Arduino Gsm Driver

Generated by Doxygen 1.8.9.1

Sun Jan 17 2016 23:12:57

Contents

1 Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

2 Class Index

2.1 Class List

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

3 File Index

3.1 File List

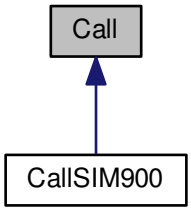
Here is a list of all files with brief descriptions:

4 Class Documentation

4.1 Call Class Reference

`#include <Call.h>`

Inheritance diagram for Call:



Public Member Functions

- virtual unsigned char answer ()=0
- virtual unsigned char callNumber (unsigned char *number)=0
- virtual unsigned char callFromPhonebook (unsigned char position)=0
- virtual unsigned char callByPhonebookMatch (unsigned char *entry)=0
- virtual unsigned char redial ()=0
- virtual unsigned char disconnect ()=0
- virtual unsigned char setAutomaticallyAnswering (unsigned char rings)=0
- virtual unsigned char answer ()=0
- virtual unsigned char callNumber (unsigned char *number)=0
- virtual unsigned char callFromPhonebook (unsigned char position)=0
- virtual unsigned char callByPhonebookMatch (unsigned char *entry)=0
- virtual unsigned char redial ()=0
- virtual unsigned char disconnect ()=0
- virtual unsigned char setAutomaticallyAnswering (unsigned char rings)=0

4.1.1 Detailed Description

Arduino - Gsm driver.

Call.h

Interface to calls.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 14 of file Call.h.

4.1.2 Member Function Documentation

4.1.2.1 virtual unsigned char Call::answer ( ) [pure virtual]

Answer an Incoming Call.

TA sends off-hook to the remote station.

Returns

Implemented in CallSIM900.

4.1.2.2 virtual unsigned char Call::answer ( ) [pure virtual]

Answer an Incoming Call.

TA sends off-hook to the remote station.

Returns

Implemented in CallSIM900.

4.1.2.3 virtual unsigned char Call::callByPhonebookMatch (unsigned char * *entry*) [pure virtual]

Originate Call to Phone Number in Memory Which Corresponds to Field.

This Command make the TA attempts to set up an outgoing call to stored number which has the name matching with the entry string.

Parameters

entry	Phonebook entry.

Returns

Implemented in CallSIM900.

4.1.2.4 virtual unsigned char Call::callByPhonebookMatch (unsigned char * *entry*) [pure virtual]

Originate Call to Phone Number in Memory Which Corresponds to Field.

This Command make the TA attempts to set up an outgoing call to stored number which has the name matching with the entry string.

Parameters

entry	Phonebook entry.

Returns

Implemented in CallSIM900.

4.1.2.5 virtual unsigned char Call::callFromPhonebook (unsigned char *position*) [pure virtual]

Originate Call to Phone Number in Current Memory.

This Command can be used to dial a phone number from current phonebook.

Parameters

position	Phonebook position.

Returns

Implemented in CallSIM900.

4.1.2.6 virtual unsigned char Call::callFromPhonebook (unsigned char *position*) [pure virtual]

Originate Call to Phone Number in Current Memory.

This Command can be used to dial a phone number from current phonebook.

Parameters

position	Phonebook position.

Returns

Implemented in CallSIM900.

4.1.2.7 virtual unsigned char Call::callNumber (unsigned char * *number*) [pure virtual]

Mobile Originated Call to Dial A Number.

This Command can be used to set up outgoing voice, data or fax calls. It also serves to control supplementary services.

Parameters

number	The number to make the call to.

Returns

Implemented in CallSIM900.

4.1.2.8 virtual unsigned char Call::callNumber (unsigned char * *number*) [pure virtual]

Mobile Originated Call to Dial A Number.

This Command can be used to set up outgoing voice, data or fax calls. It also serves to control supplementary services.

Parameters

number	The number to make the call to.

Returns

Implemented in CallSIM900.

4.1.2.9 virtual unsigned char Call::disconnect ( ) [pure virtual]

Disconnect Existing Connection.

Returns

Implemented in CallSIM900.

4.1.2.10 virtual unsigned char Call::disconnect ( ) [pure virtual]

Disconnect Existing Connection.

Returns

Implemented in CallSIM900.

4.1.2.11 virtual unsigned char Call::redial ( ) [pure virtual]

Redial Last Telephone Number Used.

This Command redials the last voice and data call number used.

Returns

Implemented in CallSIM900.

4.1.2.12 virtual unsigned char Call::redial ( ) [pure virtual]

Redial Last Telephone Number Used.

This Command redials the last voice and data call number used.

Returns

Implemented in CallSIM900.

4.1.2.13 virtual unsigned char Call::setAutomaticallyAnswering (unsigned char *rings*) [pure virtual]

Set number of rings before automatically answering the call.

Parameters

rings	Number of rings before automatically answering. 0 means disable.

Returns

0 if error, $>$ 0 otherwise.

Implemented in CallSIM900.

4.1.2.14 virtual unsigned char Call::setAutomaticallyAnswering (unsigned char *rings*) [pure virtual]

Set number of rings before automatically answering the call.

Parameters

rings	Number of rings before automatically answering. 0 means disable.

Returns

0 if error, $>$ 0 otherwise.

Implemented in CallSIM900.

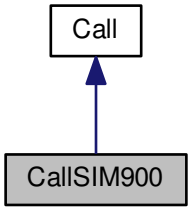
The documentation for this class was generated from the following files:

- Call.h
- PhonebookSIM900.h

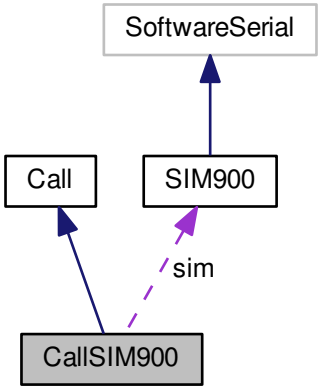
4.2 CallSIM900 Class Reference

`#include <CallSIM900.h>`

Inheritance diagram for CallSIM900:



Collaboration diagram for CallSIM900:



Public Types

- enum CallResponse {
  OK = 0, CME_ERROR = 1, NO_DIALTONE = 2, BUSY = 3,
  NO_CARRIER = 4, NO_ANSWER = 5, CONNECT_TEXT = 6 }

Public Member Functions

- CallSIM900 (SIM900 ∗sim)
- virtual ∼CallSIM900 ()
- unsigned char answer ()
- unsigned char callNumber (unsigned char ∗number)
- unsigned char callFromPhonebook (unsigned char position)

- unsigned char callByPhonebookMatch (unsigned char *entry)
- unsigned char redial ()
- unsigned char disconnect ()
- unsigned char setAutomaticallyAnswering (unsigned char rings)
- unsigned char checkResponse ()

Private Attributes

- SIM900 * sim

4.2.1 Detailed Description

Arduino - Gsm driver.

CallSIM900.h

Interface to calls.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 17 of file CallSIM900.h.

4.2.2 Member Enumeration Documentation

4.2.2.1 enum CallSIM900::CallResponse

Enumerator

OK
CME_ERROR
NO_DIALTONE
BUSY
NO_CARRIER
NO_ANSWER
CONNECT_TEXT

Definition at line 23 of file CallSIM900.h.

4.2.3 Constructor & Destructor Documentation

4.2.3.1 CallSIM900::CallSIM900 (SIM900 * *sim*)

Definition at line 17 of file CallSIM900.cpp.

4.2.3.2 CallSIM900::~CallSIM900 ( ) [virtual]

Definition at line 21 of file CallSIM900.cpp.

4.2.4 Member Function Documentation

4.2.4.1 unsigned char CallSIM900::answer ( ) [virtual]

Answer an Incoming Call.

TA sends off-hook to the remote station.

Returns

Implements Call.

Definition at line 24 of file CallSIM900.cpp.

4.2.4.2 unsigned char CallSIM900::callByPhonebookMatch (unsigned char * *entry*) [virtual]

Originate Call to Phone Number in Memory Which Corresponds to Field.

This Command make the TA attempts to set up an outgoing call to stored number which has the name matching with the entry string.

Parameters

entry	Phonebook entry.

Returns

Implements Call.

Definition at line 38 of file CallSIM900.cpp.

4.2.4.3 unsigned char CallSIM900::callFromPhonebook (unsigned char *position*) [virtual]

Originate Call to Phone Number in Current Memory.

This Command can be used to dial a phone number from current phonebook.

Parameters

position	Phonebook position.

Returns

Implements Call.

Definition at line 31 of file CallSIM900.cpp.

4.2.4.4 unsigned char CallSIM900::callNumber (unsigned char * *number*) [virtual]

Mobile Originated Call to Dial A Number.

This Command can be used to set up outgoing voice, data or fax calls. It also serves to control supplementary services.

Parameters

number	The number to make the call to.

Returns

Implements Call.

Definition at line 28 of file CallSIM900.cpp.

4.2.4.5 unsigned char CallSIM900::checkResponse ( )

Check call response.

Definition at line 49 of file CallSIM900.cpp.

4.2.4.6 unsigned char CallSIM900::disconnect ( ) [virtual]

Disconnect Existing Connection.

Returns

Implements Call.

Definition at line 45 of file CallSIM900.cpp.

4.2.4.7 unsigned char CallSIM900::redial ( ) [virtual]

Redial Last Telephone Number Used.

This Command redials the last voice and data call number used.

Returns

Implements Call.

Definition at line 41 of file CallSIM900.cpp.

4.2.4.8 unsigned char CallSIM900::setAutomaticallyAnswering (unsigned char *rings*) [virtual]

Set number of rings before automatically answering the call.

Parameters

rings	Number of rings before automatically answering. 0 means disable.

Returns

0 if error, $>$ 0 otherwise.

Implements Call.

4.2.5 Member Data Documentation

4.2.5.1 SIM900* CallSIM900::sim [private]

Definition at line 19 of file CallSIM900.h.

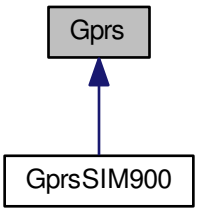
The documentation for this class was generated from the following files:

- CallSIM900.h
- CallSIM900.cpp

4.3 Gprs Class Reference

`#include <Gprs.h>`

Inheritance diagram for Gprs:



Public Member Functions

- virtual unsigned char useMultiplexer (bool use)=0
- virtual unsigned char attach (const char *apn, const char *login, const char *password)=0
- virtual unsigned char bringUp ()=0
- virtual unsigned char obtainIp (unsigned char ip[4])=0
- virtual unsigned char status ()=0
- virtual unsigned char status (char connection)=0
- virtual unsigned char configureDns (const char *primary, const char *secondary)=0
- virtual unsigned char open (const char *mode, const char *address, unsigned int port)=0
- virtual unsigned char open (char connection, const char *mode, const char *address, unsigned int port)=0
- virtual unsigned char close (char connection)=0
- virtual unsigned char close ()=0
- virtual unsigned char resolve (const char *name, unsigned char *buf)=0
- virtual unsigned int send (unsigned char *buf, unsigned int len)=0
- virtual unsigned int send (char connection, unsigned char *buf, unsigned int len)=0
- virtual unsigned char configureServer (unsigned char mode, unsigned int port)=0
- virtual unsigned char shutdown ()=0
- virtual unsigned char transmittingState (void *stateStruct)=0
- virtual unsigned char transmittingState (char connection, void *stateStruct)=0

4.3.1 Detailed Description

Arduino - Gsm driver.

Gprs.h

GPRS connection using.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 14 of file Gprs.h.

4.3.2 Member Function Documentation

4.3.2.1 virtual unsigned char Gprs::attach (const char * *apn,* const char * *login,* const char * *password*) [pure virtual]

Start Task and Set APN, LOGIN, PASSWORD.

Each parameter must be \0 teminated.

Parameters

apn	The apn access point name.
login	The GPRS user name.
password	The GPRS password.

Returns

Implemented in GprsSIM900.

4.3.2.2 virtual unsigned char Gprs::bringUp ( ) [pure virtual]

Bring Up Wireless Connection with GPRS or CSD.

Connects to the GPRS network.

Returns

Implemented in GprsSIM900.

4.3.2.3 virtual unsigned char Gprs::close (char *connection*) [pure virtual]

Close TCP or UDP Connection.

Parameters

connection	

Returns

Implemented in GprsSIM900.

4.3.2.4 virtual unsigned char Gprs::close ( ) [pure virtual]

Close TCP or UDP Connection.

Returns

Implemented in GprsSIM900.

4.3.2.5 virtual unsigned char Gprs::configureDns (const char * *primary*, const char * *secondary*) [pure virtual]

Configure Domain Name Server.

Returns

Implemented in GprsSIM900.

4.3.2.6 virtual unsigned char Gprs::configureServer (unsigned char *mode*, unsigned int *port*) [pure virtual]

Configure Module as Server.

Returns

Implemented in GprsSIM900.

4.3.2.7 virtual unsigned char Gprs::obtainIp (unsigned char *ip[4]*) [pure virtual]

Get Local IP Address.

Returns the the IP address assigned from GPRS or CSD in 4 bytes format.

Parameters

entry	Phonebook entry.

Returns

Implemented in GprsSIM900.

4.3.2.8 virtual unsigned char Gprs::open (const char * *mode*, const char * *address*, unsigned int *port*) [pure virtual]

Start Up TCP or UDP Connection.

Returns

Implemented in GprsSIM900.

4.3.2.9 virtual unsigned char Gprs::open (char *connection*, const char * *mode*, const char * *address*, unsigned int *port*) [pure virtual]

Start Up TCP or UDP Connection.

Returns

Implemented in GprsSIM900.

4.3.2.10 virtual unsigned char Gprs::resolve (const char * *name*, unsigned char * *buf*) [pure virtual]

Query the IP Address of Given Domain Name.

Returns

4.3.2.11 virtual unsigned int Gprs::send (unsigned char * *buf*, unsigned int *len*) [pure virtual]

Send Data Through TCP or UDP Connection.

Returns

Implemented in GprsSIM900.

4.3.2.12 virtual unsigned int Gprs::send (char *connection*, unsigned char * *buf*, unsigned int *len*) [pure virtual]

Send Data Through TCP or UDP Connection.

Returns

Implemented in GprsSIM900.

4.3.2.13 virtual unsigned char Gprs::shutdown ( ) [pure virtual]

Deactivate GPRS PDP Context.

Returns

Implemented in GprsSIM900.

4.3.2.14 virtual unsigned char Gprs::status ( ) [pure virtual]

Query Current Connection Status.

Returns

Implemented in GprsSIM900.

4.3.2.15 virtual unsigned char Gprs::status (char *connection*) [pure virtual]

Query Current Connection Status.

Parameters

connection	In muiti connection, it refears to the connection number. -1 otherwise;

Returns

Implemented in GprsSIM900.

4.3.2.16 virtual unsigned char Gprs::transmittingState (void * *stateStruct*) [pure virtual]

Query Previous Connection Data Transmitting State.

Parameters

stateStruct	Pointer to the state struct.

Implemented in GprsSIM900.

4.3.2.17 virtual unsigned char Gprs::transmittingState (char *connection*, void * *stateStruct*) [pure virtual]

Query Previous Connection Data Transmitting State.

Parameters

connection	In muiti connection, it refears to the connection number. -1 otherwise.
stateStruct	Pointer to the state struct.

Implemented in GprsSIM900.

4.3.2.18 virtual unsigned char Gprs::useMultiplexer (bool *use*) [pure virtual]

Start Up Multi-IP Connection.

Enable or disable multi IP connection.

Parameters

use	0 disables multi IP connection and 1 enables.

Returns

Implemented in GprsSIM900.

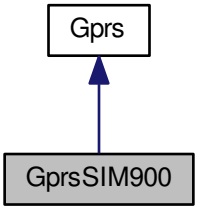
The documentation for this class was generated from the following file:

- Gprs.h

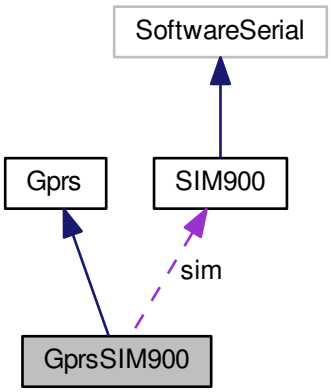
4.4 GprsSIM900 Class Reference

`#include <GprsSIM900.h>`

Inheritance diagram for GprsSIM900:



Collaboration diagram for GprsSIM900:



Classes

- struct TransmittingState

Public Types

- enum OperationResult { OK = 0, ERROR = 1, COMMAND_TOO_LONG = 2 }
- enum DnsResolution {
  NOT_AUTHORIZATION = 0, INVALID_PARAMTER = 1, NETWORK_ERROR = 2, NO_SERVER = 3,
  TIMEOUT = 4, NO_CONFIGURATION = 5, NO_MEMORY = 6, SUCCESS = 0xff }
- enum ConnectionState {
  IP_INITIAL = 0, IP_START = 1, IP_CONFIG = 2, IP_GPRSACT = 3,
  IP_STATUS = 4, CONNECTING_OR_LISTENING = 5, CONNECT_OK = 6, CLOSING = 7,
  CLOSED = 8, PDP_DEACT = 9, ERROR_WHEN_QUERING = 0xff }

Public Member Functions

- GprsSIM900 (SIM900 *sim)
- virtual ~GprsSIM900 ()
- unsigned char begin (long bound)
- unsigned char useMultiplexer (bool use)
- unsigned char attach (const char *apn, const char *login, const char *password)
- unsigned char bringUp ()
- unsigned char obtainIp (unsigned char ip[4])
- unsigned char status (char connection)
- unsigned char status ()
- unsigned char configureDns (const char *primary, const char *secondary)
- unsigned char open (const char *mode, const char *address, unsigned int port)
- unsigned char open (char connection, const char *mode, const char *address, unsigned int port)
- unsigned int send (unsigned char *buf, unsigned int len)
- unsigned int send (char connection, unsigned char *buf, unsigned int len)
- unsigned char close (char connection)
- unsigned char close ()
- unsigned char resolve (const char *name, unsigned char ip[4])
- unsigned char configureServer (unsigned char mode, unsigned int port)
- unsigned char shutdown ()
- unsigned char transmittingState (void *stateStruct)
- unsigned char transmittingState (char connection, void *stateStruct)

Static Public Member Functions

- unsigned static char parseIp (const char *buf, unsigned char ip[4])

Private Attributes

- SIM900 * sim
- bool multiplexed

4.4.1 Detailed Description

Definition at line 37 of file GprsSIM900.h.

4.4.2 Member Enumeration Documentation

4.4.2.1 enum GprsSIM900::ConnectionState

Enumerator

IP_INITIAL

IP_START

IP_CONFIG

IP_GPRSACT

IP_STATUS

CONNECTING_OR_LISTENING

CONNECT_OK

CLOSING

CLOSED

PDP_DEACT

ERROR_WHEN_QUERING

Definition at line 68 of file GprsSIM900.h.

4.4.2.2 enum GprsSIM900::DnsResolution

Enumerator

NOT_AUTHORIZATION

INVALID_PARAMTER

NETWORK_ERROR

NO_SERVER

TIMEOUT

NO_CONFIGURATION

NO_MEMORY

SUCCESS

Definition at line 57 of file GprsSIM900.h.

4.4.2.3 enum GprsSIM900::OperationResult

Enumerator

OK

ERROR

COMMAND_TOO_LONG

Definition at line 51 of file GprsSIM900.h.

4.4.3 Constructor & Destructor Documentation

4.4.3.1 GprsSIM900::GprsSIM900 (SIM900 * *sim*)

Public constructor.

Parameters

sim	The SIM900 pointer.

Definition at line 17 of file GprsSIM900.cpp.

4.4.3.2 virtual GprsSIM900::~GprsSIM900 ( ) [inline], [virtual]

Definition at line 98 of file GprsSIM900.h.

4.4.4 Member Function Documentation

4.4.4.1 unsigned char GprsSIM900::attach (const char * *apn*, const char * *login*, const char * *password*) [virtual]

Start Task and Set APN, LOGIN, PASSWORD.

Each parameter must be \0 teminated.

Example:

AT+CSTT="tim.br","tim","tim"

< OK

Parameters

apn	The apn access point name.
login	The GPRS user name.
password	The GPRS password.

Returns

Implements Gprs.

Definition at line 40 of file GprsSIM900.cpp.

4.4.4.2 unsigned char GprsSIM900::begin (long *bound*)

Initializes the device.

Parameters

The	bound rate to be used.

Returns

> 0 if success, 0 otherwise.

Definition at line 21 of file GprsSIM900.cpp.

4.4.4.3 unsigned char GprsSIM900::bringUp ( ) [virtual]

Bring Up Wireless Connection with GPRS or CSD.

Example:

AT+CIICR

< OK

Connects to the GPRS network.

Returns

Implements Gprs.

Definition at line 52 of file GprsSIM900.cpp.

4.4.4.4 unsigned char GprsSIM900::close (char *connection*) [virtual]

Close TCP or UDP Connection.

Example:

AT+CIPCLOSE=1[,connection]

< CLOSE OK

Parameters

connection	If multi-IP connection (+CIPMUX=1) 0..7 A numeric parameter which indicates the connection number

Returns

OperationResult

Implements Gprs.

Definition at line 160 of file GprsSIM900.cpp.

4.4.4.5 unsigned char GprsSIM900::close ( ) [virtual]

Close TCP or UDP Connection.

Returns

Implements Gprs.

Definition at line 175 of file GprsSIM900.cpp.

4.4.4.6 unsigned char GprsSIM900::configureDns (const char * *primary,* const char * *secondary*) [virtual]

Configure Domain Name Server.

Returns

Implements Gprs.

Definition at line 111 of file GprsSIM900.cpp.

4.4.4.7 unsigned char GprsSIM900::configureServer (unsigned char *mode,* unsigned int *port*) [virtual]

Configure Module as Server.

This command is allowed to establish a TCP server only when the state is IP INITIAL or IP STATUS when it is in single state. In multi-IP state, the state is in IP STATUS only.

Example: AT+CIPSERVER=(0-CLOSE SERVER, 1-OPEN SERVER),(1,65535)

Parameters

mode	(0-CLOSE SERVER, 1-OPEN SERVER)
port	Port number

Returns

Implements Gprs.

Definition at line 203 of file GprsSIM900.cpp.

4.4.4.8 unsigned char GprsSIM900::obtainIp (unsigned char *ip[4]*) [virtual]

Get Local IP Address.

Returns the the IP address assigned from GPRS or CSD in 4 bytes format.

Example:

AT+CIFSR

< OK < < 100.83.135.48

Parameters

entry	Phonebook entry.

Returns

Implements Gprs.

Definition at line 58 of file GprsSIM900.cpp.

4.4.4.9 unsigned char GprsSIM900::open (const char * *mode,* const char * *address,* unsigned int *port*) [inline], [virtual]

Start Up TCP or UDP Connection.

Returns

Implements Gprs.

Definition at line 245 of file GprsSIM900.h.

4.4.4.10 unsigned char GprsSIM900::open (char *connection,* const char * *mode,* const char * *address,* unsigned int *port*) [virtual]

Start Up TCP or UDP Connection.

Example:

AT+CIPSTART="TCP","dalmirdasilva.com", "3000"

< OK < < CONNECT OK

Parameters

connection	If multi-IP connection (+CIPMUX=1) 0..7 A numeric parameter which indicates the connection number
mode	A string parameter(string should be included in quotation marks) which indicates the connection type "TCP" Establish a TCP connection "UDP" Establish a UDP connection
address	A string parameter(string should be included in quotation marks) which indicates remote server IP address
port	Remote server port

Returns

OperationResult

Implements Gprs.

Definition at line 121 of file GprsSIM900.cpp.

4.4.4.11 unsigned char GprsSIM900::parseIp (const char * *buf*, unsigned char *ip[4]*) [static]

Tries to parse an IP from string.

Parameters

buf	should have the following format: [0-9]{1,4}.[0-9]{1,4}.[0-9]{1,4}.[0-9]{1,4}
ip	whre to store the parsed ip, 4 bytes.

Definition at line 238 of file GprsSIM900.cpp.

4.4.4.12 unsigned char GprsSIM900::resolve (const char * *name*, unsigned char *ip[4]*)

Query the IP Address of Given Domain Name.

Example:

AT+CDNSGIP="www.google.com"

< OK < < +CDNSGIP: 1,"www.google.com","64.233.186.99"

Returns

DnsResolution

Parameters

name	Domain name. Should contains less than 256 bytes
ip	4-byte-long array where the ip will be placed

Definition at line 180 of file GprsSIM900.cpp.

4.4.4.13 unsigned int GprsSIM900::send (unsigned char * *buf*, unsigned int *len*) [inline], [virtual]

Send Data Through TCP or UDP Connection.

Example:

AT+CIPSEND= <0-7>,<length>

< >

data

DATA ACCEPT:<length>

Returns

Implements Gprs.

Definition at line 282 of file GprsSIM900.h.

4.4.4.14 unsigned int GprsSIM900::send (char *connection,* unsigned char * *buf,* unsigned int *len*) [virtual]

Send Data Through TCP or UDP Connection.

Returns

Implements Gprs.

Definition at line 142 of file GprsSIM900.cpp.

4.4.4.15 unsigned char GprsSIM900::shutdown ( ) [virtual]

Deactivate GPRS PDP Context.

Example:

AT+CIPSHUT

< SHUT OK

Returns

Implements Gprs.

Definition at line 212 of file GprsSIM900.cpp.

4.4.4.16 unsigned char GprsSIM900::status (char *connection*) [virtual]

Query Current Connection Status.

STATE: <state> If the module is set as server S: 0, <bearer>, <port>, <server state>=""> C: <n>,<bearer>, <TCP/UDP>, <IP address>="">, <port>, <client state>="">

<n> 0-7 A numeric parameter which indicates the connection number

<bearer> 0-1 GPRS bearer, default is 0

<server state>=""> OPENING LISTENING CLOSING

<client state>=""> INITIAL CONNECTING CONNECTED REMOTE CLOSING CLOSING CLOSED

<state> A string parameter(string should be included in quotation marks) which indicates the progress of connecting 0 IP INITIAL 1 IP START 2 IP CONFIG 3 IP GPRSACT 4 IP STATUS 5 TCP CONNECTING/UDP CONNECTING/SERVER LISTENING 6 CONNECT OK 7 TCP CLOSING/UDP CLOSING 8 TCP CLOSED/UDP CLOSED 9 PDP DEACT

In Multi-IP state: 0 IP INITIAL 1 IP START 2 IP CONFIG 3 IP GPRSACT 4 IP STATUS

Example:

AT+CIPSTATUS[=n]

< OK < < STATE: IP STATUS

Returns

Implements Gprs.

Definition at line 75 of file GprsSIM900.cpp.

4.4.4.17 unsigned char GprsSIM900::status ( ) [virtual]

Status.

Implements Gprs.

Definition at line 71 of file GprsSIM900.cpp.

4.4.4.18 unsigned char GprsSIM900::transmittingState (void * *stateStruct*) [inline], [virtual]

Query Previous Connection Data Transmitting State.

Implements Gprs.

Definition at line 359 of file GprsSIM900.h.

4.4.4.19 unsigned char GprsSIM900::transmittingState (char *connection,* void * *stateStruct*) [virtual]

Query Previous Connection Data Transmitting State.

Example:

AT+CIPACK=<n>

< +CIPACK: <txlen>, <acklen>, <nacklen>

<n> A numeric parameter which indicates the connection number

<txlen> The data amount which has been sent

<acklen> The data amount confirmed successfully by the server

<nacklen> The data amount without confirmation by the server

Parameters

stateStruct	Pointer to the TransmittingState structure.

Implements Gprs.

Definition at line 216 of file GprsSIM900.cpp.

4.4.4.20 unsigned char GprsSIM900::useMultiplexer (bool *use*) [virtual]

Start Up Multi-IP Connection.

Enable or disable multi IP connection.

Example:

+CIPMUX=0|1

< OK

Parameters

use	0 disables multi IP connection and 1 enables.

Returns

Implements Gprs.

Definition at line 29 of file GprsSIM900.cpp.

4.4.5 Member Data Documentation

4.4.5.1 bool GprsSIM900::multiplexed [private]

Multi connection.

Definition at line 47 of file GprsSIM900.h.

4.4.5.2 SIM900* GprsSIM900::sim [private]

SIM900 pointer.

Definition at line 42 of file GprsSIM900.h.

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

- GprsSIM900.h
- GprsSIM900.cpp

4.5 Phonebook Class Reference

`#include <Phonebook.h>`

Public Member Functions

- virtual unsigned char findEntries ()=0
- virtual unsigned char callNumber (unsigned char *number)=0
- virtual unsigned char callFromPhonebook (unsigned char position)=0
- virtual unsigned char callByPhonebookMatch (unsigned char *entry)=0
- virtual unsigned char redial ()=0
- virtual unsigned char disconnect ()=0
- virtual unsigned char setAutomaticallyAnswering (unsigned char rings)=0

4.5.1 Detailed Description

Arduino - Gsm driver.

Interface to phonebook.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 12 of file Phonebook.h.

4.5.2 Member Function Documentation

4.5.2.1 virtual unsigned char Phonebook::callByPhonebookMatch (unsigned char * *entry*) [pure virtual]

Originate Call to Phone Number in Memory Which Corresponds to Field.

This Command make the TA attempts to set up an outgoing call to stored number which has the name matching with the entry string.

Parameters

entry	Phonebook entry.

Returns

4.5.2.2 virtual unsigned char Phonebook::callFromPhonebook (unsigned char *position*) [pure virtual]

Originate Call to Phone Number in Current Memory.

This Command can be used to dial a phone number from current phonebook.

Parameters

position	Phonebook position.

Returns

4.5.2.3 virtual unsigned char Phonebook::callNumber (unsigned char * *number*) [pure virtual]

Mobile Originated Call to Dial A Number.

This Command can be used to set up outgoing voice, data or fax calls. It also serves to control supplementary services.

Parameters

number	The number to make the call to.

Returns

4.5.2.4 virtual unsigned char Phonebook::disconnect ( ) [pure virtual]

Disconnect Existing Connection.

Returns

4.5.2.5 virtual unsigned char Phonebook::findEntries ( ) [pure virtual]

Find Phonebook Entries.

Returns

4.5.2.6 virtual unsigned char Phonebook::redial ( ) [pure virtual]

Redial Last Telephone Number Used.

This Command redials the last voice and data call number used.

Returns

4.5.2.7 virtual unsigned char Phonebook::setAutomaticallyAnswering (unsigned char *rings*) [pure virtual]

Set number of rings before automatically answering the call.

Parameters

rings	Number of rings before automatically answering. 0 means disable.

Returns

0 if error, > 0 otherwise.

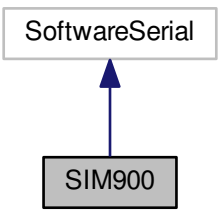
The documentation for this class was generated from the following file:

- Phonebook.h

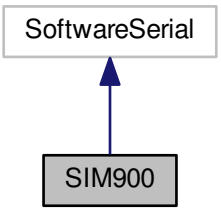
4.6 SIM900 Class Reference

`#include <SIM900.h>`

Inheritance diagram for SIM900:



Collaboration diagram for SIM900:



Public Types

- enum DisconnectParamter {
  ALL_CALLS_ON_CHANNEL = 0, ALL_CALL_ON_ALL_CHANNELS = 1, ALL_CS_ON_CHANNEL = 2, ALL_GPRS_ON_CHANNEL = 3,
  ALL_BUT_WAITING_ON_CHANNEL = 4, ALL_WAITING_ON_CHANNEL = 5 }

Public Member Functions

- SIM900 (unsigned char receivePin, unsigned char transmitPin)
- SIM900 (unsigned char receivePin, unsigned char transmitPin, unsigned char resetPin, unsigned char powerPin)
- virtual ~SIM900 ()
- unsigned char begin (long bound)
- void softReset ()
- void softPower ()
- unsigned char * getLastResponse ()
- bool sendCommandExpecting (const char *command, const char *expectation, bool append, unsigned long timeout)
- bool sendCommandExpecting (const char *command, const char *expectation, bool append)
- bool sendCommandExpecting (const char *command, const char *expectation, unsigned long timeout)
- bool sendCommandExpecting (const char *command, const char *expectation)
- bool doesResponseContains (const char *expectation)
- unsigned int sendCommand (const char *command, bool append, unsigned long timeout)
- unsigned int sendCommand (const char *command, bool append)
- unsigned int sendCommand (const char *command, unsigned long timeout)
- unsigned int sendCommand (const char *command)
- unsigned int sendCommand ()
- unsigned int readResponse (unsigned long timeout)
- unsigned int readResponse (unsigned long timeout, bool append)
- void setEcho (bool echo)
- unsigned char disconnect (DisconnectParamter param)
- bool wasResponseFullyRead ()
- const char * findInResponse (const char *str)
- int waitUntilReceive (const char *str, unsigned int timeout)
- void discardBuffer ()

Private Attributes

- unsigned char rxBuffer [SIM900_RX_BUFFER_SIZE]
- unsigned int rxBufferPos
- bool echo
- unsigned char resetPin
- unsigned char powerPin
- bool responseFullyRead
- bool softResetAndPowerEnabled

4.6.1 Detailed Description

Definition at line 26 of file SIM900.h.

4.6.2 Member Enumeration Documentation

4.6.2.1 enum SIM900::DisconnectParamter

Enumerator

ALL_CALLS_ON_CHANNEL

ALL_CALL_ON_ALL_CHANNELS

ALL_CS_ON_CHANNEL

ALL_GPRS_ON_CHANNEL

ALL_BUT_WAITING_ON_CHANNEL

ALL_WAITING_ON_CHANNEL

Definition at line 66 of file SIM900.h.

4.6.3 Constructor & Destructor Documentation

4.6.3.1 SIM900::SIM900 (unsigned char *receivePin,* unsigned char *transmitPin*)

Public constructor.

Parameters

serial	

Definition at line 17 of file SIM900.cpp.

4.6.3.2 SIM900::SIM900 (unsigned char *receivePin,* unsigned char *transmitPin,* unsigned char *resetPin,* unsigned char *powerPin*)

Public constructor.

Parameters

serial	

Definition at line 21 of file SIM900.cpp.

4.6.3.3 SIM900::~SIM900 ( ) [virtual]

Virtual destructor.

Definition at line 30 of file SIM900.cpp.

4.6.4 Member Function Documentation

4.6.4.1 unsigned char SIM900::begin (long *bound*)

Initializes the device.

Parameters

The	bound rate to be used.

Returns

0 if not success, $>$ 0 otherwise.

Definition at line 33 of file SIM900.cpp.

4.6.4.2 void SIM900::discardBuffer ( )

Discards the internal buffer.

Definition at line 145 of file SIM900.cpp.

4.6.4.3 unsigned char SIM900::disconnect (DisconnectParamter *param*)

Definition at line 122 of file SIM900.cpp.

4.6.4.4 bool SIM900::doesResponseContains (const char * *expectation*)

Checks if the last response contains the given sub-string.

Parameters

expectation	The expectation string.

Returns

Definition at line 65 of file SIM900.cpp.

4.6.4.5 const char * SIM900::findInResponse (const char * *str*)

Searches the response for a given string.

Parameters

str	The string to be searched inside response

Returns

The pointer to the first found string, NULL if not found

Definition at line 131 of file SIM900.cpp.

4.6.4.6 unsigned char* SIM900::getLastResponse ( ) [inline]

Get a pointer to the last response.

Returns

Definition at line 142 of file SIM900.h.

4.6.4.7 unsigned int SIM900::readResponse (unsigned long *timeout*) [inline]

Reads the response from the device.

Parameters

timeout	The maximum time to perform the op.

Returns

How many bytes was received. 0 if timeout.

Definition at line 253 of file SIM900.h.

4.6.4.8 unsigned int SIM900::readResponse (unsigned long *timeout*, bool *append*)

Reads the response from the device.

Parameters

timeout	The maximum time to perform the op.
append	Append the response in the internal buffer.

Returns

How many bytes was received. 0 if timeout.

Definition at line 80 of file SIM900.cpp.

4.6.4.9 unsigned int SIM900::sendCommand (const char * *command*, bool *append*, unsigned long *timeout*)

Sends a command to the device.

Parameters

command	The command string, should be \0 ended.
timeout	The maximum time to perform the op.

Returns

Definition at line 69 of file SIM900.cpp.

4.6.4.10 unsigned int SIM900::sendCommand (const char * *command*, bool *append*) [inline]

Sends a command to the device.

Parameters

command	The command string, should be \0 ended.
append	Boolean saying if the AT must be appended.

Returns

Definition at line 215 of file SIM900.h.

4.6.4.11 unsigned int SIM900::sendCommand (const char * *command*, unsigned long *timeout*) [inline]

Sends a command to the device.

Parameters

command	The command string, should be \0 ended.
append	Boolean saying if the AT must be appended.

Returns

Definition at line 226 of file SIM900.h.

4.6.4.12 unsigned int SIM900::sendCommand (const char * *command*) [inline]

Sends a command to the device.

Parameters

command	The command string, should be \0 ended.

Returns

Definition at line 236 of file SIM900.h.

4.6.4.13 unsigned int SIM900::sendCommand ( ) [inline]

Sends a command to the device.

Definition at line 243 of file SIM900.h.

4.6.4.14 bool SIM900::sendCommandExpecting (const char * *command*, const char * *expectation*, bool *append*, unsigned long *timeout*)

Sends a command expecting some result.

Parameters

command	The command string, should be \0 ended.
expectation	The expectation string.
timeout	The maximum time to perform the op.

Returns

Definition at line 58 of file SIM900.cpp.

4.6.4.15 bool SIM900::sendCommandExpecting (const char * *command*, const char * *expectation*, bool *append*) [inline]

Sends a command expecting some result.

Parameters

command	The command string, should be \0 ended.
expectation	The expectation string.
append	If should append AT+ in the command.

Returns

Definition at line 164 of file SIM900.h.

4.6.4.16 bool SIM900::sendCommandExpecting (const char * *command,* const char * *expectation,* unsigned long *timeout*) [inline]

Sends a command expecting some result.

Parameters

command	The command string, should be \0 ended.
expectation	The expectation string.
timeout	Timeout.

Returns

Definition at line 176 of file SIM900.h.

4.6.4.17 bool SIM900::sendCommandExpecting (const char * *command*, const char * *expectation*) [inline]

Sends a command expecting some result.

Parameters

command	The command string, should be \0 ended.
expectation	The expectation string.

Returns

Definition at line 187 of file SIM900.h.

4.6.4.18 void SIM900::setEcho (bool *echo*)

Configures echo mode.

Parameters

echo	

Definition at line 113 of file SIM900.cpp.

4.6.4.19 void SIM900::softPower ( )

Soft controlled Power on/off.

Definition at line 50 of file SIM900.cpp.

4.6.4.20 void SIM900::softReset ( )

Soft controlled Reset.

Definition at line 42 of file SIM900.cpp.

4.6.4.21 int SIM900::waitUntilReceive (const char * *str*, unsigned int *timeout*)

Keeps reading the response until finds the str or timeout.

Parameters

str	String it tries to find
timeout	Timeout in millis

Returns

The position to the first char it finds in the internal buffer, -1 otherwise

Definition at line 135 of file SIM900.cpp.

4.6.4.22 bool SIM900::wasResponseFullyRead ( )

Checks if the rx is fully read.

If false, a new call to readResponse will read more data from tx placing the new data over the current data in it. Consecutive calls to readResponse will write data into the same buffer.

use getLastResponse to consume the current data and then call readResponse again in case tx was not fully read.

Definition at line 127 of file SIM900.cpp.

4.6.5 Member Data Documentation

4.6.5.1 bool SIM900::echo [private]

Using echo.

Definition at line 41 of file SIM900.h.

4.6.5.2 unsigned char SIM900::powerPin [private]

Soft power on/off pin.

Definition at line 51 of file SIM900.h.

4.6.5.3 unsigned char SIM900::resetPin [private]

Soft reset pin.

Definition at line 46 of file SIM900.h.

4.6.5.4 bool SIM900::responseFullyRead [private]

Bool indicating the last command's response was fully read by readResponse method.

Definition at line 57 of file SIM900.h.

4.6.5.5 unsigned char SIM900::rxBuffer[SIM900_RX_BUFFER_SIZE] [private]

RX buffer.

Definition at line 31 of file SIM900.h.

4.6.5.6 unsigned int SIM900::rxBufferPos [private]

RX buffer position.

Definition at line 36 of file SIM900.h.

4.6.5.7 bool SIM900::softResetAndPowerEnabled [private]

Enable soft power and reset.

Definition at line 62 of file SIM900.h.

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

- SIM900.h
- SIM900.cpp

4.7 Sms Class Reference

```
#include <Sms.h>
```

Public Member Functions

- virtual unsigned char remove (unsigned char index, unsigned char flags)=0
- virtual unsigned char format (bool format)=0
- virtual unsigned char bringUp ()=0
- virtual unsigned char obtainIp (unsigned char *buf)=0
- virtual unsigned char status ()=0
- virtual unsigned char configureDns (const char *primary, const char *secondary)=0
- virtual unsigned char open (unsigned char mode, unsigned char *address, unsigned char port)=0
- virtual unsigned char open (char connection, unsigned char mode, unsigned char *address, unsigned char port)=0
- virtual unsigned char close (char connection)=0
- virtual unsigned char resolve (unsigned char *name, unsigned char *buf, unsigned int len)=0
- virtual unsigned char send (unsigned char *buf)=0
- virtual unsigned char send (char connection, unsigned char *buf, unsigned int len)=0
- virtual unsigned char setUpServer (unsigned char mode, unsigned int port)=0
- virtual unsigned char shutdown ()=0

4.7.1 Detailed Description

Arduino - Gsm driver.

Sms.h

Interface to calls.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 14 of file Sms.h.

4.7.2 Member Function Documentation

4.7.2.1 virtual unsigned char Sms::bringUp ( ) [pure virtual]

Bring Up Wireless Connection with Sms or CSD.

Connects to the Sms network.

Returns

4.7.2.2 virtual unsigned char Sms::close (char *connection*) [pure virtual]

Close TCP or UDP Connection.

Returns

4.7.2.3 virtual unsigned char Sms::configureDns (const char * *primary*, const char * *secondary*) [pure virtual]

Configure Domain Name Server.

Returns

4.7.2.4 virtual unsigned char Sms::format (bool *format*) [pure virtual]

Select SMS Message Format.

Parameters

format	Message format.

Returns

4.7.2.5 virtual unsigned char Sms::obtainIp (unsigned char * *buf*) [pure virtual]

Get Local IP Address.

Returns the the IP address assigned from Sms or CSD in 4 bytes format.

Parameters

entry	Phonebook entry.

Returns

4.7.2.6 virtual unsigned char Sms::open (unsigned char *mode*, unsigned char * *address*, unsigned char *port*) [pure virtual]

Start Up TCP or UDP Connection.

Returns

4.7.2.7 virtual unsigned char Sms::open (char *connection*, unsigned char *mode*, unsigned char * *address*, unsigned char *port*) [pure virtual]

Start Up TCP or UDP Connection.

Returns

4.7.2.8 virtual unsigned char Sms::remove (unsigned char *index*, unsigned char *flags*) [pure virtual]

Delete SMS Message.

Parameters

index	Message location.
flags	Deletion flags.

Returns

4.7.2.9 virtual unsigned char Sms::resolve (unsigned char * *name*, unsigned char * *buf*, unsigned int *len*) [pure virtual]

Query the IP Address of Given Domain Name.

Returns

4.7.2.10 virtual unsigned char Sms::send (unsigned char * *buf*) [pure virtual]

Send Data Through TCP or UDP Connection.

Returns

4.7.2.11 virtual unsigned char Sms::send (char *connection,* unsigned char * *buf,* unsigned int *len*) [pure virtual]

Send Data Through TCP or UDP Connection.

Returns

4.7.2.12 virtual unsigned char Sms::setUpServer (unsigned char *mode,* unsigned int *port*) [pure virtual]

Configure Module as Server.

Returns

4.7.2.13 virtual unsigned char Sms::shutdown ( ) [pure virtual]

Deactivate Sms PDP Context.

Returns

4.7.2.14 virtual unsigned char Sms::status ( ) [pure virtual]

Query Current Connection Status.

Returns

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

- Sms.h

4.8 GprsSIM900::TransmittingState Struct Reference

`#include <GprsSIM900.h>`

Public Member Functions

- TransmittingState ()

Public Attributes

- unsigned int txlen
- unsigned int acklen
- unsigned int nacklen

4.8.1 Detailed Description

Definition at line 82 of file GprsSIM900.h.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 GprsSIM900::TransmittingState::TransmittingState ( ) [inline]

Definition at line 86 of file GprsSIM900.h.

4.8.3 Member Data Documentation

4.8.3.1 unsigned int GprsSIM900::TransmittingState::acklen

Definition at line 84 of file GprsSIM900.h.

4.8.3.2 unsigned int GprsSIM900::TransmittingState::nacklen

Definition at line 85 of file GprsSIM900.h.

4.8.3.3 unsigned int GprsSIM900::TransmittingState::txlen

Definition at line 83 of file GprsSIM900.h.

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

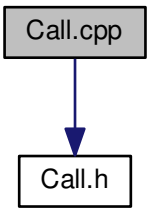
- GprsSIM900.h

5 File Documentation

5.1 Call.cpp File Reference

```
#include "Call.h"
```

Include dependency graph for Call.cpp:



Macros

- #define __ARDUINO_DRIVER_GSM_CALL_CPP__ 1

5.1.1 Macro Definition Documentation

5.1.1.1 #define __ARDUINO_DRIVER_GSM_CALL_CPP__ 1

Arduino - Gsm driver.

Call.cpp

Interface to calls.

Author

Dalmir da Silva dalmirdasilva@gmail.com

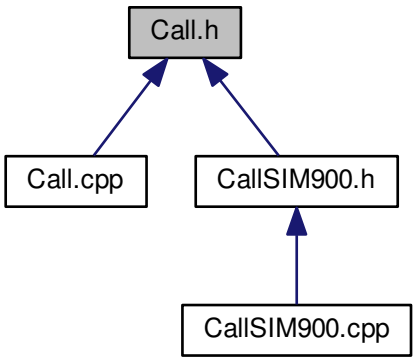
Definition at line 12 of file Call.cpp.

5.2 Call.cpp

```
00001
00011 #ifndef __ARDUINO_DRIVER_GSM_CALL_CPP__
00012 #define __ARDUINO_DRIVER_GSM_CALL_CPP__ 1
00013
00014 #include "Call.h"
00015
00016 #endif /* __ARDUINO_DRIVER_GSM_CALL_CPP__ */
```

5.3 Call.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class Call

5.4 Call.h

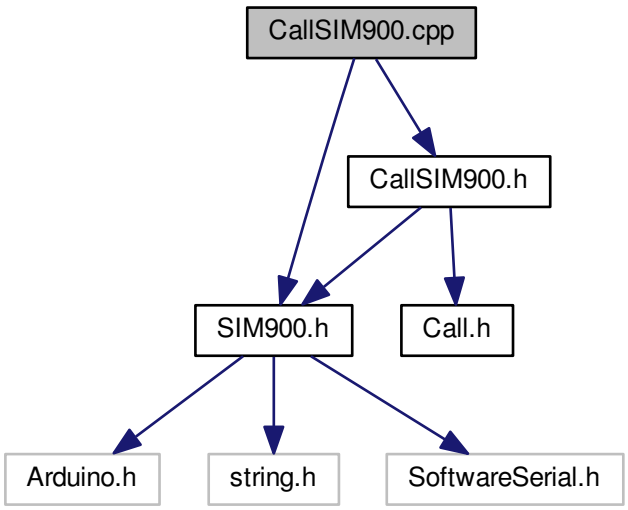
```
00001
00011 #ifndef __ARDUINO_DRIVER_GSM_CALL_H__
00012 #define __ARDUINO_DRIVER_GSM_CALL_H__ 1
00013
```

```
00014 class Call {
00015
00016 public:
00017
00025     virtual unsigned char answer() = 0;
00026
00036     virtual unsigned char callNumber(unsigned char *number) = 0;
00037
00046     virtual unsigned char callFromPhonebook(unsigned char position) = 0;
00047
00057     virtual unsigned char callByPhonebookMatch(unsigned char *entry) = 0;
00058
00066     virtual unsigned char redial() = 0;
00067
00073     virtual unsigned char disconnect() = 0;
00074
00081     virtual unsigned char setAutomaticallyAnswering(unsigned char rings) = 0;
00082 };
00083
00084 #endif /* __ARDUINO_DRIVER_GSM_CALL_H__ */
```

5.5 CallSIM900.cpp File Reference

```
#include "CallSIM900.h"
#include <SIM900.h>
```

Include dependency graph for CallSIM900.cpp:



Macros

- #define __ARDUINO_DRIVER_GSM_CALL_SIM900_CPP__ 1

5.5.1 Macro Definition Documentation

5.5.1.1 #define __ARDUINO_DRIVER_GSM_CALL_SIM900_CPP__ 1

Arduino - Gsm driver.

CallSIM900.cpp

Interface to calls.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 12 of file CallSIM900.cpp.

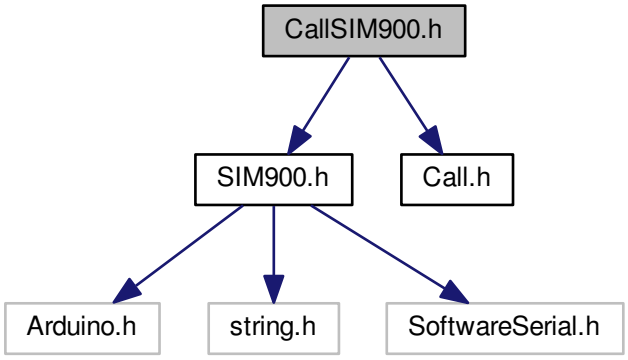
5.6 CallSIM900.cpp

```
00001
00011 #ifndef __ARDUINO_DRIVER_GSM_CALL_SIM900_CPP__
00012 #define __ARDUINO_DRIVER_GSM_CALL_SIM900_CPP__ 1
00013
00014 #include "CallSIM900.h"
00015 #include <SIM900.h>
00016
00017 CallSIM900::CallSIM900(SIM900 *sim)
00018         : sim(sim) {
00019 }
00020
00021 CallSIM900::~CallSIM900() {
00022 }
00023
00024 unsigned char CallSIM900::answer() {
00025     return (unsigned char) sim->sendCommandExpecting((const char*) "A", (const char
      *) "CONNECT", true);
00026 }
00027
00028 unsigned char CallSIM900::callNumber(unsigned char *number) {
00029 }
00030
00031 unsigned char CallSIM900::callFromPhonebook(unsigned char position) {
00032     unsigned char cmd[4] = "D";
00033     itoa(position, (char *) &cmd[1], 10);
00034     sim->sendCommand((const char *) cmd, true);
00035     return checkResponse();
00036 }
00037
00038 unsigned char CallSIM900::callByPhonebookMatch(unsigned char *entry) {
00039 }
00040
00041 unsigned char CallSIM900::redial() {
00042
00043 }
00044
00045 unsigned char CallSIM900::disconnect() {
00046     return sim->disconnect(SIM900::ALL_CALL_ON_ALL_CHANNELS);
00047 }
00048
00049 unsigned char CallSIM900::checkResponse() {
00050     CallResponse response = OK;
00051     if (sim->doesResponseContains((const char *) "OK")) {
00052         response = OK;
00053     } else if (sim->doesResponseContains((const char *) "CME")) {
00054         response = CME_ERROR;
00055     } else if (sim->doesResponseContains((const char *) "DIALTONE")) {
00056         response = NO_DIALTONE;
00057     } else if (sim->doesResponseContains((const char *) "BUSY")) {
00058         response = BUSY;
00059     } else if (sim->doesResponseContains((const char *) "CARRIER")) {
00060         response = NO_CARRIER;
00061     } else if (sim->doesResponseContains((const char *) "ANSWER")) {
00062         response = NO_ANSWER;
00063     } else if (sim->doesResponseContains((const char *) "CONNECT")) {
00064         response = CONNECT_TEXT;
00065     }
00066     return response;
00067 }
00068
00069 #endif /* __ARDUINO_DRIVER_GSM_CALL_SIM900_CPP__ */
```

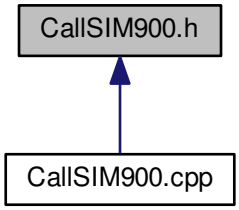
5.7 CallSIM900.h File Reference

```
#include <SIM900.h>
#include <Call.h>
```

Include dependency graph for CallSIM900.h:



This graph shows which files directly or indirectly include this file:



Classes

- class CallSIM900

5.8 CallSIM900.h

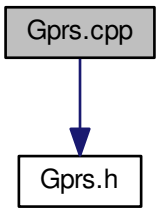
```
00001
00011 #ifndef __ARDUINO_DRIVER_GSM_CALL_SIM900_H__
00012 #define __ARDUINO_DRIVER_GSM_CALL_SIM900_H__ 1
00013
00014 #include <SIM900.h>
00015 #include <Call.h>
00016
00017 class CallSIM900 : public Call {
00018
00019     SIM900 *sim;
00020
00021 public:
00022
00023     enum CallResponse {
00024
00025         // Success
00026         OK = 0,
```

```
00027
00028         // If error is related to ME functionality
00029         CME_ERROR = 1,
00030
00031         // If no dial tone and (parameter setting ATX2 or ATX4)
00032         NO_DIALTONE = 2,
00033
00034         // If busy and (parameter setting ATX3 or ATX4)
00035         BUSY = 3,
00036
00037         // If a connection cannot be established
00038         NO_CARRIER = 4,
00039
00040         // If the remote station does not answer
00041         NO_ANSWER = 5,
00042
00043         // If connection successful and non-voice call.
00044         // CONNECT<text> TA switches to data mode.
00045         // Note: <text> output only if ATX<value> parameter setting with the
00046         // <value> >0
00047         CONNECT_TEXT = 6
00048     };
00049
00050     CallSIM900(SIM900 *sim);
00051
00052     virtual ~CallSIM900();
00053
00061     unsigned char answer();
00062
00072     unsigned char callNumber(unsigned char *number);
00073
00082     unsigned char callFromPhonebook(unsigned char position);
00083
00093     unsigned char callByPhonebookMatch(unsigned char *entry);
00094
00102     unsigned char redial();
00103
00109     unsigned char disconnect();
00110
00117     unsigned char setAutomaticallyAnswering(unsigned char rings);
00118
00122     unsigned char checkResponse();
00123 };
00124
00125 #endif /* __ARDUINO_DRIVER_GSM_CALL_SIM900_H__ */
```

5.9 Gprs.cpp File Reference

`#include "Gprs.h"`

Include dependency graph for Gprs.cpp:



Macros

- #define __ARDUINO_DRIVER_GSM_GPRS_CPP__ 1

5.9.1 Macro Definition Documentation

5.9.1.1 #define __ARDUINO_DRIVER_GSM_GPRS_CPP__ 1

Arduino - Gsm driver.

Gprs.cpp

Interface to calls.

Author

Dalmir da Silva dalmirdasilva@gmail.com

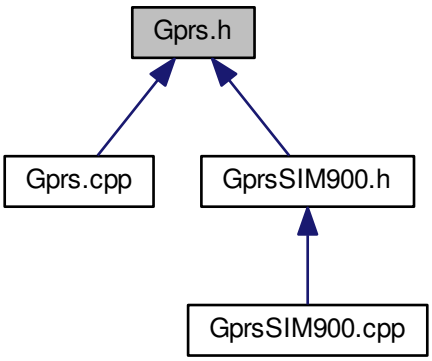
Definition at line 12 of file Gprs.cpp.

5.10 Gprs.cpp

```
00001
00011 #ifndef __ARDUINO_DRIVER_GSM_GPRS_CPP__
00012 #define __ARDUINO_DRIVER_GSM_GPRS_CPP__ 1
00013
00014 #include "Gprs.h"
00015
00016 #endif /* __ARDUINO_DRIVER_GSM_GPRS_CPP__ */
```

5.11 Gprs.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class Gprs

5.12 Gprs.h

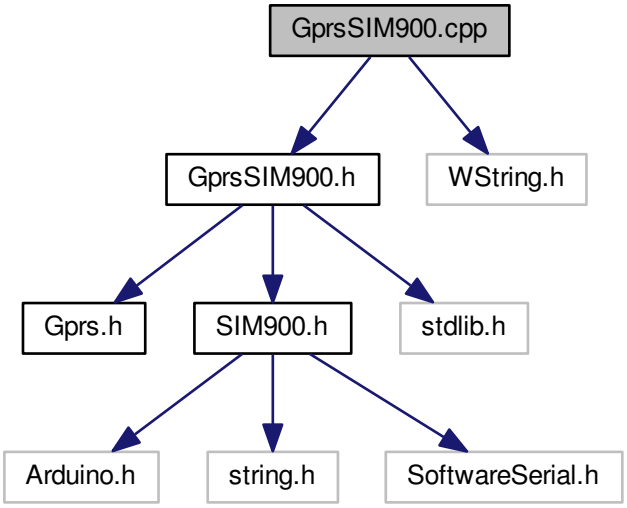
```
00001
00011 #ifndef __ARDUINO_DRIVER_GSM_GPRS_H__
00012 #define __ARDUINO_DRIVER_GSM_GPRS_H__ 1
00013
```

```
00014 class Gprs {
00015
00016 public:
00017
00026     virtual unsigned char useMultiplexer(bool use) = 0;
00027
00038     virtual unsigned char attach(const char *apn, const char *login, const char *password) = 0;
00039
00047     virtual unsigned char bringUp() = 0;
00048
00058     virtual unsigned char obtainIp(unsigned char ip[4]) = 0;
00059
00065     virtual unsigned char status() = 0;
00066
00073     virtual unsigned char status(char connection) = 0;
00074
00080     virtual unsigned char configureDns(const char *primary, const char *secondary) = 0;
00081
00087     virtual unsigned char open(const char *mode, const char *address, unsigned int port) = 0;
00088
00094     virtual unsigned char open(char connection, const char *mode, const char *address, unsigned int
      port) = 0;
00095
00102     virtual unsigned char close(char connection) = 0;
00103
00109     virtual unsigned char close() = 0;
00110
00116     virtual unsigned char resolve(const char *name, unsigned char *buf) = 0;
00117
00123     virtual unsigned int send(unsigned char *buf, unsigned int len) = 0;
00124
00130     virtual unsigned int send(char connection, unsigned char *buf, unsigned int len) = 0;
00131
00137     virtual unsigned char configureServer(unsigned char mode, unsigned int port) = 0;
00138
00144     virtual unsigned char shutdown() = 0;
00145
00151     virtual unsigned char transmittingState(void *stateStruct) = 0;
00152
00159     virtual unsigned char transmittingState(char connection, void *stateStruct) = 0;
00160 };
00161
00162 #endif /* __ARDUINO_DRIVER_GSM_GPRS_H__ */
```

5.13 GprsSIM900.cpp File Reference

```
#include "GprsSIM900.h"
#include <WString.h>
```

Include dependency graph for GprsSIM900.cpp:



Macros

- #define __ARDUINO_DRIVER_GSM_GPRS_SIM900_CPP__ 1

5.13.1 Macro Definition Documentation

5.13.1.1 #define __ARDUINO_DRIVER_GSM_GPRS_SIM900_CPP__ 1

Arduino - Gsm driver.

GprsSIM900.h

GPRS connection using SIM900.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 12 of file GprsSIM900.cpp.

5.14 GprsSIM900.cpp

```
00001
00011 #ifndef __ARDUINO_DRIVER_GSM_GPRS_SIM900_CPP__
00012 #define __ARDUINO_DRIVER_GSM_GPRS_SIM900_CPP__ 1
00013
00014 #include "GprsSIM900.h"
00015 #include <WString.h>
00016
00017 GprsSIM900::GprsSIM900(SIM900 *sim)
00018         : sim(sim), multiplexed(false) {
00019 }
00020
00021 unsigned char GprsSIM900::begin(long bound) {
00022     if (sim->begin(bound) == 0) {
00023         return 0;
```

```
00024     }
00025     sim->setEcho(false);
00026     return 1;
00027 }
00028
00029 unsigned char GprsSIM900::useMultiplexer(bool use) {
00030     bool expected;
00031     char command[] = "+CIPMUX=0";
00032     if (use) {
00033         command[8] = '1';
00034     }
00035     multiplexed = use;
00036     expected = sim->sendCommandExpecting(command, "OK", true);
00037     return (unsigned char) (expected ? GprsSIM900::OK :
      GprsSIM900::ERROR);
00038 }
00039
00040 unsigned char GprsSIM900::attach(const char *apn, const char *login, const char *password
      ) {
00041     bool expected;
00042     sim->write("AT+CSTT=\"");
00043     sim->write(apn);
00044     sim->write("\",\"");
00045     sim->write(login);
00046     sim->write("\",\"");
00047     sim->write(password);
00048     expected = sim->sendCommandExpecting("\"", "OK");
00049     return (unsigned char) (expected ? GprsSIM900::OK :
      GprsSIM900::ERROR);
00050 }
00051
00052 unsigned char GprsSIM900::bringUp() {
00053     bool expected;
00054     expected = sim->sendCommandExpecting("+CIICR", "OK", true,
      GPRS_SIM900_CIICR_TIMEOUT);
00055     return (unsigned char) (expected ? GprsSIM900::OK :
      GprsSIM900::ERROR);
00056 }
00057
00058 unsigned char GprsSIM900::obtainIp(unsigned char ip[4]) {
00059     const char* response;
00060     OperationResult result = GprsSIM900::ERROR;
00061     unsigned int receivedBytes = sim->sendCommand("+CIFSR", true,
      GPRS_SIM900_CIICR_TIMEOUT);
00062     if (receivedBytes > 0) {
00063         response = (const char*) sim->getLastResponse();
00064         if (parseIp(response, ip) == 4) {
00065             result = GprsSIM900::OK;
00066         }
00067     }
00068     return result;
00069 }
00070
00071 unsigned char GprsSIM900::status() {
00072     return status(-1);
00073 }
00074
00075 unsigned char GprsSIM900::status(char connection) {
00076     ConnectionState state = GprsSIM900::ERROR_WHEN_QUERING;
00077     int pos;
00078     sim->write("AT+CIPSTATUS");
00079     if (connection != (char) -1) {
00080         sim->write('=');
00081         sim->write('0' + connection);
00082     }
00083     sim->sendCommandExpecting("", "OK");
00084     pos = sim->waitUntilReceive("STATE",
      GPRS_SIM900_CIPSTATUS_TIMEOUT);
00085     if (pos >= 0) {
00086         if (sim->doesResponseContains("INITIAL")) {
00087             state = GprsSIM900::IP_INITIAL;
00088         } else if (sim->doesResponseContains("START")) {
00089             state = GprsSIM900::IP_START;
00090         } else if (sim->doesResponseContains("CONFIG")) {
00091             state = GprsSIM900::IP_CONFIG;
00092         } else if (sim->doesResponseContains("GPRSACT")) {
00093             state = GprsSIM900::IP_GPRSACT;
00094         } else if (sim->doesResponseContains("STATUS")) {
00095             state = GprsSIM900::IP_STATUS;
00096         } else if (sim->doesResponseContains("CONNECTING") ||
      sim->doesResponseContains("LISTENING")) {
00097             state = GprsSIM900::CONNECTING_OR_LISTENING;
00098         } else if (sim->doesResponseContains("CONNECT OK")) {
00099             state = GprsSIM900::CONNECT_OK;
00100         } else if (sim->doesResponseContains("CLOSING")) {
00101             state = GprsSIM900::CLOSING;
00102         } else if (sim->doesResponseContains("CLOSED")) {
```

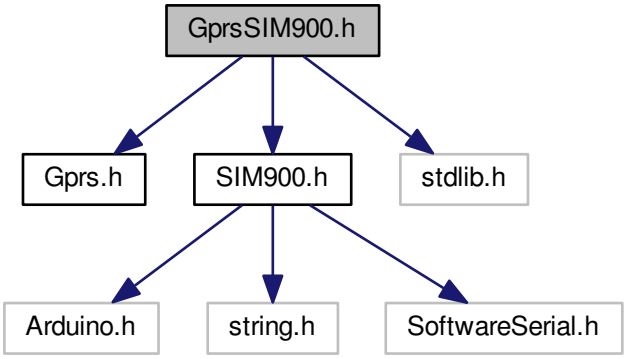
```
00103             state = GprsSIM900::CLOSED;
00104         } else if (sim->doesResponseContains("DEACT")) {
00105             state = GprsSIM900::PDP_DEACT;
00106         }
00107     }
00108     return state;
00109 }
00110
00111 unsigned char GprsSIM900::configureDns(const char *primary, const char *secondary)
      {
00112     bool expected;
00113     sim->write("AT+CDNSCFG=\"");
00114     sim->write(primary);
00115     sim->write("\",\"");
00116     sim->write(secondary);
00117     expected = sim->sendCommandExpecting("\"", "OK");
00118     return expected ? GprsSIM900::OK : GprsSIM900::ERROR;
00119 }
00120
00121 unsigned char GprsSIM900::open(char connection, const char *mode, const char *address,
      unsigned int port) {
00122     int pos;
00123     sim->write("AT+CIPSTART=");
00124     if (connection != (char) -1) {
00125         sim->write('0' + connection);
00126         sim->write(',');
00127     }
00128     sim->write('"');
00129     sim->write(mode);
00130     sim->write("\",\"");
00131     sim->write(address);
00132     sim->write("\",\"");
00133     sim->print(port, DEC);
00134     sim->sendCommand("\"");
00135     pos = sim->waitUntilReceive("CONNECT",
      GPRS_SIM900_CIPSTART_TIMEOUT);
00136     if (pos >= 0 && !sim->doesResponseContains("FAIL")) {
00137         return GprsSIM900::OK;
00138     }
00139     return (unsigned char) GprsSIM900::ERROR;
00140 }
00141
00142 unsigned int GprsSIM900::send(char connection, unsigned char *buf, unsigned int len) {
00143     bool ok;
00144     int pos = -1;
00145     unsigned int sent = 0;
00146     sim->write("AT+CIPSEND=");
00147     if (connection != (char) -1) {
00148         sim->write('0' + connection);
00149         sim->write(',');
00150     }
00151     sim->print(len, DEC);
00152     ok = sim->sendCommandExpecting("", ">");
00153     if (ok) {
00154         sent = (unsigned int) sim->write((const char *) buf, len);
00155         pos = sim->waitUntilReceive("SEND OK",
      GPRS_SIM900_SEND_TIMEOUT);
00156     }
00157     return pos >= 0 ? sent : 0;
00158 }
00159
00160 unsigned char GprsSIM900::close(char connection) {
00161     int pos;
00162     sim->write("AT+CIPCLOSE=1");
00163     if (connection != (char) -1) {
00164         sim->write(',');
00165         sim->write('0' + connection);
00166     }
00167     sim->sendCommand();
00168     pos = sim->waitUntilReceive("CLOSE OK",
      GPRS_SIM900_CIPSTART_TIMEOUT);
00169     if (pos >= 0) {
00170         return GprsSIM900::OK;
00171     }
00172     return GprsSIM900::ERROR;
00173 }
00174
00175 unsigned char GprsSIM900::close() {
00176     return close(-1);
00177 }
00178
00179 // TODO
00180 unsigned char GprsSIM900::resolve(const char *name, unsigned char ip[4]) {
00181     OperationResult result = GprsSIM900::ERROR;
00182     bool ok;
00183     int pos;
00184     const char* p;
```

```
00185     sim->write("AT+CDNSGIP=\"");
00186     sim->write(name);
00187     ok = sim->sendCommandExpecting("\"", "OK");
00188     if (ok) {
00189         pos = sim->waitUntilReceive("+CDNSGIP: 1",
      GPRS_SIM900_CDNSGIP_TIMEOUT);
00190         if (pos >= 0) {
00191             pos = sim->waitUntilReceive("\",\"",
      GPRS_SIM900_CDNSGIP_TIMEOUT);
00192             if (pos >= 0) {
00193                 p = (const char*) sim->getLastResponse();
00194                 if (parseIp(p + pos, ip) == 4) {
00195                     result = GprsSIM900::OK;
00196                 }
00197             }
00198         }
00199     }
00200     return result;
00201 }
00202
00203 unsigned char GprsSIM900::configureServer(unsigned char mode, unsigned int port)
       {
00204     mode &= 0x01;
00205     sim->write("AT+CIPSERVER=");
00206     sim->print(mode, DEC);
00207     sim->write(',');
00208     sim->print(mode, DEC);
00209     return sim->sendCommandExpecting("", "OK") ?
      GprsSIM900::OK : GprsSIM900::ERROR;
00210 }
00211
00212 unsigned char GprsSIM900::shutdown() {
00213     return sim->sendCommandExpecting("AT+CIPSHUT", "SHUT OK") ?
      GprsSIM900::OK : GprsSIM900::ERROR;
00214 }
00215
00216 unsigned char GprsSIM900::transmittingState(char connection, void *stateStruct
      ) {
00217     int pos;
00218     unsigned char *response;
00219     TransmittingState *state = (TransmittingState *) stateStruct;
00220     sim->write("AT+CIPACK=");
00221     if (connection != (char) -1) {
00222         sim->write(',');
00223         sim->write('0' + connection);
00224     }
00225     sim->sendCommand();
00226     pos = sim->waitUntilReceive("+CIPACK",
      GPRS_SIM900_CIPACK_TIMEOUT);
00227     if (pos >= 0) {
00228         response = sim->getLastResponse();
00229         // < +CIPACK: 2,2,0
00230         sscanf((const char *) (response + pos), "+CIPACK: %d,%d,%d", &state->
      txlen, &state->acklen, &state->nacklen);
00231         return GprsSIM900::OK;
00232     } else {
00233         state->txlen = state->acklen = state->nacklen = 0;
00234         return GprsSIM900::ERROR;
00235     }
00236 }
00237
00238 unsigned char GprsSIM900::parseIp(const char *buf, unsigned char ip[4]) {
00239     const char *p = buf;
00240     unsigned char j, i = 0, n = 0, part[4] = { 0 };
00241     while (*p != '\0' && n < 4) {
00242         if (*p >= '0' && *p <= '9') {
00243             part[i++ % 3] = *p;
00244         }
00245         if (*p == '.') {
00246             ip[n++] = (unsigned char) atoi((const char*) part);
00247             for (j = 0; j < 4; j++) {
00248                 part[j] = '\0';
00249             }
00250             i = 0;
00251         }
00252         p++;
00253     }
00254     if (i > 0) {
00255         ip[n++] = (unsigned char) atoi((const char*) part);
00256     }
00257     return n;
00258 }
00259
00260 #endif /* __ARDUINO_DRIVER_GSM_GPRS_SIM900_CPP__ */
00261
```

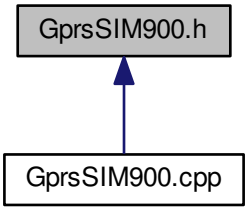
5.15 GprsSIM900.h File Reference

```
#include <Gprs.h>
#include <SIM900.h>
#include <stdlib.h>
```

Include dependency graph for GprsSIM900.h:



This graph shows which files directly or indirectly include this file:



Classes

- class GprsSIM900
- struct GprsSIM900::TransmittingState

Macros

- #define GPRS_SIM900_MAX_COMMAND_LENGHT 64
- #define GPRS_SIM900_CDNSGIP_TIMEOUT 5000UL
- #define GPRS_SIM900_CIICR_TIMEOUT 10000UL
- #define GPRS_SIM900_CIPSTART_TIMEOUT 5000UL
- #define GPRS_SIM900_SEND_TIMEOUT 10000UL

- #define GPRS_SIM900_CIPSTATUS_TIMEOUT 5000UL
- #define GPRS_SIM900_CIPACK_TIMEOUT 5000UL

5.15.1 Macro Definition Documentation

5.15.1.1 #define GPRS_SIM900_CDNSGIP_TIMEOUT 5000UL

Definition at line 26 of file GprsSIM900.h.

5.15.1.2 #define GPRS_SIM900_CIICR_TIMEOUT 10000UL

Definition at line 27 of file GprsSIM900.h.

5.15.1.3 #define GPRS_SIM900_CIPACK_TIMEOUT 5000UL

Definition at line 31 of file GprsSIM900.h.

5.15.1.4 #define GPRS_SIM900_CIPSTART_TIMEOUT 5000UL

Definition at line 28 of file GprsSIM900.h.

5.15.1.5 #define GPRS_SIM900_CIPSTATUS_TIMEOUT 5000UL

Definition at line 30 of file GprsSIM900.h.

5.15.1.6 #define GPRS_SIM900_MAX_COMMAND_LENGHT 64

Arduino - Gsm driver.

GprsSIM900.h

GPRS connection using SIM900.

Steps to connect to the GPRS net and issue a get request.

- call init
- call useMultiplexer
- call attach
- call bringUp
- call obtainIp
- configureDns("8.8.8.8", "8.8.4.4")

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 25 of file GprsSIM900.h.

5.15.1.7 #define GPRS_SIM900_SEND_TIMEOUT 10000UL

Definition at line 29 of file GprsSIM900.h.

5.16 GprsSIM900.h

```
00001
00022 #ifndef __ARDUINO_DRIVER_GSM_GPRS_SIM900_H__
00023 #define __ARDUINO_DRIVER_GSM_GPRS_SIM900_H__ 1
```

```
00024
00025 #define GPRS_SIM900_MAX_COMMAND_LENGHT  64
00026 #define GPRS_SIM900_CDNSGIP_TIMEOUT     5000UL
00027 #define GPRS_SIM900_CIICR_TIMEOUT       10000UL
00028 #define GPRS_SIM900_CIPSTART_TIMEOUT    5000UL
00029 #define GPRS_SIM900_SEND_TIMEOUT        10000UL
00030 #define GPRS_SIM900_CIPSTATUS_TIMEOUT   5000UL
00031 #define GPRS_SIM900_CIPACK_TIMEOUT      5000UL
00032
00033 #include <Gprs.h>
00034 #include <SIM900.h>
00035 #include <stdlib.h>
00036
00037 class GprsSIM900 : public Gprs {
00038
00042     SIM900 *sim;
00043
00047     bool multiplexed;
00048
00049 public:
00050
00051     enum OperationResult {
00052         OK = 0,
00053         ERROR = 1,
00054         COMMAND_TOO_LONG = 2
00055     };
00056
00057     enum DnsResolution {
00058         NOT_AUTHORIZATION = 0,
00059         INVALID_PARAMTER = 1,
00060         NETWORK_ERROR = 2,
00061         NO_SERVER = 3,
00062         TIMEOUT = 4,
00063         NO_CONFIGURATION = 5,
00064         NO_MEMORY = 6,
00065         SUCCESS = 0xff
00066     };
00067
00068     enum ConnectionState {
00069         IP_INITIAL = 0,
00070         IP_START = 1,
00071         IP_CONFIG = 2,
00072         IP_GPRSACT = 3,
00073         IP_STATUS = 4,
00074         CONNECTING_OR_LISTENING = 5,
00075         CONNECT_OK = 6,
00076         CLOSING = 7,
00077         CLOSED = 8,
00078         PDP_DEACT = 9,
00079         ERROR_WHEN_QUERING = 0xff
00080     };
00081
00082     struct TransmittingState {
00083         unsigned int txlen;
00084         unsigned int acklen;
00085         unsigned int nacklen;
00086         TransmittingState() :
00087                 txlen(0), acklen(0), nacklen(0) {
00088         }
00089     };
00090
00096     GprsSIM900(SIM900 *sim);
00097
00098     virtual ~GprsSIM900() {}
00099
00106     unsigned char begin(long bound);
00107
00120     unsigned char useMultiplexer(bool use);
00121
00136     unsigned char attach(const char *apn, const char *login, const char *password);
00137
00149     unsigned char bringUp();
00150
00166     unsigned char obtainIp(unsigned char ip[4]);
00167
00226     unsigned char status(char connection);
00227
00231     unsigned char status();
00232
00238     unsigned char configureDns(const char *primary, const char *secondary);
00239
00245     inline unsigned char open(const char *mode, const char *address, unsigned int port) {
00246         return open(-1, mode, address, port);
00247     }
00248
00269     unsigned char open(char connection, const char *mode, const char *address, unsigned int port);
00270
```

```
00282     inline unsigned int send(unsigned char *buf, unsigned int len) {
00283         return send(-1, buf, len);
00284     }
00285
00291     unsigned int send(char connection, unsigned char *buf, unsigned int len);
00292
00304     unsigned char close(char connection);
00305
00311     unsigned char close();
00312
00326     unsigned char resolve(const char *name, unsigned char ip[4]);
00327
00343     unsigned char configureServer(unsigned char mode, unsigned int port);
00344
00354     unsigned char shutdown();
00355
00359     inline unsigned char transmittingState(void *stateStruct) {
00360         return transmittingState(-1, stateStruct);
00361     }
00362
00384     unsigned char transmittingState(char connection, void *stateStruct);
00385
00392     unsigned char static parseIp(const char *buf, unsigned char ip[4]);
00393 };
00394
00395 #endif /* __ARDUINO_DRIVER_GSM_GPRS_SIM900_H__ */
```

5.17 GsmSIM900.cpp File Reference

5.18 GsmSIM900.cpp

```
00001
```

5.19 GsmSIM900.h File Reference

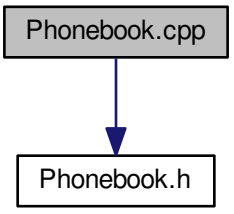
5.20 GsmSIM900.h

```
00001
```

5.21 Phonebook.cpp File Reference

```
#include "Phonebook.h"
```

Include dependency graph for Phonebook.cpp:



Macros

- #define __ARDUINO_DRIVER_GSM_CALL_CPP__ 1

5.21.1 Macro Definition Documentation

5.21.1.1 #define __ARDUINO_DRIVER_GSM_CALL_CPP__ 1

Arduino - Gsm driver.

Call.cpp

Interface to calls.

Author

Dalmir da Silva dalmirdasilva@gmail.com

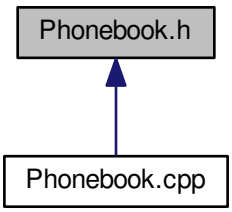
Definition at line 12 of file Phonebook.cpp.

5.22 Phonebook.cpp

```
00001
00011 #ifndef __ARDUINO_DRIVER_GSM_CALL_CPP__
00012 #define __ARDUINO_DRIVER_GSM_CALL_CPP__ 1
00013
00014 #include "Phonebook.h"
00015
00016 #endif /* __ARDUINO_DRIVER_GSM_CALL_CPP__ */
```

5.23 Phonebook.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class Phonebook

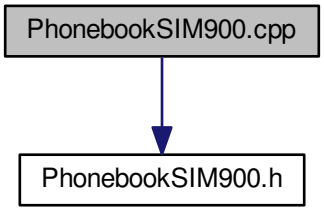
5.24 Phonebook.h

```
00001
00009 #ifndef __ARDUINO_DRIVER_GSM_PHONEBOOK_H__
00010 #define __ARDUINO_DRIVER_GSM_PHONEBOOK_H__ 1
00011
00012 class Phonebook {
00013
00014 public:
00015
00021     virtual unsigned char findEntries() = 0;
00022
00032     virtual unsigned char callNumber(unsigned char *number) = 0;
```

```
00033
00042     virtual unsigned char callFromPhonebook(unsigned char position) = 0;
00043
00053     virtual unsigned char callByPhonebookMatch(unsigned char *entry) = 0;
00054
00062     virtual unsigned char redial() = 0;
00063
00069     virtual unsigned char disconnect() = 0;
00070
00077     virtual unsigned char setAutomaticallyAnswering(unsigned char rings) = 0;
00078 };
00079
00080 #endif /* __ARDUINO_DRIVER_GSM_PHONEBOOK_H__ */
```

5.25 PhonebookSIM900.cpp File Reference

`#include "PhonebookSIM900.h"`
Include dependency graph for PhonebookSIM900.cpp:



Macros

- #define __ARDUINO_DRIVER_GSM_CALL_CPP__ 1

5.25.1 Macro Definition Documentation

5.25.1.1 #define __ARDUINO_DRIVER_GSM_CALL_CPP__ 1

Arduino - Gsm driver.

Call.cpp

Interface to calls.

Author

Dalmir da Silva dalmirdasilva@gmail.com

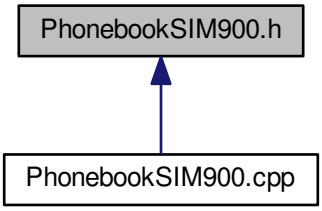
Definition at line 12 of file PhonebookSIM900.cpp.

5.26 PhonebookSIM900.cpp

```
00001
00011 #ifndef __ARDUINO_DRIVER_GSM_CALL_CPP__
00012 #define __ARDUINO_DRIVER_GSM_CALL_CPP__ 1
00013
00014 #include "PhonebookSIM900.h"
00015
00016 #endif /* __ARDUINO_DRIVER_GSM_CALL_CPP__ */
```

5.27 PhonebookSIM900.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class Call

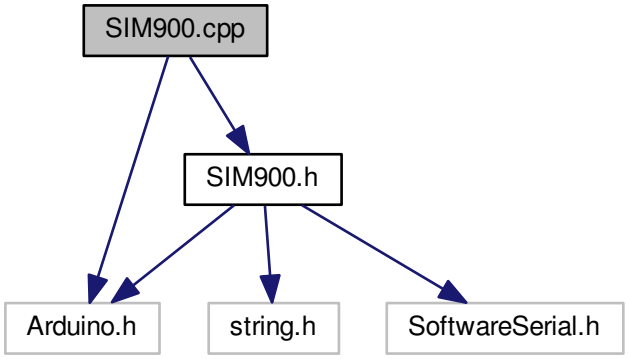
5.28 PhonebookSIM900.h

```
00001
00011 #ifndef __ARDUINO_DRIVER_GSM_CALL_H__
00012 #define __ARDUINO_DRIVER_GSM_CALL_H__ 1
00013
00014 class Call {
00015
00016 public:
00017
00025     virtual unsigned char answer() = 0;
00026
00036     virtual unsigned char callNumber(unsigned char *number) = 0;
00037
00046     virtual unsigned char callFromPhonebook(unsigned char position) = 0;
00047
00057     virtual unsigned char callByPhonebookMatch(unsigned char *entry) = 0;
00058
00066     virtual unsigned char redial() = 0;
00067
00073     virtual unsigned char disconnect() = 0;
00074
00081     virtual unsigned char setAutomaticallyAnswering(unsigned char rings) = 0;
00082 };
00083
00084 #endif /* __ARDUINO_DRIVER_GSM_CALL_H__ */
```

5.29 SIM900.cpp File Reference

```
#include <Arduino.h>
#include "SIM900.h"
```

Include dependency graph for SIM900.cpp:



Macros

- #define __ARDUINO_DRIVER_GSM_SIM900_CPP__ 1

5.29.1 Macro Definition Documentation

5.29.1.1 #define __ARDUINO_DRIVER_GSM_SIM900_CPP__ 1

Arduino - Gsm driver.

SIM900.cpp

SIM900 implementation of the SIM900 modem.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 12 of file SIM900.cpp.

5.30 SIM900.cpp

```
00001
00011 #ifndef __ARDUINO_DRIVER_GSM_SIM900_CPP__
00012 #define __ARDUINO_DRIVER_GSM_SIM900_CPP__ 1
00013
00014 #include <Arduino.h>
00015 #include "SIM900.h"
00016
00017 SIM900::SIM900(unsigned char receivePin, unsigned char transmitPin)
00018         : SIM900(receivePin, transmitPin, 0, 0) {
00019 }
00020
00021 SIM900::SIM900(unsigned char receivePin, unsigned char transmitPin, unsigned char resetPin,
      unsigned char powerPin)
00022         : SoftwareSerial(receivePin, transmitPin), rxBufferPos(0), echo(true), resetPin(resetPin), powerPin
      (powerPin), responseFullyRead(
00023         true) {
00024     rxBuffer[0] = '\0';
00025     pinMode(resetPin, OUTPUT);
00026     pinMode(powerPin, OUTPUT);
00027     softResetAndPowerEnabled = !(resetPin == 00 && powerPin == 0);
00028 }
```

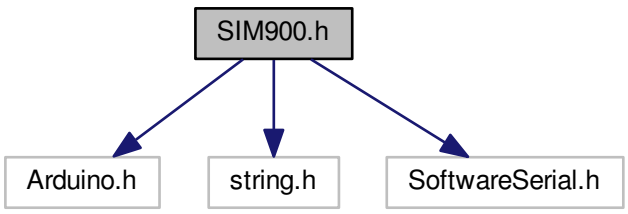
```
00029
00030 SIM900::~SIM900() {
00031 }
00032
00033 unsigned char SIM900::begin(long bound) {
00034     SoftwareSerial::begin(bound);
00035     if (sendCommandExpecting("AT", "OK")) {
00036         return 1;
00037     }
00038     softPower();
00039     return (unsigned char) (waitUntilReceive("Call Ready",
      SIM900_INITIALIZATION_TIMEOUT) >= 0);
00040 }
00041
00042 void SIM900::softReset() {
00043     if (softResetAndPowerEnabled) {
00044         digitalWrite(resetPin, HIGH);
00045         delay(100);
00046         digitalWrite(resetPin, LOW);
00047     }
00048 }
00049
00050 void SIM900::softPower() {
00051     if (softResetAndPowerEnabled) {
00052         digitalWrite(powerPin, HIGH);
00053         delay(1000);
00054         digitalWrite(powerPin, LOW);
00055     }
00056 }
00057
00058 bool SIM900::sendCommandExpecting(const char *command, const char *expectation,
       bool append, unsigned long timeout) {
00059     if (sendCommand(command, append, timeout) == 0) {
00060         return false;
00061     }
00062     return doesResponseContains(expectation);
00063 }
00064
00065 bool SIM900::doesResponseContains(const char *expectation) {
00066     return findInResponse(expectation) != NULL;
00067 }
00068
00069 unsigned int SIM900::sendCommand(const char *command, bool append, unsigned long timeout
      ) {
00070     rxBufferPos = 0;
00071     rxBuffer[0] = '\0';
00072     flush();
00073     if (append) {
00074         print("AT");
00075     }
00076     println(command);
00077     return readResponse(timeout);
00078 }
00079
00080 unsigned int SIM900::readResponse(unsigned long timeout, bool append) {
00081     int availableBytes;
00082     unsigned long pos, last, now, start;
00083     unsigned int read;
00084     last = start = millis();
00085     if (!append) {
00086         rxBufferPos = 0;
00087     }
00088     pos = rxBufferPos;
00089     responseFullyRead = true;
00090     while (available() <= 0 && (millis() - start) < timeout)
00091         ;
00092     start = millis();
00093     do {
00094         availableBytes = available();
00095         if (availableBytes > 0) {
00096             if (rxBufferPos + availableBytes >= SIM900_RX_BUFFER_SIZE) {
00097                 availableBytes = SIM900_RX_BUFFER_SIZE - (
      rxBufferPos + 1);
00098                 responseFullyRead = false;
00099             }
00100             if (availableBytes > 0) {
00101                 last = millis();
00102                 // we have the guaranty that is not going to be too big because it is constrained by the
       buffer size.
00103                 read = readBytes((char *) &rxBuffer[rxBufferPos], availableBytes);
00104                 rxBufferPos += read;
00105                 rxBuffer[rxBufferPos] = '\0';
00106             }
00107         }
00108         now = millis();
00109     } while ((availableBytes > 0 || (now - last) <
      SIM900_SERIAL_INTERBYTE_TIMEOUT) && (now - start) < timeout &&
```

```
    responseFullyRead);
00110     return rxBufferPos - pos;
00111 }
00112
00113 void SIM900::setEcho(bool echo) {
00114     this->echo = echo;
00115     char command[] = "E0";
00116     if (echo) {
00117         command[1] = '1';
00118     }
00119     sendCommand(command, true, 100);
00120 }
00121
00122 unsigned char SIM900::disconnect(DisconnectParamter param) {
00123     // TODO
00124     return 3;
00125 }
00126
00127 bool SIM900::wasResponseFullyRead() {
00128     return responseFullyRead;
00129 }
00130
00131 const char *SIM900::findInResponse(const char *str) {
00132     return strstr((const char *) &rxBuffer[0], str);
00133 }
00134
00135 int SIM900::waitUntilReceive(const char *str, unsigned int timeout) {
00136     const char *pos;
00137     while ((pos = findInResponse(str)) == NULL && readResponse(timeout,
    responseFullyRead) > 0)
00138         ;
00139     if (pos != NULL) {
00140         return (int) (pos - (const char *) &rxBuffer[0]);
00141     }
00142     return -1;
00143 }
00144
00145 void SIM900::discardBuffer() {
00146     rxBuffer[0] = '\0';
00147 }
00148
00149 #endif /* __ARDUINO_DRIVER_GSM_SIM900_CPP__ */
```

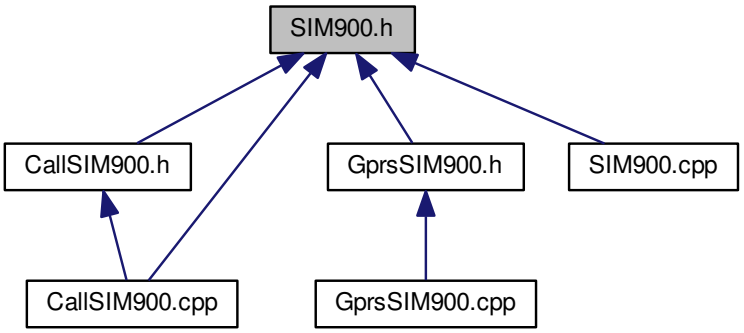
5.31 SIM900.h File Reference

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

Include dependency graph for SIM900.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SIM900

Macros

- #define SIM900_RX_BUFFER_SIZE 256
- #define SIM900_DEFAULT_COMMAND_RESPONSE_TIMEOUT 1000UL
- #define SIM900_INITIALIZATION_TIMEOUT 10000UL
- #define SIM900_SERIAL_INTERBYTE_TIMEOUT 50UL
- #define SIM900_SERIAL_FIRSTBYTE_TIMEOUT 5000UL

5.31.1 Macro Definition Documentation

5.31.1.1 #define SIM900_DEFAULT_COMMAND_RESPONSE_TIMEOUT 1000UL

Definition at line 20 of file SIM900.h.

5.31.1.2 #define SIM900_INITIALIZATION_TIMEOUT 10000UL

Definition at line 21 of file SIM900.h.

5.31.1.3 #define SIM900_RX_BUFFER_SIZE 256

Arduino - Gsm driver.

SIM900.h

SIM900 implementation of the SIM900 modem.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 18 of file SIM900.h.

5.31.1.4 #define SIM900_SERIAL_FIRSTBYTE_TIMEOUT 5000UL

Definition at line 23 of file SIM900.h.

5.31.1.5 #define SIM900_SERIAL_INTERBYTE_TIMEOUT 50UL

Definition at line 22 of file SIM900.h.

5.32 SIM900.h

```
00001
00011 #ifndef __ARDUINO_DRIVER_GSM_SIM900_H__
00012 #define __ARDUINO_DRIVER_GSM_SIM900_H__ 1
00013
00014 #include <Arduino.h>
00015 #include <string.h>
00016 #include <SoftwareSerial.h>
00017
00018 #define SIM900_RX_BUFFER_SIZE 256
00019
00020 #define SIM900_DEFAULT_COMMAND_RESPONSE_TIMEOUT 1000UL
00021 #define SIM900_INITIALIZATION_TIMEOUT           10000UL
00022 #define SIM900_SERIAL_INTERBYTE_TIMEOUT         50UL
00023 #define SIM900_SERIAL_FIRSTBYTE_TIMEOUT         5000UL
00024
00025
00026 class SIM900: public SoftwareSerial {
00027
00031     unsigned char rxBuffer[SIM900_RX_BUFFER_SIZE];
00032
00036     unsigned int rxBufferPos;
00037
00041     bool echo;
00042
00046     unsigned char resetPin;
00047
00051     unsigned char powerPin;
00052
00057     bool responseFullyRead;
00058
00062     bool softResetAndPowerEnabled;
00063
00064 public:
00065
00066     enum DisconnectParamter {
00067
00068         // Disconnect ALL calls on the channel the command is
00069         // requested. All active or waiting calls, CS data calls, GPRS call
00070         // of the channel will be disconnected
00071         ALL_CALLS_ON_CHANNEL = 0,
00072
00073         // Disconnect all calls on ALL connected channels. All active or
00074         // waiting calls, CSD calls, GPRS call will be disconnected.
00075         // (clean up of all calls of the ME)
00076         ALL_CALL_ON_ALL_CHANNELS = 1,
00077
00078         // Disconnect all connected CS data call only on the channel
00079         // the command is requested. (speech calls (active or waiting)
00080         // or GPRS calls are not disconnected)
00081         ALL_CS_ON_CHANNEL = 2,
00082
00083         // Disconnect all connected GPRS calls only on the channel
00084         // the command is requested (speech calls (active or waiting)
00085         // or CS data calls are not disconnected.
00086         ALL_GPRS_ON_CHANNEL = 3,
00087
00088         // Disconnect all CS calls (either speech or data) but does not
00089         // disconnect waiting call (either speech or data) on the
00090         // channel the command is requested.
00091         ALL_BUT_WAITING_ON_CHANNEL = 4,
00092
00093         // Disconnect waiting call (either speech or data) but does not
00094         // disconnect other active calls (either CS speech, CS data or
00095         // GPRS) on the channel the command is requested.
00096         // (rejection of incoming call)
00097         ALL_WAITING_ON_CHANNEL = 5
00098     };
00099
00105     SIM900(unsigned char receivePin, unsigned char transmitPin);
00106
00112     SIM900(unsigned char receivePin, unsigned char transmitPin, unsigned char resetPin, unsigned char
      powerPin);
00113
00117     virtual ~SIM900();
00118
00125     unsigned char begin(long bound);
00126
```

```
00130     void softReset();
00131
00135     void softPower();
00136
00142     unsigned char *getLastResponse() {
00143         return &rxBuffer[0];
00144     }
00145
00154     bool sendCommandExpecting(const char *command, const char *expectation, bool append
      , unsigned long timeout);
00155
00164     inline bool sendCommandExpecting(const char *command, const char *expectation, bool
       append) {
00165         return sendCommandExpecting(command, expectation, append,
      SIM900_DEFAULT_COMMAND_RESPONSE_TIMEOUT);
00166     }
00167
00176     inline bool sendCommandExpecting(const char *command, const char *expectation,
      unsigned long timeout) {
00177         return sendCommandExpecting(command, expectation, false, timeout);
00178     }
00179
00187     inline bool sendCommandExpecting(const char *command, const char *expectation) {
00188         return sendCommandExpecting(command, expectation, (bool) false);
00189     }
00190
00197     bool doesResponseContains(const char *expectation);
00198
00206     unsigned int sendCommand(const char *command, bool append, unsigned long timeout);
00207
00215     inline unsigned int sendCommand(const char *command, bool append) {
00216         return sendCommand(command, append, 1000);
00217     }
00218
00226     inline unsigned int sendCommand(const char *command, unsigned long timeout) {
00227         return sendCommand(command, (bool) false, timeout);
00228     }
00229
00236     inline unsigned int sendCommand(const char *command) {
00237         return sendCommand(command, (bool) false);
00238     }
00239
00243     inline unsigned int sendCommand() {
00244         return sendCommand("");
00245     }
00246
00253     inline unsigned int readResponse(unsigned long timeout) {
00254         return readResponse(timeout, false);
00255     }
00256
00264     unsigned int readResponse(unsigned long timeout, bool append);
00265
00271     void setEcho(bool echo);
00272
00273     unsigned char disconnect(DisconnectParamter param);
00274
00285     bool wasResponseFullyRead();
00286
00293     const char *findInResponse(const char *str);
00294
00302     int waitUntilReceive(const char *str, unsigned int timeout);
00303
00307     void discardBuffer();
00308
00309     /*
00310      void getProductIdentificationInformation();
00311
00312      void setMonitorSpeakerLoudness();
00313
00314      void setMonitorSpeakerMode();
00315      */
00316 };
00317
00318 #endif /* __ARDUINO_DRIVER_GSM_SIM900_H__ */
```

5.33 Sms.cpp File Reference

5.34 Sms.cpp

```
00001
```

5.35 Sms.h File Reference

Classes

- class Sms

5.36 Sms.h

```
00001
00011 #ifndef __ARDUINO_DRIVER_GSM_SMS_H__
00012 #define __ARDUINO_DRIVER_GSM_SMS_H__ 1
00013
00014 class Sms {
00015
00016 public:
00017
00026     virtual unsigned char remove(unsigned char index, unsigned char flags) = 0;
00027
00034     virtual unsigned char format(bool format) = 0;
00035
00043     virtual unsigned char bringUp() = 0;
00044
00054     virtual unsigned char obtainIp(unsigned char *buf) = 0;
00055
00061     virtual unsigned char status() = 0;
00062
00068     virtual unsigned char configureDns(const char *primary, const char *secondary) = 0;
00069
00075     virtual unsigned char open(unsigned char mode, unsigned char *address, unsigned char port) = 0;
00076
00082     virtual unsigned char open(char connection, unsigned char mode, unsigned char *address, unsigned
      char port) = 0;
00083
00089     virtual unsigned char close(char connection) = 0;
00090
00096     virtual unsigned char resolve(unsigned char *name, unsigned char *buf, unsigned int len) = 0;
00097
00103     virtual unsigned char send(unsigned char *buf) = 0;
00104
00110     virtual unsigned char send(char connection, unsigned char *buf, unsigned int len) = 0;
00111
00117     virtual unsigned char setUpServer(unsigned char mode, unsigned int port) = 0;
00118
00124     virtual unsigned char shutdown() = 0;
00125 };
00126
00127 #endif /* __ARDUINO_DRIVER_GSM_SMS_H__ */
```

Index