

SMS @Server Manual

SMS @server is a robust server that provides application code with a simple and versatile interface to GSM/GPRS/HSCSD and UMTS modems. It is a product build upon the SMSlib v1.0 library.

So, SMS @server allows any program (written in any language capable of programming TCP/IP sockets) to interface with AT modems and to interact with'em in a very simple manner.

In the following paragraphs, the list of all supported commands with their meaning and syntax will be shown.

Supported commands

NOP	Return the SMSC status word
SEND	Sends a message. Format: 'SEND <ID> <addr_list> <message>'
SENDR	Sends a message with status report. Format: 'SENDR <ID> <addr_list> <message>'
FLASH	Sends a flash message. Format: 'FLASH <ID> <addr_list> <message>'
FLASHR	Sends a flash message with status report. Format: 'FLASHR <ID> <addr_list> <message>'
SCA	Gets/sets current SCA. Format: 'SCA [<ID> [<dest_addr>]]'
QUIT	Quits the current session.
SHUTDOWN	Quits the current session and shuts down the server.
ECHO	Returns/sets the current local echo settings. Format: ECHO [<ID>] [ON OFF]
RESET	Reset the SMSC.
CONNINFO	Retrieve the connection info (SMSC specific).
RAW	Send raw data to the SMSC (applicable only for certain SMSC's). Format: 'RAW "<raw_command>''
VERSION	Shows the server version (currently, "SMS Server v.1.0").
HELP	Shows the help.

<ID> is an integer value in [6,65536] that can be used as a transaction ID. The following commands return a single line response:

NOP

SEND
SENDER
FLASH
FLASHR
SCA
QUIT
SHUTDOWN
ECHO
RESET
VERSION

Description and examples

In this section, we'll present each of the commands in a thorough manner.

Syntax

In this paragraph, the following syntax will be used. Any parameter name is enclosed in angle bracket to mean its value. For example, to mean the value of the parm named 'stat', the following syntax will be used:

<stat>

If a parm is enclosed in square brackets ('[' and ']'), it is optional.
E.g.:

[<stat>]

means that the 'stat' parm value is optional and can be omitted.

String values are typed as is, i.e., not enclosed in any couple of quotes or brackets (e.g., ECHO for the echo command).

Any set of string values enclosed in curly braces and separated by a pipe character (|) describes a set of alternatives. E.g., the string:

{ON|OFF}

or, equivalently, the string:

{ ON | OFF }

(e.g., the blanks are not significant, unless differently specified for any specific case) means that either the string ON or the string OFF are allowed (see ECHO command for an example).

The following common parm will be used throughout this section:

'ID' (<ID> means its value) ==> client transaction ID (sent to the server)

'SID' (<SID> means its value) ==> server transaction ID (sent to the client)

'MID' (<MID> means its value) ==> message ID sent back by the server to the client when an SMS is sent to the network successfully (actually, the TP-MR of that SMS)

'TID' (<TID> means its value) ==> Transaction ID sent by the server to the client when one or more SMS message were sent to the network successfully. Such 'TID' is dependent upon the specific SMSC or pseudo-SMSC type connected with. When connected with a GSM modem, <TID> is the list of TP-MR values (<MID>) related to the messages sent successfully, slash ('/') separated.

There is no connection between the various <ID> and <SID> values, nor the <SID> values are ensured to be contiguous. Both <ID> and <SID> can vary in [1, 65536].

Wherever <ID> is omitted in a command (since optional), in the response an <ID> = 0 will be used.

Any command lines and any response lines are terminated with a CRLF pair. This chars are not shown in the paragraph, but are always supposed to be present.

The NOP command

The NOP command (alone or followed by <ID>) returns the status word of the current SMSC. The status word is a 32-bits word returned by any SMSC supported by SMSlib. At present the only supported SMSC is the AT pseudo-SMSC.

Status word has the following format:

AABBCCDD

where AA, BB, CC and DD are couples of hexadecimal digits, which meaning depends on the type of SMSC. For AT pseudo-SMSC:

AA	==>	always 00
BB	==>	battery level (0-64 or FF)
CC	==>	signal quality (0-1F or FF)
DD	==>	BER (0-62 or FF)

Any FF value means unknown/not retrievable.

SYNOPSIS:

NOP [<ID>]

RESULT:

OK <ID> (followed by)
STATUS [<SID>] <status_word>

The SEND/SENDR/FLASH/FLASHR commands

All these commands request the SMSC to send a third party an SMS. The main differences are the following:

SEND sends a normal SMS
SENDR sends a normal SMS and requires a status report

FLASH sends a flash SMS
FLASHR sends a flash SMS and requires a status report

SYNOPSIS:

```
SEND <ID> <addr_list> <message>  
SENDR <ID> <addr_list> <message>  
FLASH <ID> <addr_list> <message>  
FLASHR <ID> <addr_list> <message>
```

where

<addr_list> is a slash-separated list of destination addresses (without intervening blanks), e.g.:
+391234567890/1098765432/+354536271890

<message> is the message in text format (UCS2 not directly supported), that MAY NOT contain CRLF pairs.

RESULT:

```
OK <ID> <TID>                (on success)  
ERROR <ID>                   <error_code> <error_description> (on error)  
KO <ID> <TID>                <error_code> <error_description> (on error)
```

NOTES:

The server accepts messages encoded in Western Win alphabet and automatically translates it into GSM alphabet. There is no current support for UCS2 charset.

The KO result is sent when, while sending the same message to a list of recipients, the operation is successful only for a subset of the recipients. In this case, for a GSM modem pseudo-SMSC, the returned <TID> is the list of the IDs of the successfully sent messages, in the same order as the passed addresses in <addr_list>. For GSM modems, the addresses in <addr_list> are processed by the server in the passed order. So, for example, if <addr_list> contains 5 addresses, A/B/C/D/E, the specified message is sent to the network in the given order (first A, last E). If an error occurs when C is being handled (that is, A and B were handled successfully), the client is sent a KO response, in which <TID> is the list of the message IDs respective to the successfully handled addresses, that is A and B.

The SCA command

This command sets the SCA or gets the currently set SCA.

SYNOPSIS:

```
1) SCA [<ID>]                (get current SCA)  
2) SCA <ID> <new_sca>       (set new SCA)
```

RESULT:

```
1) OK [<ID>]                 (followed by)  
   SCA <ID> <current_sca>  
  
2) OK <ID>                   (on success)
```

ERROR <ID> <error_code> <error_description> (on error)

The QUIT command

This command releases the current connection with the server, leaving the server up and running.

SYNOPSIS:

QUIT

RESULT:

OK

NOTES:

After the 'OK' response, the connection is released by the server.

The SHUTDOWN command

This command shuts down the server.

SYNOPSIS:

SHUTDOWN

RESULT:

OK

NOTES:

After the 'OK' response, the connection is released by the server. Then, the server starts performing the shutdown process.

The ECHO command

This command sets or gets the current ECHO settings. ECHO can either be ON or OFF: in the first case, the server echoes back any character sent by the client; in the second case, no chars are echoed back. Default: OFF.

SYNOPSIS:

- 1) ECHO [<ID>] (get current ECHO settings)
- 2) ECHO {ON|OFF} (set new ECHO settings)

RESULT:

- 1) ECHO is ON (or)
ECHO is OFF
- 2) OK

The RESET command

This command asks the server to perform a request.

SYNOPSIS:

RESET [<ID>]

RESULT:

OK

NOTES:

The 'OK' response is sent to the client AFTER the reset process has completed successfully. If the process does not complete successfully, an unsolicited ERROR response (see below) is sent and the server closes the connection. After that, the server shuts down and terminates.

The VERSION command

This command simply returns the current server version.

SYNOPSIS:

VERSION

RESULT:

SMS Server v.1.0

The HELP command

This command asks the server to send the list of supported commands.

SYNOPSIS:

HELP

RESULT:

The HELP response is more or less the same list as shown at the beginning of the "Supported commands" paragraph.

NOTES:

The response is not on a single line

The CONNINFO command

This command asks the server some connection and modem info

SYNOPSIS:

CONNINFO

RESULT:

The response varies according to the SMSC.

NOTES:

The response is not on a single line

The RAW command

This command sends raw data to the SMSC. It's not supported by all SMSC's. For AT modems, <command> can be any AT command not preceded by AT and not followed by CR, es., "I" for "ATI\r".

SYNOPSIS:

RAW <command>

RESULT:

The response is the raw response returned by the command.

NOTES:

The response can either be on a single line or on multiple lines.

Unsolicited messages

Besides the responses the server sent back to the client after a request (command) issued by the client, the server can send any info to the client of its own, that is, without having explicitly been requested. These info (messages) are called "unsolicited messages". These messages can be sent in any moment, without prior knowledge of this being happened for the client. No messages can/have to be sent to the server by the client in response to an unsolicited message. The following are the only unsolicited messages that are supported at present:

**RECEIVE
REPORT
CALL
ERROR
QUIT
OK**

The RECEIVE unsolicited message

This message is sent by the server when an SMS is received.

SYNOPSIS:

```
RECEIVE <SID> [<cat_cnt>/<cat_total>/<cat_ref>] <oa> <scts> <payload>
```

where:

<cat_cnt> is the index of the message of a set of concatenated messages
<cat_total> is the count concatenated messages of a set
<cat_ref> is the concatenated message index
<oa> is the originating address (typically, sender cell number)
<scts> is the service centre timestamp, with format: MM/DD/YYYY hh:mm:ss
<payload> is the sent message

NOTES:

The server sends messages (<payload>) encoded in Western WIN alphabet: it automatically translates it from the GSM alphabet. There is no current support for UCS2 charset.

<cat_cnt>, <cat_total> and <cat_ref> contain information about concatenation of messages.

They are sent to the client if and only if the "-m" parameter is specified when launching the executable (see. § "Command line arguments"). They are to be interpreted the following way:

- "cat_total": is the total number of SMS forming the concatenated message. For example, if the concatenated message is composed of 3 SMS, <cat_total> = 3;
- "cat_cnt": is the index of the current SMS with respect to the set of concatenated messages. For example, if the concatenated message is composed of 3 SMS, the first one will have <cat_cnt> = 1, the second one, <cat_cnt> = 2, etc.
- "cat_ref" is an integral number specifying the concatenated message index. The first concatenated message have <cat_ref> = 1, the second one <cat_ref> = 2, and so on. All the SMS forming the same concatenated message have the same <cat_ref> value.

A "normal" SMS, that is a non concatenated message, is formed of a single SMS for which:

- <cat_cnt> = 1
- <cat_total> = 1
- <cat_ref> = 0

The REPORT unsolicited message

This message is sent by the server when a status report (SMS-STATUS-REPORT) is received.

SYNOPSIS:

```
REPORT <SID> <ra> <scts> <MID> <status> [<payload>]
```

where:

<ra> is the original recipient address

- <scts> is the service centre timestamp, with format:
MM/DD/YYYY hh:mm:ss
- <status> is an integer indicating the status of the message which message ID (TP-MR) is <MID>. 0 stands for 'received', any other value indicates any error occurred with that SMS.
- <payload> is the accompanying message. It is optional since not all modems/networks support this feature. If no payload is present, <payload> is simply an empty (e.g., 0 length) string.

NOTES:

The server sends messages (<payload>) encoded in Western WIN alphabet: it automatically translates it from the GSM alphabet. There is no current support for UCS2 charset.

When connected with a GSM modem (AT pseudo-SMSC), the <MID> value sent with a REPORT message is just the same value as the corresponding one sent by the server in the <TID> list, as a response to a successful SEND(R)/FLASH(R) command.

The CALL unsolicited message

This message is sent by the server to inform the client a new call is incoming. At present, a call is terminated as soon as possible, so when this indication is sent by the server, probably the modem was already hung up.

SYNOPSIS:

CALL <SID> <caller_number>

NOTE:

<caller_number> can even be "(null)", meaning the caller is unknown.

The QUIT unsolicited message

This message is sent by the server to inform the client it is going to shutting down. Typically, this message is sent when an unrecoverable internal error occurs, after an ERROR unsolicited message.

SYNOPSIS:

QUIT

The OK unsolicited message

This message is sent just once per session, after a clients requests the connection, to inform the client that the connection went ok.

SYNOPSIS:

OK

The ERROR unsolicited message

This message is sent by the server when an internal error (e.g., a timeout expires or a consistency - e.g. protocol - error) occurs.

SYNOPSIS:

```
ERROR <SID> <error_code> <error_description>
```

After the unsolicited ERROR message, the server always releases the connection with the client. When the error is unrecoverable (i.e., the server cannot recover by simply reconnecting to the modem) the server also sends a QUIT unsolicited messages, after ERROR. Then, the connection is released and the server begins shutting down.

This unsolicited message is different from the ERROR response to client commands, in that the "solicited" response typically does not terminate the connection, unless the error is an unrecoverable one.

Program arguments

When the SMS @server program is launched, it first waits for some commands telling it how to behave and where to find needed information. Moreover, the server produces a log file. Typically, the server reads the commands from the "server.cfg" file, if it can find this file in the same directory in which the executable resides. If such file cannot be found, the server tries reading the commands from stdin. The "server.cfg" file name and path can be completely overwritten by means of the

```
-f <startup_file>
```

option passed as an argument to the program while launching it.
<startup_file> is the complete pathname of the new startup file.

The log file is, by default, a file named "server.log" in the same directory where the executable is stored in. Also the log file can be changed by using the

```
-o <log_file>
```

option passed as an argument to the program while launching it.
<log_file> is the complete pathname of the new log file.

By default, the program console is hidden as soon as the "server.cfg" file (or whatever its name be) is read, unless the program is passed the "-c" option. In this case, the console is kept visible.

The commands the server supports are the following:

```
CONF  
LISTEN  
DETACH  
VERSION  
HELP  
ABORT
```

The CONF command

This command specifies where the conf files are located. This command was added mainly because this server was written such to be most portable across various platforms/OSes. Since the rules defining the "position" of the conf files in a filesystem tree are different for different OSes, this command allows us to specify the conf file pathnames in a simple and portable way.

SYNOPSIS:

```
CONF <conf_file>
```

RESULT:

```
OK                                (on success)
ERROR: <error message>           (on error)
WARNING: <warning message>       (on non-heavy error)
```

NOTES:

The <conf_file> parm is passed "as is" to the OS. That is, no escape sequences are allowed but the one allowed by the OS itself.

The <conf_file> string is not parsed. For instance, no double backslash are to be specified under WinXX as a pathname separator. The following is a valid CONF line:

```
CONF C:\server\modem.cfg
```

The following one is not valid:

```
CONF C:\\server\\modem.cfg
```

The <conf_file> path must specify the SMSlib modem configuration file (see "configuration.txt" for details). In the original distribution, this file is named "modem.cfg" and is located in the same directory of the executable.

The LISTEN command

The server was written to "live" on a local station, that is, it can be used only on the same station to which the modem is physically connected. Note that this does not mean at all the server cannot be used on a network: in fact, the commands sent to the server can come from any ****authorized**** station/PC capable of connecting to the server with a TCP/IP connection.

Moreover, it accepts only a single connection at a time, taking all the problems related to concurrent use of a physical device off. This allow to build up a scalable system in an easy manner, by combining one or more server occurrences (running on one or more machines) thru' a "higher level" application that concurrently communicates with all the server processes.

The server accepts a single connection from a specific network interface (IP address): this constitutes a quite strict "access control" rule.

The LISTEN command specifies the local port the server has to listen at, the local network interface (IPv4 address) and the only remote interface (IPv4 address) that is allowed to connect to the server.

SYNOPSIS:

LISTEN <local_port> <local_address> <remote_address>

RESULT:

OK (on success)
Any error or warning message (on error)

NOTES:

If any non authorized client requests connection, the request is simply ignored. The same happens if two or more clients try connecting: even when all are authorized, just the first client can succeed.

The DETACH command

This command detaches the program from running in foreground and puts its execution in background. After this command, the server is able to accept incoming client connections.

NOTE: at present, the server does not actually detach, allowing any output to be viewed.

SYNOPSIS:

DETACH

RESULT:

Nothing, since the console is no more visible (only verbose output at the moment, since the console is not released).

The VERSION command

This command prints out the CONF version (not the server one).

SYNOPSIS:

VERSION

RESULT:

Verbose output.

The HELP command

This command prints on the console a list of available commands.

SYNOPSIS:

HELP

RESULT:

Verbose, multi-line output.

The ABORT command

This command aborts the sequence of commands given so far and terminates the program execution.

SYNOPSIS:

ABORT

RESULT:

Verbose output

The commands order

In order to properly set up the server for the connection, only some commands are to be given and in the proper order, that is:

- 1) CONF
- 2) LISTEN
- 3) DETACH

If the above sequence is not honoured, the "wrong" command fails. For example, if we give the DETACH command before LISTEN, the DETACH command fails. If we give the same command two or more times, the command itself fails from the second time on.