Gammu SMSD Daemon Manual

Release 1.42.0

Michal Čihař <michal@cihar.com>

Jun 20, 2023

CONTENTS

CHAPTER

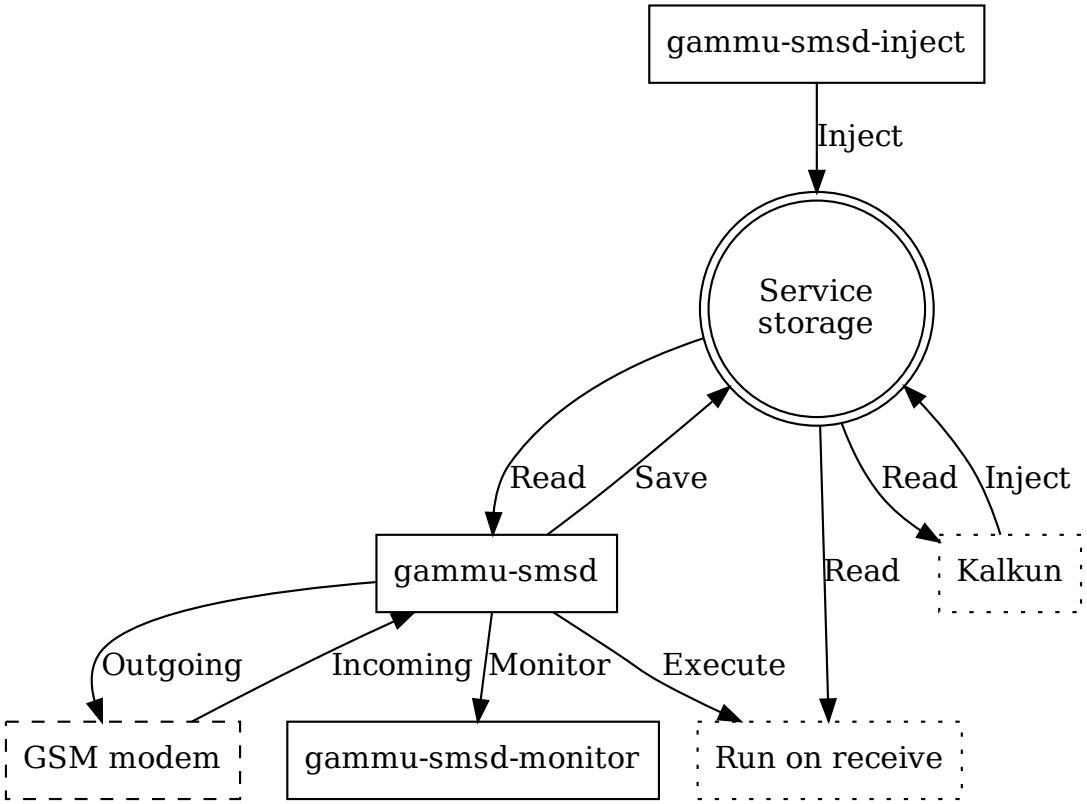
ONE

OVERVIEW

Gammu SMS Daemon is a program that periodically scans GSM modem for received messages, stores them in defined storage and also sends messages enqueued in this storage.

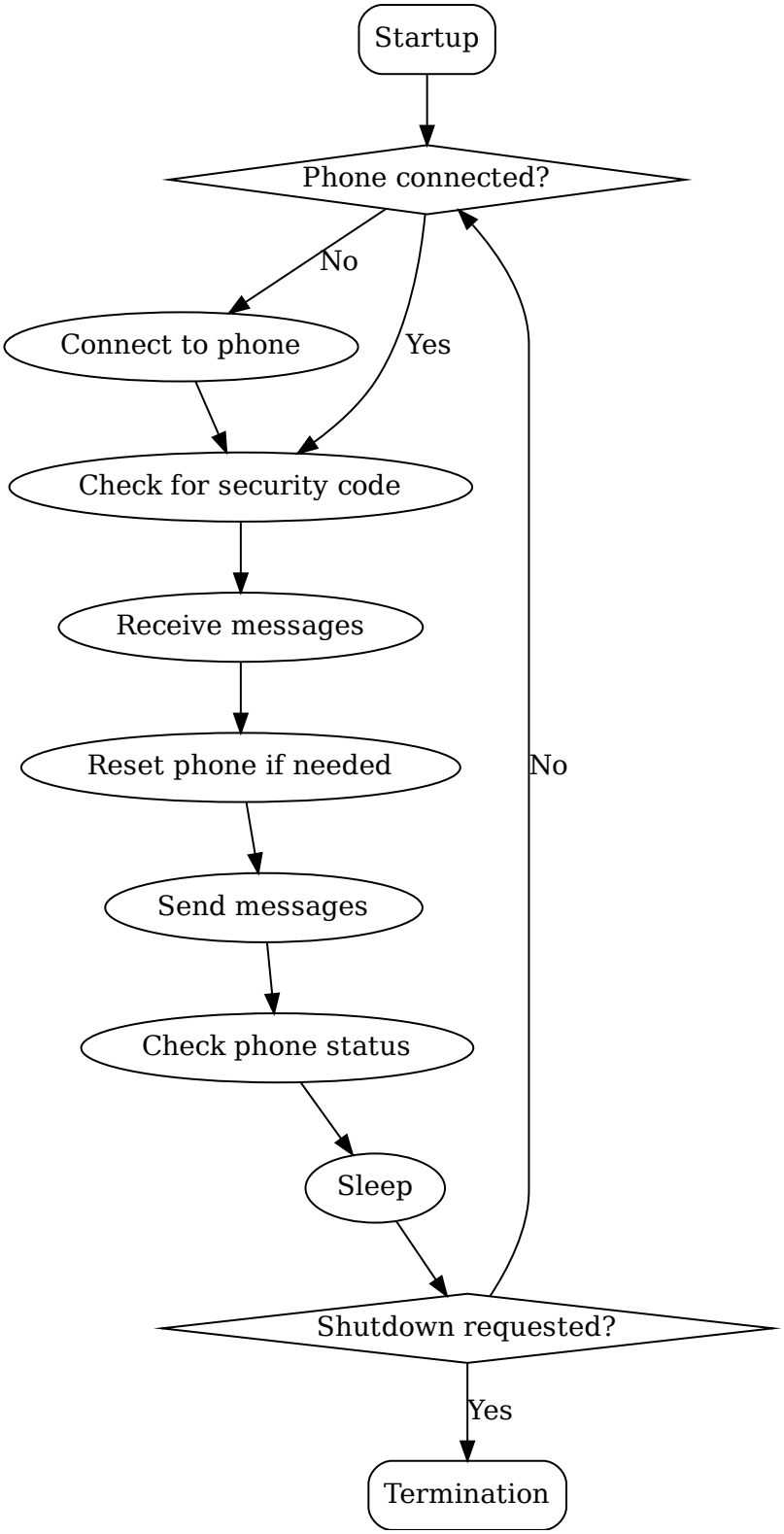
1.1 Overall schema

The interactions of SMS Daemon and related components can be seen on following picture.



1.2 SMSD operation

The SMSD operation consist of several steps.


Startup
Phone connected?
No
Yes
Connect to phone
Check for security code
Receive messages
Reset phone if needed
No
Send messages
Check phone status
Sleep
Shutdown requested?
Yes
Termination

1. Process command line options.
2. Configure backend service.
3. **Main loop is executed until it is signalled to be terminated.**
    1. Try to connect to phone if not connected.
    2. Check for security code if configured (configured by *CheckSecurity*).
    3. Check for received messages (frequency configured by *ReceiveFrequency*).
    4. Check for reset of the phone if configured (frequency configured by *ResetFrequency*).
    5. Check for messages to send (frequency configured by *CommTimeout*).
    6. Check phone status (frequency configured by *StatusFrequency*).
    7. Sleep for defined time (*LoopSleep*).
4. Backend service is freed.

CHAPTER

TWO

USAGE

This chapter will describe basic ways of using SMSD. It's use is not limited to these, but they can give you overview of SMSD abilities.

2.1 Storing Messages in Backend

The standard mode of operating SMSD. You simply configure backend service, and all received messages will end up in it and any message you put into outbox storage will be sent.

2.2 Creating Messages to Send

Creating of messages to send heavily depends on service backend you use. Most of them support *gammu-smsd-inject*, which can be used to construct the message, or you can just insert message manually to the backend storage.

Alternatively you can use `SMSD_InjectSMS()` (from C) or using `gammu.smsd.SMSD.InjectSMS()` (from Python).

2.3 Notification about Received Messages

Once SMSD receives message and stores it in backend service, it can invoke your own program to do any message processing, see *RunOnReceive Directive*.

2.4 Monitoring SMSD Status

You can use *gammu-smsd-monitor* to monitor status of SMSD. It uses shared memory segment to get current status of running SMSD.

Alternatively you can get the same functionality from libGammu using `SMSD_GetStatus()` or python-gammu using `gammu.smsd.SMSD.GetStatus()`.

2.5 Reporting Bugs

Please report bugs to <https://github.com/gammu/gammu/issues>.

Before reporting a bug, please enable verbose logging in SMSD configuration by *DebugLevel* and *LogFile*:

```
[gammu]
connection = your connection setting
port = your port name
logformat = textalldate

[smsd]
debuglevel = 255
logfile = smsd.log
```

and include this verbose log within bug report.

CHAPTER

THREE

PROGRAM MANUALS

3.1 gammu-smsd

3.1.1 Synopsis

```
gammu-smsd [OPTION]...
```

3.1.2 Description

This manual page documents briefly the **gammu-smsd** command.

gammu-smsd is a program that periodically scans GSM modem for received messages, stores them in defined storage and also sends messages enqueued in this storage.

The daemon can reload configuration file after sending hangup signal (SIGHUP) and properly terminates itself on SIGINT and SIGTERM.

Program accepts following options (please note that long options might be not accepted on some platforms):

-h, **--help**
: Shows help.

-v, **--version**
: Shows version information and compiled in features.

-c, **--config**=file
: Configuration file to use, default is /etc/gammu-smsdrc, on Windows there is no default and configuration file path has to be always specified.

  If you run SMSD as a system daemon (or service), it is recommended to use absolute path to configuration file as startup directory might be different than you expect.

  See *SMSD Configuration File* for configuration file documentation.

-p, **--pid**=file
: Lock file for storing pid, empty for no locking. Not supported on Windows.

-U, **--user**=user
: Drop daemon privileges to chosen user after starting.

-G, **--group**=group
: Drop daemon privileges to chosen group after starting.

-d, --daemon
Daemonize program on startup. Not supported on Windows.

-i, --install-service
Installs SMSD as a Windows service.

-u, --uninstall-service
Uninstalls SMSD as a Windows service.

-s, --start-service
Starts SMSD Windows service.

-k, --stop-service
Stops SMSD Windows service.

-f, --max-failures=count
Terminate after defined number of failures. Use 0 to not terminate (this is default).

-X, --suicide=seconds
Kills itself after number of seconds.

-S, --run-service
Runs pogram as SMSD Windows service. This should not be used manually, but only Windows Service manager should use this command.

-n, --service-name=name
Defines name of a Windows service. Each service requires an unique name, so if you want to run several SMSD instances, you have to name each service differently. Default is "GammuSMSD".

-l, --use-log
Use logging as configured in config file (default).

-L, --no-use-log
Do not use logging as configured in config file.

-e, --install-event-log
Installs Windows EventLog description to registry.

New in version 1.31.90.

-E, --uninstall-event-log
Uninstalls Windows EventLog description to registry.

New in version 1.31.90.

3.1.3 Signals

SMSD can be controlled using following POSIX signals (if your platform supports this):

SIGHUP
Reload configuration and reconnect to phone.

SIGINT, SIGTERM
Gracefully shutdown the daemon.

SIGALRM
Used internally for `gammu-smsd -X`

SIGUSR1
Suspends SMSD operation, closing connection to phone and database.

SIGUSR2
Resumes SMSD operation (after previous suspend).

Changed in version 1.22.91: Added support for SIGHUP.

Changed in version 1.22.95: Added support for SIGALRM.

Changed in version 1.31.90: Added support for SIGUSR1 and SIGUSR2.

3.1.4 Examples

Linux/Unix Examples

Start SMSD as a daemon on Linux:

```
gammu-smsd --config /etc/gammu-smsdrc --pid /var/run/gammu-smsd.pid --daemon
```

Start SMSD as a daemon on Linux with reduced privileges:

```
gammu-smsd --config /etc/gammu-smsdrc --pid /var/run/gammu-smsd.pid --daemon --user␣
↪gammu --group gammu
```

SMSD as a system wide daemon

To use SMSD as a daemon, you might want to use init script which is shipped with Gammu in contrib/init directory. It is not installed by default, either install it manually or check INSTALL file for instructions.

Under Windows 7 you might need to disable UAC (user account control) before you will be able to install SMSD service.

Windows Service Examples

Install Gammu SMSD Windows service:

```
gammu-smsd.exe -c c:\Gammu\smsdrc -i
```

Install two instances of SMSD Windows service:

```
gammu-smsd.exe -c c:\Gammu\smsdrc-1 -n Gammu-first-phone -i

gammu-smsd.exe -c c:\Gammu\smsdrc-2 -n Gammu-second-phone -i
```

To uninstall a Windows service:

```
gammu-smsd.exe -u
```

Troubleshooting Windows Service

If Gammu fails to start as a Windows service (you will usually get "Error 1053: The service did not respond to the start or control request in a timely fashion"), first check your SMSD logs. If they do not contain any useful hint, try starting SMSD manually with exactly same parameters as you installed the service (without -i).

For example the command line can look like:

```
gammu-smsd.exe -c smsdrc
```

You now should be able to get errors from SMSD even if it fails to start as a service.

Invoking Gammu and suspending SMSD

As you can not run Gammu and Gammu SMSD at same time on single device, you can workaround this limitation by suspending SMSD temporarily using *SIGUSR1* and *SIGUSR2* signals (see also *Signals*):

```
SMSD_PID=`pidof gammu-smsd`
if [ -z "$SMSD_PID" ] ; then
    echo "Failed to figure out SMSD PID!"
else
    kill -SIGUSR1 $SMSD_PID
    gammu identify
    kill -SIGUSR2 $SMSD_PID
fi
```

Or even create a *gammu-safe* script:

```
#!/bin/bash
SMSD_PID=`pidof gammu-smsd`
if [ -z "$SMSD_PID" ] ; then
  gammu $@
else
  tty=$(lsof |grep -E "gammu-sms\s+$SMSD_PID\s+.*/dev/tty*"|awk {'print $NF'})
  kill -SIGUSR1 $SMSD_PID
  while test "$(fuser $ttyfuser $tty 2> /dev/null|xargs)" = $SMSD_PID
  do
    sleep 1
  done
  sleep 1
  gammu $@
  kill -SIGUSR2 $SMSD_PID
  while test "$(fuser $ttyfuser $tty 2> /dev/null|xargs)" != $SMSD_PID
  do
    sleep 1
  done
  sleep 1
fi
```

3.1.5 Known Limitations

You can not use same phone by more programs in same time. However in case you did not enable locking in `[gammu]` section, it might be able to start the communication with phone from more programs. In this case neither of the programs will probably work, see *Invoking Gammu and suspending SMSD* for workaround.

There is no way to detect that SMS message is reply to another by looking at message headers. The only way to achieve this is to add some token to the message and let the user include it in the message on reply.

3.2 gammu-smsd-inject

3.2.1 Synopsis

```
gammu-smsd-inject [OPTION]... MESSAGETYPE RECIPIENT [MESSAGE_PARAMETER]...
```

3.2.2 Description

This manual page documents briefly the **gammu-smsd-inject** command.

gammu-smsd-inject is a program that enqueues message in Gammu SMS Daemon, which will be later sent by the daemon using connected GSM modem.

Support for this program depends on features available in currently used SMSD service backend, however currently it is supported by all of them.

Program accepts following options (please note that long options might be not accepted on some platforms):

-h, **--help**
: Shows help.

-v, **--version**
: Shows version information and compiled in features.

-c, **--config**=file
: Configuration file to use, default is /etc/gammu-smsdrc, on Windows there is no default and configuration file path has to be always specified.

-l, **--use-log**
: Use logging as configured in config file.

-L, **--no-use-log**
: Do not use logging as configured in config file (default).

For description of message types and their parameters, please check documentation for `gammu savesms`.

3.2.3 Examples

To check it out, you need to have configuration file for SMSD, see *SMSD Configuration File* for more details about it.

Inject text message up to standard 160 chars:

```
echo "All your base are belong to us" | gammu-smsd-inject TEXT 123456
```

or

```
gammu-smsd-inject TEXT 123456 -text "All your base are belong to us"
```

Inject unicode text message:

```
gammu-smsd-inject TEXT 123456 -unicode -text "Zkouška sirén"
```

Inject long text message:

```
echo "All your base are belong to us" | gammu-smsd-inject TEXT 123456 -len 400
```

or

```
gammu-smsd-inject TEXT 123456 -len 400 -text "All your base are belong to us"
```

or

```
gammu-smsd-inject EMS 123456 -text "All your base are belong to us"
```

Inject some funky message with predefined sound and animation from 2 bitmaps:

```
gammu-smsd-inject EMS 123456 -text "Greetings" -defsound 1 -text "from Gammu" -tone10␣
↪axelf.txt -animation 2 file1.bmp file2.bmp
```

Inject protected message with ringtone:

```
gammu-smsd-inject EMS 123456 -protected 2 -variablebitmaplong ala.bmp -toneSElong axelf.
↪txt -toneSE ring.txt
```

Inject USSD query:

```
gammu-smsd-inject USSD '*101#'
```

3.3 gammu-smsd-monitor

3.3.1 Synopsis

```
gammu-smsd-monitor [OPTION]...
```

3.3.2 Description

This manual page documents briefly the **gammu-smsd-monitor** command.

gammu-smsd-monitor is a program that monitors state of Gammu SMS Daemon. It periodically displays information about phone and number of processed messages.

Program accepts following options (please note that long options might be not accepted on some platforms):

-h, **--help**

Shows help.

-v, **--version**

Shows version information and compiled in features.

-c, **--config**=file

Configuration file to use, default is /etc/gammu-smsdrc, on Windows there is no default and configuration file path has to be always specified.

-n, **--loops**=count

Number of loops, by default monitor loops infinitely.

-d, **--delay**=seconds

Delay between polling SMSD state, default is 20 seconds.

-C, **--csv**

Print output in comma separated values format:

```
client;phone ID;IMEI;sent;received;failed;battery;signal
```

-l, **--use-log**

Use logging as configured in config file.

-L, **--no-use-log**

Do not use logging as configured in config file (default).

CHAPTER

FOUR

SMSD CONFIGURATION FILE

4.1 Description

gammu-smsd reads configuration from a config file. It's location can be specified on command line, otherwise default path `/etc/gammu-smsdrc` is used.

This file use ini file syntax, see ini.

Configuration file of gammu-smsd consists of at least two sections - `[gammu]` and *`[smsd]`*. For *SQL Service* you can also use *`[sql]`* and *`[tables]`*.

The `[gammu]` section is configuration of a phone connection and is same as described in gammurc with the only exception that *`LogFile`* is ignored and common logging for gammu library and SMS daemon is used. However the `LogFormat` directive still configures how much messages gammu emits.

`[smsd]`

The *`[smsd]`* section configures SMS daemon itself, which are described in following subsections. First general parameters of SMS daemon are listed and then specific parameters for storage backends.

`[include_numbers]`

List of numbers from which accept messages, see *Message filtering*.

`[exclude_numbers]`

List of numbers from which reject messages, see *Message filtering*.

`[include_smsc]`

List of SMSC numbers from which accept messages, see *Message filtering*.

`[exclude_smsc]`

List of SMSC numbers from which reject messages, see *Message filtering*.

`[sql]`

Configure SQL queries used by *SQL Service*, you usually don't have to modify them.

See also:

Configurable queries

`[tables]`

Configure SQL table names used by *SQL Service*, you usually don't have to modify them.

See also:

Tables

4.2 General parameters of SMS daemon

Service

SMSD service to use, one of following choices:

FILES
Stores messages in files, see *Files backend* for details.

NULL
Does not store messages at all, see *Null Backend* for details.

SQL
Stores messages in SQL database, see *SQL Service* for details, choose database type to use by `Driver`.

New in version 1.28.93.

MYSQL
Deprecated since version 1.28.93: Use `Service` = `SQL` and `Driver` = `native_mysql` instead.

Compatibility option for older configuration files, stores messages in MySQL database, see *MySQL Backend* for details.

PGSQL
Deprecated since version 1.28.93: Use `Service` = `SQL` and `Driver` = `native_pgsql` instead.

Compatibility option for older configuration files, stores messages in PostgreSQL database, see *PostgreSQL Backend* for details.

DBI
Deprecated since version 1.28.93: Use `Service` = `SQL` and `Driver` = DBI driver instead.

Compatibility option for older configuration files, stores messages in any database supported by libdbi, see *DBI Backend* for details.

Note: Availability of backends depends on platform and compile time configuration.

PIN

PIN for SIM card. This is optional, but you should set it if your phone after power on requires PIN.

NetworkCode

Network personalisation password. This is optional, but some phones require it after power on.

PhoneCode

Phone lock password. This is optional, but some phones require it after power on.

LogFile

File where SMSD actions are being logged. You can also use special value `syslog` which will send all messages to syslog daemon. On Windows another special value `eventlog` exists, which will send logs to Windows Event Log.

If you run SMSD as a system daemon (or service), it is recommended to use absolute path to log file as startup directory might be different than you expect.

Default is to provide no logging.

Note: For logging to Windows Event Log, it is recommended to install Event Log source by invoking `gammu-smsd -e` (this is automatically done during installation of Gammu).

LogFacility

Facility to use on logging backends which support it (currently only syslog). One of following chouces:

- DAEMON (default)
- USER
- LOCAL0
- LOCAL1
- LOCAL2
- LOCAL3
- LOCAL4
- LOCAL5
- LOCAL6
- LOCAL7

New in version 1.30.91.

DebugLevel

Debug level for SMSD. The integer value should be sum of all flags you want to enable.

1
: enables basic debugging information

2
: enables logging of SQL queries of service backends

4
: enables logging of gammu debug information

Generally to get as much debug information as possible, use 255.

Default is 0, what should mean no extra information.

CommTimeout

How many seconds should SMSD wait after there is no message in outbox before scanning it again.

Default is 30.

SendTimeout

Shows how many seconds SMSD should wait for network answer during sending sms. If nothing happen during this time, sms will be resent.

Default is 30.

MaxRetries

How many times will SMSD try to resend message if sending fails. This is tracked per message and currently supported only with SQL backends.

Default is 1.

RetryTimeout

How long to wait before resending failed message (needs to be enabled by *MaxRetries*).

Is used in *update_retries*.

Default is 600.

ReceiveFrequency

The number of seconds between testing for received SMSes, when the phone is busy sending SMSes. Normally a test for received SMSes is done every *CommTimeout* seconds and after each sent SMS.

Default is 15.

StatusFrequency

The number of seconds between refreshing phone status (battery, signal) stored in shared memory and possibly in service backends. Use 0 to disable.

You might want to increase this for higher throughput.

Default is 60.

LoopSleep

The number of seconds how long will SMSD sleep before checking for some activity. Please note that setting this to higher value than 1 will have effects to other time based configurations, because they will be effectively rounded to multiply of this value.

Setting this to 0 disables sleeping. Please note this might cause Gammu to consume quite a lot of CPU power as it will effectively do busy loop.

This sleep is utilized only if the main loop (sending and receiving messages) takes less than defined time. For example if you set LoopSleep to 5 seconds and sending messages take 10 seconds, no sleep will be done in the iteration which is sending messages. Also the sleep time is lowered by the already processed time.

Default is 1.

MultipartTimeout

The number of seconds how long will SMSD wait for all parts of multipart message. If all parts won't arrive in time, parts will be processed as separate messages.

Default is 600 (10 minutes).

CheckSecurity

Whether to check if phone wants to enter PIN.

Default is 1 (enabled).

HangupCalls

New in version 1.34.0.

Whether to automatically hangup any incoming calls.

Default is 0 (disabled).

CheckBattery

Whether to check phone battery state periodically.

Default is 1 (enabled).

CheckSignal

Whether to check signal level periodically.

Default is 1 (enabled).

CheckNetwork

New in version 1.37.90.

Whether to check network status periodically.

If phone is reported to be not on the network, SMSD tries to power it on.

Default is 1 (enabled).

ResetFrequency

The number of seconds between performing a preventive soft reset in order to minimize the cases of hanging phones e.g. Nokia 5110 will sometimes freeze to a state when only after unmounting the battery the phone will be functional again.

Default is 0 (not used).

HardResetFrequency

New in version 1.28.92.

> **Warning:** For some phones hard reset means deleting all data in it. Use *ResetFrequency* instead, unless you know what you are doing.

The number of seconds between performing a preventive hard reset in order to minimize the cases of hanging phones.

Default is 0 (not used).

DeliveryReport

Whether delivery reports should be used, one of `no`, `log`, `sms`.

log
one line log entry,

sms
store in inbox as a received SMS

no
no delivery reports

Default is `no`.

DeliveryReportDelay

Delay in seconds how long is still delivery report considered valid. This depends on brokenness of your network (delivery report should have same timestamp as sent message). Increase this if delivery reports are not paired with sent messages.

Default is 600 (10 minutes).

PhoneID

String with info about phone used for sending/receiving. This can be useful if you want to run several SMS daemons (see *Multiple modems*).

When you set PhoneID, all messages (including injected ones) will be marked by this string (stored as SenderID in the database) and it allows more SMS daemons to share a single database.

SMSD daemon will in such case send *outbox* messages only with matching or empty SenderID.

This option has actually no effect with *Files backend*.

SMSC

New in version 1.36.2.

SMSC number to use for sending messages if not specified in the message (see options of *gammu-smsd-inject*).

In most cases you don't need this settings as Gammu tries to read correct SMSC from phone, but sometimes this fails (try `gammu getsmsc`).

RunOnReceive

Executes a program after receiving message.

This parameter is executed through shell, so you might need to escape some special characters and you can include any number of parameters. Additionally parameters with identifiers of received messages are appended to the command line. The identifiers depend on used service backend, typically it is ID of inserted row for database backends or file name for file based backends.

Gammu SMSD waits for the script to terminate. If you make some time consuming there, it will make SMSD not receive new messages. However to limit breakage from this situation, the waiting time is limited to two minutes. After this time SMSD will continue in normal operation and might execute your script again.

The process has available lot of information about received message in environment, check *RunOnReceive Directive* for more details.

RunOnFailure

New in version 1.28.93.

Executes a program on failure.

This can be used to proactively react on some failures or to interactively detect failure of sending message.

The program will receive optional parameter, which can currently be either `INIT` (meaning failure during phone initialization) or message ID, which would indicate error while sending the message.

Note: The environment with message (as is in *RunOnReceive*) is not passed to the command.

RunOnSent

New in version 1.36.4.

Executes a program after sending message.

The program will receive optional parameter a message ID and environment with message details as described in *RunOnReceive Directive*.

RunOnIncomingCall

New in version 1.38.5.

Executes a program after cancelling incoming call.

The program will receive a parameter with a phone number of the call. This requires *HangupCalls* to be enabled.

IncludeNumbersFile

File with list of numbers which are accepted by SMSD. The file contains one number per line, blank lines are ignored. The file is read at startup and is reread only when configuration is being reread. See Message filtering for details.

ExcludeNumbersFile

File with list of numbers which are not accepted by SMSD. The file contains one number per line, blank lines are ignored. The file is read at startup and is reread only when configuration is being reread. See Message filtering for details.

IncludeSMSCFile

File with list of SMSC numbers which are accepted by SMSD. The file contains one number per line, blank lines are ignored. The file is read at startup and is reread only when configuration is being reread. See Message filtering for details.

ExcludeSMSCFile

File with list of SMSC numbers which are not accepted by SMSD. The file contains one number per line, blank lines are ignored. The file is read at startup and is reread only when configuration is being reread. See Message filtering for details.

BackendRetries

How many times will SMSD backend retry operation.

The implementation on different backends is different, for database backends it generally means how many times it will try to reconnect to the server.

Default is 10.

Send

New in version 1.28.91.

Whether to enable sending of messages.

Default is True.

Receive

New in version 1.28.91.

Whether to enable receiving of messages.

Default is True.

4.3 Database backends options

All DBI, ODBC, MYSQL and PGSQL backends (see *MySQL Backend*, *ODBC Backend*, *PostgreSQL Backend*, *DBI Backend* for their documentation) supports same options for configuring connection to a database:

User

User name used for connection to a database.

Password

Password used for connection to a database.

Host

Database server address. It can also contain port or socket path after semicolon, for example `localhost:/path/to/socket` or `192.168.1.1:8000`.

For ODBC this is used as Data source name.

Note: Some database servers differentiate usage of `localhost` (to use local socket) and `127.0.0.1` (to use locat TCP/IP connection). Please make sure your SMSD settings match the database server ones.

New in version 1.28.92.

PC

Deprecated since version 1.28.92: Please use *Host* instead.

Synonym for *Host*, kept for backwards compatibility.

`Database`

Name of database (or schema) to use and where SMSD can find it's tables.

Please note that you should create tables in this database before using gammu-smsd. SQL files for creating needed tables are included in documentation for individual database backends: *MySQL Backend*, *ODBC Backend*, *PostgreSQL Backend*, *DBI Backend*

`SkipSMSCNumber`

When you send sms from some SMS centers you can have delivery reports from other SMSC number. You can set here number of this SMSC used by you and Gammu will not check it's number during assigning reports to sent SMS.

`Driver`

SQL driver to use, Gammu supports several native drivers and generic interface using ODBC and DBI. Availability of the backends depends on compile time options.

Available drivers:

`odbc`

Connects to the database using ODBC, see *ODBC Backend*.

`native_mysql`

Stores messages in MySQL database, see *MySQL Backend* for details.

`native_pgsql`

Stores messages in PostgreSQL database, see *PostgreSQL Backend* for details.

`db2`, `firebird`, `freetds`, `ingres`, `msql`, `mysql`, `oracle`, `pgsql`, `sqlite`, `sqlite3`

Stores messages using DBI library in given backend. You need to have installed appropriate DBI driver to make it work. See *DBI Backend* for details.

`SQL`

SQL dialect to use. This is specially useful with *ODBC Backend* where SMSD does not know which server it is actually talking to.

Possible values:

- `mysql` - MySQL
- `pgsql` - PostgreSQL
- `sqlite` - SQLite
- `mssql` - Microsoft SQL Server
- `sybase` - Sybase
- `access` - Microsoft Access
- `oracle` - Oracle
- `odbc` - Generic ODBC

New in version 1.28.93.

See also:

You can also completely customize SQL queries used as described in *SQL Queries*.

`DriversPath`

Path, where DBI drivers are stored, this usually does not have to be set if you have properly installed drivers.

DBDir

Database directory for some (currently only sqlite) DBI drivers. Set here path where sqlite database files are stored.

4.3.1 Files backend options

The FILES backend accepts following configuration options. See *Files backend* for more detailed service backend description. Please note that all path should contain trailing path separator (/ on Unix systems):

InboxPath

Where the received SMSes are stored.

Default is current directory.

OutboxPath

Where SMSes to be sent should be placed.

Default is current directory.

SentSMSPath

Where the transmitted SMSes are placed, if same as *OutboxPath* transmitted messages are deleted.

Default is to delete transmitted messages.

ErrorSMSPath

Where SMSes with error in transmission is placed.

Default is same as *SentSMSPath*.

InboxFormat

The format in which the SMS will be stored: `detail`, `unicode`, `standard`.

detail
format used for message backup by gammu, see gammu-smsbackup.

unicode
message text stored in unicode (UTF-16)

standard
message text stored in system charset

The `standard` and `unicode` settings do not apply for 8-bit messages, which are always written raw as they are received with extension .bin.

Default is `unicode`.

Note: In `detail` format, all message parts are stored into single file, for all others each message part is saved separately.

OutboxFormat

The format in which messages created by *gammu-smsd-inject* will be stored, it accepts same values as InboxFormat.

Default is `detail` if Gammu is compiled in with backup functions, `unicode` otherwise.

TransmitFormat

The format for transmitting the SMS: `auto`, `unicode`, `7bit`.

This option is used only if *OutboxFormat* is not set to `detail`. In such case encoding specified in the message is used (you can specify it to *gammu-smsd-inject*).

Default is `auto`.

4.4 Message filtering

SMSD allows one to process only limited subset of incoming messages. You can define filters for sender number in *[include_numbers]* and *[exclude_numbers]* sections or using *IncludeNumbersFile* and *ExcludeNumbersFile* directives.

If *[include_numbers]* section exists, all values (keys are ignored) from it are used as allowed phone numbers and no other message is processed. On the other side, in *[exclude_numbers]* you can specify numbers which you want to skip.

Lists from both sources are merged together. If there is any number in include list, only include list is used and only messages in this list are being accepted. If include list is empty, exclude list can be used to ignore messages from some numbers. If both lists are empty, all messages are accepted.

Similar filtering rules can be used for SMSC number filtering, they just use different set of configuration options - *[include_smsc]* and *[exclude_smsc]* sections or *IncludeSMSCFile* and *ExcludeSMSCFile* directives.

4.5 Examples

There is more complete example available in Gammu documentation. Please note that for simplicity following examples do not include `[gammu]` section, you can look into gammurc for some examples how it can look like.

4.5.1 Files service

SMSD configuration file for FILES backend could look like:

```
[smsd]
Service = files
PIN = 1234
LogFile = syslog
InboxPath = /var/spool/sms/inbox/
OutboxPath = /var/spool/sms/outbox/
SentSMSPath = /var/spool/sms/sent/
ErrorSMSPath = /var/spool/sms/error/
```

4.5.2 MySQL service

If you want to use MYSQL backend, you will need something like this:

```
[smsd]
Service = sql
Driver = native_mysql
PIN = 1234
LogFile = syslog
User = smsd
Password = smsd
PC = localhost
Database = smsd
```

4.5.3 DBI service using SQLite

For *DBI Backend* backend, in this particular case SQLite:

```
[smsd]
Service = sql
Driver = sqlite3
DBDir = /var/lib/sqlite3
Database = smsd.db
```

4.5.4 ODBC service using MySQL

For *ODBC Backend* backend, in this particular case using DSN `smsd` server:

```
[smsd]
Service = sql
Driver = odbc
Host = smsd
```

The DSN definition (in `~/.odbc.ini` on UNIX) for using MySQL server would look like:

```
[smsd]
Description         = MySQL
Driver              = MySQL
Server              = 127.0.0.1
Database            = smsd
Port                =
Socket              =
Option              =
Stmt                =

[smsdsuse]
Driver              = MySQL ODBC 3.51.27r695 Driver
DATABASE            = smsd
SERVER              = 127.0.0.1
```

4.5.5 Numbers filtering

Process only messages from 123456 number:

```
[include_numbers]
number1 = 123456
```

Do not process messages from evil number 666:

```
[exclude_numbers]
number1 = 666
```

4.5.6 Debugging

Enabling debugging:

```
[smsd]
debuglevel = 255
logfile = smsd.log
```

4.5.7 Multiple modems

You can run any number of SMSD instances and they can even share same backend database. For routing the messages, you need to set different *PhoneID* for each instance and set `SenderID` column in *outbox* table.

Following example shows configuration for two modems, but you can have any number of SMSD instances. The only limitation is performance of your hardware, especially if all modems are connected using USB.

Configuration for first SMSD:

```
[gammu]
device = /dev/ttyACM0
connection = at

[smsd]
Service = sql
Driver = native_mysql
PIN = 1234
LogFile = syslog
User = smsd
Password = smsd
PC = localhost
Database = smsd
PhoneID = first
```

Configuration for second SMSD:

```
[gammu]
device = /dev/ttyACM1
connection = at

[smsd]
```

(continues on next page)

(continued from previous page)

```
Service = sql
Driver = native_mysql
PIN = 1234
LogFile = syslog
User = smsd
Password = smsd
PC = localhost
Database = smsd
PhoneID = second
```

You can then start two separate instances of SMSD:

```
gammu-smsd -c /path/to/first-smsdrc
gammu-smsd -c /path/to/second-smsdrc
```

CHAPTER

FIVE

RUNONRECEIVE DIRECTIVE

5.1 Description

Gammu SMSD can be configured by `RunOnReceive` directive (see *SMSD Configuration File* for details) to run defined program after receiving every message. It can receive single message or more messages, which are parts of one multipart message.

This parameter is executed through shell, so you might need to escape some special characters and you can include any number of parameters. Additionally parameters with identifiers of received messages are appended to the command line. The identifiers depend on used service backend, typically it is ID of inserted row for database backends or file name for file based backends.

Gammu SMSD waits for the script to terminate. If you make some time consuming there, it will make SMSD not receive new messages. However to limit breakage from this situation, the waiting time is limited to two minutes. After this time SMSD will continue in normal operation and might execute your script again.

Note: All input and output file descriptors are closed when this program is invoked, so you have to ensure to open files on your own.

5.2 Environment

New in version 1.28.0.

Program is executed with environment which contains lot of information about the message. You can use it together with NULL service (see *Null Backend*) to implement completely own processing of messages.

5.2.1 Global variables

SMS_MESSAGES

Number of physical messages received.

DECODED_PARTS

Number of decoded message parts.

PHONE_ID

New in version 1.38.2.

Value of `PhoneID`. Useful when running multiple instances (see *Multiple modems*).

5.2.2 Per message variables

The variables further described as `SMS_1_...` are generated for each physical message, where 1 is replaced by current number of message.

SMS_1_CLASS

Class of message.

SMS_1_NUMBER

Sender number.

SMS_1_TEXT

Message text. Text is not available for 8-bit binary messages.

SMS_1_REFERENCE

New in version 1.38.5.

Message Reference. If delivery status received, this variable contains TPMR of original message

5.2.3 Per part variables

The variables further described as `DECODED_1_...` are generated for each message part, where 1 is replaced by current number of part. Set are only those variables whose content is present in the message.

DECODED_1_TEXT

Decoded long message text.

DECODED_1_MMS_SENDER

Sender of MMS indication message.

DECODED_1_MMS_TITLE

title of MMS indication message.

DECODED_1_MMS_ADDRESS

Address (URL) of MMS from MMS indication message.

See also:

faq-mms-download

DECODED_1_MMS_SIZE

Size of MMS as specified in MMS indication message.

5.3 Examples

5.3.1 Activating RunOnReceive

To activate this feature you need to set *RunOnReceive* in the *SMSD Configuration File*.

```
[smsd]
RunOnReceive = /path/to/script.sh
```

5.3.2 Processing messages from the files backend

Following script (if used as *RunOnReceive* handler) passes message data to other program. This works only with the *Files backend*.

```
#!/bin/sh
INBOX=/path/to/smsd/inbox
PROGRAM=/bin/cat
for ID in "$@" ; do
    $PROGRAM < $INBOX/$ID
done
```

5.3.3 Invoking commands based on message text

Following script (if used as *RunOnReceive* handler) executes given programs based on message text.

```
#!/bin/sh

# Check for sender number
if [ "$SMS_1_NUMBER" != "+420123456789" ] ; then
    exit
fi

# Handle commands
case "$SMS_1_TEXT" in
    "DMS A")
        /usr/bin/dms-a
        ;;
    "DMS B")
        /usr/bin/dms-b
        ;;
esac
```

5.3.4 Passing message text to program

Following script (if used as *RunOnReceive* handler) passes message text and sender to external program.

```
#!/bin/sh
PROGRAM=/bin/echo
for i in `seq $SMS_MESSAGES` ; do
    eval "$PROGRAM \"\${SMS_${i}_NUMBER}\" \"\${SMS_${i}_TEXT}\""
done
```

5.3.5 Passing MMS indication parameters to external program

Following script (if used as *RunOnReceive* handler) will write information about each received MMS indication to the log file. Just replace echo command with your own program to do custom processing.

```
#!/bin/sh
if [ $DECODED_PARTS -eq 0 ] ; then
    # No decoded parts, nothing to process
    exit
fi
if [ "$DECODED_1_MMS_ADDRESS" ] ; then
    echo "$DECODED_1_MMS_ADDRESS" "$DECODED_1_MMS_SENDER" "$DECODED_1_MMS_TITLE" >> /tmp/
↪smsd-mms.log
fi
```

5.3.6 Processing message text in Python

Following script (if used as *RunOnReceive* handler) written in Python will concatenate all text from received message:

```
#!/usr/bin/env python

import os
import sys

numparts = int(os.environ["DECODED_PARTS"])

text = ""
# Are there any decoded parts?
if numparts == 0:
    text = os.environ["SMS_1_TEXT"]
# Get all text parts
else:
    for i in range(1, numparts + 1):
        varname = "DECODED_%d_TEXT" % i
        if varname in os.environ:
            text = text + os.environ[varname]

# Do something with the text
print("Number {} have sent text: {}".format(os.environ["SMS_1_NUMBER"], text))
```

CHAPTER

SIX

BACKEND SERVICES

The backend service is used to store messages (both incoming and queue of outgoing ones).

6.1 Files backend

6.1.1 Description

FILES backend stores all data on a filesystem in folders defined by configuration (see *SMSD Configuration File* for description of configuration options).

6.1.2 Receiving of messages

Received messages are stored in a folder defined by configuration. The filename will be `IN<date>_<time>_<serial>_<sender>_<sequence>.<ext>`, for example `IN20021130_021531_00_+45409000931640979_00.txt`.

Explanation of fields:

<date>
: date in format `YYYYMMDD`

<time>
: time in format `HHMMSS`

<sender>
: sender number

<serial>
: order of a message (in case more messages were received at same time), in format `NN`

<sequence>
: part of the message for multipart messages, in format `NN`

<ext>
: `txt` for text message, 8-bit messages are stored with `bin` extension, `smsbackup` for gammu-smsbackup

The content of the file is content of the message and the format is defined by configuration directive *InboxFormat* (see *SMSD Configuration File*).

6.1.3 Transmitting of messages

Transmitted messages are read from a folder defined by configuration. The filename should be one of the following formats:

- `OUT<recipient>.<ext>`
- `OUT<priority>_<recipient>_<serial>.<ext>`
- `OUT<priority><date>_<time>_<serial>_<recipient>_<note>.<ext>`

Explanation of fields:

<recipient>
: recipient number where to send message

<priority>
: an alphabetic character (A-Z) A = highest priority

<ext>
: `txt` for normal text SMS, `smsbackup` for gammu-smsbackup

<note>
: any arbitrary text which is ignored

For text messages, you can additionally append flags to extension:

d
: delivery report requested

f
: flash SMS

b
: WAP bookmark as name,URL

Other fields are same as for received messages.

For example OUTG20040620_193810_123_+4512345678_xpq.txtdf is a flash text SMS requesting delivery reports.

SMSes will be transmitted sequentially based on the file name. The contents of the file is the SMS to be transmitted (in Unicode or standard character set).

The contents of the file is the SMS to be transmitted (in Unicode or standard character set), for WAP bookmarks it is split on as Name,URL, for text messages whole file content is used.

Please note that if file is not in Unicode, encoding is detected based on locales, which do not have to be configured if SMSD is running from init script. If this is your case, please add locales definition to init script.

6.2 SQL Service

6.2.1 Description

SQL service stores all its data in database. It can use one of these SQL backends (configuration option *Driver* in smsd section):

- `native_mysql` for *MySQL Backend*
- `native_pgsql` for *PostgreSQL Backend*
- `odbc` for *ODBC Backend*

- **drivers supported by DBI for *DBI Backend*, which include:**
  - `sqlite3` - for SQLite 3
  - `mysql` - for MySQL
  - `pgsql` - for PostgeSQL
  - `freetds` - for MS SQL Server or Sybase

6.2.2 SQL connection parameters

Common for all backends:

- *`User`* - user connecting to database
- *`Password`* - password for connecting to database
- *`Host`* - database host or data source name
- *`Database`* - database name
- *`Driver`* - `native_mysql`, `native_pgsql`, `odbc` or DBI one
- *`SQL`* - SQL dialect to use

Specific for DBI:

- *`DriversPath`* - path to DBI drivers
- *`DBDir`* - sqlite/sqlite3 directory with database

See also:

The variables are fully described in gammurc documentation.

6.2.3 Tables

New in version 1.37.1.

You can customize name of all tables in the *`[tables]`*. The SQL queries will reflect this, so it's enough to change table name in this section.

`gammu`
Name of the *gammu* table.

`inbox`
Name of the *inbox* table.

`sentitems`
Name of the *sentitems* table.

`outbox`
Name of the *outbox* table.

`outbox_multipart`
Name of the *outbox_multipart* table.

phones
: Name of the *phones* table.

You can change any table name using these:

```
[tables]
inbox = special_inbox
```

6.2.4 SQL Queries

Almost all queries are configurable. You can edit them in *[sql]* section. There are several variables used in SQL queries. We can separate them into three groups:

- phone specific, which can be used in every query, see *Phone Specific Parameters*
- SMS specific, which can be used in queries which works with SMS messages, see *SMS Specific Parameters*
- query specific, which are numeric and are specific only for given query (or set of queries), see *Configurable queries*

Phone Specific Parameters

%I
: IMEI of phone

%S
: SIM IMSI

%P
: PHONE ID (hostname)

%N
: client name (eg. Gammu 1.12.3)

%O
: network code

%M
: network name

SMS Specific Parameters

%R
: remote number[1]

%C
: delivery datetime

%e
: delivery status on receiving or status error on sending

%t
: message reference

%d
: receiving datetime for received sms

[1] Sender number for received messages (insert to inbox or delivery notifications), destination otherwise.

%E
encoded text of SMS

%c
SMS coding (ie 8bit or UnicodeNoCompression)

%F
sms centre number

%u
UDH header

%x
class

%T
decoded SMS text

%A
CreatorID of SMS (sending sms)

%V
relative validity

6.2.5 Configurable queries

All configurable queries can be set in *[sql]* section. Sequence of rows in selects are mandatory.

All default queries noted here are noted for MySQL. Actual time and time addition are selected for default queries during initialization.

delete_phone
Deletes phone from database.

Default value:

```
DELETE FROM phones WHERE IMEI = %I
```

insert_phone
Inserts phone to database.

Default value:

```
INSERT INTO phones (IMEI, ID, Send, Receive, InsertIntoDB, TimeOut, Client, Battery,
↪ Signal)
VALUES (%I, %P, %1, %2, NOW(), (NOW() + INTERVAL 10 SECOND) + 0, %N, -1, -1)
```

Query specific parameters:

%1
enable send (yes or no) - configuration option Send

%2
enable receive (yes or no) - configuration option Receive

save_inbox_sms_select
Select message for update delivery status.

Default value:

```
SELECT ID, Status, SendingDateTime, DeliveryDateTime, SMSCNumber FROM sentitems
WHERE DeliveryDateTime IS NULL AND SenderID = %P AND TPMR = %t AND␣
↪DestinationNumber = %R
```

save_inbox_sms_update_delivered

Update message delivery status if message was delivered.

Default value:

```
UPDATE sentitems SET DeliveryDateTime = %C, Status = %1, StatusError = %e WHERE ID␣
↪= %2 AND TPMR = %t
```

Query specific parameters:

%1
: delivery status returned by GSM network

%2
: ID of message

save_inbox_sms_update

Update message if there is an delivery error.

Default value:

```
UPDATE sentitems SET Status = %1, StatusError = %e WHERE ID = %2 AND TPMR = %t
```

Query specific parameters:

%1
: delivery status returned by GSM network

%2
: ID of message

save_inbox_sms_insert

Insert received message.

Default value:

```
INSERT INTO inbox (ReceivingDateTime, Text, SenderNumber, Coding, SMSCNumber, UDH,
Class, TextDecoded, RecipientID) VALUES (%d, %E, %R, %c, %F, %u, %x, %T, %P)
```

update_received

Update statistics after receiving message.

Default value:

```
UPDATE phones SET Received = Received + 1 WHERE IMEI = %I
```

refresh_send_status

Update messages in outbox.

Default value:

```
UPDATE outbox SET SendingTimeOut = (NOW() + INTERVAL 60 SECOND) + 0
WHERE ID = %1 AND (SendingTimeOut < NOW() OR SendingTimeOut IS NULL)
```

The default query calculates sending timeout based on *LoopSleep* value.

Query specific parameters:

%1
ID of message

find_outbox_sms_id

Find sms messages for sending.

Default value:

```
SELECT ID, InsertIntoDB, SendingDateTime, SenderID FROM outbox
WHERE SendingDateTime < NOW() AND SendingTimeOut <  NOW() AND
SendBefore >= CURTIME() AND SendAfter <= CURTIME() AND
( SenderID is NULL OR SenderID = '' OR SenderID = %P ) ORDER BY InsertIntoDB ASC␣
↪LIMIT %1
```

Query specific parameters:

%1
limit of sms messages sended in one walk in loop

find_outbox_body

Select body of message.

Default value:

```
SELECT Text, Coding, UDH, Class, TextDecoded, ID, DestinationNumber, MultiPart,
RelativeValidity, DeliveryReport, CreatorID FROM outbox WHERE ID=%1
```

Query specific parameters:

%1
ID of message

find_outbox_multipart

Select remaining parts of sms message.

Default value:

```
SELECT Text, Coding, UDH, Class, TextDecoded, ID, SequencePosition
FROM outbox_multipart WHERE ID=%1 AND SequencePosition=%2
```

Query specific parameters:

%1
ID of message

%2
Number of multipart message

delete_outbox

Remove messages from outbox after threir successful send.

Default value:

```
DELETE FROM outbox WHERE ID=%1
```

Query specific parameters:

%1
ID of message

delete_outbox_multipart

Remove messages from outbox_multipart after threir successful send.

Default value:

```
DELETE FROM outbox_multipart WHERE ID=%1
```

Query specific parameters:

%1
ID of message

create_outbox

Create message (insert to outbox).

Default value:

```
INSERT INTO outbox (CreatorID, SenderID, DeliveryReport, MultiPart,
InsertIntoDB, Text, DestinationNumber, RelativeValidity, Coding, UDH, Class,
TextDecoded) VALUES (%1, %P, %2, %3, NOW(), %E, %R, %V, %c, %u, %x, %T)
```

Query specific parameters:

%1
creator of message

%2
delivery status report - yes/default

%3
multipart - FALSE/TRUE

%4
Part (part number)

%5
ID of message

create_outbox_multipart

Create message remaining parts.

Default value:

```
INSERT INTO outbox_multipart (SequencePosition, Text, Coding, UDH, Class,
TextDecoded, ID) VALUES (%4, %E, %c, %u, %x, %T, %5)
```

Query specific parameters:

%1
creator of message

%2
delivery status report - yes/default

%3
multipart - FALSE/TRUE

%4
Part (part number)

%5
: ID of message

add_sent_info
: Insert to sentitems.

Default value:

```
INSERT INTO sentitems (CreatorID,ID,SequencePosition,Status,SendingDateTime,
SMSCNumber, TPMR, SenderID,Text,DestinationNumber,Coding,UDH,Class,TextDecoded,
InsertIntoDB,RelativeValidity)
VALUES (%A, %1, %2, %3, NOW(), %F, %4, %P, %E, %R, %c, %u, %x, %T, %5, %V)
```

Query specific parameters:

%1
: ID of sms message

%2
: part number (for multipart sms)

%3
: message state (SendingError, Error, SendingOK, SendingOKNoReport)

%4
: message reference (TPMR)

%5
: time when inserted in db

update_sent
: Update sent statistics after sending message.

Default value:

```
UPDATE phones SET Sent= Sent + 1 WHERE IMEI = %I
```

refresh_phone_status
: Update phone status (battery, signal).

Default value:

```
UPDATE phones SET TimeOut= (NOW() + INTERVAL 10 SECOND) + 0,
Battery = %1, Signal = %2 WHERE IMEI = %I
```

Query specific parameters:

%1
: battery percent

%2
: signal percent

update_retries
: Update number of retries for outbox message. The interval can be configured by *RetryTimeout*.

```
UPDATE outbox SET SendngTimeOut = (NOW() + INTERVAL 600 SECOND) + 0,
Retries = %2 WHERE ID = %1
```

Query specific parameters:

%1
: message ID

%2
: number of retries

6.3 MySQL Backend

6.3.1 Description

MYSQL backend stores all data in a MySQL database server, which parameters are defined by configuration (see *SMSD Configuration File* for description of configuration options).

For tables description see *SMSD Database Structure*.

This backend is based on *SQL Service*.

6.3.2 Configuration

Before running *gammu-smsd* you need to create necessary tables in the database, which is described below.

The configuration file then can look like:

```
[smsd]
service = sql
driver = native_mysql
host = localhost
```

See also:

SMSD Configuration File

6.3.3 Privileges

The user accessing the database does not need much privileges, the following privileges should be enough:

```
GRANT USAGE ON *.* TO 'smsd'@'localhost' IDENTIFIED BY 'password';

GRANT SELECT, INSERT, UPDATE, DELETE ON `smsd`.* TO 'smsd'@'localhost';

CREATE DATABASE smsd;
```

Note: For creating the SQL tables you need more privileges, especially for creating triggers, which are used for some functionality.

6.3.4 Creating tables for MySQL

Depending on MySQL version and settings please choose best fitting script to create tables:

- `mysql.sql`, requires MySQL 5.6.5 or newer
- `mysql-legacy.sql` supports legacy MySQL versions, but requires neither of *NO_ZERO_DATE*, *ANSI* or *STRICT* modes to be set in the server

SQL script `mysql.sql` for creating tables in MySQL database:

```
--
-- Database for Gammu SMSD
--
-- In case you get errors about not supported charset, please
-- replace utf8mb4 with utf8.

-- --------------------------------------------------------

--
-- Table structure for table `gammu`
--

CREATE TABLE `gammu` (
  `Version` integer NOT NULL default '0' PRIMARY KEY
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;

--
-- Dumping data for table `gammu`
--

INSERT INTO `gammu` (`Version`) VALUES (17);

-- --------------------------------------------------------

--
-- Table structure for table `inbox`
--

CREATE TABLE `inbox` (
  `UpdatedInDB` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,
  `ReceivingDateTime` timestamp NOT NULL default CURRENT_TIMESTAMP,
  `Text` text NOT NULL,
  `SenderNumber` varchar(20) NOT NULL default '',
  `Coding` enum('Default_No_Compression','Unicode_No_Compression','8bit','Default_
↪Compression','Unicode_Compression') NOT NULL default 'Default_No_Compression',
  `UDH` text NOT NULL,
  `SMSCNumber` varchar(20) NOT NULL default '',
  `Class` integer NOT NULL default '-1',
  `TextDecoded` text NOT NULL,
  `ID` integer unsigned NOT NULL auto_increment,
  `RecipientID` text NOT NULL,
  `Processed` enum('false','true') NOT NULL default 'false',
  `Status` integer NOT NULL default '-1',
  PRIMARY KEY `ID` (`ID`)
```

(continues on next page)

(continued from previous page)

```
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4 AUTO_INCREMENT=1 ;

--
-- Dumping data for table `inbox`
--


-- --------------------------------------------------------

--
-- Table structure for table `outbox`
--

CREATE TABLE `outbox` (
  `UpdatedInDB` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,
  `InsertIntoDB` timestamp NOT NULL default CURRENT_TIMESTAMP,
  `SendingDateTime` timestamp NOT NULL default CURRENT_TIMESTAMP,
  `SendBefore` time NOT NULL DEFAULT '23:59:59',
  `SendAfter` time NOT NULL DEFAULT '00:00:00',
  `Text` text,
  `DestinationNumber` varchar(20) NOT NULL default '',
  `Coding` enum('Default_No_Compression','Unicode_No_Compression','8bit','Default_
↪Compression','Unicode_Compression') NOT NULL default 'Default_No_Compression',
  `UDH` text,
  `Class` integer default '-1',
  `TextDecoded` text NOT NULL,
  `ID` integer unsigned NOT NULL auto_increment,
  `MultiPart` enum('false','true') default 'false',
  `RelativeValidity` integer default '-1',
  `SenderID` varchar(255),
  `SendingTimeOut` timestamp NULL default CURRENT_TIMESTAMP,
  `DeliveryReport` enum('default','yes','no') default 'default',
  `CreatorID` text NOT NULL,
  `Retries` int(3) default 0,
  `Priority` integer default 0,
  `Status` enum('SendingOK','SendingOKNoReport','SendingError','DeliveryOK',
↪'DeliveryFailed','DeliveryPending','DeliveryUnknown','Error','Reserved') NOT NULL␣
↪default 'Reserved',
  `StatusCode` integer NOT NULL default '-1',
  PRIMARY KEY `ID` (`ID`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;

CREATE INDEX outbox_date ON outbox(SendingDateTime, SendingTimeOut);
CREATE INDEX outbox_sender ON outbox(SenderID(250));

--
-- Dumping data for table `outbox`
--


-- --------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
--
-- Table structure for table `outbox_multipart`
--

CREATE TABLE `outbox_multipart` (
  `Text` text,
  `Coding` enum('Default_No_Compression','Unicode_No_Compression','8bit','Default_
↪Compression','Unicode_Compression') NOT NULL default 'Default_No_Compression',
  `UDH` text,
  `Class` integer default '-1',
  `TextDecoded` text,
  `ID` integer unsigned NOT NULL default '0',
  `SequencePosition` integer NOT NULL default '1',
  `Status` enum('SendingOK','SendingOKNoReport','SendingError','DeliveryOK',
↪'DeliveryFailed','DeliveryPending','DeliveryUnknown','Error','Reserved') NOT NULL␣
↪default 'Reserved',
  `StatusCode` integer NOT NULL default '-1',
  PRIMARY KEY (`ID`, `SequencePosition`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;

--
-- Dumping data for table `outbox_multipart`
--

-- --------------------------------------------------------

--
-- Table structure for table `phones`
--

CREATE TABLE `phones` (
  `ID` text NOT NULL,
  `UpdatedInDB` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,
  `InsertIntoDB` timestamp NOT NULL default CURRENT_TIMESTAMP,
  `TimeOut` timestamp NOT NULL default CURRENT_TIMESTAMP,
  `Send` enum('yes','no') NOT NULL default 'no',
  `Receive` enum('yes','no') NOT NULL default 'no',
  `IMEI` varchar(35) NOT NULL,
  `IMSI` varchar(35) NOT NULL,
  `NetCode` varchar(10) default 'ERROR',
  `NetName` varchar(35) default 'ERROR',
  `Client` text NOT NULL,
  `Battery` integer NOT NULL DEFAULT -1,
  `Signal` integer NOT NULL DEFAULT -1,
  `Sent` int NOT NULL DEFAULT 0,
  `Received` int NOT NULL DEFAULT 0,
  PRIMARY KEY (`IMEI`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;

--
-- Dumping data for table `phones`
--
```

(continues on next page)

(continued from previous page)

```
-- --------------------------------------------------------

--
-- Table structure for table `sentitems`
--

CREATE TABLE `sentitems` (
  `UpdatedInDB` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,
  `InsertIntoDB` timestamp NOT NULL default CURRENT_TIMESTAMP,
  `SendingDateTime` timestamp NOT NULL default CURRENT_TIMESTAMP,
  `DeliveryDateTime` timestamp NULL,
  `Text` text NOT NULL,
  `DestinationNumber` varchar(20) NOT NULL default '',
  `Coding` enum('Default_No_Compression','Unicode_No_Compression','8bit','Default_
↪Compression','Unicode_Compression') NOT NULL default 'Default_No_Compression',
  `UDH` text NOT NULL,
  `SMSCNumber` varchar(20) NOT NULL default '',
  `Class` integer NOT NULL default '-1',
  `TextDecoded` text NOT NULL,
  `ID` integer unsigned NOT NULL default '0',
  `SenderID` varchar(255) NOT NULL,
  `SequencePosition` integer NOT NULL default '1',
  `Status` enum('SendingOK','SendingOKNoReport','SendingError','DeliveryOK',
↪'DeliveryFailed','DeliveryPending','DeliveryUnknown','Error') NOT NULL default
↪'SendingOK',
  `StatusError` integer NOT NULL default '-1',
  `TPMR` integer NOT NULL default '-1',
  `RelativeValidity` integer NOT NULL default '-1',
  `CreatorID` text NOT NULL,
  `StatusCode` integer NOT NULL default '-1',
  PRIMARY KEY (`ID`, `SequencePosition`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;

CREATE INDEX sentitems_date ON sentitems(DeliveryDateTime);
CREATE INDEX sentitems_tpmr ON sentitems(TPMR);
CREATE INDEX sentitems_dest ON sentitems(DestinationNumber);
CREATE INDEX sentitems_sender ON sentitems(SenderID(250));

--
-- Dumping data for table `sentitems`
--
```

Note: You can find the script in `docs/sql/mysql.sql` as well.

SQL script `mysql-legacy.sql` for creating tables in MySQL database:

```
--
-- Database for Gammu SMSD
--
-- In case you get errors about not supported charset, please
```

(continues on next page)

(continued from previous page)

```
-- replace utf8mb4 with utf8.

-- --------------------------------------------------------

--
-- Table structure for table `gammu`
--

CREATE TABLE `gammu` (
  `Version` integer NOT NULL default '0' PRIMARY KEY
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;

--
-- Dumping data for table `gammu`
--

INSERT INTO `gammu` (`Version`) VALUES (17);

-- --------------------------------------------------------

--
-- Table structure for table `inbox`
--

CREATE TABLE `inbox` (
  `UpdatedInDB` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,
  `ReceivingDateTime` timestamp NOT NULL default '0000-00-00 00:00:00',
  `Text` text NOT NULL,
  `SenderNumber` varchar(20) NOT NULL default '',
  `Coding` enum('Default_No_Compression','Unicode_No_Compression','8bit','Default_
↪Compression','Unicode_Compression') NOT NULL default 'Default_No_Compression',
  `UDH` text NOT NULL,
  `SMSCNumber` varchar(20) NOT NULL default '',
  `Class` integer NOT NULL default '-1',
  `TextDecoded` text NOT NULL,
  `ID` integer unsigned NOT NULL auto_increment,
  `RecipientID` text NOT NULL,
  `Processed` enum('false','true') NOT NULL default 'false',
  `Status` integer NOT NULL default '-1',
  PRIMARY KEY `ID` (`ID`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4 AUTO_INCREMENT=1 ;

--
-- Dumping data for table `inbox`
--


-- --------------------------------------------------------

--
-- Table structure for table `outbox`
--
```

(continues on next page)

(continued from previous page)

```
CREATE TABLE `outbox` (
  `UpdatedInDB` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,
  `InsertIntoDB` timestamp NOT NULL default '0000-00-00 00:00:00',
  `SendingDateTime` timestamp NOT NULL default '0000-00-00 00:00:00',
  `SendBefore` time NOT NULL DEFAULT '23:59:59',
  `SendAfter` time NOT NULL DEFAULT '00:00:00',
  `Text` text,
  `DestinationNumber` varchar(20) NOT NULL default '',
  `Coding` enum('Default_No_Compression','Unicode_No_Compression','8bit','Default_
↪Compression','Unicode_Compression') NOT NULL default 'Default_No_Compression',
  `UDH` text,
  `Class` integer default '-1',
  `TextDecoded` text NOT NULL,
  `ID` integer unsigned NOT NULL auto_increment,
  `MultiPart` enum('false','true') default 'false',
  `RelativeValidity` integer default '-1',
  `SenderID` varchar(255),
  `SendingTimeOut` timestamp NULL default '0000-00-00 00:00:00',
  `DeliveryReport` enum('default','yes','no') default 'default',
  `CreatorID` text NOT NULL,
  `Retries` int(3) default 0,
  `Priority` integer default 0,
  `Status` enum('SendingOK','SendingOKNoReport','SendingError','DeliveryOK',
↪'DeliveryFailed','DeliveryPending','DeliveryUnknown','Error','Reserved') NOT NULL␣
↪default 'Reserved',
  `StatusCode` integer NOT NULL default '-1',
  PRIMARY KEY `ID` (`ID`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;

CREATE INDEX outbox_date ON outbox(SendingDateTime, SendingTimeOut);
CREATE INDEX outbox_sender ON outbox(SenderID(250));

--
-- Dumping data for table `outbox`
--


-- --------------------------------------------------------

--
-- Table structure for table `outbox_multipart`
--

CREATE TABLE `outbox_multipart` (
  `Text` text,
  `Coding` enum('Default_No_Compression','Unicode_No_Compression','8bit','Default_
↪Compression','Unicode_Compression') NOT NULL default 'Default_No_Compression',
  `UDH` text,
  `Class` integer default '-1',
  `TextDecoded` text,
  `ID` integer unsigned NOT NULL default '0',
```

(continues on next page)

(continued from previous page)

```
  `SequencePosition` integer NOT NULL default '1',
  `Status` enum('SendingOK','SendingOKNoReport','SendingError','DeliveryOK',
↪'DeliveryFailed','DeliveryPending','DeliveryUnknown','Error','Reserved') NOT NULL␣
↪default 'Reserved',
  `StatusCode` integer NOT NULL default '-1',
  PRIMARY KEY (`ID`, `SequencePosition`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;

--
-- Dumping data for table `outbox_multipart`
--

-- --------------------------------------------------------

--
-- Table structure for table `phones`
--

CREATE TABLE `phones` (
  `ID` text NOT NULL,
  `UpdatedInDB` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,
  `InsertIntoDB` timestamp NOT NULL default '0000-00-00 00:00:00',
  `TimeOut` timestamp NOT NULL default '0000-00-00 00:00:00',
  `Send` enum('yes','no') NOT NULL default 'no',
  `Receive` enum('yes','no') NOT NULL default 'no',
  `IMEI` varchar(35) NOT NULL,
  `IMSI` varchar(35) NOT NULL,
  `NetCode` varchar(10) default 'ERROR',
  `NetName` varchar(35) default 'ERROR',
  `Client` text NOT NULL,
  `Battery` integer NOT NULL DEFAULT -1,
  `Signal` integer NOT NULL DEFAULT -1,
  `Sent` int NOT NULL DEFAULT 0,
  `Received` int NOT NULL DEFAULT 0,
  PRIMARY KEY (`IMEI`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;

--
-- Dumping data for table `phones`
--

-- --------------------------------------------------------

--
-- Table structure for table `sentitems`
--

CREATE TABLE `sentitems` (
  `UpdatedInDB` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,
  `InsertIntoDB` timestamp NOT NULL default '0000-00-00 00:00:00',
  `SendingDateTime` timestamp NOT NULL default '0000-00-00 00:00:00',
  `DeliveryDateTime` timestamp NULL,
```

(continues on next page)

(continued from previous page)

```
  `Text` text NOT NULL,
  `DestinationNumber` varchar(20) NOT NULL default '',
  `Coding` enum('Default_No_Compression','Unicode_No_Compression','8bit','Default_
↪Compression','Unicode_Compression') NOT NULL default 'Default_No_Compression',
  `UDH` text NOT NULL,
  `SMSCNumber` varchar(20) NOT NULL default '',
  `Class` integer NOT NULL default '-1',
  `TextDecoded` text NOT NULL,
  `ID` integer unsigned NOT NULL default '0',
  `SenderID` varchar(255) NOT NULL,
  `SequencePosition` integer NOT NULL default '1',
  `Status` enum('SendingOK','SendingOKNoReport','SendingError','DeliveryOK',
↪'DeliveryFailed','DeliveryPending','DeliveryUnknown','Error') NOT NULL default
↪'SendingOK',
  `StatusError` integer NOT NULL default '-1',
  `TPMR` integer NOT NULL default '-1',
  `RelativeValidity` integer NOT NULL default '-1',
  `CreatorID` text NOT NULL,
  `StatusCode` integer NOT NULL default '-1',
  PRIMARY KEY (`ID`, `SequencePosition`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;

CREATE INDEX sentitems_date ON sentitems(DeliveryDateTime);
CREATE INDEX sentitems_tpmr ON sentitems(TPMR);
CREATE INDEX sentitems_dest ON sentitems(DestinationNumber);
CREATE INDEX sentitems_sender ON sentitems(SenderID(250));

--
-- Dumping data for table `sentitems`
--


--
-- Triggers for setting default timestamps
--

DELIMITER //

CREATE TRIGGER inbox_timestamp BEFORE INSERT ON inbox
FOR EACH ROW
BEGIN
    IF NEW.ReceivingDateTime = '0000-00-00 00:00:00' THEN
        SET NEW.ReceivingDateTime = CURRENT_TIMESTAMP();
    END IF;
END;//

CREATE TRIGGER outbox_timestamp BEFORE INSERT ON outbox
FOR EACH ROW
BEGIN
    IF NEW.InsertIntoDB = '0000-00-00 00:00:00' THEN
        SET NEW.InsertIntoDB = CURRENT_TIMESTAMP();
    END IF;
```

(continues on next page)

(continued from previous page)

```
    IF NEW.SendingDateTime = '0000-00-00 00:00:00' THEN
        SET NEW.SendingDateTime = CURRENT_TIMESTAMP();
    END IF;
    IF NEW.SendingTimeOut = '0000-00-00 00:00:00' THEN
        SET NEW.SendingTimeOut = CURRENT_TIMESTAMP();
    END IF;
END;//

CREATE TRIGGER phones_timestamp BEFORE INSERT ON phones
FOR EACH ROW
BEGIN
    IF NEW.InsertIntoDB = '0000-00-00 00:00:00' THEN
        SET NEW.InsertIntoDB = CURRENT_TIMESTAMP();
    END IF;
    IF NEW.TimeOut = '0000-00-00 00:00:00' THEN
        SET NEW.TimeOut = CURRENT_TIMESTAMP();
    END IF;
END;//

CREATE TRIGGER sentitems_timestamp BEFORE INSERT ON sentitems
FOR EACH ROW
BEGIN
    IF NEW.InsertIntoDB = '0000-00-00 00:00:00' THEN
        SET NEW.InsertIntoDB = CURRENT_TIMESTAMP();
    END IF;
    IF NEW.SendingDateTime = '0000-00-00 00:00:00' THEN
        SET NEW.SendingDateTime = CURRENT_TIMESTAMP();
    END IF;
END;//

DELIMITER ;
```

Note: You can find the script in `docs/sql/mysql-legacy.sql` as well.

6.3.5 Upgrading tables

The easiest way to upgrade database structure is to backup old one and start with creating new one based on example above.

For upgrading existing database, you can use changes described in *History of database structure* and then manually update `Version` field in `gammu` table.

6.4 PostgreSQL Backend

6.4.1 Description

PGSQL backend stores all data in a PostgreSQL database server, which parameters are defined by configuration (see *SMSD Configuration File* for description of configuration options).

For tables description see *SMSD Database Structure*.

This backend is based on *SQL Service*.

6.4.2 Configuration

Before running *gammu-smsd* you need to create necessary tables in the database, which is described below.

The configuration file then can look like:

```
[smsd]
service = sql
driver = native_pgsql
host = localhost
```

See also:

SMSD Configuration File

6.4.3 Creating tables for PostgreSQL

SQL script for creating tables in PostgreSQL database:

```
--
-- Database: "smsd"
--
-- CREATE USER "smsd" WITH NOCREATEDB NOCREATEUSER;
-- CREATE DATABASE "smsd" WITH OWNER = "smsd" ENCODING = 'UTF8';
-- \connect "smsd" "smsd"
-- COMMENT ON DATABASE "smsd" IS 'Gammu SMSD Database';

-- ------------------------------------------------------

--
-- Function declaration for updating timestamps
--
CREATE EXTENSION IF NOT EXISTS plpgsql;
CREATE OR REPLACE FUNCTION update_timestamp() RETURNS trigger AS $update_timestamp$
  BEGIN
    NEW."UpdatedInDB" := LOCALTIMESTAMP(0);
    RETURN NEW;
  END;
$update_timestamp$ LANGUAGE plpgsql;

-- ------------------------------------------------------

```

(continues on next page)

(continued from previous page)

```
--
-- Sequence declarations for tables' primary keys
--

--CREATE SEQUENCE inbox_ID_seq;

--CREATE SEQUENCE outbox_ID_seq;

--CREATE SEQUENCE outbox_multipart_ID_seq;

--CREATE SEQUENCE sentitems_ID_seq;

-- -----------------------------------------------------

--
-- Index declarations for tables' primary keys
--

--CREATE UNIQUE INDEX inbox_pkey ON inbox USING btree ("ID");

--CREATE UNIQUE INDEX outbox_pkey ON outbox USING btree ("ID");

--CREATE UNIQUE INDEX outbox_multipart_pkey ON outbox_multipart USING btree ("ID");

--CREATE UNIQUE INDEX sentitems_pkey ON sentitems USING btree ("ID");

-- -----------------------------------------------------

--
-- Table structure for table "gammu"
--

CREATE TABLE gammu (
  "Version" smallint NOT NULL DEFAULT '0' PRIMARY KEY
);

--
-- Dumping data for table "gammu"
--

INSERT INTO gammu ("Version") VALUES (17);

-- -----------------------------------------------------

--
-- Table structure for table "inbox"
--

CREATE TABLE inbox (
  "UpdatedInDB" timestamp(0) WITHOUT time zone NOT NULL DEFAULT LOCALTIMESTAMP(0),
  "ReceivingDateTime" timestamp(0) WITHOUT time zone NOT NULL DEFAULT LOCALTIMESTAMP(0),
  "Text" text NOT NULL,
```

(continues on next page)

(continued from previous page)

```
  "SenderNumber" varchar(20) NOT NULL DEFAULT '',
  "Coding" varchar(255) NOT NULL DEFAULT 'Default_No_Compression',
  "UDH" text NOT NULL,
  "SMSCNumber" varchar(20) NOT NULL DEFAULT '',
  "Class" integer NOT NULL DEFAULT '-1',
  "TextDecoded" text NOT NULL DEFAULT '',
  "ID" serial PRIMARY KEY,
  "RecipientID" text NOT NULL,
  "Processed" boolean NOT NULL DEFAULT 'false',
  "Status" integer NOT NULL DEFAULT '-1',
  CHECK ("Coding" IN
  ('Default_No_Compression','Unicode_No_Compression','8bit','Default_Compression',
↪'Unicode_Compression'))
);

--
-- Dumping data for table "inbox"
--

-- --------------------------------------------------------

--
-- Create trigger for table "inbox"
--

CREATE TRIGGER update_timestamp BEFORE UPDATE ON inbox FOR EACH ROW EXECUTE PROCEDURE␣
↪update_timestamp();

-- --------------------------------------------------------

--
-- Table structure for table "outbox"
--

CREATE TABLE outbox (
  "UpdatedInDB" timestamp(0) WITHOUT time zone NOT NULL DEFAULT LOCALTIMESTAMP(0),
  "InsertIntoDB" timestamp(0) WITHOUT time zone NOT NULL DEFAULT LOCALTIMESTAMP(0),
  "SendingDateTime" timestamp NOT NULL DEFAULT LOCALTIMESTAMP(0),
  "SendBefore" time NOT NULL DEFAULT '23:59:59',
  "SendAfter" time NOT NULL DEFAULT '00:00:00',
  "Text" text,
  "DestinationNumber" varchar(20) NOT NULL DEFAULT '',
  "Coding" varchar(255) NOT NULL DEFAULT 'Default_No_Compression',
  "UDH" text,
  "Class" integer DEFAULT '-1',
  "TextDecoded" text NOT NULL DEFAULT '',
  "ID" serial PRIMARY KEY,
  "MultiPart" boolean NOT NULL DEFAULT 'false',
  "RelativeValidity" integer DEFAULT '-1',
  "SenderID" varchar(255),
  "SendingTimeOut" timestamp(0) WITHOUT time zone NOT NULL DEFAULT LOCALTIMESTAMP(0),
  "DeliveryReport" varchar(10) DEFAULT 'default',
```

(continues on next page)

(continued from previous page)

```
  "CreatorID" text NOT NULL,
  "Retries" integer DEFAULT '0',
  "Priority" integer DEFAULT '0',
  "Status" varchar(255) NOT NULL DEFAULT 'Reserved',
  "StatusCode" integer NOT NULL DEFAULT '-1',
  CHECK ("Coding" IN
  ('Default_No_Compression','Unicode_No_Compression','8bit','Default_Compression',
↪'Unicode_Compression')),
  CHECK ("DeliveryReport" IN ('default','yes','no')),
  CHECK ("Status" IN
  ('SendingOK','SendingOKNoReport','SendingError','DeliveryOK','DeliveryFailed',
↪'DeliveryPending',
  'DeliveryUnknown','Error','Reserved'))
);

CREATE INDEX outbox_date ON outbox("SendingDateTime", "SendingTimeOut");
CREATE INDEX outbox_sender ON outbox("SenderID");

--
-- Dumping data for table "outbox"
--

-- --------------------------------------------------------

--
-- Create trigger for table "outbox"
--

CREATE TRIGGER update_timestamp BEFORE UPDATE ON outbox FOR EACH ROW EXECUTE PROCEDURE␣
↪update_timestamp();

-- --------------------------------------------------------

--
-- Table structure for table "outbox_multipart"
--

CREATE TABLE outbox_multipart (
  "Text" text,
  "Coding" varchar(255) NOT NULL DEFAULT 'Default_No_Compression',
  "UDH" text,
  "Class" integer DEFAULT '-1',
  "TextDecoded" text DEFAULT NULL,
  "ID" serial,
  "SequencePosition" integer NOT NULL DEFAULT '1',
  "Status" varchar(255) NOT NULL DEFAULT 'Reserved',
  "StatusCode" integer NOT NULL DEFAULT '-1',
  PRIMARY KEY ("ID", "SequencePosition"),
  CHECK ("Coding" IN
  ('Default_No_Compression','Unicode_No_Compression','8bit','Default_Compression',
↪'Unicode_Compression')),
  CHECK ("Status" IN
```

(continues on next page)

(continued from previous page)

```
  ('SendingOK','SendingOKNoReport','SendingError','DeliveryOK','DeliveryFailed',
↪'DeliveryPending',
  'DeliveryUnknown','Error','Reserved'))
);

--
-- Dumping data for table "outbox_multipart"
--


-- ------------------------------------------------------------

--
-- Table structure for table "phones"
--

CREATE TABLE phones (
  "ID" text NOT NULL,
  "UpdatedInDB" timestamp(0) WITHOUT time zone NOT NULL DEFAULT LOCALTIMESTAMP(0),
  "InsertIntoDB" timestamp(0) WITHOUT time zone NOT NULL DEFAULT LOCALTIMESTAMP(0),
  "TimeOut" timestamp(0) WITHOUT time zone NOT NULL DEFAULT LOCALTIMESTAMP(0),
  "Send" boolean NOT NULL DEFAULT 'no',
  "Receive" boolean NOT NULL DEFAULT 'no',
  "IMEI" varchar(35) PRIMARY KEY NOT NULL,
  "IMSI" varchar(35) NOT NULL,
  "NetCode" varchar(10) DEFAULT 'ERROR',
  "NetName" varchar(35) DEFAULT 'ERROR',
  "Client" text NOT NULL,
  "Battery" integer NOT NULL DEFAULT -1,
  "Signal" integer NOT NULL DEFAULT -1,
  "Sent" integer NOT NULL DEFAULT 0,
  "Received" integer NOT NULL DEFAULT 0
);

--
-- Dumping data for table "phones"
--

-- ------------------------------------------------------------

--
-- Create trigger for table "phones"
--

CREATE TRIGGER update_timestamp BEFORE UPDATE ON phones FOR EACH ROW EXECUTE PROCEDURE␣
↪update_timestamp();

-- ------------------------------------------------------------

--
-- Table structure for table "sentitems"
--
```

(continues on next page)

(continued from previous page)

```
CREATE TABLE sentitems (
  "UpdatedInDB" timestamp(0) WITHOUT time zone NOT NULL DEFAULT LOCALTIMESTAMP(0),
  "InsertIntoDB" timestamp(0) WITHOUT time zone NOT NULL DEFAULT LOCALTIMESTAMP(0),
  "SendingDateTime" timestamp(0) WITHOUT time zone NOT NULL DEFAULT LOCALTIMESTAMP(0),
  "DeliveryDateTime" timestamp(0) WITHOUT time zone NULL,
  "Text" text NOT NULL,
  "DestinationNumber" varchar(20) NOT NULL DEFAULT '',
  "Coding" varchar(255) NOT NULL DEFAULT 'Default_No_Compression',
  "UDH" text NOT NULL,
  "SMSCNumber" varchar(20) NOT NULL DEFAULT '',
  "Class" integer NOT NULL DEFAULT '-1',
  "TextDecoded" text NOT NULL DEFAULT '',
  "ID" serial,
  "SenderID" varchar(255) NOT NULL,
  "SequencePosition" integer NOT NULL DEFAULT '1',
  "Status" varchar(255) NOT NULL DEFAULT 'SendingOK',
  "StatusError" integer NOT NULL DEFAULT '-1',
  "TPMR" integer NOT NULL DEFAULT '-1',
  "RelativeValidity" integer NOT NULL DEFAULT '-1',
  "CreatorID" text NOT NULL,
  "StatusCode" integer NOT NULL DEFAULT '-1',
  CHECK ("Status" IN
  ('SendingOK','SendingOKNoReport','SendingError','DeliveryOK','DeliveryFailed',
↪'DeliveryPending',
  'DeliveryUnknown','Error')),
  CHECK ("Coding" IN
  ('Default_No_Compression','Unicode_No_Compression','8bit','Default_Compression',
↪'Unicode_Compression')),
  PRIMARY KEY ("ID", "SequencePosition")
);

CREATE INDEX sentitems_date ON sentitems("DeliveryDateTime");
CREATE INDEX sentitems_tpmr ON sentitems("TPMR");
CREATE INDEX sentitems_dest ON sentitems("DestinationNumber");
CREATE INDEX sentitems_sender ON sentitems("SenderID");

--
-- Dumping data for table "sentitems"
--

-- ------------------------------------------------------

--
-- Create trigger for table "sentitems"
--

CREATE TRIGGER update_timestamp BEFORE UPDATE ON sentitems FOR EACH ROW EXECUTE␣
↪PROCEDURE update_timestamp();
```

Note: You can find the script in `docs/sql/pgsql.sql` as well.

6.4.4 Upgrading tables

The easiest way to upgrade database structure is to backup old one and start with creating new one based on example above.

For upgrading existing database, you can use changes described in *History of database structure* and then manually update `Version` field in `gammu` table.

6.5 DBI Backend

6.5.1 Description

DBI backend stores all data in any database supported by libdbi, which parameters are defined by configuration (see *SMSD Configuration File* for description of configuration options).

For tables description see *SMSD Database Structure*.

This backend is based on *SQL Service*.

Note: The DBI driver is currently not supported on Windows because libdbi library does not support this platform.

6.5.2 Configuration

Before running *gammu-smsd* you need to create necessary tables in the database. You can use examples given in database specific backends parts of this manual to do that.

The configuration file then can look like:

```
[smsd]
service = sql
driver = DBI_DRIVER
host = localhost
```

See also:

SMSD Configuration File

6.5.3 Supported drivers

For complete list of drivers for libdbi see libdbi-drivers project. The drivers for example include:

- `sqlite3` - for SQLite 3
- `mysql` - for MySQL
- `pgsql` - for PostgeSQL
- `freetds` - for MS SQL Server or Sybase

6.5.4 Creating tables for SQLite

SQL script for creating tables in SQLite database:

```
CREATE TABLE gammu (
  Version INTEGER NOT NULL DEFAULT '0' PRIMARY KEY
);

INSERT INTO gammu (Version) VALUES (17);

CREATE TABLE inbox (
  UpdatedInDB NUMERIC NOT NULL DEFAULT (datetime('now')),
  ReceivingDateTime NUMERIC NOT NULL DEFAULT (datetime('now')),
  Text TEXT NOT NULL,
  SenderNumber TEXT NOT NULL DEFAULT '',
  Coding TEXT NOT NULL DEFAULT 'Default_No_Compression',
  UDH TEXT NOT NULL,
  SMSCNumber TEXT NOT NULL DEFAULT '',
  Class INTEGER NOT NULL DEFAULT '-1',
  TextDecoded TEXT NOT NULL DEFAULT '',
  ID INTEGER PRIMARY KEY AUTOINCREMENT,
  RecipientID TEXT NOT NULL,
  Processed TEXT NOT NULL DEFAULT 'false',
  Status INTEGER NOT NULL DEFAULT '-1',
  CHECK (Coding IN
  ('Default_No_Compression','Unicode_No_Compression','8bit','Default_Compression',
↪'Unicode_Compression'))
);

CREATE TRIGGER update_inbox_time UPDATE ON inbox
  BEGIN
    UPDATE inbox SET UpdatedInDB = datetime('now') WHERE ID = old.ID;
  END;

CREATE TABLE outbox (
  UpdatedInDB NUMERIC NOT NULL DEFAULT (datetime('now')),
  InsertIntoDB NUMERIC NOT NULL DEFAULT (datetime('now')),
  SendingDateTime NUMERIC NOT NULL DEFAULT (datetime('now')),
  SendBefore time NOT NULL DEFAULT '23:59:59',
  SendAfter time NOT NULL DEFAULT '00:00:00',
  Text TEXT,
  DestinationNumber TEXT NOT NULL DEFAULT '',
  Coding TEXT NOT NULL DEFAULT 'Default_No_Compression',
  UDH TEXT,
  Class INTEGER DEFAULT '-1',
  TextDecoded TEXT NOT NULL DEFAULT '',
  ID INTEGER PRIMARY KEY AUTOINCREMENT,
  MultiPart TEXT NOT NULL DEFAULT 'false',
  RelativeValidity INTEGER DEFAULT '-1',
  SenderID TEXT,
  SendingTimeOut NUMERIC NOT NULL DEFAULT (datetime('now')),
  DeliveryReport TEXT DEFAULT 'default',
  CreatorID TEXT NOT NULL,
```

(continues on next page)

(continued from previous page)

```
  Retries INTEGER DEFAULT '0',
  Priority INTEGER DEFAULT '0',
  Status TEXT NOT NULL DEFAULT 'Reserved',
  StatusCode INTEGER NOT NULL DEFAULT '-1',
  CHECK (Coding IN
  ('Default_No_Compression','Unicode_No_Compression','8bit','Default_Compression',
↪'Unicode_Compression')),
  CHECK (DeliveryReport IN ('default','yes','no')),
  CHECK (Status IN
  ('SendingOK','SendingOKNoReport','SendingError','DeliveryOK','DeliveryFailed',
↪'DeliveryPending',
  'DeliveryUnknown','Error','Reserved'))
);

CREATE INDEX outbox_date ON outbox(SendingDateTime, SendingTimeOut);
CREATE INDEX outbox_sender ON outbox(SenderID);

CREATE TRIGGER update_outbox_time UPDATE ON outbox
  BEGIN
    UPDATE outbox SET UpdatedInDB = datetime('now') WHERE ID = old.ID;
  END;

CREATE TABLE outbox_multipart (
  Text TEXT,
  Coding TEXT NOT NULL DEFAULT 'Default_No_Compression',
  UDH TEXT,
  Class INTEGER DEFAULT '-1',
  TextDecoded TEXT DEFAULT NULL,
  ID INTEGER,
  SequencePosition INTEGER NOT NULL DEFAULT '1',
  Status TEXT NOT NULL DEFAULT 'Reserved',
  StatusCode INTEGER NOT NULL DEFAULT '-1',
  CHECK (Coding IN
  ('Default_No_Compression','Unicode_No_Compression','8bit','Default_Compression',
↪'Unicode_Compression')),
  CHECK (Status IN
  ('SendingOK','SendingOKNoReport','SendingError','DeliveryOK','DeliveryFailed',
↪'DeliveryPending',
  'DeliveryUnknown','Error','Reserved')),
 PRIMARY KEY (ID, SequencePosition)
);

CREATE TABLE phones (
  ID TEXT NOT NULL,
  UpdatedInDB NUMERIC NOT NULL DEFAULT (datetime('now')),
  InsertIntoDB NUMERIC NOT NULL DEFAULT (datetime('now')),
  TimeOut NUMERIC NOT NULL DEFAULT (datetime('now')),
  Send TEXT NOT NULL DEFAULT 'no',
  Receive TEXT NOT NULL DEFAULT 'no',
  IMEI TEXT PRIMARY KEY NOT NULL,
  IMSI TEXT NOT NULL,
  NetCode TEXT DEFAULT 'ERROR',
```

(continues on next page)

(continued from previous page)

```
  NetName TEXT DEFAULT 'ERROR',
  Client TEXT NOT NULL,
  Battery INTEGER NOT NULL DEFAULT -1,
  Signal INTEGER NOT NULL DEFAULT -1,
  Sent INTEGER NOT NULL DEFAULT 0,
  Received INTEGER NOT NULL DEFAULT 0
);

CREATE TRIGGER update_phones_time UPDATE ON phones
  BEGIN
    UPDATE phones SET UpdatedInDB = datetime('now') WHERE IMEI = old.IMEI;
  END;

CREATE TABLE sentitems (
  UpdatedInDB NUMERIC NOT NULL DEFAULT (datetime('now')),
  InsertIntoDB NUMERIC NOT NULL DEFAULT (datetime('now')),
  SendingDateTime NUMERIC NOT NULL DEFAULT (datetime('now')),
  DeliveryDateTime NUMERIC NULL,
  Text TEXT NOT NULL,
  DestinationNumber TEXT NOT NULL DEFAULT '',
  Coding TEXT NOT NULL DEFAULT 'Default_No_Compression',
  UDH TEXT NOT NULL,
  SMSCNumber TEXT NOT NULL DEFAULT '',
  Class INTEGER NOT NULL DEFAULT '-1',
  TextDecoded TEXT NOT NULL DEFAULT '',
  ID INTEGER,
  SenderID TEXT NOT NULL,
  SequencePosition INTEGER NOT NULL DEFAULT '1',
  Status TEXT NOT NULL DEFAULT 'SendingOK',
  StatusError INTEGER NOT NULL DEFAULT '-1',
  TPMR INTEGER NOT NULL DEFAULT '-1',
  RelativeValidity INTEGER NOT NULL DEFAULT '-1',
  CreatorID TEXT NOT NULL,
  StatusCode INTEGER NOT NULL DEFAULT '-1',
  CHECK (Status IN
  ('SendingOK','SendingOKNoReport','SendingError','DeliveryOK','DeliveryFailed',
↪'DeliveryPending',
  'DeliveryUnknown','Error')),
  CHECK (Coding IN
  ('Default_No_Compression','Unicode_No_Compression','8bit','Default_Compression',
↪'Unicode_Compression')) ,
  PRIMARY KEY (ID, SequencePosition)
);

CREATE INDEX sentitems_date ON sentitems(DeliveryDateTime);
CREATE INDEX sentitems_tpmr ON sentitems(TPMR);
CREATE INDEX sentitems_dest ON sentitems(DestinationNumber);
CREATE INDEX sentitems_sender ON sentitems(SenderID);

CREATE TRIGGER update_sentitems_time UPDATE ON sentitems
  BEGIN
    UPDATE sentitems SET UpdatedInDB = datetime('now') WHERE ID = old.ID;
```

(continues on next page)

(continued from previous page)

```
END;
```

Note: You can find the script in `docs/sql/sqlite.sql` as well. There are also scripts for other databases in same folder.

6.5.5 Upgrading tables

The easiest way to upgrade database structure is to backup old one and start with creating new one based on example above.

For upgrading existing database, you can use changes described in *History of database structure* and then manually update `Version` field in `gammu` table.

6.6 ODBC Backend

6.6.1 Description

New in version 1.29.92.

ODBC backend stores all data in any database supported by ODBC, which parameters are defined by configuration (see *SMSD Configuration File* for description of configuration options).

For tables description see *SMSD Database Structure*.

This backend is based on *SQL Service*.

6.6.2 Supported drivers

On Microsoft Windows, Gammu uses native ODBC, on other platforms, unixODBC can be used.

6.6.3 Limitations

Due to limits of the ODBC interface, Gammu can not reliably detect which SQL engine it is connected to.

In most cases this can be solved by setting *SQL* setting to correct dialect.

If that fails, you can also tweak the SQL queries to work in used SQL server, see *SQL Queries* for more details. Still you should set *SQL* to closest matching SQL dialect.

6.6.4 Configuration

Before running *gammu-smsd* you need to create necessary tables in the database. You can use examples given in database specific backends parts of this manual to do that.

You specify data source name (DSN) as `Host` in *SMSD Configuration File*. The data source is configured depending on your platform.

Note: Please remember that SMSD might be running in different context than your user (separate account on Linux or as as service on Windows), so the ODBC DSN needs to be configured as system wide in this case (system DSN on Windows or in global configuration on Linux).

On Microsoft Windows, you can find instructions on Microsoft website: https://support.microsoft.com/kb/305599

For unixODBC this is documented in the user manual: http://www.unixodbc.org/doc/UserManual/

6.6.5 Creating tables

Prior to starting SMSD you have to create tables it will use. Gammu ships SQL scripts for several databases to do that:

- *Creating tables for MySQL*
- *Creating tables for PostgreSQL*
- *Creating tables for SQLite*

6.6.6 Example

Example configuration:

```
[smsd]
service = sql
driver = odbc
host = dsn_of_your_database
sql = sql_variant_to_use
user = username
password = password
```

See also:

SMSD Configuration File

6.7 Null Backend

6.7.1 Description

NULL backend does not store data at all. It could be useful in case you don't want to store messages at all and you want to process then in `RunOnReceive` handler.

6.7.2 Configuration

The configuration file then can look like:

```
[smsd]
Service = null
RunOnReceive = /usr/local/bin/process-sms
```

See also:

SMSD Configuration File

6.8 SMSD Database Structure

The backends themselves are described in their sections, this document describes general database structure and required tables.

More SMS daemons can share single database. If you do not specify PhoneID in their configuration, all are treated equally and you have no guarantee which one sends outgoing message. If you configure PhoneID and use it when inserting message to the `outbox` table (*gammu-smsd-inject* does this), each SMS daemon will have separate outbox queue. See also *Multiple modems*.

Note: SQL scripts to create all needed tables for most databases are included in Gammu documentation docs/sql.

6.8.1 Receiving of messages

Received messages are stored in *inbox* table.

6.8.2 Transmitting of messages

Transmitted messages are read from table *outbox* and possible subsequent parts of the same message from *outbox_multipart*.

6.8.3 Description of tables

gammu

Table holding single field `Version` - version of a database schema. See *History of database structure* for details what has changed.

inbox

Table where received messages will be stored.

Fields description:

UpdatedInDB (timestamp)
when somebody (daemon, user, etc.) updated it

ReceivingDateTime (timestamp)
when SMS was received

Text (text)
encoded SMS text (for all SMS)

SenderNumber (varchar(20))
decoded SMS sender number

Coding (enum('Default_No_Compression', 'Unicode_No_Compression', '8bit', 'Default_Compression', 'Unicode_Compression'))
SMS text coding

UDH (text)
encoded User Data Header text

SMSCNumber (varchar(20))
decoded SMSC number

Class (integer)
SMS class or -1 (0 is flash SMS, 1 is normal one, 127 is USSD)

TextDecoded (varchar(160))
decoded SMS text (for Default Alphabet/Unicode SMS)

ID (integer unsigned)
SMS identificator (for using with external applications)

RecipientID (text)
which Gammu daemon has added it

Processed (enum('false', 'true'))
you can use for marking, whether SMS was processed or not

Status (integer)
Status of incoming message. Currently only used for `Class` 127 (USSD) messages with following meaning:

`1`
Unknown status.

`2`
No action is needed, maybe network initiated USSD.

`3`
Reply is expected.

`4`
USSD dialog terminated.

`5`
Another client replied.

`6`
Operation not supported.

7
: Network timeout.

New in version 1.38.5.

outbox

Messages enqueued for sending should be placed in this table. If message is multipart, subsequent parts are stored in table *outbox_multipart*.

Fields description:

UpdatedInDB (timestamp)
: when somebody (daemon, user, etc.) updated it

InsertIntoDB (timestamp)
: when message was inserted into database

SendingDateTime (timestamp)
: set it to some value, when want to force sending after some planned time

SendBefore (time)
: Send message before specified time, can be used to limit messages from being sent in night. Default value is 23:59:59

  New in version 1.29.90.

SendAfter (time)
: Send message after specified time, can be used to limit messages from being sent in night. Default value is 00:00:00

  New in version 1.29.90.

Text (text)
: SMS text encoded using hex values in proper coding. If you want to use TextDecoded field, keep this NULL (or empty).

DestinationNumber (varchar(20))
: recipient number

Coding (enum('Default_No_Compression', 'Unicode_No_Compression', '8bit', 'Default_Compression', 'Unicode_Compression'))
: SMS text coding

UDH (text)
: User Data Header encoded using hex values which will be used for constructing the message. Without this, message will be sent as plain text.

Class (integer)
: SMS class or -1 (0 is normal SMS, 1 is flash one, 127 is USSD)

TextDecoded (varchar(160))
: SMS text in "human readable" form

ID (integer unsigned)
: SMS/SMS sequence ID

  Please note that this number has to be unique also for sentitems table, so reusing message IDs might not be a good idea.

MultiPart (enum('false','true'))
: info, whether there are more SMS from this sequence in outbox_multipart

RelativeValidity (integer)
SMS relative validity like encoded using GSM specs

SenderID (text)
which SMSD instance should send this one sequence, see *PhoneID* and *Multiple modems*. If blank, first SMSD who sees this message first will process it.

SendingTimeOut (timestamp)
used by SMSD instance for own targets

DeliveryReport (enum('default','yes','no'))
when default is used, Delivery Report is used or not according to SMSD instance settings; yes forces Delivery Report.

CreatorID (text)
identification of program created the message

Retries (integer)
number of attempted retries when sending this message

Priority (integer)
priority of message, messages with higher priority are processed first

Status (enum('SendingOK', 'SendingOKNoReport', 'SendingError', 'DeliveryOK', 'DeliveryFailed', 'DeliveryPending', 'DeliveryUnknown', 'Error', 'Reserved'))
Status of message sending. SendingError means that phone failed to send the message, Error indicates some other error while processing message.

SendingOK
Message has been sent, waiting for delivery report.

SendingOKNoReport
Message has been sent without asking for delivery report.

SendingError
Sending has failed.

DeliveryOK
Delivery report arrived and reported success.

DeliveryFailed
Delivery report arrived and reports failure.

DeliveryPending
Delivery report announced pending deliver.

DeliveryUnknown
Delivery report reported unknown status.

Error
Some other error happened during sending (usually bug in SMSD).

Reserved
Initial value, meaning the status has not been set.

New in version 1.38.5.

StatusCode (integer)
GSM status code

New in version 1.38.5.

outbox_multipart

Data for outgoing multipart messages.

Fields description:

ID (integer unsigned)
the same meaning as values in outbox table

Text (text)
the same meaning as values in outbox table

Coding (enum('Default_No_Compression', 'Unicode_No_Compression', '8bit', 'Default_Compression', 'Unicode_Compression'))
the same meaning as values in outbox table

UDH (text)
the same meaning as values in outbox table

Class (integer)
the same meaning as values in outbox table

TextDecoded (varchar(160))
the same meaning as values in outbox table

ID (integer unsigned)
the same meaning as values in outbox table

SequencePosition (integer)
info, what is SMS number in SMS sequence (start at 2, first part is in *outbox* table).

Status (enum('SendingOK', 'SendingOKNoReport', 'SendingError', 'DeliveryOK', 'DeliveryFailed', 'DeliveryPending', 'DeliveryUnknown', 'Error', 'Reserved'))
Status of message sending. SendingError means that phone failed to send the message, Error indicates some other error while processing message.

`SendingOK`
Message has been sent, waiting for delivery report.

`SendingOKNoReport`
Message has been sent without asking for delivery report.

`SendingError`
Sending has failed.

`DeliveryOK`
Delivery report arrived and reported success.

`DeliveryFailed`
Delivery report arrived and reports failure.

`DeliveryPending`
Delivery report announced pending deliver.

`DeliveryUnknown`
Delivery report reported unknown status.

`Error`
Some other error happened during sending (usually bug in SMSD).

`Reserved`
Initial value, meaning the status has not been set.

New in version 1.38.5.

StatusCode (integer)
GSM status code

New in version 1.38.5.

phones

Information about connected phones. This table is periodically refreshed and you can get information such as battery or signal level from here.

Fields description:

ID (text)
PhoneID value

UpdatedInDB (timestamp)
when this record has been updated

InsertIntoDB (timestamp)
when this record has been created (when phone has been connected)

TimeOut (timestamp)
when this record expires

Send (boolean)
indicates whether SMSD is sending messages, depends on configuration directive *Send*

Receive (boolean)
indicates whether SMSD is receiving messages, depends on configuration directive *Receive*

IMEI (text)
IMEI of phone

IMSI (text)
SIM IMSI

Client (text)
client name, usually string Gammu with version

Battery (integer)
battery level in percent (or -1 if unknown)

Signal (integer)
signal level in percent (or -1 if unknown)

Sent (integer)
Number of sent SMS messages (SMSD does not reset this counter, so it might overflow).

Received (integer)
Number of received SMS messages (SMSD does not reset this counter, so it might overflow).

sentitems

Log of sent messages (and unsent ones with error code). Also if delivery reports are enabled, message state is updated after receiving delivery report.

Fields description:

UpdatedInDB (timestamp)
when somebody (daemon, user, etc.) updated it

InsertIntoDB (timestamp)
when message was inserted into database

SendingDateTime (timestamp)
when message has been sent

DeliveryDateTime (timestamp)
Time of receiving delivery report (if it has been enabled).

Status (enum('SendingOK', 'SendingOKNoReport', 'SendingError', 'DeliveryOK', 'DeliveryFailed', 'DeliveryPending', 'DeliveryUnknown', 'Error'))
Status of message sending. SendingError means that phone failed to send the message, Error indicates some other error while processing message.

SendingOK
Message has been sent, waiting for delivery report.

SendingOKNoReport
Message has been sent without asking for delivery report.

SendingError
Sending has failed.

DeliveryOK
Delivery report arrived and reported success.

DeliveryFailed
Delivery report arrived and reports failure.

DeliveryPending
Delivery report announced pending deliver.

DeliveryUnknown
Delivery report reported unknown status.

Error
Some other error happened during sending (usually bug in SMSD).

StatusError (integer)
Status of delivery from delivery report message, codes are defined in GSM specification 03.40 section 9.2.3.15 (TP-Status).

Text (text)
SMS text encoded using hex values

DestinationNumber (varchar(20))
decoded destination number for SMS

Coding (enum('Default_No_Compression', 'Unicode_No_Compression', '8bit', 'Default_Compression', 'Unicode_Compression'))
SMS text coding

UDH (text)
User Data Header encoded using hex values

SMSCNumber (varchar(20))
decoded number of SMSC, which sent SMS

Class (integer)
SMS class or -1 (0 is normal SMS, 1 is flash one, 127 is USSD)

TextDecoded (varchar(160))
SMS text in "human readable" form

ID (integer unsigned)
SMS ID

SenderID (text)
which SMSD instance sent this one sequence, see *PhoneID*

SequencePosition (integer)
SMS number in SMS sequence

TPMR (integer)
Message Reference like in GSM specs

RelativeValidity (integer)
SMS relative validity like encoded using GSM specs

CreatorID (text)
copied from CreatorID from outbox table

StatusCode (integer)
GSM status code

New in version 1.38.5.

6.8.4 History of database structure

Note: Testing versions (see versioning) do not have to keep same table structure as final releases. Below mentioned versions are for informational purposes only, you should always use stable versions in production environment.

History of schema versions:

17

- Added `Status` field to *outbox* and *outbox_multipart*.
- Added `StatusCode` field to *sentitems*, *outbox* and *outbox_multipart*.
- Added `Status` field to *inbox*.

Changed in version 1.38.5.

16

- Removed unused `daemons`, `pbk` and `pbk_groups` tables.
- Added primary key to the `gammu` table.
- Added `Priority` field to the *outbox*.
- Added `IMSI` field to the *phones*.

Changed in version 1.37.90.

15

Added `Retries` field to the *outbox*.

Changed in version 1.36.7.

14

Added `NetCode` and `NetName` fields.

Changed in version 1.34.0.

13

Added `SendBefore` and `SendAfter` fields.

Changed in version 1.29.90.

Also PostgreSQL fields are now case sensitive (same as other backends).

Changed in version 1.29.93.

12

the changes only affect MySQL structure changing default values for timestamps from `0000-00-00 00:00:00` to `CURRENT_TIMESTAMP()` by using triggers, to update to this version, just execute triggers definition at the end of SQL file.

Changed in version 1.28.94.

11

all fields for storing message text are no longer limited to 160 chars, but are arbitrary length text fields.

Changed in version 1.25.92.

10

`DeliveryDateTime` is now NULL when message is not delivered, added several indexes

Changed in version 1.22.95.

9

added sent/received counters to phones table

Changed in version 1.22.93.

8

Signal and battery state are now stored in database.

Changed in version 1.20.94.

7

Added `CreatorID` to several tables.

Changed in version 1.07.00.

6

Many fields in outbox can now be NULL.

Changed in version 1.06.00.

5

Introduced daemons table and various other changes.

Changed in version 1.03.00.

3

Introduced phones table and various other changes.

Changed in version 0.98.0.

6.8.5 Examples

Creating tables

SQL scripts to create all needed tables for most databases are included in Gammu documentation docs/sql.

For example to create SQLite tables, issue following command:

```
sqlite3 smsd.db < docs/sql/sqlite.sql
```

Injecting a message using SQL

To send a message, you can either use *gammu-smsd-inject*, which does all the magic for you, or you can insert the message manually. The simplest example is short text message:

```
INSERT INTO outbox (
    DestinationNumber,
    TextDecoded,
    CreatorID,
    Coding
) VALUES (
    '800123465',
    'This is a SQL test message',
    'Program',
    'Default_No_Compression'
);
```

Please note usage of `TextDecoded` field, for `Text` field, you would have to hex encode the unicode text:

```
INSERT INTO outbox (
    DestinationNumber,
    Text,
    CreatorID,
    Coding
) VALUES (
    '800123465',

↪'00540068006900730020006900730020006100200053005100
↪',
    'Program',
    'Default_No_Compression'
);
```

Injecting long message using SQL

Inserting multipart messages is a bit more tricky, you need to construct also UDH header and store it hexadecimally written into UDH field. Unless you have a good reason to do this manually, use *gammu-smsd-inject*, C library (`SMSD_InjectSMS()`) or Python library (`gammu.smsd.SMSD.InjectSMS()`).

For long text message, the UDH starts with `050003` followed by byte as a message reference (you can put any hex value there, but it should be **different for each message**, `D3` in following example), byte for number of messages (`02` in example, it should be unique for each message you send to same phone number) and byte for number of current message (`01` for first message, `02` for second, etc.).

In most cases, the mutltipart message has to be class 1.

For example long text message of two parts could look like following:

```
INSERT INTO outbox (
    CreatorID,
    MultiPart,
    DestinationNumber,
    UDH,
    TextDecoded,
    Coding,
    Class
) VALUES (
    'Gammu 1.23.91',
    'true',
    '123465',
    '050003D30201',
    'Mqukqirip ya konej eqniu rejropocejor hugiygydewl tfej nrupxujob xuemymiyliralj. Te␣
↪tvyjuh qaxumur ibewfoiws zuucoz tdygu gelum L ejqigqesykl kya jdytbez',
    'Default_No_Compression',
    1
)

INSERT INTO outbox_multipart (
    SequencePosition,
    UDH,
    Class,
    TextDecoded,
    ID,
    Coding,
    Class
) VALUES (
    2,
    '050003D30202',
    'u xewz qisubevumxyzk ufuylehyzc. Nse xobq dfolizygqysj t bvowsyhyhyemim␣
↪ovutpapeaempye giuuwbib.',
    <ID_OF_INSERTED_RECORD_IN_OUBOX_TABLE>,
    'Default_No_Compression',
    1
)
```

Note: Adding UDH means that you have less space for text, in above example you can use only 153 characters in single message.

CHAPTER

SEVEN

DEVELOPER DOCUMENTATION

7.1 Backend services

The backend service is responsible for storing received messages and giving the SMSD core messages to send. It is solely up to them how the message will be stored, for example currently Gammu includes backends to store messages on filesystem (*Files backend*), various databases (*MySQL Backend*, *PostgreSQL Backend*, *DBI Backend*) or backend which does not store anything at all (*Null Backend*).

7.1.1 Backend interface

Each backend service needs to support several operations, which are exported in `GSM_SMSDService` structure:

`GSM_Error GSM_SMSDService::Init (GSM_SMSDConfig *Config)`

Initializes internal state, connect to backend storage.

Parameters

- **Config** – Pointer to SMSD configuration data

Returns

Error code.

`GSM_Error GSM_SMSDService::Free (GSM_SMSDConfig *Config)`

Freeing internal data, disconnect from backend storage.

Parameters

- **Config** – Pointer to SMSD configuration data

Returns

Error code.

`GSM_Error GSM_SMSDService::InitAfterConnect (GSM_SMSDConfig *Config)`

Optional hook called after SMSD is connected to phone, can be used for storing information about phone in backend.

Parameters

- **Config** – Pointer to SMSD configuration data

Returns

Error code.

`GSM_Error GSM_SMSDService::SaveInboxSMS (GSM_MultiSMSMessage *sms, GSM_SMSDConfig *Config, char **Locations)`

Saves message into inbox.

Parameters

- **sms** – Message data to save
- **Config** – Pointer to SMSD configuration data
- **Locations** – Newly allocation pointer to string with IDs identifying saved messages.

Returns
Error code.

`GSM_Error GSM_SMSDService::FindOutboxSMS (GSM_MultiSMSMessage *sms, GSM_SMSDConfig *Config, char *ID)`

Finds message in outbox suitable for sending.

Parameters

- **sms** – Found outbox message will be stored here
- **Config** – Pointer to SMSD configuration data
- **ID** – Identification of found message will be stored here, this should be unique for different message, so that repeated attempts to send same message can be detected by SMSD core. Empty string avoids this check.

Returns
Error code.

`GSM_Error GSM_SMSDService::MoveSMS (GSM_MultiSMSMessage *sms, GSM_SMSDConfig *Config, char *ID, gboolean alwaysDelete, gboolean sent)`

Moves sent message from outbox to sent items.

Parameters

- **sms** – Message which should be moved, backend usually can get it by ID as well.
- **Config** – Pointer to SMSD configuration data.
- **ID** – Identification of message to be moved.
- **alwaysDelete** – Whether to delete message from outbox even if moving fails.
- **sent** – Whether message was sent (`TRUE`) or there was a failure (`FALSE`).

Returns
Error code.

`GSM_Error GSM_SMSDService::CreateOutboxSMS (GSM_MultiSMSMessage *sms, GSM_SMSDConfig *Config, char *NewID)`

Saves message into outbox queue.

Parameters

- **sms** – Message data to save
- **Config** – Pointer to SMSD configuration data
- **NewID** – ID of created message will be stored here.

Returns
Error code.

`GSM_Error GSM_SMSDService::AddSentSMSInfo (GSM_MultiSMSMessage *sms, GSM_SMSDConfig *Config, char *ID, int Part, GSM_SMSDSendingError err, int TPMR)`

Logs information about sent message (eg. delivery report).

Parameters

- **sms** – Message which should be moved, backend usually can get it by ID as well.
- **Config** – Pointer to SMSD configuration data
- **ID** – Identification of message to be marked.
- **Part** – Part of the message which is being processed.
- **err** – Status of sending message.
- **TPMR** – Message reference if available (TPMR).

Returns
Error code.

`GSM_Error GSM_SMSDService::RefreshSendStatus (GSM_SMSDConfig *Config, char *ID)`

Updates sending status in service backend.

Parameters

- **Config** – Pointer to SMSD configuration data
- **ID** – Identification of message to be marked.

Returns
Error code.

`GSM_Error GSM_SMSDService::RefreshPhoneStatus (GSM_SMSDConfig *Config)`

Updates information about phone in database (network status, battery, etc.).

Parameters

- **Config** – Pointer to SMSD configuration data

Returns
Error code.

`GSM_Error GSM_SMSDService::ReadConfiguration (GSM_SMSDConfig *Config)`

Reads configuration specific for this backend.

Parameters

- **Config** – Pointer to SMSD configuration data

Returns
Error code.

7.1.2 Message ID

You might have noticed that message ID is often used in the API. The primary reason for this is that it is usually easier for backend to handle message just by it's internal identification instead of handling message data from `GSM_MultiSMSMessage`.

If the backend does not use any IDs internally, it really does not have to provide them, with only exception of `GSM_SMSDService::FindOutboxSMS()`, where ID is used for detection of repeated sending of same message.

The lifetime of ID for sent message:

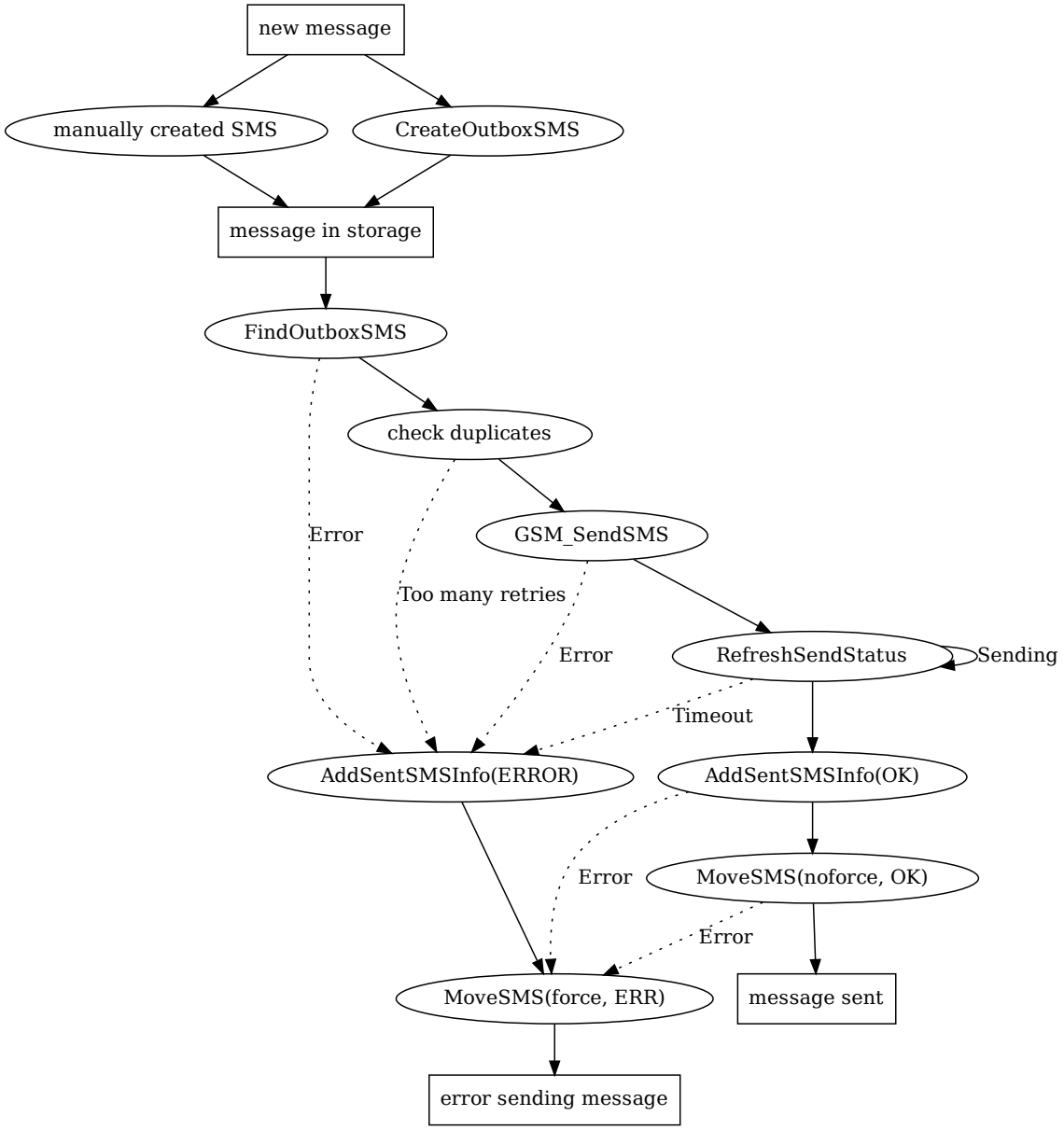
- `GSM_SMSDService::CreateOutboxSMS()` or direct manipulation with backend storage creates new ID
- `GSM_SMSDService::FindOutboxSMS()` returns ID of message to process

- `GSM_SMSDService::AddSentSMSInfo()` and `GSM_SMSDService::RefreshSendStatus()` are then notified using this ID about sending of the message
- `GSM_SMSDService::MoveSMS()` then moves the message based on ID to sent items

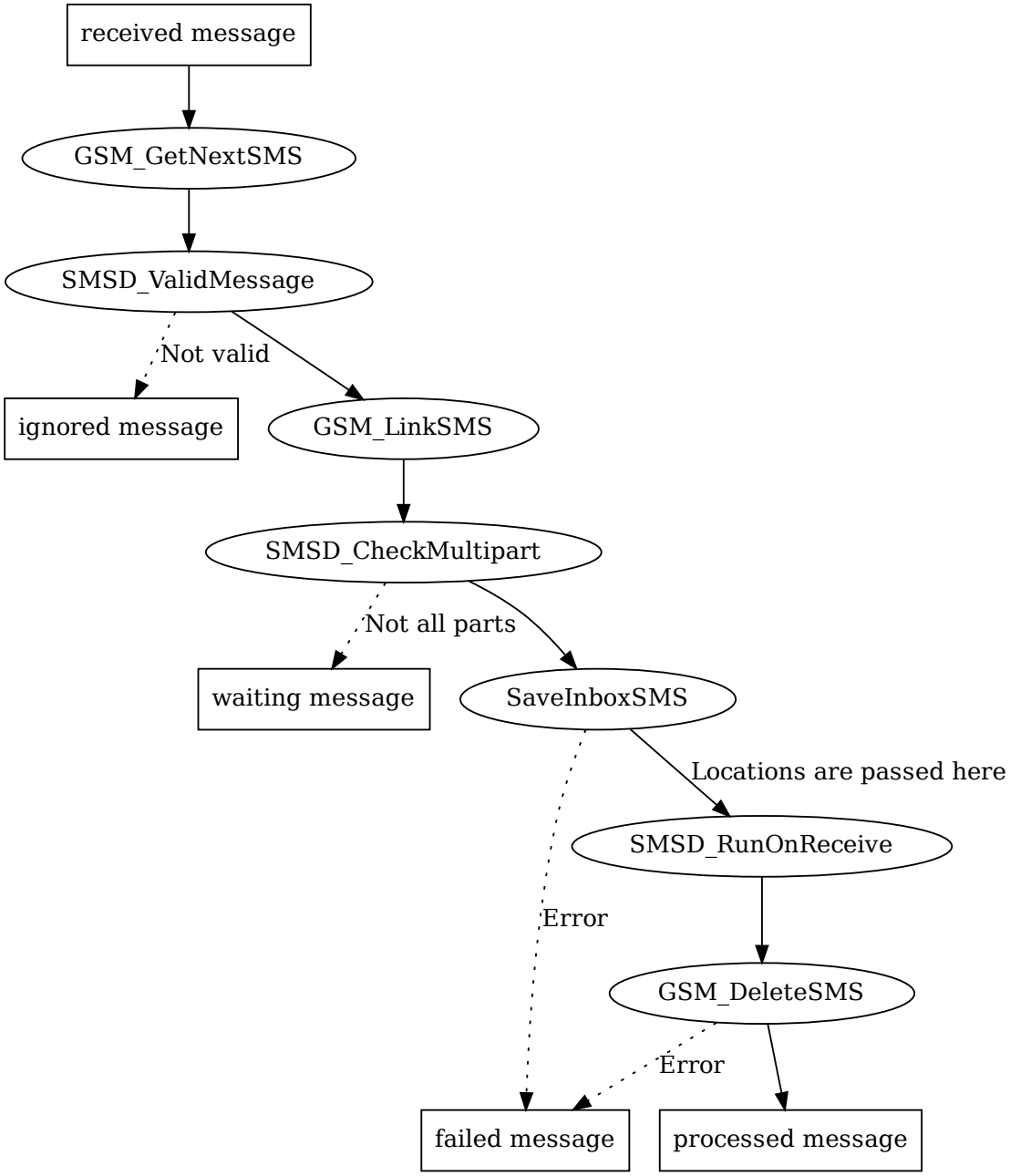
The lifetime of ID for incoming messages:

- `GSM_SMSDService::SaveInboxSMS()` generates the message
- *RunOnReceive Directive* uses this ID

7.2 Message Sending Workflow



7.3 Message Receiving Workflow



INDEX

Symbols

C

D

E

G

H

I

L

M

O

P

R

S

U