

The Handbook for SMSC-Gateway

Send and receive short messages via different connections to and from SMSCs

Dirk Gouders

Copyright © 2002, 2003 Dirk Gouders

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided also that the sections entitled “Copying” and “GNU General Public License” are included exactly as in the original, and provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the Free Software Foundation.

Table of Contents

1	Introduction	1
1.1	Why SMSC-Gateway?	1
1.2	Architecture of SMSC-Gateway	1
1.3	Special Terms	2
2	Installation	3
2.1	Getting the Sources	3
2.2	Building the Executables	3
2.3	Preparing the Config-Files	3
2.4	Testing the Connections	4
3	Server, Clients, Connections	7
3.1	smscgw	7
3.1.1	How smscgw works	7
3.1.2	smscgw.conf	7
3.1.2.1	The CALL_MANAGER section	8
3.1.2.2	The CLIENT_HANDLER section	8
3.1.2.3	The CONNECTION section	8
3.1.2.4	Examples	9
3.1.3	Starting smscgw	11
3.1.4	Stopping smscgw	11
3.2	smscgwc	12
3.2.1	Syntax of smscgwc	12
3.2.2	Example smscgwc session	12
3.3	Connections	13
3.3.1	m20_conn	13
3.3.1.1	Syntax of m20_conn	13
3.3.1.2	m20_conn.conf	13
3.3.2	ta_conn	14
3.3.2.1	ta_conn.conf	15
3.3.3	tcp_conn	15
4	Acknowledgments	17
	Appendix A Hardware that has been used with SMSC-Gateway	19
	Appendix B Acronyms	21

1 Introduction

This is the handbook for SMSC-Gateway, a software package that can be used to send and receive short messages via various connections to SMSCs.

It tries to help you, if you want to use SMSC-Gateway or want to understand how SMSC-Gateway works and then decide if it might be useful to you.

This first chapter gives a short introduction to SMSC-Gateway, chapter two will describe how to get and install SMSC-Gateway and the third chapter will show how to configure SMSC-Gateway and try to help to understand what the configuration files are used for. In the appendix you will find a list of hardware that has already been used with SMSC-Gateway.

1.1 Why SMSC-Gateway?

SMSC-Gateway has been built as an experimental alarming system for firemen. We wanted to find out if it would be useful to use short messages to GSM phones instead of messages to pagers to call firemen in case of emergency. There were two main advantages expected by this system:

1. Nowadays, most of the firemen already have GSM phones and it was thought somewhat inconvenient for a fireman to also carry a pager device everywhere he goes.
2. Short messages to GSM devices enable a bidirectional communication, i.e. the firemen can be alarmed and reply with an answer if they are on their way or currently unavailable.

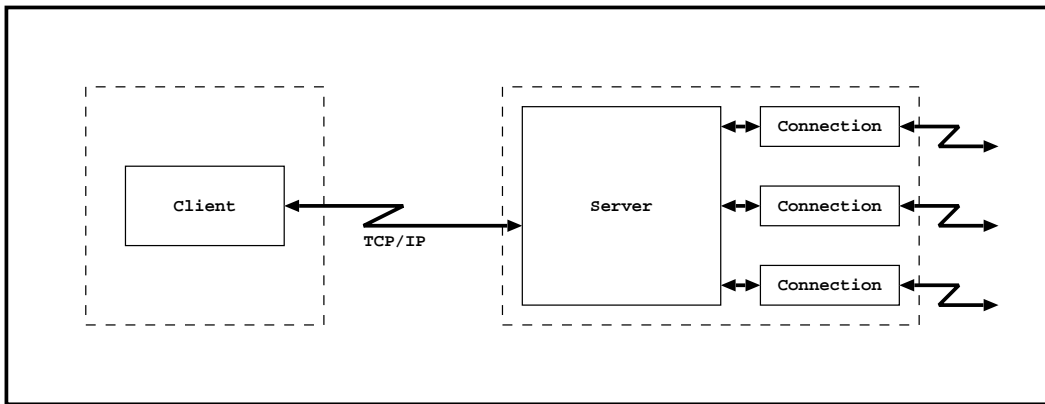
This point has been seen as a possibility to save resources, because there would be no need to always alarm more firemen than actually needed and send those firemen for whom was no need back home.

Because SMSC-Gateway's area of operation was in the field of rescue service it was build to be very reliable. For that reason different connection types have been used so that certain connections can be used as fallback connections should other ones fail for some reason.

1.2 Architecture of SMSC-Gateway

SMSC-Gateway was build in a modular way. There is a main server part, a client part that can be used to contact the server to send short messages and finally there are connections that actually hand over the messages to the SMSCs for delivery to the recipients. The different connections work with different hardware; currently available connection types are:

1. GSM modem attached to a serial line (AT commands)
2. Terminal adapter attached to a serial line (X.31 connection, EMI protocol)
3. TCP/IP connection (EMI protocol)



In the future it could be possible that connections are added that not only operate on short messages but also on other things like mail, fax or even synthesize voice mail.

SMSC-Gateway also comes with a rudimentary web-interface that can be used to monitor certain statistics (number of active connections, number of restarted connections, sent messages, received messages...) and to start, stop and restart the server. However, this interface was tailored for the needs of the firestation project and you maybe do not find it useful.

1.3 Special Terms

Throughout this manual I will use some terms that may be unfamiliar to you or that should be defined, to avoid misunderstandings. Here is a list of such terms:

incoming message

An incoming message is a message that arrives at one of SMSC-Gateway's connections.

outgoing message

An outgoing message is a message that was handed over to SMSC-Gateway by a Client and that SMSC-Gateway will send to the recipient via one of its connections.

send request

This term is used when I talk about the data that is handed over to `smscgwc` or a connection to make it transmit a message to one or more recipients. A send request consists of a list of recipient addresses and a message text.

2 Installation

2.1 Getting the Sources

The sources for SMSC-Gateway can be downloaded from <http://smscgw.ccamp.de/>. Once arrived there, go to the download section and get the latest tar-ball.

2.2 Building the Executables

Once you downloaded the sources, follow the next steps to build the binaries. You do not necessarily have to build the sources in the directory `‘/usr/src/’`. You can choose any other location.

1. Change to a directory under which you want to build the binaries, e.g.:

```
cd /usr/src/
```

2. Unpack and extract the files of the archive:

```
tar -zxf path_to_the_tar_ball
```

3. Change to the SMSC-Gateway source directory. If you downloaded the tar-ball `smscgw-1.1.11.tar.gz` then the command will be:

```
cd smscgw-1.1.11
```

4. Run the `configure` script to prepare the compilation process:

```
./configure
```

There are some options to `configure` that you might want to use:

`‘--prefix’`

This option can be used to explicitly specify a directory under which the command `make install` will install binaries, configuration files, etc. The default location is `‘/home/smscgw’`.

`‘--with-httpdir’`

This option can be used to install the HTML files that belong to the web interface to a different place location than the default `‘/home/smscgw/www/data’`.

`‘--with-cgidir’`

This option can be used to install the CGI scripts that belong to the web interface to a different location than the default `‘/home/smscgw/www/cgi-bin/’`.

For a description of the other standard options that come with `configure` run

```
./configure --help
```

5. Build and install everything¹ :

```
make  
make install
```

¹ Under FreeBSD you maybe need to use `gmake` instead of `make`.

2.3 Preparing the Config-Files

SMSC-Gateway comes with four configuration files. By default they will be installed in the directory `‘/home/smscgw/etc/’`. Here is an overview about which purposes they serve:

`‘smscgw.conf’`

This file is needed to configure the server process that handles clients that connect to `smscgw` to send messages and the various connections one wants to use to pass the client’s send requests to SMSCs. Briefly, this file is to tell `smscgw` about connections and routes it should use.

`‘m20_conn.conf’`

This file is needed to configure the GSM modem connection. Here you specify the serial port to which the modem is attached, and other information that is needed to work with the modem. If you do not use a modem connection you can remove or just ignore this file.

`‘ta_conn.conf’`

This file is needed to configure the connection via an ISDN terminal adapter (TA). Here you specify the serial port to which the TA is attached, and other information that is needed to work with the TA. If you do not use this kind of connection you can remove or just ignore this file.

`‘tcp_conn.conf’`

This file is needed for a TCP/IP connection to a SMSC. Here you specify the IP-address of the SMSC, the destination port, etc.

So, before you can start `smscgw`, you have to edit `‘smscgw.conf’` and one of the connection’s configuration files, at least. In `‘smscgw.conf’` you have to specify which connection(s) you want to use and, which connection to use for what kind of recipient address. See [The description of `smscgw.conf`], page 7, and the following pages for a detailed description of the configuration files for the various connections.

2.4 Testing the Connections

Before running SMSC-Gateway with its server process, clients and connections it is a good idea to check whether each of the connections works, i.e. if the hardware is OK and if the configuration files are correct. This can easily be done, since each connection can be run as a standalone program and in this case produces error output to the console. If you, for example, want to run SMSC-Gateway with a GSM phone attached to a serial interface, you have to edit the configuration file `‘/home/smscgw/etc/m20_conn.conf’` and after that can run `‘/home/smscgw/bin/m20_conn’` and see if it starts up correctly. Here is an example for a startup with an incorrect configuration file:

```
# /home/smscgw/bin/m20_conn
read_config_file: M20_DEVICE MUST be specified!
Error in configuration file /home/smscgw/etc/m20_conn.conf!
```

A startup with a correct configuration file looks like this:

```
# /home/smscgw/bin/m20_conn
Modem device opened.
init_m20: Checking for PIN...PIN is OK.
init_m20: Setting PDU mode...done.
init_m20: Entering SMSC number...done.
Modem ready:
```

At this point, you can try to send a message. Each connection accepts input similar to that of `smscgwc`:

1. One or several lines containing recipient addresses
2. One empty line
3. The message text
4. A final line containing only a dot (".").

For example, you can send yourself a message and this way see, if outgoing and incoming messages work. If you tested each of the connections and then start the server, there may still appear problems with the configuration file `‘/home/smscgw/etc/smscgw.conf’`, but problems with the connections are excluded.

3 Server, Clients, Connections

In this chapter, I will go into the details of each part of SMSC-Gateway: the server process, the client process and the various processes that manage connections to SMSCs.

3.1 smscgw

`smscgw` is the server process of SMSC-Gateway. It communicates with clients that want to send messages and with the connections one uses to connect to SMSCs.

`smscgw` produces verbose messages that are passed to the `syslog` daemon. Lots of the messages are for debugging purposes and are given the corresponding level of severity. So, if you don't like the huge amount of logged messages, you may want to configure your `syslogd` to ignore `smscgw`'s debug level messages.

3.1.1 How smscgw works

On startup, `smscgw` reads its configuration file and prepares an internal table of known connections and a routing table. After that, a *call manager* process and a *transceiver* process are created.

The *call manager* waits for clients to connect. For each connecting client it forks a new *client handler* process that hands over send requests to the *transceiver* and results from the *transceiver* back to the connected client.

After creation, the *transceiver* process first starts connections that are marked **permanent** and then waits for either send requests from clients or incoming data from connections.

If a send request from a client arrives, the *transceiver* determines what connection(s) to use to transmit the request. That decision is based on the routing table and on the priorities of the connections. If different connections have to be used for the recipients of a single send request, the send request is divided in sub-requests that contain recipients that can all be reached via a single connection. After that, the requests are passed to the connection(s). The *transceiver* process then waits for result messages from the connection(s) and passes them back to the client handler.

If an incoming message from a connection arrives, the *transceiver* saves that message in a daily file in the incoming directory. This directory defaults to `'/tmp'` and can be changed with the `INCOMING_DIR` directive in the configuration file `'/home/smscgw/etc/smscgw.conf'`.

Further, if for the connection that delivers the incoming message, an `INCOMING_SCRIPT` was specified, that script is executed and the incoming messages is passed to the script via its *standard input*.

3.1.2 smscgw.conf

In this section the configuration file of `'smscgw.conf'` will be described.

`'smscgw.conf'` consists of sections with various directives. Sections start with a section name and an opening brace `"{"` and end with a closing brace `"}"`.

Empty lines and text following a hash character ("`#`") are interpreted as comments and ignored.

The directives inside the sections are *keyword = value* pairs where the values can be one of three types:

- integer* An integer numeric value
- boolean* A boolean value may be one of the case insensitive words *0*, *1*, *no*, *yes*, *true* or *false*.
- string* A string is a sequence of characters enclosed in quotes, e.g. "This is a string".

There are three sections that may appear in '`smscgw.conf`' and they must be given exactly the section name as shown below:

```
CALL_MANAGER
CLIENT_HANDLER
CONNECTION
```

Among these sections the two first ones may appear only once and the latter is mandatory but may appear several times. The CONNECTION section must appear at least once, because it would not make sense to start `smscgw` without knowing about a connection that it could use to deliver messages handed over by a client.

3.1.2.1 The CALL_MANAGER section

In the CALL_MANAGER section configuration directives for the call manager can be specified. There are currently two directives that can be used inside this section:

PORT = integer

This directive can be used to tell the call manager on which port to listen for client connections. If unspecified, the default value of 1025 will be used.

MAX_CLIENTS = integer

This directive can be used to specify how many simultaneous client connections are allowed. If unspecified, the default value of 5 will be used.

3.1.2.2 The CLIENT_HANDLER section

In the client handler section only one directive can be used:

MAX_WAIT_RESULT_TIMEOUT = integer

This option tells a client handler that serves a client how many seconds at maximum it should wait for a result from the used connection(s) after the client submitted a send request. If that time elapses without the receipt of a complete result from the used connection(s) the connection to the waiting client will be closed, anyway.

If this directive is unspecified, the default value of 60 seconds will be used.

3.1.2.3 The CONNECTION section

As already stated, this section may be used several times – one for each connection you want to use to deliver messages to SMSCs. In a *CONNECTION* section the following directives can be used; some of them are mandatory, others are optional:

NAME = string

With this mandatory directive a symbolic name will be associated with the connection. That symbolic name will be used when the connection will be used to deliver messages and log messages are generated.

PROGRAM = string

This mandatory directive is used to tell `smscgw` which program actually implements the connection and what command line arguments it will be passed.

PRIORITY = integer

This mandatory directive gives the connection a priority and the value must be out of the range [0-254].

Priorities are used when `smscgw` is about to decide which connection to use to deliver send requests; higher priority connections are considered prior to lower priority ones.

ROUTE = string

This directive tells `smscgw` what recipients can be reached via a connection. The string is interpreted as a regular expression according to `regex(3)`. This directive may appear multiple times inside a single section.

Currently, this directive is optional, but if it left out completely, it means that the connection cannot deliver messages to anywhere.

This will be corrected in a future release.

INCOMING_DIR = string

This directive tells `smscgw` in which directory to save incoming messages. It defaults to `‘/tmp/’`. When incoming messages arrive at a connection, they will be saved in daily files in the directory specified by `INCOMING_DIR`.

INCOMING_SCRIPT = string

This directive can be used if incoming messages should be processed by a program or script. When an incoming message arrives, the program or script specified by *string* will be executed and fed the incoming message via *standard input*.

NOTE: Incoming scripts will become new processes and it is possible that multiple of them run in parallel. So, one has to take care for racing conditions when writing programs or scripts for incoming messages!

3.1.2.4 Examples

In the following I will present some examples of configuration files for `smscgw`:

1. At first an example where only one connection via TCP/IP will be used.

Note: In this example the program `tcp_conn` will expect its configuration parameters to be in its default configuration file, i.e. `‘/home/smscgw/etc/tcp_conn.conf’`.

```
#
# We accept default values and specify the only one mandatory section.■
#
CONNECTION {
    NAME = "tcp-connection"
    PROGRAM = "/home/smsgw/sbin/tcp_conn"
    PRIORITY = 99
    ROUTE = "[0-9]+"
}
```

- Now, an example where three GSM connections will be used simultaneously to deliver messages to different GSM providers.

Here, we have to use the command line option ‘-c’ for each connection to tell it where it finds its configuration file. Further, all connections get the same priority but different routes, so that each connection only serves recipients whose addresses have a prefix that belongs to the GSM network that that connection should send messages to.

```
#
# Use GSM connection for a fictitious GSM network
#
CONNECTION {
    NAME = "X1"
    PROGRAM = "/home/smscgw/sbin/m20_conn -c /home/smscgw/etc/x1.conf"
    PRIORITY = 99
    ROUTE = "0171[0-9] +"
    ROUTE = "0175[0-9] +"
}
#
# And another GSM device for another GSM network.
#
CONNECTION {
    NAME = "X2"
    PROGRAM = "/home/smscgw/sbin/m20_conn -c /home/smscgw/etc/x2.conf"
    PRIORITY = 99
    ROUTE = "0162[0-9] +"
    ROUTE = "0172[0-9] +"
}
#
# And a further GSM device for the last GSM network.
#
CONNECTION {
    NAME = "X3"
    PROGRAM = "/home/smscgw/sbin/m20_conn -c /home/smscgw/etc/x3.conf"
    PRIORITY = 99
    ROUTE = "0179[0-9] +"
}
}
```

3.1.3 Starting smscgw

Starting `smscgw` is simple – just type in the path to the binary, that's it. If you for some reason don't want `smscgw` to become a daemon, you can use the option ‘--debug’ to it, e.g:

```
/home/smscgw/sbin/smscgw --debug
```

3.1.4 Stopping smscgw

If you want to stop `smscgw`, you need to send its call manager a TERM signal. It then will take care that all processes that belong to the `smscgw` session will terminate. To send the

call manager a TERM signal you need to know its process ID which can be found in the file `/var/run/smscgw.pid`. So, here is an example on how to terminate a smscgw daemon:

```
kill -TERM `cat /var/run/smscgw.pid`
```

3.2 smscgwc

`smscgwc` is a client program that can be used to connect to a machine running the server process `smscgw` and hand it over a message it should send to one or a list of recipients.

Input to `smscgwc` is identical to that of the connections:

1. One or several lines containing recipient addresses
2. One empty line
3. The message text
4. A final line containing only a dot (".").

After a `smscgwc` handed over a send request to `smscgw` it waits for the result from the server. However, the server may close the connection with a timeout message and in this case the client should consider the delivery to have failed for some reason.

3.2.1 Syntax of smscgwc

```
smscgwc 'ip-address' 'port'
```

`smscgwc` must be given the IP-address of the server running `smscgw` and the port on which it is listening for client connections. After the connection to the server has been established, `smscgwc` displays the server's greeting message and then silently waits for input of a send request. After receipt of a send request, it hands the request over to the server, then waits for the server's acknowledge message, displays it and terminates.

3.2.2 Example smscgwc session

Here is the output of a sample `smscgwc`-session:

```
client: smscgw 192.168.0.2 1025
Welcome to smscgw
01234567890

An example for the handbook;-)
.
Thanks - please wait for result...
Request for 01234567890 successfully transmitted.

.

client:
```

3.3 Connections

In this section, I will talk about the connections `smscgw` uses to deliver messages to SMSCs. The connections are standalone programs that can also be used independently of `smscgw`. They all follow three simple rules:

1. Send requests are expected from standard input (`'stdin'`) and have to be of the following format:
 1. One or several lines containing recipient addresses
 2. One empty line
 3. The message text
 4. A final line containing only a dot (`"."`).

Example:

```
01621234567
01721234567
01711234567
```

```
This is the text that should be sent to the above recipients
.
```

2. Incoming messages are written to standard output (`'stdout'`). The format of incoming messages is identical to the format of send requests, they just start with one sender address instead of one or several recipient addresses.
3. Results and error messages are written to standard error (`'stderr'`).

3.3.1 m20_conn

`m20_conn` is a connection that can talk to GSM devices attached to a serial port of the computer.

`m20_conn` only works with devices that understand AT commands. It supports text mode and PDU mode operation and is known to work with devices listed in the supported hardware section.

Once, `m20_conn` has been started, it waits for send requests to deliver to the SMSC from standard input and checks for incoming messages from the GSM device in intervals specified by the configuration directive `'CHECK_INCOMING_INTERVAL'`. If new incoming messages are present, they are printed to standard output and deleted from the GSM device's memory.

3.3.1.1 Syntax of m20_conn

```
m20_conn { '-c' | '--config-file' } filename
```

The only one option, `m20_conn` understands is the one to tell it to read another configuration file, than the default `'/home/smscgw/etc/m20_conn.conf'`.

3.3.1.2 m20_conn.conf

This section describes the configuration file for `m20_conn`. It must not necessarily be called `'m20_conn.conf'`, but I will use this name to refer to **the configuration file for the program m20_conn**.

`'m20_conn.conf'` mainly consists of *keyword=value* directives. Empty lines and lines starting with a hash character `"#"` are comments and ignored. The following table lists the keywords, `m20_conn` understands; they must be written exactly as they appear here:

MAX_ACK_TIMEOUT

This directive specifies the maximum amount of seconds to wait for acknowledges to a send request from the GSM device.

CHECK_INCOMING_INTERVAL

This option can be used to specify an interval (in seconds) at which checks for new incoming messages are performed. The default is 10s.

M20_CHAR

In some future, this option will probably be used to specify the number of databits, parity and the number of stop bits. Currently it is ignored and the value of 8N1 (eight data bits, no parity and one stop bit) is used.

M20_DEVICE

This directive is used to specify the serial interface, the GSM device is connected to. On a FreeBSD machine, they are called `'/dev/cuaa0'`, `'/dev/cuaa1'`... and on a GNU/Linux system possible values are `'/dev/ttyS0'`, `'/dev/ttyS1'`...

M20_SPEED

This directive must be used to specify the baud-rate of the serial interface. Possible values are:

4800

9600

19200

38400

M20_PIN This directive can be used for setting the PIN of the GSM device should it be necessary. If you think this directive is insecure then you can leave it out and set the PIN with a terminal program. The AT command to do that is `"AT+CPIN=PIN"`.

SMSC_NUMBER

This directive can be used to change the internal SMSC setting of the connected GSM device. If you don't want to change the setting, then just leave this directive out or set it to `"0"`.

USE_PDU_MODE

Use this keyword if you use a GSM device that can only operate in PDU mode (like the S35, for example). A value of `'0'` turns PDU mode off, other values turn PDU mode on.

3.3.2 ta_conn

`ta_conn` is a connection that can talk to an ISDN terminal adapter which on the other side can talk to a SMSC via X.31. Up to know only one terminal adapter has been used and I am pretty sure that other products would cause problems.

3.3.2.1 ta_conn.conf

3.3.3 tcp_conn

`tcp_conn` is a connection that can directly talk to a SMSC via a TCP/IP connection using the EMI protocol.

Should you ask yourself what provider offers a publicly reachable IP-address for such a connection, I am pretty sure the answer is **none**. I used a TCP/IP connection via a ISDN line and needed a quite expensive large account at the GSM network provider.

4 Acknowledgments

I would like to thank the fire-brigade in Bocholt, Germany for the opportunity to build SMSC-Gateway and for giving me access to hardware and SMSC connections I would otherwise never get my hands on.

Further, many thanks to the following individuals:

Alexander Bergmann

Tested SMSC-Gateway with serveral different phones.

Harald Knapp

Wanted to use SMSC-Gateway with a WAVECOM GSM modem and did several tests with updated versions, until it worked.

Salvatore Mantaci

Tested SMSC-Gateway with a GSM modem from DIGICOM.

Roland Moritz

Tested SMSC-Gateway with a Siemens S35 phone.

Appendix A Hardware that has been used with SMSC-Gateway

The following table lists connections and hardware that has already been used with them, i.e. is known to work with SMSC-Gateway:

<code>m20_conn</code>	Siemens M20 GSM modem
<code>m20_conn</code>	GSM modem from DIGICOM (Tested by Salvatore Mantaci - thanks!)
<code>m20_conn</code>	WAVECOM FASTTRACK WMOD2 GSM modem (Tested by Harald Knapp - thanks!)
<code>m20_conn</code>	Siemens C35 GSM phone (Tested by Alex Bergmann - thanks!)
<code>m20_conn</code>	Siemens C55 GSM phone (Tested by Alex Bergmann - thanks!)
<code>m20_conn</code>	Siemens S25 GSM phone
<code>m20_conn</code>	Siemens S35 GSM phone (Tested by Roland Moriz - thanks!)
<code>m20_conn</code>	Siemens S55 GSM phone - also via Bluetooth interface (Tested by Alex Bergmann - thanks!)
<code>ta_conn</code>	TA MICRO from TDT (http://www.tdt.de/products/ota/mainotae.htm)
<code>tcp_conn</code>	Any ethernet NIC that works with your operating system can be used.

Appendix B Acronyms

EMI	External Machine Interface
PDU	Protocol Data Unit
SM	Short Message
SMS	Short Message Service
SMSC	Short Message Service Center

