>content</i>	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

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

#### 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

<i>number</i>	valid destination number without +
<i>text</i>	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**

<i>charset</i>	the required charset. Could be on of the following: GSM UCS2 HEX PCCP936
----------------	--

**Returns**

true on success.

**3.1.3.30 setPreferredStorage()**

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

Set the modem preferred storage area. It's set to SMSStorageArea::ME by default.

**Parameters**

<i>area</i>	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

<i>baud</i>	the desired baud rate to start with
<i>max_retry</i>	the maximum number of time <a href="#">A6lib</a> object try to establish 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 `SMSSStorageArea` { `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

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

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

- ~A6lib
  - A6lib, [7](#)
- A6lib, [5](#)
  - ~A6lib, [7](#)
  - A6lib, [6](#), [7](#)
  - addHandler, [7](#)
  - answer, [8](#)
  - checkCallStatus, [8](#)
  - deleteSMS, [8](#)
  - dial, [8](#)
  - enableSpeaker, [8](#)
  - getFirmWareVer, [9](#)
  - getIMEI, [9](#)
  - getRSSI, [9](#)
  - getRealTimeClock, [9](#)
  - getRegisterStatus, [9](#)
  - getSMSList, [10](#)
  - getSMSSca, [10](#)
  - getSignalQuality, [10](#)
  - handle, [11](#)
  - hangUp, [11](#)
  - hardReset, [11](#)
  - onSMSReceived, [11](#)
  - onSMSSent, [11](#)
  - onSMSSStorageFull, [12](#)
  - powerUp, [12](#)
  - readSMS, [12](#)
  - redial, [13](#)
  - registerStatusToString, [13](#)
  - sendCommand, [13](#)
  - sendPDU, [13](#), [14](#)
  - sendSMS, [14](#)
  - setCharSet, [15](#)
  - setPreferredStorage, [15](#)
  - setVol, [15](#)
  - softReset, [15](#)
  - start, [16](#)
- A6lib.cpp, [19](#)
- A6lib.h, [19](#)
  - call\_direction, [20](#)
  - call\_mode, [21](#)
  - call\_state, [21](#)
  - RegisterStatus, [21](#)
  - SMSRecordType, [22](#)
  - SMSStorageArea, [22](#)
  - sms\_full\_cb\_t, [20](#)
  - sms\_rx\_cb\_t, [20](#)
  - sms\_tx\_cb\_t, [20](#)
  - void\_cb\_t, [20](#)
- addHandler
  - A6lib, [7](#)
- answer
  - A6lib, [8](#)
- call\_direction
  - A6lib.h, [20](#)
- call\_mode
  - A6lib.h, [21](#)
- call\_state
  - A6lib.h, [21](#)
- callInfo, [16](#)
  - direction, [16](#)
  - index, [17](#)
  - mode, [17](#)
  - multiparty, [17](#)
  - number, [17](#)
  - state, [17](#)
  - type, [17](#)
- checkCallStatus
  - A6lib, [8](#)
- date
  - SMSInfo, [18](#)
- deleteSMS
  - A6lib, [8](#)
- dial
  - A6lib, [8](#)
- direction
  - callInfo, [16](#)
- enableSpeaker
  - A6lib, [8](#)
- getFirmWareVer
  - A6lib, [9](#)
- getIMEI
  - A6lib, [9](#)
- getRSSI
  - A6lib, [9](#)
- getRealTimeClock
  - A6lib, [9](#)
- getRegisterStatus
  - A6lib, [9](#)
- getSMSList
  - A6lib, [10](#)
- getSMSSca
  - A6lib, [10](#)
- getSignalQuality
  - A6lib, [10](#)

- handle
  - A6lib, 11
- hangUp
  - A6lib, 11
- hardReset
  - A6lib, 11
- index
  - callInfo, 17
- message
  - SMSInfo, 18
- mode
  - callInfo, 17
- multiparty
  - callInfo, 17
- number
  - callInfo, 17
  - SMSInfo, 18
- onSMSReceived
  - A6lib, 11
- onSMSSent
  - A6lib, 11
- onSMSStorageFull
  - A6lib, 12
- pdu.h, 22
  - pdu\_encode, 23
  - pdu\_encodew, 23
- pdu\_encode
  - pdu.h, 23
- pdu\_encodew
  - pdu.h, 23
- powerUp
  - A6lib, 12
- readSMS
  - A6lib, 12
- redial
  - A6lib, 13
- RegisterStatus
  - A6lib.h, 21
- registerStatusToString
  - A6lib, 13
- SMSInfo, 18
  - date, 18
  - message, 18
  - number, 18
  - SMSInfo, 18
- SMSRecordType
  - A6lib.h, 22
- SMSStorageArea
  - A6lib.h, 22
- sendCommand
  - A6lib, 13
- sendPDU
  - A6lib, 13, 14
- sendSMS
  - A6lib, 14
- setCharSet
  - A6lib, 15
- setPreferredStorage
  - A6lib, 15
- setVol
  - A6lib, 15
- sms\_full\_cb\_t
  - A6lib.h, 20
- sms\_rx\_cb\_t
  - A6lib.h, 20
- sms\_tx\_cb\_t
  - A6lib.h, 20
- softReset
  - A6lib, 15
- start
  - A6lib, 16
- state
  - callInfo, 17
- type
  - callInfo, 17
- void\_cb\_t
  - A6lib.h, 20