Gammu Manual

Release 1.42.0

Michal Čihař <michal@cihar.com>

Feb 20, 2024

CONTENTS

CHAPTER

ONE

GAMMU PROJECT

1.1 About Gammu

Gammu is library and command line utility for mobile phones. It is released under GNU GPL version 2.

It has been initiated by Marcin Wiacek and other people. Originally the code was based on Gnokii and later MyGnokii projects. Gammu was former (up to version 0.58) called MyGnokii2.

Currently the project is lead by Michal Čihař with help of many contributors.

1.2 Motivation to fork Gnokii

Note: Please note that this is original list of differences written by Marcin when forking Gnokii, so it represents state of the code in that time.

1. **Unicode used almost everywhere. In MyGnokii and Gnokii with modern**
   phones (they return everything in Unicode) things are converted from Unicode and again to Unicode in other places. No more unnecessary conversions.
2. **Almost everything is structural. In Gnokii some things are declared**
   in files, not in "main" phone structure. It can make some problems, when will try to support two phones on two serial ports in one application.
3. **in Gammu you can make support for some things without adding source**
   to "main" phone modules. Very good idea for things, which are available only for few models and for all other will be UNIMPLEMENTED. It includes also some obsolete functions - why should we compile RLP source, when all new better phones have modems built in ?
4. Gnokii/MyGnokii has to have some compatibility with previously written source. In Gammu some solutions are reimplemented and done easier.
5. no more reimplementing C libraries in source - see snprintf in gnokii.
6. more OS supported.
7. better sharing source. Less source = smaller application easier to debug.
8. better user friendly interface
9. no more 2 years rewriting source…
10. it's easier to see, what frames are implemented, what not (in phone modules they're put line after line).
11. better compatibility with ANSI C = no warnings in MS VC 6

12. all locations for user start from 0 (in Gnokii some from 0, some from 1)
13. some things like SMS can be accessed few ways
14. when possible, there are used "constant" locations. I will explain on the example:
    1. save two calendar notes in any Nokia 61xx phone. Call them "reminder" and "call" notes. Reminder will be returned by phone of 1'st location, Call on 2'nd.
    2. Now Reminder will be deleted (for example, from phone keypad). Call will be moved from 2'nd to 1'st.
    3. When will read calendar notes again, have to read all notes again because of changed locations (let's say, we won't read Call note again. It will have location 2 in PC. Now you will write new note into phone (for keypad) and it will save in under location 2. When will try to save Call not with location 2, it will overwrite new saved note !).

    This is not good. When for example delete one entry from phonebook, other locations "stays" on their places. These are "constant" locations.

    With "constant" locations, when delete one location from PC, don't have to read full memory from phone.

etc. etc.

Of course, some of these things can be in the future in gnokii too…

1.3 Installing Gammu

1.3.1 Prebuilt Binaries for Linux

Many distributions come with prebuilt Gammu binaries, if you can use them, it is definitely the easiest thing. There are also binary packages of latest release built for many distributions available on Gammu home page <https://wammu.eu/gammu/>.

You can usually also find Gammu in your distribution, so unless you need a newer version, just install package from your distribution.

Debian

Gammu packages are included in Debian (testing versions go to experimental and stable to unstable). If you want to build Debian package on your own, you can find packaging in Git repository at `https://anonscm.debian.org/git/collab-maint/gammu.git` (you can browse it on <https://anonscm.debian.org/git/collab-maint/gammu.git>).

RPM

Gammu packages are included in openSUSE and Fedora. Additionally source tarball contains `gammu.spec` which you can use for building RPM package.

Slackware

Gammu packages are included in Gentoo. Additionally source tarball contains `description-pak` which you can use for building Slackware package.

1.3.2 Prebuilt Binaries for Windows

You can download Windows binaries from <https://wammu.eu/gammu/>. For Windows 95, 98 and NT 4.0 you will also need ShFolder DLL, which can be downloaded from Microsoft:

http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=6AE02498-07E9-48F1-A5D6-DBFA18D37E0F

1.3.3 Dependencies

You need CMake from <https://cmake.org/> for compiling Gammu.

Additionally pkg-config <https://www.freedesktop.org/wiki/Software/pkg-config/> is used for detecting available libraries.

1.3.4 Optional Dependencies

Gammu does not require any special libraries at all to build, but you might miss some features. Optional libraries include:

Bluez-libs

Required for Bluetooth support on Linux.

See also:

http://www.bluez.org/

libusb-1.0

Required for fbususb/dku2 connection support on Linux.

See also:

http://libusb.sourceforge.net/

libCURL

Required for new versions notification (see `gammu checkversion`).

See also:

https://curl.haxx.se/libcurl/

libiconv

Used to support more character sets in AT engine.

See also:

https://www.gnu.org/software/libiconv/

Gettext

Localization of strings.

See also:

https://www.gnu.org/software/gettext/

MySQL

Required for *MySQL Backend* in *SMS Daemon*.

See also:

https://www.mysql.com/

PostgreSQL

Required for *PostgreSQL Backend* in *SMS Daemon*.

See also:

https://www.postgresql.org/

unixODBC

Required for *ODBC Backend* in *SMS Daemon*.

Note: Not needed on platforms having native ODBC support such as Microsoft Windows.

See also:

http://www.unixodbc.org/

libdbi

Required for *DBI Backend* in *SMS Daemon*.

Note: Required at least version 0.8.2.

See also:

http://libdbi.sourceforge.net/

Python

Gammu has a Python bindings, see *python-gammu*.

See also:

https://www.python.org/

SQLite + libdbi-drivers with SQLite

Needed for testing of SMSD using libdbi driver (libdbd-sqlite3), see *Testing Gammu*.

See also:

https://www.sqlite.org/

glib

Currently needed only for *gammu-detect*.

See also:

https://www.gtk.org/

gudev

Currently needed only for *gammu-detect*.

See also:

http://gudev.sourceforge.net/

1.3.5 Compiling on Linux/Unix Systems

First install all *Dependencies* and *Optional Dependencies*. Do not forget to install corresponding devel packages as well, they are usually named with `-dev` or `-devel` suffix, depending on your distribution.

For example on Debian or Ubuntu, you can install all optional packages by following command:

```
apt-get install cmake python-dev pkg-config libmysqlclient-dev libpq-dev \
    libcurl4-gnutls-dev libusb-1.0-0-dev libdbi0-dev libbluetooth-dev \
    libgudev-1.0-dev libglib2.0-dev unixodbc-dev
```

For openSUSE, the installation all optional packages could look like:

```
zypper install libusb-1_0-devel libdbi-devel bluez-devel postgresql-devel \
    mysql-devel python-devel libcurl-devel cmake pkgconfig unixODBC-devel \
    glib2-devel libgudev-1_0-devel
```

Configure like wrapper

For compatibility reasons, configure like wrapper is provided, if you don't need much specific tuning, you can use usual set of commands:

```
./configure
make
sudo make install
```

The configure wrapper will create directory build-configure and build all binaries there (nothing is changed in source tree), for example gammu binary is in build-configure/gammu directory.

Using CMake

If you need/want to tweak build a bit more than configure wrapper provides, you have to use CMake directly. For now, only out of source build is supported, so you have to create separate directory for build:

```
mkdir build
cd build
```

Then just configure project:

```
cmake ..
```

Build it:

```
make
```

Test that everything is okay:

```
make test
```

And finally install it:

```
sudo make install
```

You can configure build parameters either by command line (see parameters below), or using TUI - ccmake.

Useful cmake parameters:

- `-DBUILD_SHARED_LIBS=ON` enables shared library
- `-DCMAKE_BUILD_TYPE="Debug"` enables debug build
- `-DCMAKE_INSTALL_PREFIX="/usr"` change installation prefix
- `-DENABLE_PROTECTION=OFF` disables various compile time protections against buffer overflows and similar attacks

You can also disable support for whole set of phones, e.g.:

- `-DWITH_NOKIA_SUPPORT=OFF` disables Nokia phones support
- `-DWITH_BLUETOOTH=OFF` disables Bluetooth support
- `-DWITH_IRDA=OFF` disables IrDA support

Library search paths

By installing Gammu to non default system paths, you might need to add path where libGammu and other Gammu liraries are installed to **ldconfig** search path.

You can do this by editing `/etc/ld.so.conf` or adding new file to `/etc/ld.so.conf.d/` directory containing path, wherge Gammu library has been installed. Some examples:

```
# Gammu on 64-bit Fedora installed to /opt/gammu
echo /opt/gammu/lib64 > /etc/ld.so.conf.d/gammu.conf

# Gammu installed to /usr/local
echo /usr/local/lib > /etc/ld.so.conf.d/gammu.conf
```

The similar situation exists with Python modules, if you install in path when your Python interpreter does not search it won't load newly installed Gammu bindings.

You can also avoid changing ldconfig configuration by installing Gammu to paths where it already searches, for examble by:

```
cmake .. -DCMAKE_INSTALL_PREFIX="/usr"
```

1.3.6 Compiling on Microsoft Windows

First install all *Dependencies* and *Optional Dependencies*.

CMake is able to generate projects for various tools including Microsoft Visual Studio, Borland toolchains, Cygwin or Mingw32. Just click on CMakeLists.txt in project sources and configure CMake to be able to find optional libraries (see cross compilation section for more information about getting those). The result should be project for your compiler where you should be able to work with it as with any other project.

Compiling using MS Visual C++

You will probably need additional SDKs:

- Microsoft Windows Platform SDK (required especially for Bluetooth). It's given for free. Below are links to different releases (if you have problems with latest one, use older). They work for various Windows versions, even though Microsoft named them Windows Server 2003 Platform SDK.
- For free Visual C++ Express 2005 you need to set compiler to work with Platform SDK (see description).
- MySQL include/library files from MySQL install package (for MySQL support in SMSD).
- PostgreSQL include/library files from PostgreSQL install package (for PostgreSQL support in SMSD).
- For gettext (internationalization) support, you will need gettext packages from GnuWin32 project.
- As build is now based on CMake, you will need to get it from https://cmake.org/.

After downloading and installing them into your system:

- Now you should be able to execute cmake by clicking on CMakeLists.txt file in Gammu sources, this should pop up dialog with configuration options.
  - You can also start CMakeSetup from start menu and select source directory (just point to it to Gammu sources).
  - Select directory where binaries will be stored, I suggest this is different than source one, eg. append subdirectory build.

  - Select compiler you want to use in Build for select.
- In list below, you can tweak paths to some optional libraries and project configuration.
- Then just press Configure button, which will do the hard job. After this, just click OK button to generate Visual Studio project.
- Project files for Visual Studio should be now generated in directory you selected, just open it in Visual Studio and compile :-).
  - Project file should be named Gammu.dsw or Gammu.sln depending on what MSVC version you choose.
  - You should see ALL_BUILD target, which builds everything needed, similar to make all on Linux.
- For running testsuite, you need working sh and sed. The easiest way to install them is from MinGW project <http://mingw.org/>.
- I know this guide is incomplete, I don't have environment to test, you're welcome to improve it!. Some more information can be found in howtos for other projects using CMake, eg. Blender, SIM, KDE, VTK, ISGTK. ITK, [wxWidgets http://www.wxwidgets.org/wiki/index.php/CMake].

Compiling using Borland C++

Borland toolchain - you can download compiler at <http://www.codegear.com/downloads/free/cppbuilder>. You need to add c:/Borland/BCC55/Bin to system path (or manually set it when running CMake) and add -Lc:/Borland/BCC55/Lib -Ic:/Borland/BCC55/Include -Lc:/Borland/BCC55/Lib/PSDK to CMAKE_C_FLAGS in CMake (otherwise compilation fails).

Compiling using Cygwin

This should work pretty much same as on Linux.

1.3.7 Compiling on Mac OS X

First install all *Dependencies* and *Optional Dependencies*.

Gammu should be compilable on Mac OS X, you need to have installed Developer Tools (version 2.4.1 was tested) and CMake (there is a Mac OS X "Darwin" DMG download). For database support in SMSD, install wanted database, eg. MySQL.

The rest of the compilation should be pretty same as on Linux, see Linux section for more details about compile time options.

If you get some errors while linking with iconv, it is caused by two incompatible iconv libraries available on the system. You can override the library name:

```
cmake -D ICONV_LIBRARIES="/opt/local/lib/libiconv.dylib" ..
```

Or completely disable iconv support:

```
cmake -DWITH_Iconv=OFF ..
```

To build backward compatible binaries, you need CMake 2.8 or newer. The command line then would look like:

```
cmake -DCMAKE_OSX_ARCHITECTURES="ppc;i386;x86_64" -DCMAKE_OSX_DEPLOYMENT_TARGET=10.4
```

1.3.8 Cross compilation for Windows on Linux

First install all *Dependencies* and *Optional Dependencies* into your mingw build environment.

Only cross compilation using CMake has been tested. You need to install MinGW cross tool chain and run time. On Debian you can do it by apt-get install mingw32. Build is then quite simple:

```
mkdir build-win32
cd build-win32
cmake .. -DCMAKE_TOOLCHAIN_FILE=../cmake/Toolchain-mingw32.cmake
make
```

There is also toolchain configuration for Win64 available:

```
mkdir build-win64
cd build-win64
cmake .. -DCMAKE_TOOLCHAIN_FILE=../cmake/Toolchain-mingw64.cmake
make
```

If your MinGW cross compiler binaries are not found automatically, you can specify their different names in cmake/Toolchain-mingw32.cmake.

To build just bare static library without any dependencies, use:

```
cmake .. -DCMAKE_TOOLCHAIN_FILE=../cmake/Toolchain-mingw32.cmake \
    -DBUILD_SHARED_LIBS=OFF \
    -DWITH_MySQL=OFF \
    -DWITH_Postgres=OFF \
    -DWITH_GettextLibs=OFF \
    -DWITH_Iconv=OFF \
    -DWITH_CURL=OFF
```

To be compatible with current Python on Windows, we need to build against matching Microsoft C Runtime library. For Python 2.4 and 2.5 MSVCR71 was used, for Python 2.6 the right one is MSVCR90. To achieve building against different MSVCRT, you need to adjust compiler specifications, example is shown in cmake/mingw.spec, which is used by CMakeLists.txt. You might need to tune it for your environment.

Third party libraries

The easiest way to link with third party libraries is to add path to their installation to cmake/Toolchain-mingw32.cmake or to list these paths in CMAKE_FIND_ROOT_PATH when invoking cmake.

MySQL

You can download MySQL binaries from <http://dev.mysql.com/>, but then need some tweaks:

```
cd mysql/lib/opt
reimp.exe -d libmysql.lib
i586-mingw32msvc-dlltool --kill-at --input-def libmysql.def \
    --dllname libmysql.dll --output-lib libmysql.a
```

reimp.exe is part of mingw-utils and can be run through wine, I didn't try to compile native binary from it.

PostgreSQL

You can download PostgreSQL binaries from <http://www.postgresql.org/>, but then you need to add wldap32.dll library to bin.

Gettext

For Gettext (internationalization support), you need gettext-0.14.4-bin.zip, gettext-0.14.4-dep.zip, gettext-0.14.4-lib.zip from <http://gnuwin32.sourceforge.net/>. Unpack these to same directory.

CURL

For CURL support, you need curl-7.19.0-devel-mingw32.zip from <http://curl.haxx.se/>.

1.3.9 Crosscompiling to different platform

To cross compile Gammu to different architecture (or platform) you need to provide CMake toolchain file for that and invoke CMake with it:

```
cmake -DCMAKE_TOOLCHAIN_FILE=~/Toolchain-eldk-ppc74xx.cmake ..
```

More information on creating that is described in CMake Cross Compiling wiki page. Also distributions like OpenEmbedded usually already come with prepared recipes for CMake.

1.3.10 Advanced Build Options

The build system accepts wide range of options. You can see them all by running GUI version of CMake or by inspecting `CMakeCache.txt` in build directory.

Limiting set of installed data

By setting following flags you can control which additional parts will be installed:

- INSTALL_GNAPPLET - Install Gnapplet binaries
- INSTALL_MEDIA - Install sample media files
- INSTALL_PHP_EXAMPLES - Install PHP example scripts
- INSTALL_BASH_COMPLETION - Install bash completion script for Gammu
- INSTALL_LSB_INIT - Install LSB compatible init script for Gammu
- INSTALL_DOC - Install documentation
- INSTALL_LOC - Install locales data

For example:

```
cmake -DINSTALL_DOC=OFF
```

Debugging build failures

If there is some build failure (eg. some dependencies are not correctly detected), please attach `CMakeCache.txt`, `CMakeFiles/CMakeError.log` and `CMakeFiles/CMakeOutput.log` files to the report. It will help diagnose what was detected on the system and possibly fix these errors.

To find out what is going on during compilation, add `-DCMAKE_VERBOSE_MAKEFILE=ON` to **cmake** command line or run **make** with `VERBOSE=1`:

```
make VERBOSE=1
```

Debugging crashes

To debug program crashes, you might want to build Gammu with `-DENABLE_PROTECTION=OFF`, otherwise debugging tools are somehow confused with protections GCC makes and produce bogus back traces.

1.3.11 Installing python-gammu

You need to have `gammu` and `libgammu-dev` installed for using python-gammu.

```
apt-get install gammu libgammu-dev
pip3 install python-gammu
```

The location of the libraries is discovered using `pkg-config`, `GAMMU_PATH` environment variable and falls back to generic locations. In case it does not work, either install `pkg-config` or set `GAMMU_PATH`. `GAMMU_PATH` is recommended when building on Windows.

Compiling python-gammu

Currently python-gammu is distributed as a separate package, which follows Python usual method for building modules - distutils, so use `setup.py` is placed in the top level directory:

```
./setup.py build
sudo ./setup.py install
```

Running with `GAMMU_PATH`:

On Linux something like this should work:

```
GAMMU_PATH=/opt/gammu python setup.py build
```

On Windows:

```
SET GAMMU_PATH="C:\Gammu"
python setup.py build
```

1.4 Contributing

We welcome contribution in any area, if you don't have developer skills, you can always contribute to *Localization* or just donate us money. In case you are interested in fixing some code, please read *Gammu internals* to understand structure of Gammu code. We also maintain list of wanted skills where you can find in which areas we currently mostly lack manpower.

1.4.1 Creating Pull Requests

The Gammu project is hosted on Github which uses Git as version control system in the Background.

So start with forking & cloning our Git repository:

```
git clone https://github.com/gammu/gammu.git gammu
```

Once you have done that, do some fixes and commit them (see Git tutorial for information how to work with Git).

Once you're satisfied with your results, you can share your changes as Pull Request with us.

1.5 Localization

Localization uses Gettext po files for both program translations and the documentation. The documentation translation is managed using po4a.

1.5.1 Using Translation

You can set locales you want to use by specifying LANG or LC_* environment variables (on Linux you usually don't care about this, on Windows just export e.g. `LANG=cs_CZ`).

1.5.2 Improving Translation

If you want to improve existing translation, please visit translation server. For adding new one, you need to contact Michal Čihař and then you will be able to edit it on former mentioned URL.

You can also go ahead with traditional way of creating/updating po files in locale/ folder and then sending updated ones to bug tracker.

1.5.3 Translation Areas

There are several po files to translate:

`libgammu.po`
: Messages used in the Gammu library (see *libGammu*).

`gammu.po`
: Messages used by command line utilities (mostly *Gammu Utility*).

`docs.po`
: Basic documentation shipped within package (eg. `README.rst` and `INSTALL` files).

1.6 Testing

Gammu comes with quite big test suite. It covers some basic low level functions, handling replies from the phone and also does testing of command line utilities and SMSD.

See also:

See *Testing Gammu* for more details.

1.7 Releasing Gammu

1. Ensure that all tests pass on both Linux and Windows.
2. Update changelog and fill in release date in `ChangeLog`.
3. Update man pages using `make update-man`.
4. Run `./admin/make-release` to verify release build works.
5. (optional) Test created tarballs.
6. Run `./admin/make-release branch` to make final release.
7. Push created tag to GitHub.
8. Wait for AppVeyor to produce Windows binaries.
9. Import release to the website.

1.8 Coding Style

Please follow coding style when touching Gammu code. We know that there are still some parts which really do not follow it and fixes to that are also welcome.

The coding style is quite similar to what Linux kernel uses, the only major differences are requested block braces and switch indentation.

1. Use indentation, tab is tab and is 8 chars wide.
2. Try to avoid long lines (though there is currently no hard limit on line length).
3. **Braces are placed according to K&R:**

```
int function(int x)
{
    body of function
}

do {
    body of do-loop
} while (condition);

if (x == y) {
    ..
} else if (x > y) {
    ...
```

(continues on next page)

(continued from previous page)

```
} else {
    ...
}
```

4. **All blocks should have braces, even if the statements are one liners:**

```
if (a == 2) {
    foo();
}
```

5. **There should be no spaces after function names, but there should be space after do/while/if/... statements:**

```
while (TRUE) {
    do_something(work, FALSE);
}
```

6. **Each operand should have spaces around, no spaces after opening parenthesis or before closing parenthesis:**

```
if ((i + 1) == ((j + 2) / 5)) {
    return *bar;
}
```

7. Generally all enums start from 1, not from 0. 0 is used for not set value.

You can use **admin/Gindent** to adjust coding style of your file to match our coding style.

1.9 Versioning

There are two types of releases - testing and stable, both having version x.y.z. Stable releases have usually z = 0 or some small number, while testing ones have z >= 90. Testing releases usually provide latest features, but everything does not have to be stabilized yet.

1.10 Project Documentation

The documentation for Gammu consists of two major parts - The Gammu Manual, which you are currently reading and comments in the sources, which are partly included in this manual as well.

1.10.1 The Gammu Manual

This manual is in written in rst format and built using Sphinx with breathe extension.

To generate the documentation there are various `manual-*` targets for make. You can build HTML, PDF, PS, HTML-HELP and Latex versions of it:

```
# Generates HTML version of manual in docs/manual/html
make manual-html

# Generates PS version of manual in docs/manual/latex/gammu.ps
```

(continues on next page)

(continued from previous page)

```
make manual-ps

# Generates PDF version of manual in docs/manual/latex/gammu.pdf
make manual-pdf

# Generates HTML version of manual in docs/manual/htmlhelp
make manual-htmlhelp

# Generates HTML version of manual in docs/manual/latex
make manual-latex
```

1.10.2 Man pages

The man pages for all commands are generated using Sphinx as well:

```
# Generates HTML version of manual in docs/manual/man
make manual-man
```

However man pages are stored in Git as well, so you should update generated copy on each change:

```
# Updates generated man pages in Git
make update-man
```

1.10.3 Code comments

The code comments in C code should be parseable by Doxygen, what is more or less standard way to document C code.

1.11 Directory structure

1.11.1 libgammu directory

This directory contains sources of Gammu library. You can find all phone communication and data encoding functionality here.

There are following subdirectories:

device
: drivers for devices such serial ports or irda

device/serial
: drivers for serial ports

device/irda
: drivers for infrared over sockets

protocol
: protocol drivers

protocol/nokia
: Nokia specific protocols

phone
: phone modules

phone/nokia
: modules for different Nokia phones

misc
: different services. They can be used for any project

service
: different gsm services for logos, ringtones, etc.

1.11.2 gammu directory

Sources of Gammu command line utility. It contains interface to libGammu and some additional functionality as well.

1.11.3 smsd directory

Sources of SMS Daemon as well as all it's service backends.

The `services` subdirectory contains source code for *Backend services*.

1.11.4 helper directory

These are some helper functions used either as replacement for functionality missing on some platforms (eg. strptime) or used in more places (message command line processing which is shared between SMSD and Gammu utility).

1.11.5 docs directory

Documentation for both end users and developers as well as SQL scripts for creating SMSD database.

config
: configuration file samples

examples
: examples using libGammu

manual
: sources of The Gammu Manual which you are reading

sql
: SQL scripts to create table structures for *SMS Daemon*

user
: user documentation like man pages

1.11.6 admin directory

Administrative scripts for updating locales, making release etc.

1.11.7 cmake directory

CMake include files and templates for generated files.

1.11.8 include directory

Public headers for libGammu.

1.11.9 locale directory

Gettext po files for translating Gammu, libGammu and user documentation. See *Localization* for more information.

1.11.10 tests directory

CTest based test suite for libGammu. See *Testing* for more information.

1.11.11 utils directory

Various utilities usable with Gammu.

1.11.12 contrib directory

This directory contains various things which might be useful with Gammu. Most of them were contributed by Gammu users.

Note: Please note that that code here might have different license terms than Gammu itself.

Warning: Most of scripts provided here are not actively maintained and might be broken.

bash-completion

Completion script for bash.

conversion

Various scripts for converting data.

init

Init scripts for Gammu SMSD.

media

Sample media files which can be used with Gammu.

perl

Various perl scripts which interface to Gammu or SMSD.

php

Various PHP frontends to SMSD or Gammu directly.

sms

This directory contains SMS default alphabet saved in Unicode text file (`charset.txt`) and table used for converting chars during saving SMS with default alphabet (`convert.txt`).

sms-gammu2android

Perl script to convert *SMS Backup Format* into XML suitable for Android SMS Backup & Restore application.

See also:

http://blog.ginkel.com/2009/12/transferring-sms-from-nokia-to-android/

smscgi

Simple cgi application gor handling SMS messages (a bit lighter version of SMSD).

sql

Various SQL snippets and triggers useful with SMSD.

testing

Helper scripts for automatic testing or git bisect.

sqlreply

System for automatic replying to SMS messages.

symbian

GNapplet sources and binaries. This comes from Gnokii project, but Gammu includes slightly modified version.

s60

Series60 applet to use with recent Symbian phones.

See also:

Series60 Remote Protocol

win32

Unsupported applications built on top of libGammu.dll on Windows.

1.12 Roadmap for Gammu

There are some major issues which should be addressed in Gammu soon. This list is not sorted at all, but includes bad design decisions made in Gammu past which we would like to fix.

1.12.1 Locations handling

One problem is locations handling, because current scheme (using numbers) really does not match majority of current phones and it should be converted to using path based locations for messages, phonebook, calendar, etc.

1.12.2 Unicode strings

The another major obstacle which is all around Gammu code is own implementation of unicode (UCS-2-BE) strings. This code should be dropped and use some standard library for that. Note that wchar_t is probably not a good choice here as it's 16-bit on Windows and thus can not store emojis and other supplemental plan unicode chars.

1.12.3 Hardcoded length for strings

Most of the strings have hardcoded length limits. This limitation should be removed and strings allocated on the fly.

1.12.4 Unsigned char mess

In many cases `unsigned char` is used without any reason.

1.12.5 Extensibility of libGammu

Current way of adding protocol specific functionality from applications using libGammu is broken. Actually only application using this is Gammu utility. This option should be either completely removed or rewritten from scratch not to be dependent on libGammu internals.

1.12.6 Built time configuration

Avoid heavy usage of gsmstate.h header and move the `#ifdef...#define...#endif` blocks to `gammu-config.h`. Or rather cleanup them and have only single define for single compile time option.

1.12.7 Config file handling

Drop multiple configurations handling in libGammu, it should provide just API to read some section from Gammurc and possible fall-back logic should be in application.

1.12.8 AT module

There should be simpler way to generate AT command with proper escaping and charset conversion of fields. Something like reverse ATGEN_ParseReply.

CHAPTER

TWO

QUICK STARTER GUIDE

2.1 Gammu family

Gammu family consists of several programs and libraries:

Gammu Utility
: Command line utility to talk to the phone. It performs one time operations only.

Wammu
: Graphical interface for Gammu, providing basic functions.

gammu-smsd
: Daemon to receive and send messages using your phone.

gammu-smsd-inject
: Injects outgoing messages into *gammu-smsd* queue.

gammu-smsd-monitor
: Monitors state of Gammu SMS Daemon. It periodically displays information about phone and number of processed messages.

gammu-detect
: Simple utility to detect phones or modems connected to computer.

python-gammu
: Python bindings for Gammu, use it from Python scripts.

libGammu
: Core library, used by all other parts and you can use it directly in your C programs.

2.2 Installing Gammu

On most platforms you can install Gammu from binaries - most Linux distributions ship Gammu and for Windows you can download binaries from Gammu website. You can find more detailed instructions (including instructions for compiling from source) in *Installing Gammu*.

2.3 Starting with Gammu on Linux

First you need to find out device name where your phone/modem is connected. In most cases you can rely on *gammu-detect* to find it (it will also list all serial ports in your systems, where probably nothing is connected).

Generally for most current modems you will end up with `/dev/ttyUSB0`.

The next step is to create configuration file in `~/.gammurc` (see *Gammu Configuration File*):

```
[gammu]
device = /dev/ttyUSB0
connection = at
```

And you can connect to the phone:

```
$ gammu identify
Device                : /dev/ttyUSB0
Manufacturer          : Wavecom
Model                 : MULTIBAND  900E  1800 (MULTIBAND  900E  1800)
Firmware              : 641b09gg.Q2403A 1320676 061804 14:38
IMEI                  : 123456789012345
SIM IMSI              : 987654321098765
```

2.4 Starting with Gammu on Windows

First you need to find out device name where your phone/modem is connected. The easiest way is to look into *Device manager* under *Ports (COM & LPT)* and lookup correct COM port there.

Generally for most current modems you will end up with something like `COM12`.

The next step is to create configuration file in `$PROFILE\Application Data\gammurc` (see *Gammu Configuration File*):

```
[gammu]
device = COM12:
connection = at
```

And you can connect to the phone:

```
C:\Program Files\Gammu 1.33.0\bin> gammu identify
Device                : COM12:
Manufacturer          : Wavecom
Model                 : MULTIBAND  900E  1800 (MULTIBAND  900E  1800)
Firmware              : 641b09gg.Q2403A 1320676 061804 14:38
IMEI                  : 123456789012345
SIM IMSI              : 987654321098765
```

2.5 Starting with SMSD

Note: Before starting with SMSD, make sure you can connect to your phone using Gammu (see chapters above for guide how to do that).

Once you have configured Gammu, running *gammu-smsd* is pretty easy. You need to decide where you want to store messages (see *Service*). For this example we will stick with MySQL database, but the instructions are quite similar for any storage service.

Note: 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* and *Invoking Gammu and suspending SMSD*):

2.5.1 Configuring the storage

First we have to setup the actual storage. With MySQL, we need access to the MySQL server. Now connect as administrative user to the server (usually `root`), grant privileges to the `smsd` user and create `smsd` database:

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

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

CREATE DATABASE smsd;
```

Once this is ready, you should import the tables structure. It is shipped as `docs/sql/mysql.sql` with Gammu, so all you have to do is to import this file (see *Creating tables for MySQL* for more details):

```
$ mysql -u root -p password smsd < docs/sql/mysql.sql
```

2.5.2 Configuring SMSD

Now we just have to tell SMSD what service it is supposed to use. This is done in the SMSD configuration file. You can place it anywhere and tell SMSD on startup where it can find it, but on Linux the recommended location for system wide service is `/etc/gammu-smsdrc` (see *SMSD Configuration File* for more information).

You have to put both modem and storage service configuration into this file:

```
[gammu]
device = /dev/ttyUSB0
connection = at

[smsd]
service = SQL
driver = native_mysql
host = localhost
database = smsd
user = smsd
password = password
```

There are many ways to customize SMSD, but the defaults should work fine in most environments. You can find more information on customizing SMSD in *SMSD Configuration File*.

2.5.3 Running SMSD

With configuration file ready, you can actually start SMSD. You can do this manually or as a system wide service.

For manual startup, just execute it:

```
$ gammu-smsd
```

Alternatively you can specify path to the configuration file:

```
$ gammu-smsd -c /path/to/gammu-smsdrc
```

The binary packages on Linux usually come with support for starting SMSD as a system wide daemon.

With systemd, you can start it by:

```
$ systemctl start gammu-smsd.service
```

2.5.4 Sending message through SMSD

Once SMSD is up and running, you can send some messages using it:

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

You can find more examples in the *gammu-smsd-inject* documentation: *Examples*.

CHAPTER

THREE

FREQUENTLY ASKED QUESTIONS

3.1 General Gammu FAQ

3.1.1 Will Gammu work on my system?

Gammu is known to run on wide range of systems. It can be compiled natively on Linux, Mac OS X, FreeBSD, OpenBSD and Microsoft Windows. It can be probably compiled also elsewhere, but nobody has yet tried. On some platforms however you might lack support for some specific kind of devices (eg. Bluetooth or USB).

See also:

Installing Gammu

3.1.2 How to set sender number in message?

You can quite often see messages sent from textual address or with some other nice looking sender number. However this needs to be done in the GSM network and it is not possible to influence this from the terminal device (phone). Usually it is set by SMSC and some network providers allow you to set this based on some contract. Alternatively you can use their SMS gateways, which also allow this functionality.

See also:

SMS and EMS commands

3.1.3 Can I use Gammu to send MMS?

MMS contains of two parts - the actual MMS data in SMIL format and the SMS containing notification about the data. Gammu can create the notification SMS, where you just need to put URL of the data (use `gammu sendsms` MMSINDICATOR for that). However you need to encode MMS data yourself or use other program to do that.

3.1.4 Can I use Gammu to receive MMS?

MMS contains of two parts - the actual MMS data in SMIL format and the SMS containing notification about the data. Gammu (or SMSD) will receive the notification SMS, where URL to download the MMS content is included.

However in most situations the URL is accessible only from the network and APN specific for the MMS messages, so downloading it is a bit tricky and needs to connect using GSM modem to the network using this APN.

3.1.5 Device name always changes on Linux, how to solve that?

You can use udev to assign persistent device name (used as *Device*). You can either use standard persistent names based on serial number (located in `/dev/serial/by-id/`) or define own rules:

```
ACTION=="add", SUBSYSTEMS=="usb", ATTRS{manufacturer}=="Nokia", KERNEL=="ttyUSB*",␣
↪SYMLINK+="phone"
```

Better is to use vendor and product IDs (you can get them for example using **lsusb**):

```
ACTION=="add", SUBSYSTEMS=="usb", ATTRS{idVendor}=="xxxx", ATTRS{idProduct}=="yyyy",␣
↪SYMLINK+="phone"
```

If you're using 3G modem, it's quite likely that it exposes multiple interfaces and only one of them is good for Gammu usage. In this case you should match against interface number as well:

```
ACTION=="add", SUBSYSTEMS=="usb", ATTRS{idVendor}=="xxxx", ATTRS{idProduct}=="yyyy",␣
↪ATTRS{bInterfaceNumber}=="00", SYMLINK+="phone"
```

You can match by various attributes, you can figure them using udevadm command:

```
udevadm info --name=/dev/ttyUSB1 --attribute-walk
```

See also:

Various documentation on creating persistent device names using udev is available online, for example on the Debian wiki or in Writing udev rules document.

3.1.6 Multiple programs using same device cause various errors, how to fix that?

Gammu needs to be the only program using the device, otherwise you will get strange errors from both programs as they will read answer to command sent by something else.

In gammu, it can happen quite early with error message "Phone does not support enabled echo, it can not work with Gammu!", but it can be spotted later as well, depending on various conditions.

In case you see such behavior, check what other programs are using given device. This can be done using **fuser** tool:

```
fuser -va /dev/ttyACM0
```

The usual programs involve:

- NetworkManager with ModemManager, you need to disable mobile networking to stop it using the device, disabling the modem connection does not seem to be enough.
- Other Gammu instance, in case you want to interact with modem while SMSD is running see *Invoking Gammu and suspending SMSD*.

3.1.7 What are free alternatives to Gammu?

It depends on your phone. For Nokia or AT based phones, you can try Gnokii, but Gammu should be superior in most cases. For Symbian phone you can try using Series60-Remote, which works pretty well with S60 phones, though Gammu brings various fixes to their applet.

If you are looking for synchronisation, try using something what supports SyncML to retrieve contacts or calendar from your phone, for example OpenSync or syncEvolution.

3.2 Configuring Gammu FAQ

3.2.1 How to configure 3G/UMTS/... modem or AT capable phone?

As most modems support AT commands, this is pretty easy and you should use `at` *Connection*. For *Device* you should use device name as modem appears in your system, for example `/dev/ttyACM0` or `COM7:`.

Some modems expose more serial ports and you need to carefully choose the right one - for example only one of them can receive USSD notifications.

Note: On Linux, you might have to install usb-modeswitch to make your modem actually behave like a modem and not like a disk containing drivers for Windows.

See also:

Device name always changes on Linux, how to solve that?, *Gammu Configuration File*

Example configuration on Linux:

```
[gammu]
device = /dev/ttyACM3
connection = at
```

Example configuration on Windows:

```
[gammu]
device = COM12:
connection = at
```

3.2.2 How to configure Symbian based phone?

The only support for Symbian phones is using applet installed to phone and Bluetooth connection. You should use `blues60` *Connection* and Bluetooth address of phone as *Device*. On older Symbian phones you will have to use gnapplet and `bluerfgnapbus` connection.

See also:

Series60 Remote Protocol, *Gammu Configuration File*

Note: Do not forget to start the applet before trying to connect to the phone.

Example configuration:

```
[gammu]
device = 11:22:33:44:55:66 # Bluetooth address of your phone
connection = blues60
```

3.2.3 How to configure Nokia phone?

If you have Series 40 (S40) phone, it should work using either Bluetooth or USB cable.

For Bluetooth connection, `bluephonet` *Connection* is always the right choice with Bluetooth address of phone as *Device*.

For USB cable choosing the right connection type is more tricky and depends on generation of your phone. Newest phones usually work with `dku2` and the older ones with `dlr3` as *Connection*.

Should you have old phone with serial cable (and USB to serial converter), `fbus` *Connection* is the right one.

See also:

Gammu Configuration File

Example configuration for Bluetooth:

```
[gammu]
device = 11:22:33:44:55:66 # Bluetooth address of your phone
connection = bluephonet
```

Example configuration for newer phones:

```
[gammu]
connection = dku2
```

Example configuration for older phones on Linux:

```
[gammu]
device = /dev/ttyACM3
connection = dlr3
```

Example configuration for older phones on Windows:

```
[gammu]
device = COM12:
connection = dlr3
```

3.2.4 How to configure phone not listed above?

First check whether your phone is supported. In case it is, it most likely falls into one of above categories.

You can also find additional user experiences in Gammu Phones Database.

See also:

Is my phone supported?, *Gammu Configuration File*

3.3 Phone Support FAQ

3.3.1 Is my phone supported?

Generally any phone capable of AT commands or IrMC should be supported. Also any Nokia phone using Nokia OS should work. For Symbian please check separate topic. You can check other user experiences in Gammu Phones Database.

For information how to configure your phone, see *Configuring Gammu FAQ*.

See also:

Are Nokia phones supported?, *Are Symbian phones supported?*, *Are Android phones supported?*, *Are Blackberry phones supported?*, *Are iPhone phones supported?*, *Configuring Gammu FAQ*, *Gammu Configuration File*

3.3.2 Which phone is best supported?

It really depends on what you expect. If you want to use SMSD, this topic is covered in separate FAQ (see *Which phone is best for SMSD gateway?*). For backing up your contacts or calendar, most of Nokia (S40 or S60) phones should work as well as any other capable of AT commands. Gammu also supports wide range of extensions for Samsung, Motorola, Siemens or Sony-Ericsson phones.

See also:

Which phone is best for SMSD gateway?

3.3.3 Are Nokia phones supported?

It depends on used operating systems Series40 and older phones should work (see *How to configure Nokia phone?* for information how to configure them), Symbian based phones are covered in separate topic, check *Are Symbian phones supported?*.

3.3.4 Are Symbian phones supported?

You need to install applet to the phone to allow Gammu talk to it. For older phones (Symbian 9.0 and older), install gnapplet (see *Gnapplet Protocol*). Newer phones can use Python based applet called Series60-remote (see *Series60 Remote Protocol*). This option is supported since Gammu 1.29.90.

See also:

How to configure Symbian based phone?

3.3.5 Are Android phones supported?

Unfortunately no at the moment. Any help in this area is welcome.

See also:

See our issue tracker for more details.

3.3.6 Are Blackberry phones supported?

Unfortunately no at the moment. Any help in this area is welcome.

3.3.7 Are iPhone phones supported?

Unfortunately no at the moment. Any help in this area is welcome.

3.4 SMSD FAQ

3.4.1 Which databases does SMSD support?

SMSD natively supports MySQL and PostgreSQL. However it has also support for libdbi, which provides access to wide range of database engines (eg. SQLite, MS SQL Server, Sybase, Firebird,...). Unfortunately libdbi currently does not work natively on Microsoft Windows, so you can use it only on Unix platforms.

Since version 1.29.92, SMSD can also connect to any ODBC data source, so you should be able to connect to virtually any database engine using this standard.

See also:

SQL Service

3.4.2 Is there some user interface for SMSD?

Yes. You can use some of example interfaces distributed with gammu in `contrib` directory. Or there is full featured separate interface written in PHP called Kalkun.

3.4.3 Which phone is best for SMSD gateway?

Standard phones usually do not perform good when used long term as a modem. So it's always better to choose some GSM (GPRS, EDGE, UMTS) terminals/modems, which are designed to be used long for term in connection with computer.

The best option seem to be Siemens modems (eg. ES75/MC35i/MC55i). Slightly cheaper, while still good are modems made by Huawei (eg. E160/E220/E1750/...). We have heard also positive experiences with cheap modems from various Chinese resellers like DealExtreme or Alibaba.

See also:

You can check other user experiences in Gammu Phones Database.

3.4.4 The RunOnReceive script fails, how to fix that?

There can be various reasons why the script you've supplied as *RunOnReceive* has failed. You can usually find more information in the debug log (see *Reporting Bugs*). For example it can look like following:

```
gammu-smsd[9886]: Starting run on receive: ../received.sh
gammu-smsd[9875]: Process failed with exit status 2
gammu-smsd[9875]: Subprocess output: ../received.sh: 7: ../received.sh: Syntax error:␣
↪end of file unexpected (expecting "then")
```

From here it's quite easy to diagnose it's a syntax error in the script causing troubles.

Note: If process output is missing from your debug log, you're using older version, which didn't support this. Please upgrade to version newer than 1.36.4.

See also:

RunOnReceive Directive, *RunOnReceive*

3.4.5 Why received delivery reports are not matched to sent messages?

This can occasionally happen and can have several reasons.

- If reports are arriving late, you can adjust *DeliveryReportDelay*.
- If reports are coming from different SMSC than you're using for sending, set *SkipSMSCNumber*.
- If SMSD is unable to match sent message with delivery report, it might be due to missing international prefix in one of the numbers. Generally the best approach is to always send messages to international number (eg. use `+32485xxxxxx` instead of `0485xxxxxx`).

Note: If using Gammu 1.36.3 or newer, whenever first two cases happen, you will see hint to adjust the configuration in the log.

3.5 Python-gammu FAQ

3.5.1 Where can I download python-gammu?

The python-gammu project has been merged into Gammu, so you just need to grab Gammu and it includes python-gammu. Binaries for Windows are distributed separately.

3.5.2 How can I use python-gammu?

There are lot of examples shipped with Gammu, you can find them in the `examples` subdirectory.

See also:

python-gammu, *python-gammu Examples*

CHAPTER

FOUR

PYTHON-GAMMU

4.1 A taste of python-gammu

Python-gammu allows you to easily access the phone. Following code will connect to phone based on your Gammu configuration (usually stored in `~/.gammurc`) and gets network information from it:

```
import gammu
import sys

# Create state machine object
sm = gammu.StateMachine()

# Read ~/.gammurc
sm.ReadConfig()

# Connect to phone
sm.Init()

# Reads network information from phone
netinfo = sm.GetNetworkInfo()

# Print information
print 'Network name: %s' % netinfo['NetworkName']
print 'Network code: %s' % netinfo['NetworkCode']
print 'LAC: %s' % netinfo['LAC']
print 'CID: %s' % netinfo['CID']
```

4.2 API documentation

4.2.1 gammu – Mobile phone access

This module wraps all python-gammu functionality.

gammu.StateMachine

class gammu.**StateMachine**(*Locale*)

StateMachine object, that is used for communication with phone.

Parameters
Locale (*str*) – What locales to use for gammu error messages, default is `auto` which does autodetection according to user locales

AddCalendar(*Value*)

Adds calendar entry.

Parameters
Value (*dict*) – Calendar entry data, see *Calendar Object*

Returns
Location of newly created entry

Return type
int

AddCategory(*Type*, *Name*)

Adds category to phone.

Parameters

- **Type** (*str*) – Type of category to read, one of `ToDo`, `Phonebook`
- **Name** (*str*) – Category name

Returns
Location of created category

Return type
int

AddFilePart(*File*)

Adds file part to filesystem.

Parameters
File (*dict*) – File data, see *File Object*

Returns
File data for subsequent calls (Finished indicates transfer has been completed)

Return type
dict

AddFolder(*ParentFolderID*, *Name*)

Adds folder to filesystem.

Parameters

- **ParentFolderID** (*str*) – Folder where to create subfolder
- **Name** (*str*) – New folder name

Returns
New folder ID.

Return type
str

AddMemory(*Value*)

Adds memory (phonebooks or calls) entry.

Parameters
Value (*dict*) – Memory entry, see *Phonebook Object*

Returns
Location of created entry

Return type
int

AddSMS(*Value*)

Adds SMS to specified folder.

Parameters
Value (*dict*) – SMS data, see *SMS Object*

Returns
Tuple for location and folder.

Return type
tuple

AddSMSFolder(*Name*)

Creates SMS folder.

Parameters
Name (*str*) – Name of new folder

Returns
None

Return type
None

AddToDo(*Value*)

Adds ToDo in phone.

Parameters
Value (*dict*) – ToDo data, see *Todo Object*

Returns
Location of created entry

Return type
int

AnswerCall(*ID*, *All*)

Accept current incoming call.

Parameters

- **ID** (*int*) – ID of call
- **All** (*bool*) – Answer all calls? Defaults to True

Returns
None

Return type
None

`CancelAllDiverts`()

New in version 1.31.90.

Cancels all call diverts.

Returns
None

Return type
None

`CancelCall`(*ID*, *All*)

Deny current incoming call.

Parameters

- **ID** (*int*) – ID of call
- **All** (*bool*) – Cancel all calls? Defaults to True

Returns
None

Return type
None

`ConferenceCall`(*ID*)

Initiates conference call.

Parameters
ID (*int*) – ID of call

Returns
None

Return type
None

`DeleteAllCalendar`()

Deletes all calendar entries.

Returns
None

Return type
None

`DeleteAllMemory`(*Type*)

Deletes all memory (phonebooks or calls) entries of specified type.

Parameters
Type (*str*) – Memory type, one of `ME`, `SM`, `ON`, `DC`, `RC`, `MC`, `MT`, `FD`, `VM`

Returns
None

Return type
None

`DeleteAllToDo`()

Deletes all todo entries in phone.

Returns
None

Return type
None

DeleteCalendar(*Location*)

Deletes calendar entry.

Parameters
Location (*int*) – Calendar entry to delete

Returns
None

Return type
None

DeleteFile(*FileID*)

Deletes file from filesystem.

Parameters
FileID (*str*) – File to delete

Returns
None

Return type
None

DeleteFolder(*FolderID*)

Deletes folder on filesystem.

Parameters
FolderID (*str*) – Folder to delete

Returns
None

Return type
None

DeleteMemory(*Type*, *Location*)

Deletes memory (phonebooks or calls) entry.

Parameters

- **Type** (*str*) – Memory type, one of `ME`, `SM`, `ON`, `DC`, `RC`, `MC`, `MT`, `FD`, `VM`
- **Location** (*int*) – Location of entry to delete

Returns
None

Return type
None

DeleteSMS(*Folder*, *Location*)

Deletes SMS.

Parameters

- **Folder** (*int*) – Folder where to read entry (0 is emulated flat memory)
- **Location** (*int*) – Location of entry to delete

Returns
None

Return type
None

Note: In most cases you want to use Folder=0 as in this mode it will accept locations as GetNextSMS returns them.

`DeleteSMSFolder`(*ID*)

Deletes SMS folder.

Parameters
ID (*int*) – Index of folder to delete

Returns
None

Return type
None

`DeleteToDo`(*Location*)

Deletes ToDo entry in phone.

Parameters
Location (*int*) – Location of entry to delete

Returns
None

Return type
None

`DialService`(*Number*)

Dials number and starts voice call.

Parameters
Number (*str*) – Number to dial

Returns
None

Return type
None

`DialVoice`(*Number*, *ShowNumber*)

Dials number and starts voice call.

Parameters

- **Number** (*str*) – Number to dial
- **ShowNumber** (*bool or None*) – Identifies whether to enable CLIR (None = keep default phone settings). Default is None

Returns
None

Return type
None

EnterSecurityCode(*Type*, *Code*, *NewPIN*)

Enters security code.

Parameters

- **Type** (*str*) – What code to enter, one of PIN, PUK, PIN2, PUK2, Phone.
- **Code** (*str*) – Code value
- **NewPIN** (*str*) – New PIN value in case entering PUK

Returns
None

Return type
None

GetAlarm(*Location*)

Reads alarm set in phone.

Parameters
Location (*int*) – Which alarm to read. Many phone support only one alarm. Default is 1.

Returns
Alarm dict

Return type
dict

GetBatteryCharge()

Gets information about battery charge and phone charging state.

Returns
Dictionary containing information about battery state (BatteryPercent and ChargeState)

Return type
dict

GetCalendar(*Location*)

Retrieves calendar entry.

Parameters
Location (*int*) – Calendar entry to read

Returns
Dictionary with calendar values, see *Calendar Object*

Return type
dict

GetCalendarStatus()

Retrieves calendar status (number of used entries).

Returns
Dictionary with calendar status (Used)

Return type
dict

GetCallDivert(*Divert='AllTypes'*, *Type='All'*)

New in version 1.31.90.

Gets call diverts.

Parameters

- **Divert** (*Divert Type*) – When to do the divert.
- **Type** (*Call Type*) – What call types to divert.

Returns
List of call diverts.

Return type
Call Divert Objects

`GetCategory`(*Type*, *Location*)

Reads category from phone.

Parameters

- **Type** (*str*) – Type of category to read, one of `ToDo`, `Phonebook`
- **Location** (*int*) – Location of category to read

Returns
Category name as str

Return type
str

`GetCategoryStatus`(*Type*)

Reads category status (number of used entries) from phone.

Parameters
Type (*str*) – Type of category to read, one of `ToDo`, `Phonebook`

Returns
Dictionary containing information about category status (Used)

Return type
dict

`GetConfig`(*Section*)

Gets specified config section. Configuration consists of all params which can be defined in gammurc config file:

- Model
- DebugLevel
- Device
- Connection
- SyncTime
- LockDevice
- DebugFile
- StartInfo
- UseGlobalDebugFile

Parameters
Section (*int*) – Index of config section to read. Defaults to 0.

Returns
Dictionary containing configuration

Return type
dict

`GetDateTime()`

Reads date and time from phone.

Returns
Date and time from phone as datetime.datetime object.

Return type
datetime.datetime

`GetDisplayStatus()`

Acquired display status.

Returns
List of indicators displayed on display

Return type
list

`GetFilePart(File)`

Gets file part from filesystem.

Parameters
File (*dict*) – File data, see *File Object*

Returns
File data for subsequent calls (Finished indicates transfer has been completed), see *File Object*

Return type
dict

`GetFileSystemStatus()`

Acquires filesystem status.

Returns
Dictionary containing filesystem status (Used and Free)

Return type
dict

`GetFirmware()`

Reads firmware information from phone.

Returns
Tuple from version, date and numeric version.

Return type
tuple

`GetFolderListing(Folder, Start)`

Gets next filename from filesystem folder.

Parameters

- **Folder** (*str*) – Folder to list
- **Start** (*bool*) – Whether we're starting listing. Defaults to False.

Returns
File data as dict, see *File Object*

Return type
dict

`GetHardware()`

Gets hardware information about device.

Returns
Hardware information as str.

Return type
str

`GetIMEI()`

Reads IMEI/serial number from phone.

Returns
IMEI of phone as str.

Return type
str

`GetLocale()`

Gets locale information from phone.

Returns
Dictionary of locale settings. *SetLocale()* lists them all.

Return type
dict

`GetManufactureMonth()`

Gets month when device was manufactured.

Returns
Month of manufacture as str.

Return type
str

`GetManufacturer()`

Reads manufacturer from phone.

Returns
String with manufacturer name

Return type
str

`GetMemory(`*Type*, *Location*`)`

Reads entry from memory (phonebooks or calls). Which entry should be read is defined in entry.

Parameters
Type (*str*) – Memory type, one of `ME`, `SM`, `ON`, `DC`, `RC`, `MC`, `MT`, `FD`, `VM`

Returns
Memory entry as dict, see *Phonebook Object*

Return type
dict

`GetMemoryStatus`(*Type*)

Gets memory (phonebooks or calls) status (eg. number of used and free entries).

Parameters
Type (*str*) – Memory type, one of `ME`, `SM`, `ON`, `DC`, `RC`, `MC`, `MT`, `FD`, `VM`

Returns
Dictionary with information about memory (Used and Free)

Return type
dict

`GetModel`()

Reads model from phone.

Returns
Tuple containing gammu identification and real model returned by phone.

Return type
tuple

`GetNetworkInfo`()

Gets network information.

Returns
Dictionary with information about network (NetworkName, State, NetworkCode, CID and LAC)

Return type
dict

`GetNextCalendar`(*Start*, *Location*)

Retrieves calendar entry. This is useful for continuous reading of all calendar entries.

Parameters

- **Start** (*bool*) – Whether to start. This can not be used together with Location
- **Location** (*int*) – Last read location. This can not be used together with Start

Returns
Dictionary with calendar values, see *Calendar Object*

Return type
dict

`GetNextFileFolder`(*Start*)

Gets next filename from filesystem.

Parameters
Start (*bool*) – Whether we're starting listing. Defaults to False.

Returns
File data as dict, see *File Object*

Return type
dict

`GetNextMemory`(*Type*, *Start*, *Location*)

Reads entry from memory (phonebooks or calls). Which entry should be read is defined in entry. This can be easily used for reading all entries.

Parameters

- **Type** (*str*) – Memory type, one of ME, SM, ON, DC, RC, MC, MT, FD, VM
- **Start** (*bool*) – Whether to start. This can not be used together with Location
- **Location** (*int*) – Last read location. This can not be used together with Start

Returns
Memory entry as dict, see *Phonebook Object*

Return type
dict

GetNextRootFolder(*Folder*)

Gets next root folder from filesystem. Start with empty folder name.

Parameters
Folder (*str*) – Previous read folder. Start with empty folder name.

Returns
Structure with folder information

GetNextSMS(*Folder*, *Start*, *Location*)

Reads next (or first if start set) SMS message. This might be faster for some phones than using *GetSMS()* for each message.

Parameters

- **Folder** (*int*) – Folder where to read entry (0 is emulated flat memory)
- **Start** (*bool*) – Whether to start. This can not be used together with Location
- **Location** (*int*) – Location last read entry. This can not be used together with Start

Returns
Dictionary with SMS data, see *SMS Object*

Return type
dict

GetNextToDo(*Start*, *Location*)

Reads ToDo from phone.

Parameters

- **Start** (*bool*) – Whether to start. This can not be used together with Location
- **Location** (*int*) – Last read location. This can not be used together with Start

Returns
Dictionary with ToDo values, see *Todo Object*

Return type
dict

GetOriginalIMEI()

Gets original IMEI from phone.

Returns
Original IMEI of phone as string.

Return type
str

`GetPPM`()

Gets PPM (Post Programmable Memory) from phone.

Returns
PPM as string

Return type
str

`GetProductCode`()

Gets product code of device.

Returns
Product code as string.

Return type
str

`GetSIMIMSI`()

Gets SIM IMSI from phone.

Returns
SIM IMSI as string

Return type
str

`GetSMS`(*Folder*, *Location*)

Reads SMS message.

Parameters

- **Folder** (*int*) – Folder where to read entry (0 is emulated flat memory)
- **Location** (*int*) – Location of entry to read

Returns
Dictionary with SMS data, see *SMS Object*

Return type
dict

`GetSMSC`(*Location*)

Gets SMS Service Center number and SMS settings.

Parameters
Location (*int*) – Location of entry to read. Defaults to 1

Returns
Dictionary with SMSC information, see *SMSC Object*

Return type
dict

`GetSMSFolders`()

Returns SMS folders information.

Returns
List of SMS folders.

Return type
list

`GetSMSStatus()`

Gets information about SMS memory (read/unread/size of memory for both SIM and phone).

Returns
Dictionary with information about phone memory (SIMUnRead, SIMUsed, SIMSize, PhoneUnRead, PhoneUsed, PhoneSize and TemplatesUsed)

Return type
dict

`GetSecurityStatus()`

Queries whether some security code needs to be entered.

Returns
String indicating which code needs to be entered or None if none is needed

Return type
str

`GetSignalQuality()`

Reads signal quality (strength and error rate).

Returns
Dictionary containing information about signal state (SignalStrength, SignalPercent and BitErrorRate)

Return type
dict

`GetSpeedDial(`*Location*`)`

Gets speed dial.

Parameters
Location (*int*) – Location of entry to read

Returns
Dictionary with speed dial (Location, MemoryLocation, MemoryNumberID, MemoryType)

Return type
dict

`GetToDo(`*Location*`)`

Reads ToDo from phone.

Parameters
Location (*int*) – Location of entry to read

Returns
Dictionary with ToDo values, see *Todo Object*

Return type
dict

`GetToDoStatus()`

Gets status of ToDos (count of used entries).

Returns
Dictionary of status (Used)

Return type
dict

`HoldCall`(*ID*)

Holds call.

Parameters
ID (*int*) – ID of call

Returns
None

Return type
None

`Init`(*Replies*)

Initialises the connection with phone.

Parameters
Replies (*int*) – Number of replies to wait for on each request. Defaults to 1. Higher value makes sense only on unreliable links.

Returns
None

Return type
None

`PressKey`(*Key*, *Press*)

Emulates key press.

Parameters

- **Key** (*str*) – What key to press
- **Press** (*bool*) – Whether to emulate press or release.

Returns
None

Return type
None

`ReadConfig`(*Section*, *Configuration*, *Filename*)

Reads specified section of gammurc

Parameters

- **Section** (*int*) – Index of config section to read. Defaults to 0.
- **Configuration** (*int*) – Index where config section will be stored. Defaults to Section.
- **Filename** (*str*) – Path to configuration file (otherwise it is autodetected).

Returns
None

Return type
None

`ReadDevice`(*Wait*)

Reads data from device. This should be used in busy wait loop in case you are waiting for incoming events on the device.

Parameters
Wait (*bool*) – Whether to wait, default is not to wait.

Returns
Number of bytes read

Return type
int

`Reset`(*Hard*)

Performs phone reset.

Parameters
Hard (*bool*) – Whether to make hard reset

Returns
None

Return type
None

`ResetPhoneSettings`(*Type*)

Resets phone settings.

Parameters
Type (*str*) – What to reset, one of `PHONE`, `UIF`, `ALL`, `DEV`, `FACTORY`

Returns
None

Return type
None

`SendDTMF`(*Number*)

Sends DTMF (Dual Tone Multi Frequency) tone.

Parameters
Number (*str*) – Number to dial

Returns
None

Return type
None

`SendFilePart`(*File*)

Sends file part to phone.

Parameters
File (*dict*) – File data, see *File Object*

Returns
File data for subsequent calls (Finished indicates transfer has been completed), see *File Object*

Return type
dict

`SendSMS`(*Value*)

Sends SMS.

Parameters
Value (*dict*) – SMS data, see *SMS Object*

Returns
Message reference as int

Return type
int

SendSavedSMS(*Folder*, *Location*)

Sends SMS saved in phone.

Parameters

- **Folder** (*int*) – Folder where to read entry (0 is emulated flat memory)
- **Location** (*int*) – Location of entry to send

Returns
Message reference as int

Return type
int

SetAlarm(*DateTime*, *Location*, *Repeating*, *Text*)

Sets alarm in phone.

Parameters

- **DateTime** (*datetime.datetime*) – When should alarm happen.
- **Location** (*int*) – Location of alarm to set. Defaults to 1.
- **Repeating** (*bool*) – Whether alarm should be repeating. Defaults to True.
- **Text** (*str*) – Text to be displayed on alarm. Defaults to empty.

Returns
None

Return type
None

SetAutoNetworkLogin()

Enables network auto login.

Returns
None

Return type
None

SetCalendar(*Value*)

Sets calendar entry

Parameters
Value (*dict*) – Calendar entry data, see *Calendar Object*

Returns
Location of set entry

Return type
int

SetConfig(*Section*, *Values*)

Sets specified config section.

Parameters

- **Section** (*int*) – Index of config section to modify

- **Values** (*dict*) – Config values, see *GetConfig()* for description of accepted

Returns
None

Return type
None

SetCallDivert(*Divert*, *Type*, *Number*, *Timeout=0*)

New in version 1.31.90.

Sets call divert.

Parameters

- **Divert** (*Divert Type*) – When to do the divert.
- **Type** (*Call Type*) – What call types to divert.
- **Number** (*str*) – Phone number where to divert.
- **Timeout** (*int*) – Optional timeout when divert happens.

Returns
None

Return type
None

SetDateTime(*Date*)

Sets date and time in phone.

Parameters
Date (*datetime.datetime*) – Date to set

Returns
None

Return type
None

SetDebugFile(*File*, *Global*)

Sets state machine debug file.

Parameters

- **File** (*mixed*) – File where to write debug stuff (as configured by *SetDebugLevel()*). Can be either None for no file, Python file object or filename.
- **Global** (*bool*) – Whether to use global debug structure (overrides File)

Returns
None

Return type
None

SetDebugLevel(*Level*)

Sets state machine debug level according to passed string. You need to configure output file using *SetDebugFile()* to activate it.

Parameters
Level (*str*) – name of debug level to use, currently one of: - nothing - text - textall - binary - errors - textdate - textalldate - errorsdate

Returns
None

Return type
None

`SetFileAttributes`(*Filename*, *ReadOnly*, *Protected*, *System*, *Hidden*)

Sets file attributes.

Parameters

- **Filename** (*str*) – File to modify
- **ReadOnly** (*bool*) – Whether file is read only. Default to False.
- **Protected** (*bool*) – Whether file is protected. Default to False.
- **System** (*bool*) – Whether file is system. Default to False.
- **Hidden** (*bool*) – Whether file is hidden. Default to False.

Returns
None

Return type
None

`SetIncomingCB`(*Enable*)

Gets network information from phone.

Parameters
Enable (*bool*) – Whether to enable notifications, default is True

Returns
None

Return type
None

`SetIncomingCall`(*Enable*)

Activates/deactivates noticing about incoming calls.

Parameters
Enable (*bool*) – Whether to enable notifications, default is True

Returns
None

Return type
None

`SetIncomingCallback`(*Callback*)

Sets callback function which is called whenever any (enabled) incoming event appears. Please note that you have to enable each event type by calling SetIncoming* functions.

The callback function needs to accept three parameters: StateMachine object, event type and it's data in dictionary.

Parameters
Callback (*function*) – callback function or None for disabling

Returns
None

Return type
None

`SetIncomingSMS`(*Enable*)

Enable/disable notification on incoming SMS.

Parameters
Enable (*bool*) – Whether to enable notifications, default is True

Returns
None

Return type
None

`SetIncomingUSSD`(*Enable*)

Activates/deactivates noticing about incoming USSDs (UnStructured Supplementary Services).

Parameters
Enable (*bool*) – Whether to enable notifications, default is True

Returns
None

Return type
None

`SetLocale`(*DateSeparator*, *DateFormat*, *AMPMTime*)

Sets locale of phone.

Parameters

- **DateSeparator** (*str*) – Date separator.
- **DateFormat** (*str*) – Date format, one of `DDMMYYYY`, `MMDDYYYY`, `YYYYMMDD`
- **AMPMTime** (*bool*) – Whether to use AM/PM time.

Returns
None

Return type
None

`SetMemory`(*Value*)

Sets memory (phonebooks or calls) entry.

Parameters
Value (*dict*) – Memory entry, see *Phonebook Object*

Returns
Location of created entry

Return type
int

`SetSMS`(*Value*)

Sets SMS.

Parameters
Value (*dict*) – SMS data, see *SMS Object*

Returns
Tuple for location and folder.

Return type
tuple

`SetSMSC`(*Value*)

Sets SMS Service Center number and SMS settings.

Parameters
Value (*dict*) – SMSC information, see *SMSC Object*

Returns
None

Return type
None

`SetSpeedDial`(*Value*)

Sets speed dial.

Parameters
Value (*dict*) – Speed dial data, see *GetSpeedDial()* for listing.

Returns
None

Return type
None

`SetToDo`(*Value*)

Sets ToDo in phone.

Parameters
Value (*dict*) – ToDo data, see *Todo Object*

Returns
Location of created entry

Return type
int

`SplitCall`(*ID*)

Splits call.

Parameters
ID (*int*) – ID of call

Returns
None

Return type
None

`SwitchCall`(*ID*, *Next*)

Switches call.

Parameters
ID (*int*) – ID of call

Returns
None

Return type
None

Terminate()

Terminates the connection with phone.

Returns
None

Return type
None

Abort()

Aborts current operation.

Returns
None

Return type
None

TransferCall(*ID*, *Next*)

Transfers call.

Parameters
ID (*int*) – ID of call

Returns
None

Return type
None

UnholdCall(*ID*)

Unholds call.

Parameters
ID (*int*) – ID of call

Returns
None

Return type
None

Generic functions

gammu.**Version**()

Get version information.

Returns
Tuple of version information - Gammu runtime version, python-gammu version, build time Gammu version.

Return type
tuple

Debugging configuration

gammu.**SetDebugFile**(*File*)

Sets global debug file.

Parameters
File (*mixed*) – File where to write debug stuff (as configured by *SetDebugLevel()*). Can be either None for no file, Python file object or filename.

Returns
None

Return type
None

gammu.**SetDebugLevel**(*Level*)

Sets global debug level according to passed string. You need to configure output file using *SetDebugFile()* to activate it.

Parameters
Level (*str*) – name of debug level to use, currently one of:

- nothing
- text
- textall
- binary
- errors
- textdate
- textalldate
- errorsdate

Returns
None

Return type
None

Message processing

gammu.**LinkSMS**(*Messages*, *EMS*)

Links multi part SMS messages.

Parameters

- **Messages** (*list*) – List of messages to link, see *SMS Object*
- **EMS** (*bool*) – Whether to detect ems, defaults to True

Returns
List of linked messages, see *SMS Object*

Return type
list

gammu.**SMSCounter**(*Text*, *UDH='NoUDH'*, *Coding='Default'*)

Calculates number of SMS and free chars in SMS.

Parameters

- **Text** (*str*) – Message text
- **UDH** (*str*) – Message UDH
- **Coding** (*str*) – Message coding (eg. Unicode or Default)

Returns
Number of messages and number of free chars

Return type
tuple

New in version 1.29.90.

gammu.**DecodeSMS**(*Messages*, *EMS*)

Decodes multi part SMS message.

Parameters

- **Messages** (*list*) – Nessages to decode, see *SMS Object*
- **EMS** (*bool*) – Whether to use EMS, defaults to True

Returns
Multi part message information, see *SMS Info Object*

Return type
dict

gammu.**EncodeSMS**(*MessageInfo*)

Encodes multi part SMS message.

Parameters
MessageInfo (*dict*) – Description of message, see *SMS Info Object*

Returns
List of dictionaries with raw message, see *SMS Object*

Return type
dict

gammu.**DecodePDU**(*Data*, *SMSC=False*)

Parses PDU packet.

Parameters

- **Data** (*str*) – PDU data, need to be binary not hex encoded
- **SMSC** (*bool*) – Whether PDU includes SMSC.

Returns
Message data, see *SMS Object*

Return type
dict

Example:

```
gammu.DecodePDU(
    "0681678968986811000a815256455755001 0ff0d3bf67aed5ebbddeb1d7bed06".decode("hex")
)
```

gammu.**EncodePDU**(*SMS*, *Layout=Submit*)

Creates PDU packet.

Parameters

- **SMS** (*dict*) – SMS dictionary, see *SMS Object*
- **Layout** (*str*) – Layout (one of Submit, Deliver, StatusReport), Submit is default

Returns
Message data

Return type
str

New in version 1.27.93.

Encoding and decoding entries

gammu.**DecodeVCARD**(*Text*)

Decodes memory entry v from a string.

Parameters
Text (*str*) – String to decode

Returns
Memory entry, see *Phonebook Object*

Return type
dict

gammu.**EncodeVCARD**(*Entry*)

Encodes memory entry to a vCard.

Parameters
Entry (*dict*) – Memory entry, see *Phonebook Object*

Returns
String with vCard

Return type
str

gammu.**DecodeVCS**(*Text*)

Decodes todo/calendar entry v from a string.

Parameters
Text (*str*) – String to decode

Returns
Calendar or todo entry (whatever one was included in string), see *Calendar Object*, *Todo Object*

Return type
dict

gammu.**DecodeICS**(*Text*)

Decodes todo/calendar entry v from a string.

Parameters
Text (*str*) – String to decode

Returns
Calendar or todo entry (whatever one was included in string), see *Calendar Object*, *Todo Object*

Return type
dict

gammu.**EncodeVCALENDAR**(*Entry*)

Encodes calendar entry to a vCalendar.

Parameters
Entry (*dict*) – Calendar entry, see *Calendar Object*

Returns
String with vCalendar

Return type
str

gammu.**EncodeICALENDAR**(*Entry*)

Encodes calendar entry to a iCalendar.

Parameters
Entry (*dict*) – Calendar entry, see *Calendar Object*

Returns
String with iCalendar

Return type
str

gammu.**EncodeVTODO**(*Entry*)

Encodes todo entry to a vTodo.

Parameters
Entry (*dict*) – Todo entry, see *Todo Object*

Returns
String with vTodo

Return type
str

gammu.**EncodeITODO**(*Entry*)

Encodes todo entry to a iTodo.

Parameters
Entry (*dict*) – Todo entry, see *Todo Object*

Returns
String with vCard

Return type
str

Backup reading and writing

gammu.**SaveRingtone**(*Filename*, *Ringtone*, *Format*)

Saves ringtone into file.

Parameters

- **Filename** (*str*) – Name of file where ringote will be saved
- **Ringtone** (*dict*) – Ringtone to save
- **Format** (*str*) – One of ott, mid, rng, imy, wav, rttl

Returns
None

Return type
None

gammu.**SaveBackup**(*Filename*, *Backup*, *Format*)

Saves backup into file.

Parameters

- **Filename** (*str*) – Name of file to read backup from
- **Backup** (*dict*) – Backup data, see *ReadBackup()* for description
- **Format** (*str*) – File format to use (Auto, AutoUnicode, LMB, VCalendar, VCard, LDIF, ICS, Gammu, GammuUnicode, the default is AutoUnicode)

Returns
None

Return type
None

gammu.**ReadBackup**(*Filename*, *Format*)

Reads backup into file.

Parameters

- **Filename** (*str*) – Name of file where backup is stored
- **Format** (*str*) – File format to use (Auto, AutoUnicode, LMB, VCalendar, VCard, LDIF, ICS, Gammu, GammuUnicode, the default is AutoUnicode)

Returns

Dictionary of read entries, it contains following keys, each might be empty:

- IMEI
- Model
- Creator
- PhonePhonebook
- SIMPhonebook
- Calendar
- ToDo
- DateTime

Return type
dict

gammu.**SaveSMSBackup**(*Filename*, *Backup*)

Saves SMS backup into file.

Parameters

- **Filename** (*str*) – Name of file where to save SMS backup
- **Backup** (*list*) – List of messages to store

Returns
None

Return type
None

gammu.**ReadSMSBackup**(*Filename*)

Reads SMS backup into file.

Parameters
Filename (*str*) – Name of file where SMS backup is stored

Returns
List of messages read from file

Return type
list

Various data

gammu.**GSMNetworks**

Dictionary with GSM network codes.

gammu.**GSMCountries**

Dictionary with GSM country codes.

4.2.2 gammu.smsd – SMSD access

SMSD

class gammu.smsd.**SMSD**(*Config*)

SMSD main class, that is used for communication with phone.

You don't need to run the SMS daemon in the python script to control or ask it for status, this can be also done on separately running instances (gammu-smsd). All you need to do for this is to give same configuration file as that instance is using. For more infos look into *Gammu SMSD Overview*.

Parameters
Config (*string*) – Path to SMSD configuration file.

MainLoop(*MaxFailures*)

Runs SMS daemon main loop.

Please note that this will run until some serious error occurs or until terminated by *Shutdown()*.

Parameters
MaxFailures (*int*) – After how many init failures SMSD ends. Defaults to 0, what means never.

Returns
None

Return type
None

Shutdown()

Signals SMS daemon to stop.

Returns
None

Return type
None

GetStatus()

Returns SMSD status.

The following values are set in resulting dictionary:

Client

Client software name.

PhoneID

PhoneID which can be used for multiple SMSD setup.

IMEI

IMEI of currently connected phone.

Sent

Number of sent messages.

Received

Number of received messages.

Failed

Number of failed messages.

BatterPercent

Last battery state as reported by connected phone.

NetworkSignal

Last signal level as reported by connected phone.

Returns
Dict with status values

Return type
dict

InjectSMS(*Message*)

Injects SMS message into outgoing messages queue in SMSD.

Parameters
Message (list of *SMS Object*) – Message to inject (can be multipart)

Returns
ID of inserted message

Return type
string

4.2.3 gammu.data – Generic data usable with Gammu

gammu.data.**Connections**

Provides list of connection strings known to Gammu. They can be used for example when giving user a choice of connection string.

gammu.data.**MemoryValueTypes**

Provides list of types of memory entry values.

gammu.data.**CalendarTypes**

Provides list of calendar event types.

gammu.data.**CalendarValueTypes**

Provides list of types of calendar entry values.

gammu.data.**TodoPriorities**

Provides list of todo priorities.

gammu.data.**TodoValueTypes**

Provides list of types of todo entry values.

gammu.data.**InternationalPrefixes**

List of known international prefixes.

gammu.data.**Errors**

Mapping of text representation of errors to gammu error codes. Reverse to *ErrorNumbers*.

gammu.data.**ErrorNumbers**

Mapping of gammu error codes to text representation. Reverse to *Errors*.

4.2.4 gammu.worker - Asynchronous communication to phone.

Mostly you should use only *GammuWorker* class, others are only helpers which are used by this class.

class gammu.worker.**GammuCommand**(*command*, *params=None*, *percentage=100*)

Storage of single command for gammu.

get_command()

Returns command name.

get_params()

Returns command params.

get_percentage()

Returns percentage of current task.

class gammu.worker.**GammuTask**(*name*, *commands*)

Storage of task for gammu.

get_name()

Returns task name.

get_next()

Returns next command to be executed as *GammuCommand*.

class gammu.worker.**GammuThread**(*queue*, *config*, *callback*)

Thread for phone communication.

join(*timeout=None*)

Terminates thread and waits for it.

kill()

Forces thread end without emptying queue.

run()

Thread body, which handles phone communication. This should not be used from outside.

class gammu.worker.**GammuWorker**(*callback*)

Wrapper class for asynchronous communication with Gammu. It spawns own thread and then passes all commands to this thread. When task is done, caller is notified via callback.

abort()

Aborts any remaining operations.

configure(*config*)

Configures gammu instance according to config.

Parameters

config (*hash*) – Gammu configuration, same as *gammu.StateMachine.SetConfig()* accepts.

enqueue(*command*, *params=None*, *commands=None*)

Enqueues command or task.

Parameters

- **command** (*tuple of list of tuples*) – Command(s) to execute. Each command is tuple containing function name and it's parameters.
- **params** (*tuple or string*) – Parameters to command.
- **commands** (*list of tuples or strings*) – List of commands to execute. When this is not none, params are ignored and command is taken as task name.

enqueue_command(*command*, *params*)

Enqueues command.

Parameters

- **command** (*tuple of list of tuples*) – Command(s) to execute. Each command is tuple containing function name and it's parameters.
- **params** (*tuple or string*) – Parameters to command.

enqueue_task(*command*, *commands*)

Enqueues task.

Parameters

- **command** (*tuple of list of tuples*) – Command(s) to execute. Each command is tuple containing function name and it's parameters.
- **commands** (*list of tuples or strings*) – List of commands to execute.

initiate()

Connects to phone.

terminate(*timeout=None*)

Terminates phone connection.

exception gammu.worker.**InvalidCommand**(*value*)

Exception indicating invalid command.

gammu.worker.**check_worker_command**(*command*)

Checks whether command is valid.

Parameters

command (*string*) – Name of command.

4.2.5 gammu.exception – Gammu exception handling

exception gammu.**GSMError**

Generic class as parent for all Gammu exceptions. This is never raised directly, but should be used to catch any Gammu related exception.

exception gammu.**ERR_ABORTED**

Exception corresponding to gammu error ERR_ABORTED. Verbose error description: Operation aborted.

exception gammu.**ERR_BADFEATURE**

Exception corresponding to gammu error ERR_BADFEATURE. Verbose error description: Bad feature string in configuration.

exception gammu.**ERR_BUG**

Exception corresponding to gammu error ERR_BUG. Verbose error description: Nobody is perfect, some bug appeared in protocol implementation. Please contact authors.

exception gammu.**ERR_BUSY**

Exception corresponding to gammu error ERR_BUSY. Verbose error description: Command rejected because device was busy. Wait and restart.

exception gammu.**ERR_CANCELED**

Exception corresponding to gammu error ERR_CANCELED. Verbose error description: Transfer was canceled by phone, maybe you pressed cancel on phone.

exception gammu.**ERR_CANTOPENFILE**

Exception corresponding to gammu error ERR_CANTOPENFILE. Verbose error description: Can not open specified file.

exception gammu.**ERR_CORRUPTED**

Exception corresponding to gammu error ERR_CORRUPTED. Verbose error description: Corrupted data returned by phone.

exception gammu.**ERR_COULDNT_CONNECT**

Exception corresponding to gammu error ERR_COULDNT_CONNECT. Verbose error description: Could not connect to the server.

exception gammu.**ERR_COULDNT_RESOLVE**

Exception corresponding to gammu error ERR_COULDNT_RESOLVE. Verbose error description: Could not resolve the host name.

exception gammu.**ERR_DATACONVERTED**

Exception corresponding to gammu error ERR_DATACONVERTED. Verbose error description: Data were converted.

exception gammu.**ERR_DEVICEBUSY**

Exception corresponding to gammu error ERR_DEVICEBUSY. Verbose error description: Error opening device, it is already opened by other application.

exception gammu.**ERR_DEVICECHANGESPEEDERROR**

Exception corresponding to gammu error ERR_DEVICECHANGESPEEDERROR. Verbose error description: Error setting device speed. Maybe speed not supported.

exception gammu.**ERR_DEVICEDTRRTSERROR**

Exception corresponding to gammu error ERR_DEVICEDTRRTSERROR. Verbose error description: Error setting device DTR or RTS.

exception gammu.**ERR_DEVICELOCKED**

Exception corresponding to gammu error ERR_DEVICELOCKED. Verbose error description: Error opening device, it is locked.

exception gammu.**ERR_DEVICENODRIVER**

Exception corresponding to gammu error ERR_DEVICENODRIVER. Verbose error description: Error opening device. No required driver in operating system.

exception gammu.**ERR_DEVICENOPERMISSION**

Exception corresponding to gammu error ERR_DEVICENOPERMISSION. Verbose error description: Error opening device, you don't have permissions.

exception gammu.**ERR_DEVICENOTEXIST**

Exception corresponding to gammu error ERR_DEVICENOTEXIST. Verbose error description: Error opening device, it doesn't exist.

exception gammu.**ERR_DEVICENOTWORK**

Exception corresponding to gammu error ERR_DEVICENOTWORK. Verbose error description: Error opening device. Some hardware not connected/wrongly configured.

exception gammu.**ERR_DEVICEOPENERROR**

Exception corresponding to gammu error ERR_DEVICEOPENERROR. Verbose error description: Error opening device. Unknown, busy or no permissions.

exception gammu.**ERR_DEVICEPARITYERROR**

Exception corresponding to gammu error ERR_DEVICEPARITYERROR. Verbose error description: Can't set parity on the device.

exception gammu.**ERR_DEVICEREADERROR**

Exception corresponding to gammu error ERR_DEVICEREADERROR. Verbose error description: Error during reading from the device.

exception gammu.**ERR_DEVICEWRITEERROR**

Exception corresponding to gammu error ERR_DEVICEWRITEERROR. Verbose error description: Error writing to the device.

exception gammu.**ERR_DISABLED**

Exception corresponding to gammu error ERR_DISABLED. Verbose error description: Desired functionality has been disabled on compile time.

exception gammu.**ERR_EMPTY**

Exception corresponding to gammu error ERR_EMPTY. Verbose error description: Empty entry.

exception gammu.**ERR_EMPTYSMSC**

Exception corresponding to gammu error ERR_EMPTYSMSC. Verbose error description: No SMSC number given. Provide it manually or use the one configured in phone.

exception gammu.**ERR_FILEALREADYEXIST**

Exception corresponding to gammu error ERR_FILEALREADYEXIST. Verbose error description: File with specified name already exists.

exception gammu.**ERR_FILENOTEXIST**

Exception corresponding to gammu error ERR_FILENOTEXIST. Verbose error description: File with specified name doesn't exist.

exception gammu.**ERR_FILENOTSUPPORTED**

Exception corresponding to gammu error ERR_FILENOTSUPPORTED. Verbose error description: File format not supported by Gammu.

exception gammu.**ERR_FOLDERNOTEMPTY**

Exception corresponding to gammu error ERR_FOLDERNOTEMPTY. Verbose error description: Folder must be empty.

exception gammu.**ERR_FOLDERPART**

Exception corresponding to gammu error ERR_FOLDERPART. Verbose error description: Only part of folder has been listed.

exception gammu.**ERR_FRAMENOTREQUESTED**

Exception corresponding to gammu error ERR_FRAMENOTREQUESTED. Verbose error description: Frame not requested right now. See <https://wammu.eu/support/bugs/> for information how to report it.

exception gammu.**ERR_FULL**

Exception corresponding to gammu error ERR_FULL. Verbose error description: Memory full.

exception gammu.**ERR_GETTING_SMSC**

Exception corresponding to gammu error ERR_GETTING_SMSC. Verbose error description: Failed to get SMSC number from phone.

exception gammu.**ERR_GNAPPLETWRONG**

Exception corresponding to gammu error ERR_GNAPPLETWRONG. Verbose error description: Wrong GNAPPLET version in phone. Use version from currently used Gammu.

exception gammu.**ERR_INSIDEPHONEMENU**

Exception corresponding to gammu error ERR_INSIDEPHONEMENU. Verbose error description: You're inside phone menu (maybe editing?). Leave it and try again.

exception gammu.**ERR_INSTALL_NOT_FOUND**

Exception corresponding to gammu error ERR_INSTALL_NOT_FOUND. Verbose error description: Installation data not found, please consult debug log and/or documentation for more details.

exception gammu.**ERR_INVALIDDATA**

Exception corresponding to gammu error ERR_INVALIDDATA. Verbose error description: Invalid data given to phone.

exception gammu.**ERR_INVALIDDATETIME**

Exception corresponding to gammu error ERR_INVALIDDATETIME. Verbose error description: Invalid date or time specified.

exception gammu.**ERR_INVALIDLOCATION**

Exception corresponding to gammu error ERR_INVALIDLOCATION. Verbose error description: Invalid location. Maybe too high?

exception gammu.**ERR_MEMORY**

Exception corresponding to gammu error ERR_MEMORY. Verbose error description: Phone memory error, maybe it is read only.

exception gammu.**ERR_MOREMEMORY**

Exception corresponding to gammu error ERR_MOREMEMORY. Verbose error description: More memory required…

exception gammu.**ERR_NEEDANOTHERANSWER**

Exception corresponding to gammu error ERR_NEEDANOTHERANSWER. Verbose error description: Phone module need to send another answer frame.

exception gammu.**ERR_NETWORK_ERROR**

Exception corresponding to gammu error ERR_NETWORK_ERROR. Verbose error description: Network error.

exception gammu.**ERR_NONE**

Exception corresponding to gammu error ERR_NONE. Verbose error description: No error.

exception gammu.**ERR_NONE_SECTION**

Exception corresponding to gammu error ERR_NONE_SECTION. Verbose error description: No such section exists.

exception gammu.**ERR_NOSERVICE**

Exception corresponding to gammu error ERR_NOSERVICE. Verbose error description: Service configuration is missing.

exception gammu.**ERR_NOSIM**

Exception corresponding to gammu error ERR_NOSIM. Verbose error description: Can not access SIM card.

exception gammu.**ERR_NOTCONNECTED**

Exception corresponding to gammu error ERR_NOTCONNECTED. Verbose error description: Phone is not connected.

exception gammu.**ERR_NOTIMPLEMENTED**

Exception corresponding to gammu error ERR_NOTIMPLEMENTED. Verbose error description: Functionality not implemented. You are welcome to help authors with it.

exception gammu.**ERR_NOTRUNNING**

Exception corresponding to gammu error ERR_NOTRUNNING. Verbose error description: Service is not running.

exception gammu.**ERR_NOTSUPPORTED**

Exception corresponding to gammu error ERR_NOTSUPPORTED. Verbose error description: Function not supported by phone.

exception gammu.**ERR_OTHERCONNECTIONREQUIRED**

Exception corresponding to gammu error ERR_OTHERCONNECTIONREQUIRED. Verbose error description: Current connection type doesn't support called function.

exception gammu.**ERR_PERMISSION**

Exception corresponding to gammu error ERR_PERMISSION. Verbose error description: Operation not allowed by phone.

exception gammu.**ERR_PHONEOFF**

Exception corresponding to gammu error ERR_PHONEOFF. Verbose error description: Phone is disabled and connected to charger.

exception gammu.**ERR_PHONE_INTERNAL**

Exception corresponding to gammu error ERR_PHONE_INTERNAL. Verbose error description: Internal phone error.

exception gammu.**ERR_READ_ONLY**

Exception corresponding to gammu error ERR_READ_ONLY. Verbose error description: Entry is read only.

exception gammu.**ERR_SECURITYERROR**

Exception corresponding to gammu error ERR_SECURITYERROR. Verbose error description: Security error. Maybe no PIN?

exception gammu.**ERR_SHOULDBEFILE**

Exception corresponding to gammu error ERR_SHOULDBEFILE. Verbose error description: You have to give file name and not folder name.

exception gammu.**ERR_SHOULDBEFOLDER**

Exception corresponding to gammu error ERR_SHOULDBEFOLDER. Verbose error description: You have to give folder name and not file name.

exception gammu.**ERR_SOURCENOTAVAILABLE**

Exception corresponding to gammu error ERR_SOURCENOTAVAILABLE. Verbose error description: Some functions not available for your system (disabled in config or not implemented).

exception gammu.**ERR_SPECIFYCHANNEL**

Exception corresponding to gammu error ERR_SPECIFYCHANNEL. Verbose error description: Bluetooth configuration requires channel option.

exception gammu.**ERR_TIMEOUT**

Exception corresponding to gammu error ERR_TIMEOUT. Verbose error description: No response in specified timeout. Probably phone not connected.

exception gammu.**ERR_UNCONFIGURED**

Exception corresponding to gammu error ERR_UNCONFIGURED. Verbose error description: Gammu is not configured.

exception gammu.**ERR_UNKNOWN**

Exception corresponding to gammu error ERR_UNKNOWN. Verbose error description: Unknown error.

exception gammu.**ERR_UNKNOWNCONNECTIONTYPESTRING**

Exception corresponding to gammu error ERR_UNKNOWNCONNECTIONTYPESTRING. Verbose error description: Unknown connection type string. Check config file.

exception gammu.**ERR_UNKNOWNFRAME**

Exception corresponding to gammu error ERR_UNKNOWNFRAME. Verbose error description: Unknown frame. See <https://wammu.eu/support/bugs/> for information how to report it.

exception gammu.**ERR_UNKNOWNMODELSTRING**

Exception corresponding to gammu error ERR_UNKNOWNMODELSTRING. Verbose error description: Unknown model type string. Check config file.

exception gammu.**ERR_UNKNOWNRESPONSE**

Exception corresponding to gammu error ERR_UNKNOWNRESPONSE. Verbose error description: Unknown response from phone. See <https://wammu.eu/support/bugs/> for information how to report it.

exception gammu.**ERR_USING_DEFAULTS**

Exception corresponding to gammu error ERR_USING_DEFAULTS. Verbose error description: Using default values.

exception gammu.**ERR_WORKINPROGRESS**

Exception corresponding to gammu error ERR_WORKINPROGRESS. Verbose error description: Function is currently being implemented. If you want to help, please contact authors.

exception gammu.**ERR_WRITING_FILE**

Exception corresponding to gammu error ERR_WRITING_FILE. Verbose error description: Error writing file to disk.

exception gammu.**ERR_WRONGCRC**

Exception corresponding to gammu error ERR_WRONGCRC. Verbose error description: CRC error.

exception gammu.**ERR_WRONGFOLDER**

Exception corresponding to gammu error ERR_WRONGFOLDER. Verbose error description: Wrong folder used.

4.2.6 Objects

For various (mostly historical) reasons, all objects you get from Gammu are not real objects but rather a dictionaries. This has quite a big impact of usability and will most likely change in the future.

All the objects basically map to C structures, so you might also refer to *libGammu* chapter.

SMS Object

Object describing single SMS message in a way GSM network handles is (140 bytes of data). You can construct it from *SMS Info Object* using *gammu.EncodeSMS()*.

Message dictionary can consist of following fields:

SMSC

SMSC information, see *SMSC Object*.

Number

Recipient number, needs to be set for sending.

Name

Name of the message, does not make any effect on sending, some phones might store it.

UDH

User defined headers for SMS, see *UDH Object*.

Text

Message text

`Folder`
: Folder where the message is stored

`Location`
: Location where the message is stored

`InboxFolder`
: Indication whether folder is an inbox

`DeliveryStatus`
: Message delivery status, used only for received messages

`ReplyViaSameSMSC`
: Flag indicating whether reply using same SMSC is requested

`Class`
: Message class

`MessageReference`
: Message reference number, used mostly to identify delivery reports

`ReplaceMessage`
: Id of message which this message is supposed to replace

`RejectDuplicates`
: Whether to reject duplicates

`Memory`
: Memory where the message is stored

`Type`
: Message type, one of:

- `Submit` - message to be send
- `Deliver` - delivered message
- `Status_Report` - when creating new message this will create submit message with request for delivery report

`Coding`
: Message encoding, one of:

- `Unicode_No_Compression` - unicode message which can contain any chars, but can be only 70 chars long
- `Unicode_Compression` - not supported by Gammu and most phones
- `Default_No_Compression` - message with GSM alphabet only, up to 160 chars long
- `Default_Compression` - not supported by Gammu and most phones
- `8bit` - for binary messages

`DateTime`
: Timestamp when the message was received or sent.

  Please note that most phones do no record timestamp of sent messages.

`SMSCDateTime`
: Timestamp when the message was at SMSC.

State
: Message state, one of:

- Sent
- UnSent
- Read
- UnRead

Examples:

```
# Simple message to send, using SMSC from phone
SMS_1 = {
    'Number': '123465',
    'SMSC': {'Location': 1},
    'Text': 'Hello world!',
}

# Class 0 (on display) message using custom SMSC number
SMS_2 = {
    'Number': '123465',
    'SMSC': {'Number': '+420987654321'},
    'Text': 'Hello world!',
    'Class': 0,
}
```

UDH Object

UDH dictionary can consist of following fields:

ID8bit
: 8-bit ID of the message, not required

ID16bit
: 16-bit ID of the message, not required

PartNumber
: Number of current part

AllParts
: Count of all message parts

Type
: UDH type, one of predefined strings:

- NoUDH
- ConcatenatedMessages
- ConcatenatedMessages16bit
- DisableVoice
- DisableFax
- DisableEmail
- EnableVoice

- `EnableFax`
- `EnableEmail`
- `VoidSMS`
- `NokiaRingtone`
- `NokiaRingtoneLong`
- `NokiaOperatorLogoLong`
- `NokiaCallerLogo`
- `NokiaWAP`
- `NokiaWAPLong`
- `NokiaCalendarLong`
- `NokiaProfileLong`
- `NokiaPhonebookLong`
- `UserUDH`

Text
: UDH content

Example:

```
UDH = {
    "ID8bit": 0xCD,
    "PartNumber": 1,
    "AllParts": 2,
    "Type": "ConcatenatedMessages",
}
```

SMSC Object

SMSC dictionary can consist of following fields:

Location
: Location where the SMSC is stored

Number
: SMSC number

Name
: Name of the SMSC configuration

DefaultNumber
: Default recipient number, ignored on most phones

Format
: Default message format, one of:

- `Text`
- `Pager`
- `Fax`

- `Email`

`Validity`

Default message validity as a string

- `NA` - validity not available
- `Max` - maximum validity allowed by network
- `nM`, `nH`, `nD`, `nW` - period defined in minutes, hours, days or weeks, eg. `3W`

Example:

```
SMSC = {
    "Location": 1,
    "Number": "+420987654321",
    "Format": "Text",
    "Validity": "Max",
}
```

SMS Info Object

Message info dictionary can consist of following fields:

`Unicode`

Whether to use Unicode for the message.

`ReplaceMessage`

Id of message which this message is supposed to replace

`Unknown`

Boolean flag indicating there was some part which Gammu could not decode.

`Class`

Message class

`Entries`

Actual message data, see *SMS Info Part Object*.

Example:

```
SMSINFO = {
    "Class": 1,
    "Entries": [
        {"ID": "Text", "Buffer": "This is a "},
        {"ID": "Text", "Buffer": "message", "Italic": True},
        {"ID": "Text", "Buffer": " from "},
        {"ID": "Text", "Buffer": "Gammu", "Bold": True},
    ],
}
```

SMS Info Part Object

Message component can consist of following fields:

`ID`

Identification of the part type:

- `Text`
- `ConcatenatedTextLong` - Contacenated SMS, when longer than 1 SMS.
- `ConcatenatedAutoTextLong` - Contacenated SMS, auto Default/Unicode coding.
- `ConcatenatedTextLong16bit`
- `ConcatenatedAutoTextLong16bit`
- `NokiaProfileLong` - Nokia profile = Name`` Ringtone`` ScreenSaver
- `NokiaPictureImageLong` - Nokia Picture Image + (text)
- `NokiaScreenSaverLong` - Nokia screen saver + (text)
- `NokiaRingtone` - Nokia ringtone - old SM2.0 format`` 1 SMS
- `NokiaRingtoneLong` - Nokia ringtone concatenated`` when very long
- `NokiaOperatorLogo` - Nokia 72x14 operator logo`` 1 SMS
- `NokiaOperatorLogoLong` - Nokia 72x14 op logo or 78x21 in 2 SMS
- `NokiaCallerLogo` - Nokia 72x14 caller logo`` 1 SMS
- `NokiaWAPBookmarkLong` - Nokia WAP bookmark in 1 or 2 SMS
- `NokiaWAPSettingsLong` - Nokia WAP settings in 2 SMS
- `NokiaMMSSettingsLong` - Nokia MMS settings in 2 SMS
- `NokiaVCARD10Long` - Nokia VCARD 1.0 - only name and default number
- `NokiaVCARD21Long` - Nokia VCARD 2.1 - all numbers + text
- `NokiaVCALENDAR10Long` - Nokia VCALENDAR 1.0 - can be in few sms
- `NokiaVTODOLong`
- `VCARD10Long`
- `VCARD21Long`
- `DisableVoice`
- `DisableFax`
- `DisableEmail`
- `EnableVoice`
- `EnableFax`
- `EnableEmail`
- `VoidSMS`
- `EMSSound10` - IMelody 1.0
- `EMSSound12` - IMelody 1.2
- `EMSSonyEricssonSound` - IMelody without header - SonyEricsson extension

- EMSSound10Long - IMelody 1.0 with UPI.
- EMSSound12Long - IMelody 1.2 with UPI.
- EMSSonyEricssonSoundLong - IMelody without header with UPI.
- EMSPredefinedSound
- EMSPredefinedAnimation
- EMSAnimation
- EMSFixedBitmap - Fixed bitmap of size 16x16 or 32x32.
- EMSVariableBitmap
- EMSVariableBitmapLong
- MMSIndicatorLong - MMS message indicator.
- WAPIndicatorLong
- AlcatelMonoBitmapLong - Variable bitmap with black and white colors
- AlcatelMonoAnimationLong - Variable animation with black and white colors
- AlcatelSMSTemplateName
- SiemensFile - Siemens OTA

Left
: Text formatting

Right
: Text formatting

Center
: Text formatting

Large
: Text formatting

Small
: Text formatting

Bold
: Text formatting

Italic
: Text formatting

Underlined
: Text formatting

Strikethrough
: Text formatting

Protected
: Whether message part should be protected (DRM)

Number
: Number to encode in message.

Ringtone
Ringtone to encode in message.

Bitmap
Bitmap to encode in message.

Bookmark
Bookmark to encode in message.

Settings
Settings to encode in message.

MMSIndicator
MMS indication to encode in message.

Phonebook
Phonebook entry to encode in message, see *Phonebook Object*.

Calendar
Calendar entry to encode in message, see *Calendar Object*.

ToDo
Todo entry to encode in message, see *Todo Object*.

File
File to encode in message, see *File Object*.

Buffer
String to encode in message.

Todo Object

Todo entry is a dictionary consisting of following fields:

Location
Location where the entry is stored

Type
Type of entry, one of:

- `REMINDER` - Reminder or Date
- `CALL` - Call
- `MEETING` - Meeting
- `BIRTHDAY` - Birthday or Anniversary or Special Occasion
- `MEMO` - Memo or Miscellaneous
- `TRAVEL` - Travel
- `VACATION` - Vacation
- `T_ATHL` - Training - Athletism
- `T_BALL` - Training - Ball Games
- `T_CYCL` - Training - Cycling
- `T_BUDO` - Training - Budo

- `T_DANC` - Training - Dance
- `T_EXTR` - Training - Extreme Sports
- `T_FOOT` - Training - Football
- `T_GOLF` - Training - Golf
- `T_GYM` - Training - Gym
- `T_HORS` - Training - Horse Race
- `T_HOCK` - Training - Hockey
- `T_RACE` - Training - Races
- `T_RUGB` - Training - Rugby
- `T_SAIL` - Training - Sailing
- `T_STRE` - Training - Street Games
- `T_SWIM` - Training - Swimming
- `T_TENN` - Training - Tennis
- `T_TRAV` - Training - Travels
- `T_WINT` - Training - Winter Games
- `ALARM` - Alarm
- `DAILY_ALARM` - Alarm repeating each day.

`Priority`

Entry priority, one of:

- `High`
- `Medium`
- `Low`
- `None`

`Entries`

Actual entries, see *Todo Entries Object*

Example:

```
TODO = {
    "Type": "MEMO",
    "Entries": [
        {
            "Type": "END_DATETIME",
            "Value": datetime.datetime.now() + datetime.timedelta(days=1),
        },
        {"Type": "TEXT", "Value": "Buy some milk"},
    ],
}
```

Todo Entries Object

Type
Type of entry, one of:

- `END_DATETIME` - Due date (Date).
- `COMPLETED` - Whether is completed (Number).
- `ALARM_DATETIME` - When should alarm be fired (Date).
- `SILENT_ALARM_DATETIME` - When should silent alarm be fired (Date).
- `TEXT` - Text of to do (Text).
- `DESCRIPTION` - Description of to do (Text).
- `LOCATION` - Location of to do (Text).
- `PRIVATE` - Whether entry is private (Number).
- `CATEGORY` - Category of entry (Number).
- `CONTACTID` - Related contact ID (Number).
- `PHONE` - Number to call (Text).
- `LUID` - IrMC LUID which can be used for synchronisation (Text).
- `LAST_MODIFIED` - Date and time of last modification (Date).
- `START_DATETIME` - Start date (Date).

Value
Actual value, corresponding type to Type field.

Calendar Object

Calendar entry is a dictionary consisting of following fields:

Location
Location where the entry is stored

Type
Type of entry, one of:

- `REMINDER` - Reminder or Date
- `CALL` - Call
- `MEETING` - Meeting
- `BIRTHDAY` - Birthday or Anniversary or Special Occasion
- `MEMO` - Memo or Miscellaneous
- `TRAVEL` - Travel
- `VACATION` - Vacation
- `T_ATHL` - Training - Athletism
- `T_BALL` - Training - Ball Games
- `T_CYCL` - Training - Cycling

- T_BUDO - Training - Budo
- T_DANC - Training - Dance
- T_EXTR - Training - Extreme Sports
- T_FOOT - Training - Football
- T_GOLF - Training - Golf
- T_GYM - Training - Gym
- T_HORS - Training - Horse Race
- T_HOCK - Training - Hockey
- T_RACE - Training - Races
- T_RUGB - Training - Rugby
- T_SAIL - Training - Sailing
- T_STRE - Training - Street Games
- T_SWIM - Training - Swimming
- T_TENN - Training - Tennis
- T_TRAV - Training - Travels
- T_WINT - Training - Winter Games
- ALARM - Alarm
- DAILY_ALARM - Alarm repeating each day.

Entries

Actual entries, see *Calendar Entries Object*

Example:

```
CAL = {
    "Type": "MEMO",
    "Entries": [
        {"Type": "START_DATETIME", "Value": datetime.datetime.now()},
        {
            "Type": "END_DATETIME",
            "Value": datetime.datetime.now() + datetime.timedelta(days=1),
        },
        {"Type": "LOCATION", "Value": "Home"},
        {"Type": "TEXT", "Value": "Relax for one day"},
    ],
}
```

Calendar Entries Object

Type
Type of entry, one of:

- START_DATETIME - Date and time of event start.
- END_DATETIME - Date and time of event end.
- TONE_ALARM_DATETIME - Alarm date and time.
- SILENT_ALARM_DATETIME - Date and time of silent alarm.
- TEXT - Text.
- DESCRIPTION - Detailed description.
- LOCATION - Location.
- PHONE - Phone number.
- PRIVATE - Whether this entry is private.
- CONTACTID - Related contact id.
- REPEAT_DAYOFWEEK - Repeat each x'th day of week.
- REPEAT_DAY - Repeat each x'th day of month.
- REPEAT_DAYOFYEAR - Repeat each x'th day of year.
- REPEAT_WEEKOFMONTH - Repeat x'th week of month.
- REPEAT_MONTH - Repeat x'th month.
- REPEAT_FREQUENCY - Repeating frequency.
- REPEAT_STARTDATE - Repeating start.
- REPEAT_STOPDATE - Repeating end.
- REPEAT_COUNT - Number of repetitions.
- LUID - IrMC LUID which can be used for synchronisation.
- LAST_MODIFIED - Date and time of last modification.

Value
Actual value, corresponding type to Type field.

Phonebook Object

Phonebook entry is a dictionary consisting of following fields:

Location
Location where the entry is stored

MemoryType
Memory where the message is stored

Entries
Actual entries, see *Phonebook Entries Object*

Example:

```
PBK = {
    "Location": 1000,
    "MemoryType": "ME",
    "Entries": [
        {"Type": "Number_General", "Value": "+420123456789"},
        {"Type": "Text_Name", "Value": "Stojan Jakotyc"},
    ],
}
```

Phonebook Entries Object

Type

Type of entry, one of:

- `Number_General` - General number. (Text)
- `Number_Mobile` - Mobile number. (Text)
- `Number_Fax` - Fax number. (Text)
- `Number_Pager` - Pager number. (Text)
- `Number_Other` - Other number. (Text)
- `Text_Note` - Note. (Text)
- `Text_Postal` - Complete postal address. (Text)
- `Text_Email` - Email. (Text)
- `Text_Email2` - Second email. (Text)
- `Text_URL` - URL (Text)
- `Date` - Date and time of last call. (Date)
- `Caller_Group` - Caller group. (Number)
- `Text_Name` - Name (Text)
- `Text_LastName` - Last name. (Text)
- `Text_FirstName` - First name. (Text)
- `Text_Company` - Company. (Text)
- `Text_JobTitle` - Job title. (Text)
- `Category` - Category. (Number, if -1 then text)
- `Private` - Whether entry is private. (Number)
- `Text_StreetAddress` - Street address. (Text)
- `Text_City` - City. (Text)
- `Text_State` - State. (Text)
- `Text_Zip` - Zip code. (Text)
- `Text_Country` - Country. (Text)
- `Text_Custom1` - Custom information 1. (Text)
- `Text_Custom2` - Custom information 2. (Text)

- Text_Custom3 - Custom information 3. (Text)
- Text_Custom4 - Custom information 4. (Text)
- RingtoneID - Ringtone ID. (Number)
- PictureID - Picture ID. (Number)
- Text_UserID - User ID. (Text)
- CallLength - Length of call (Number)
- Text_LUID - LUID - Unique Identifier used for synchronisation (Text)
- LastModified - Date of last modification (Date)
- Text_NickName - Nick name (Text)
- Text_FormalName - Formal name (Text)
- Text_PictureName - Picture name (on phone filesystem). (Text)
- PushToTalkID - Push-to-talk ID (Text)
- Number_Messaging - Favorite messaging number. (Text)
- Photo - Photo (Picture).
- SecondName - Second name. (Text)
- VOIP - VOIP address (Text).
- SIP - SIP address (Text).
- DTMF - DTMF (Text).
- Video - Video number. (Text)
- SWIS - See What I See address. (Text)
- WVID - Wireless Village user ID. (Text)
- NamePrefix - Name prefix (Text)
- NameSuffix - Name suffix (Text)

Location

Location for the field:

- Unknown - not define
- Home - home
- Work - work

Value

Actual value, corresponding type to Type field.

PictureType

Type of picture which is stored in Value field (only for Picture fields).

File Object

File is a dictionary consisting of following fields:

Used
: Number of bytes used by this file.

Name
: File name.

Folder
: Boolean value indicating whether this is a folder.

Level
: Depth of file on the filesystem.

Type
: File type, one of:

- `Other`
- `Java_JAR`
- `Image_JPG`
- `Image_BMP`
- `Image_GIF`
- `Image_PNG`
- `Image_WBMP`
- `Video_3GP`
- `Sound_AMR`
- `Sound_NRT` - DCT4 binary format
- `Sound_MIDI`
- `MMS`

ID_FullName
: Full file name including path.

Buffer
: Content of the file.

Modified
: Timestamp of last change

Protected
: Boolean value indicating whether file is protected (DRM).

ReadOnly
: Boolean value indicating whether file is read only.

Hidden
: Boolean value indicating whether file is hidden.

System
: Boolean value indicating whether file is system.

Pos

Current position of file upload

Finished

Boolean value indicating completed file transfer.

Example:

```
FILE = {
    "ID_FullName": PATH,
    "Name": os.path.basename(PATH),
    "Buffer": data,
    "Protected": 0,
    "ReadOnly": 0,
    "Hidden": 0,
    "System": 0,
    "Folder": 0,
    "Level": 0,
    "Type": "Other",
    "Finished": 0,
    "Pos": 0,
}
```

Divert Type

The divert type can have one of following values:

- `Busy` - Divert when busy.
- `NoAnswer` - Divert when not answered.
- `OutOfReach` - Divert when phone off or no coverage.
- `AllTypes` - Divert all calls without ringing.

Call Type

The call type for diverts can have one of following values:

- `Voice` - Voice calls.
- `Fax` - Fax calls.
- `Data` - Data calls.
- `All` - All calls.

Call Divert Objects

DivertType
When to do the divert, see *Divert Type*.

CallType
What call types to divert, see *Call Type*.

Number
Phone number where to divert.

Timeout
Timeout after which the divert will happen.

4.3 python-gammu Examples

See also:

Many examples are available in /examples directory in the python-gammu git repository.

4.3.1 Sending a message

```
#!/usr/bin/env python
# Sample script to show how to send SMS

from __future__ import print_function
import gammu
import sys

# Create object for talking with phone
state_machine = gammu.StateMachine()

# Optionally load config file as defined by first parameter
if len(sys.argv) > 2:
    # Read the configuration from given file
    state_machine.ReadConfig(Filename=sys.argv[1])
    # Remove file name from args list
    del sys.argv[1]
else:
    # Read the configuration (~/.gammurc)
    state_machine.ReadConfig()

# Check parameters
if len(sys.argv) != 2:
    print("Usage: sendsms.py [configfile] RECIPIENT_NUMBER")
    sys.exit(1)

# Connect to the phone
state_machine.Init()

# Prepare message data
```

(continues on next page)

(continued from previous page)

```
# We tell that we want to use first SMSC number stored in phone
message = {
    "Text": "python-gammu testing message",
    "SMSC": {"Location": 1},
    "Number": sys.argv[1],
}

# Actually send the message
state_machine.SendSMS(message)
```

4.3.2 Sending a long message

```
#!/usr/bin/env python
# Sample script to show how to send long (multipart) SMS

from __future__ import print_function
import gammu
import sys

# Create object for talking with phone
state_machine = gammu.StateMachine()

# Optionally load config file as defined by first parameter
if len(sys.argv) > 2:
    # Read the configuration from given file
    state_machine.ReadConfig(Filename=sys.argv[1])
    # Remove file name from args list
    del sys.argv[1]
else:
    # Read the configuration (~/.gammurc)
    state_machine.ReadConfig()

# Check parameters
if len(sys.argv) != 2:
    print("Usage: sendlongsms.py [configfile] RECIPIENT_NUMBER")
    sys.exit(1)

# Connect to the phone
state_machine.Init()


# Create SMS info structure
smsinfo = {
    "Class": -1,
    "Unicode": False,
    "Entries": [
        {
            "ID": "ConcatenatedTextLong",
            "Buffer": "Very long python-gammu testing message "
            "sent from example python script. "
```

(continues on next page)

(continued from previous page)

```
            "Very long python-gammu testing message "
            "sent from example python script. "
            "Very long python-gammu testing message "
            "sent from example python script. ",
        }
    ],
}

# Encode messages
encoded = gammu.EncodeSMS(smsinfo)

# Send messages
for message in encoded:
    # Fill in numbers
    message["SMSC"] = {"Location": 1}
    message["Number"] = sys.argv[1]

    # Actually send the message
    state_machine.SendSMS(message)
```

4.3.3 Initiating a voice call

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

from __future__ import print_function
import gammu
import sys

# Create object for talking with phone
state_machine = gammu.StateMachine()

# Read the configuration (~/.gammurc or from command line)
if len(sys.argv) > 2:
    state_machine.ReadConfig(Filename=sys.argv[1])
    del sys.argv[1]
else:
    state_machine.ReadConfig()

# Connect to the phone
state_machine.Init()

# Check whether we have a number to dial
if len(sys.argv) != 2:
    print("Usage: dialvoice.py NUMBER")
    sys.exit(1)

# Dial a number
state_machine.DialVoice(sys.argv[1])
```

4.3.4 Reading calendar from phone

```
#!/usr/bin/env python
# Example for reading calendar from phone

from __future__ import print_function
import gammu

# Create object for talking with phone
state_machine = gammu.StateMachine()

# Read the configuration (~/.gammurc)
state_machine.ReadConfig()

# Connect to the phone
state_machine.Init()

# Get number of calendar entries
status = state_machine.GetCalendarStatus()

remain = status["Used"]

start = True

while remain > 0:
    # Read the entry
    if start:
        entry = state_machine.GetNextCalendar(Start=True)
        start = False
    else:
        entry = state_machine.GetNextCalendar(Location=entry["Location"])
    remain = remain - 1

    # Display it
    print()
    print("%-20s: %d" % ("Location", entry["Location"]))
    print("%-20s: %s" % ("Type", entry["Type"]))
    for v in entry["Entries"]:
        print("%-20s: %s" % (v["Type"], str(v["Value"])))
```

CHAPTER

FIVE

LIBGAMMU

The libGammu library exposes all Gammu functionality for various phones in standard API. It can be used to do anything with your phone, however for easier tasks you might prefer to use Python and *python-gammu*.

If you intend to use libGammu in your application, all you should need is to `#include <gammu.h>` and then use Gammu functions. You can check docs/examples/ for some small example applications. You don't need real phone for testing, use *Dummy Driver* instead.

5.1 Hints for libGammu Novices

This is very short overview of libGammu usage. You will probably need to study *libGammu C API* to find out what functions you want to use.

5.1.1 Basic library usage

You need to include main header file:

```
#include <gammu.h>
```

To compile you need to pass flags from pkg-config:

```
pkg-config --cflags gammu
```

To link you need to pass from pkg-config:

```
pkg-config --libs gammu
```

Gammu stores all its data in a *GSM_StateMachine*. This structure is not public, so all you can define is a pointer to it:

```
GSM_StateMachine *state_machine;
```

You'll want to check for errors from time to time. Do it using a function something like this with help of *GSM_ErrorString()*:

```
void check_error(GSM_Error err)
{
  if (err == ERR_NONE) {
    return;
  }
  fprintf(stderr, "Gammu failure: %s\n", GSM_ErrorString(error));
```

(continues on next page)

(continued from previous page)

```
  exit(1);
}
```

As libGammu does interact with strings in your local encoding, it is good idea to initialize locales subsystem first (otherwise you would get broken non ASCII characters) by calling *GSM_InitLocales()*:

```
GSM_InitLocales(NULL);
```

You first need to allocate a state machine structure using *GSM_AllocStateMachine()*:

```
state_machine = GSM_AllocStateMachine();
```

Now think about the configuration file. To use the default `~/.gammurc`, do this:

```
INI_Section *cfg;

/* Find it */
error = GSM_FindGammuRC(&cfg, NULL);
check_error(error);

/* Read it */
error = GSM_ReadConfig(cfg, GSM_GetConfig(state_machine, 0), 0);
check_error(error);

/* Free allocated memory */
INI_Free(cfg);

/* We care onlu about first configuration */
GSM_SetConfigNum(s, 1);
```

OK, now initialise the connection (1 means number of replies you want to wait for in case of failure) by *GSM_InitConnection()*:

```
error = GSM_InitConnection(s, 1);
check_error(error);
```

Now you are ready to communicate with the phone, for example you can read manufacturer name by *GSM_GetManufacturer()*:

```
error = GSM_GetManufacturer(s, buffer);
check_error(error);
```

When you're finished, you need to disconnect and free allocated memory using *GSM_FreeStateMachine()*:

```
error = GSM_TerminateConnection(s);
check_error(error);

/* Free up used memory */
GSM_FreeStateMachine(s);
check_error(error);
```

There are also other *Examples*.

5.1.2 Compiling the code

To compile program using Gammu library, you need to pass include path to the compiler and library name and search path to the linker. This can be easiest achieved by using **pkg-config**. See following `Makefile` for example:

```
# Sample Makefile which can be used to build examples shipped with Gammu

CFLAGS=$(shell pkg-config --cflags --libs gammu-smsd) -Wall
LDFLAGS=$(shell pkg-config --cflags --libs gammu)

ALL=phone-info sms-send smsd

.PHONY: all clean

all: $(ALL)

clean:
    rm -f $(ALL)

%:%.c
    $(CC) $< $(CFLAGS) $(LDFLAGS) -o $@
```

5.1.3 Unicode

Gammu stores all strings internally in UCS-2-BE encoding (terminated by two zero bytes). This is used mostly for historical reasons and today the obvious choice would be `wchar_t`. To work with these strings, various functions are provided (*UnicodeLength()*, *DecodeUnicode()*, *EncodeUnicode()*, *CopyUnicodeString()*, etc.).

For printing on console you should use:

```
printf("%s\n", DecodeUnicodeConsole(unicode_string));
```

For giving string to some GUI toolkit:

```
printf("%s\n", DecodeUnicodeString(unicode_string));
```

Note: These functions differ only on platforms where console uses historically different character set than GUI, what effectively means only Microsoft Windows.

5.1.4 Debugging

You can either enabled debug logging globally or per state machine.

To enable global debugging use:

```
debug_info = GSM_GetGlobalDebug();
GSM_SetDebugFileDescriptor(stderr, FALSE, debug_info);
GSM_SetDebugLevel("textall", debug_info);
```

For per state machine configuration:

```
debug_info = GSM_GetDebug(s);
GSM_SetDebugGlobal(FALSE, debug_info);
GSM_SetDebugFileDescriptor(stderr, FALSE, debug_info);
GSM_SetDebugLevel("textall", debug_info);
```

5.1.5 Waiting for incoming events

If you expect some incoming events, you need to maintain communication with the phone. The best way it can be *GSM_ReadDevice()*. For example you can use following busy loop:

```
while (!gshutdown) {
        GSM_ReadDevice(s, TRUE);
}
```

5.2 Examples

All these examples are also available in docs/examples/ directory in Gammu sources.

5.2.1 Getting phone information

```
#include <gammu.h>
#include <stdlib.h>
#include <stdio.h>

GSM_StateMachine *s;
INI_Section *cfg;
GSM_Error error;
char buffer[100];

/* Function to handle errors */
void error_handler(void)
{
        if (error != ERR_NONE) {
                printf("ERROR: %s\n", GSM_ErrorString(error));
                if (GSM_IsConnected(s))
                        GSM_TerminateConnection(s);
                exit(error);
        }
}

int main(int argc UNUSED, char **argv UNUSED)
{
        GSM_Debug_Info *debug_info;

        /*
         * We don't need gettext, but need to set locales so that
         * charset conversion works.
         */
```

(continues on next page)

(continued from previous page)

```
        GSM_InitLocales(NULL);

        /* Enable global debugging to stderr */
        debug_info = GSM_GetGlobalDebug();
        GSM_SetDebugFileDescriptor(stderr, FALSE, debug_info);
        GSM_SetDebugLevel("textall", debug_info);

        /* Allocates state machine */
        s = GSM_AllocStateMachine();
        if (s == NULL)
                return 3;

        /*
         * Enable state machine debugging to stderr
         * Same could be achieved by just using global debug config.
         */
        debug_info = GSM_GetDebug(s);
        GSM_SetDebugGlobal(FALSE, debug_info);
        GSM_SetDebugFileDescriptor(stderr, FALSE, debug_info);
        GSM_SetDebugLevel("textall", debug_info);

        /*
         * Find configuration file (first command line parameter or
         * defaults)
         */
        error = GSM_FindGammuRC(&cfg, argc == 2 ? argv[1] : NULL);
        error_handler();

        /* Read it */
        error = GSM_ReadConfig(cfg, GSM_GetConfig(s, 0), 0);
        error_handler();

        /* Free config file structures */
        INI_Free(cfg);

        /* We have one valid configuration */
        GSM_SetConfigNum(s, 1);

        /* Connect to phone */
        /* 1 means number of replies you want to wait for */
        error = GSM_InitConnection(s, 1);
        error_handler();

        /* Here you can do some stuff with phone... */

        /* As an example we read some information about phone: */

        /* Manufacturer name */
        error = GSM_GetManufacturer(s, buffer);
        error_handler();
        printf("Manufacturer  : %s\n", buffer);
```

(continues on next page)

(continued from previous page)

```
        /* Model name */
        error = GSM_GetModel(s, buffer);
        error_handler();
        printf("Model         : %s (%s)\n",
                GSM_GetModelInfo(s)->model,
                buffer);

        /* Terminate connection */
        error = GSM_TerminateConnection(s);
        error_handler();

        /* Free up used memory */
        GSM_FreeStateMachine(s);

        return 0;
}

/* Editor configuration
 * vim: noexpandtab sw=8 ts=8 sts=8 tw=72:
 */
```

5.2.2 Reading SMS message

```
#include <gammu.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <signal.h>

GSM_StateMachine *s;
INI_Section *cfg;
GSM_Error error;
volatile GSM_Error sms_send_status;
volatile gboolean gshutdown = FALSE;

/* Function to handle errors */
void error_handler(void)
{
        if (error != ERR_NONE) {
                printf("ERROR: %s\n", GSM_ErrorString(error));
                if (GSM_IsConnected(s))
                        GSM_TerminateConnection(s);
                exit(error);
        }
}

/* Interrupt signal handler */
void interrupt(int sign)
{
        signal(sign, SIG_IGN);
```

(continues on next page)

(continued from previous page)

```
        gshutdown = TRUE;
}

int main(int argc UNUSED, char **argv UNUSED)
{
        GSM_Debug_Info *debug_info;
        gboolean start;
        GSM_MultiSMSMessage         sms;
        int i;

        /* Register signal handler */
        signal(SIGINT, interrupt);
        signal(SIGTERM, interrupt);

        /*
         * We don't need gettext, but need to set locales so that
         * charset conversion works.
         */
        GSM_InitLocales(NULL);

        /* Enable global debugging to stderr */
        debug_info = GSM_GetGlobalDebug();
        GSM_SetDebugFileDescriptor(stderr, TRUE, debug_info);
        GSM_SetDebugLevel("textall", debug_info);

        /* Allocates state machine */
        s = GSM_AllocStateMachine();
        if (s == NULL)
                return 3;

        /*
         * Enable state machine debugging to stderr
         * Same could be achieved by just using global debug config.
         */
        debug_info = GSM_GetDebug(s);
        GSM_SetDebugGlobal(FALSE, debug_info);
        GSM_SetDebugFileDescriptor(stderr, TRUE, debug_info);
        GSM_SetDebugLevel("textall", debug_info);

        /*
         * Find configuration file (first command line parameter or
         * defaults)
         */
        error = GSM_FindGammuRC(&cfg, argc == 2 ? argv[1] : NULL);
        error_handler();

        /* Read it */
        error = GSM_ReadConfig(cfg, GSM_GetConfig(s, 0), 0);
        error_handler();

        /* Free config file structures */
        INI_Free(cfg);
```

(continues on next page)

(continued from previous page)

```
        /* We have one valid configuration */
        GSM_SetConfigNum(s, 1);

        /* Connect to phone */
        /* 1 means number of replies you want to wait for */
        error = GSM_InitConnection(s, 1);
        error_handler();

        /* Read all messages */
        error = ERR_NONE;
        start = TRUE;
        sms.Number = 0;
        sms.SMS[0].Location = 0;
        sms.SMS[0].Folder = 0;
        while (error == ERR_NONE && !gshutdown) {
                error = GSM_GetNextSMS(s, &sms, start);
                if (error == ERR_EMPTY) break;
                error_handler();
                start = FALSE;

                /* Now we can do something with the message */
                for (i = 0; i < sms.Number; i++) {
                        printf("Location: %d, Folder: %d\n", sms.SMS[i].Location, sms.
↪SMS[i].Folder);
                        printf("Number: \"%s\"\n", DecodeUnicodeConsole(sms.SMS[i].
↪Number));
                        /*
                         * Decoding with GSM_DecodeMultiPartSMS is also an option here,
                         * but for simplicity we use this approach which will handle only
                         * text messages.
                         */
                        if (sms.SMS[i].Coding == SMS_Coding_8bit) {
                                printf("8-bit message, can not display\n");
                        } else {
                                printf("Text: \"%s\"\n", DecodeUnicodeConsole(sms.SMS[i].
↪Text));
                        }
                        printf("\n");
                }
        }

        /* Terminate connection */
        error = GSM_TerminateConnection(s);
        error_handler();

        /* Free up used memory */
        GSM_FreeStateMachine(s);

        return 0;
}
```

(continues on next page)

(continued from previous page)

```
/* Editor configuration
 * vim: noexpandtab sw=8 ts=8 sts=8 tw=72:
 */
```

5.2.3 Sending SMS message

```
#include <gammu.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <signal.h>

GSM_StateMachine *s;
INI_Section *cfg;
GSM_Error error;
volatile GSM_Error sms_send_status;
volatile gboolean gshutdown = FALSE;

/* Handler for SMS send reply */
void send_sms_callback (GSM_StateMachine *sm, int status, int MessageReference, void *␣
↪user_data)
{
        printf("Sent SMS on device: \"%s\"\n", GSM_GetConfig(sm, -1)->Device);
        if (status==0) {
                printf("..OK");
                sms_send_status = ERR_NONE;
        } else {
                printf("..error %i", status);
                sms_send_status = ERR_UNKNOWN;
        }
        printf(", message reference=%d\n", MessageReference);
}

/* Function to handle errors */
void error_handler(void)
{
        if (error != ERR_NONE) {
                printf("ERROR: %s\n", GSM_ErrorString(error));
                if (GSM_IsConnected(s))
                        GSM_TerminateConnection(s);
                exit(error);
        }
}

/* Interrupt signal handler */
void interrupt(int sign)
{
        signal(sign, SIG_IGN);
        gshutdown = TRUE;
}
```

(continues on next page)

(continued from previous page)

```
int main(int argc UNUSED, char **argv UNUSED)
{
        GSM_SMSMessage sms;
        GSM_SMSC PhoneSMSC;
        char recipient_number[] = "+1234567890";
        char message_text[] = "Sample Gammu message";
        GSM_Debug_Info *debug_info;
        int return_value = 0;

        /* Register signal handler */
        signal(SIGINT, interrupt);
        signal(SIGTERM, interrupt);

        /*
         * We don't need gettext, but need to set locales so that
         * charset conversion works.
         */
        GSM_InitLocales(NULL);

        /* Enable global debugging to stderr */
        debug_info = GSM_GetGlobalDebug();
        GSM_SetDebugFileDescriptor(stderr, TRUE, debug_info);
        GSM_SetDebugLevel("textall", debug_info);

        /* Prepare message */
        /* Cleanup the structure */
        memset(&sms, 0, sizeof(sms));
        /* Encode message text */
        EncodeUnicode(sms.Text, message_text, strlen(message_text));
        /* Encode recipient number */
        EncodeUnicode(sms.Number, recipient_number, strlen(recipient_number));
        /* We want to submit message */
        sms.PDU = SMS_Submit;
        /* No UDH, just a plain message */
        sms.UDH.Type = UDH_NoUDH;
        /* We used default coding for text */
        sms.Coding = SMS_Coding_Default_No_Compression;
        /* Class 1 message (normal) */
        sms.Class = 1;

        /* Allocates state machine */
        s = GSM_AllocStateMachine();
        if (s == NULL)
                return 3;

        /*
         * Enable state machine debugging to stderr
         * Same could be achieved by just using global debug config.
         */
        debug_info = GSM_GetDebug(s);
        GSM_SetDebugGlobal(FALSE, debug_info);
```

(continues on next page)

(continued from previous page)

```
        GSM_SetDebugFileDescriptor(stderr, TRUE, debug_info);
        GSM_SetDebugLevel("textall", debug_info);

        /*
         * Find configuration file (first command line parameter or
         * defaults)
         */
        error = GSM_FindGammuRC(&cfg, argc == 2 ? argv[1] : NULL);
        error_handler();

        /* Read it */
        error = GSM_ReadConfig(cfg, GSM_GetConfig(s, 0), 0);
        error_handler();

        /* Free config file structures */
        INI_Free(cfg);

        /* We have one valid configuration */
        GSM_SetConfigNum(s, 1);

        /* Connect to phone */
        /* 1 means number of replies you want to wait for */
        error = GSM_InitConnection(s, 1);
        error_handler();

        /* Set callback for message sending */
        /* This needs to be done after initiating connection */
        GSM_SetSendSMSStatusCallback(s, send_sms_callback, NULL);

        /* We need to know SMSC number */
        PhoneSMSC.Location = 1;
        error = GSM_GetSMSC(s, &PhoneSMSC);
        error_handler();

        /* Set SMSC number in message */
        CopyUnicodeString(sms.SMSC.Number, PhoneSMSC.Number);

        /*
         * Set flag before callind SendSMS, some phones might give
         * instant response
         */
        sms_send_status = ERR_TIMEOUT;

        /* Send message */
        error = GSM_SendSMS(s, &sms);
        error_handler();

        /* Wait for network reply */
        while (!gshutdown) {
                GSM_ReadDevice(s, TRUE);
                if (sms_send_status == ERR_NONE) {
                        /* Message sent OK */
```

(continues on next page)

(continued from previous page)

```
                        return_value = 0;
                        break;
                }
                if (sms_send_status != ERR_TIMEOUT) {
                        /* Message sending failed */
                        return_value = 100;
                        break;
                }
        }

        /* Terminate connection */
        error = GSM_TerminateConnection(s);
        error_handler();

        /* Free up used memory */
        GSM_FreeStateMachine(s);

        return return_value;
}

/* Editor configuration
 * vim: noexpandtab sw=8 ts=8 sts=8 tw=72:
 */
```

5.2.4 Sending Long SMS message

```
#include <gammu.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <signal.h>

GSM_StateMachine *s;
INI_Section *cfg;
GSM_Error error;
volatile GSM_Error sms_send_status;
volatile gboolean gshutdown = FALSE;

/* Handler for SMS send reply */
void send_sms_callback (GSM_StateMachine *sm, int status, int MessageReference, void *␣
↪user_data)
{
        printf("Sent SMS on device: \"%s\"\n", GSM_GetConfig(sm, -1)->Device);
        if (status==0) {
                printf("..OK");
                sms_send_status = ERR_NONE;
        } else {
                printf("..error %i", status);
                sms_send_status = ERR_UNKNOWN;
        }
```

(continues on next page)

(continued from previous page)

```
        printf(", message reference=%d\n", MessageReference);
}

/* Function to handle errors */
void error_handler(void)
{
        if (error != ERR_NONE) {
                printf("ERROR: %s\n", GSM_ErrorString(error));
                if (GSM_IsConnected(s))
                        GSM_TerminateConnection(s);
                exit(error);
        }
}

/* Interrupt signal handler */
void interrupt(int sign)
{
        signal(sign, SIG_IGN);
        gshutdown = TRUE;
}

int main(int argc UNUSED, char **argv UNUSED)
{
        GSM_MultiSMSMessage SMS;
        int i;
        GSM_MultiPartSMSInfo SMSInfo;
        GSM_SMSC PhoneSMSC;
        char recipient_number[] = "+1234567890";
        char message_text[] = "Very long example Gammu message to show how to construct␣
↪concatenated messages using libGammu. Very long example Gammu message to show how to␣
↪construct concatenated messages using libGammu.";
        unsigned char message_unicode[(sizeof(message_text) + 1) * 2];
        GSM_Debug_Info *debug_info;
        int return_value = 0;

        /* Register signal handler */
        signal(SIGINT, interrupt);
        signal(SIGTERM, interrupt);

        /*
         * We don't need gettext, but need to set locales so that
         * charset conversion works.
         */
        GSM_InitLocales(NULL);

        /* Enable global debugging to stderr */
        debug_info = GSM_GetGlobalDebug();
        GSM_SetDebugFileDescriptor(stderr, TRUE, debug_info);
        GSM_SetDebugLevel("textall", debug_info);

        /*
         * Fill in SMS info structure which will be used to generate
```

(continues on next page)

(continued from previous page)

```
         * messages.
         */
        GSM_ClearMultiPartSMSInfo(&SMSInfo);
        /* Class 1 message (normal) */
        SMSInfo.Class = 1;
        /* Message will be consist of one part */
        SMSInfo.EntriesNum = 1;
        /* No unicode */
        SMSInfo.UnicodeCoding = FALSE;
        /* The part has type long text */
        SMSInfo.Entries[0].ID = SMS_ConcatenatedTextLong;
        /* Encode message text */
        EncodeUnicode(message_unicode, message_text, strlen(message_text));
        SMSInfo.Entries[0].Buffer = message_unicode;

        printf("%s\n", DecodeUnicodeConsole(SMSInfo.Entries[0].Buffer));

        /* Encode message into PDU parts */
        error = GSM_EncodeMultiPartSMS(debug_info, &SMSInfo, &SMS);
        error_handler();

        /* Allocates state machine */
        s = GSM_AllocStateMachine();
        if (s == NULL)
                return 3;

        /*
         * Enable state machine debugging to stderr
         * Same could be achieved by just using global debug config.
         */
        debug_info = GSM_GetDebug(s);
        GSM_SetDebugGlobal(FALSE, debug_info);
        GSM_SetDebugFileDescriptor(stderr, TRUE, debug_info);
        GSM_SetDebugLevel("textall", debug_info);

        /*
         * Find configuration file (first command line parameter or
         * defaults)
         */
        error = GSM_FindGammuRC(&cfg, argc == 2 ? argv[1] : NULL);
        error_handler();

        /* Read it */
        error = GSM_ReadConfig(cfg, GSM_GetConfig(s, 0), 0);
        error_handler();

        /* Free config file structures */
        INI_Free(cfg);

        /* We have one valid configuration */
        GSM_SetConfigNum(s, 1);
```

(continues on next page)

(continued from previous page)

```
        /* Connect to phone */
        /* 1 means number of replies you want to wait for */
        error = GSM_InitConnection(s, 1);
        error_handler();

        /* Set callback for message sending */
        /* This needs to be done after initiating connection */
        GSM_SetSendSMSStatusCallback(s, send_sms_callback, NULL);

        /* We need to know SMSC number */
        PhoneSMSC.Location = 1;
        error = GSM_GetSMSC(s, &PhoneSMSC);
        error_handler();

        /* Send message parts */
        for (i = 0; i < SMS.Number; i++) {
                /* Set SMSC number in message */
                CopyUnicodeString(SMS.SMS[i].SMSC.Number, PhoneSMSC.Number);

                /* Prepare message */
                /* Encode recipient number */
                EncodeUnicode(SMS.SMS[i].Number, recipient_number, strlen(recipient_
↪number));
                /* We want to submit message */
                SMS.SMS[i].PDU = SMS_Submit;

                /*
                 * Set flag before callind SendSMS, some phones might give
                 * instant response
                 */
                sms_send_status = ERR_TIMEOUT;

                /* Send message */
                error = GSM_SendSMS(s, &SMS.SMS[i]);
                error_handler();

                /* Wait for network reply */
                while (!gshutdown) {
                        GSM_ReadDevice(s, TRUE);
                        if (sms_send_status == ERR_NONE) {
                                /* Message sent OK */
                                return_value = 0;
                                break;
                        }
                        if (sms_send_status != ERR_TIMEOUT) {
                                /* Message sending failed */
                                return_value = 100;
                                break;
                        }
                }
        }
```

(continues on next page)

(continued from previous page)

```
        /* Terminate connection */
        error = GSM_TerminateConnection(s);
        error_handler();

        /* Free up used memory */
        GSM_FreeStateMachine(s);

        return return_value;
}

/* Editor configuration
 * vim: noexpandtab sw=8 ts=8 sts=8 tw=72:
 */
```

5.2.5 SMSD example

```
/* Simple C program to start SMSD without all magic normal gammu-smsd does */
#include <gammu-smsd.h>
#include <assert.h>

int main(int argc UNUSED, char **argv UNUSED)
{
        GSM_SMSDConfig *config;
    GSM_Error error;
    char *config_file = NULL; /* Use default compiled in path */

        /*
         * We don't need gettext, but need to set locales so that
         * charset conversion works.
         */
        GSM_InitLocales(NULL);

    /* Initialize configuration with program name */
        config = SMSD_NewConfig("smsd-example");
        assert(config != NULL);

    /* Read configuration file */
        error = SMSD_ReadConfig(config_file, config, TRUE);
        if (error != ERR_NONE) {
                printf("Failed to read config!\n");
                SMSD_FreeConfig(config);
                return 2;
        }

    /* Start main SMSD loop which processes messages */
    /*
     * This normally never terminates, you need to signal it
     * by SMSD_Shutdown(config); (for example from signal handler)
     * to make it stop.
     */
```

(continues on next page)

(continued from previous page)

```
        error = SMSD_MainLoop(config, FALSE, 0);
        if (error != ERR_NONE) {
                printf("Failed to run SMSD!\n");
                SMSD_FreeConfig(config);
                return 2;
        }

    /* Free configuration structure */
        SMSD_FreeConfig(config);

    return 0;
}
```

5.2.6 Custom configuration

```
/*
 * libGammu example to show how to set configuration manually instead
 * of parsing ~/.gammurc
 */
#include <gammu.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

GSM_StateMachine *s;
GSM_Error error;
char buffer[100];

/* Function to handle errors */
void error_handler(void)
{
        if (error != ERR_NONE) {
                printf("ERROR: %s\n", GSM_ErrorString(error));
                if (GSM_IsConnected(s))
                        GSM_TerminateConnection(s);
                exit(error);
        }
}

int main(int argc, char **argv)
{
        GSM_Debug_Info *debug_info;
        GSM_Config *cfg;

        if (argc != 4) {
                printf("Usage: custom-config DEVICE CONNECTION MODEL\n");
        }

        /*
         * We don't need gettext, but need to set locales so that
```

(continues on next page)

(continued from previous page)

```
         * charset conversion works.
         */
        GSM_InitLocales(NULL);

        /* Enable global debugging to stderr */
        debug_info = GSM_GetGlobalDebug();
        GSM_SetDebugFileDescriptor(stderr, FALSE, debug_info);
        GSM_SetDebugLevel("textall", debug_info);

        /* Allocates state machine */
        s = GSM_AllocStateMachine();
        if (s == NULL)
                return 3;

        /*
         * Enable state machine debugging to same config as global one.
         */
        debug_info = GSM_GetDebug(s);
        GSM_SetDebugGlobal(TRUE, debug_info);

        /*
         * Get pointer to config structure.
         */
        cfg = GSM_GetConfig(s, 0);

        /*
         * Set configuration, first freeing old values.
         */
        free(cfg->Device);
        cfg->Device = strdup(argv[1]);
        free(cfg->Connection);
        cfg->Connection = strdup(argv[2]);
        /* For historical reasons this is not a pointer */
        strcpy(cfg->Model, argv[3]);

        /* We have one valid configuration */
        GSM_SetConfigNum(s, 1);

        /* Connect to phone */
        /* 1 means number of replies you want to wait for */
        error = GSM_InitConnection(s, 1);
        error_handler();

        /* Here you can do some stuff with phone... */

        /* As an example we read some information about phone: */

        /* Manufacturer name */
        error = GSM_GetManufacturer(s, buffer);
        error_handler();
        printf("Manufacturer  : %s\n", buffer);
```

(continues on next page)

(continued from previous page)

```
        /* Model name */
        error = GSM_GetModel(s, buffer);
        error_handler();
        printf("Model         : %s (%s)\n",
                GSM_GetModelInfo(s)->model,
                buffer);

        /* Terminate connection */
        error = GSM_TerminateConnection(s);
        error_handler();

        /* Free up used memory */
        GSM_FreeStateMachine(s);

        return 0;
}

/* Editor configuration
 * vim: noexpandtab sw=8 ts=8 sts=8 tw=72:
 */
```

5.3 libGammu C API

5.3.1 Backup

GSM_Error **GSM_ReadSMSBackupFile**(const char *FileName, *GSM_SMS_Backup* *backup)

Reads SMS backup file.

Parameters

- **FileName** – file name
- **backup** – structure where backup will be stored

Returns

Error code

GSM_Error **GSM_AddSMSBackupFile**(const char *FileName, *GSM_SMS_Backup* *backup)

Adds data to SMS backup file.

Parameters

- **FileName** – file name
- **backup** – structure holding backup data

Returns

Error code

void **GSM_ClearSMSBackup**(*GSM_SMS_Backup* *backup)

Clears SMS backup structure

Parameters

- **backup** – structure where backup data will be stored

void **GSM_FreeSMSBackup**(*GSM_SMS_Backup* *backup)

Deallocates all members of SMS backup structure

Parameters

- **backup** – structure where backup data will be stored

GSM_Error **GSM_SaveBackupFile**(char *FileName, *GSM_Backup* *Backup, *GSM_BackupFormat* Format)

Save backup file.

Parameters

- **FileName** – Name of file (format is detected from it).
- **Backup** – structure holding backup data
- **Format** – Backup format.

Returns

Error code

GSM_BackupFormat **GSM_GuessBackupFormat**(const char *FileName, const *gboolean* UseUnicode)

Guesses backup format based on filename.

Parameters

- **FileName** – Name of backup filename.
- **UseUnicode** – Whether to prefer unicode variant when guessing.

Returns

Backup format on success -1 on error.

GSM_Error **GSM_ReadBackupFile**(const char *FileName, *GSM_Backup* *backup, *GSM_BackupFormat* Format)

Reads data from backup file.

Parameters

- **FileName** – Name of file (format is detected from it).
- **backup** – structure where backup data will be stored
- **Format** – Format of backup. For Gammu backups, unicode subformats are ignored.

Returns

Error code

void **GSM_ClearBackup**(*GSM_Backup* *backup)

Clears backup structure

Parameters

- **backup** – structure where backup data will be stored

void **GSM_FreeBackup**(*GSM_Backup* *backup)

Deallocates all members of backup structure

Parameters

- **backup** – structure where backup data will be stored

void **GSM_GetBackupFormatFeatures**(*GSM_BackupFormat* Format, *GSM_Backup_Info* *info)

Gets information about format features.

Parameters

- **Format** – Format of backup.
- **info** – Output information about backup features.

void **GSM_GetBackupFileFeatures**(*GSM_BackupFormat* Format, *GSM_Backup_Info* *info, *GSM_Backup* *backup)

Gets information about backup data features (resp. which data it contains).

Parameters

- **Format** – Format of backup.
- **info** – Output information about backup features.
- **backup** – Backup data to chech.

struct **GSM_SMS_Backup**

SMS backup data.

Public Members

GSM_SMSMessage ***SMS**[*GSM_BACKUP_MAX_SMS* + 1]

List of SMS messages.

struct **GSM_Backup**

Backup data.

Public Members

char **IMEI**[GSM_MAX_IMEI_LENGTH]

IMEI of phone which has been backed up

char **Model**[GSM_MAX_MODEL_LENGTH + GSM_MAX_VERSION_LENGTH]

Model of phone which has been backed up

char **Creator**[512]

Name of program which created backup

GSM_DateTime **DateTime**

Timestamp of backup

gboolean **DateTimeAvailable**

Whether timestamp is present

char **MD5Original**[100]

Original MD5 of backup from file

char **MD5Calculated**[100]

Calculated MD5 of backup

GSM_MemoryEntry ***PhonePhonebook**[GSM_BACKUP_MAX_PHONEPHONEBOOK + 1]

Phone phonebook

GSM_MemoryEntry ***SIMPhonebook**[GSM_BACKUP_MAX_SIMPHONEBOOK + 1]

SIM phonebook

GSM_CalendarEntry ***Calendar**[GSM_MAXCALENDARTODONOTES + 1]

Calendar

GSM_Bitmap ***CallerLogos**[GSM_BACKUP_MAX_CALLER + 1]

Caller logos

GSM_SMSC ***SMSC**[GSM_BACKUP_MAX_SMSC + 1]

SMS configuration

GSM_WAPBookmark ***WAPBookmark**[GSM_BACKUP_MAX_WAPBOOKMARK + 1]

WAP bookmarks

GSM_MultiWAPSettings ***WAPSettings**[GSM_BACKUP_MAX_WAPSETTINGS + 1]

WAP settings

GSM_MultiWAPSettings ***MMSSettings**[GSM_BACKUP_MAX_MMSSETTINGS + 1]

MMS settings

GSM_SyncMLSettings ***SyncMLSettings**[GSM_BACKUP_MAX_SYNCMLSETTINGS + 1]

SyncMC settings

GSM_ChatSettings ***ChatSettings**[GSM_BACKUP_MAX_CHATSETTINGS + 1]

Chat settings

GSM_Ringtone ***Ringtone**[GSM_BACKUP_MAX_RINGTONES + 1]

Ringtones

GSM_ToDoEntry ***ToDo**[GSM_MAXCALENDARTODONOTES + 1]

To do tasks

GSM_Profile ***Profiles**[GSM_BACKUP_MAX_PROFILES + 1]

Progiles

GSM_FMStation ***FMStation**[GSM_BACKUP_MAX_FMSTATIONS + 1]

FM stations

GSM_GPRSAccessPoint ***GPRSPoint**[GSM_BACKUP_MAX_GPRSPOINT + 1]

GPRS configurations

GSM_NoteEntry ***Note**[GSM_BACKUP_MAX_NOTE + 1]

Notes

GSM_Bitmap ***StartupLogo**

Statup logo

GSM_Bitmap ***OperatorLogo**

Operator logo

enum **GSM_BackupFormat**

Backup data.

Values:

enumerator **GSM_Backup_Auto**

Compatibility with old gboolean used instead of format.

File type is guessed for extension, non unicode format used for Gammu backup.

enumerator **GSM_Backup_AutoUnicode**

Compatibility with old gboolean used instead of format.

File type is guessed for extension, unicode format used for Gammu backup.

enumerator **GSM_Backup_LMB**

LMB format, compatible with Logo manager, can store phonebooks and logos.

enumerator **GSM_Backup_VCalendar**

vCalendar standard, can store todo and calendar entries.

enumerator **GSM_Backup_VCard**

vCard standard, can store phone phonebook entries.

enumerator **GSM_Backup_LDIF**

LDIF (LDAP Data Interchange Format), can store phone phonebook entries.

enumerator **GSM_Backup_ICS**

iCalendar standard, can store todo and calendar entries.

enumerator **GSM_Backup_Gammu**

Gammu own format can store almost anything from phone.

This is ASCII version of the format, Unicode strings are HEX encoded. Use GSM_Backup_GammuUCS2 instead if possible.

enumerator **GSM_Backup_GammuUCS2**

Gammu own format can store almost anything from phone.

This is UCS2-BE version of the format.

enumerator **GSM_Backup_VNote**

vNote standard, can store phone notes.

struct **GSM_Backup_Info**

Information about supported backup features.

GSM_BACKUP_MAX_SMS

Maximal number of SMSes in backup.

Todo:

This should not be hardcoded.

5.3.2 Bitmap

GSM_Error **GSM_GetBitmap**(*GSM_StateMachine* *s, *GSM_Bitmap* *Bitmap)

Gets bitmap from phone.

GSM_Error **GSM_SetBitmap**(*GSM_StateMachine* *s, *GSM_Bitmap* *Bitmap)

Sets bitmap in phone.

void **GSM_PrintBitmap**(FILE *file, *GSM_Bitmap* *bitmap)

Prints bitmap to file descriptor.

Parameters

- **file** – Where to print.
- **bitmap** – Bitmap to print.

GSM_Error **GSM_SaveBitmapFile**(char *FileName, *GSM_MultiBitmap* *bitmap)

Saves bitmap to file.

Parameters

- **FileName** – Where to save.
- **bitmap** – Bitmap to save.

Returns

Error code

GSM_Error **GSM_ReadBitmapFile**(char *FileName, *GSM_MultiBitmap* *bitmap)

Reads bitmap from file.

Parameters

- **FileName** – Where to load from.
- **bitmap** – Pointer where to load bitmap.

Returns

Error code

gboolean **GSM_IsPointBitmap**(*GSM_Bitmap* *bmp, int x, int y)

Checks whether point is set in bitmap.

Parameters

- **bmp** – Bitmap
- **x** – Horizontal coordinate.
- **y** – Vertical coordinate.

Returns

True if point is set.

void **GSM_SetPointBitmap**(*GSM_Bitmap* *bmp, int x, int y)

Sets point in bitmap.

Parameters

- **bmp** – Bitmap
- **x** – Horizontal coordinate.
- **y** – Vertical coordinate.

void **GSM_ClearPointBitmap**(*GSM_Bitmap* *bmp, int x, int y)

Clears point in bitmap.

Parameters

- **bmp** – Bitmap
- **x** – Horizontal coordinate.
- **y** – Vertical coordinate.

void **GSM_ClearBitmap**(*GSM_Bitmap* *bmp)

Clears bitmap.

Parameters

- **bmp** – Bitmap

enum **GSM_BinaryPicture_Types**

Binary picture types.

Values:

enumerator **PICTURE_BMP**

enumerator **PICTURE_GIF**

enumerator **PICTURE_JPG**

enumerator **PICTURE_ICN**

enumerator **PICTURE_PNG**

struct **GSM_BinaryPicture**

Binary picture data.

enum **GSM_Bitmap_Types**

Enum to handle all possible bitmaps, which are not saved in various filesystems.

Values:

enumerator **GSM_None**

enumerator **GSM_ColourStartupLogo_ID**

ID of static file in filesystem displayed during startup

enumerator **GSM_StartupLogo**

Static mono bitmap/ID of animated mono bitmap displayed during startup

enumerator **GSM_ColourOperatorLogo_ID**

ID of static file in filesystem displayed instead of operator name

enumerator **GSM_OperatorLogo**

Mono bitmap displayed instead of operator name

enumerator **GSM_ColourWallPaper_ID**

ID of static file in filesystem displayed as wallpaper

enumerator **GSM_CallerGroupLogo**

Mono bitmap assigned to caller group

enumerator **GSM_DealerNote_Text**

Text displayed during startup, which can't be removed from phone menu

enumerator **GSM_WelcomeNote_Text**

Text displayed during startup

enumerator **GSM_PictureImage**

Image defined in Smart Messaging specification

enumerator **GSM_PictureBinary**

Binary picture (BMP, GIF, etc.)

struct **GSM_Bitmap**

Structure for all possible bitmaps, which are not saved in various filesystems

Public Members

GSM_Bitmap_Types **Type**

For all: bitmap type

unsigned char **Location**

For caller group logos: number of group For startup logos: number of animated bitmap

unsigned char **Text**[2 * (GSM_BITMAP_TEXT_LENGTH + 1)]

For dealer/welcome note text: text For caller group logo: name of group For picture images: text assigned to it

gboolean **BitmapEnabled**

For caller group logo: TRUE, when logo is enabled in group

gboolean **DefaultName**

For caller group logo: TRUE, when group has default name

gboolean **DefaultBitmap**

For caller group logo: TRUE, when group has default bitmap

gboolean **DefaultRingtone**

For caller group logo: TRUE, when group has default ringtone

unsigned char **RingtoneID**

For caller group logo: ringtone ID. Phone model specific

int **PictureID**

For caller group logo: picture ID. Phone model specific

unsigned char **BitmapPoints**[GSM_BITMAP_SIZE]

For mono bitmaps: body of bitmap

size_t **BitmapHeight**

For mono bitmaps: height specified in pixels

size_t **BitmapWidth**

For mono bitmaps: width specified in pixels

char **NetworkCode**[10]

For operator logos: Network operator code

unsigned char **Sender**[2 * (GSM_MAX_NUMBER_LENGTH + 1)]

For picture images: number of sender

unsigned char **ID**

For colour bitmaps: ID

GSM_BinaryPicture **BinaryPic**

For binary pictures (GIF, BMP, etc.): frame and length

unsigned char **Name**[2 * (GSM_BITMAP_TEXT_LENGTH + 1)]

Bitmap name

struct **GSM_MultiBitmap**

Structure to handle more than one bitmap

Public Members

unsigned char **Number**

Number of bitmaps

GSM_Bitmap **Bitmap**[GSM_MAX_MULTI_BITMAP]

All bitmaps

GSM_Error **GSM_GetScreenshot**(*GSM_StateMachine* *s, *GSM_BinaryPicture* *picture)

Gets phone screenshot.

Parameters

- **s** – State machine pointer.
- **picture** – Structure which will hold data.

5.3.3 Calendar

void **GSM_CalendarFindDefaultTextTimeAlarmPhone**(*GSM_CalendarEntry* *entry, int *Text, int *Time, int *Alarm, int *Phone, int *EndTime, int *Location)

Finds inxedes of default entries.

GSM_Error **GSM_EncodeVTODO**(char *Buffer, const size_t buff_len, size_t *Length, const *GSM_ToDoEntry* *note, const *gboolean* header, const *GSM_VToDoVersion* Version)

Encodes vTodo to buffer.

Parameters

- **Buffer** – Storage for data.
- **buff_len** – **[in]** Size of output buffer.
- **Length** – Pointer to current position in data (will be incremented).
- **note** – Note to encode.
- **header** – Whether to include vCalendar header.
- **Version** – Format of vTodo to create.

Returns
Error code.

GSM_Error **GSM_EncodeVCALENDAR**(char *Buffer, const size_t buff_len, size_t *Length, *GSM_CalendarEntry* *note, const *gboolean* header, const *GSM_VCalendarVersion* Version)

Encodes vCalendar to buffer.

Parameters

- **Buffer** – Storage for data.
- **buff_len** – **[in]** Size of output buffer.
- **Length** – Pointer to current position in data (will be incremented).
- **note** – Note to encode.
- **header** – Whether to include vCalendar header.
- **Version** – Format of vCalendar to create.

Returns
Error code.

GSM_Error **GSM_DecodeVNOTE**(char *Buffer, size_t *Pos, *GSM_NoteEntry* *Note)

Decodes vNote from buffer.

Parameters

- **Buffer** – Buffer to decode.
- **Pos** – Current position in buffer (will be updated).
- **Note** – Storage for note entry.

Returns
Error code.

GSM_Error **GSM_EncodeVNTFile**(char *Buffer, const size_t buff_len, size_t *Length, *GSM_NoteEntry* *Note)

Encodes vNote to buffer.

Parameters

- **Buffer** – Storage for data.
- **buff_len** – **[in]** Size of output buffer.
- **Length** – Pointer to current position in data (will be incremented).
- **Note** – Note to encode.

Returns
Error code.

GSM_Error **GSM_DecodeVCALENDAR_VTODO**(*GSM_Debug_Info* *di, char *Buffer, size_t *Pos, *GSM_CalendarEntry* *Calendar, *GSM_ToDoEntry* *ToDo, *GSM_VCalendarVersion* CalVer, *GSM_VToDoVersion* ToDoVer)

Decodes vCalendar and vTodo buffer.

Parameters

- **di** – Pointer to debugging description.
- **Buffer** – Buffer to decode.
- **Pos** – Current position in buffer (will be updated).

- **Calendar** – Storage for calendar entry.
- **ToDo** – Storage for todo entry.
- **CalVer** – Format of vCalendar.
- **ToDoVer** – Format of vTodo.

Returns
Error code

gboolean **GSM_IsCalendarNoteFromThePast**(*GSM_CalendarEntry* *note)

Detects whether calendar note is in past.

Parameters

- **note** – Note to check.

Returns
Whether entry is in past.

GSM_Error **GSM_GetAlarm**(*GSM_StateMachine* *s, *GSM_Alarm* *Alarm)

Reads alarm set in phone.

Parameters

- **s** – State machine pointer.
- **Alarm** – Storage for alarm.

Returns
Error code

GSM_Error **GSM_SetAlarm**(*GSM_StateMachine* *s, *GSM_Alarm* *Alarm)

Sets alarm in phone.

Parameters

- **s** – State machine pointer.
- **Alarm** – Alarm to set.

Returns
Error code

GSM_Error **GSM_GetToDoStatus**(*GSM_StateMachine* *s, *GSM_ToDoStatus* *status)

Gets status of ToDos (count of used entries).

Parameters

- **s** – State machine pointer.
- **status** – Storage for todo status.

Returns
Error code

GSM_Error **GSM_GetToDo**(*GSM_StateMachine* *s, *GSM_ToDoEntry* *ToDo)

Reads ToDo from phone.

Parameters

- **s** – State machine pointer.
- **ToDo** – Storage for note.

Returns
Error code

GSM_Error **GSM_GetNextToDo**(*GSM_StateMachine* *s, *GSM_ToDoEntry* *ToDo, *gboolean* start)

Reads ToDo from phone.

Parameters

- **s** – State machine pointer.
- **ToDo** – Storage for note, if start is FALSE, should contain data from previous read (at least position).
- **start** – Whether we're doing initial read or continue in reading.

Returns
Error code

GSM_Error **GSM_SetToDo**(*GSM_StateMachine* *s, *GSM_ToDoEntry* *ToDo)

Sets ToDo in phone.

Parameters

- **s** – State machine pointer.
- **ToDo** – ToDo to set, should contain valid location.

Returns
Error code

GSM_Error **GSM_AddToDo**(*GSM_StateMachine* *s, *GSM_ToDoEntry* *ToDo)

Adds ToDo in phone.

Parameters

- **s** – State machine pointer.
- **ToDo** – ToDo to add.

Returns
Error code

GSM_Error **GSM_DeleteToDo**(*GSM_StateMachine* *s, *GSM_ToDoEntry* *ToDo)

Deletes ToDo entry in phone.

Parameters

- **s** – State machine pointer.
- **ToDo** – ToDo to delete, only location is actually used.

Returns
Error code

GSM_Error **GSM_DeleteAllToDo**(*GSM_StateMachine* *s)

Deletes all todo entries in phone.

Parameters

- **s** – State machine pointer.

Returns
Error code

GSM_Error **GSM_GetCalendarStatus**(*GSM_StateMachine* *s, *GSM_CalendarStatus* *Status)

Retrieves calendar status (number of used entries).

Parameters

- **s** – State machine pointer.
- **Status** – Storage for status.

Returns
Error code

GSM_Error **GSM_GetCalendar**(*GSM_StateMachine* *s, *GSM_CalendarEntry* *Note)

Retrieves calendar entry.

Parameters

- **s** – State machine pointer.
- **Note** – Storage for note.

Returns
Error code

GSM_Error **GSM_GetNextCalendar**(*GSM_StateMachine* *s, *GSM_CalendarEntry* *Note, *gboolean* start)

Retrieves calendar entry. This is useful for continuous reading of all calendar entries.

Parameters

- **s** – State machine pointer.
- **Note** – Storage for note, if start is FALSE, should contain data from previous read (at least position).
- **start** – Whether we're doing initial read or continue in reading.

Returns
Error code

GSM_Error **GSM_SetCalendar**(*GSM_StateMachine* *s, *GSM_CalendarEntry* *Note)

Sets calendar entry

Parameters

- **s** – State machine pointer.
- **Note** – New note values, needs to contain valid position.

Returns
Error code

GSM_Error **GSM_AddCalendar**(*GSM_StateMachine* *s, *GSM_CalendarEntry* *Note)

Adds calendar entry.

Parameters

- **s** – State machine pointer.
- **Note** – Note to add.

Returns
Error code

GSM_Error **GSM_DeleteCalendar**(*GSM_StateMachine* *s, *GSM_CalendarEntry* *Note)

Deletes calendar entry.

Parameters

- **s** – State machine pointer.
- **Note** – Note to delete, must contain position.

Returns
Error code

GSM_Error **GSM_DeleteAllCalendar**(*GSM_StateMachine* *s)

Deletes all calendar entries.

Parameters

- **s** – State machine pointer.

Returns
Error code

GSM_Error **GSM_GetCalendarSettings**(*GSM_StateMachine* *s, *GSM_CalendarSettings* *settings)

Reads calendar settings.

Parameters

- **s** – State machine pointer.
- **settings** – Storage for settings.

Returns
Error code

GSM_Error **GSM_SetCalendarSettings**(*GSM_StateMachine* *s, *GSM_CalendarSettings* *settings)

Sets calendar settings.

Parameters

- **s** – State machine pointer.
- **settings** – New calendar settings.

Returns
Error code

GSM_Error **GSM_GetNotesStatus**(*GSM_StateMachine* *s, *GSM_ToDoStatus* *status)

Retrieves notes status (number of used entries).

Parameters

- **s** – State machine pointer.
- **status** – Storage for status.

Returns
Error code

GSM_Error **GSM_GetNote**(*GSM_StateMachine* *s, *GSM_NoteEntry* *Note)

Retrieves notes entry.

Parameters

- **s** – State machine pointer.
- **Note** – Storage for note.

Returns
Error code

GSM_Error **GSM_GetNextNote**(*GSM_StateMachine* *s, *GSM_NoteEntry* *Note, *gboolean* start)

Retrieves note entry. This is useful for continuous reading of all notes entries.

Parameters

- **s** – State machine pointer.
- **Note** – Storage for note, if start is FALSE, should contain data from previous read (at least position).
- **start** – Whether we're doing initial read or continue in reading.

Returns
Error code

GSM_Error **GSM_SetNote**(*GSM_StateMachine* *s, *GSM_NoteEntry* *Note)

Sets note entry

Parameters

- **s** – State machine pointer.
- **Note** – New note values, needs to contain valid position.

Returns
Error code

GSM_Error **GSM_AddNote**(*GSM_StateMachine* *s, *GSM_NoteEntry* *Note)

Adds note entry.

Parameters

- **s** – State machine pointer.
- **Note** – Note to add.

Returns
Error code

GSM_Error **GSM_DeleteNote**(*GSM_StateMachine* *s, *GSM_NoteEntry* *Note)

Deletes note entry.

Parameters

- **s** – State machine pointer.
- **Note** – Note to delete, must contain position.

Returns
Error code

GSM_Error **GSM_DeleteAllNotes**(*GSM_StateMachine* *s)

Deletes all notes entries.

Parameters

- **s** – State machine pointer.

Returns
Error code

struct **GSM_CalendarSettings**

Calendar settings structure.

Public Members

int **StartDay**

Monday = 1, Tuesday = 2,...

int **AutoDelete**

0 = no delete, 1 = after day,...

struct **GSM_ToDoStatus**

Status of to do entries.

Public Members

int **Free**

Number of free positions.

int **Used**

Number of used positions.

struct **GSM_CalendarStatus**

Structure used for returning calendar status.

Public Members

int **Free**

Number of free positions.

int **Used**

Number of used positions.

enum **GSM_CalendarNoteType**

Enum defines types of calendar notes

Values:

enumerator **GSM_CAL_REMINDER**

Reminder or Date

enumerator **GSM_CAL_CALL**

Call

enumerator **GSM_CAL_MEETING**
Meeting

enumerator **GSM_CAL_BIRTHDAY**
Birthday or Anniversary or Special Occasion

enumerator **GSM_CAL_MEMO**
Memo or Miscellaneous

enumerator **GSM_CAL_TRAVEL**
Travel

enumerator **GSM_CAL_VACATION**
Vacation

enumerator **GSM_CAL_T_ATHL**
Training - Athletism

enumerator **GSM_CAL_T_BALL**
Training - Ball Games

enumerator **GSM_CAL_T_CYCL**
Training - Cycling

enumerator **GSM_CAL_T_BUDO**
Training - Budo

enumerator **GSM_CAL_T_DANC**
Training - Dance

enumerator **GSM_CAL_T_EXTR**
Training - Extreme Sports

enumerator **GSM_CAL_T_FOOT**
Training - Football

enumerator **GSM_CAL_T_GOLF**
Training - Golf

enumerator **GSM_CAL_T_GYM**
Training - Gym

enumerator **GSM_CAL_T_HORS**
Training - Horse Race

enumerator **GSM_CAL_T_HOCK**
Training - Hockey

enumerator **GSM_CAL_T_RACE**
Training - Races

enumerator **GSM_CAL_T_RUGB**
Training - Rugby

enumerator **GSM_CAL_T_SAIL**
Training - Sailing

enumerator **GSM_CAL_T_STRE**
Training - Street Games

enumerator **GSM_CAL_T_SWIM**
Training - Swimming

enumerator **GSM_CAL_T_TENN**
Training - Tennis

enumerator **GSM_CAL_T_TRAV**
Training - Travels

enumerator **GSM_CAL_T_WINT**
Training - Winter Games

enumerator **GSM_CAL_ALARM**
Alarm

enumerator **GSM_CAL_DAILY_ALARM**
Alarm repeating each day.

enumerator **GSM_CAL_SHOPPING**
Shopping

enum **GSM_CalendarType**
One value of calendar event.
Values:

enumerator **CAL_START_DATETIME**
Date and time of event start.

enumerator **CAL_END_DATETIME**
Date and time of event end.

enumerator **CAL_TONE_ALARM_DATETIME**

Alarm date and time.

enumerator **CAL_SILENT_ALARM_DATETIME**

Date and time of silent alarm.

enumerator **CAL_TEXT**

Text.

enumerator **CAL_DESCRIPTION**

Detailed description.

enumerator **CAL_LOCATION**

Location.

enumerator **CAL_PHONE**

Phone number.

enumerator **CAL_PRIVATE**

Whether this entry is private.

enumerator **CAL_CONTACTID**

Related contact id.

enumerator **CAL_REPEAT_DAYOFWEEK**

Repeat each x'th day of week.

enumerator **CAL_REPEAT_DAY**

Repeat each x'th day of month.

enumerator **CAL_REPEAT_DAYOFYEAR**

Repeat each x'th day of year.

enumerator **CAL_REPEAT_WEEKOFMONTH**

Repeat x'th week of month.

enumerator **CAL_REPEAT_MONTH**

Repeat x'th month.

enumerator **CAL_REPEAT_FREQUENCY**

Repeating frequency.

enumerator **CAL_REPEAT_STARTDATE**

Repeating start.

enumerator **CAL_REPEAT_STOPDATE**

Repeating end.

enumerator **CAL_REPEAT_COUNT**

Number of repetitions.

enumerator **CAL_LUID**

IrMC LUID which can be used for synchronisation.

enumerator **CAL_LAST_MODIFIED**

Date and time of last modification.

struct **GSM_SubCalendarEntry**

One value of calendar event.

Public Members

GSM_CalendarType **EntryType**

Type of value.

GSM_DateTime **Date**

Date and time of value, if applicable.

int **Number**

Number of value, if applicable.

GSM_Error **AddError**

During adding SubEntry Gammu can return here info, if it was done OK

unsigned char **Text**[(GSM_MAX_CALENDAR_TEXT_LENGTH + 1) * 2]

Text of value, if applicable.

struct **GSM_CalendarEntry**

Calendar note values.

Public Members

GSM_CalendarNoteType **Type**

Type of calendar note.

int **Location**

Location in memory.

int **EntriesNum**

Number of entries.

GSM_SubCalendarEntry **Entries**[GSM_CALENDAR_ENTRIES]

Values of entries.

enum **GSM_ToDoType**

Types of to do values. In parenthesis is member of *GSM_SubToDoEntry*, where value is stored.

Values:

enumerator **TODO_END_DATETIME**

Due date (Date).

enumerator **TODO_COMPLETED**

Whether is completed (Number).

enumerator **TODO_ALARM_DATETIME**

When should alarm be fired (Date).

enumerator **TODO_SILENT_ALARM_DATETIME**

When should silent alarm be fired (Date).

enumerator **TODO_TEXT**

Text of to do (Text).

enumerator **TODO_DESCRIPTION**

Description of to do (Text).

enumerator **TODO_LOCATION**

Location of to do (Text).

enumerator **TODO_PRIVATE**

Whether entry is private (Number).

enumerator **TODO_CATEGORY**

Category of entry (Number).

enumerator **TODO_CONTACTID**

Related contact ID (Number).

enumerator **TODO_PHONE**

Number to call (Text).

enumerator **TODO_LUID**

IrMC LUID which can be used for synchronisation (Text).

enumerator **TODO_LAST_MODIFIED**

Date and time of last modification (Date).

enumerator **TODO_START_DATETIME**

Start date (Date).

enumerator **TODO_COMPLETED_DATETIME**

Completed date (Date).

enum **GSM_ToDo_Priority**

Priority of to do.

Values:

enumerator **GSM_Priority_None**

enumerator **GSM_Priority_High**

enumerator **GSM_Priority_Medium**

enumerator **GSM_Priority_Low**

enumerator **GSM_Priority_INVALID**

struct **GSM_SubToDoEntry**

Value of to do entry.

Public Members

GSM_ToDoType **EntryType**

Type of entry.

GSM_DateTime **Date**

Date of value, if appropriate, see *GSM_ToDoType*.

unsigned int **Number**

Number of value, if appropriate, see *GSM_ToDoType*.

unsigned char **Text**[(GSM_MAX_TODO_TEXT_LENGTH + 1) * 2]

Text of value, if appropriate, see *GSM_ToDoType*.

struct **GSM_ToDoEntry**

To do entry.

Public Members

GSM_CalendarNoteType **Type**

Type of todo note.

GSM_ToDo_Priority **Priority**

Priority of entry.

int **Location**

Location in memory.

int **EntriesNum**

Number of entries.

GSM_SubToDoEntry **Entries**[GSM_TODO_ENTRIES]

Values of current entry.

struct **GSM_NoteEntry**

Note entry.

Public Members

int **Location**

Location in memory.

char **Text**[(GSM_MAX_NOTE_TEXT_LENGTH + 1) * 2]

Text of note.

struct **GSM_Alarm**

Alarm values.

Public Members

int **Location**

Location where it is stored.

GSM_DateTime **DateTime**

Date and time of alarm.

gboolean **Repeating**

Whether it repeats each day.

unsigned char **Text**[(GSM_MAX_CALENDAR_TEXT_LENGTH + 1) * 2]

Text that is shown on display.

enum **GSM_VToDoVersion**

Format of vTodo.

Values:

enumerator **Nokia_VToDo**

Format compatible with Nokia - limited subsed of standard.

enumerator **SonyEricsson_VToDo**

Format compatible with SonyEricsson - complete standard.

enumerator **Mozilla_VToDo**

Format compatible with Mozilla - iCalendar based.

enum **GSM_VCalendarVersion**

Format of vCalendar export.

Values:

enumerator **Nokia_VCalendar**

vCalendar specially hacked for Nokia .

enumerator **Siemens_VCalendar**

vCalendar specially hacked for Siemens.

enumerator **SonyEricsson_VCalendar**

Standard vCalendar (which works for Sony-Ericsson phones)

enumerator **Mozilla_iCalendar**

iCalendar as compatible with Mozilla.

5.3.4 Callback

void **GSM_SetIncomingCallCallback**(*GSM_StateMachine* *s, *IncomingCallCallback* callback, void *user_data)

Sets callback for incoming calls.

Parameters

- **s** – State machine.
- **callback** – Pointer to callback function.
- **user_data** – Second parameter which will be passed to callback.

void **GSM_SetIncomingSMSCallback**(*GSM_StateMachine* *s, *IncomingSMSCallback* callback, void *user_data)

Sets callback for incoming SMSes.

Parameters

- **s** – State machine.
- **callback** – Pointer to callback function.

- **user_data** – Second parameter which will be passed to callback.

void **GSM_SetIncomingCBCallback**(*GSM_StateMachine* *s, *IncomingCBCallback* callback, void *user_data)

Sets callback for incoming CB.

Parameters

- **s** – State machine.
- **callback** – Pointer to callback function.
- **user_data** – Second parameter which will be passed to callback.

void **GSM_SetIncomingUSSDCallback**(*GSM_StateMachine* *s, *IncomingUSSDCallback* callback, void *user_data)

Sets callback for incoming USSD.

Parameters

- **s** – State machine.
- **callback** – Pointer to callback function.
- **user_data** – Second parameter which will be passed to callback.

void **GSM_SetSendSMSStatusCallback**(*GSM_StateMachine* *s, *SendSMSStatusCallback* callback, void *user_data)

Sets callback for sending SMS.

Parameters

- **s** – State machine.
- **callback** – Pointer to callback function.
- **user_data** – Second parameter which will be passed to callback.

typedef void (***IncomingCallCallback**)(*GSM_StateMachine* *s, *GSM_Call* *call, void *user_data)

Callback for incoming calls.

typedef void (***IncomingSMSCallback**)(*GSM_StateMachine* *s, *GSM_SMSMessage* *sms, void *user_data)

Callback for incoming SMS.

typedef void (***IncomingCBCallback**)(*GSM_StateMachine* *s, *GSM_CBMessage* *cb, void *user_data)

Callback for incoming cell broadcast.

typedef void (***IncomingUSSDCallback**)(*GSM_StateMachine* *s, *GSM_USSDMessage* *ussd, void *user_data)

Callback for icoming USSD.

typedef void (***SendSMSStatusCallback**)(*GSM_StateMachine* *s, int status, int MessageReference, void *user_data)

Callback for sending SMS.

5.3.5 Call

GSM_Error **GSM_DialVoice**(*GSM_StateMachine* *s, char *Number, *GSM_CallShowNumber* ShowNumber)

Dials number and starts voice call.

Parameters

- **s** – State machine pointer.
- **Number** – Number to dial.
- **ShowNumber** – Whether we want to display number on phone.

Returns
Error code

GSM_Error **GSM_DialService**(*GSM_StateMachine* *s, char *Number)

Dials service number (usually for USSD).

Parameters

- **s** – State machine pointer.
- **Number** – Number to dial.

Returns
Error code

GSM_Error **GSM_AnswerCall**(*GSM_StateMachine* *s, int ID, *gboolean* all)

Accept current incoming call.

Parameters

- **s** – State machine pointer.
- **ID** – ID of call.
- **all** – Whether to handle all call and not only the one specified by ID.

Returns
Error code

GSM_Error **GSM_CancelCall**(*GSM_StateMachine* *s, int ID, *gboolean* all)

Deny current incoming call.

Parameters

- **s** – State machine pointer.
- **ID** – ID of call.
- **all** – Whether to handle all call and not only the one specified by ID.

Returns
Error code

GSM_Error **GSM_HoldCall**(*GSM_StateMachine* *s, int ID)

Holds call.

Parameters

- **s** – State machine pointer.
- **ID** – ID of call.

Returns
Error code

GSM_Error **GSM_UnholdCall**(*GSM_StateMachine* *s, int ID)

Unholds call.

Parameters

- **s** – State machine pointer.
- **ID** – ID of call.

Returns
Error code

GSM_Error **GSM_ConferenceCall**(*GSM_StateMachine* *s, int ID)

Initiates conference call.

Parameters

- **s** – State machine pointer.
- **ID** – ID of call.

Returns
Error code

GSM_Error **GSM_SplitCall**(*GSM_StateMachine* *s, int ID)

Splits call.

Parameters

- **s** – State machine pointer.
- **ID** – ID of call.

Returns
Error code

GSM_Error **GSM_TransferCall**(*GSM_StateMachine* *s, int ID, *gboolean* next)

Transfers call.

Parameters

- **s** – State machine pointer.
- **ID** – ID of call.
- **next** – Switches next call and ignores ID.

Returns
Error code

GSM_Error **GSM_SwitchCall**(*GSM_StateMachine* *s, int ID, *gboolean* next)

Switches call.

Parameters

- **s** – State machine pointer.
- **ID** – ID of call.
- **next** – Switches next call and ignores ID.

Returns
Error code

GSM_Error **GSM_GetCallDivert**(*GSM_StateMachine* *s, *GSM_CallDivert* *request, *GSM_MultiCallDivert* *result)

Gets call diverts.

Parameters

- **s** – State machine pointer.
- **request** – Which diverts to get.
- **result** – Storage for diversions information.

Returns
Error code

GSM_Error **GSM_SetCallDivert**(*GSM_StateMachine* *s, *GSM_CallDivert* *divert)

Sets call diverts.

Parameters

- **s** – State machine pointer.
- **divert** – Diversions information to set.

Returns
Error code

GSM_Error **GSM_CancelAllDiverts**(*GSM_StateMachine* *s)

Cancels all diverts.

Parameters

- **s** – State machine pointer.

Returns
Error code

GSM_Error **GSM_SetIncomingCall**(*GSM_StateMachine* *s, *gboolean* enable)

Activates/deactivates noticing about incoming calls.

Parameters

- **s** – State machine pointer.
- **enable** – Whether to enable notifications.

Returns
Error code

GSM_Error **GSM_SendDTMF**(*GSM_StateMachine* *s, char *sequence)

Sends DTMF (Dual Tone Multi Frequency) tone.

Parameters

- **s** – State machine pointer.
- **sequence** – Sequence to press.

Returns
Error code

enum **GSM_CallStatus**

Enum with status of call.

Values:

enumerator **GSM_CALL_IncomingCall**
Somebody calls to us

enumerator **GSM_CALL_OutgoingCall**
We call somewhere

enumerator **GSM_CALL_CallStart**
Call started

enumerator **GSM_CALL_CallEnd**
End of call from unknown side

enumerator **GSM_CALL_CallRemoteEnd**
End of call from remote side

enumerator **GSM_CALL_CallLocalEnd**
End of call from our side

enumerator **GSM_CALL_CallEstablished**
Call established. Waiting for answer or dropping

enumerator **GSM_CALL_CallHeld**
Call held

enumerator **GSM_CALL_CallResumed**
Call resumed

enumerator **GSM_CALL_CallSwitched**
We switch to call

struct **GSM_Call**
Call information.

Public Members

GSM_CallStatus **Status**
Call status.

int **CallID**
Call ID

gboolean **CallIDAvailable**
Whether Call ID is available.

int **StatusCode**

Status code.

unsigned char **PhoneNumber**[(GSM_MAX_NUMBER_LENGTH + 1) * 2]

Remote phone number.

enum **GSM_Divert_DivertTypes**

Defines when diversion is active.

Values:

enumerator **GSM_DIVERT_Busy**

Divert when busy.

enumerator **GSM_DIVERT_NoAnswer**

Divert when not answered.

enumerator **GSM_DIVERT_OutOfReach**

Divert when phone off or no coverage.

enumerator **GSM_DIVERT_AllTypes**

Divert all calls without ringing.

enum **GSM_Divert_CallTypes**

Which type of calls should be diverted.

Values:

enumerator **GSM_DIVERT_VoiceCalls**

Voice calls.

enumerator **GSM_DIVERT_FaxCalls**

Fax calls.

enumerator **GSM_DIVERT_DataCalls**

Data calls.

enumerator **GSM_DIVERT_AllCalls**

All calls.

struct **GSM_CallDivert**

Call diversion definition.

Public Members

GSM_Divert_DivertTypes **DivertType**

When diversion is active.

GSM_Divert_CallTypes **CallType**

Type of call to divert.

unsigned int **Timeout**

Timeout for diversion.

unsigned char **Number**[(GSM_MAX_NUMBER_LENGTH + 1) * 2]

Number where to divert.

struct **GSM_MultiCallDivert**

Multiple call diversions.

enum **GSM_CallShowNumber**

How to handle number when initiating voice call.

Values:

enumerator **GSM_CALL_ShowNumber**

Show number.

enumerator **GSM_CALL_HideNumber**

Hide number.

enumerator **GSM_CALL_DefaultNumberPresence**

Keep phone default settings.

5.3.6 Category

GSM_Error **GSM_GetCategory**(*GSM_StateMachine* *s, *GSM_Category* *Category)

Reads category from phone.

Parameters

- **s** – State machine pointer.
- **Category** – Storage for category, containing its type and location.

Returns

Error code

GSM_Error **GSM_AddCategory**(*GSM_StateMachine* *s, *GSM_Category* *Category)

Adds category to phone.

Parameters

- **s** – State machine pointer.

- **Category** – New category, containing its type and location.

Returns
Error code

GSM_Error **GSM_GetCategoryStatus**(*GSM_StateMachine* *s, *GSM_CategoryStatus* *Status)

Reads category status (number of used entries) from phone.

Parameters

- **s** – State machine pointer.
- **Status** – Category status, fill in type before calling.

Returns
Error code

enum **GSM_CategoryType**

Type of category

Values:

enumerator **Category_ToDo**

Todo entry category

enumerator **Category_Phonebook**

Phonebook entry category

struct **GSM_Category**

Category entry.

Public Members

GSM_CategoryType **Type**

Type of category

int **Location**

Location of category

unsigned char **Name**[(GSM_MAX_CATEGORY_NAME_LENGTH + 1) * 2]

Name of category

struct **GSM_CategoryStatus**

Status of categories.

Public Members

GSM_CategoryType **Type**

Type of category.

int **Used**

Number of used category names.

5.3.7 Date and time

char ***DayOfWeek**(unsigned int year, unsigned int month, unsigned int day)

Returns string for current day of week.

Parameters

- **year** – Year.
- **month** – Month.
- **day** – Day.

Returns

Pointer to static buffer containing day of week string.

void **GSM_GetCurrentDateTime**(*GSM_DateTime* *Date)

Returns current timestamp.

Parameters

- **Date** – Storage for date time structure.

time_t **Fill_Time_T**(*GSM_DateTime* DT)

Converts *GSM_DateTime* to time_t.

Parameters

- **DT** – Input timestamp.

Returns

time_t value.

int **GSM_GetLocalTimezoneOffset**(void)

Returns the local timezone offset in seconds. For example 7200 for CEST.

Returns

Timezone offset seconds.

void **Fill_GSM_DateTime**(*GSM_DateTime* *Date, time_t timet)

Converts time_t to gammu *GSM_DateTime* structure.

Parameters

- **Date** – Storage for date.
- **timet** – Input date.

void **GSM_DateTimeFromTimestamp**(*GSM_DateTime* *Date, const char *str)

Converts string (seconds since epoch) to gammu *GSM_DateTime* structure.

Parameters

- **Date** – Storage for date.
- **str** – Input date.

char ***OSDateTime**(*GSM_DateTime* dt, *gboolean* TimeZone)

Converts timestamp to string according to OS settings.

Parameters

- **dt** – Input timestamp.
- **TimeZone** – Whether to include time zone.

Returns

Pointer to static buffer containing string.

char ***OSDate**(*GSM_DateTime* dt)

Converts date from timestamp to string according to OS settings.

Parameters

- **dt** – Input timestamp.

Returns

Pointer to static buffer containing string.

gboolean **CheckDate**(*GSM_DateTime* *date)

Checks whether date is valid. This does not check time, see *CheckTime* for this.

Parameters

- **date** – Structure where to check date.

Returns

True if date is correct.

gboolean **CheckTime**(*GSM_DateTime* *date)

Checks whether time is valid. This does not check date, see *CheckDate* for this.

Parameters

- **date** – Structure where to check time.

Returns

True if time is correct.

GSM_Error **GSM_GetDateTime**(*GSM_StateMachine* *s, *GSM_DateTime* *date_time)

Reads date and time from phone.

Parameters

- **s** – State machine pointer.
- **date_time** – Storage for date.

Returns

Error code

GSM_Error **GSM_SetDateTime**(*GSM_StateMachine* *s, *GSM_DateTime* *date_time)

Sets date and time in phone.

Parameters

- **s** – State machine pointer.
- **date_time** – Date to set.

Returns

Error code

struct **GSM_DateTime**

Structure used for saving date and time

Public Members

int **Timezone**

The difference between local time and GMT in seconds

int **Second**

Seconds.

int **Minute**

Minutes.

int **Hour**

Hours.

int **Day**

Days.

int **Month**

January = 1, February = 2, etc.

int **Year**

Complete year number. Not 03, but 2003

struct **GSM_DeltaTime**

Structure used for saving relative date and time

Public Members

int **Timezone**

The difference of timezones in seconds

int **Second**

Seconds diff.

int **Minute**

Minutes diff.

int **Hour**

Hours diff.

int **Day**

Days diff.

int **Month**

Months diff.

int **Year**

Years diff.

5.3.8 Debug

GSM_Error **GSM_SetDebugFunction**(*GSM_Log_Function* info, void *data, *GSM_Debug_Info* *privdi)

Sets logging function.

Parameters

- **info** – Function to call.
- **data** – User data to pass as a second parameter to callback.
- **privdi** – Pointer to debug information data.

Returns

Error code.

GSM_Error **GSM_SetDebugFile**(const char *info, *GSM_Debug_Info* *privdi)

Sets debug file.

Parameters

- **info** – File path.
- **privdi** – Pointer to debug information data.

Returns

Error code.

GSM_Error **GSM_SetDebugFileDescriptor**(FILE *fd, *gboolean* closable, *GSM_Debug_Info* *privdi)

Sets debug file.

Parameters

- **fd** – File descriptor.
- **privdi** – Pointer to debug information data.
- **closable** – Whether Gammu can close the file when it is no longer needed for debug output. Please note that stderr or stdout are never closed.

Returns
Error code.

GSM_Debug_Info ***GSM_GetGlobalDebug**(void)

Returns global debug settings.

Returns
Pointer to global settings.

GSM_Debug_Info ***GSM_GetDebug**(*GSM_StateMachine* *s)

Gets debug information for state machine.

Parameters

- **s** – State machine data

Returns
Debug information.

GSM_Debug_Info ***GSM_GetDI**(*GSM_StateMachine* *s)

Returns debug information active for state machine. Please note that it can be either global debug or state machine debug structure, depending on use_global flag. For configuring usite GSM_GetDebug.

Parameters

- **s** – State machine data

Returns
Debug information.

gboolean **GSM_SetDebugLevel**(const char *info, *GSM_Debug_Info* *privdi)

Sets debug level.

Parameters

- **info** – Level as text.
- **privdi** – Pointer to debug information data.

Returns
True on success.

gboolean **GSM_SetDebugCoding**(const char *info, *GSM_Debug_Info* *privdi)

Sets debug encoding.

Parameters

- **info** – Encoding to set.
- **privdi** – Pointer to debug information data.

Returns
True on success.

gboolean **GSM_SetDebugGlobal**(*gboolean* info, *GSM_Debug_Info* *privdi)

Enables using of global debugging configuration. Makes no effect on global debug configuration.

Parameters

- **info** – Enable global debug usage..
- **privdi** – Pointer to debug information data.

Returns

True on success.

void **GSM_LogError**(*GSM_StateMachine* *s, const char *message, const *GSM_Error* err)

Logs error to debug log with additional message.

Parameters

- **s** – State machine structure pointer.
- **message** – String to be show in message.
- **err** – Error code.

int **smprintf**(*GSM_StateMachine* *s, const char *format, ...)

Prints string to defined debug log.

Parameters

- **s** – State machine, where to print.
- **format** – Format string as for printf.

Returns

Upon successful return, these functions return the number of characters printed (as printf).

typedef struct _GSM_Debug_Info **GSM_Debug_Info**

Debugging configuration.

5.3.9 Error handling

const char ***GSM_ErrorString**(*GSM_Error* e)

Returns text for error.

Parameters

- **e** – Error code.

Returns

Text (in current locales) describing error

const char ***GSM_ErrorName**(*GSM_Error* e)

Returns name for error.

Parameters

- **e** – Error code.

Returns

Text with error name

enum **GSM_Error**

Error types.

Values:

enumerator **ERR_NONE**

No error

enumerator **ERR_DEVICEOPENERROR**

Error during opening device

enumerator **ERR_DEVICELOCKED**

Device locked

enumerator **ERR_DEVICENOTEXIST**

Device does not exits

enumerator **ERR_DEVICEBUSY**

Device is busy

enumerator **ERR_DEVICENOPERMISSION**

No permissions to open device

enumerator **ERR_DEVICENODRIVER**

No driver installed for a device

enumerator **ERR_DEVICENOTWORK**

Device doesn't seem to be working

enumerator **ERR_DEVICEDTRRTSERROR**

Error during setting DTR/RTS in device

enumerator **ERR_DEVICECHANGESPEEDERROR**

10 Error during changing speed in device

enumerator **ERR_DEVICEWRITEERROR**

Error during writing device

enumerator **ERR_DEVICEREADERROR**

Error during reading device

enumerator **ERR_DEVICEPARITYERROR**

Can't set parity on device

enumerator **ERR_TIMEOUT**

Command timed out

enumerator **ERR_FRAMENOTREQUESTED**

Frame handled, but not requested in this moment

enumerator **ERR_UNKNOWNRESPONSE**

Response not handled by gammu

enumerator **ERR_UNKNOWNFRAME**

Frame not handled by gammu

enumerator **ERR_UNKNOWNCONNECTIONTYPESTRING**

Unknown connection type given by user

enumerator **ERR_UNKNOWNMODELSTRING**

Unknown model given by user

enumerator **ERR_SOURCENOTAVAILABLE**

20 Some functions not compiled in your OS

enumerator **ERR_NOTSUPPORTED**

Not supported by phone

enumerator **ERR_EMPTY**

Empty entry or transfer end.

enumerator **ERR_SECURITYERROR**

Not allowed

enumerator **ERR_INVALIDLOCATION**

Too high or too low location...

enumerator **ERR_NOTIMPLEMENTED**

Function not implemented

enumerator **ERR_FULL**

Memory is full

enumerator **ERR_UNKNOWN**

Unknown response from phone

enumerator **ERR_CANTOPENFILE**

Error during opening file

enumerator **ERR_MOREMEMORY**

More memory required

enumerator **ERR_PERMISSION**

30 No permission

enumerator **ERR_EMPTYSMSC**

SMSC number is empty

enumerator **ERR_INSIDEPHONEMENU**

Inside phone menu - can't make something

enumerator **ERR_NOTCONNECTED**

Phone NOT connected - can't make something

enumerator **ERR_WORKINPROGRESS**

Work in progress

enumerator **ERR_PHONEOFF**

Phone is disabled and connected to charger

enumerator **ERR_FILENOTSUPPORTED**

File format not supported by Gammu

enumerator **ERR_BUG**

Found bug in implementation or phone

enumerator **ERR_CANCELED**

Action was canceled by user

enumerator **ERR_NEEDANOTHERANSWER**

Inside Gammu: phone module need to send another answer frame

enumerator **ERR_OTHERCONNECTIONREQUIRED**

40 You need other connectin for this operation.

enumerator **ERR_WRONGCRC**

Wrong CRC

enumerator **ERR_INVALIDDATETIME**

Invalid date/time

enumerator **ERR_MEMORY**

Phone memory error, maybe it is read only

enumerator **ERR_INVALIDDATA**

Invalid data given to phone

enumerator **ERR_FILEALREADYEXIST**

File with specified name already exist

enumerator **ERR_FILENOTEXIST**

File with specified name doesn't exist

enumerator **ERR_SHOULDBEFOLDER**

You have to give folder (not file) name

enumerator **ERR_SHOULDBEFILE**

You have to give file (not folder) name

enumerator **ERR_NOSIM**

Can not access SIM card

enumerator **ERR_GNAPPLETWRONG**

50 Invalid gnapplet version

enumerator **ERR_FOLDERPART**

Only part of folders listed

enumerator **ERR_FOLDERNOTEMPTY**

Folder is not empty

enumerator **ERR_DATACONVERTED**

Data were converted

enumerator **ERR_UNCONFIGURED**

Gammu is not configured.

enumerator **ERR_WRONGFOLDER**

Wrong folder selected (eg. for SMS).

enumerator **ERR_PHONE_INTERNAL**

Internal phone error (phone got crazy).

enumerator **ERR_WRITING_FILE**

Could not write to a file (on local filesystem).

enumerator **ERR_NONE_SECTION**

No such section exists.

enumerator **ERR_USING_DEFAULTS**

Using default values.

enumerator **ERR_CORRUPTED**

60 Corrupted data returned by phone.

enumerator **ERR_BADFEATURE**

Bad feature string.

enumerator **ERR_DISABLED**

Some functions not compiled in your OS

enumerator **ERR_SPECIFYCHANNEL**

Bluetooth configuration requires channel option.

enumerator **ERR_NOTRUNNING**

Service is not running.

enumerator **ERR_NOSERVICE**

Service setup is missing.

enumerator **ERR_BUSY**

Command failed. Try again.

enumerator **ERR_COULDNT_CONNECT**

Can not connect to server.

enumerator **ERR_COULDNT_RESOLVE**

Can not resolve host name.

enumerator **ERR_GETTING_SMSC**

Failed to get SMSC number from phone.

enumerator **ERR_ABORTED**

70 Operation aborted.

enumerator **ERR_INSTALL_NOT_FOUND**

Installation data not found.

enumerator **ERR_READ_ONLY**

Entry is read only.

enumerator **ERR_NETWORK_ERROR**

Network error.

enumerator **ERR_DB_VERSION**

Invalid database version.

enumerator **ERR_DB_DRIVER**

Failed to initialize DB driver.

enumerator **ERR_DB_CONFIG**

Failed to configure DB driver.

enumerator **ERR_DB_CONNECT**

Failed to connect to database.

enumerator **ERR_DB_TIMEOUT**

Database connection timeout.

enumerator **ERR_SQL**

Error in executing SQL query.

enumerator **ERR_INVALID_OPERATION**

The operation cannot be performed.

enumerator **ERR_MEMORY_NOT_AVAILABLE**

The type of memory is not available or has been disabled.

enumerator **ERR_LAST_VALUE**

Just marker of highest error code, should not be used.

5.3.10 File

GSM_Error **GSM_JADFindData**(*GSM_File* *File, char *Vendor, char *Name, char *JAR, char *Version, int *Size)

Parses JAD file.

Parameters

- **File** – JAD file data.
- **Vendor** – Buffer for vendor name.
- **Name** – Buffer for application name.
- **JAR** – Buffer for JAR URL.
- **Version** – Buffer for version of application.
- **Size** – Pointer to integer to store size.

Returns

Error code.

GSM_Error **GSM_ReadFile**(const char *FileName, *GSM_File* *File)

Reads file from filesystem to *GSM_File* structure.

Parameters

- **FileName** – File to read.
- **File** – Storage for data.

Returns
Error code.

void **GSM_IdentifyFileFormat**(*GSM_File* *File)

Identifies file format by checking it's content.

Parameters

- **File** – File data, Type member will be filled in.

GSM_Error **GSM_GetNextFileFolder**(*GSM_StateMachine* *s, *GSM_File* *File, *gboolean* start)

Gets next filename from filesystem.

Parameters

- **s** – State machine pointer.
- **File** – File structure where path will be stored, if start is FALSE, it should contain data from previous reading (at least ID).
- **start** – Whether we're starting transfer.

Returns
Error code.

GSM_Error **GSM_GetFolderListing**(*GSM_StateMachine* *s, *GSM_File* *File, *gboolean* start)

Gets listing of folder.

Parameters

- **s** – State machine pointer.
- **File** – File structure where path will be stored, if start is FALSE, it should contain data from previous reading (at least ID). On start it should contain path to directory.
- **start** – Whether we're starting transfer.

Returns
Error code.

GSM_Error **GSM_GetNextRootFolder**(*GSM_StateMachine* *s, *GSM_File* *File)

Gets next root folder.

Parameters

- **s** – State machine pointer.
- **File** – File structure where path will be stored.

Returns
Error code.

GSM_Error **GSM_SetFileAttributes**(*GSM_StateMachine* *s, *GSM_File* *File)

Sets file system attributes.

Parameters

- **s** – State machine pointer.
- **File** – File structure with path and attributes.

Returns
Error code.

GSM_Error **GSM_GetFilePart**(*GSM_StateMachine* *s, *GSM_File* *File, int *Handle, size_t *Size)

Retrieves file part.

Parameters

- **s** – State machine pointer.
- **File** – File structure with path, data will be stored here.
- **Size** – Size of transmitted data.
- **Handle** – Handle for saving file, some drivers need this information to be kept between function calls.

Returns

Error code, *ERR_EMPTY* after transfer end.

GSM_Error **GSM_AddFilePart**(*GSM_StateMachine* *s, *GSM_File* *File, size_t *Pos, int *Handle)

Adds file to filesystem. Call repeatedly until function returns *ERR_EMPTY*.

Parameters

- **s** – State machine pointer.
- **File** – File structure and data.
- **Pos** – Position of transmitted data. Should be 0 on start.
- **Handle** – Handle for saving file, some drivers need this information to be kept between function calls.

Returns

Error code, *ERR_EMPTY* after transfer end.

GSM_Error **GSM_SendFilePart**(*GSM_StateMachine* *s, *GSM_File* *File, size_t *Pos, int *Handle)

Sends file to phone, it's up to phone to decide what to do with it. It is usually same as when you receive file over Bluetooth from other phone. Use in same way as *GSM_AddFilePart*.

Parameters

- **s** – State machine pointer.
- **File** – File structure and data.
- **Pos** – Position of transmitted data. Should be 0 on start.
- **Handle** – Handle for saving file, some drivers need this information to be kept between function calls.

Returns

Error code, *ERR_EMPTY* after transfer end.

GSM_Error **GSM_GetFileSystemStatus**(*GSM_StateMachine* *s, *GSM_FileSystemStatus* *Status)

Acquires filesystem status.

Parameters

- **s** – State machine pointer.
- **Status** – Storage for status information.

Returns

Error code.

GSM_Error **GSM_DeleteFile**(*GSM_StateMachine* *s, unsigned char *ID)

Deletes file from filesystem.

Parameters

- **s** – State machine pointer.
- **ID** – ID of folder.

Returns
Error code.

GSM_Error **GSM_AddFolder**(*GSM_StateMachine* *s, *GSM_File* *File)

Adds folder to filesystem.

Parameters

- **s** – State machine pointer.
- **File** – Structure containing information about new folder (Name and FullName).

Returns
Error code.

GSM_Error **GSM_DeleteFolder**(*GSM_StateMachine* *s, unsigned char *ID)

Deletes folder from filesystem.

Parameters

- **s** – State machine pointer.
- **ID** – ID of folder.

Returns
Error code.

struct **GSM_FileSystemStatus**

Status of filesystem.

enum **GSM_FileType**

File type identifier.

Values:

enumerator **GSM_File_Other**

enumerator **GSM_File_Java_JAR**

enumerator **GSM_File_Image_JPG**

enumerator **GSM_File_Image_BMP**

enumerator **GSM_File_Image_GIF**

enumerator **GSM_File_Image_PNG**

enumerator **GSM_File_Image_WBMP**

enumerator **GSM_File_Video_3GP**

enumerator **GSM_File_Sound_AMR**

enumerator **GSM_File_Sound_NRT**
DCT4 binary format

enumerator **GSM_File_Sound_MIDI**

enumerator **GSM_File_MMS**

enumerator **GSM_File_INVALID**

struct **GSM_File**
Structure for holding file information and data.

Public Members

size_t **Used**
How many bytes are used.

unsigned char **Name**[2 * (GSM_MAX_FILENAME_LENGTH + 1)]
Name in Unicode

gboolean **Folder**
True, when folder

int **Level**
How much file is nested on filesystem.

GSM_FileType **Type**
Type of file.

unsigned char **ID_FullName**[2 * (GSM_MAX_FILENAME_ID_LENGTH + 1)]
ID in Unicode

unsigned char ***Buffer**
Pointer to file data.

GSM_DateTime **Modified**
Last modification date.

gboolean **ModifiedEmpty**

Whether modification date is empty.

gboolean **Protected**

Protected file attribute.

gboolean **ReadOnly**

Read only file attribute.

gboolean **Hidden**

Hidden file attribute.

gboolean **System**

System file attribute.

5.3.11 Info

const unsigned char ***GSM_GetNetworkName**(const char *NetworkCode)

Find network name from given network code.

const unsigned char ***GSM_GetCountryName**(const char *CountryCode)

Find country name from given country code.

const char ***GSM_FeatureToString**(*GSM_Feature* feature)

Converts feature value to string.

Parameters

- **feature** – GSM_Feature to convert.

Returns

Pointer to static string with string for specified feature, NULL on failure.

GSM_Feature **GSM_FeatureFromString**(const char *feature)

Converts feature string to value.

Parameters

- **feature** – GSM_Feature string to convert.

Returns

GSM_Feature value, 0 on failure.

gboolean **GSM_IsPhoneFeatureAvailable**(*GSM_PhoneModel* *model, *GSM_Feature* feature)

Checks whether phone supports features.

Parameters

- **model** – Model information (you can get it using *GSM_GetModelInfo*).
- **feature** – GSM_Feature to check for.

Returns

True if phone has defined this feature.

gboolean **GSM_AddPhoneFeature**(*GSM_PhoneModel* *model, *GSM_Feature* feature)

Adds feature to phone configuration.

Parameters

- **model** – Model information (you can get it using *GSM_GetModelInfo*).
- **feature** – GSM_Feature to check for.

Returns

True if phone has defined this feature.

GSM_Error **GSM_GetManufacturer**(*GSM_StateMachine* *s, char *value)

Reads manufacturer from phone.

Parameters

- **s** – State machine pointer.
- **value** – Pointer where to store manufacturer name

Returns

Error code.

GSM_Error **GSM_GetModel**(*GSM_StateMachine* *s, char *value)

Reads model from phone.

Parameters

- **s** – State machine pointer.
- **value** – Pointer where to store model name

Returns

Error code.

GSM_PhoneModel ***GSM_GetModelInfo**(*GSM_StateMachine* *s)

Reads model info from state machine.

Parameters

- **s** – State machine pointer.

Returns

Pointer to phone information structure.

GSM_Error **GSM_GetFirmware**(*GSM_StateMachine* *s, char *value, char *date, double *num)

Reads firmware information from phone.

Parameters

- **s** – State machine pointer.
- **value** – Pointer where to store revision text
- **date** – Pointer where to store revision date
- **num** – Pointer where to store revision number

Returns

Error code.

GSM_Error **GSM_GetIMEI**(*GSM_StateMachine* *s, char *value)

Reads IMEI/serial number from phone.

Parameters

- **s** – State machine pointer.
- **value** – Pointer where to store IMEI, NULL to ignore.

Returns
Error code.

GSM_Error **GSM_GetOriginalIMEI**(*GSM_StateMachine* *s, char *value)

Gets date and time from phone.

GSM_Error **GSM_GetManufactureMonth**(*GSM_StateMachine* *s, char *value)

Gets month when device was manufactured.

GSM_Error **GSM_GetProductCode**(*GSM_StateMachine* *s, char *value)

Gets product code of device.

GSM_Error **GSM_GetHardware**(*GSM_StateMachine* *s, char *value)

Gets hardware information about device.

GSM_Error **GSM_GetPPM**(*GSM_StateMachine* *s, char *value)

Gets PPM (Post Programmable Memory) info from phone (in other words for Nokia get, which language pack is in phone)

GSM_Error **GSM_GetSIMIMSI**(*GSM_StateMachine* *s, char *IMSI)

Gets SIM IMSI from phone.

GSM_Error **GSM_GetBatteryCharge**(*GSM_StateMachine* *s, *GSM_BatteryCharge* *bat)

Gets information about batery charge and phone charging state.

GSM_Error **GSM_GetSignalQuality**(*GSM_StateMachine* *s, *GSM_SignalQuality* *sig)

Reads signal quality (strength and error rate).

GSM_Error **GSM_GetNetworkInfo**(*GSM_StateMachine* *s, *GSM_NetworkInfo* *netinfo)

Gets network information.

GSM_Error **GSM_GetDisplayStatus**(*GSM_StateMachine* *s, *GSM_DisplayFeatures* *features)

Acquired display status.

enum **GSM_NetworkInfo_State**

Status of network logging

Values:

enumerator **GSM_HomeNetwork**

Home network for used SIM card.

enumerator **GSM_NoNetwork**

No network available for used SIM card.

enumerator **GSM_RoamingNetwork**

SIM card uses roaming.

enumerator **GSM_RegistrationDenied**

Network registration denied - card blocked or expired or disabled.

enumerator **GSM_NetworkStatusUnknown**

Unknown network status.

enumerator **GSM_RequestingNetwork**

Network explicitely requested by user.

enum **GSM_GPRS_State**

Status of GPRS connection.

Values:

enumerator **GSM_GPRS_Detached**

GRPS is detached.

enumerator **GSM_GPRS_Attached**

GRPS is attached.

struct **GSM_NetworkInfo**

Structure for getting the current network info.

Public Members

char **CID**[10]

Cell ID (CID)

char **NetworkCode**[10]

GSM network code.

GSM_NetworkInfo_State **State**

Status of network logging. If phone is not logged into any network, some values are not filled

char **LAC**[10]

LAC (Local Area Code).

unsigned char **NetworkName**[20 * 2]

Name of current network returned from phone (or empty). The buffer needs to have twice the capacity of the longest supported network name to account for decoding.

GSM_GPRS_State **GPRS**

GPRS state.

char **PacketCID**[10]

Cell ID (CID) for packet network

GSM_NetworkInfo_State **PacketState**

Status of network logging for packet data. If phone is not logged into any network, some values are not filled

char **PacketLAC**[10]

LAC (Local Area Code) for packet data.

struct **GSM_SignalQuality**

Information about signal quality, all these should be -1 when unknown.

Public Members

int **SignalPercent**

Signal strength in percent.

int **BitErrorRate**

Bit error rate in percent.

enum **GSM_ChargeState**

Power source

Values:

enumerator **GSM_BatteryPowered**

Powered from battery

enumerator **GSM_BatteryConnected**

Powered from AC, battery connected

enumerator **GSM_BatteryCharging**

Powered from AC, battery is charging

enumerator **GSM_BatteryNotConnected**

Powered from AC, no battery

enumerator **GSM_BatteryFull**

Powered from AC, battery is fully charged

enumerator **GSM_PowerFault**

Power failure

enum **GSM_BatteryType**

Power source

Values:

enumerator **GSM_BatteryUnknown**

Unknown battery

enumerator **GSM_BatteryNiMH**

NiMH battery

enumerator **GSM_BatteryLiIon**

Lithium Ion battery

enumerator **GSM_BatteryLiPol**

Lithium Polymer battery

struct **GSM_BatteryCharge**

Battery status

Public Members

int **BatteryPercent**

Signal strength in percent, -1 = unknown

GSM_ChargeState **ChargeState**

Charge state

int **BatteryVoltage**

Current battery voltage (in mV).

int **ChargeVoltage**

Voltage from charger (in mV)

int **ChargeCurrent**

Current from charger (in mA)

int **PhoneCurrent**

Phone current consumption (in mA)

int **BatteryTemperature**

Battery temperature (in degrees Celsius)

int **PhoneTemperature**

Phone temperature (in degrees Celsius)

int **BatteryCapacity**

Remaining battery capacity (in mAh)

GSM_BatteryType **BatteryType**

Battery type

enum **GSM_DisplayFeature**

Display feature

Values:

enumerator **GSM_CallActive**

enumerator **GSM_SMSMemoryFull**

blinking envelope

enumerator **GSM_FaxCall**

enumerator **GSM_UnreadSMS**

enumerator **GSM_DataCall**

enumerator **GSM_VoiceCall**

enumerator **GSM_KeypadLocked**

struct **GSM_DisplayFeatures**

Display features

enum **GSM_Feature**

Phone features definition. This is usually used for things, which can not be determined on run time.

Values:

enumerator **F_CAL33**

Calendar,3310 style - 10 reminders, Unicode, 3 coding types

enumerator **F_CAL52**

Calendar,5210 style - full Unicode, etc.

enumerator **F_CAL82**

Calendar,8250 style - “normal”, but with Unicode

enumerator **F_RING_SM**

Ringtones returned in SM format - 33xx

enumerator **F_NORING**

No ringtones

enumerator **F_NOPBKUNICODE**

No phonebook in Unicode

enumerator **F_NOWAP**

No WAP

enumerator **F_NOCALLER**

No caller groups

enumerator **F_NOPICTURE**

No Picture Images

enumerator **F_NOPICTUREUNI**

No Picture Images text in Unicode

enumerator **F_NOSTARTUP**

No startup logo

enumerator **F_NOCALENDAR**

No calendar

enumerator **F_NOSTARTANI**

Startup logo is not animated

enumerator **F_POWER_BATT**

Network and battery level get from netmonitor

enumerator **F_PROFILES33**

Phone profiles in 3310 style

enumerator **F_PROFILES51**

Phone profiles in 5110 style

enumerator **F_MAGICBYTES**

Phone can make authentication with magic bytes

enumerator **F_NODTMF**

Phone can't send DTMF

enumerator **F_DISPSTATUS**

Phone return display status

enumerator **F_NOCALLINFO**

Phone does not return call info

enumerator **F_DAYMONTH**

Day and month reversed in pbk, when compare to GSM models

enumerator **F_PBK35**

Phonebook in 3510 style with ringtones ID

enumerator **F_PBKIMG**

Phonebook in 7250 style with picture ID

enumerator **F_PBKTONEGAL**

Phonebook with selecting ringtones from gallery

enumerator **F_PBKSMSLIST**

Phonebook with SMS list

enumerator **F_PBKUSER**

Phonebook with user ID

enumerator **F_6230iCALLER**

Caller groups like in 6230i

enumerator **F_RADIO**

Phone with FM radio

enumerator **F_TODO63**

ToDo in 6310 style - 0x55 msg type

enumerator **F_TODO66**

ToDo in 6610 style - like calendar, with date and other

enumerator **F_NOMIDI**

No ringtones in MIDI

enumerator **F_BLUETOOTH**

Bluetooth support

enumerator **F_NOFILESYSTEM**

No images, ringtones, java saved in special filesystem

enumerator **F_NOMMS**

No MMS sets in phone

enumerator **F_NOGPRSPOINT**

GPRS point are not useable

enumerator **F_CAL35**

Calendar,3510 style - Reminder,Call,Birthday

enumerator **F_CAL65**

Calendar,6510 style - CBMM, method 3

enumerator **F_WAPMMSPROXY**

WAP & MMS settings contains first & second proxy

enumerator **F_CHAT**

Phone with Chat settings

enumerator **F_SYNCML**

Phone with SyncML settings

enumerator **F_FILES2**

Filesystem version 2

enumerator **F_NOFILE1**

No filesystem version 1

enumerator **F_6230iWAP**

WAP, MMS, etc. settings like in 6230i - unknown now

enumerator **F_PROFILES**

Profiles support available

enumerator **F_SERIES40_30**

Series 40 3.0

enumerator **F_SMS_FILES**

SMS are read from filesystem files like in Series 40 3.0

enumerator **F_3220_MMS**

MMS storage as in 3320

enumerator **F_VOICETAGS**

Voice tags available

enumerator **F_CAL62**

Calendar,6210 style - Call,Birthday,Memo,Meeting

enumerator **F_NOTES**

Notes supported

enumerator **F_SMSONLYSENT**

Phone supports only sent/unsent messages

enumerator **F_BROKENCPBS**

CPBS on some memories can hang phone

enumerator **F_M20SMS**

Siemens M20 like SMS handling

enumerator **F_SLOWWRITE**

Use slower writing which some phone need

enumerator **F_SMSME900**

SMS in ME start from location 900 - case of Sagem

enumerator **F_ALCATEL**

Phone supports Alcatel protocol

enumerator **F_OBEX**

Phone can switch to OBEX protocol from AT mode

enumerator **F_IRMC_LEVEL_2**

Phone supports IrMC level 2 even if it doesn't report it

enumerator **F_MODE22**

Switching to OBEX mode is done using AT+MODE=22

enumerator **F_SMS_LOCATION_0**

Locations of SMS memories start from 0

enumerator **F_NO_UCS2**

Phone does not support UCS2 even if it reports it.

enumerator **F_FORCE_UTF8**

Phone returns strings in utf-8 even if it reports GSM.

enumerator **F_SMS_SM**

Phone supports SM storage for SMS even if it does not report so.

enumerator **F_SMS_ME**

Phone supports ME storage for SMS even if it does not report so.

enumerator **F_XLNK**

Switching to OBEX mode is done using AT+XLNK.

enumerator **F_SUBMIT_SIM_ONLY**

Submit messages can be saved on SM memory only.

enumerator **F_PBK_UNICODE**

Prefer Unicode for phone book manipulations.

enumerator **F_SQWE**

Switching to OBEX mode using AT^SQWE=3.

enumerator **F_NO_ATOBEX**

Do not use OBEX/AT switching even if available.

enumerator **F_LENGTH_BYTES**

Length of text for contact is in bytes and not chars.

enumerator **F_BROKEN_CMGL**

CMGL does not list real locations for CMGR, these should be sequential.

enumerator **F_EXTRA_PBK_FIELD**

Phonebook has extra numeric field at the end.

enumerator **F_CKPD_NO_UNICODE**

Key presses can not be in unicode.

enumerator **F_CPROT**

OBEX switching using AT+CPROT even if phone does not report it properly.

enumerator **F_PBKFAVORITEMESSAGE**

Phonebook with favorite messaging numbers

enumerator **F_PBKNOPOSTAL**

No support for postal entry in phonebook.

enumerator **F_PBK_ENCODENUMBER**

Encode number in HEX charset.

enumerator **F_NO_CLIP**

Do not use CLIP (phone hangs on it).

enumerator **F_ENCODED_USSD**

USSD propmts and responses are encoded like PDU in SMS (packed 7-bit GSM encoding).

enumerator **F_USE_SMSTEXTMODE**

Phone has better support for SMS text mode (rather than PDU mode)

enumerator **F_CPIN_NO_OK**

Phone does not end CPIN reply with OK/ERROR.

enumerator **F_FOUR_DIGIT_YEAR**

Phone require four digit year in time.

enumerator **F_SMS_NO_ME**

Phone does not have a phone SMS memory even if it reports so.

enumerator **F_SMS_NO_SM**

Phone does not have a SIM SMS memory even if it reports so.

enumerator **F_SIEMENS_PBK**

Phone supports Siemens style phonebook even if it does not tell so.

enumerator **F_NO_ATSYNCML**

Disable AT+SYNCML probing.

enumerator **F_MOBEX**

Phone supports m-obex (usually Samsung phones).

enumerator **F_TSSPCSW**

Phone supports m-obex (usually Samsung phones) using AT$TSSPCSW=1.

enumerator **F_DISABLE_GETNEXT**

Disable GetNext* operations on the dummy phone.

enumerator **F_DISABLE_GETNEXTSMS**

Disable GetNextSMS operations on the dummy phone.

enumerator **F_DISABLE_CMGL**

CMGL hangs, so should not be used.

enumerator **F_NO_UTF8**

Phone does not support UTF8 even if it reports it.

enumerator **F_SAMSUNG_UTF8**

Samsung B2100 in UCS-2 mode provides a garbled UTF-8 instead.

enumerator **F_SMS_UTF8_ENCODED**

SMS text is always UTF-8 encoded.

enumerator **F_NO_STOP_CUSD**

Avoid forcibly stopping CUSD session.

enumerator **F_READ_SMSTEXTMODE**

Reading og SMSes in text mode.

enumerator **F_RESET_AFTER_TIMEOUT**

Reset phone after timeout.

enumerator **F_HUAWEI_INIT**

Huawei style init.

enumerator **F_ZTE_INIT**

ZTE style init.

enumerator **F_USSD_GSM_CHARSET**

Prefer GSM charset for USSD (default is unicode).

enumerator **F_SMS_SR**

Phone supports SR storage even if it does not report so.

enumerator **F_SMS_NO_SR**

Phone does not have a SR memory even if it reports so.

enumerator **F_LAST_VALUE**

Just marker of highest feature code, should not be used.

struct **GSM_PhoneModel**

Model identification, used for finding phone features.

Public Members

const char ***model**

Model as returned by phone

const char ***number**

Identification by Gammu

const char ***irdamodel**

Model as used over IrDA

GSM_Feature **features**[GSM_MAX_PHONE_FEATURES + 1]

List of supported features

5.3.12 INI files

void **INI_Free**(*INI_Section* *head)

Free INI data.

Parameters

- **head** – INI section data.

GSM_Error **INI_ReadFile**(const char *FileName, *gboolean* Unicode, *INI_Section* **result)

Reads INI data.

Parameters

- **FileName** – File to read.
- **Unicode** – Whether file shoul be treated like unicode.
- **result** – Pointer where file will be read.

Returns
Error code

INI_Entry ***INI_FindLastSectionEntry**(*INI_Section* *file_info, const unsigned char *section, const *gboolean* Unicode)

Returns pointer to last INI entry of given section.

Bug:

Unicode should be part of file_info.

Parameters

- **file_info** – File data as returned by *INI_ReadFile*.
- **section** – Section to scan.
- **Unicode** – Whether file is unicode.

Returns
Last entry in section.

unsigned char ***INI_GetValue**(*INI_Section* *file_info, const unsigned char *section, const unsigned char *key, const *gboolean* Unicode)

Returns value of INI file entry.

Bug:

Unicode should be part of file_info.

Parameters

- **file_info** – File data as returned by *INI_ReadFile*.
- **section** – Section to scan.
- **key** – Name of key to read.
- **Unicode** – Whether file is unicode.

Returns
Entry value.

int **INI_GetInt**(*INI_Section* *cfg, const unsigned char *section, const unsigned char *key, int fallback)

Returns integer value from configuration. The file is automatically handled as not unicode.

Parameters

- **cfg** – File data as returned by *INI_ReadFile*.
- **section** – Section to scan.
- **key** – Name of key to read.
- **fallback** – Fallback value.

Returns
Key value or fallback in case of failure.

gboolean **INI_GetBool**(*INI_Section* *cfg, const unsigned char *section, const unsigned char *key, *gboolean* fallback)

Returns boolean value from configuration. The file is automatically handled as not unicode.

Parameters

- **cfg** – File data as returned by *INI_ReadFile*.
- **section** – Section to scan.
- **key** – Name of key to read.
- **fallback** – Fallback value.

Returns
Key value or fallback in case of failure.

gboolean **GSM_StringToBool**(const char *value)

Converts value to boolean.

It just takes the string and checks whether there is true/yes/t/y/1 or false/no/f/n/0.

Parameters

- **value** – String to parse.

Returns
Boolean value, -1 on failure.

typedef struct *_INI_Entry* **INI_Entry**

Private structure holding information INI entry.

typedef struct *_INI_Section* **INI_Section**

Private structure holding information INI section.

struct **_INI_Entry**

Structure used to save value for single key in INI style file

Todo:

This should be probably private.

struct **_INI_Section**

Structure used to save section in INI style file

Todo:

This should be probably private.

5.3.13 Keys

GSM_Error **MakeKeySequence**(char *text, *GSM_KeyCode* *KeyCode, size_t *Length)

Creates key sequence from string.

Parameters

- **text** – Text to convert.
- **KeyCode** – Storage for key codes.
- **Length** – Storage for resulting length.

Returns

Error code.

GSM_Error **GSM_PressKey**(*GSM_StateMachine* *s, *GSM_KeyCode* Key, *gboolean* Press)

Emulates key press or key release.

enum **GSM_KeyCode**

Key event identifiers.

Values:

enumerator **GSM_KEY_NONE**

enumerator **GSM_KEY_1**

enumerator **GSM_KEY_2**

enumerator **GSM_KEY_3**

enumerator **GSM_KEY_4**

enumerator **GSM_KEY_5**

enumerator **GSM_KEY_6**

enumerator **GSM_KEY_7**

enumerator **GSM_KEY_8**

enumerator **GSM_KEY_9**

enumerator **GSM_KEY_0**

enumerator **GSM_KEY_HASH**

#

enumerator **GSM_KEY_ASTERISK**

•

enumerator **GSM_KEY_POWER**

Power key.

enumerator **GSM_KEY_GREEN**

in some phone ie. N5110 sometimes works identical to POWER

enumerator **GSM_KEY_RED**

(c) key in some phone: ie. N5110

enumerator **GSM_KEY_INCREASEVOLUME**

Not available in some phones as separate button: ie. N5110

enumerator **GSM_KEY_DECREASEVOLUME**

Not available in some phones as separate button: ie. N5110

enumerator **GSM_KEY_UP**

enumerator **GSM_KEY_DOWN**

enumerator **GSM_KEY_MENU**

enumerator **GSM_KEY_NAMES**

Not available in some phone: ie. N5110

enumerator **GSM_KEY_LEFT**

Left arrow

enumerator **GSM_KEY_RIGHT**

Right arrow

enumerator **GSM_KEY_SOFT1**

Software key which has assigned mening on display.

enumerator **GSM_KEY_SOFT2**

Software key which has assigned mening on display.

enumerator **GSM_KEY_HEADSET**

Button on headset

enumerator **GSM_KEY_JOYSTICK**

Joystick pressed

enumerator **GSM_KEY_CAMERA**

Camera button pressed

enumerator **GSM_KEY_MEDIA**

Media player button

enumerator **GSM_KEY_DESKTOP**

Multi function key, desktop

enumerator **GSM_KEY_OPERATOR**

Operator button

enumerator **GSM_KEY_RETURN**

Return button

enumerator **GSM_KEY_CLEAR**

Clear button

5.3.14 Limits

5.3.15 Memory

GSM_MemoryType **GSM_StringToMemoryType**(const char *s)

Converts memory type from string.

Parameters

- **s** – String with memory type.

Returns

Parsed memory type or 0 on failure.

GSM_Error **GSM_GetMemoryStatus**(*GSM_StateMachine* *s, *GSM_MemoryStatus* *status)

Gets memory (phonebooks or calls) status (eg. number of used and free entries).

Parameters

- **s** – State machine pointer.
- **status** – Storage for status information, MemoryType has to be set.

Returns
Error code.

GSM_Error **GSM_GetMemory**(*GSM_StateMachine* *s, *GSM_MemoryEntry* *entry)

Reads entry from memory (phonebooks or calls). Which entry should be read is defined in entry.

Parameters

- **s** – State machine pointer.
- **entry** – Storage for retrieved entry, MemoryType and Location has to be set.

Returns
Error code.

GSM_Error **GSM_GetNextMemory**(*GSM_StateMachine* *s, *GSM_MemoryEntry* *entry, *gboolean* start)

Reads entry from memory (phonebooks or calls). Which entry should be read is defined in entry. This can be easily used for reading all entries.

Parameters

- **s** – State machine pointer.
- **entry** – Storage for retrieved entry. MemoryType has to be set for first call (with start set to TRUE), for subsequent calls Location has to stay intact from previous reading.
- **start** – Whether we should start from beginning.

Returns
Error code.

GSM_Error **GSM_SetMemory**(*GSM_StateMachine* *s, *GSM_MemoryEntry* *entry)

Sets memory (phonebooks or calls) entry.

Parameters

- **s** – State machine pointer.
- **entry** – Entry to set, Location and MemoryType has to be set.

Returns
Error code.

GSM_Error **GSM_AddMemory**(*GSM_StateMachine* *s, *GSM_MemoryEntry* *entry)

Adds memory (phonebooks or calls) entry.

Parameters

- **s** – State machine pointer.
- **entry** – Entry to add, Location is ignored, MemoryType has to be set.

Returns
Error code.

GSM_Error **GSM_DeleteMemory**(*GSM_StateMachine* *s, *GSM_MemoryEntry* *entry)

Deletes memory (phonebooks or calls) entry.

Parameters

- **s** – State machine pointer.
- **entry** – Entry to delete, Location and MemoryType has to be set.

Returns
Error code.

GSM_Error **GSM_DeleteAllMemory**(*GSM_StateMachine* *s, *GSM_MemoryType* MemoryType)

Deletes all memory (phonebooks or calls) entries of specified type.

Parameters

- **s** – State machine pointer.
- **MemoryType** – Where to delete all entries.

Returns
Error code.

GSM_Error **GSM_GetSpeedDial**(*GSM_StateMachine* *s, *GSM_SpeedDial* *Speed)

Gets speed dial.

Parameters

- **s** – State machine pointer.
- **Speed** – Storage for speed dial, Location has to be set.

Returns
Error code.

GSM_Error **GSM_SetSpeedDial**(*GSM_StateMachine* *s, *GSM_SpeedDial* *Speed)

Sets speed dial.

Parameters

- **s** – State machine pointer.
- **Speed** – Sspeed dial to set.

Returns
Error code.

unsigned char ***GSM_PhonebookGetEntryName**(const *GSM_MemoryEntry* *entry)

Returns name of entry. It might be possibly concatenated from first and last names.

Parameters

- **entry** – Entry to process.

Returns
Static unicode string containing name.

void **GSM_PhonebookFindDefaultNameNumberGroup**(const *GSM_MemoryEntry* *entry, int *Name, int *Number, int *Group)

Finds default name, number and group for entry.

Parameters

- **entry** – Entry to process.
- **Name** – Output index of name.
- **Number** – Output index of number.
- **Group** – Output index of group.

GSM_Error **GSM_EncodeVCARD**(*GSM_Debug_Info* *di, char *Buffer, const size_t buff_len, size_t *Pos, *GSM_MemoryEntry* *pbk, const *gboolean* header, const *GSM_VCardVersion* Version)

Encodes memory entry to vCard.

Parameters

- **di** – Pointer to debugging description.
- **Buffer** – **[out]** Buffer to store vCard text.
- **buff_len** – **[in]** Size of output buffer.
- **Pos** – **[inout]** Position in output buffer.
- **pbk** – **[inout]** Phonebook data, AddError will be set on non converted entries.
- **header** – **[in]** Whether to include vCard header in output.
- **Version** – **[in]** What vCard version to create.

Returns
Error code.

GSM_Error **GSM_DecodeVCARD**(*GSM_Debug_Info* *di, char *Buffer, size_t *Pos, *GSM_MemoryEntry* *Pbk, const *GSM_VCardVersion* Version)

Decodes memory entry from vCard.

Parameters

- **di** – Pointer to debugging description.
- **Buffer** – **[in]** Buffer to readCard text.
- **Pos** – **[inout]** Position in output buffer.
- **Pbk** – **[out]** Phonebook data read from vCard.
- **Version** – **[in]** What vCard version to parse.

Returns
Error code.

void **GSM_FreeMemoryEntry**(*GSM_MemoryEntry* *Entry)

Frees any dynamically allocated memory inside memory entry structure.

Parameters

- **Entry** – **[in]** Pointer to memory entry to process.

enum **GSM_MemoryType**

Enum defines ID for various phone and SIM memories. Phone modules can translate them to values specific for concrete models. Two letter codes (excluding VM and SL) are from GSM 07.07.

Values:

enumerator **MEM_ME**

Internal memory of the mobile equipment

enumerator **MEM_SM**

SIM card memory

enumerator **MEM_ON**

Own numbers

enumerator **MEM_DC**

Dialled calls

enumerator **MEM_RC**

Received calls

enumerator **MEM_MC**

Missed calls

enumerator **MEM_MT**

Combined ME and SIM phonebook

enumerator **MEM_FD**

Fixed dial

enumerator **MEM_VM**

Voice mailbox

enumerator **MEM_SL**

Sent SMS logs

enumerator **MEM_QD**

Quick dialing choices.

enumerator **MEM_SR**

Status report memory

enumerator **MEM_INVALID**

Invalid memory type.

struct **GSM_MemoryStatus**

Structure contains info about number of used/free entries in phonebook memory.

Public Members

int **MemoryUsed**

Number of used entries

GSM_MemoryType **MemoryType**

Memory type

int **MemoryFree**

Number of free entries

enum **GSM_EntryType**

Type of specific phonebook entry. In parenthesis is specified in which member of *GSM_SubMemoryEntry* value is stored.

Values:

enumerator **PBK_Number_General**

General number. (Text)

enumerator **PBK_Number_Mobile**

Mobile number. (Text)

enumerator **PBK_Number_Fax**

Fax number. (Text)

enumerator **PBK_Number_Pager**

Pager number. (Text)

enumerator **PBK_Number_Other**

Other number. (Text)

enumerator **PBK_Text_Note**

Note. (Text)

enumerator **PBK_Text_Postal**

Complete postal address. (Text)

enumerator **PBK_Text_Email**

Email. (Text)

enumerator **PBK_Text_Email2**

enumerator **PBK_Text_URL**

URL (Text)

enumerator **PBK_Date**

Date and time of last call. (Date)

enumerator **PBK_Caller_Group**

Caller group. (Number)

enumerator **PBK_Text_Name**

Name (Text)

enumerator **PBK_Text_LastName**

Last name. (Text)

enumerator **PBK_Text_FirstName**
First name. (Text)

enumerator **PBK_Text_Company**
Company. (Text)

enumerator **PBK_Text_JobTitle**
Job title. (Text)

enumerator **PBK_Category**
Category. (Number, if -1 then text)

enumerator **PBK_Private**
Whether entry is private. (Number)

enumerator **PBK_Text_StreetAddress**
Street address. (Text)

enumerator **PBK_Text_City**
City. (Text)

enumerator **PBK_Text_State**
State. (Text)

enumerator **PBK_Text_Zip**
Zip code. (Text)

enumerator **PBK_Text_Country**
Country. (Text)

enumerator **PBK_Text_Custom1**
Custom information 1. (Text)

enumerator **PBK_Text_Custom2**
Custom information 2. (Text)

enumerator **PBK_Text_Custom3**
Custom information 3. (Text)

enumerator **PBK_Text_Custom4**
Custom information 4. (Text)

enumerator **PBK_RingtoneID**
Ringtone ID. (Number)

enumerator **PBK_PictureID**

Picture ID. (Number)

enumerator **PBK_Text_UserID**

User ID. (Text)

enumerator **PBK_CallLength**

Length of call (Number)

enumerator **PBK_Text_LUID**

LUID - Unique Identifier used for synchronisation (Text)

enumerator **PBK_LastModified**

Date of last modification (Date)

enumerator **PBK_Text_NickName**

Nick name (Text)

enumerator **PBK_Text_FormalName**

Formal name (Text)

enumerator **PBK_Text_PictureName**

Picture name (on phone filesystem). (Text)

enumerator **PBK_PushToTalkID**

Push-to-talk ID (Text)

enumerator **PBK_Number_Messaging**

Favorite messaging number. (Text)

enumerator **PBK_Photo**

Photo (Picture).

enumerator **PBK_Text_SecondName**

Second name. (Text)

enumerator **PBK_Text_VOIP**

VOIP address (Text).

enumerator **PBK_Text_SIP**

SIP address (Text).

enumerator **PBK_Text_DTMF**

DTMF (Text).

enumerator **PBK_Number_Video**

Video number. (Text)

enumerator **PBK_Text_SWIS**

See What I See address. (Text)

enumerator **PBK_Text_WVID**

Wireless Village user ID. (Text)

enumerator **PBK_Text_NamePrefix**

Name prefix (Text)

enumerator **PBK_Text_NameSuffix**

Name suffix (Text)

enum **GSM_EntryLocation**

Location of memory contact.

Values:

enumerator **PBK_Location_Unknown**

No/Unknown location.

enumerator **PBK_Location_Home**

Home

enumerator **PBK_Location_Work**

Work

struct **GSM_SubMemoryEntry**

One value of phonebook memory entry.

Public Members

GSM_EntryType **EntryType**

Type of entry.

GSM_EntryLocation **Location**

Location for the entry.

GSM_DateTime **Date**

Text of entry (if applicable, see *GSM_EntryType*).

int **Number**

Number of entry (if applicable, see *GSM_EntryType*).

int **VoiceTag**

Voice dialling tag.

GSM_Error **AddError**

During adding SubEntry Gammu can return here info, if it was done OK

unsigned char **Text**[(GSM_PHONEBOOK_TEXT_LENGTH + 1) * 2]

Text of entry (if applicable, see *GSM_EntryType*).

GSM_BinaryPicture **Picture**

Picture data.

struct **GSM_MemoryEntry**

Structure for saving phonebook entries.

Public Members

GSM_MemoryType **MemoryType**

Used memory for phonebook entry

int **Location**

Used location for phonebook entry

int **EntriesNum**

Number of SubEntries in Entries table.

GSM_SubMemoryEntry **Entries**[GSM_PHONEBOOK_ENTRIES]

Values of SubEntries.

struct **GSM_SpeedDial**

Structure for saving speed dials.

Public Members

int **Location**

Number of speed dial: 2,3..,8,9

int **MemoryNumberID**

ID of phone number used in phonebook entry

GSM_MemoryType **MemoryType**

Memory, where is saved used phonebook entry

int **MemoryLocation**

Location in memory, where is saved used phonebook entry

enum **GSM_VCardVersion**

Types of vCard.

Values:

enumerator **Nokia_VCard10**

vCard 1.0 hacked for Nokia.

enumerator **Nokia_VCard21**

vCard 2.1 hacked for Nokia.

enumerator **SonyEricsson_VCard10**

vCard 1.0 hacked for Sony-Ericsson (should be standard vCard).

enumerator **SonyEricsson_VCard21**

vCard 2.1 hacked for Sony-Ericsson (should be standard vCard).

enumerator **SonyEricsson_VCard21_Phone**

vCard 2.1 hacked for Sony-Ericsson (should be standard vCard) from phone (no parsing of location and memory type).

5.3.16 Messages

GSM_Error **GSM_DecodePDUFrame**(*GSM_Debug_Info* *di, *GSM_SMSMessage* *SMS, const unsigned char *buffer, size_t length, size_t *final_pos, *gboolean* SMSC)

Decodes PDU data.

Parameters

- **di** – Debug information structure.
- **SMS** – Pointer where to store parsed message.
- **buffer** – PDU data.
- **length** – Length of PDU data.
- **final_pos** – Optional pointer where end position will be stored.
- **SMSC** – Whether PDU includes SMSC data.

GSM_Error **GSM_DecodeSMSFrame**(*GSM_Debug_Info* *di, *GSM_SMSMessage* *SMS, unsigned char *buffer, *GSM_SMSMessageLayout* Layout)

Decodes SMS frame.

GSM_Coding_Type **GSM_GetMessageCoding**(*GSM_Debug_Info* *di, const char TPDCS)

Finds out coding type based on TPDCS header byte as defined by GSM 03.38.

GSM_Error **GSM_EncodeSMSFrame**(*GSM_Debug_Info* *di, *GSM_SMSMessage* *SMS, unsigned char *buffer, *GSM_SMSMessageLayout* Layout, int *length, *gboolean* clear)

Encodes SMS frame.

GSM_Error **GSM_DecodeSMSFrameStatusReportData**(*GSM_Debug_Info* *di, *GSM_SMSMessage* *SMS, unsigned char *buffer, *GSM_SMSMessageLayout* Layout)

Decodes SMS frame for status report.

GSM_Error **GSM_DecodeSMSFrameText**(*GSM_Debug_Info* *di, *GSM_SMSMessage* *SMS, unsigned char *buffer, *GSM_SMSMessageLayout* Layout)

Decodes SMS frame in textual representation.

void **GSM_DecodeUDHHeader**(*GSM_Debug_Info* *di, *GSM_UDHHeader* *UDH)

Decodes UDH header.

void **GSM_EncodeUDHHeader**(*GSM_Debug_Info* *di, *GSM_UDHHeader* *UDH)

Encodes UDH header.

void **GSM_SetDefaultReceivedSMSData**(*GSM_SMSMessage* *SMS)

Sets default content for SMS except for changing locations. Use this for clearing structure while keeping location of message.

Parameters

- **SMS** – Pointer to structure which should be cleaned up.

void **GSM_SetDefaultSMSData**(*GSM_SMSMessage* *SMS)

Sets default content for SMS. Use this for clearing structure.

Parameters

- **SMS** – Pointer to structure which should be cleaned up.

gboolean **GSM_DecodeSiemensOTASMS**(*GSM_Debug_Info* *di, *GSM_SiemensOTASMSInfo* *Info, *GSM_SMSMessage* *SMS)

Decodes Siemens OTA data.

GSM_Error **PHONE_EncodeSMSFrame**(*GSM_StateMachine* *s, *GSM_SMSMessage* *SMS, unsigned char *buffer, *GSM_SMSMessageLayout* Layout, int *length, *gboolean* clear)

Encodes SMS frame according to layout.

Returns
Error code.

GSM_Error **GSM_EncodeMultiPartSMS**(*GSM_Debug_Info* *di, *GSM_MultiPartSMSInfo* *Info, *GSM_MultiSMSMessage* *SMS)

Encodes multi part SMS from "readable" format.

Returns
Error code.

gboolean **GSM_DecodeMultiPartSMS**(*GSM_Debug_Info* *di, *GSM_MultiPartSMSInfo* *Info, *GSM_MultiSMSMessage* *SMS, *gboolean* ems)

Decodes multi part SMS to "readable" format.

void **GSM_ClearMultiPartSMSInfo**(*GSM_MultiPartSMSInfo* *Info)

Clears *GSM_MultiPartSMSInfo* to default values.

void **GSM_FreeMultiPartSMSInfo**(*GSM_MultiPartSMSInfo* *Info)

Frees any allocated structures inside *GSM_MultiPartSMSInfo*.

GSM_Error **GSM_LinkSMS**(*GSM_Debug_Info* *di, *GSM_MultiSMSMessage* **INPUT, *GSM_MultiSMSMessage* **OUTPUT, *gboolean* ems)

Links SMS messages according to IDs.

Returns
Error code.

GSM_Error **GSM_DecodeMMSFileToMultiPart**(*GSM_Debug_Info* *di, *GSM_File* *file, *GSM_EncodedMultiPartMMSInfo* *info)

Decodes MMS data.

GSM_Error **GSM_ClearMMSMultiPart**(*GSM_EncodedMultiPartMMSInfo* *info)

Clears MMS data, used to initialize structure.

GSM_Error **GSM_GetSMSC**(*GSM_StateMachine* *s, *GSM_SMSC* *smsc)

Gets SMS Service Center number and SMS settings.

Parameters

- **s** – State machine pointer.
- **smsc** – **[inout]** SMSC structure, should contain location.

Returns
Error code.

GSM_Error **GSM_SetSMSC**(*GSM_StateMachine* *s, *GSM_SMSC* *smsc)

Sets SMS Service Center number and SMS settings.

Parameters

- **s** – State machine pointer.
- **smsc** – **[in]** SMSC structure.

Returns
Error code.

GSM_Error **GSM_GetSMSStatus**(*GSM_StateMachine* *s, *GSM_SMSMemoryStatus* *status)

Gets information about SMS memory (read/unread/size of memory for both SIM and phone).

Parameters

- **s** – State machine pointer.
- **status** – **[out]** Pointer to SMS status structure.

Returns
Error code.

GSM_Error **GSM_GetSMS**(*GSM_StateMachine* *s, *GSM_MultiSMSMessage* *sms)

Reads SMS message.

Parameters

- **s** – State machine pointer.
- **sms** – **[inout]** SMS message data read from phone, location and folder should be set.

Returns
Error code.

GSM_Error **GSM_GetNextSMS**(*GSM_StateMachine* *s, *GSM_MultiSMSMessage* *sms, *gboolean* start)

Reads next (or first if start set) SMS message. This might be faster for some phones than using *GSM_GetSMS* for each message.

Please note that this commend does not have to mark message as read in phone. To do so, you have to call *GSM_GetSMS*.

Parameters

- **s** – State machine pointer.
- **sms** – **[inout]** SMS message data read from phone, for subsequent reads, location and folder might be used by phone driver to determine reading state.
- **start** – **[in]** Whether we start reading from beginning.

Returns
Error code.

GSM_Error **GSM_SetSMS**(*GSM_StateMachine* *s, *GSM_SMSMessage* *sms)

Sets SMS.

Parameters

- **s** – State machine pointer.
- **sms** – **[in]** SMS message data.

Returns
Error code.

GSM_Error **GSM_AddSMS**(*GSM_StateMachine* *s, *GSM_SMSMessage* *sms)

Adds SMS to specified folder.

Parameters

- **s** – State machine pointer.
- **sms** – **[inout]** SMS message data, location will be updated.

Returns
Error code.

GSM_Error **GSM_DeleteSMS**(*GSM_StateMachine* *s, *GSM_SMSMessage* *sms)

Deletes SMS.

Parameters

- **s** – State machine pointer.
- **sms** – **[in]** SMS structure with SMS location and folder.

Returns
Error code.

GSM_Error **GSM_SendSMS**(*GSM_StateMachine* *s, *GSM_SMSMessage* *sms)

Sends SMS.

Parameters

- **s** – State machine pointer.
- **sms** – **[in]** SMS structure with SMS data to send.

Returns
Error code.

GSM_Error **GSM_SendSavedSMS**(*GSM_StateMachine* *s, int Folder, int Location)

Sends SMS already saved in phone.

Parameters

- **s** – State machine pointer.
- **Folder** – **[in]** Folder, where message is stored.
- **Location** – **[in]** Location, where message is stored.

Returns
Error code.

GSM_Error **GSM_SetFastSMSSending**(*GSM_StateMachine* *s, *gboolean* enable)

Configures fast SMS sending.

Parameters

- **s** – State machine pointer.
- **enable** – **[in]** Whether to enable notifications.

Returns
Error code.

GSM_Error **GSM_SetIncomingSMS**(*GSM_StateMachine* *s, *gboolean* enable)

Enable/disable notification on incoming SMS.

Parameters

- **s** – State machine pointer.
- **enable** – **[in]** Whether to enable notifications.

Returns
Error code.

GSM_Error **GSM_SetIncomingCB**(*GSM_StateMachine* *s, *gboolean* enable)

Gets network information from phone.

Parameters

- **s** – State machine pointer.
- **enable** – **[in]** Whether to enable notifications.

Returns
Error code.

GSM_Error **GSM_GetSMSFolders**(*GSM_StateMachine* *s, *GSM_SMSFolders* *folders)

Returns SMS folders information.

Parameters

- **s** – State machine pointer.
- **folders** – **[out]** folders Pointer to folders structure, which will be filled in.

Returns
Error code.

GSM_Error **GSM_AddSMSFolder**(*GSM_StateMachine* *s, unsigned char *name)

Creates SMS folder.

Parameters

- **s** – State machine pointer.
- **name** – **[in]** Name of SMS folder which should be created.

Returns
Error code.

GSM_Error **GSM_DeleteSMSFolder**(*GSM_StateMachine* *s, int ID)

Deletes SMS folder.

Parameters

- **s** – State machine pointer.
- **ID** – **[in]** ID of SMS folder to delete.

Returns
Error code.

GSM_Error **GSM_GetMMSFolders**(*GSM_StateMachine* *s, *GSM_MMSFolders* *folders)

Lists MMS folders.

Parameters

- **s** – State machine pointer.
- **folders** – Pointer to structure, whehe folder information will be stored.

Returns
Error code.

GSM_Error **GSM_GetNextMMSFileInfo**(*GSM_StateMachine* *s, unsigned char *FileID, int *MMSFolder, *gboolean* start)

Retrieves next part of MMS file information.

Parameters

- **s** – State machine pointer.
- **FileID** – **[inout]** File ID will be stored here, might be used for consequent reads.
- **MMSFolder** – **[inout]** MMS folder ID will be stored here, might be used for consequent reads.
- **start** – **[in]** Whether to start reading.

Returns
Error code.

GSM_Error **GSM_SetIncomingUSSD**(*GSM_StateMachine* *s, *gboolean* enable)

Activates/deactivates noticing about incoming USSDs (UnStructured Supplementary Services).

Parameters

- **s** – State machine pointer.
- **enable** – **[in]** Whether to enable notifications.

Returns
Error code.

void **GSM_SMSCounter**(*GSM_Debug_Info* *di, unsigned char *MessageBuffer, *GSM_UDH* UDHType, *GSM_Coding_Type* Coding, int *SMSNum, size_t *CharsLeft)

Calculates number of messages and free chars needed for text.

Parameters

- **di** – Debug settings.
- **MessageBuffer** – **[in]** Actual message text in unicode.
- **UDHType** – **[in]** UDH type.
- **Coding** – **[in]** GSM Encoding type.
- **SMSNum** – **[out]** Number of messages needed to store the text.
- **CharsLeft** – **[out]** Number of free chars in the message.

enum **GSM_MMS_Class**

MMS message class.

Values:

enumerator **GSM_MMS_None**

None class specified.

enumerator **GSM_MMS_Personal**

Personal message.

enumerator **GSM_MMS_Advertisement**

Advertisement message.

enumerator **GSM_MMS_Info**

Informational message.

enumerator **GSM_MMS_Auto**

Automatic message class.

enumerator **GSM_MMS_INVALID**

struct **GSM_MMSIndicator**

MMS indicator data.

Public Members

char **Address**[500]

Message address (URL for download).

char **Title**[200]

Message title (subject).

char **Sender**[200]

Message sender.

size_t **MessageSize**

Message size, if 0 it won't be decoded or was not decoded.

GSM_MMS_Class **Class**

Class of a message.

struct **GSM_CBMessage**

Structure for Cell Broadcast messages.

Public Members

int **Channel**

Channel number.

char **Text**[300]

Message text.

enum **GSM_USSDStatus**

Status of USSD message.

Values:

enumerator **USSD_Unknown**

Unknown status

enumerator **USSD_NoActionNeeded**

No action is needed, maybe network initiated USSD

enumerator **USSD_ActionNeeded**

Reply is expected

enumerator **USSD_Terminated**

USSD dialog terminated

enumerator **USSD_AnotherClient**

Another client replied

enumerator **USSD_NotSupported**

Operation not supported

enumerator **USSD_Timeout**

Network timeout

struct **GSM_USSDMessage**

Structure for USSD messages.

Public Members

unsigned char **Text**[2 * (GSM_MAX_USSD_LENGTH + 1)]

Message text.

GSM_USSDStatus **Status**

Message status.

struct **GSM_SMSMemoryStatus**

Status of SMS memory.

Public Members

int **SIMUnRead**

Number of unread messages on SIM.

int **SIMUsed**

Number of all saved messages (including unread) on SIM.

int **SIMSize**

Number of all possible messages on SIM.

int **TemplatesUsed**

Number of used templates (62xx/63xx/7110/etc.).

int **PhoneUnRead**

Number of unread messages in phone.

int **PhoneUsed**

Number of all saved messages in phone.

int **PhoneSize**

Number of all possible messages on phone.

enum **GSM_SMSFormat**

Enum defines format of SMS messages. See GSM 03.40 section 9.2.3.9

Values:

enumerator **SMS_FORMAT_Pager**

enumerator **SMS_FORMAT_Fax**

enumerator **SMS_FORMAT_Email**

enumerator **SMS_FORMAT_Text**

enum **GSM_ValidityPeriod**

Enum defines some the most often used validity lengths for SMS messages for relative validity format. See GSM 03.40 section 9.2.3.12.1 - it gives more values.

Values:

enumerator **SMS_VALID_1_Hour**

enumerator **SMS_VALID_6_Hours**

enumerator **SMS_VALID_1_Day**

enumerator **SMS_VALID_3_Days**

enumerator **SMS_VALID_1_Week**

enumerator **SMS_VALID_Max_Time**

enum **GSM_ValidityPeriodFormat**

Enum defines format of validity period for SMS messages. See GSM 03.40 section 9.2.3.12

Values:

enumerator **SMS_Validity_NotAvailable**

enumerator **SMS_Validity_RelativeFormat**

struct **GSM_SMSValidity**

Structure for validity of SMS messages

Public Members

GSM_ValidityPeriod **Relative**

Value defines period for relative format

struct **GSM_SMSC**

Structure for SMSC (SMS Center) information.

Public Members

int **Location**

Number of the SMSC on SIM

unsigned char **Name**[(GSM_MAX_SMSC_NAME_LENGTH + 1) * 2]

Name of the SMSC

unsigned char **Number**[(GSM_MAX_NUMBER_LENGTH + 1) * 2]

SMSC phone number.

GSM_SMSValidity **Validity**

Validity of SMS messages.

GSM_SMSFormat **Format**

Format of sent SMS messages.

unsigned char **DefaultNumber**[(GSM_MAX_NUMBER_LENGTH + 1) * 2]

Default recipient number. In old DCT3 ignored

enum **GSM_SMS_State**

Status of SMS message.

Values:

enumerator **SMS_Sent**

enumerator **SMS_UnSent**

enumerator **SMS_Read**

enumerator **SMS_UnRead**

enum **GSM_Coding_Type**

Coding type of SMS.

Values:

enumerator **SMS_Coding_Unicode_No_Compression**

Unicode

enumerator **SMS_Coding_Unicode_Compression**

enumerator **SMS_Coding_Default_No_Compression**

Default GSM alphabet.

enumerator **SMS_Coding_Default_Compression**

enumerator **SMS_Coding_8bit**

8-bit.

enum **GSM_UDH**

Types of UDH (User Data Header).

Values:

enumerator **UDH_NoUDH**

enumerator **UDH_ConcatenatedMessages**

Linked SMS.

enumerator **UDH_ConcatenatedMessages16bit**

Linked SMS with 16 bit reference.

enumerator **UDH_DisableVoice**

enumerator **UDH_DisableFax**

enumerator **UDH_DisableEmail**

enumerator **UDH_EnableVoice**

enumerator **UDH_EnableFax**

enumerator **UDH_EnableEmail**

enumerator **UDH_VoidSMS**

enumerator **UDH_NokiaRingtone**

enumerator **UDH_NokiaRingtoneLong**

enumerator **UDH_NokiaOperatorLogo**

enumerator **UDH_NokiaOperatorLogoLong**

enumerator **UDH_NokiaCallerLogo**

enumerator **UDH_NokiaWAP**

enumerator **UDH_NokiaWAPLong**

enumerator **UDH_NokiaCalendarLong**

enumerator **UDH_NokiaProfileLong**

enumerator **UDH_NokiaPhonebookLong**

enumerator **UDH_UserUDH**

enumerator **UDH_MMSIndicatorLong**

struct **GSM_UDHHeader**

Structure for User Data Header.

Public Members

GSM_UDH **Type**

UDH type.

int **Length**

UDH length.

unsigned char **Text**[GSM_MAX_UDH_LENGTH]

UDH text.

int **ID8bit**

8-bit ID, when required (-1 otherwise).

int **ID16bit**

16-bit ID, when required (-1 otherwise).

int **PartNumber**

Number of current part.

int **AllParts**

Total number of parts.

enum **GSM_SMSMessageType**

TP-Message-Type-Indicator. See GSM 03.40 section 9.2.3.1.

Values:

enumerator **SMS_Deliver**

SMS in Inbox.

enumerator **SMS_Status_Report**

Delivery Report

enumerator **SMS_Submit**

SMS for sending or in Outbox

struct **GSM_SMSMessage**

SMS message data.

Public Members

unsigned char **ReplaceMessage**

Message to be replaced.

gboolean **RejectDuplicates**

Whether to reject duplicates.

GSM_UDHHeader **UDH**

UDH (User Data Header)

unsigned char **Number**[(GSM_MAX_NUMBER_LENGTH + 1) * 2]

Sender or recipient number.

GSM_SMSC **SMSC**

SMSC (SMS Center)

GSM_MemoryType **Memory**

For saved SMS: where exactly it's saved (SIM/phone)

int **Location**

For saved SMS: location of SMS in memory.

int **Folder**

For saved SMS: number of folder, where SMS is saved

gboolean **InboxFolder**

For saved SMS: whether SMS is really in Inbox.

int **Length**

Length of the SMS message.

GSM_SMS_State **State**

Status (read/unread/...) of SMS message.

unsigned char **Name**[(GSM_MAX_SMS_NAME_LENGTH + 1) * 2]

Name in Nokia with SMS memory (6210/7110, etc.) Ignored in other.

unsigned char **Text**[(GSM_MAX_SMS_LENGTH + 1) * 2]

Text for SMS.

GSM_SMSMessageType **PDU**

Type of message.

GSM_Coding_Type **Coding**

Type of coding.

GSM_DateTime **DateTime**

Date and time, when SMS was saved or sent

GSM_DateTime **SMSCTime**

Date of SMSC response in DeliveryReport messages.

unsigned char **DeliveryStatus**

In delivery reports: status.

gboolean **ReplyViaSameSMSC**

Indicates whether "Reply via same center" is set.

signed char **Class**

SMS class (0 is flash SMS, 1 is normal one).

unsigned char **MessageReference**

Message reference.

struct **GSM_SMSMessageLayout**

Public Members

unsigned char **Text**

TP-User-Data. GSM 03.40 section 9.2.3.24.

unsigned char **Number**

- In SMS-Deliver: TP-Originating-Address. GSM 03.40 section 9.2.3.7.
- In SMS-Submit: TP-Destination-Address. GSM 03.40 section 9.2.3.8.
- In SMS-Status-Report: TP-Recipient-Address. GSM 03.40 section 9.2.3.14.

unsigned char **SMSCNumber**

SMSC number

unsigned char **TPDCS**

TP-Data-Coding-Scheme. GSM 03.40 section 9.2.3.10. Contains alphabet type, SMS class (and some others)

unsigned char **DateTime**

- For SMS-Submit: TP-Validity-Period. GSM 03.40 section 9.2.3.12.
- For SMS-Status-Report: TP-Discharge Time. GSM 03.40 section 9.2.3.13.

unsigned char **SMSCTime**

TP-Service-Centre-Time-Stamp in SMS-Status-Report. GSM 03.40 section 9.2.3.11.

unsigned char **TPStatus**

TP-Status in SMS-Status-Report. GSM 03.40 section 9.2.3.15.

unsigned char **TPUDL**

TP-User-Data-Length. GSM 03.40 section 9.2.3.16.

unsigned char **TPVP**

TP-Validity Period in SMS-Submit. GSM 03.40 section 9.2.3.12.

unsigned char **firstbyte**

Byte contains in SMS-Deliver:

- TP-Message-Type-Indicator (2 bits) GSM 03.40 section 9.2.3.1
- TP-More-Messages-To-Send (1 bit). GSM 03.40 section 9.2.3.2
- TP-Reply-Path (1 bit). GSM 03.40 section 9.2.3.17
- TP-User-Data-Header-Indicator (1 bit). GSM 03.40 section 9.2.3.23
- TP-Status-Report-Indicator (1 bit). GSM 03.40 section 9.2.3.4

Byte contains in SMS-Submit:

- TP-Message-Type-Indicator (2 bits) GSM 03.40 section 9.2.3.1
- TP-Reject-Duplicates (1 bit). GSM 03.40 section
- TP-Validity-Period-Format (2 bits).GSM 03.40 section 9.2.3.3
- TP-Reply-Path (1 bit). GSM 03.40 section 9.2.3.17
- TP-User-Data-Header-Indicator (1 bit). GSM 03.40 section 9.2.3.23
- TP-Status-Report-Request (1 bit). GSM 03.40 section 9.2.3.5

unsigned char **TPMR**

TP-Message Reference in SMS-Submit. GSM 03.40 section 9.2.3.6

unsigned char **TPPID**

TP-Protocol-Identifier. GSM 03.40 section 9.2.3.9

struct **GSM_OneSMSFolder**

Information about SMS folder.

Public Members

gboolean **InboxFolder**

Whether it is inbox.

gboolean **OutboxFolder**

Whether it is outbox.

GSM_MemoryType **Memory**

Where exactly it's saved.

unsigned char **Name**[(GSM_MAX_SMS_FOLDER_NAME_LEN + 1) * 2]

Name of the folder

struct **GSM_SMSFolders**

List of SMS folders.

Public Members

GSM_OneSMSFolder **Folder**[GSM_MAX_SMS_FOLDERS]

Array of structures holding information about each folder.

int **Number**

Number of SMS folders.

struct **GSM_SiemensOTASMSInfo**

Siemens OTA data.

struct **GSM_MultiSMSMessage**

Multiple SMS messages, used for Smart Messaging 3.0/EMS.

Public Members

int **Number**

Number of messages.

GSM_SMSMessage **SMS**[GSM_MAX_MULTI_SMS]

Array of SMSes.

gboolean **Processed**

Boolean flag for processing

struct **GSM_OneMMSFolder**

Information about MMS folder.

Public Members

gboolean **InboxFolder**

Whether it is really inbox.

char **Name**[(GSM_MAX_MMS_FOLDER_NAME_LEN + 1) * 2]

Name for MMS folder.

struct **GSM_MMSFolders**

List of MMS folders.

Public Members

unsigned char **Number**

Number of MMS folders.

GSM_OneMMSFolder **Folder**[GSM_MAX_MMS_FOLDERS]

Array of structures holding information about each folder.

enum **EncodeMultiPartSMSID**

ID during packing SMS for Smart Messaging 3.0, EMS and other

Values:

enumerator **SMS_Text**

1 text SMS.

enumerator **SMS_ConcatenatedTextLong**

Contacenated SMS, when longer than 1 SMS.

enumerator **SMS_ConcatenatedAutoTextLong**

Contacenated SMS, auto Default/Unicode coding.

enumerator **SMS_ConcatenatedTextLong16bit**

enumerator **SMS_ConcatenatedAutoTextLong16bit**

enumerator **SMS_NokiaProfileLong**

Nokia profile = Name, Ringtone, ScreenSaver

enumerator **SMS_NokiaPictureImageLong**

Nokia Picture Image + (text)

enumerator **SMS_NokiaScreenSaverLong**

Nokia screen saver + (text)

enumerator **SMS_NokiaRingtone**

Nokia ringtone - old SM2.0 format, 1 SMS

enumerator **SMS_NokiaRingtoneLong**

Nokia ringtone contacenated, when very long

enumerator **SMS_NokiaOperatorLogo**

Nokia 72x14 operator logo, 1 SMS

enumerator **SMS_NokiaOperatorLogoLong**

Nokia 72x14 op logo or 78x21 in 2 SMS

enumerator **SMS_NokiaCallerLogo**

Nokia 72x14 caller logo, 1 SMS

enumerator **SMS_NokiaWAPBookmarkLong**

Nokia WAP bookmark in 1 or 2 SMS

enumerator **SMS_NokiaWAPSettingsLong**

Nokia WAP settings in 2 SMS

enumerator **SMS_NokiaMMSSettingsLong**

Nokia MMS settings in 2 SMS

enumerator **SMS_NokiaVCARD10Long**

Nokia VCARD 1.0 - only name and default number

enumerator **SMS_NokiaVCARD21Long**

Nokia VCARD 2.1 - all numbers + text

enumerator **SMS_NokiaVCALENDAR10Long**
Nokia VCALENDAR 1.0 - can be in few sms

enumerator **SMS_NokiaVTODOLong**

enumerator **SMS_VCARD10Long**

enumerator **SMS_VCARD21Long**

enumerator **SMS_DisableVoice**

enumerator **SMS_DisableFax**

enumerator **SMS_DisableEmail**

enumerator **SMS_EnableVoice**

enumerator **SMS_EnableFax**

enumerator **SMS_EnableEmail**

enumerator **SMS_VoidSMS**

enumerator **SMS_EMSSound10**
IMelody 1.0

enumerator **SMS_EMSSound12**
IMelody 1.2

enumerator **SMS_EMSSonyEricssonSound**
IMelody without header - SonyEricsson extension

enumerator **SMS_EMSSound10Long**
IMelody 1.0 with UPI.

enumerator **SMS_EMSSound12Long**
IMelody 1.2 with UPI.

enumerator **SMS_EMSSonyEricssonSoundLong**
IMelody without header with UPI.

enumerator **SMS_EMSPredefinedSound**

enumerator **SMS_EMSPredefinedAnimation**

enumerator **SMS_EMSAnimation**

enumerator **SMS_EMSFixedBitmap**
Fixed bitmap of size 16x16 or 32x32.

enumerator **SMS_EMSVariableBitmap**

enumerator **SMS_EMSVariableBitmapLong**

enumerator **SMS_MMSIndicatorLong**
MMS message indicator.

enumerator **SMS_WAPIndicatorLong**

enumerator **SMS_AlcatelMonoBitmapLong**
Variable bitmap with black and white colors

enumerator **SMS_AlcatelMonoAnimationLong**
Variable animation with black and white colors

enumerator **SMS_AlcatelSMSTemplateName**

enumerator **SMS_SiemensFile**
Siemens OTA

enumerator **SMS_USSD**

struct **GSM_MultiPartSMSEntry**
Entry of multipart SMS.

struct **GSM_MultiPartSMSInfo**
Multipart SMS information.

enum **MMSAddressType**
MMS address type.
Values:

enumerator **MMSADDRESS_PHONE**

enumerator **MMSADDRESS_UNKNOWN**

struct **GSM_EncodedMultiPartMMSEntry**
MMS entry.

Public Members

unsigned char **ContentType**[400]
CT in Unicode

unsigned char **SMIL**[400]
Smil ID in Unicode

struct **GSM_EncodedMultiPartMMSInfo**
MMS part.

Public Members

unsigned char **Source**[200]
in Unicode

unsigned char **Destination**[200]
in Unicode

unsigned char **CC**[200]
in Unicode

unsigned char **Subject**[200]
in Unicode

unsigned char **ContentType**[400]
CT in Unicode

unsigned char **MSGType**[50]
no Unicode

GSM_EncodedMultiPartMMSEntry **Entries**[GSM_MAX_MULTI_MMS]
Subparts.

5.3.17 Miscellaneous

size_t **GetLine**(FILE *File, char *Line, int count)
Reads single line from file.

Parameters

- **File** – File descriptor to read from.
- **Line** – Buffer where t ostore result.
- **count** – Maximal length of text which can be stored in buffer.

Returns
Length of read line, -1 on error.

const char *GetGammuVersion(void)

Gets Gammu library version.

const char *GetCompiler(void)

Gets compiler which was used to compile Gammu library.

const char *GetOS(void)

Gets host OS.

const char *GetGammuLocalePath(void)

Returns path to Gammu locales.

void GSM_InitLocales(const char *path)

Initializes locales. This sets up things needed for proper string conversion from local charset as well as initializes gettext based translation.

Parameters

- **path** – Path to gettext translation. If NULL compiled in default is used.

void EncodeHexBin(char *dest, const unsigned char *src, size_t len)

Encodes text to hexadecimal binary representation.

gboolean GSM_IsNewerVersion(const char *latest_version, const char *current_version)

Returns TRUE if firmware version is newer.

Parameters

- **latest_version** – String containing version (eg. latest available).
- **current_version** – String containing version (eg. current one).

Returns
True if latest_version > current_version.

5.3.18 Nokia

void NOKIA_GetDefaultCallerGroupName(*GSM_Bitmap* *Bitmap)

Gets default caller group name.

Parameters

- **Bitmap** – Storage for default bitmap.

void NOKIA_GetDefaultProfileName(*GSM_Profile* *Profile)

Gets default profile name.

Parameters

- **Profile** – Storage for default profile.

5.3.19 Ringtone

GSM_Error **PHONE_RTTLPlayOneNote**(*GSM_StateMachine* *s, *GSM_RingNote* note, *gboolean* first)

Play one note using state machine interface.

GSM_Error **PHONE_Beep**(*GSM_StateMachine* *s)

Makes phone beek using state machine interface.

GSM_Error **GSM_GetRingtone**(*GSM_StateMachine* *s, *GSM_Ringtone* *Ringtone, *gboolean* PhoneRingtone)

Gets ringtone from phone.

GSM_Error **GSM_SetRingtone**(*GSM_StateMachine* *s, *GSM_Ringtone* *Ringtone, int *maxlength)

Sets ringtone in phone.

GSM_Error **GSM_GetRingtonesInfo**(*GSM_StateMachine* *s, *GSM_AllRingtonesInfo* *Info)

Acquires ringtone informaiton.

GSM_Error **GSM_DeleteUserRingtones**(*GSM_StateMachine* *s)

Deletes user defined ringtones from phone.

GSM_Error **GSM_PlayTone**(*GSM_StateMachine* *s, int Herz, unsigned char Volume, *gboolean* start)

Plays tone.

GSM_Error **GSM_RingtoneConvert**(*GSM_Ringtone* *dest, *GSM_Ringtone* *src, *GSM_RingtoneFormat* Format)

GSM_Error **GSM_ReadRingtoneFile**(char *FileName, *GSM_Ringtone* *ringtone)

GSM_Error **GSM_SaveRingtoneFile**(char *FileName, *GSM_Ringtone* *ringtone)

GSM_Error **GSM_SaveRingtoneOtt**(FILE *file, *GSM_Ringtone* *ringtone)

GSM_Error **GSM_SaveRingtoneMidi**(FILE *file, *GSM_Ringtone* *ringtone)

GSM_Error **GSM_SaveRingtoneIMelody**(FILE *file, *GSM_Ringtone* *ringtone)

GSM_Error **GSM_SaveRingtoneWav**(FILE *file, *GSM_Ringtone* *ringtone)

GSM_Error **GSM_SaveRingtoneRttl**(FILE *file, *GSM_Ringtone* *ringtone)

const unsigned char ***GSM_GetRingtoneName**(const *GSM_AllRingtonesInfo* *Info, const int ID)

Returns ringtone name, NULL if not found.

int **GSM_RTTLGetTempo**(int Beats)

enum **GSM_RingNoteStyle**

Values:

enumerator **NaturalStyle**

Natural style (rest between notes)

enumerator **ContinuousStyle**

Continuous style (no rest between notes)

enumerator **StaccatoStyle**

Staccato style (shorter notes and longer rest period)

enumerator **INVALIDStyle**

enum **GSM_RingNoteNote**

Values:

enumerator **Note_Pause**

enumerator **Note_C**

enumerator **Note_Cis**

enumerator **Note_D**

enumerator **Note_Dis**

enumerator **Note_E**

enumerator **Note_F**

enumerator **Note_Fis**

enumerator **Note_G**

enumerator **Note_Gis**

enumerator **Note_A**

enumerator **Note_Ais**

enumerator **Note_H**

enumerator **Note_INVALID**

enum **GSM_RingNoteDuration**

Values:

enumerator **Duration_Full**

enumerator **Duration_1_2**

enumerator **Duration_1_4**

enumerator **Duration_1_8**

enumerator **Duration_1_16**

enumerator **Duration_1_32**

enumerator **Duration_INVALID**

enum **GSM_RingNoteDurationSpec**

Values:

enumerator **NoSpecialDuration**

enumerator **DottedNote**

enumerator **DoubleDottedNote**

enumerator **Length_2_3**

enumerator **DurationSpec_INVALID**

enum **GSM_RingNoteScale**

Values:

enumerator **Scale_55**

55 Hz for note A

enumerator **Scale_110**

110 Hz for note A

enumerator **Scale_220**

enumerator **Scale_440**

first scale for Nokia

enumerator **Scale_880**

enumerator **Scale_1760**

enumerator **Scale_3520**

last scale for Nokia

enumerator **Scale_7040**

enumerator **Scale_14080**

struct **GSM_RingNote**

enum **GSM_RingCommandType**

Values:

enumerator **RING_Note**

enumerator **RING_EnableVibra**

enumerator **RING_DisableVibra**

enumerator **RING_EnableLight**

enumerator **RING_DisableLight**

enumerator **RING_EnableLED**

enumerator **RING_DisableLED**

enumerator **RING_Repeat**

struct **GSM_RingCommand**

struct **GSM_NoteRingtone**

struct **GSM_NokiaBinaryRingtone**

struct **GSM_BinaryTone**

enum **GSM_RingtoneFormat**

Values:

enumerator **RING_NOTETONE**

enumerator **RING_NOKIABINARY**

enumerator **RING_MIDI**

enumerator **RING_MMF**

struct **GSM_Ringtone**

Structure for saving various ringtones formats

Public Members

GSM_NokiaBinaryRingtone **NokiaBinary**

Ringtone saved in one of three formats

GSM_RingtoneFormat **Format**

Ringtone format

unsigned char **Name**[(GSM_MAX_RINGTONE_NAME_LENGTH + 1) * 2]

Ringtone name

int **Location**

Ringtone location

struct **GSM_RingtoneInfo**

Public Members

int **Group**

Nokia specific

struct **GSM_AllRingtonesInfo**

5.3.20 Security

GSM_Error **GSM_EnterSecurityCode**(*GSM_StateMachine* *s, *GSM_SecurityCode* *Code)

Enters security code (PIN, PUK,...) .

GSM_Error **GSM_GetSecurityStatus**(*GSM_StateMachine* *s, *GSM_SecurityCodeType* *Status)

Queries whether some security code needs to be entered.

enum **GSM_SecurityCodeType**

Definition of security codes.

Values:

enumerator **SEC_SecurityCode**

Security code.

enumerator **SEC_Pin**

PIN.

enumerator **SEC_Pin2**

PIN 2.

enumerator **SEC_Puk**
PUK.

enumerator **SEC_Puk2**
PUK 2.

enumerator **SEC_None**
Code not needed.

enumerator **SEC_Phone**
Phone code needed.

enumerator **SEC_Network**
Network code needed.

struct **GSM_SecurityCode**
Security code definition.

Public Members

GSM_SecurityCodeType **Type**
Type of the code.

char **Code**[GSM_SECURITY_CODE_LEN + 1]
Actual code.

char **NewPIN**[GSM_SECURITY_CODE_LEN + 1]
New PIN code.

Some phones require to set PIN on entering PUK, you can provide it here.

5.3.21 Settings

GSM_Error **GSM_GetLocale**(*GSM_StateMachine* *s, *GSM_Locale* *locale)
Gets locale from phone.

GSM_Error **GSM_SetLocale**(*GSM_StateMachine* *s, *GSM_Locale* *locale)
Sets locale of phone.

GSM_Error **GSM_GetSyncMLSettings**(*GSM_StateMachine* *s, *GSM_SyncMLSettings* *settings)
Acquires SyncML settings.

GSM_Error **GSM_SetSyncMLSettings**(*GSM_StateMachine* *s, *GSM_SyncMLSettings* *settings)
Changes SyncML settings.

GSM_Error **GSM_GetChatSettings**(*GSM_StateMachine* *s, *GSM_ChatSettings* *settings)
Acquires chat/presence settings.

GSM_Error **GSM_SetChatSettings**(*GSM_StateMachine* *s, *GSM_ChatSettings* *settings)
Changes chat/presence settings.

GSM_Error **GSM_GetMMSSettings**(*GSM_StateMachine* *s, *GSM_MultiWAPSettings* *settings)
Acquires MMS settings.

GSM_Error **GSM_SetMMSSettings**(*GSM_StateMachine* *s, *GSM_MultiWAPSettings* *settings)
Changes MMS settings.

GSM_Error **GSM_SetAutoNetworkLogin**(*GSM_StateMachine* *s)
Enables network auto login.

GSM_Error **GSM_Reset**(*GSM_StateMachine* *s, *gboolean* hard)
Performs phone reset.

GSM_Error **GSM_ResetPhoneSettings**(*GSM_StateMachine* *s, *GSM_ResetSettingsType* Type)
Resets phone settings.

GSM_Error **GSM_GetProfile**(*GSM_StateMachine* *s, *GSM_Profile* *Profile)
Reads profile.

GSM_Error **GSM_SetProfile**(*GSM_StateMachine* *s, *GSM_Profile* *Profile)
Updates profile.

GSM_Error **GSM_GetFMStation**(*GSM_StateMachine* *s, *GSM_FMStation* *FMStation)
Reads FM station.

GSM_Error **GSM_SetFMStation**(*GSM_StateMachine* *s, *GSM_FMStation* *FMStation)
Sets FM station.

GSM_Error **GSM_ClearFMStations**(*GSM_StateMachine* *s)
Clears defined FM stations.

GSM_Error **GSM_GetGPRSAccessPoint**(*GSM_StateMachine* *s, *GSM_GPRSAccessPoint* *point)
Gets GPRS access point.

GSM_Error **GSM_SetGPRSAccessPoint**(*GSM_StateMachine* *s, *GSM_GPRSAccessPoint* *point)
Sets GPRS access point.

struct **GSM_SyncMLSettings**

enum **GSM_ResetSettingsType**
Values:

enumerator **GSM_RESET_PHONESETTINGS**

enumerator **GSM_RESET_USERINTERFACE**

enumerator **GSM_RESET_USERINTERFACE_PHONESETTINGS**

enumerator **GSM_RESET_DEVICE**

enumerator **GSM_RESET_FULLFACTORY**

struct **GSM_ChatSettings**

enum **GSM_Profile_Feat_Value**

Values:

enumerator **PROFILE_KEYPAD_LEVEL1**

enumerator **PROFILE_KEYPAD_LEVEL2**

enumerator **PROFILE_KEYPAD_LEVEL3**

enumerator **PROFILE_KEYPAD_OFF**

enumerator **PROFILE_CALLALERT_RINGING**

enumerator **PROFILE_CALLALERT_BEEPONCE**

enumerator **PROFILE_CALLALERT_OFF**

enumerator **PROFILE_CALLALERT_RINGONCE**

enumerator **PROFILE_CALLALERT_ASCENDING**

enumerator **PROFILE_CALLALERT_CALLERGROUPS**

enumerator **PROFILE_VOLUME_LEVEL1**

enumerator **PROFILE_VOLUME_LEVEL2**

enumerator **PROFILE_VOLUME_LEVEL3**

enumerator **PROFILE_VOLUME_LEVEL4**

enumerator **PROFILE_VOLUME_LEVEL5**

enumerator **PROFILE_MESSAGE_NOTONE**

enumerator **PROFILE_MESSAGE_STANDARD**

enumerator **PROFILE_MESSAGE_SPECIAL**

enumerator **PROFILE_MESSAGE_BEEPONCE**

enumerator **PROFILE_MESSAGE_ASCENDING**

enumerator **PROFILE_MESSAGE_PERSONAL**

enumerator **PROFILE_VIBRATION_OFF**

enumerator **PROFILE_VIBRATION_ON**

enumerator **PROFILE_VIBRATION_FIRST**

enumerator **PROFILE_WARNING_ON**

enumerator **PROFILE_WARNING_OFF**

enumerator **PROFILE_AUTOANSWER_ON**

enumerator **PROFILE_AUTOANSWER_OFF**

enumerator **PROFILE_LIGHTS_OFF**

enumerator **PROFILE_LIGHTS_AUTO**

enumerator **PROFILE_SAVER_ON**

enumerator **PROFILE_SAVER_OFF**

enumerator **PROFILE_SAVER_TIMEOUT_5SEC**

enumerator **PROFILE_SAVER_TIMEOUT_20SEC**

enumerator **PROFILE_SAVER_TIMEOUT_1MIN**

enumerator **PROFILE_SAVER_TIMEOUT_2MIN**

enumerator **PROFILE_SAVER_TIMEOUT_5MIN**

enumerator **PROFILE_SAVER_TIMEOUT_10MIN**

enum **GSM_Profile_Feat_ID**

Values:

enumerator **Profile_KeypadTone**

enumerator **Profile_CallAlert**

enumerator **Profile_RingtoneVolume**

enumerator **Profile_MessageTone**

enumerator **Profile_Vibration**

enumerator **Profile_WarningTone**

enumerator **Profile_AutoAnswer**

enumerator **Profile_Lights**

enumerator **Profile_ScreenSaverTime**

enumerator **Profile_ScreenSaver**

enumerator **Profile_ScreenSaverNumber**

enumerator **Profile_RingtoneID**

enumerator **Profile_MessageToneID**

enumerator **Profile_CallerGroups**

struct **GSM_Profile**

It contains phone profiles

Public Members

int **Location**

Profile number

char **Name**[40 * 2]

Profile name

gboolean **DefaultName**

Is it default name for profile ?

struct **GSM_FMStation**

struct **GSM_GPRSAccessPoint**

enum **GSM_DateFormat**

Values:

enumerator **GSM_Date_DDMMYYYY**

enumerator **GSM_Date_MMDDYYYY**

enumerator **GSM_Date_YYYYMMDD**

enumerator **GSM_Date_DDMMMYY**

enumerator **GSM_Date_MMDDYY**

enumerator **GSM_Date_DDMMYY**

enumerator **GSM_Date_YYMMDD**

enumerator **GSM_Date_OFF**

struct **GSM_Locale**

struct **GSM_Profile_PhoneTableValue**

5.3.22 SMSD

GSM_Error **SMSD_InjectSMS**(*GSM_SMSDConfig* *Config, *GSM_MultiSMSMessage* *sms, char *NewID)

Enqueues SMS message in SMS daemon queue.

Parameters

- **Config** – SMSD configuration pointer.
- **sms** – Message data to send.
- **NewID** – Pointer to string where ID of new message will be written. Can be NULL and then it is ignored.

Returns

Error code

GSM_Error **SMSD_GetStatus**(*GSM_SMSDConfig* *Config, *GSM_SMSDStatus* *status)

Gets SMSD status via shared memory.

Parameters

- **Config** – SMSD configuration pointer.
- **status** – pointer where status will be copied

Returns
Error code

GSM_Error **SMSD_Shutdown**(*GSM_SMSDConfig* *Config)

Flags SMSD daemon to terminate itself gracefully.

Parameters

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

Returns
Error code

GSM_Error **SMSD_ReadConfig**(const char *filename, *GSM_SMSDConfig* *Config, *gboolean* uselog)

Reads SMSD configuration.

Parameters

- **filename** – File name of configuration.
- **Config** – Pointer to SMSD configuration data.
- **uselog** – Whether to log errors to configured log.

Returns
Error code

GSM_Error **SMSD_MainLoop**(*GSM_SMSDConfig* *Config, *gboolean* exit_on_failure, int max_failures)

Main SMS daemon loop. It connects to phone, scans for messages and sends messages from inbox. Can be interrupted by SMSD_Shutdown.

See also:

SMSD_Shutdown

Parameters

- **Config** – Pointer to SMSD configuration data.
- **exit_on_failure** – Whether failure should lead to terminaton of program.
- **max_failures** – Maximal number of failures after which SMSD will terminate. Use 0 to not terminate on failures.

Returns
Error code

GSM_SMSDConfig ***SMSD_NewConfig**(const char *name)

Creates new SMSD configuration.

Parameters

- **name** – Name of process, will be used for logging. If NULL, gammu-smsd text is used.

Returns
Pointer to SMSD configuration data block.

void **SMSD_FreeConfig**(*GSM_SMSDConfig* *config)

Frees SMSD configuration.

Parameters

- **config** – Pointer to SMSD configuration data.

struct **GSM_SMSDStatus**

Status structure, which can be found in shared memory (if supported on platform).

Public Members

int **Version**

Version of this structure (2 for now).

char **PhoneID**[SMSD_TEXT_LENGTH + 1]

PhoneID from configuration.

char **Client**[SMSD_TEXT_LENGTH + 1]

Client software name.

GSM_BatteryCharge **Charge**

Current phone battery state.

GSM_SignalQuality **Network**

Current network state.

int **Received**

Number of received messages.

int **Sent**

Number of sent messages.

int **Failed**

Number of messages which failed to be send.

char **IMEI**[GSM_MAX_IMEI_LENGTH + 1]

Phone IMEI.

char **IMSI**[GSM_MAX_INFO_LENGTH + 1]

SIM IMSI.

GSM_NetworkInfo **NetInfo**

Network information.

typedef struct _GSM_SMSDConfig **GSM_SMSDConfig**

SMSD configuration data, these are not expected to be manipulated directly by application.

5.3.23 State machine

GSM_Error **GSM_InitConnection_Log**(*GSM_StateMachine* *s, int ReplyNum, *GSM_Log_Function* log_function, void *user_data)

Initiates connection with custom logging callback.

See also:

GSM_SetDebugFunction

Parameters

- **s** – State machine data
- **ReplyNum** – Number of replies to await (usually 3).
- **log_function** – Logging function, see GSM_SetDebugFunction.
- **user_data** – User data for logging function, see GSM_SetDebugFunction.

Returns
Error code

GSM_Error **GSM_InitConnection**(*GSM_StateMachine* *s, int ReplyNum)

Initiates connection.

Parameters

- **s** – State machine data
- **ReplyNum** – Number of replies to await (usually 3).

Returns
Error code

GSM_Error **GSM_TerminateConnection**(*GSM_StateMachine* *s)

Terminates connection.

Parameters

- **s** – State machine data

Returns
Error code

GSM_Error **GSM_AbortOperation**(*GSM_StateMachine* *s)

Aborts current operation.

This is thread safe call to abort any existing operations with the phone.

Parameters

- **s** – State machine data

Returns
Error code

GSM_Error **GSM_Install**(*GSM_StateMachine* *s, const char *ExtraPath, *gboolean* Minimal)

Installs applet required for configured connection to the phone.

Parameters

- **s** – State machine data.
- **ExtraPath** – Extra path where to search for installation data.
- **Minimal** – Whether to do minimal installation (eg. without support libraries), useful for applet updates

Returns
Result of operation.

typedef struct _GSM_StateMachine **GSM_StateMachine**

Private structure holding information about phone connection. Should be allocated by *GSM_AllocStateMachine* and freed by *GSM_FreeStateMachine*.

enum **GSM_ConnectionType**

Connection types definitions.

Values:

enumerator **GCT_MBUS2**

enumerator **GCT_FBUS2**

enumerator **GCT_FBUS2DLR3**

enumerator **GCT_DKU2AT**

enumerator **GCT_DKU2PHONET**

enumerator **GCT_DKU5FBUS2**

enumerator **GCT_ARK3116FBUS2**

enumerator **GCT_FBUS2PL2303**

enumerator **GCT_FBUS2BLUE**

enumerator **GCT_FBUS2IRDA**

enumerator **GCT_PHONETBLUE**

enumerator **GCT_AT**

enumerator **GCT_BLUEGNAPBUS**

enumerator **GCT_IRDAOBEX**

enumerator **GCT_IRDAGNAPBUS**

enumerator **GCT_IRDAAT**

enumerator **GCT_IRDAPHONET**

enumerator **GCT_BLUEFBUS2**

enumerator **GCT_BLUEAT**

enumerator **GCT_BLUEPHONET**

enumerator **GCT_BLUEOBEX**

enumerator **GCT_FBUS2USB**

enumerator **GCT_BLUES60**

enumerator **GCT_PROXYGNAPBUS**

enumerator **GCT_PROXYFBUS2**

enumerator **GCT_PROXYAT**

enumerator **GCT_PROXYPHONET**

enumerator **GCT_PROXYOBEX**

enumerator **GCT_PROXYS60**

enumerator **GCT_NONE**

struct **GSM_Config**

Configuration of state machine.

Public Members

char **Model**[50]

Model from config file

char **DebugLevel**[50]

Debug level

char *Device

Device name from config file

char *Connection

Connection type as string

gboolean SyncTime

Synchronize time on startup?

gboolean LockDevice

Lock device ? (Unix)

char *DebugFile

Name of debug file

gboolean StartInfo

Display something during start ?

gboolean UseGlobalDebugFile

Should we use global debug file?

char TextReminder[32]

Text for reminder calendar entry category in local language

char TextMeeting[32]

Text for meeting calendar entry category in local language

char TextCall[32]

Text for call calendar entry category in local language

char TextBirthday[32]

Text for birthday calendar entry category in local language

char TextMemo[32]

Text for memo calendar entry category in local language

GSM_Feature PhoneFeatures[GSM_MAX_PHONE_FEATURES + 1]

Phone features override.

int CNMIParams[5]

Used to override default CNMI arguments for generic AT protocol.

typedef void (*GSM_Log_Function)(const char *text, void *data)

Callback function for logging.

Param text

Text to be printed,

will be also sent (as a separate message).

Param data
Arbitrary logger data, as passed to *GSM_InitConnection_Log*.

int **GSM_ReadDevice**(*GSM_StateMachine* *s, *gboolean* waitforreply)

Attempts to read data from phone. This can be used for getting status of incoming events, which would not be found out without polling device.

Parameters

- **s** – State machine data
- **waitforreply** – Whether to wait for some event

Returns
Number of read bytes

gboolean **GSM_IsConnected**(*GSM_StateMachine* *s)

Detects whether state machine is connected.

Parameters

- **s** – State machine data

Returns
Whether phone is connected.

GSM_Error **GSM_FindGammuRC**(*INI_Section* **result, const char *force_config)

Finds and reads gammu configuration file. The search order depends on platform. On POSIX systems it looks for ~/.gammurc and then for /etc/gammurc, on Windows for gammurc in Application data folder, then in home and last fallback is in current driectory.

Parameters

- **result** – Ini file representation
- **force_config** – Forcing of custom path instead of autodetected one (if NULL, autodetection is performed).

Returns
Error code

GSM_Error **GSM_ReadConfig**(*INI_Section* *cfg_info, *GSM_Config* *cfg, int num)

Processes gammu configuration.

See also:

GSM_FallbackConfig

Parameters

- **cfg_info** – Ini file representation.
- **cfg** – Where to store configuration.
- **num** – Number of section to read.

Returns
Whether we got valid configuration. Especially check for ERR_USING_DEFAULTS.

GSM_Config ***GSM_GetConfig**(*GSM_StateMachine* *s, int num)

Gets gammu configuration from state machine. This actually returns pointer to internal configuration storage, so you can use it also for updating existing settings.

Parameters

- **s** – State machine data
- **num** – Number of section to read, -1 for currently used.

Returns
Pointer to configuration.

int **GSM_GetConfigNum**(const *GSM_StateMachine* *s)

Gets number of active gammu configurations.

Parameters

- **s** – State machine data

Returns
Number of sections.

void **GSM_SetConfigNum**(*GSM_StateMachine* *s, int sections)

Gets number of active gammu configurations.

Parameters

- **s** – State machine data
- **sections** – Number of sections.

GSM_StateMachine ***GSM_AllocStateMachine**(void)

Allocates new clean state machine structure. You should free it then by *GSM_FreeStateMachine*.

Returns
Pointer to state machine structure.

void **GSM_FreeStateMachine**(*GSM_StateMachine* *s)

Frees state machine structure allocated by *GSM_AllocStateMachine*.

Parameters

- **s** – Pointer to state machine structure.

GSM_ConnectionType **GSM_GetUsedConnection**(*GSM_StateMachine* *s)

Gets number of active gammu configurations.

Parameters

- **s** – State machine data

Returns
Connection type.

5.3.24 Types

typedef int **gboolean**

gboolean definition, compatible with glib.

5.3.25 Unicode

size_t **UnicodeLength**(const unsigned char *str)

Returns length of unicode string.

char ***DecodeUnicodeString**(const unsigned char *src)

Converts string to locale charset.

Returns
Pointer to static string.

char ***DecodeUnicodeConsole**(const unsigned char *src)

Converts string to console charset.

Returns
Pointer to static string.

void **DecodeUnicode**(const unsigned char *src, char *dest)

Converts string from unicode to local charset.

void **EncodeUnicode**(unsigned char *dest, const char *src, size_t len)

Encodes string from local charset to unicode.

void **ReadUnicodeFile**(unsigned char *Dest, const unsigned char *Source)

Decodes unicode file data with byte order mark (BOM).

void **CopyUnicodeString**(unsigned char *Dest, const unsigned char *Source)

Copies unicode string.

gboolean **EncodeUTF8QuotedPrintable**(char *dest, const unsigned char *src)

Encodes string to UTF-8 quoted printable.

void **DecodeUTF8QuotedPrintable**(unsigned char *dest, const char *src, size_t len)

Decodes UTF-8 quoted printable string.

int **EncodeWithUTF8Alphabet**(unsigned long src, unsigned char *ret)

Encodes string to UTF-8.

Warning: doxygenfunction: Cannot find function “DecodeWithUTF8Alphabet” in doxygen xml output for project “api” from directory: /home/runner/work/gammu/gammu/build-configure/gammu-doc/xml

gboolean **DecodeHexUnicode**(unsigned char *dest, const char *src, size_t len)

Decodes string from hex quoted unicode.

void **EncodeHexUnicode**(char *dest, const unsigned char *src, size_t len)

Encodes string to hex quoted unicode.

gboolean **mywstrncmp**(const unsigned char *a, const unsigned char *b, int num)

Compares two unicode strings.

unsigned char ***mywstrstr**(const unsigned char *haystack, const unsigned char *needle)

Locates unicode substring.

gboolean **mywstrncasecmp**(const unsigned char *a, const unsigned char *b, int num)

Compares two unicode strings case insensitive.

gboolean **EncodeUTF8**(char *dest, const unsigned char *src)

Encode text to UTF-8.

void **DecodeUTF8**(unsigned char *dest, const char *src, size_t len)

Decode text from UTF-8.

gboolean **DecodeHexBin**(unsigned char *dest, const unsigned char *src, size_t len)

Decode hex encoded binary text.

Warning: doxygenfunction: Cannot find function "EncodeWithUnicodeAlphabet" in doxygen xml output for project "api" from directory: /home/runner/work/gammu/gammu/build-configure/gammu-doc/xml

Warning: doxygenfunction: Cannot find function "DecodeWithUnicodeAlphabet" in doxygen xml output for project "api" from directory: /home/runner/work/gammu/gammu/build-configure/gammu-doc/xml

5.3.26 WAP

GSM_Error **GSM_EncodeURLFile**(unsigned char *Buffer, size_t *Length, *GSM_WAPBookmark* *bookmark)

Encodes URL to VBKM file.

Parameters

- **Buffer** – Storage for text.
- **Length** – Pointer to storage, will be updated.
- **bookmark** – Bookmark to encode.

Returns

Error code.

GSM_Error **GSM_GetWAPBookmark**(*GSM_StateMachine* *s, *GSM_WAPBookmark* *bookmark)

Reads WAP bookmark.

Parameters

- **s** – State machine pointer.
- **bookmark** – Bookmark storage, need to contain location.

Returns

Error code

GSM_Error **GSM_SetWAPBookmark**(*GSM_StateMachine* *s, *GSM_WAPBookmark* *bookmark)

Sets WAP bookmark.

Parameters

- **s** – State machine pointer.
- **bookmark** – Bookmark data.

Returns
Error code

GSM_Error **GSM_DeleteWAPBookmark**(*GSM_StateMachine* *s, *GSM_WAPBookmark* *bookmark)

Deletes WAP bookmark.

Parameters

- **s** – State machine pointer.
- **bookmark** – Bookmark data, need to contain location.

Returns
Error code

GSM_Error **GSM_GetWAPSettings**(*GSM_StateMachine* *s, *GSM_MultiWAPSettings* *settings)

Acquires WAP settings.

Parameters

- **s** – State machine pointer.
- **settings** – Settings storage.

Returns
Error code

GSM_Error **GSM_SetWAPSettings**(*GSM_StateMachine* *s, *GSM_MultiWAPSettings* *settings)

Changes WAP settings.

Parameters

- **s** – State machine pointer.
- **settings** – Settings data.

Returns
Error code

struct **GSM_WAPBookmark**

WAP bookmark data.

Public Members

int **Location**

Location where it is stored.

unsigned char **Address**[(255 + 1) * 2]

Bookmark URL.

unsigned char **Title**[(50 + 1) * 2]

Bookmark title.

enum **WAPSettings_Speed**

Connection speed configuration.

Values:

enumerator **WAPSETTINGS_SPEED_9600**

enumerator **WAPSETTINGS_SPEED_14400**

enumerator **WAPSETTINGS_SPEED_AUTO**

enum **WAPSettings_Bearer**

Connection bearer configuration.

Values:

enumerator **WAPSETTINGS_BEARER_SMS**

enumerator **WAPSETTINGS_BEARER_DATA**

enumerator **WAPSETTINGS_BEARER_USSD**

enumerator **WAPSETTINGS_BEARER_GPRS**

struct **GSM_WAPSettings**

WAP setting.

Public Members

char **Title**[(20 + 1) * 2]

Settings name.

char **HomePage**[(100 + 1) * 2]

Home page.

WAPSettings_Bearer **Bearer**

Bearer of WAP connection.

gboolean **IsSecurity**

Secure connection?

gboolean **IsContinuous**

Is this connectin continuous?

gboolean **IsISDNCall**

Whether is ISDN for data bearer

gboolean **IsNormalAuthentication**

Whether is normal auth for data bearer

char **Server**[(21 + 1) * 2]

Server for sms bearer.

char **Service**[(20 + 1) * 2]

Service for sms or ussd bearer.

gboolean **IsIP**

Whether is IP, for sms or ussd bearer.

char **Code**[(10 + 1) * 2]

Code for ussd bearer.

char **IPAddress**[(20 + 1) * 2]

IP address for data or gprs.

gboolean **ManualLogin**

Login for data or gprs.

char **DialUp**[(20 + 1) * 2]

Dial up number for data or gprs.

char **User**[(50 + 1) * 2]

User name for data or gprs.

Todo:

Is length okay?

char **Password**[(50 + 1) * 2]

User password for data or gprs.

Todo:

Is length okay?

WAPSettings_Speed **Speed**

Speed settings for data or gprs.

struct **GSM_MultiWAPSettings**

Set of WAP settings.

Public Members

int **Location**

Location.

unsigned char **Number**

Number of elements in Settings.

GSM_WAPSettings **Settings**[4]

Real WAP settings.

gboolean **Active**

Whether this configuration is active.

gboolean **ReadOnly**

Whether this configuration is read only.

char **Proxy**[(100 + 1) * 2]

Proxy server.

int **ProxyPort**

Proxy port.

char **Proxy2**[(100 + 1) * 2]

Second proxy server.

int **Proxy2Port**

Second proxy port.

WAPSettings_Bearer **ActiveBearer**

Bearer of current connection.

5.4 Porting from libGammu older than 1.12.0

5.4.1 Rationale for API change

This document describes what you have to change in your code, if you used Gammu older than 1.12.0. This release came with huge changes to API, which has to be done for various reasons:

- ABI stability. - Till now almost every change in internals of any driver lead to ABI change. If we would correctly increase soname on each ABI change, we would be somewhere near 200, what is not something we could be proud of.
- Centralisation of variables cleanup. - Currently all phone drivers have to do some common things in each function. New API allows one to centralize those operations in one place.
- Exposing of internals. - Old API exposed too much of Gammu internals, what could be misused by programmers and could lead to unexpected behaviour when some internals are changed.

5.4.2 Changes you have to do in your code

Below examples expect `sm` to be state machine structure in your current code, change it to appropriate variable name if it differs.

1. Use pointer to *GSM_StateMachine* instead of it. API now do not expose this structure, so you will get compiler error. You should allocate this pointer by *GSM_AllocStateMachine()* and free by *GSM_FreeStateMachine()*.
2. Change all phone functions from `sm.Phone.Functions->SomeFunction` to `GSM_SomeFunction`. Only functions which results were stored inside state machine structure have changed signature to include results of the operation.
3. All callbacks are set by function `GSM_Set*Callback` instead of directly accessing structure.
4. Some function have been renamed to follow `GSM_*` naming conventions.

As there might be some functions still missing from new API, don't hesitate to contact author or ask on mailing list if you miss something.

API documentation can be generated using Doxygen (make apidoc in build tree) or Sphinx and is part of this manual.

See also:

libGammu

CHAPTER

SIX

GAMMU INTERNALS

Gammu project internals are a bit more complicated than required, mostly for historical reasons. Before digging into source code, you should look at *Directory structure* and *Coding Style*.

6.1 Reply functions

When phone gives answers, we check if we requested received info and we redirect it to concrete reply function, which will decode it. Different phone answers can go to one reply function let's say responsible for getting sms status.

type **GSM_Reply_Function**

Defines reply function for phone driver.

GSM_Error (***Function**)(GSM_Protocol_Message *msg, *GSM_StateMachine* *s);

Callback on reply match.

const unsigned char ***msgtype**;

String match on the message.

const size_t **subtypechar**;

Position for char match inside reply. If 0, message type is checked.

const int **subtype**;

Match for char/message type check (see above).

const GSM_Phone_RequestID **requestID**;

Match for request ID. this is filled in when calling `GSM_WaitFor()`.

There are three types of answer matching:

6.1.1 Binary

Example:

```
{N6110_ReplySaveSMSMessage,"\x14",0x03,0x05,ID_SaveSMSMessage},
```

ID_SaveSMSMessage request function reply. Frame is type "x14", 0x03 char of frame must be 0x05. If yes, we go to N6110_ReplySaveSMSMessage. Of course, things like frame type are found in protocol (here FBUS, MBUS, etc.) functions. If don't need anything more than frame type, 0x03,0x05 should be 0x00, 0x00 - it means then, that we check only frame type.

6.1.2 Text

Example:

```
{ATGEN_ReplyIncomingCallInfo,"+CLIP",0x00,0x00,ID_IncomingFrame},
```

All incoming (not requested in the moment, sent by phone, who likes us - ID_IncomingFrame) responses starting from "+CLIP" will go to the ATGEN_ReplyIncomingCallInfo.

6.1.3 Numeric

Example:

```
{S60_Reply_Generic, "", 0x00, NUM_QUIT, ID_Terminate },
```

When match string is empty and match char position is zero, matching on message type is performed.

6.1.4 Requests

This is how GSM_Reply_Function is filled. Now how to make phone requests ?

Example:

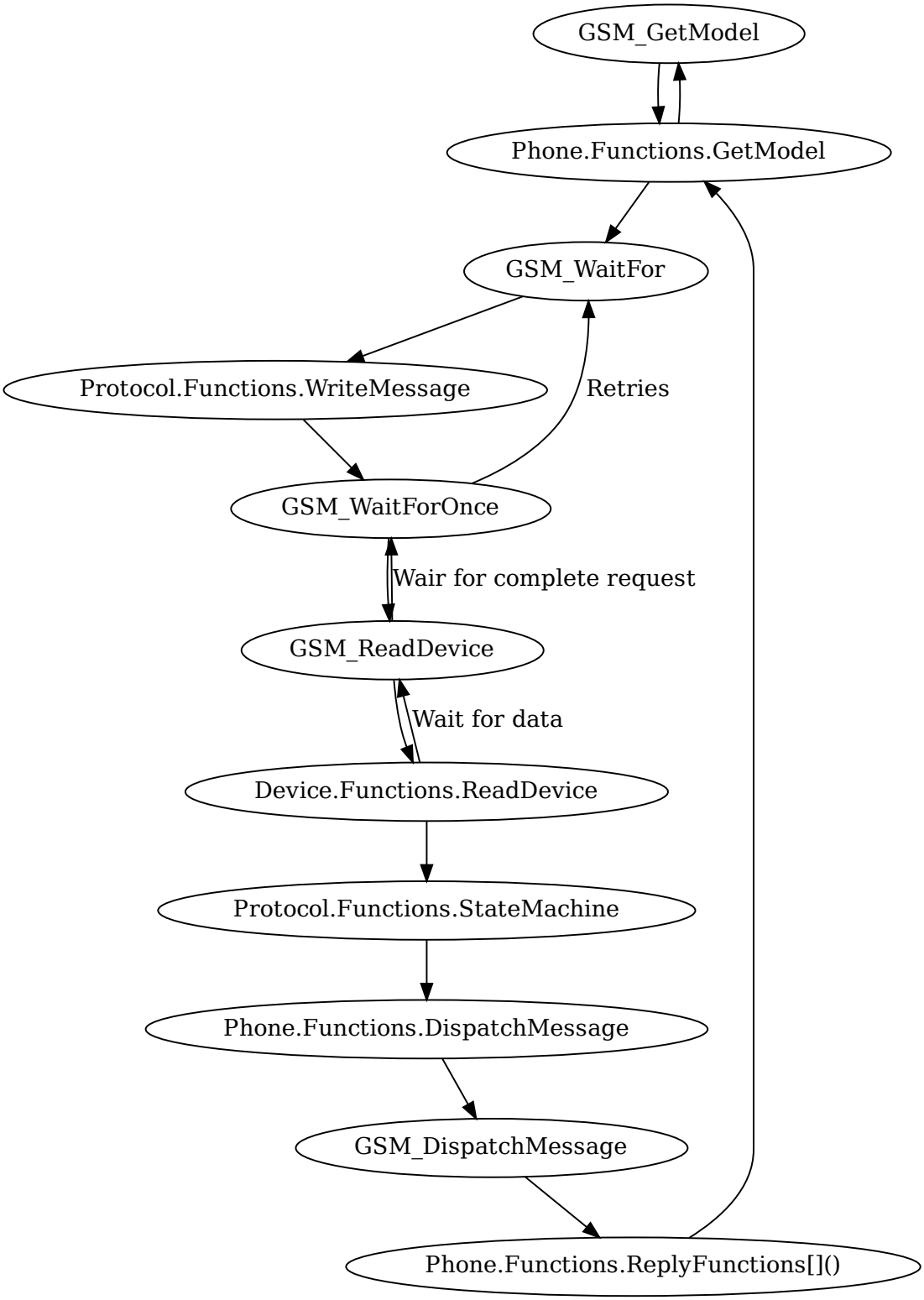
```
static GSM_Error N6110_GetMemory (GSM_StateMachine   *s,
                                  GSM_PhonebookEntry *entry)
{
  unsigned char req[] = {
       N6110_FRAME_HEADER, 0x01,
       0x00,            /* memory type */
       0x00,            /* location */
       0x00};

  req[4] = NOKIA_GetMemoryType(entry->MemoryType,N6110_MEMORY_TYPES);
  if (req[4]==0xff) return GE_NOTSUPPORTED;

  req[5] = entry->Location;

  s->Phone.Data.Memory=entry;
  dprintf("Getting phonebook entry\n");
  return GSM_WaitFor (s, req, 7, 0x03, 4, ID_GetMemory);
}
```

First we fill req according to values in *entry. Later set pointer in s->Phone.Data (it's available for reply functions and they set responses exactly to it) and use GSM_WaitFor. It uses s statemachine, sends req frame with length 7, msg type is 0x03, we wait for answer during 4 seconds, request id is ID_GetMemory. GSM_WaitFor internally checks incoming bytes from phone and redirect them to protocol functions. If they found full frame, there is checked GSM_Reply_Function, where is called ReplyFunction or showed debug info, that frame is unknown. If there is ReplyFunction, it has access to s->Phone.Data and decodes answer. Returns error or not (and this is value for GSM_WaitFor). If there is no requested answer during time, GSM_WaitFor returns GE_TIMEOUT.

6.2 State Machine

The state machine is core of libGammu operations. It gets the data from the phone and dispatches them through protocol layer to phone drivers.

To see how it operates, following figure shows example of what happens when *GSM_GetModel()* is called from the program:


GSM_GetModel
Phone.Functions.GetModel
GSM_WaitFor
Protocol.Functions.WriteMessage
Retries
GSM_WaitForOnce
Wair for complete request
GSM_ReadDevice
Wait for data
Device.Functions.ReadDevice
Protocol.Functions.StateMachine
Phone.Functions.DispatchMessage
GSM_DispatchMessage
Phone.Functions.ReplyFunctions[]()

6.3 Adding support for new phone

This document covers basic information on adding support for new phone into Gammu. It will never cover all details, but will give you basic instructions.

6.3.1 Adding support for new AT commands

The easiest situation is when all you need to support new device is to add support for new AT commands. All the protocol infrastructure is there, you only need to hook new code into right places.

The main code for AT driver is in `libgammu/phone/at/atgen.c`. At the bottom of the file, you can find two arrays, one defining driver interface (`GSM_Phone_Functions`) and second one defining callbacks (see *Reply functions* for more detailed description. You will definitely need to define callbacks for newly introduced commands, but the interface for desired functionality might already exist.

Detecting whether command is supported

As Gammu is trying to support as much phones as possible, you should try to make it automatically detect whether connected phone supports the command. This can be done on first invocation of affected operation or on connecting to phone. As we want to avoid lengthy connecting to phone, in most cases you should probe for support on first attempt to use given functionality. The code might look like following:

```
GSM_Error ATGEN_GetFoo(GSM_StateMachine *s) {
    GSM_Phone_ATGENData     *Priv = &s->Phone.Data.Priv.ATGEN;

    if (Priv->Foo_XXXX == 0) {
        ATGEN_CheckXXXX(s);
    }

    if (Priv->Foo_XXXX == AT_AVAILABLE) {
        /* Perform reading */
    }

    /* Fail with error or fallback to other methods */
    return ERR_NOTSUPPORTED;
}

GSM_Error ATGEN_CheckXXXX(GSM_StateMachine *s) {
    GSM_Error       error;
    GSM_Phone_ATGENData     *Priv = &s->Phone.Data.Priv.ATGEN;

    smprintf(s, "Checking availability of XXXX\n");
    ATGEN_WaitForAutoLen(s, "AT+XXXX=?\r", 0x00, 4, ID_GetProtocol);
    if (error == ERR_NONE) {
        Priv->Foo_XXXX = AT_AVAILABLE;
    } else {
        Priv->Foo_XXXX = AT_NOTAVAILABLE;
    }
    return error;
}
```

(continues on next page)

(continued from previous page)

```
GSM_Reply_Function ATGENReplyFunctions[] = {
...
{ATGEN_GenericReply,        "AT+XXXX=?"             ,0x00,0x00,ID_GetProtocol           ␣
↪    },
...
```

Alternatively (if detection is not possible), you can use features and phones database (see `libgammu/gsmphones.c`) or vendor based decision to use some commands.

Invoking AT command

The AT commands are invoked using `GSM_WaitFor()`, or a wrapper `ATGEN_WaitForAutoLen()`, where you don't have to specify length for text commands and automatically sets error variable.

Generally you need to construct buffer and then invoke it. For some simple functions it is pretty straight forward:

```
GSM_Error ATGEN_GetBatteryCharge(GSM_StateMachine *s, GSM_BatteryCharge *bat)
{
    GSM_Error error;

    GSM_ClearBatteryCharge(bat);
    s->Phone.Data.BatteryCharge = bat;
    smprintf(s, "Getting battery charge\n");
    ATGEN_WaitForAutoLen(s, "AT+CBC\r", 0x00, 4, ID_GetBatteryCharge);
    return error;
}
```

As you can see, it is often required to store pointer to data store somewhere, for most data types `s->Phone.Data` does contain the pointer to do that.

Parsing reply

For parsing reply, you should use `ATGEN_ParseReply()`, which should be able to handle all encoding and parsing magic. You can grab lines from the reply using `GetLineString()`.

The reply function needs to be hooked to the reply functions array, so that it is invoked when reply is received from the phone.

Continuing in above example for getting battery status, the (simplified) function would look like:

```
GSM_Error ATGEN_ReplyGetBatteryCharge(GSM_Protocol_Message *msg, GSM_StateMachine *s)
{
    GSM_Error error;
    GSM_Phone_ATGENData *Priv = &s->Phone.Data.Priv.ATGEN;
    GSM_BatteryCharge *BatteryCharge = s->Phone.Data.BatteryCharge;
    int bcs = 0, bcl = 0;

    switch (s->Phone.Data.Priv.ATGEN.ReplyState) {
        case AT_Reply_OK:
            smprintf(s, "Battery level received\n");
            error = ATGEN_ParseReply(s,
                GetLineString(msg->Buffer, &Priv->Lines, 2),
```

(continues on next page)

(continued from previous page)

```
                "+CBC: @i, @i",
                &bcs,
                &bcl);

            BatteryCharge->BatteryPercent = bcl;

            switch (bcs) {
                case 0:
                    BatteryCharge->ChargeState = GSM_BatteryPowered;
                    break;
                case 1:
                    BatteryCharge->ChargeState = GSM_BatteryConnected;
                    break;
                case 2:
                    BatteryCharge->ChargeState = GSM_BatteryCharging;
                    break;
                default:
                    BatteryCharge->ChargeState = 0;
                    smprintf(s, "WARNING: Unknown battery state: %d\n", bcs);
                    break;
            }
            return ERR_NONE;
        case AT_Reply_Error:
            smprintf(s, "Can't get battery level\n");
            return ERR_NOTSUPPORTED;
        case AT_Reply_CMSError:
            smprintf(s, "Can't get battery level\n");
            return ATGEN_HandleCMSError(s);
        case AT_Reply_CMEError:
            return ATGEN_HandleCMEError(s);
        default:
            return ERR_UNKNOWNRESPONSE;
    }
}


GSM_Reply_Function ATGENReplyFunctions[] = {
...
{ATGEN_ReplyGetBatteryCharge,       "AT+CBC"                ,0x00,0x00,ID_
↪GetBatteryCharge   },
...
```

As you can see, all reply function first need to handle which error code did they receive and return appropriate error if needed. Functions `ATGEN_HandleCMSError()` and `ATGEN_HandleCMEError()` simplify this, but you might need to customize it by handling some error codes manually (eg. when phone returns error on empty location).

The rest of the function is just call to `ATGEN_ParseReply()` and processing parsed data.

CHAPTER

SEVEN

FILE FORMATS USED BY GAMMU

Gammu understands wide range of standard formats as well as introduces own formats for storing some data.

7.1 INI file format

The INI file format is widely used in Gammu, for both configuration (see *Gammu Configuration File*) and storing data (see *Backup Format* and *SMS Backup Format*).

This file use ini file syntax, with comment parts being marked with both `;` and `#`. Sections of config file are identified in square brackets line `[this]`. All key values are case insensitive.

7.1.1 Examples

You most likely know INI files from other programs, however to illustrate, here is some example:

```
; comment

[section]
key = value

[another section]

key =  longer value


# another comment
```

7.2 SMS Backup Format

The SMS backup format is text file encoded in current encoding of platform where Gammu is running.

This file use ini file syntax, see *INI file format*.

7.2.1 Sections

The file consists of sections, whose name starts with `SMSBackup`. When creating the backup file, three digits are appended to this text defining order. While reading the backup, any part after `SMSBackup` text is ignored and everything which begins with this is processed. So you can as well give the section name `SMSBackupFoo` and it will be processed.

The number of messages in backup file is currently limited by *GSM_BACKUP_MAX_SMS* (100000 at time of writing this document).

`SMSBackup` section

Each section interprets one physical SMS message (eg. one message part in case of multipart messages).

Decoded text

For SMS backups created by Gammu, there is a decoded text as a comment just after the section name:

```
[SMSBackup001]
; This is message text
```

The text can be split to more lines if it is too long or of original message included new lines.

Note: This is easiest way to get message text, however also the least reliable one, because it is stored in the comments in the file.

Variables

The following variables can be defined for each SMS:

`SMSC`
Text representation of SMSC number, not used by Gammu if `SMSCUnicode` exists.

`SMSCUnicode`
Hex encoded UCS-2 string with SMSC number.

`Class`
Message class.

`Sent`
Timestamp, when message has been sent.

`PDU`
Message type, one of:

- `Deliver` - received message
- `Submit` - message to send
- `Status_Report` - message to send with delivery report

`DateTime`
Timestamp of message (sent or received).

`RejectDuplicates`
Whether receiver should reject duplicates.

ReplaceMessage
ID of message to replace.

MessageReference
Message reference number as generated by network.

State
State of the message:

- `Read`
- `UnRead`
- `Sent`
- `UnSent`

Number
Recipient number.

Name
Name of the message.

Length
Length of message text.

Coding
Coding of the message:

- `8bit` - binary message
- `Default` - GSM encoding, up to 160 chars in message
- `Unicode` - Unicode encoding, up to 70 chars in message

Text00 ... TextNN
Numbered parts of the message payload.

Folder
ID of folder where the message was saved.

UDH
User defined header of the message.

7.2.2 Example

The backup of message can look like following:

```
[SMSBackup000]
#ABCDEFGHIJKLMNOPQRSTUVWXYZ
#
SMSC = "+4540590000"
SMSCUnicode = 002B00340035003400300035003900300030003000300030
Sent = 20021201T025023
State = UnRead
Number = "+4522706947"
NumberUnicode = 002B00340035003200320037003000360039003400 37
Name = ""
NameUnicode =
Text00 =␣
```

(continues on next page)

(continued from previous page)

```
↪00410042004300440045004600470048004900 4A004B004C004D004E004F00500051005200530054005500560057005800590
Coding = Default
Folder = 1
Length = 27
Class = -1
ReplySMSC = False
RejectDuplicates = True
ReplaceMessage = 0
MessageReference = 0
```

7.3 Backup Format

The backup format is text file encoded in either ASCII or UCS-2-BE encodings.

This file use ini file syntax, see *INI file format*.

7.3.1 Examples

If you will backup settings to Gammu text file, it will be possible to edit it. It's easy: many things in this file will be written double - once in Unicode, once in ASCII. When you will remove Unicode version Gammu will use ASCII on **restore** (and you can easy edit ASCII text) and will convert it according to your OS locale. When will be available Unicode version of text, it will be used instead of ASCII (useful with Unicode phones - it isn't important, what locale is set in computer and no conversion Unicode -> ASCII and ASCII -> Unicode is done).

You can use any editor with regular expressions function to edit backup text file. Examples of such editors can be vim or TextPad which both do support regular expressions.

Remove info about voice tags

Find:

```
^Entry\([0-9][0-9]\)VoiceTag = \(.*\)\n
```

Replace:

```
<blank>
```

Change all numbers starting from +3620, +3630, +3660, +3670 to +3620

Find:

```
Type = NumberGeneral\nEntry\([0-9][0-9]\)Text = "\+36\(20\|30\|60\|70\)\n
```

Replace:

```
Type = NumberMobile\nEntry\1Text = "\+3620
```

Change phone numbers type to mobile for numbers starting from +3620, +3630,... and removing the corresponding TextUnicode line

Find:

```
Type = NumberGeneral\nEntry\([0-9][0-9]\)Text = "\+36\([2367]0\)\([^\"]*\)"\nEntry\([0-
↪9][0-9]\)TextUnicode = \([^\n]*\)\n
```

Replace:

```
Type = NumberMobile\nEntry\1Text = "\+36\2\3"\n
```

See also:

Converting file formats

CHAPTER

EIGHT

GAMMU CONFIGURATION FILE

8.1 Synopsis

On Linux, MacOS X, BSD and other Unix-like systems, the config file is searched in following order:

1. `$XDG_CONFIG_HOME/gammu/config`
2. `~/.config/gammu/config`
3. `~/.gammurc`
4. `/etc/gammurc`

On Microsoft Windows:

1. `%PROFILE%\Application Data\gammurc`
2. `.\gammurc`

8.2 Description

Gammu requires configuration to be able to properly talk to your phone. *Gammu Utility* reads configuration from a config file. It's location is determined on runtime, see above for search paths.

You can use *gammu-config* or *gammu-detect* to generate configuration file or start from *Fully documented example*.

For hints about configuring your phone, you can check Gammu Phone Database <https://wammu.eu/phones/> to see what user users experienced.

This file use ini file syntax, see *INI file format*.

Configuration file for gammu can contain several sections - `[gammu]`, `[gammu1]`, `[gammuN]`, ... Each section configures one connection setup and in default mode gammu tries all of them in numerical order. You can also specify which configuration section to use by giving it's number (*`[gammu]`* has number 0) as a parameter to *Gammu Utility* and it will then use only this section.

`[gammu]`

This section is read by default unless you specify other on command line.

8.2.1 Device connection parameters

Connection

Protocol which will be used to talk to your phone.

For Nokia cables you want to use one of following:

fbus
serial FBUS connection

dlr3
DLR-3 and compatible cables

dku2
DKU-2 and compatible cables

dku5
DKU-5 and compatible cables

mbus
serial MBUS connection

If you use some non original cable, you might need to append `-nodtr` (eg. for ARK3116 based cables) or `-nopower`, but Gammu should be able to detect this automatically.

For non-Nokia phones connected using cable you generally want:

at
generic AT commands based connection

You can optionally specify speed of the connection, eg. `at19200`, but it is not needed for modern USB cables.

For IrDA connections use one of following:

irdaphonet
Phonet connection for Nokia phones.

irdaat
AT commands connection for most of phones (this is not supported on Linux).

irdaobex
OBEX (IrMC or file transfer) connection for most of phones.

irdagnapbus
GNapplet based connection for Symbian phones, see *Gnapplet Protocol*.

For Bluetooth connection use one of following:

bluephonet
Phonet connection for Nokia phones.

bluefbus
FBUS connection for Nokia phones.

blueat
AT commands connection for most of phones.

blueobex
OBEX (IrMC or file transfer) connection for most of phones.

bluerfgnapbus
GNapplet based connection for Symbian phones, see *Gnapplet Protocol*.

blues60
Connection to Series60 applet in S60 phones, see *Series60 Remote Protocol*.

New in version 1.29.90.

New in version 1.36.7: Gammu now supports connecting using proxy command.

You can also proxy the connection using shell command, for example to different host. This can be done using proxy connections:

proxyphonet
Phonet connection for Nokia phones.

proxyfbus
FBUS connection for Nokia phones.

proxyat
AT commands connection for most of phones.

proxyobex
OBEX (IrMC or file transfer) connection for most of phones.

proxygnapbus
GNapplet based connection for Symbian phones, see *Gnapplet Protocol*.

proxys60
Connection to Series60 applet in S60 phones, see *Series60 Remote Protocol*.

See also:

Configuring Gammu FAQ

Device

New in version 1.27.95.

Device node or address of phone. It depends on used connection.

For **cables** or emulated serial ports, you enter device name (for example `/dev/ttyS0`, `/dev/ttyACM0`, `/dev/ircomm0`, `/dev/rfcomm0` on Linux, `/dev/cuad0` on FreeBSD or `COM1:` on Windows). The special exception are DKU-2 and DKU-5 cables on Windows, where the device is automatically detected from driver information and this parameters is ignored.

Note: Some USB modems expose several interfaces, in such cases Gammu works best with "User" one, you can find more information on <http://www.dd-wrt.com/wiki/index.php/Mobile_Broadband>.

For **USB** connections (currently only fbususb and dku2 on Linux), you can specify to which USB device Gammu should connect. You can either provide vendor/product IDs or device address on USB:

```
Device = 0x1234:0x5678  # Match device by vendor and product id
Device = 0x1234:-1      # Match device by vendor id
Device = 1.10           # Match device by usb bus and device address
Device = 10             # Match device by usb device address
Device = serial:123456  # Match device by serial string
```

Note: On Linux systems, you might lack permissions for some device nodes. You might need to be member of some group (eg. `plugdev` or `dialout`) or or add special udev rules to enable you access these devices as non-root.

For Nokia phones you can put following file (also available in sources as `contrib/udev/69-gammu-acl.rules`) as `/etc/udev/rules.d/69-gammu-acl.rules`:

```
#
# udev rule to give users access to USB device to be used by Gammu
#

ACTION!="add|change", GOTO="gammu_acl_rules_end"

KERNEL!="ttyACM[0-9]*", GOTO="gammu_acl_rules_end"
SUBSYSTEM!="tty", GOTO="gammu_acl_rules_end"

# Nokia devices
ATTRS{manufacturer}=="Nokia", TAG+="uaccess"

# Example for Sony Ericsson J108i Cedar
# ATTRS{idVendor}=="0fce", ATTRS{idProduct}=="d14e", TAG+="uaccess"

LABEL="gammu_acl_rules_end"
```

In case your USB device appears as the serial port in the system (eg. `/dev/ttyACM0` on Linux or `COM5:` on Windows), just use same setup as with serial port.

For **Bluetooth** connection you have to enter Bluetooth address of your phone (you can list Bluetooth devices in range on Linux using **`hcitool scan`** command). Optionally you can also force Gammu to use specified channel by including channel number after slash.

Before using Gammu, your device should be paired with computer or you should have set up automatic pairing.

For **Proxy** connections, you need to specify command which should be executed. It is supposed to pass bidirectional communication from Gammu to the device. This can happen for example over network.

For **IrDA** connections, this parameters is not used at all.

If IrDA does not work on Linux, you might need to bring up the interface and enable discovery (you need to run these commands as root):

```
ip l s dev irda0 up          # Enables irda0 device
sysctl net.irda.discovery=1  # Enables device discovery on IrDA
```

Note: Native IrDA is not supported on Linux, you need to setup virtual serial port for it (eg. `/dev/ircomm0`) and use it same way as cable. This can be usually achieved by loading modules `ircomm-tty` and `irtty-sir`:

```
modprobe ircomm-tty
modprobe irtty-sir
```

See also:

Configuring Gammu FAQ

`Port`

Deprecated since version 1.27.95: Please use *`Device`* instead.

Alias for *`Device`*, kept for backward compatibility.

Model

Do not use this parameter unless really needed! The only use case for this is when Gammu does not know your phone and misdetects it's features.

The only special case for using model is to force special type of OBEX connection instead of letting Gammu try the best suited for selected operation:

obexfs
force using of file browsing service (file system support)

obexirmc
force using of IrMC service (contacts, calendar and notes support)

obexnone
none service chosen, this has only limited use for sending file (`gammu sendfile` command)

mobex
m-obex service for Samsung phones

Use_Locking

On Posix systems, you might want to lock serial device when it is being used using UUCP-style lock files. Enabling this option (setting to yes) will make Gammu honor these locks and create it on startup. On most distributions you need additional privileges to use locking (eg. you need to be member of uucp group).

This option has no meaning on Windows.

8.2.2 Connection options

SynchronizeTime

If you want to set time from computer to phone during starting connection.

StartInfo

This option allows one to set, that you want (setting `yes`) to see message on the phone screen or phone should enable light for a moment during starting connection. Phone will not beep during starting connection with this option. This works only with some Nokia phones.

8.2.3 Debugging options

LogFile

Path to file where information about communication will be stored.

Note: For most debug levels (excluding `errors`) the log file is overwritten on each execution.

LogFormat

Determines what all will be logged to *LogFile*. Possible values are:

nothing
no debug level

text
transmission dump in text format

textall
all possible info in text format

textalldate
all possible info in text format, with time stamp

errors
errors in text format

errorsdate
errors in text format, with time stamp

binary
transmission dump in binary format

For debugging use either `textalldate` or `textall`, it contains all needed information to diagnose problems.

Features

Custom features for phone. This can be used as override when values coded in `common/gsmphones.c` are bad or missing. Consult `include/gammu-info.h` for possible values (all *GSM_Feature* values without leading `F_` prefix). Please report correct values to Gammu authors.

8.2.4 Locales and character set options

GammuCoding

Forces using specified codepage (for example `1250` will force CP-1250 or `utf8` for UTF-8). This should not be needed, Gammu detects it according to your locales.

GammuLoc

Path to directory with localisation files (the directory should contain `LANG/LC_MESSAGES/gammu.mo`). If gammu is properly installed it should find these files automatically.

8.2.5 Advanced options

Advanced options are used to alter default logic, when using these options the user is responsible for ensuring any settings are correct for the target device and that they produce the desired behaviour.

atgen_setCNMI

For configurations using the generic AT command protocol it is possible to override the default indicators used when a new SMS message is received.

The value for the setting is a comma delimited list of single digits corresponding to the values for the `AT+CNMI` modem command. If a digit is not provided, or if the provided digit is outside of the acceptable range for the device the default value is used.

For example setting `atgen_setcnmi = ,,2` would set the third parameter of the CNMI command to the value 2, leaving the rest of the parameters at default, and `atgen_setcnmi = 1,,,1` would set the first and fourth parameters respectively.

8.2.6 Other options

DataPath

Additional path where to search for data files. The default path is configured on build time (and defaults to `/usr/share/data/gammu` on Unix systems). Currently it is used only for searching files to upload to phone using `gammu install`.

8.3 Examples

There is more complete example available in Gammu documentation, see *Gammu Utility*.

8.3.1 Connection examples

Gammu configuration for Nokia phone using DLR-3 cable:

```
[gammu]
device = /dev/ttyACM0
connection = dlr3
```

Gammu configuration for Sony-Ericsson phone (or any other AT compatible phone) connected using USB cable:

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

Gammu configuration for Sony-Ericsson (or any other AT compatible phone) connected using bluetooth:

```
[gammu]
device = B0:0B:00:00:FA:CE
connection = blueat
```

Gammu configuration for phone which needs to manually adjust Bluetooth channel to use channel 42:

```
[gammu]
device = B0:0B:00:00:FA:CE/42
connection = blueat
```

8.3.2 Working with multiple phones

Gammu can be configured for multiple phones (however only one connection is used at one time, you can choose which one to use with `gammu -s` parameter). Configuration for phones on three serial ports would look like following:

```
[gammu]
device = /dev/ttyS0
connection = at

[gammmu1]
device = /dev/ttyS1
connection = at

```

(continues on next page)

(continued from previous page)

```
[gammmu2]
device = /dev/ttyS2
connection = at
```

8.3.3 Connecting to remote phone

New in version 1.36.7.

You can connect using Gammu to phone running on different host. This can be achieved using proxy connection, which executes command to forward bi-directional communication with the phone.

```
[gammu]
device = ssh root@my.router /usr/local/bin/myscript /dev/ttyUSB0
connection = proxyat
```

You can find sample script which can be used on the remote side in `contrib/proxy/gammu-backend`.

8.3.4 Fully documented example

You can find this sample file as `docs/config/gammurc` in Gammu sources.

```
; This is a sample ~/.gammurc file.
; In Unix/Linux  copy it into your home directory and name it .gammurc
;                or into /etc and name it gammurc
; In Win32       copy it into directory with Gammu.exe and name gammurc
; More about parameters later
; Anything behind ; or # is comment.
; --------------------------------------------------------------------------

[gammu]

device = com8:
connection = irdaphonet
; Do not use model configuration unless you really need it
;model = 6110
;synchronizetime = yes
;logfile = gammulog
;logformat = textall
;use_locking = yes
;gammuloc = locfile
;startinfo = yes
;gammucoding = utf8
;usephonedb = yes

[gammu1]

device = com8:
;model = 6110
connection = fbusblue
;synchronizetime = yes
;logfile = gammulog
```

(continues on next page)

(continued from previous page)

```
;logformat = textall
;use_locking = yes
;gammuloc = locfile
;startinfo = yes
;gammucoding = utf8

; Step 1. Please find required Connection parameter and look into assigned
; with it device type. With some Connection you must set concrete model

; ================================================================= cables =====
; New Nokia protocol for FBUS/DAU9P
;    Connection "fbus", device type serial
; New Nokia protocol for DLR3/DLR3P
;    Connection "fbusdlr3"/"dlr3", device type serial
; New Nokia protocol for DKU2 (and phone with USB converter on phone mainboard
;                              like 6230)
;    Connection "dku2phonet"/"dku2", device type dku2 on Windows
;    Connection "fbususb" on Linux
; New Nokia protocol for DKU5 (and phone without USB converter on phone
;                              mainboard like 5100)
;    Connection "dku5fbus"/"dku5", device type dku5
; New Nokia protocol for PL2303 USB cable (and phone without USB converter
;                                          on phone mainboard like 5100)
;    Connection "fbuspl2303", device type usb
; Old Nokia protocol for MBUS/DAU9P
;    Connection "mbus", device type serial
; Variants:
; You can modify a bit behaviour of connection using additional flags
; specified just after connection name like connection-variant.
; If you're using ARK3116 cable (or any other which does not like dtr
; handling), you might need -nodtr variant of connection, eg. dlr3-nodtr.
; If cable you use is not powered over DTR/RTS, try using -nopower variant of
; connection, eg. fbus-nopower.
; ------------------------------------------------------------------------------
; AT commands for DLR3, DKU5 or other AT compatible cable (8 bits, None
; parity, no flow control, 1 stop bit). Used with Nokia, Alcatel, Siemens, etc.
;    Connection "at19200"/"at115200"/.., device type serial
; AT commands for DKU2 cable
;    Connection "dku2at", device type dku2
; ============================================================== infrared =====
; Nokia protocol for infrared with Nokia 6110/6130/6150
;    Connection "fbusirda"/"infrared", device type serial
; Nokia protocol for infrared with other Nokia models
;    Connection "irdaphonet"/"irda", device type irda
; ------------------------------------------------------------------------------
; AT commands for infrared. Used with Nokia, Alcatel, Siemens, etc.
;    Connection "irdaat", device type irda
; ------------------------------------------------------------------------------
; OBEX for infrared
;    Connection "irdaobex", device type irda.
; ============================================================== Bluetooth =====
; Nokia protocol with serial device set in BT stack (WidComm, other) from
```

(continues on next page)

(continued from previous page)

```
; adequate service and Nokia 6210
;    Connection "fbusblue", device type serial
; Nokia protocol with serial device set in BT stack (WidComm, other) from
; adequate service and other Nokia models
;    Connection "phonetblue", device type serial
; ------------------------------------------------------------------------
; Nokia protocol for Bluetooth stack with Nokia 6210
;    Connection "bluerffbus", device type BT
; Nokia protocol for Bluetooth stack with DCT4 Nokia models, which don't inform
; about services correctly (6310, 6310i with firmware lower than 5.50, 8910,..)
;    Connection "bluerfphonet", device type BT
; Nokia protocol for Bluetooth stack with other DCT4 Nokia models
;    Connection "bluephonet", device type BT
; ------------------------------------------------------------------------
; AT commands for Bluetooth stack and 6210 / DCT4 Nokia models, which don't
; inform about BT services correctly (6310, 6310i with firmware lower
; than 5.50, 8910,..)
;    Connection "bluerfat", device type BT
; AT commands for Bluetooth stack with other phones (Siemens, other Nokia,etc.)
;    Connection "blueat", device type BT
; ------------------------------------------------------------------------
; OBEX for Bluetooth stack with DCT4 Nokia models, which don't inform about
; BT services correctly (6310, 6310i with firmware lower than 5.50, 8910,...)
;    Connection "bluerfobex", device type BT
; OBEX for Bluetooth stack with other phones (Siemens, other Nokia, etc.)
;    Connection "blueobex", device type BT.
; ------------------------------------------------------------------------
;    Connection "bluerfgnapbus", device type BT, model "gnap"
;    Connection "irdagnapbus", device type irda, model "gnap"

; Step2. According to device type from Step1 and used OS set Port parameter

; ------------------------------------------------------------------------
; Port type | "Port" parameter in Windows/DOS | "Port" parameter in Linux/Unix
; ----------|---------------------------------|--------------------------------
; serial    | "com*:"                         | "/dev/ttyS*"
;           | (example "com1:")               | (example "/dev/ttyS1")
;           |                                 | or "/dev/tts/**" (with DevFS)
;           |                                 | virtual serial ports like
;           |                                 | "/dev/ircomm*" or "/dev/rfcomm*"
; ----------|---------------------------------|--------------------------------
; irda      | ignored (can be empty)          | ignored (can be empty)
; ----------|---------------------------------|--------------------------------
; BT        | Bluetooth device address (example "00:11:22:33:44:55").
;           | Optionally you can also include channel after slash
;           | (example "00:11:22:33:44:55/12"). Can be also empty.
; ----------|---------------------------------|--------------------------------
; dku2      | ignored (can be empty)          | /dev/ttyUSB* or /dev/ttyACM*
; ----------|---------------------------------|--------------------------------
; dku5      | ignored (can be empty)          | connection with it not possible
; ----------|---------------------------------|--------------------------------
; usb       | connection with it not possible | "/dev/ttyUSB*"
```

(continues on next page)

(continued from previous page)

```
; Step3. Set other config parameters

; -------------------------------------------------------------------------
; Parameter name  | Description
; ----------------|--------------------------------------------------------
; Model           | Should not be used unless you have a good reason to do so.
;                 | If Gammu doesn't recognize your phone model, put it here.
;                 | Example values: "6110", "6150", "6210", "8210"
; SynchronizeTime | if you want to set time from computer to phone during
;                 | starting connection. Do not rather use this option when
;                 | when to reset phone during connection (in some phones need
;                 | to set time again after restart)
; GammuLoc        | name of localisation file
; StartInfo       | this option allows one to set, that you want (setting "yes")
;                 | to see message on the phone screen or phone should enable
;                 | light for a moment during starting connection. Phone
;                 | WON'T beep during starting connection with this option.
; GammuCoding     | forces using specified codepage (in win32 - for example
;                 | "1250" will force CP1250) or UTF8 (in Linux - "utf8")
; ----------------|--------------------------------------------------------
; Logfile         | Use, when want to have logfile from communication.
; Logformat       | What debug info and format should be used:
;                 |   "nothing" - no debug level (default)
;                 |   "text"    - transmission dump in text format
;                 |   "textall" - all possible info in text format
;                 |   "errors"  - errors in text format
;                 |   "binary"  - transmission dump in binary format
; ----------------|--------------------------------------------------------
; Features        | Custom features for phone. This can be used as override
;                 | when values coded in common/gsmphones.c are bad or
;                 | missing. Consult include/gammu-info.h for possible values
;                 | (all Feature values without leading F_ prefix).
;                 | Please report correct values to Gammu authors.
; ----------------|--------------------------------------------------------
; Use_Locking     | under Unix/Linux use "yes", if want to lock used device
;                 | to prevent using it by other applications. In win32 ignored

; vim: et ts=4 sw=4 sts=4 tw=78 spell spelllang=en_us
```

CHAPTER

NINE

GAMMU UTILITY

9.1 Synopsis

```
gammu [parameters] <command> [options]
```

Commands actually indicate which operation should Gammu perform. They can be specified with or without a leading `--`.

9.2 Description

This program is a tool for mobile phones. Many vendors and phones are supported, for actual listing see Gammu Phones Database.

9.2.1 Options

Parameters before command configure gammu behaviour:

-c, --config <filename>
: name of configuration file (see *Gammu Configuration File*)

-s, --section <confign>
: section of config file to use, eg. 42

-d, --debug <level>
: debug level (see *LogFormat* in *Gammu Configuration File* for possible values)

-f, --debug-file <filename>
: file for logging debug messages

9.2.2 Phone information commands

battery
: Displays information about battery and power source.

getdisplaystatus

getsecuritystatus
: Show, if phone wait for security code (like PIN, PUK, etc.) or not

identify

Show the most important phone data.

monitor [times]

Retrieves phone status and writes it continuously to standard output. Press Ctrl+C to interrupt this command.

If no parameter is given, the program runs until interrupted, otherwise only given number of iterations is performed.

This command outputs almost all information Gammu supports:

- Number of contacts, calendar and todo entries, messages, calls, etc.
- Signal strength.
- Battery state.
- Currently used network.
- Notifications of incoming messages and calls.

9.2.3 Call commands

answercall [id]

Answer incoming call.

cancelcall [id]

Cancel incoming call

canceldiverts

Cancel all existing call diverts.

conferencecall id

Initiates a conference call.

dialvoice number [show|hide]

Make voice call from SIM card line set in phone.

show|hide - optional parameter whether to disable call number indication.

divert get|set all|busy|noans|outofreach all|voice|fax|data [number timeout]

Manage or display call diverts.

get or set
whether to get divert information or to set it.

all or busy or noans or outofreach
condition when apply divert

all or voice or fax or data
call type when apply divert

number
number where to divert

timeout
timeout when the diversion will happen

getussd code

Retrieves USSD information - dials a service number and reads response.

holdcall `id`
: Holds call.

maketerminatedcall `number length [show|hide]`
: Make voice call from SIM card line set in phone which will be terminated after `length` seconds.

senddtmf `sequence`
: Plays DTMF sequence. In some phones available only during calls

splitcall `id`
: Splits call.

switchcall `[id]`
: Switches call.

transfercall `[id]`
: Transfers call.

unholdcall `id`
: Unholds call.

9.2.4 SMS and EMS commands

Sending messages might look a bit complicated on first attempt to use. But be patient, the command line has been written in order to allow almost every usage. See EXAMPLE section for some hints on usage.

There is also an option to use *gammu-smsd* when you want to send or receive more messages and process them automatically.

Introduction to SMS formats

Gammu has support for many SMS formats like:

Nokia Smart Messaging
: used for monochromatic picture images, downloadable profiles, monochromatic operator logos, monochromatic caller logos and monophonic ringtones

Linked SMS
: both with 8 and 16-bit identification numbers in headers

EMS
: this is SMS format used for saving monochromatic images, monophonic ringtones, animations, text formatting and others

MMS notifications
: contains links where phone should download MMS

Alcatel logo messages
: proprietary format for logos

You need to ensure that the target phone supports message type you want to send. Otherwise the phone will not be able to display it or will even crash, because firmware of phone did not expect this possibility.

Encoding chars in SMS text

Text in SMS can be coded using two ways:

GSM Default Alphabet

With *GSM Default Alphabet* you can fit at most 160 chars into single SMS (Gammu doesn't support compressing such texts according to GSM standards, but it isn't big limit, because there are no phones supporting them), but they're from limited set:

- all Latin small and large
- all digits
- some Greek
- some other national
- some symbols like @ ! " # & / ( ) % * + = - , . : ; < > ?
- few others

Unicode

With *Unicode* single SMS can contain at most 70 chars, but these can be any chars including all national and special ones.

Warning: Please note, that some older phones might have problems displaying such message.

Conversion

Gammu tries to do the best to handle non ASCII characters in your message. Everything is internally handled in Unicode (the input is converted depending on your locales configuration) and in case message uses Unicode the text will be given as such to the message.

Should the message be sent in GSM Default Alphabet, Gammu will try to convert all characters to keep message readable. Gammu does support multi byte encoding for some characters in GSM Default Alphabet (it is needed for `^ { } \ [ ] ~ |`). The characters which are not present in GSM Default Alphabet are transliterated to closest ASCII equivalent (accents are removed). Remaining not known characters are replaced by question mark.

SMS commands

addsmsfolder `name`

deleteallsms `folder`

Delete all SMS from specified SMS folder.

deletesms `folder start [stop]`

Delete SMS from phone. See description for `gammu getsms` for info about sms folders naming convention.

Locations are numerated from 1.

displaysms ... (options like in sendsms)

Displays PDU data of encoded SMS messages. It accepts same parameters and behaves same like sendsms.

getallsms -pbk

Get all SMS from phone. In some phones you will have also SMS templates and info about locations used to save Picture Images. With each sms you will see location. If you want to get such sms from phone alone, use `gammu getsms`.

geteachsms -pbk

Similarly to `gammu getallsms`. Difference is, that links all concatenated sms

getsms folder start [stop]

Get SMS.

Locations are numerated from 1.

Folder 0 means that sms is being read from "flat" memory (all sms from all folders have unique numbers). It's sometimes emulated by Gammu. You can use it with all phones.

Other folders like 1, 2, etc. match folders in phone such as Inbox, Outbox, etc. and each sms has unique number in his folder. Name of folders can depend on your phone (the most often 1="Inbox", 2="Outbox", etc.). This method is not supported by all phones (for example, not supported by Nokia 3310, 5110, 6110). If work with your phone, use `gammu getsmsfolders` to get folders list.

getsmsc [start [stop]]

Get SMSC settings from SIM card.

Locations are numerated from 1.

getsmsfolders

Get names for SMS folders in phone

savesms TYPE [type parameters] [type options] [-folder id] [-unread] [-read] [-unsent] [-sent] [-sender

Saves SMS to phone, see below for TYPE options.

-smscset number

SMSC number will be taken from phone stored SMSC configuration number.

Default: 1

-smscnumber number

SMSC number

-reply

reply SMSC is set

-folder number

save to specified folder.

Folders are numerated from 1.

The most often folder 1 = "Inbox", 2 = "Outbox",etc. Use `gammu getsmsfolders` to get folder list.

-unread

makes message unread. In some phones (like 6210) you won't see unread sms envelope after saving such sms. In some phones with internal SMS memory (like 6210) after using it with folder 1 SIM SMS memory will be used

-read

makes message read. In some phones with internal SMS memory (like 6210) after using it with folder 1 SIM SMS memory will be used

-unsent

makes message unsent

-sent

makes message sent

-smsname name

set message name

-sender number

set sender number (default: Gammu)

-maxsms num

Limit maximum number of messages which will be created. If there are more messages, Gammu will terminate with failure.

Types of messages:

ANIMATION frames file1 file2...

Save an animation as a SMS. You need to give number of frames and picture for each frame. Each picture can be in any picture format which Gammu supports (B/W bmp, gif, wbmp, nol, nlm...).

BOOKMARK file location

Read WAP bookmark from file created by gammu backup command and saves in Nokia format as SMS

CALENDAR file location

Read calendar note from file created by gammu backup command and saves in VCALENDAR 1.0 format as SMS. The location identifies position of calendar item to be read in backup file (usually 1, but can be useful in case the backup contains more items).

CALLER file

Save caller logo as sms in Nokia (Smart Messaging) format - size 72x14, two colors.

> **Warning:** Please note, that it isn't designed for colour logos available for example in DCT4/TIKU - you need to put bitmap file there inside phone using filesystem commands.

USSD

Send USSD query instead of SMS.

New in version 1.38.5.

EMS [-unicode] [-16bit] [-format lcrasbiut] [-text text] [-unicodefiletext file] [-defsound ID] [-defsound ID ..] [-protected number]

Saves EMS sequence. All format specific parameters (like -defsound) can be used few times.

-text

adds text

-unicodefiletext

adds text from Unicode file

-defanimation

adds default animation with ID specified by user. ID for different phones are different.

-animation

adds "frames" frames read from file1, file2, etc.

-defsound

adds default sound with ID specified by user. ID for different phones are different.

-tone10

adds IMelody version 1.0 read from RTTL or other compatible file

-tone10long

IMelody version 1.0 saved in one of few SMS with UPI. Phones compatible with UPI (like Sony-Ericsson phones) will read such ringtone as one

-tone12

adds IMelody version 1.2 read from RTTL or other compatible file

-tone12long

IMelody version 1.2 saved in one of few SMS with UPI. Phones compatible with UPI (like Sony-Ericsson phones) will read such ringtone as one

-toneSE

adds IMelody in "short" form supported by Sony-Ericsson phones

-toneSElong

add Sony-Ericsson IMelody saved in one or few SMS with UPI

-variablebitmap

bitmap in any size saved in one SMS

-variablebitmaplong

bitmap with maximum size 96x128 saved in one or few sms

-fixedbitmap

bitmap 16x16 or 32x32

-protected

all ringtones and bitmaps after this parameter (excluding default ringtones and logos) will be "protected" (in phones compatible with ODI like SonyEricsson products it won't be possible to forward them from phone menu)

-16bit

Gammu uses SMS headers with 16-bit numbers for saving linking info in SMS (it means less chars available for user in each SMS)

-format `lcrasbiut`

last text will be formatted. You can use combinations of chars:

Character	Formatting
l	left aligned
c	centered
r	right aligned
a	large font
s	small font
b	bold font
i	italic font
u	underlined font
t	strikethrough font

MMSINDICATOR URL Title Sender

Creates a MMS indication SMS. It contains URL where the actual MMS payload is stored which needs to be SMIL encoded. The phone usually downloads the MMS data using special APN, which does not count to transmitted data, however there might be limitations which URLs can be accessed.

MMSSETTINGS file location

Saves a message with MMS configuration. The configuration will be read from Gammu backup file from given location.

OPERATOR file [-netcode netcode] [-biglogo]

Save operator logo as sms in Nokia (Smart Messaging) format - size 72x14 in two colors.

-biglogo

Use 78x21 formatted logo instead of standard 72x14.

Note: This isn't designed for colour logos available for example in newer phones - you need to put bitmap file there inside phone using filesystem commands.

PICTURE file [-text text] [-unicode] [-alcatelbmmi]

Read bitmap from 2 colors file (bmp, nlm, nsl, ngg, nol, wbmp, etc.), format into bitmap in Smart Messaging (72x28, 2 colors, called often Picture Image and saved with text) or Alcatel format and send/save over SMS.

PROFILE [-name name] [-bitmap bitmap] [-ringtone ringtone]

Read ringtone (RTTL) format, bitmap (Picture Image size) and name, format into Smart Messaging profile and send/save as SMS.

Warning: Please note, that this format is abandoned by Nokia and supported by some (older) devices only like Nokia 3310.

RINGTONE file [-long] [-scale]

Read RTTL ringtone from file and save as SMS into SIM/phone memory. Ringtone is saved in Nokia (Smart Messaging) format.

-long

ringtone is saved using Profile style. It can be longer (and saved in 2 SMS), but decoded only by newer phones (like 33xx)

-scale

ringtone will have Scale info for each note. It will allow one to edit it correctly later in phone composer (for example, in 33xx)

SMSTEMPLATE [-unicode] [-text text] [-unicodefiletext file] [-defsound ID] [-defanimation ID] ..]

Saves a SMS template (for Alcatel phones).

TEXT [-inputunicode] [-16bit] [-flash] [-len len] [-autolen len] [-unicode] [-enablevoice] [-

Take text from stdin (or commandline if -text specified) and save as text SMS into SIM/phone memory.

-flash

Class 0 SMS (should be displayed after receiving on recipients' phone display after receiving without entering Inbox)

-len len

specify, how many chars will be read. When use this option and text will be longer than 1 SMS, will be split into more linked SMS

-autolen len

specify, how many chars will be read. When use this option and text will be longer than 1 SMS, will be split into more linked SMS.Coding type (SMS default alphabet/Unicode) is set according to input text

-enablevoice

sms will set voice mail indicator. Text will be cut to 1 sms.

-disablevoice

sms will not set voice mail indicator. Text will be cut to 1 sms.

-enablefax

sms will set fax indicator. Text will be cut to 1 sms.

-disablefax

sms will not set fax indicator. Text will be cut to 1 sms.

-enableemail

sms will set email indicator. Text will be cut to 1 sms.

-disableemail

sms will not set email indicator. Text will be cut to 1 sms.

-voidsms

many phones after receiving it won't display anything, only beep, vibrate or turn on light. Text will be cut to 1 sms.

-unicode

SMS will be saved in Unicode format

Note: The ~ char in SMS text and -unicode option (Unicode coding required) can cause text of SMS after ~ char blink in some phones (like Nokia 33xx).

-inputunicode

input text is in Unicode.

Note: You can create Unicode file using WordPad in Windows (during saving select "Unicode Text Document" format). In Unix can use for example YUdit or vim.

-text

get text from command line instead of stdin.

-textutf8

get text in UTF-8 from command line instead of stdin.

Note: Gammu detects your locales and uses by default encoding based on this. Use this option only when you know the input will be in UTF-8 in all cases.

-16bit

Gammu uses SMS headers with 16-bit numbers for saving linking info in SMS (it means less chars available for user in each SMS)

-replacemessages `ID`

`ID` can be 1..7. When you will use option and send more single SMS to one recipient with the same ID, each another SMS will replace each previous with the same ID

-replacefile `file`

File with replacement table in unicode (UCS-2), preferably with byte order mark (BOM). It contains pairs of chars, first one is to replace, second is replacement one. The replacement is done after reading text for the message.

For example replacement `1` (0x0061) with `a` (0x0031) would be done by file with following content (hex dump, first two bytes is BOM):

```
ff fe 61 00 31 00
```

TODO `file location`

Saves a message with a todo entry. The content will be read from any backup format which Gammu supports and from given location.

VCARD10|VCARD21 `file SM|ME location [-nokia]`

Read phonebook entry from file created by `gammu backup` command and saves in VCARD 1.0 (only name and default number) or VCARD 2.1 (all entry details with all numbers, text and name) format as SMS. The location identifies position of contact item to be read in backup file (usually 1, but can be useful in case the backup contains more items).

WAPINDICATOR `URL Title`

Saves a SMS with a WAP indication for given URL and title.

WAPSETTINGS `file location DATA|GPRS`

Read WAP settings from file created by `gammu backup` command and saves in Nokia format as SMS

sendsms `TYPE destination [type parameters] [type options] [-smscset number] [-smscnumber number] [-repl`

Sends a message to a `destination` number, most parameters are same as for `gammu savesms`.

-save

will also save message which is being sent

-report

request delivery report for message

-validity HOUR|6HOURS|DAY|3DAYS|WEEK|MAX

sets how long will be the message valid (SMSC will the discard the message after this time if it could not deliver it).

setsmsc location number

Set SMSC settings on SIM card. This keeps all SMSC configuration intact, it just changes the SMSC number.

Locations are numerated from 1.

9.2.5 Memory (phonebooks and calls) commands

Memory types

Gammu recognizes following memory types:

DC

Dialled calls

MC

Missed calls

RC

Received calls

ON

Own numbers

VM

voice mailbox

SM

SIM phonebook

ME

phone internal phonebook

FD

fixed dialling

SL

sent SMS log

Memory commands

deleteallmemory DC|MC|RC|ON|VM|SM|ME|MT|FD|SL

Deletes all entries from specified memory type.

For memory types description see *Memory types*.

deletememory DC|MC|RC|ON|VM|SM|ME|MT|FD|SL start [stop]

Deletes entries in specified range from specified memory type.

For memory types description see *Memory types*.

getallmemory DC|MC|RC|ON|VM|SM|ME|MT|FD|SL

Get all memory locations from phone.

For memory types description see *Memory types*.

getmemory DC|MC|RC|ON|VM|SM|ME|MT|FD|SL start [stop [-nonempty]]

Get memory location from phone.

For memory types description see *Memory types*.

Locations are numerated from 1.

getspeeddial start [stop]

Gets speed dial choices.

searchmemory text

Scans all memory entries for given text. It performs case insensitive substring lookup. You can interrupt searching by pressing Ctrl+C.

9.2.6 Filesystem commands

Gammu allows one to access phones using native protocol (Nokias) or OBEX. Your phone can also support usb storage, which is handled on the operating system level and Gammu does not use that.

addfile folderID name [-type JAR|BMP|PNG|GIF|JPG|MIDI|WBMP|AMR|3GP|NRT] [-readonly] [-protected] [-syst

Add file with specified name to folder with specified folder ID.

-type

File type was required for filesystem 1 in Nokia phones (current filesystem 2 doesn't need this).

-readonly

Sets the read only attribute.

-protected

Sets the protected attribute (file can't be for example forwarded from phone menu).

-system

Sets the system attribute.

-hidden

Sets the hidden attribute (file is hidden from phone menu).

-newtime

After using it date/time of file modification will be set to moment of uploading.

addfolder parentfolderID name

Create a folder in phone with specified name in a folder with specified folder ID.

deletefiles fileID

Delete files with given IDs.

deletefolder name

Delete folder with given ID.

getfilefolder fileID, **fileID**

Retrieve files or all files from folder with given IDs from a phone filesystem.

getfiles fileID, **fileID**

Retrieve files with given IDs from a phone filesystem.

getfilesystem [-flatall|-flat]

Display info about all folders and files in phone memory/memory card. By default there is tree displayed, you can change it:

-flatall

there are displayed full file/folder details like ID (first parameter in line)

-flat

Note: In some phones (like N6230) content of some folders (with more files) can be cut (only part of files will be displayed) for example on infrared connection. This is not Gammu issue, but phone firmware problem.

---

getfilesystemstatus

Display info filesystem status - number of bytes available, used or used by some specific content.

getfolderlisting folderID

Display files and folders available in folder with given folder ID. You can get ID's using getfilesystem -flatall.

Warning: Please note, that in some phones (like N6230) content of some folders (with more files) can be cut (only part of files will be displayed) for example on infrared connection. This is not Gammu issue, but phone firmware problem.

getrootfolders

Display info about drives available in phone/memory card.

sendfile name

Sends file to a phone. It's up to phone to decide where to store this file and how to handle it (for example when you send vCard or vCalendar, most of phones will offer you to import it.

setfileattrib folderID [-system] [-readonly] [-hidden] [-protected]

9.2.7 Logo and pictures commands

These options are mainly (there are few exceptions) for monochromatic logos and images available in older phones. Recognized file formats: xpm (only saving), 2-colors bmp, nlm, nsl, ngg, nol, wbmp, gif (for Samsung).

In new models all bitmaps are saved in filesystem and should go into filesystem section

copybitmap inputfile [outputfile [OPERATOR|PICTURE|STARTUP|CALLER]]

Allow one to convert logos files to another. When give ONLY inputfile, output will be written to stdout using ASCII art. When give output file and format, in some file formats (like NLM) will be set indicator informing about logo type to given.

getbitmap TYPE [type options]

Reads bitmap from phone, following types are supported:

CALLER location [file]

Get caller group logo from phone. Locations 1-5.

DEALER

In some models it's possible to save dealer welcome note - text displayed during enabling phone, which can't be edited from phone menu. Here you can get it.

OPERATOR `[file]`

Get operator logo (picture displayed instead of operator name) from phone.

PICTURE `location [file]`

Get Picture Image from phone.

STARTUP `[file]`

Get static startup logo from phone. Allow one to save it in file.

TEXT

Get startup text from phone.

setbitmap `TYPE [type options]`

Sets bitmap in phone, following types are supported:

CALLER `location [file]`

Set caller logo.

COLOUROPERATOR `[fileID [netcode]]`

Sets color operator logo in phone.

COLOURSTARTUP `[fileID]`

DEALER `text`

Sets welcome message configured by dealer, which usually can not be changed in phone menus.

OPERATOR `[file [netcode]]`

Set operator logo in phone. When won't give file and netcode, operator logo will be removed from phone. When will give only filename, operator logo will be displayed for your current GSM operator. When you give additionally network code, it will be displayed for this operator.

PICTURE `file location [text]`

Sets picture image in phone.

STARTUP `file|1|2|3`

Set startup logo in phone. It can be static (then you will have to give file name) or one of predefined animated (only some phones like Nokia 3310 or 3330 supports it, use location 1, 2 or 3 for these).

TEXT `text`

Sets startup text in phone.

WALLPAPER `fileID`

Sets wallpaper in phone.

9.2.8 Ringtones commands

Ringtones are mostly supported only for older phones. For recent phones you usually just upload them to some folder in phone filesystem.

There are recognized various file formats by options described below: rttl, binary format created for Gammu, mid (saving), re (reading), ott, communicator, ringtones format found in fkn.pl, wav (saving), ime/imy (saving), rng, mmf (for Samsung).

copyringtone `source destination [RTTL|BINARY]`

Copy source ringtone to destination.

getphoneringtone `location [file]`

Get one of "default" ringtones and saves into file

getringtone `location [file]`

Get ringtone from phone in RTTL or BINARY format.

Locations are numerated from 1.

getringtoneslist

playringtone `file`

Play approximation of ringtone over phone buzzer. File can be in RTTL or BINARY (Nokia DCT3) format.

playsavedringtone `number`

Play one of built-in ringtones. This option is available for DCT4 phones. For getting ringtones list use `gammu getringtoneslist`.

setringtone `file [-location location] [-scale] [-name name]`

Set ringtone in phone. When don't give location, it will be written "with preview" (in phones supporting this feature like 61xx or 6210).

-scale

Scale information will be added to each note of RTTL ringtone. It will avoid scale problems available during editing ringtone in composer from phone menu (for example, in Nokia 33xx).

Note: When use ~ char in ringtone name, in some phones (like 33xx) name will blink later in phone menus.

9.2.9 Calendar notes commands

In Nokia 3310, 3315 and 3330 these are named "Reminders" and have some limitations (depending on phone firmware version).

deletecalendar `start [stop]`

Deletes selected calendar entries in phone.

getallcalendar

Retrieves all calendar entries from phone.

getcalendar `start [stop]`

Retrieves selected calendar entries from phone.

9.2.10 To do list commands

deletetodo `start [stop]`

Deletes selected todo entries in phone.

getalltodo

Retrieves all todo entries from phone.

gettodo `start [stop]`

Retrieves selected todo entries from phone.

9.2.11 Notes commands

getallnotes

Reads all notes from the phone.

Note: Not all phones supports this function, especially most Sony Ericsson phones even if they have notes inside phone.

9.2.12 Date, time and alarm commands

getalarm `[start]`

Get alarm from phone, if no location is specified, 1 is used.

getdatetime

Get date and time from phone

setalarm `hour minute`

Sets repeating alarm in phone on selected time.

setdatetime `[HH:MM[:SS]] [YYYY/MM/DD]`

Set date and time in phone to date and time set in computer. Please note, that this option doesn't show clock on phone screen. It only set date and time.

Note: You can make such synchronization each time, when will connect your phone and use Gammu. See *SynchronizeTime* in *Gammu Configuration File* for details.

9.2.13 Categories commands

Note: Categories are supported only on few phones (Alcatel).

---

addcategory `TODO|PHONEBOOK text`

getallcategory `TODO|PHONEBOOK`

getcategory `TODO|PHONEBOOK start [stop]`

listmemorycategory `text|number`

listtodocategory `text|number`

9.2.14 Backing up and restoring commands

addnew `file [-yes] [-memory ME|SM|..]`

Adds data written in file created using `gammu backup` command. All things backed up `gammu backup` can be restored (when made backup to Gammu text file).

Please note that this adds all content of backup file to phone and does not care about current data in the phone (no duplicates are detected).

Use -yes parameter to answer yes to all questions (you want to automatically restore all data).

Use -memory parameter to force usage of defined memory type for storing entries regardless what backup format says.

addsms `folder file [-yes]`

Adds SMSes from file (format like `gammu backupsms` uses) to selected folder in phone.

backup `file [-yes]`

Backup your phone to file. It's possible to backup (depends on phone and backup format):

- phonebook from SIM and phone memory
- calendar notes
- SMSC settings
- operator logo
- startup (static) logo or startup text
- WAP bookmarks
- WAP settings
- caller logos and groups
- user ringtones

There are various backup formats supported and the backup format is guessed based on file extension:

- `.lmb` - Nokia backup, supports contacts, caller logos and startup logo.
- `.vcs` - vCalendar, supports calendar and todo.
- `.vcf` - vCard, supports contacts.
- `.ldif` - LDAP import, supports contacts.
- `.ics` - iCalendar, supports calendar and todo.
- Any other extension is Gammu backup file and it supports all data mentioned above, see *Backup Format* for more details.

By default this command is interactive and asks which items tou want to backup.

Use -yes for answering yes to all questions.

backupsms file [-yes|-all]

Stores all SMSes from phone to file into *SMS Backup Format*.

By default this command is interactive and asks which folders you want to backup and whether you want to remove messages from phone afterwards.

Use -yes for answering yes to all questions (backup all messages and delete them from phone), or -all to just backup all folders while keeping messages in phone.

restore file [-yes]

> **Warning:** Please note that restoring deletes all current content in phone. If you want only to add entries to phone, use `gammu addnew`.

Restore settings written in file created using `gammu backup` command.

In some phones restoring calendar notes will not show error, but won't be done, when phone doesn't have set clock inside.

restoresms file [-yes]

> **Warning:** Please note that this overwrites existing messages in phone (if it supports it).

Restores SMSes from file (format like `gammu backupsms` uses) to selected folder in phone.

savefile TYPE [type options]

Converts between various file formats supported by Gammu, following types are supported:

BOOKMARK target.url file location

Converts backup format supported by Gammu to vBookmark file.

CALENDAR target.vcs file location

Allows one to convert between various backup formats which gammu supports for calendar events. The file type is guessed (for input file guess is based on extension and file content, for output solely on extension).

TODO target.vcs file location

Allows one to convert between various backup formats which gammu supports for todo events. The file type is guessed (for input file guess is based on extension and file content, for output solely on extension).

VCARD10|VCARD21 target.vcf file SM|ME location

Allows one to convert between various backup formats which gammu supports for phonebook events. The file type is guessed (for input file guess is based on extension and file content, for output solely on extension).

See also:

`gammu convertbackup`

convertbackup source.file output.file

New in version 1.28.94.

Converts backup between formats supported by Gammu. Unlike `gammu savefile`, this does not give you any options what to convert, it simply takes converts all what can be saved into output file.

See also:

`gammu savefile`

9.2.15 Nokia specific commands

nokiaaddfile TYPE [type options]

Uploads file to phone to specific location for the type:

APPLICATION|GAME file [-readonly] [-overwrite] [-overwriteall]

Install the *.jar/*.jad file pair of a midlet in the application or game menu of the phone. You need to specify filename without the jar/jad suffix, both will be added automatically.

-overwrite

Delete the application's .jad and .jar files before installing, but doesn't delete the application data.

-overwriteall

Delete the application (same as -overwrite) and all it's data.

You can use *jadmaker* to generate a .jad file from a .jar file.

GALLERY|GALLERY2|CAMERA|TONES|TONES2|RECORDS|VIDEO|PLAYLIST|MEMORYCARD file [-name name] [-protecte

nokiaaddplaylists

Goes through phone memory and generated playlist for all music files found.

To manually manage playlists:

```
gammu addfile a:\\predefplaylist filename.m3u
```

Will add playlist filename.m3u

```
gammu getfilesystem
```

Will get list of all files (including names of files with playlists)

```
gammu deletefiles a:\\predefplaylist\\filename.m3u
```

Will delete playlist filename.m3u

Format of m3u playlist is easy (standard mp3 playlist):

First line is #EXTM3U, next lines contain names of files (b:\file1.mp3, b:\folder1\file2.mp3, etc.). File needs t have \r\n terminated lines. So just run **unix2dos** on the resulting file before uploading it your your phone.

nokiacomposer file

Show, how to enter RTTL ringtone in composer existing in many Nokia phones (and how should it look like).

nokiadebug filename [[v11-22] [,v33-44]...]

nokiadisplayoutput

nokiadisplaytest number

nokiagetadc

nokiagetoperatorname

6110.c phones have place for name for one GSM network (of course, with flashing it's possible to change all names, but Gammu is not flasher ;-)). You can get this name using this option.

nokiagetpbkfeatures memorytype

nokiagett9

This option should display T9 dictionary content from DCT4 phones.

nokiagetvoicerecord `location`

Get voice record from location and save to WAV file. File is coded using GSM 6.10 codec (available for example in win32). Name of file is like name of voice record in phone.

Created WAV files require GSM 6.10 codec to be played. In Win XP it's included by Microsoft. If you deleted it by accident in this operating system, make such steps:

1. Control Panel
2. Add hardware
3. click Next
4. select "Yes. I have already connected the hardware
5. select "Add a new hardware device
6. select "Install the hardware that I manually select from a list
7. select "Sound, video and game controllers
8. select "Audio codecs
9. select "windows\system32" directory and file "mmdriver.inf
10. if You will be asked for file msgsm32.acm, it should unpacked from Windows CD
11. now You can be asked if want to install unsigned driver (YES), about select codec configuration (select what you want) and rebotting PC (make it)

nokiamakecamerashoot

nokianetmonitor `test`

Takes output or set netmonitor for Nokia DCT3 phones.

See also:

For more info about this option, please visit Marcin's page and read netmonitor manual there.

Note: test `243` enables all tests (after using command **gammu nokianetmonitor 243** in some phones like 6210 or 9210 have to reboot them to see netmonitor menu)

---

nokianetmonitor36

Reset counters from netmonitor test 36 in Nokia DCT3 phones.

See also:

For more info about this option, please visit Marcin's page and read netmonitor manual there.

nokiasecuritycode

Get/reset to "12345" security code

nokiaselftests

Perform tests for Nokia DCT3 phones.

Note: EEPROM test can show an error when your phone has an EEPROM in flash (like 82xx/7110/62xx/33xx). The clock test will show an error when the phone doesn't have an internal battery for the clock (like 3xxx).

---

nokiasetlights `keypad|display|torch on|off`

nokiasetoperatorname `[networkcode name]`

nokiasetphonemenus

Enable all (?) possible menus for DCT3 Nokia phones:

1. ALS (Alternative Line Service) option menu
2. vibra menu for 3210
3. 3315 features in 3310 5.45 and higher
4. two additional games (React and Logic) for 3210 5.31 and higher
5. WellMate menu for 6150
6. NetMonitor

and for DCT4:

1. ALS (Alternative Line Service) option menu
2. Bluetooth, WAP bookmarks and settings menu, ... (6310i)
3. GPRS Always Online
4. and others...

nokiasetvibralevel `level`

Set vibra power to "level" (given in percent)

nokiatuneradio

nokiavibratest

9.2.16 Siemens specific commands

siemensnetmonact `netmon_type`

Enables network monitor in Siemens phone. Currently known values for type are 1 for full and 2 for simple mode.

siemensnetmonitor `test`

siemenssatnetmon

9.2.17 Network commands

getgprspoint `start [stop]`

listnetworks `[country]`

Show names/codes of GSM networks known for Gammu

networkinfo

Show information about network status from the phone.

setautonetworklogin

9.2.18 WAP settings and bookmarks commands

deletewapbookmark `start [stop]`
Delete WAP bookmarks from phone.

Locations are numerated from 1.

getchatsettings `start [stop]`

getsyncmlsettings `start [stop]`

getwapbookmark `start [stop]`
Get WAP bookmarks from phone.

Locations are numerated from 1.

getwapsettings `start [stop]`
Get WAP settings from phone.

Locations are numerated from 1.

9.2.19 MMS and MMS settings commands

getallmms `[-save]`

geteachmms `[-save]`

getmmsfolders

getmmssettings `start [stop]`

readmmsfile `file [-save]`

9.2.20 FM radio commands

getfmstation `start [stop]`
Show info about FM stations in phone

9.2.21 Phone settings commands

getcalendarsettings
Displays calendar settings like first day of week or automatic deleting of old entries.

getprofile `start [stop]`

resetphonesettings `PHONE|DEV|UIF|ALL|FACTORY`

> **Warning:** This will delete user data, be careful.

Reset phone settings.

PHONE
Clear phone settings.

DEV
Clear device settings.

ALL
Clear user settings.

- removes or set logos to default
- set default phonebook and other menu settings
- clear T9 words,
- clear call register info
- set default profiles settings
- clear user ringtones

UIF
Clear user settings and disables hidden menus.

- changes like after `ALL`
- disables netmon and PPS (all "hidden" menus)

FACTORY
Reset to factory defaults.

- changes like after `UIF`
- clear date/time

9.2.22 Dumps decoding commands

Note: These commands are available only if Gammu was compiled with debugging options.

decodebinarydump `file [phonemodel]`
Decodes a dump made by Gammu with *LogFormat* set to `binary`.

decodesniff `MBUS2|IRDA file [phonemodel]`
Allows one to decode sniffs. See *Discovering protocol* for more details.

9.2.23 Other commands

entersecuritycode `PIN|PUK|PIN2|PUK2|PHONE|NETWORK code|- [newpin|-]`
Allow one to enter security code from PC. When code is `-`, it is read from stdin.

In case entering PUK, some phones require you to set new PIN as well.

presskeysequence `mMnNpPuUdD+-123456789*0#gGrR<>[]hHcCjJfFoOmMdD@`
Press specified key sequence on phone keyboard

mM
Menu

nN
Names key

pP
Power

uU
Up

dD
Down

+-
+-

gG
Green

rR
Red

123456789*0#
numeric keyboard

reset SOFT|HARD

Make phone reset:

SOFT
without asking for PIN

HARD
with asking for PIN

Note: Some phones will ask for PIN even with SOFT option.

Warning: Some phones will reset user data on HARD reset.

setpower ON|OFF

New in version 1.33.90.

Turns off or on the phone.

Note: This is usually required for built in modules in notebooks.

screenshot filename

Captures phone screenshot and saves it as filename. The extension is automatically appended to filename based on what data phone provides.

9.2.24 Batch mode commands

batch [file]

Starts Gammu in a batch mode. In this mode you can issue several commands each on one line. Lines starting with # are treated as a comments.

By default, commands are read from standard input, but you can optionally specify a file from where they would be read (special case - means standard input).

9.2.25 Configuration commands

searchphone [-debug]

Attempts to search for a connected phone.

> **Warning:** Please note that this can take a very long time, but in case you have no clue how to configure phone connection, this is a convenient way to find working setup for Gammu.

install [-minimal]

Installs applet for currently configured connection to the phone.

You can configure search path for installation files by *DataPath*.

The -minimal parameter forces installation of applet only without possible support libraries, this can be useful for updates.

9.2.26 Gammu information commands

checkversion [STABLE]

Checks whether there is newer Gammu version available online (if Gammu has been compiled with CURL). If you pass additional parameter STABLE, only stable versions will be checked.

features

Print information about compiled in features.

help [topic]

Print help. By default general help is printed, but you can also specify a help category to get more detailed help on some topic.

version

Print version information and license.

9.3 Return values

gammu returns 0 on success. In case of failure non zero code is returned.

1

Out of memory or other critical error.

2

Invalid command line parameters.

3
Failed to open file specified on command line.

4
Program was interrupted.

98
Gammu library version mismatch.

99
Functionality has been moved. For example to *gammu-smsd*.

Errors codes greater than 100 map to the GSM_Error values increased by 100:

101
No error.

102
Error opening device. Unknown, busy or no permissions.

103
Error opening device, it is locked.

104
Error opening device, it doesn't exist.

105
Error opening device, it is already opened by other application.

106
Error opening device, you don't have permissions.

107
Error opening device. No required driver in operating system.

108
Error opening device. Some hardware not connected/wrongly configured.

109
Error setting device DTR or RTS.

110
Error setting device speed. Maybe speed not supported.

111
Error writing to the device.

112
Error during reading from the device.

113
Can't set parity on the device.

114
No response in specified timeout. Probably phone not connected.

115
Frame not requested right now. See <https://wammu.eu/support/bugs/> for information how to report it.

116
Unknown response from phone. See <https://wammu.eu/support/bugs/> for information how to report it.

117
Unknown frame. See <https://wammu.eu/support/bugs/> for information how to report it.

118
Unknown connection type string. Check config file.

119
Unknown model type string. Check config file.

120
Some functions not available for your system (disabled in config or not implemented).

121
Function not supported by phone.

122
Empty entry.

123
Security error. Maybe no PIN?

124
Invalid location. Maybe too high?

125
Functionality not implemented. You are welcome to help authors with it.

126
Memory full.

127
Unknown error.

128
Can not open specified file.

129
More memory required...

130
Operation not allowed by phone.

131
No SMSC number given. Provide it manually or use the one configured in phone.

132
You're inside phone menu (maybe editing?). Leave it and try again.

133
Phone is not connected.

134
Function is currently being implemented. If you want to help, please contact authors.

135
Phone is disabled and connected to charger.

136
File format not supported by Gammu.

137
Nobody is perfect, some bug appeared in protocol implementation. Please contact authors.

138
Transfer was canceled by phone, maybe you pressed cancel on phone.

139
Phone module need to send another answer frame.

140
Current connection type doesn't support called function.

141
CRC error.

142
Invalid date or time specified.

143
Phone memory error, maybe it is read only.

144
Invalid data given to phone.

145
File with specified name already exists.

146
File with specified name doesn't exist.

147
You have to give folder name and not file name.

148
You have to give file name and not folder name.

149
Can not access SIM card.

150
Wrong GNAPPLET version in phone. Use version from currently used Gammu.

151
Only part of folder has been listed.

152
Folder must be empty.

153
Data were converted.

154
Gammu is not configured.

155
Wrong folder used.

156
Internal phone error.

157
Error writing file to disk.

158
No such section exists.

159
Using default values.

160
Corrupted data returned by phone.

161
Bad feature string in configuration.

162
Desired functionality has been disabled on compile time.

163
Bluetooth configuration requires channel option.

164
Service is not running.

165
Service configuration is missing.

166
Command rejected because device was busy. Wait and restart.

167
Could not connect to the server.

168
Could not resolve the host name.

169
Failed to get SMSC number from phone.

170
Operation aborted.

171
Installation data not found, please consult debug log and/or documentation for more details.

172
Entry is read only.

9.4 Examples

9.4.1 Configuration

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

9.4.2 Sending messages

Note: All messages below are sent to number 123456, replace it with proper destination.

Send text message up to standard 160 chars:

```
echo "All your base are belong to us" | gammu sendsms TEXT 123456
```

or

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

Send long text message:

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

or

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

or

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

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

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

Send protected message with ringtone:

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

9.4.3 Retrieving USSD replies

For example for retrieving prepaid card status or retrieving various network info:

```
gammu getussd '#555#'
```

9.4.4 Uploading files to Nokia

Add Alien to applications in your phone (you need to have files Alien.JAD and Alien.JAR in current directory):

```
gammu nokiaaddfile APPLICATION Alien
```

Add file.mid to ringtones folder:

```
gammu nokiaaddfile TONES file.mid
```

9.4.5 Setting operator logo

Set logo for network `230 03` (Vodafone CZ):

```
gammu setbitmap OPERATOR ala.bmp "230 03"
```

9.4.6 Converting file formats

The formats conversion can done using `gammu savefile` or `gammu convertbackup` commands.

Convert single entry (at position 260) from *Backup Format* to vCalendar:

```
gammu savefile CALENDAR output.vcs myCalendar.backup 260
```

Convert first phonebook entry from *Backup Format* to vCard:

```
gammu savefile VCARD21 output.vcf phone.backup ME 1
```

Convert all contacts from backup to vCard:

```
gammu convertbackup phone.backup output.vcf
```

9.4.7 Reporting bugs

There are definitely many bugs, reporting to author is welcome. Please include some useful information when sending bug reports (especially debug logs, operating system, it's version and phone information are needed).

To generate debug log, enable it in *Gammu Configuration File*:

```
[gammu]
YOUR CONNECTION SETTINGS
logfile = /tmp/gammu.log
logformat = textall
```

Alternatively you can specify logging on command line:

```
gammu -d textall -f /tmp/gammu.log ...
```

With this settings, Gammu generates /tmp/gammu.log on each connection to phone and stores dump of communication there. You can also find some hints for improving support for your phone in this log.

See <https://wammu.eu/support/bugs/> for more information on reporting bugs.

Please report bugs to Gammu bug tracker.

CHAPTER

TEN

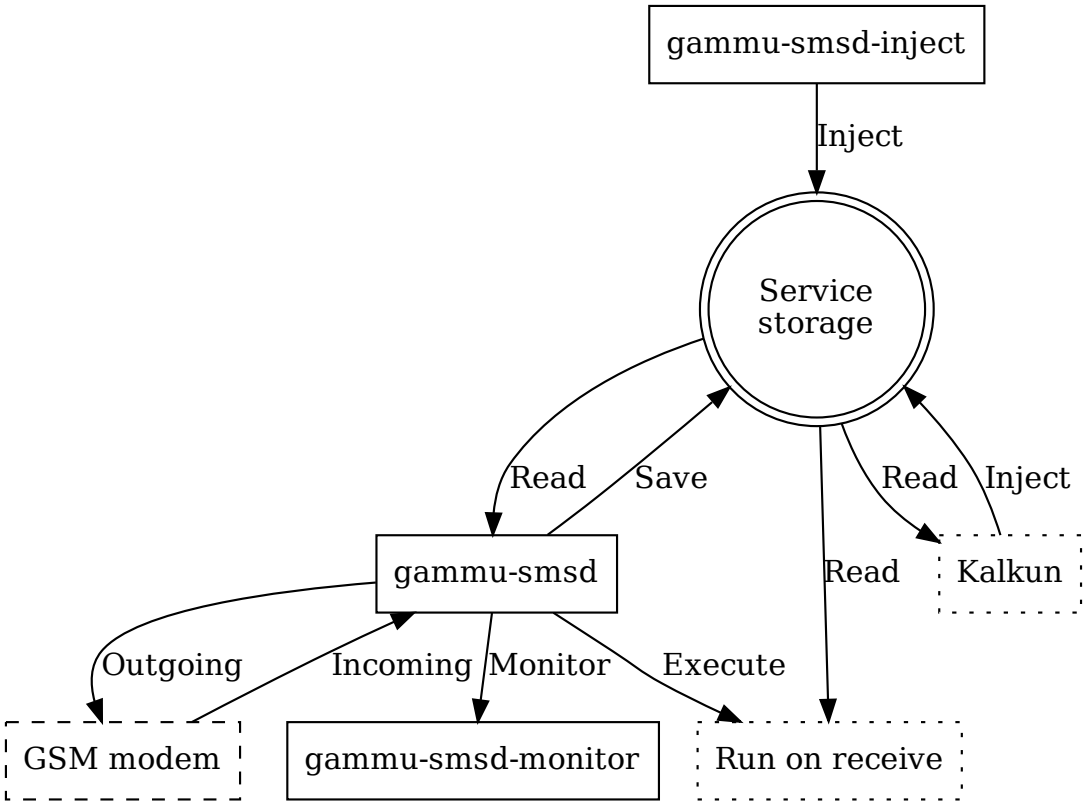
SMS DAEMON

10.1 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.

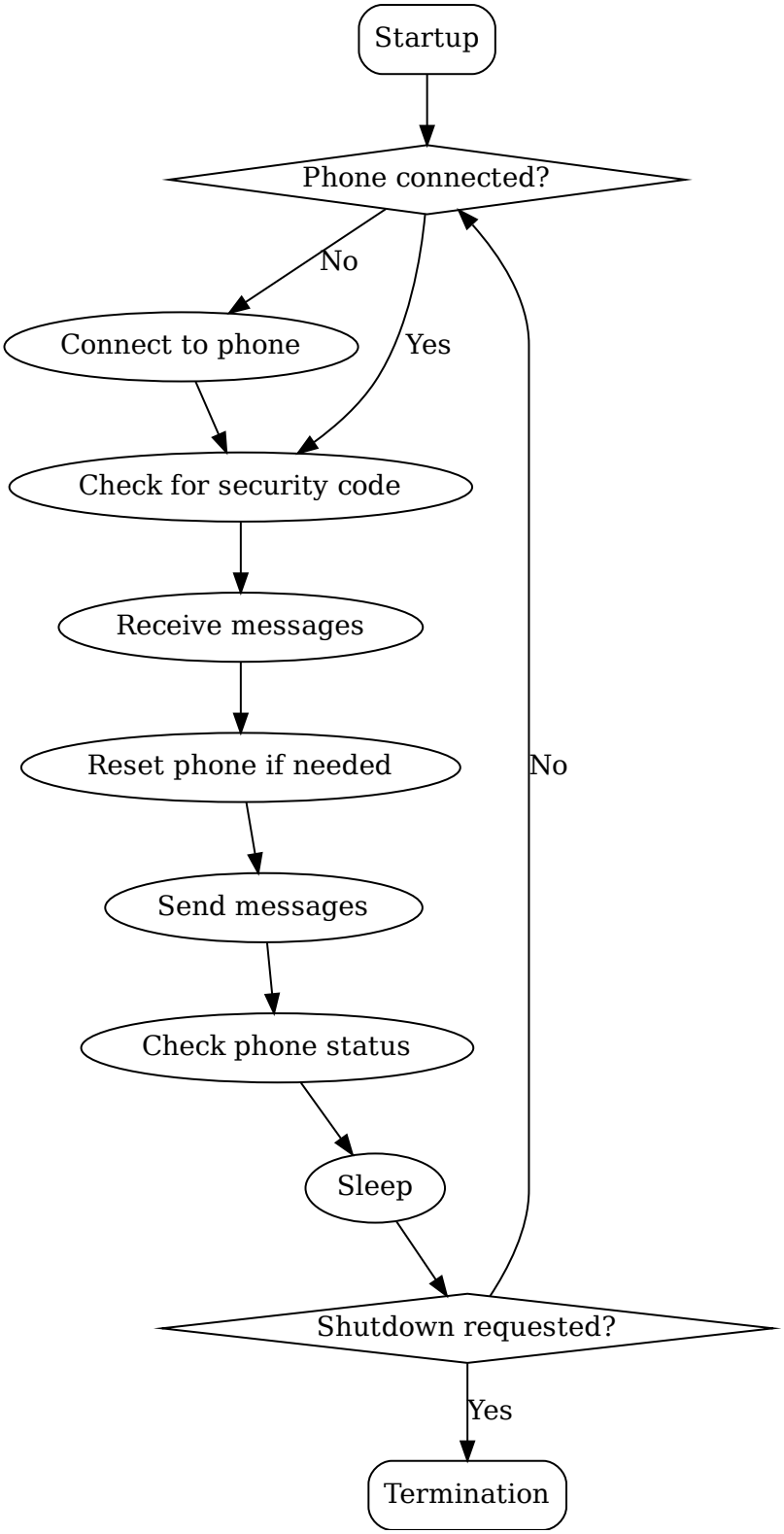
10.1.1 Overall schema

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



10.1.2 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.

10.2 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.

10.2.1 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.

10.2.2 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).

10.2.3 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*.

10.2.4 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()*.

10.2.5 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.

10.3 Program Manuals

10.3.1 gammu-smsd

Synopsis

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

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.

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.

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
```

(continues on next page)

(continued from previous page)

```
  do
    sleep 1
  done
  sleep 1
fi
```

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.

10.3.2 gammu-smsd-inject

Synopsis

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

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`.

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#'
```

10.3.3 gammu-smsd-monitor

Synopsis

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

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).

10.4 SMSD Configuration File

10.4.1 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 file format*.

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 *Gammu Configuration File* 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

10.4.2 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.

10.4.3 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.

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 Utility*, see *SMS Backup Format*.

`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`.

10.4.4 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.

10.4.5 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 *Gammu Configuration File* for some examples how it can look like.

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/
```

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
```

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
```

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
```

Numbers filtering

Process only messages from 123456 number:

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

Do not process messages from evil number 666:

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

Debugging

Enabling debugging:

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

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]
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
```

10.5 RunOnReceive Directive

10.5.1 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.

10.5.2 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.

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*).

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

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:

Can I use Gammu to receive MMS?

DECODED_1_MMS_SIZE

Size of MMS as specified in MMS indication message.

10.5.3 Examples

Activating RunOnReceive

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

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

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
```

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
```

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
```

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
```

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

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))
```

10.6 Backend services

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

10.6.1 Files backend

Description

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

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 *SMS Backup Format*

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

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 *SMS Backup Format*

<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.

10.6.2 SQL Service

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

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 *Gammu Configuration File* documentation.

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
```

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

%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

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:

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

```
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

10.6.3 MySQL Backend

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*.

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

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.

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`
--
```

(continues on next page)

(continued from previous page)

```
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`)
) 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',
```

(continues on next page)

(continued from previous page)

```
↪'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',
  `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',
```

(continues on next page)

(continued from previous page)

```
  `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 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);
```

(continues on next page)

(continued from previous page)

```
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
-- 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,
```

(continues on next page)

(continued from previous page)

```
  `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`
--

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`
--
```

(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 '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;
```

(continues on next page)

(continued from previous page)

```
--
-- 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,
  `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
--
```

(continues on next page)

(continued from previous page)

```
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;
    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.

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.

10.6.4 PostgreSQL Backend

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*.

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

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
```

(continues on next page)

(continued from previous page)

```
    NEW."UpdatedInDB" := LOCALTIMESTAMP(0);
    RETURN NEW;
  END;
$update_timestamp$ LANGUAGE plpgsql;

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

--
-- 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);

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

--
```

(continues on next page)

(continued from previous page)

```
-- 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,
  "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',
```

(continues on next page)

(continued from previous page)

```
  "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',
  "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',
```

(continues on next page)

(continued from previous page)

```
  "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
  ('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␣
```

(continues on next page)

(continued from previous page)

```
↪update_timestamp();

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

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

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"
```

(continues on next page)

(continued from previous page)

```
--

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.

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.

10.6.5 DBI Backend

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.

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

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

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',
```

(continues on next page)

(continued from previous page)

```
  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,
  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)
);
```

(continues on next page)

(continued from previous page)

```
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',
  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)
```

(continues on next page)

(continued from previous page)

```
);

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;
  END;
```

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

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.

10.6.6 ODBC Backend

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*.

Supported drivers

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

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.

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/

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*

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

10.6.7 Null Backend

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.

Configuration

The configuration file then can look like:

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

See also:

SMSD Configuration File

10.6.8 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.

Receiving of messages

Received messages are stored in *inbox* table.

Transmitting of messages

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

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.

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.

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 4c00200074006500730074002000 6d00650073007300610067
↪',
    '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.

10.7 Developer documentation

10.7.1 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*).

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.

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:

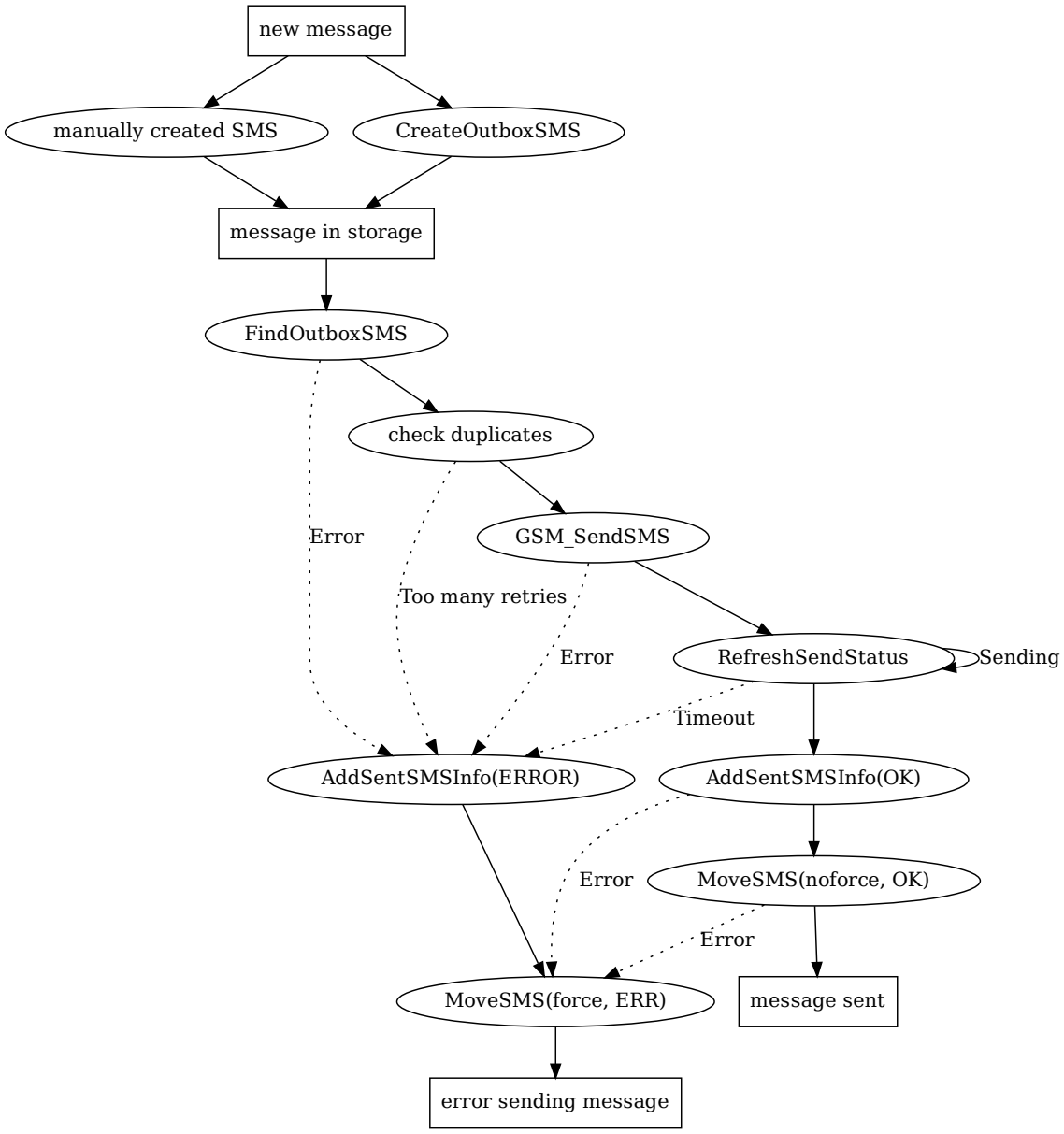
- `GSM_SMSDService::CreateOutboxSMS()` or direct manipulation with backend storage creates new ID
- `GSM_SMSDService::FindOutboxSMS()` returns ID of message to process
- `GSM_SMSDService::AddSentSMSInfo()` and `GSM_SMSDService::RefreshSendStatus()` are then notified using this ID about sending of the message
- `GSM_SMSDService::MoveSMS()` then moves the message based on ID to sent items

The lifetime of ID for incoming messages:

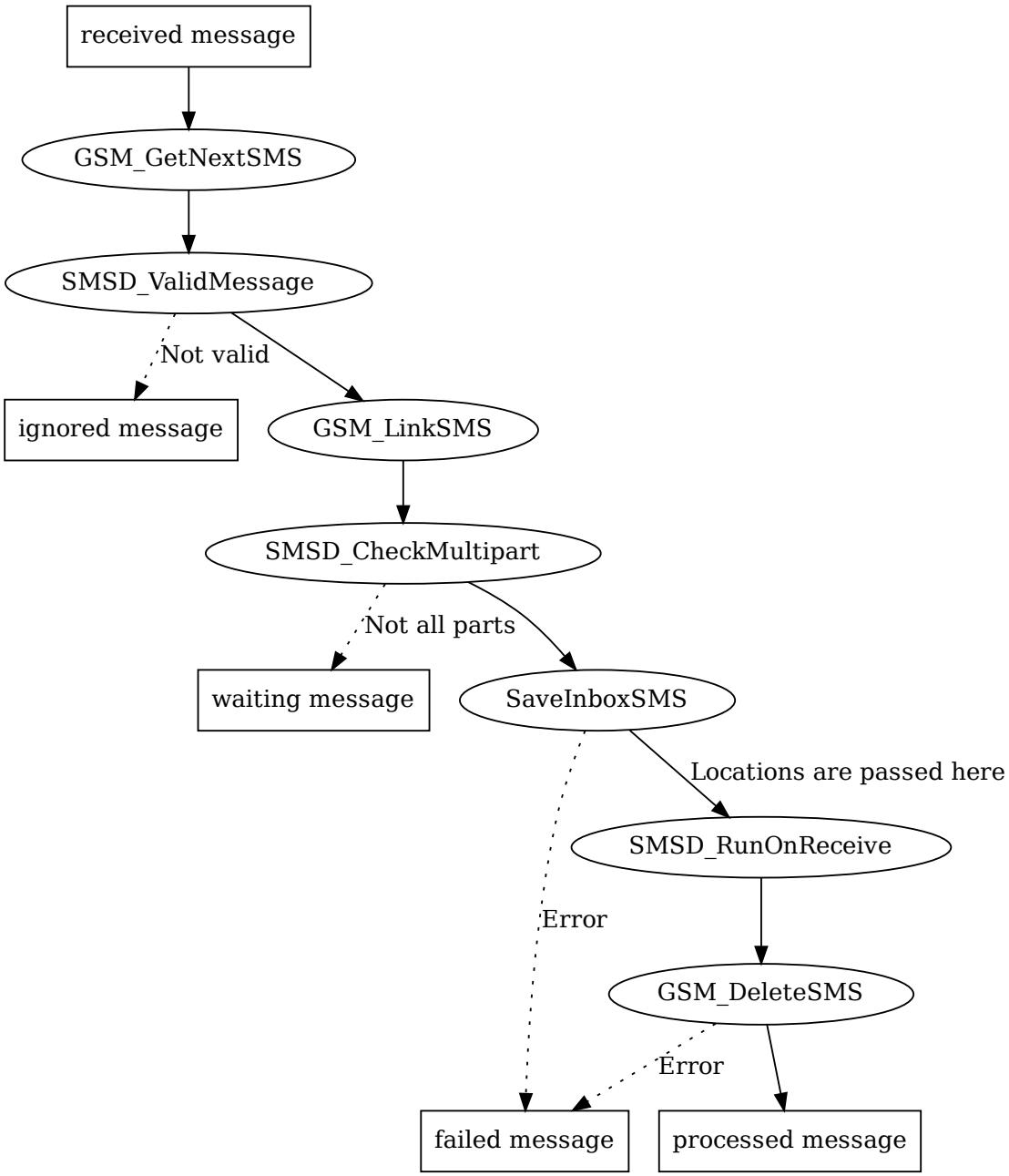
- `GSM_SMSDService::SaveInboxSMS()` generates the message

- *RunOnReceive Directive* uses this ID

10.7.2 Message Sending Workflow



10.7.3 Message Receiving Workflow



CHAPTER

ELEVEN

MISCELLANEOUS UTILITIES

11.1 gammu-detect

New in version 1.28.95.

11.1.1 Synopsis

```
gammu-detect [OPTIONS]
```

11.1.2 Description

Script to detect available devices, which might be suitable for *Gammu Utility*.

Note: This program lists all devices, which might be suitable, it does not do any probing on devices them self.

Currently it supports following devices:

- USB devices using udev
- Serial ports using udev
- Serial ports on Windows
- Bluetooth devices using Bluez

Note: Supported devices depend on platform you are using and compiled in features. You can find out what is actually compiled in by running `gammu-detect -v`.

This program follows the usual GNU command line syntax, with long options starting with two dashes (`--`). A summary of options is included below.

`-h, --help`
Show summary of options.

`-d, --debug`
Show debugging output for detecting devices.

`-v, --version`
Show version information and compiled in features.

-u, --no-udev
Disables scanning of udev.

-b, --no-bluez
Disables scanning using Bluez.

-w, --no-win32-serial
Disables scanning of Windows serial ports.

11.1.3 Output

The output of *gammu-detect* is configuration file for Gammu (see *Gammu Configuration File*) with configuration section for every device which might be used with *Gammu Utility*.

Note: You can choose which section to use in *Gammu Utility* by `gammu -s`.

When invoked as `gammu-detect -d`, also all examined devices are listed as comments in the output.

11.1.4 Example

```
; Configuration file generated by gammu-detect.
; Please check The Gammu Manual for more information.

[gammu]
device = /dev/ttyACM0
name = Nokia E52
connection = at

[gammu1]
device = /dev/ttyACM1
name = Nokia E52
connection = at

[gammu2]
device = /dev/ttyS0
name = Phone on serial port 0
connection = at

[gammu3]
device = /dev/ttyS1
name = Phone on serial port 1
connection = at

[gammu4]
device = /dev/ttyS2
name = Phone on serial port 2
connection = at

[gammu5]
device = /dev/ttyS3
```

(continues on next page)

(continued from previous page)

```
name = Phone on serial port 3
connection = at

[gammu6]
device = 5C:57:C8:BB:BB:BB
name = Nokia E52
connection = bluephonet
```

11.2 gammu-config

11.2.1 Synopsis

```
gammu-config [-f|--force] [-c|--config CONFIG]
```

11.2.2 Description

Script to help configuring *Gammu Utility*.

This program follows the usual GNU command line syntax, with long options starting with two dashes (-). A summary of options is included below.

-h, --help
: Show summary of options.

-f, --force
: Force configuring even if config already exists.

-c, --config CONFIG
: Define which configuration file to use.

11.3 jadmaker

11.3.1 Synopsis

```
jadmaker [-f|--force] [-u|--url URL] <filename.jar>...
```

11.3.2 Description

Script to generate JAD file from JAR file.

This program follows the usual GNU command line syntax, with long options starting with two dashes (-). A summary of options is included below.

-h, --help
: Show summary of options.

-f, --force

Force rewriting of JAD file even if exists.

-u, --url URL

Define URL to be included in JAD file.

CHAPTER

TWELVE

TESTING GAMMU

12.1 Gammu Testsuite

Gammu comes with quite big test suite. It covers some basic low level functions, handling replies from the phone and also does testing of command line utilities and SMSD.

12.1.1 Running the tests

You can run the test suite this using `make test`. CMake build system uses for testing CTest, which also includes option to connect to dashboard and submit test results there, so that they can be reviewed and fixed by others. To participate in this testing, you need just to run `make Experimental` which also does submission to the dashboard.

There are some more options for testing:

`make test`

Runs testsuite with no uploading of results.

`make Experimental`

Runs testsuite and uploads results to the dashboard.

`make ExperimentalMemCheck`

This checks memory accesses using valgrind during tests and submits report. You need to do this after `make Experimental` and you can submit results using `make ExperimentalSubmit`.

Coverage reports

To get test coverage reports, you need to configure project using `cmake -DCOVERAGE=ON`

Nightly testing

Currently several machines do compile and test Gammu every night. If you want to tak part of this, just ensure that your machine executes test suite every night (preferably after 3:00 CET). You can select either `make Nightly` to do regular testing or `make NightlyMemoryCheck` to test with valgrind. Also you can enable coverage tests as described above.

Running single test

You can run single test by directly calling ctest:

```
ctest -R test-name
```

Adding `-V` runs it in verbose mode with all test output:

```
ctest -V -R test-name
```

12.1.2 Collecting results

The tests are ran daily on several platforms and you can find the results on Travis.

The coverage reports are at Coveralls.

12.1.3 Testing of SMSD

SMSD tests are performed using *Dummy Driver* and uses file backend and sqlite database by default. For this you nee Gammu compiled with libdbi, have installed sqlite driver for libdbi and have **`sqlite3`** binary available on the system.

Testing of additional database backends must be enabled separately:

`MYSQL_TESTING`:
: you need to have setup MySQL server with database where SMSD can play.

`PSQL_TESTING`
: you need to have setup PostgreSQL server with database where SMSD can play.

12.1.4 Testing of command line utility

Gammu command line tests are performed using *Dummy Driver* where required. It covers most of command line interface, but some parts need to be explicitly enabled:

`ONLINE_TESTING`:
: enable testing of features which require internet access

12.1.5 Testing of Python interface

Python module tests are performed using *Dummy Driver* where required. It does also cover testing of SMSD interface, which is done using libdbi(sqlite) driver.

12.1.6 Testing of reply functions

The `tests` directory contains various tests which do inject data into reply functions and check their response.

12.1.7 Testing of data parsing

The `tests` directory contains various tests which just try to parse various file formats supported by libGammu.

12.1.8 Configuration of the test suite

You can pass various parameters to configure the test suite:

Programs used for testing

SH_BIN
Path to the **sh** program

BASH_BIN
Path to the **bash** program

SQLITE_BIN
Path to the **sqlite3** program

SED_BIN
Path to the **sed** program

MYSQL_BIN
Path to the **mysql** program

PSQL_BIN
Path to the **psql** program

Limiting testsuite

ONLINE_TESTING
Enable testing of parts which use remote servers, requires connection to interned

PSQL_TESTING
Enable testing of PostgreSQL SMSD backend, requires configured PostgreSQL database

MYSQL_TESTING
Enable testing of MySQL SMSD backend, requires configured MySQL database

Database backends configuration

PSQL_HOST
Host to use for PostgreSQL tests (default: `127.0.0.1`)

PSQL_DATABASE
Database to use for PostgreSQL tests (default: `smsd`)

PSQL_USER
User to use for PostgreSQL tests (default: `smsd`)

PSQL_PASSWORD
Password to use for PostgreSQL tests (default: `smsd`)

MYSQL_HOST
Host to use for MySQL tests (default: `127.0.0.1`)

MYSQL_DATABASE
Database to use for MySQL tests (default: `smsd`)

MYSQL_USER
User to use for MySQL tests (default: `smsd`)

MYSQL_PASSWORD
Password to use for MySQL tests (default: `smsd`)

ODBC_DSN`
ODBC DSN to use for ODBC tests (default: `smsd`). Currently needs to point to MySQL database.

12.2 Dummy Driver

New in version 1.22.93.

The `dummy` driver in Gammu emulates all operations on filesystem. It is used by *Gammu Testsuite*, but it is also very helpful for application developers, because they can test the functionality without using real phone and avoiding risk of corrupting data in the phone.

12.2.1 Filesystem structure

The dummy driver emulates all phone functionality on filesystem. The *Device* configuration directive sets top level directory, where all data are stored.

This directory contains file `operations.log`, where are logged operations which do not modify any data in the dummy phone (eg. sending message).

Messages

Messages are stored in `sms/<FOLDER>` directories (`<FOLDER>` is in range 1-5) in Gammu native smsbackup format.

Phonebook

Phonebook (and calls registers) are stored in `pbk/<MEMORY>` (`<MEMORY>` is type of memory like `ME` or `SM`) directories in vCard format.

Notes

Notes are stored in `note` directory in vNote format.

Calendar

Calendar entries are stored in `calendar` directory in vCalendar format.

Todo

Todo entries are stored in `todo` directory in vCalendar format.

Filesystem

Filesystem is stored in `fs` directory. You can create another subdirectories there.

12.2.2 Other features

By specifying *Features* you can configure some specific behavior:

DISABLE_GETNEXT
: Makes the dummy driver fail all GetNext* calls as not supported (with exception of GetNextSMS* and GetNextFile*).

DISABLE_GETNEXTSMS
: Makes the dummy driver fail all GetNextSMS* calls as not supported.

12.2.3 Examples

To use dummy driver, you need something like following in `~/.gammurc`:

```
[gammu]
model = dummy
connection = none
device = /path/to/directory/
```

For disabling GetNext* functions within dummy driver, you need something like following in `~/.gammurc`:

```
[gammu]
model = dummy
connection = none
features = DISABLE_GETNEXT
device = /path/to/directory/
```

CHAPTER

THIRTEEN

PHONE PROTOCOLS

13.1 Discovering protocol

You need to get a communication dump to be able to understand protocol or discover new commands. As most vendors provide some software for Windows, all following sections assume you do the sniffing on Windows.

13.1.1 USB

For USB there exist various tools to dump USB communication. The dumps can be later analyzed and used to discover protocol details or unknown commands. One of the best free tools available currently is UsbSnoop.

In directory `contrib/usbsnoop` in Gammu sources you can find some tools to decode the output.

13.1.2 Serial port

Download Portmon, which allows one to capture bytes sent and received by ready binary software.

If you have log saved by PortMon and protocol is the same to "old" Nokia protocols, can use Gammu to decode it. It's simple:

```
gammu --decodesniff MBUS2 file 6210 > log
```

saves in log decoded MBUS2 dump session. There is used phone module for 6210 and have you have debug info about 6210 specific frames (you don't have to add model). Dump file for –decodesniff and MBUS should be specific:

1. without bytes sent to phone (in Portmon you set it here: "Edit", "Filter/Highlight")
2. in Hex format ("Options", "Show Hex")
3. without Date & Time ("Options", "Show Time" & "Clock Time")

13.1.3 Infrared

First of all you need two computers with IrDA. One running linux, that will sniff and one running windows, which will communicate with the phone and whatever software you want (Nokia, Logomanager, Oxygen Phone Manager). Then you have to get the software from http://www.dev-thomynet.de/nokworld/noktrace/

You have to disable IrDA services on the linux machine and eventually you have to change the default port the 'irda_intercept' program is sniffing from (default ttyS1). On the windows machine you should decrease the maximum transmission speed to 9600bps if possible, because the intercept program doesn't seem to handle speed changes. (9600 is for searching devices in range and then the highest possible speed is chosen) If it isn't possible you have to change the default bitrate in intercept source code, too. Then you won't see anything until the windows machine and

the phone start transmitting data, which isn't too bad. At least here in my setup I could sniff the data coming from phone and sent to it in one go, like that:

```
  win     ------------------>   Nokia
machine   <------------------    phone
                ^^
                ||
              sniffing
               device
```

You get a raw data file (.trc) from the intercept program, which you can then decode to hex with the second program from the above mentioned page. You should possibly be able to use Marcin's magnokii for decoding the trc files, too, but it didn't work for me so I just figured things out from the hex files. In the hex files you should look for primary frames with 00 01 00 in it, because this is the FBUS header which is in every valuable frame sent to phone. It's not really joy to do that, but if it brings support for a new phone it's worth it :-)

13.2 Nokia protocols

Document describing protocol used in Nokia phones.

The data provided is for information purposes only. Some of the frames might be hazardous to your phone. Be careful!!! We do not take any responsibility or liability for damages, etc.

Last update 23.06.2003

Assembled by Balazs Nagy <js@iksz.hu> Alfred R. Nurnberger <arnu@flosys.com> Hugh Blemings <Hugh.Blemings@vsb.com.au> Mike Bradley <mike@trumpington.st> Odinokov Serge <serge@takas.lt> Pavel Janik <Pavel@Janik.cz> Pawel Kot <pkot@linuxnews.pl> Marcin Wiacek <Marcin@MWiacek.com> Jens Bennfors <jens.bennfors@ing.hj.se> Michael Hund <michael@drhund.de> Jay Bertrand <jay.bertrand@libertysurf.fr> <arnu@venia.net> Andrew Kozin Pavel Machek <pavel@ucw.cz> Diego Betancor <dbetancor@duocom.net> ... and other members of gnokii mailing list and authors of some WWW pages.

Note: this information isn't (and can't be) complete. If you know anything about features not listed here or you noticed a bug in this list, please notify us via e-mail. Thank you.

13.2.1 Frame format for MBUS version 1

Request from Computer/Answer from Phone:

```
{ DestDEV, SrcDEV, FrameLength, MsgType, {block}, id, ChkSum }

    where DestDEV, SrcDEV:   0x00: phone
                             0xf8: PC (wakeup msg)
                             0xe4: PC (normal msg)
          FrameLength:       length of data frame. Maximal 0x78. Longer
                             frames are divided into smaller.
          MsgType:           see List
          {block}:           main frame
          id:                request identity number 1..n, incremented after
                             the request is accepted
          ChkSum:            XOR on frame's all numbers
```

Ack from Phone:

```
{ DestDEV, 0x00, FrameLength, MsgType, {block} , id, ChkSum }

   where DestDEV:           taken from original request packet
         FrameLength:       0x7f, when DestDEV = 0xe4
                            0x7e, when DestDEV = 0xf8
         MsgType:           see List. Present only, when DestDEV = 0xf8
         {block}:           main frame. Present only, when DestDEV = 0xf8
         id:                request identity number 1..?, corresponding
                            to the original request packet id
                            the request is accepted
         ChkSum:            XOR on frame's all numbers
```

Update: description above according to the http://www.gadgets.demon.co.uk/nokia21xx/protocol.html.

Pavel Machek <pavel@ucw.cz> wrote:

0x7e is actually registration acknowledge. Both have nothing to do with DestDEV, except that special device needs to be used for registration.

Ack from Computer:

```
{ 0x00, SrcDEV, 0x7f, id, ChkSum }

   where SrcDEV:            taken from response packet
         id:                request identity number 1..?, corresponding
                            to the response packet id
                            the request is accepted
         ChkSum:            XOR on frame's all numbers
```

Port settings:

Speed 9600 bps, Bits 8, ParityOdd, Stop Bits 1, DTR and RTS logic 0

In the MBUS bus, the phone has only one connector for transmission and reception.

Because of this characteristics of the phone connector, every time that the PC writes into the phone it is writing as well into its own Rx. So every time the PC sends info into the phone it finds that same information in its own Rx buffers, like a mirror copy. This should be discarded.

The communications is made like an old cb radio, only one talking at a time. Many transmission are made this way:

- <computer sends request>
- <phone sends ack>
- <phone sends response>
- <computer sends ack>

Some frames are sent from phone without asking for them

You have to implement collision protocol. IE. you should listen for what you are transmitting, and if it does not come back, you have collision.

You should wait for bus to be free for 3 milliseconds before normal message, and for 2.5 milliseconds before acknowledge. You should wait for acknowledge for 200 milliseconds, then retransmit.

13.2.2 Frame format for FBUS version 1

All frames:

```
{ FrameID, FrameLength, MsgType, {block}, SeqNo, ChkSum }

     where FrameID:         0x01 Command frame from computer to Nokia
                            0x02 ??? - Data call frame from computer to Nokia - ???
                            0x03 Data call frame from Nokia to computer
                            0x04 Command frame from Nokia to computer
           FrameLength:     {block} + 2
           MsgType:         see List
           SeqNum:          Sequence number of command in case where direction is
                            from ME to computer, the sequence number is
                            counting from 0x30 to 0x37 and resetting back to 0x30.
                            When direction is from computer to ME,
                            sequence number counts from 0x08 to 0x0f and resets back to␣
↪0x08.
                            It may not be required to be this way.
                            Sequence numbers are used in acknowledging commands.
           ChkSum1:         CRC = 0;
                            for (i = 0; i < (2 + CMD_LEN); i++)
                              CRC ^= frame[i];
```

13.2.3 Frame format for FBUS version 2/Direct IRDA

All frames:

```
{ FrameID, DestDEV, SrcDEV, MsgType, 0x00, FrameLength, {block}, FramesToGo,
  SeqNo, PaddingByte?, ChkSum1, ChkSum2 }

     where FrameID:         0x1c: IR / FBUS
                            0x1e: Serial / FBUS
           DestDev, SrcDev: 0x00: mobile phone
                            0x0c: TE (FBUS) [eg. PC]
           MsgType:         see List
           FrameLength:     {block} + 2 (+ 1 if PaddingByte exists)
           FramesToGo:      0x01 means the last frame
           SeqNo:           [0xXY]
                              X: 4: first block
                                 0: continuing block
                                 Y: sequence number
           PaddingByte:     0x00 if FrameLength would be an odd number
                            anyways it doesn't exists
           ChkSum1:         XOR on frame's odd numbers
           ChkSum2?:        XOR on frame's even numbers
```

13.2.4 Frame format for MBUS version 2

Cable:

```
{ FrameID, DestDEV, SrcDEV, MsgType, FrameLengthLO, FrameLengthHI, {block},
  SeqNo, ChkSum }

     where FrameID:          0x1f: Serial / M2BUS
           DestDev, SrcDev: 0x00: mobile phone
                            0x1d: TE (M2BUS)
                            0x10: TE (M2BUS) (Service Software ?)
                            0x04: Carkit?
                            0x48: DLR3 cable?
                            0xF8: unknown target?
                            0xFF: global target?
           MsgType:         see List
           FrameLength:     {block}
           SeqNo:           sequence number
           ChkSum:          XOR on frame's all numbers
```

Please note that M2BUS has only one checksum: XOR on frame[FrameID..SeqNo]

Ack:

```
{ FrameID, DestDEV, SrcDEV, 0x7f, Id_SeqNo, ChkSum }

     where Id_SeqNo:         Is the sequence number that you are
                             acknowledging (from the other part).
```

Frame format for Infrared:

```
{ FrameID, DestDEV, SrcDEV, MsgType, FrameLengthLo, FrameLengthHi, {block}}

     where FrameID:          0x14
           DestDev, SrcDev: 0x00: mobile phone
                            0x0c: TE [eg. PC]
           MsgType:         see List
           FrameLength:     {block}
```

Frame format for Bluetooth:

```
{ FrameID, DestDEV, SrcDEV, MsgType, FrameLengthLo, FrameLengthHi, {block} }

     where FrameID:          0x19
           DestDev, SrcDev: 0x00: mobile phone
                            0x10: TE [eg. PC]
           MsgType:         see List
           FrameLength:     {block}
```

Frames list format:

```
hex: Short description
   x msg desc                { ... }
    0xXX   -> one byte
    0xXXYY -> two bytes (== 0xXX, 0xYY)
```

(continues on next page)

(continued from previous page)

```
where hex:     message type
      x:       s=send (eg. to mobile), r=receive
      { ... }: data after 0x00, 0x01 header
      {+... }: raw data (without header)
```

13.2.5 Misc (about MBUS version 2)

0x4E commands

(sent from a 5160i TDMA / 6160i TDMA / 6185 CDMA or 7110 GSM phone to the uC in the DLR-3 cable)

DLR-3 req:

1F 48 00 4E 00 02 01 XX SQ CS

frame sent from the phone to the DLR-3 cable (after 15kOhm resistor detected betw. XMIC (3) and DGND (9).) DSR,DCD,CTS flow control data is coded into the 2nd databyte

XX:

- bit.0=/CTS
- bit.1=/DCD
- bit.2=CMD/DATA
- bit.3=DSR
- bit.4-7=0

0x78 / 0x79 commands

(used by handsfree carkit) Works also on GSM phones (5110 / 6110 / etc)

These commands are used by the Nokia Carkits to switch the phone audio path to XMiC and XEAR , turn the phone on/off according to the car ignition, and control the PA loudspeaker amplifier in the carkit and the car radio mute output which silences the car radio during a call

mute status tone:

1F 04 00 78 00 04 01 02 0E 00 SQ CS
status indication = disable carkit audio amplifier (no audio / no tone)

mute status tone:

1F 04 00 78 00 04 01 02 0E 03 SQ CS
status indication = enable carkit audio amplifier (audio / tone present)

mute status call:

1F 04 00 78 00 04 01 02 07 00 SQ CS
status indication = disable radio mute output (no call)

mute status call:

1F 04 00 78 00 04 01 02 07 01 SQ CS
status indication = enable radio mute output (call active)

enable ???:

1F 04 00 78 00 04 01 02 08 01 SQ CS
status indication = enable ??? sent to HFU-2 on power on byte 9 (07,08,0E) seems to be a pointer to a memory location, byte 10 is the data at this memory location.

response from HFU:

1F 00 04 78 00 03 02 01 03 SQ CS
response message from HFU-2 (use unknown)

go HF and IGN on:

1F 00 04 79 00 05 02 01 01 63 00 SQ CS
enables carkit mode + turns phone on + req. mute status

go HF and IGN off:

1F 00 04 79 00 05 02 01 01 61 00 SQ CS
enables carkit mode + powers phone off (1 min delay) + req. mute status

ext. HS Offhk:

1F 00 04 79 00 05 02 01 01 23 00 SQ CS
enables carkit mode + external handset lifted (OFF-Hook)

ext. HS Onhk:

1F 00 04 79 00 05 02 01 01 63 00 SQ CS
enables carkit mode + external handset put back (ON-Hook) Ignition and Hook are coded into one byte

- bit.0 = 0:on power on 1:when in operation
- bit.1 = IGNITION STATUS
- bit.2 = x can be 1 or 0
- bit.3 = 0
- bit.4 = 0
- bit.5 = 1
- bit.6 = Hook (inverted)
- bit.7 = 0

HFU-2 version:
1F 00 04 79 00 12 02 01 02 06 00 56 20 30 36 2E 30 30 0A 48 46 55 32 00 SQ CS

for HFU-2:

1F 04 00 DA 00 02 00 02 SQ CS
function unknown - sent from Nokia phone to HFU-2mute output (call active)

0xD0 commands

init:

1F 00 1D D0 00 01 04 SQ CS
sent by the Service Software or HFU-2 on startup

init resp:

1F 1D 00 D0 00 01 05 SQ CS
response from phone to above frame

13.3 Nokia S40 filesystem SMS format

This text is work in progress and does not claim to be correct or accurate. It is solely based on Gammu dumps received from users. Analysed by Michal Cihar <michal@cihar.com>.

13.3.1 File structure

- **176 bytes header**
  - at offset 7 is length of PDU data
  - at offset 94 is stored remote number in unicode
  - rest is not known
- **PDU data (without SMSC)**
  - here can be sometimes also some failure block, which is not known yet
- structured data header: 0x01 0x00 <LEN>, where <LEN> is length of rest
- structured blocks:

Block: <TYPE = byte> <LENGTH = word> <DATA . . . >

13.3.2 Blocks

0x01
Unknown x00 / x01 (maybe received / sent)

0x02
SMSC number, ASCII

0x03
Text, unicode

0x04
Sender, unicode

0x05
Recipient, unicode

0x06
Unknown x00x00x00x00

0x07
Unknown x00

0x08
Unknown x02 / x00

0x09
Unknown x00x00x00x00

0x0a
Unknown x00

0x0b
Unknown x00

0x0c
Unknown, several values (maybe message reference per number)

0x0d
Unknown x00x00

0x0e
Unknown x00x00

0x0f
Unknown x00x00

0x22
Unknown x00

0x23
Unknown x00x00x00x00

0x24
Unknown x00

0x26
Unknown x00

0x27
Unknown x00

0x2a
Unknown x00

0x2b
some text (Sender?), unicode

To test:

- multiple recipients sms

13.4 Nokia 6110

Assembled by Balazs Nagy <js@iksz.hu> Harri Yli-Torkko <hyt@surfeu.fi> Alfred R. Nurnberger <arnu@flosys.com> Hugh Blemings <Hugh.Blemings@vsb.com.au> Mike Bradley <mike@trumpington.st> Odinokov Serge <serge@takas.lt> Pavel Janik <Pavel@Janik.cz> BORBELY Zoltan <bozo@andrews.hu> Pawel Kot <pkot@linuxnews.pl> Marcin Wiacek <Marcin@MWiacek.com> Walek <walek@pa98.opole.sdi.tpnet.pl> ... and other members of gnokii mailing list and authors of some WWW pages.

The data provided is for information purposes only. Some of the frames might be hazardous to your phone. Be careful!!! We do not take any responsibility or liability for damages, etc.

Note: this information isn't (and can't be) complete. If you know anything about features not listed here or you noticed a bug in this list, please notify us via e-mail. Thank you.

Document describing frames used in GSM/PCN Nokia 6110 and derivatives (Nokia 6130, 6150, 6190, 5110, 5130, 5150, 5190, 3210, 3310)

Correct format is FBUS version 2/Direct IRDA/MBUS version 2 (see nokia.txt for protocol details):

List:

```
0x00: Monitoring values
    r monitoring value      {+0x01, 0x01, block... }
      where block: 0x5e, 0x05, 0x7a(?), 0xd0(?), 0x85(?), 0x02, percentHI, percentLO
                      Battery percent level
                   0x5e, 0x0c, 0x52(?), 0x4b(?), 0x6f(?), 0x02, voltageHI, voltageLO
                      Battery standby voltage
                   ...
0x01: Call Information
    s Make call             { 0x0001, "number", type, block }
                              where type:
                                      0x01 - data call
                                      0x05 - voice call
                        block:
                      data call (non digital lines):
                        0x02,0x01,0x05,0x81,0x01,0x00,0x00,0x01,0x02,0x0a,
                        0x07,0xa2,0x88,0x81,0x21,0x15,0x63,0xa8,0x00,0x00
                      data call (digital lines):
                        0x02,0x01,0x05,0x81,0x01,0x00,0x00,0x01,0x02,0x0a,
                        0x07,0xa1,0x88,0x89,0x21,0x15,0x63,0xa0,0x00,0x06,
                        0x88,0x90,0x21,0x48,0x40,0xbb
                                      voice call:
                        0x01, 0x01, 0x05, 0x81/0x00, sendnum, 0x00, 0x00, 0x01
                                        where:
                                          sendnum (own number sending):
                                              0x01: preset (depends on network)
                                              0x03: on
                                              0x02: off
    r Call going msg        { 0x0002 }
    r Call in progress      { 0x0003, seqnr }
    r Remote end hang up    { 0x0004, seqnr, ?, error (like in netmon in 39) }
    r incoming call alert   { 0x0005, seqnr, numlen, "number", namelen, "name" }
    s Answer call part 2    { 0x0006, seqnr, 0x00 }
    r answered call         { 0x0007, seqnr }
    s Hang up               { 0x0008, seqnr, 0x85 }
    r terminated call       { 0x0009, seqnr }
    r call msg              { 0x000a, seqnr }
    r call held             { 0x0023, seqnr, 0x01 }
    r call resumed          { 0x0025, seqnr, 0x01 }
    r Send DTMF/voice call  { 0x0040}
    s Answer call part 1    { 0x0042,0x05,0x01,0x07,0xa2,0x88,0x81,0x21,0x15,0x63,0xa8,
↪0x00,0x00,
                     0x07,0xa3,0xb8,0x81,0x20,0x15,0x63,0x80 }
    s Sent after issuing    { 0x0042,0x05,0x81,0x07,0xa1,0x88,0x89,0x21,0x15,0x63,0xa0,
↪0x00,0x06,
      data call                          0x88,0x90,0x21,0x48,0x40,0xbb,0x07,0xa3,
      (digital lines)                 0xb8,0x81,0x20,0x15,0x63,0x80 }
    s Sent after issuing    { 0x0042,0x05,0x01,0x07,0xa2,0xc8,0x81,0x21,0x15,0x63,0xa8,
↪0x00,0x00,
      data call                          0x07,0xa3,0xb8,0x81,0x20,0x15,0x63,0x80,
      (non digital lines)             0x01,0x60 }
    s Send DTMF             { 0x0050, length, {ascii codes for DTMF}, 0x01 }

    Note:
```

(continues on next page)

(continued from previous page)

```
      to make data call (non digital lines):
        1.send "Make call" for non digital lines
    2.send "Sent after issuing data call (non digital lines)"
      to make data call (digital lines):
        1.send "Answer call part 1"
    2.send "Sent after issuing data call (digital lines)"
        3.send "Make call" for digital lines
      to answer call:
        1.send "Answer call part 1"
        2.send "Answer call part 2"

0x02: SMS handling
    s Send SMS message      { 0x0001, 0x02, 0x00 (SEND REQUEST), ... }
    r Message sent          { 0x0002 }
    r Send failed           { 0x0003, ?, ?, error (like in netmon in 65)}
    s Get SMS message       { 0x0007, 0x02, location, 0x01, 0x64 }
    s Initiate connection   { 0x000d, 0x00, 0x00, 0x02 }
    r Initiate ACK          { 0x000e, 0x01 }
    r SMS message received  { 0x0010, ..... } (whole message)
    s Set CellBroadcast     { 0x0020, 0x01, 0x01, 0x00, 0x00, 0x01, 0x01 }
                                      for enable cell broadcast ?
                                      0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }
                                      for disable cell broadcast ?
    r Set CellBroadcast OK  { 0x0021, 0x01 }
    r Read CellBroadcast    { 0x0023, ?, ?, ?, channel, ?, message... } ?
    s Set SMS center        { 0x0030, 0x64, priority, checksum? ,0?, format,
                                      validity, {DefaultRecipient no.}[12],
                                      {SMScenter no.}[12], {SMSC name}, 0x00}
                              where tel.no.[12]: {len, type, {number(BCD)}}
                                    type: 0x81: normal
                                          0x91: + (international)
                                          0xd0: alphanumeric
                                    format: 0x00: text
                                            0x22: fax
                                            0x24: voice
                                            0x25: ERMES
                                            0x26: paging
                                            0x31: X.400
                                            0x32: email
                                    validity: 0x0b:  1 hour
                                              0x47:  6 hours
                                              0xa7: 24 hours
                                              0xa9: 72 hours
                                              0xad:  1 week
                                              0xff: max.time
    r Set SMS center OK     { 0x0031 }
    r Set SMS center error  { 0x0032, reason }
    s Get SMS center        { 0x0033, 0x64, priority }
    r SMS center received   { 0x0034, priority, checksum?, format, 0x00?,
                                      validity, {DefaultRecipient no.}[12],
                                      {SMScenter no.}[12], {SMSC name}, 0x00}
                                      tel.no[12]: {len, type, {number(BCD)}}
```

(continues on next page)

(continued from previous page)

```
                              where priority, checksum, type, validity,
                                    tel.no.[12]: see 0x02/0x0030
    r SMS center error recv { 0x0035, reason }
0x03: Phonebook functions
    s Get mem location      { 0x0001, memtype, location, 0 }
                            where memory:
                                     0x01: telephone and SIM phonebook (in one)
                                     0x02: telephone phonebook
                                     0x03: SIM phonebook
                                     0x04: SIM fixdialling-phonebook (?)
                                     0x05: Own numbers
                                     0x07: Dialled numbers
                                     0x08: Missed calls
                                     0x09: Received calls
                                     0x0b: voice mailbox (location not important)
    r mem location recvd    { 0x0002, 0x00,namelen,"name",numlen,"number",groupID, 0x01?,
↪ yearLO, yearHI, month, day, hour, minute, sec. }
                            Note: in 3310 all entries have null name ("feature" of bug ?)
    r mem loc error recvd   { 0x0003, errtype }
                            where errtype:
                                     0x7d: invalid memory type
                                     0x74: empty location ?
                                     0x8d: no PIN
    s Set mem location      { 0x0004, memtype,location,namelen,"Name",numlen,"number",
↪groupID }
    r mem set OK            { 0x0005 }
    r mem set error         { 0x0006, errtype }
                            where errtype: 0x7d: name is too long
    s Mem status request    { 0x0007, memtype }
    r Mem status recvd      { 0x0008, memtype, free, used }
    r Mem status error recv { 0x0009, errtype }
                            where errtype: 0x6f: mem status error
                                           0x7d: invalid memory type
                                           0x8d: waiting for pin
    s Get caller group data { 0x0010, groupID }
    r Get caller group data { 0x0011, groupID, size, "Name", ringtoneID, graphic_on?1:0,␣
↪lenHI, lenLO, OTABitmap (72x14 logo) }
    r Get call.group error  { 0x0012, reason }
                            where reason: 0x7d: invalid location
    s Set caller group data { 0x0013, groupID, size, "Name", ringtoneID, graphic_on?1:0,␣
↪lenHI, lenLO, OTABitmap (72x14 logo) }
    r Set caller group OK   { 0x0014 }
    r Set call.group error  { 0x0015, reason }
                            where reason: 0x7d: invalid location
    s Get speed dial        { 0x0016, index(1-9) }
    r Get speed dial OK     { 0x0017, mem.type, location }
                            where mem.type: 0x02: ME        (== 0 if not stored)
                                            0x03: SIM
                                  location: memory location (== 0 if not stored)
    r Get speed dial error  { 0x0018 }
    s Set speed dial        { 0x0019, index(1-9), mem.type, location }
    r Set speed dial OK     { 0x001a }
```

(continues on next page)

(continued from previous page)

```
    r Set speed dial error  { 0x001b }
0x04: Phone Status
    s Phone status          { 0x0001 }
    r Phone status          { 0x0002, mode, signal str, ???, pwr, batt.level }
                            where mode: 1: registered within the network
                                        2: call in progress
                                        3: waiting for pin
                                        4: powered off
                                  pwr: 1: AC/DC
                                       2: battery
    s Request Phone ID      { 0x0003 }
    r RequestPhone ID       { 0x0004, 0x01,"NOKIA""imei", 0, "model", 0, "prod.code", 0,
↪"HW", 0, "firmware", 0x00, 0x01 }
0x05: Profile settings
    s Set profile feature   { 0x0010, 1, nr, feature, a, 1 }
                            where nr: see 0x05/0x0013
                                  feature: see 0x05/0x0014
                                  a: see 0x05/0x0014
    r Set profile feat. OK  { 0x0011, 1 }
    s Get profile feature   { 0x0013, 1, nr, feature, 1 }
                            where nr is profile number (general=0, silent, meeting,␣
↪outdoor, pager, car, headset=6)
                                  feature: see 0x05/0x0014
    r Get profile feature   { 0x0014, 1, nr, feature, 4, a, b, c, d, 1 }

                             Note: Settings num 0x00 .. 0x09 can be assigned
                             separately to each profile (0x00 .. 0x05), but rest are␣
↪common
                             to all profiles.

                             6110

                             Feature  Description                    Value
                             -------  --------------------          -----------------
                             0x00     keypad notes                  0xff=off,␣
↪0x00=level 1, 0x01=level 2, 0x02=level 3
                             0x01     lights (? only in car profile) 0x00=off, 0x??=on␣
↪(maybe 0x01)
                             0x02     incoming call alert           1=ringing, 2=beep␣
↪once, 3=unknown, 4=off, 5=ring once,
                                                                    6=ascending,␣
↪7=caller groups (see feature #0x08)
                             0x03     ringing tone ID               for original 6110:␣
↪0x12=ring ring, 0x13=low, etc.
                             0x04     ringing volume                level 1 (0x06) -␣
↪level 5 (0x0a)
                             0x05     message alert tone            0=no tone,␣
↪1=standard, 2=special, 3=beep once, 4=ascending
                             0x06     vibration                     0=off, 1=on
                             0x07     warning and game tones        0xff=off, 0x04=on
                             0x08     incoming caller groups        1=family, 2=VIP,␣
↪4=friends, 8=colleagues, 16=other
```

(continues on next page)

(continued from previous page)

```
                            0x09     automatic answer              0x00=off, 0x01=on
                            ------------------------
                            0x16     Anykey answer                 0x00=Off, 0x01=On
                            0x17     ???                           0x00 | 0x01
                            0x18     Memory in use                 0x00=Phone,␣
↪0x01=SIM card
                            0x19     Network selection             0x00=Automatic,␣
↪0x01=Manual
                            0x1a     Automatic redial              0x00=Off, 0x01=On
                            0x1b     ???                           0x00 | 0x01
                            0x1c     ???                           0x00...0x18
                            0x1d     Speed dialling                0x00=Off, 0x01=On
                            0x1e     Own number sending            0x00=Preset,␣
↪0x01=On, 0x02=Off
                            0x1f     Cell info display             0x00=Off, 0x01=On
                            0x21     Language                      0x00=English
                                                                   0x01=Deutsch
                                                                   0x02=Francais
                                                                   0x03=Italiano
                                                                   0x06=Nederlands
                                                                   0x07=Dansk
                                                                   0x08=Svenska
                                                                   0x09=Suomi
                                                                   0x0e=Norsk
                                                                   0x10=Automatic
                            0x26     Reply via same centre         0x00=No, 0x01=Yes
                            0x27     Delivery reports              0x00=No, 0x01=Yes
                            0x28     Hide clock                    0x00=Show clock,␣
↪0x01=Hide clock
                            0x29     Time format                   0x00=24-hour,␣
↪0x01=12-hour
                            0x2a     Selected profile              0x00=General, 0x01..
↪ the rest

                            33x0

                            Feature  Description                  Value
                            -------  -------------------         -----------------
                            0x00     keypad notes                  0xff=off,␣
↪0x00=level 1, 0x01=level 2, 0x02=level 3
                            0x01     incoming call alert           1=ringing, 2=beep␣
↪once, 3=unknown, 4=off, 5=ring once,
                                                                   6=ascending
                            0x02     ringing tone ID
                            0x03     ringing volume                level 1 (0x06) -␣
↪level 5 (0x0a)
                            0x04     message alert tone            0=no tone,␣
↪1=standard, 2=special, 3=beep once, 4=ascending
                            0x05     vibration                     0=off, 1=on,␣
↪2=vibrate first
                            0x06     warning tones                 0xff=off, 0x04=on
                            0x07     screen saver                  1=on, 0=off
```

(continues on next page)

(continued from previous page)

```
                            0x08     Screen saver -> Timeout         0x00=5 sec, 0x01=20␣
↪sec,....
                            0x09     Screen saver -> Screen saver    0x00 ... 0x0d =␣
↪Number of picture image
                            -------------------------
                            0x0a:    ???:
                            ...:     ???:
                            0x15:    ???: Read only?
                            -------------------------
                            0x16:    ???:                            0x00=??? 0x01=???
                            0x17:    Memory in use (Nokia 3330):     0x00=Phone,␣
↪0x01=SIM card
                            0x18:    Network selection:              0x00=Automatic,␣
↪0x01=Manual
                            0x19:    Automatic redial:               0x00=Off, 0x01=On
                            0x1a:    Speed dialling:                 0x00=Off, 0x01=On
                            0x1b:    Own number sending:             0x00=Set by network,
↪ 0x01=On, 0x02=Off
                            0x1c:    Cell info display:              0x00=Off
                            0x1d:    Type of view:                   0x00=Name list,
↪0x01=Name, number,0x02=Large font
                            0x1e:    Language:                       0x00=English
                                                                     0x07=Dansk
                                                                     0x08=Svenska
                                                                     0x09=Suomi
                                                                     0x0c=Turcke
                                                                     0x0e=Norsk
                                                                     0x10=Automatic
                            0x32:    Reboots ME (3330)
                            0x1f:    ???: Read only? (3330)
                            0x20:    Reply via same centre:          0x00=No, 0x01=Yes
                            0x21:    Delivery reports:               0x00=No, 0x01=Yes
                            0x22:    Show/Hide clock:                0x00=Show, 0x01=Hide
                            0x23:    Time format:                    0x00=24-hour,␣
↪0x01=12-hour
                            0x24:    Select profile:                 0x00=General, 0x01 .
↪.. 0x05=rest of them
                            0x25:    ???: Read only? (N3330)
                            0x26:    Confirm SIM service actions:    0x00=Not asked,␣
↪0x01=Asked
                            0x27:    T9 Dictionary:                  0x00=Off,␣
↪0x01=English, 0x0a=Suomi
                            0x28:    Messages -> Character support:  0x00=Automatic,␣
↪0x01=GSM alphabet, 0x02=Unicode
                            0x29:    Startup logo settings:          0x00=Your own␣
↪uploaded logo,0x01=Nokia
                                                                     0x02=Draft HUMAN␣
↪technology(tm),0x03=Itineris
                            0x2a:    ???:                            0x00=??? 0x01=???
                            0x2b:    ???:                            0x00=??? 0x01=???
                            0x2c:    ???: Read only? (N3330)
                            0x2d:    Auto update of date and time:   0x00=Off,
```

(continues on next page)

(continued from previous page)

```
↪0x01=Confirm first,0x02=On

   s Get welcome message   { 0x0016 }
   r Get welcome message   { 0x0017, no.of blocks, { block } * }
                           where block: { id, {blockspecific} }
                                 id: 1: startup logo { y, x, picture (coding?) }
                                     2: welcome note { len, "message" }
                                     3: operator msg { len, "message" }
   s Set welcome message   { 0x0018, no.of blocks, { block } * }
                           where block: see 0x05/0x0017
   r Set welcome OK        { 0x0019, 0x01 }
   s Get profile name      { 0x001a, nr }
                           where nr: see 0x05/0x0013
   r Profile name          { 0x001b, 1, 1, 3, flen, nr, len, {text} }
                           where nr: see 0x05/0x0013
                                 len: text length
                                 flen len + len(nr, len) = len + 2
                           Note: in Nokia 3310 name is in Unicode
   s ???                   { 0x001c }
   r ???                   { 0x001d, 0x93 }
   s Set oplogo            { 0x0030, location, MCC1, MCC2, MNC, lenhi=0x00, lenlo=0x82,␣
↪OTABitmap }
   r Set oplogo OK         { 0x0031 }
   r Set oplogo error      { 0x0032, reason }
                           where reason: 0x7d invalid location
   s Get oplogo            { 0x0033, location }
                           where location: 1 (doesn't seem to matter)
   r Get oplogo            { 0x0034, location, MCC1, MCC2, MNC, lenhi=0x00, lenlo=0x82,␣
↪OTABitmap }
   r Get oplogo error      { 0x0035, reason }
                           where reason: 0x7d invalid location
   s Set ringtone          { 0x0036, location,0x00,0x78, ringtone packed according to␣
↪SM2.0}
   r Set ringtone OK       { 0x0037 }
   r Set ringtone error    { 0x0038, reason }
                           where reason=0x7d, when not supported location
   s Get services settings { 0x0080, setting (2 bytes) }
                           where: setting: 0x02,0x00=Nokia access number 1
                                           0x02,0x01=Operator access number 1
                                           0x01,0x00=Personal bookmark 1 settings (name␣
↪only ?)
                                           0x01,0x01=?
                                           0x02,0x02=?
   r Get services sett.OK  { 0x0081, .... }
   r Get services sett.err { 0x0082, 0x7b }
0x06: Calling line restriction/Call forwarding etc
   r Get call divert       { 0x0001, 0x02, x, 0x00, divtype, 0x02, calltype, y, z, 0x0b,
↪ number, 0x00...0x00, timeout (byte 45) }
   s Set call divert       { 0x0001, 0x03, 0x00, divtype, calltype, 0x01, number(packed␣
↪like in SMS), 0x00 ... 0x00,
                                   length of number (byte 29), 0x00 ... 0x00, timeout␣
↪(byte 52), 0x00, 0x00, 0x00}
```

(continues on next page)

(continued from previous page)

```
                            NOTE: msglen=0x37
                            where timeout:
                              0x00: not set ?
                              0x05: 5 second
                              0x0a: 10 second
                              0x0f: 15 second
                              0x14: 20 second
                              0x19: 25 second
                              0x1e: 30 second
                            where divtype:
                              0x02: all diverts for all call types ?
                                    Found only, when deactivate all diverts for all call␣
↪types (with call type 0x00)
                              0x15: all calls
                              0x43: when busy
                              0x3d: when not answered
                              0x3e: if not reached
                            calltype:
                              0x00: all calls (data, voice, fax)
                              0x0b: voice calls
                              0x0d: fax calla
                              0x19: data calls
    s Deactivate calldiverts{ 0x0001, 0x04, 0x00, divtype, calltype, 0x00 }
                            where divtype, calltype: see above
    r Deactivate calldiverts{ 0x0002, 0x04, 0x00, divtype, 0x02, calltype, data }
    s Get call diverts      { 0x0001, 0x05, 0x00, divtype, calltype, 0x00 }
                            where divtype, calltype: see above
    r Get call diverts ok   { 0x0002, 0x05, 0x00, divtype, 0x02, calltype, data }
                            where divtype, calltype: see above
                      data: { 0x01, 0x00 } - isn't active
                        { 0x02, 0x01, number(packed like in SMS), 0x00, 0x00..., timeout␣
↪}
    r Get prepaid(?) info   { 0x0005, ?,?,?,length,message(packed like in 7bit SMS)}
    r Call diverts active   { 0x0006, ??? }
0x07:
    s ???                   { 0x0022, ? (1&2 sounds OK) }
    r ??? OK                { 0x0023, ?,?,? }
    r ??? error             { 0x0024, reason }
    s ???                   { 0x0025, ??? }
    r ??? OK                { 0x0026, ??? }
    r ??? error             { 0x0027, reason }
0x08: Security codes
    s Change code           { 0x0004, code, "current", 0x00, "new", 0x00 }
                            where code: 1: security code (5 chars)
                                        2: PIN (4 chars)
                                        3: PIN2 (4 chars)
                                        4: PUK (8 chars)
                                        5: PUK2 (8 chars)
    s Status request        { 0x0007, 0x01 }
    r pin recvd             { 0x0008, accepted }
                            where accepted: 0x0c (or 0x06): OK
                                            code: waiting for (0x08/0x0004) code
```

(continues on next page)

(continued from previous page)

```
    s entering code         { 0x000a, code, "code", 0x00 }
                            where code: see 0x08/0x0004
0x09: SIM login
    r login                 { 0x0080 }
    r logout                { 0x0081 }
0x0a: Network status
    s Key duplication on/off{ 0x0044, on? 0x01: 0x02 }
    s get used network      { 0x0070 }
    r network registration  { 0x0071, ?,?,?,length,netstatus,netsel,cellIDH,cellIDL,lacH,
↪lacL,netcode,netcode,netcode }
0x0c: Keys
    s Get key assignments   { 0x0040, 0x01 }
    r Get key assignments   { 0x0041, {key '1'}, 0x00, {key '2'} ... {key '0'}, 0,0,0,
↪{symbols}, 0 }
                            where {key '0'} => ' ', '0'
    s Press key             { 0x0042, press: 0x01; release: 0x02, button, 0x01 }
                            where button: 0x01 - 0x09: 1-9
                                          0x0a: 0
                                          0x0b: #
                                          0x0c: *
                                          0x0d: Power
                                          0x0e: Pick up phone
                                          0x0f: Hang
                                          0x10: Volume +
                                          0x11: Volume -
                                          0x17: Up
                                          0x18: Down
                                          0x19: Menu
                                          0x1a: Names
                                          0x1B onwards: don't know but they do produce
                                               a beep and light up the keypad as if
                                               a key had been pressed.
    r Press key ack         { 0x0043, press/release/error(0x05) }
    s ???                   { 0x0044 }
    r ??? ack               { 0x0045, 0x01 }
0x0d: Status
    r Display               { 0x0050, 0x01, y, x, len, "string"(unicode) }
    s Status request        { 0x0051 }
    r Status                { 0x0052, no. of byte pairs, {byte pair} }
                            where {byte pair}: {cmd, 1:off 2:on}
                            cmd: 1: call in progress
                                 2: ???
                                 3: have unread sms
                                 4: voice call active
                                 5: fax call active
                                 6: data call active
                                 7: key lock active
                                 8: is SMS storage full
    s Display status        { 0x0053, 1:on 2:off }
                            (will send displayed messages with x,y coordinates)
    r Display status ack    { 0x0054, 1 }
0x11: Phone clock & alarm
```

(continues on next page)

(continued from previous page)

```
    s set date and time     { 0x0060, 1,1,7,yearh,yearl,month,mday,hour,min,0x00 }
    r date and time set     { 0x0061 }
    s get date and time     { 0x0062 }
    r date and time recvd   { 0x0063,date_set?,time_set?,?,?,yearh,yearl,month,mday,hour,
↪min,second }
                            where: date_set & time_set==0x01 - set
                                           0x00 - not set, ?,?,yearh,yearl,month,mday,
↪hour,min,second
                                                                  not available in frame
    s set alarm             { 0x006b, 1,32,3,0x02(on-off),hour,min,0x00 }
    r alarm set             { 0x006c }
    s get alarm             { 0x006d }
    r alarm received        { 0x006e,?,?,?,?,alrm(==2:on),hour,min }
0x12: Connect to NBS port (61xx only ?)
    s Send                  {+0x0c, 0x01, UDH header, data}
                            (without 0,1 header  -- for oplogo, cli, ringtone etc upload)
                where: UDH header = 0x06, 0x05, 0x04,destporth,destportl,srcporth,
↪srcportl
0x13: Calendar notes
    s Write calendar note   { 0x0064, 0x01, 0x10, length, type, yearH, yearL, month, day,
↪ hour, timezone,
                              alarm?(alarm yearH, yearL, month, day, hour, timezone): (0,
↪0,0,0,0,0),
                              textlen, "text" }
    r Write cal.note report { 0x0065, return }
                            where return: 0x01: ok
                                          0x73: failure
                       0x81: calendar functions busy. Exit Calendar menu and try again
    s Calendar notes set    { 0x0066... }
    r Calendar note recvd   { 0x0067, 0x01, ?, length, type, yrH,yrL,mon,day,hr,tz,alrm␣
↪yrH,yrL,mon,day,hr,tz,textlen, "text" }
    r Cal.note recvd error  { 0x0067, err }
                            where err: 0x93: not available
                                       (0x01: OK)
                                        other: error
    s Delete cal.note       { 0x0068, location }
    r Del. cal.note report  { 0x0069, err }
                            where err: 0x01: OK
                                       0x93: cannot delete
0x14: SMS funcs
    s Write SMS to SIM      { 0x0004, .... }
    s Mark SMS as read      { 0x0007, 0x02, location, 0x00, 0x64 }
    r SMS message frame rcv { 0x0008,subtype,?,num,?,BCD(smscenter)...} 20->type, 22->
↪status
                            where type: 0x06: delivery report
                                  status: 0x00: delivered
                                          0x30: pending
                                          0x46: failed
                                          0x09: reading failed
                                  subtype: 0x02: invalid mem type
                                           0x07: empty SMS location
                     0x0c: no access to memory (no PIN in card, etc.)
```

(continues on next page)

(continued from previous page)

```
    s Delete SMS message    { 0x000a, 0x02, location }
    r Delete OK             { 0x000b }
    s SMS status request    { 0x0036, 0x64 }
    r SMS status            { 0x0037,?,?,?,?,?,?,msgnumber,unread }
    r SMS status error      { 0x0038 }
0x3f: WAP
    s Enable WAP frames     { 0x0000}
    r Enable WAP frames     { 0x0002, 0x01}

    s  ??                   { 0x0003}
    r  ??                   { 0x0004}

    s Get WAP bookmark      { 0x0006, 0x00, location}
                              where location: 0 - 14
    r Get WAP bookmark      { 0x0007, 0x00, name_len, name(unicode),
                              url_len, url(unicode), 0x01,0x80,0x00[7]}
    r Get WAP bookmark err  { 0x0008, error }
                              where error:
                                0x00(?)invalid position
                                0x01   user inside "Bookmarks" menu. Must leave it
                                0x02   invalid/too high/empty location

    s Set WAP bookmark      { 0x0009, 0xff, 0xff, name_len, name(unicode),
                              url_len, url(unicode), 0x01,0x80,0x00[7] }
                              Note: bookmark is added to the first free location.
    r Set WAP bookmark OK   {+0x01, 0x36, 0x0a, block }
                              where block:
                                0x0a, location_of_just_written_bookmark(?),
                                0x00, next_free_location(?)
    r Set WAP bookmark err  {+0x01, 0x36, 0x0b, error }
                              where error:
                               0x04 - memory is full
                               0x01 - we are in the bookmark menu
                               0x00 - unknown reason for now ;(

?   s Delete WAP bookmark   { 0x000c, 0x00, location }
                              where: location = 0-14
?   r Delete WAR bookmark OK{ 0x000d }
?   r Delete WAPbookmark err{ 0x000e, 0x02 }

    s ??                    { 0x000F}
    r ??                    { 0x0010, 0x00}

    s Get WAP settings 1    { 0x0015, location}
                            where location: 0x00 - 0x05
    r Get WAP settings 1 OK { 0x0016, title length, title (Unicode), URL length,␣
↪URL(Unicode),con_type, ???[6 bytes],location, ???[5 bytes],security,...}
                            where:
                              con_type: 0x00 - temporary
                                        0x01 - continuous
                              location: when use "Get WAP settings 2 frame", must give it
                              security: 0x00 = no, 0x01 = yes
```

(continues on next page)

(continued from previous page)

```
    r Get WAP settings 1 err{ 0x0017, error }
                              where error:
                                0x01   user inside "Settings" menu. Must leave it
                                0x02   invalid/too high/empty location

    s Get WAP settings 2    { 0x001b, location}
                            where location: 0x00 - 0x1d (you get it in "Get WAP settings␣
↪1" frame)
    r Get WAP settings 2 OK { 0x001c, 0x01, type, frame...}
                            where type : 0x00 - SMS bearer
                                              frame:
                                                service_num_len, service_num (Unicode),␣
↪server_num_len, server_num(Unicode)
                                           0x01 - data bearer
                                             frame:
                                                auth, call_type, call_speed, ?, IP len, IP␣
↪(Unicode), dialup len, dialup (Unicode),
                                                user len, user (Unicode), password len,␣
↪password (Unicode)
                                                where auth: 0x00 - normal, 0x01 - secure
                                                      call_type: 0x00 - analogue, 0x01 -␣
↪ISDN
                                                      call_speed: 0x00 - 9600, 0x01 - 14400
                      0x02 - USSD bearer
                        frame: type, service number len/IP len,service num (Unicode)/IP␣
↪(Unicode),service code len,
                               service code (Unicode)
                          where type: 0x01 - service number, 0x00 - IP
    r Get WAP settings 2 err{ 0x001d,error}
                            where: error=0x05
0x40: Security commands
    s ???                   {+0x00, 0x00, 0x07, 0x11, 0x00, 0x10, 0x00, 0x00}
                            This frame hangs phone (N3310 4.02). Meaning unknown !
    s Open simlock 1        { 0x02, 0x03, 0x1f, 0x11, 0x01, 0x01, 0x10, 0x00}
    r Open simlock 1        { 0x02 }
    s ???(N6150)            { 0x08, 0x00 }
    r ???(N6150)            { 0x08 }
    s Enable extended cmds  { 0x64, cmd }
                            where cmd: 0x00: off
                                       0x01: on
                         0x02: enter service mode ?
                                       0x03: reset (doesn't ask for PIN again)
                                       0x04: reset (PIN is requested)
                                             In 5110 makes reset without PIN
                                       0x06: CONTACT SERVICE!!! Don't try it!
    s Reset phone settings  { 0x65, value, 0x00 }
                            where value: 0x08 - reset UI (User Interface) settings
                             0x38 - reset UI, SCM and call counters
                                        0x40 - reset test 36 in netmonitor
    r Reset phone settings  { 0x65, 0x00 }
    s Get IMEI              { 0x66 }
    r Get IMEI              { 0x66, 0x01, IMEI, 0x00}
```

(continues on next page)

(continued from previous page)

```
    s (ACD Readings)?(N6150 { 0x68 }
    r (ACD Readings)?(N6150 { 0x68, ... }
    s Get Product Profile
      Settings              { 0x6a}
    r Get Product Profile
      Settings              { 0x6a, 4bytes with Product Profile Settings }
    s Set Product Profile
      Settings              { 0x6b, 4bytes with Product Profile Settings }
    r Set Product Profile
      Settings OK ?         { 0x6b }
    s Get code              { 0x6e, code }
                            where code: see 0x08/0x0004 (only sec.code is allowed)
    r Get code              { 0x6e, code, allowed, allowed? (sec code (text)) }
                            where code: see 0x08/0x0004
                                  allowed: 0: no
                                           1: yes
    s Set code              { 0x6f, code, sec code(text), 0x00 }
                            where code: see 0x08/0x0004
    s Start monitoring      { 0x70, block }
                            where block(N6150):
                              0x7f,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
                              0xff,0xff,0xff,0xff,0xff,0xf9,0x76,0x65,0x20,0x00,
                              0x00,0x00,0x00,0x00,0x18,0x26,0x15,0x7d,0x0a,0x00,
                              0xf5,0x82,0x7f,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
                              0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xf0,0x77,0x80,
                              0x77,0x80,0xf2,0x82,0x20,0x20,0x20,0x20,0x20,0x20,
                              0x20,0x20,0x20,0x20,0x20
                            This block enables probably all possible monitored␣
↪parameters.
                            After it phone sends 0x00 message type values
    s Break monitoring      { 0x71 }
    r Break monitoring      { 0x71 }
?   s ????                  { 0x74, 0x01, 0x01, 0x0e }
?   r ????                  { 0x74 }
    s Call commands         { 0x7c, block }
                            where where: command, (values)
                        command: 0x01
                        values: number(ASCII), 0x00 - makes voice call
                    command: 0x02 - answer call
                    command: 0x03 - release call
    r Call commands         { 0x7c, command }
    s Netmonitor            { 0x7e, field }
                            where: field: 00: next
                                          F0: reset
                                          F1: off
                                          F2: field test menus
                                          F3: developer menus
    s Open simlock 2        { 0x81, 0x01, locknumber, 0x10, 0x10, 0x10, 0x10, 0x10 }
                            Note: sets simlock type to factory?
                where locknumber: 1,2,4,8
    s Open simlock 2        { 0x81, 0x01, locknumber }
                where locknumber: 1,2,4,8
```

(continues on next page)

(continued from previous page)

```
   s Close simlock         { 0x82, 0x01, locknumber, 0x00, 0x00, locksinfo(lock1,4,2,3),
↪ 0x00 }
                           where locknumber: 1,2,4,8
   r Close simlock         { 0x82, the rest like in 0x40/0x8a }
   s Get simlock info      { 0x8a, 0x00}
   r Get simlock info      { 0x8a, 0x00, 0x01, lockstype, locksclosed, 0x00, 0x00,␣
↪locksinfo(lock1,4,2,3), counter1,counter2,counter4,counter4,0x00 }
                           where: lockstype:   bit1,bit2,bit3,bit4 - if set, selected␣
↪lock is user lock
                                  locksclosed: bit1,bit2,bit3,bit4 - if set, selected␣
↪lock is closed
                                  counter1 - counter4: counters for locks
   s Set downloaded OpName { 0x8b, 0x00, MCC1, MCC2, MNC, Name, 0x00 }
   r SetdownloadedOpNameOK?{ 0x8b, 0x00, 0x01 }
   s Get downloaded OpName { 0x8c, 0x00 }
   r Get downloaded OpName { 0x8c, 0x00, 0x01, MCC1, MCC2, MNC, Name, 0x00,...}
   s Buzzer pitch          { 0x8f, volume, hzLO, hzHI }
                           if volume and hz is 0, it's off
   r Buzzer pitch          { 0x8f}
   s ACD Readings ?        { 0x91, parameter?(0x02,0x03,0x04,0x05,0x07) }
   r ACD Readings ?        { 0x91, parameter?, value? }
   s Sleep mode test       { 0x92, 0x00, 0x00, howlong(2 bytes), enable }
                           where: enable == 0x01 - enable after test
                                  0x00 - don't enable after test
                howlong (ms) = 0x07, 0xd0 = 2000
   s ???(N6150)            { 0x98, 0x00 }
   r ???(N6150)            { 0x98, 0x00, 0x04 }
   s Get bin ringtone      { 0x9e, location }
                           where: location=0,1,etc.
   r Get bin ringtone      { 0x9e, location, error, contents... }
                           where location=0,1,etc.
                                 error=0x0a, ringtone NOT available
                                       0x00, OK
   s Set bin ringtone      { 0xa0, location, 0x00, contenst... }
                           where: location=0,1,etc.
   r Set bin ringtone      { 0xa0, location, error }
                             where location=0,1,etc.
                                   error=0x0a, ringtone NOT set
                                         0x00, ringtone set OK
   r Get MSid              { 0xb5, 0x01, 0x2f, msid, 0x25 }
   s Get info about phone  { 0xc8, 0x01 }
   r Get info about phone  { 0xc8, 0x01, 0x00, "V ", "firmware", 0x0a, "firmware date",␣
↪0x0a, "model", 0x0a, "(c) NMP.", 0x00 }
   s Get MCU SW Checksum   { 0xc8, 0x02 }
   r Get MCU SW Checksum   { 0xc8, 0x02, 0x00, checksum (4 bytes),0x00 }
   s DPS External SW       { 0xc7, 0x03 }
   r DSP External SW       { 0xc7, 0x03, 0x00, string,0x00 }
   s Get HW                { 0xc8, 0x05 }
   r Get HW                { 0xc8, 0x05, 0x00, HW version (4 bytes), 0x00 }
   s Get "Made" Date       { 0xc8, 0x05 }
   r Get "Made" Date       { 0xc8, 0x05, 0x00, date(4 bytes), 0x00 }
   s Get DSP Internal SW   { 0xc8, 0x09 }
```

(continues on next page)

(continued from previous page)

```
   r Get DSP Internal SW   { 0xc8, 0x09, 0x00, version (1 bytes), 0x00 }
   s Get PCI version       { 0xc8, 0x0b }
   r Get PCI version       { 0xc8, 0x0b, 0x00, version, 0x00 }
   s Get system ASIC       { 0xc8, 0x0c }
   r Get system ASIC       { 0xc8, 0x0c, 0x00, string, 0x00 }
   s Get COBBA             { 0xc8, 0x0d }
   r Get COBBA             { 0xc8, 0x0d, 0x00, string, 0x00 }
   s Get PLUSSA            { 0xc8, 0x0e }
   r Get PLUSSA            { 0xc8, 0x0e, available, 0x00 }
                           where available: 0x01: not available
   s Get CCONT             { 0xc8, 0x0f }
   r Get CCONT             { 0xc8, 0x0f, available, 0x00 }
                           where available: 0x01: not available
   s Get PPM version       { 0xc8, 0x10 }
   r Get PPM version       { 0xc8, 0x10, 0x00, "V ", "firmware", 0x0a, "firmware date",␣
↪0x0a, "model", 0x0a, "(c) NMP.", 0x00 }
   s Get PPM info          { 0xc8, 0x12 }
   r Get PPM info          { 0xc8, 0x12, 0x00, PPM version ("B", "C", etc.), 0x00 }
   s Set HW version        { 0xc9, 0x05, version, 0x00 }
   s Get Product Code      { 0xca, 0x01 }
   r Get Product Code      { 0xca, 0x01, 0x00, number, 0x00 }
   s Get Order Number      { 0xca, 0x02 }
   r Get Order Number      { 0xca, 0x02, 0x00, string, 0x00 }
   s Get Prod.Ser.Number   { 0xca, 0x03 }
   r Get Prod.Ser.Number   { 0xca, 0x03, 0x00, number, 0x00 }
   s Get Basic Prod.Code   { 0xca, 0x04 }
   r Get Basic Prod.Code   { 0xca, 0x04, 0x00, number, 0x00 }
   s Set Product Code      { 0xcb, 0x01, product code, 0x00 }
   s Set Order Number      { 0xcb, 0x02, number, 0x00 }
   s Set Prod.Ser.Number   { 0xcb, 0x03, number, 0x00 }
   s Get (original ?)IMEI  { 0xcc, 0x01 }
   r Get (original ?)IMEI  { 0xcc, 0x01, IMEI, 0x00 }
   s Get Manufacture Month { 0xcc, 0x02 }
   r Get Manufacture Month { 0xcc, 0x02, 0x00, string, 0x00 }
   s Get Purchare date     { 0xcc, 0x04 }
   r Get Purchare date     { 0xcc, 0x04, 0x00, string, 0x00 }
   s Set "Made" date       { 0xcd, 0x02, string, 0x00 }
   s Make "all" phone tests{ 0xce,0x1d,0xfe,0x23,0x00,0x00}
   s Make one phone test   { 0xce,0x1d,num1,num2,num3,num4}
                           Where num1-num4: 0x02,0x00,0x00,0x00;
                                            0x04,0x00,0x00,0x00;
                                            0x08,0x00,0x00,0x00;
                                            0x10,0x00,0x00,0x00;
                                            0x20,0x00,0x00,0x00;
                                            0x40,0x00,0x00,0x00;
                                            0x80,0x00,0x00,0x00;
                                            0x00,0x01,0x00,0x00;
                                            0x00,0x02,0x00,0x00;
                                            0x00,0x04,0x00,0x00; - "Power off"
                                              No test for "Security data"
                                            0x00,0x10,0x00,0x00;
                                            0x00,0x20,0x00,0x00;
```

(continues on next page)

(continued from previous page)

```
                                          0x00,0x40,0x00,0x00;
                                          0x00,0x80,0x00,0x00;
                                          0x00,0x00,0x01,0x00;
                                          ....
                                          0x00,0x00,0x10,0x00;
    s Result of phone tests { 0xcf }
    r Result of phone tests { 0xcf, number of tests, results of next tests }
    s ???                   { 0xd1 }
    r ???(N5110)            { 0xd1, 0x00, 0x1d, 0x00, 0x01, 0x08, 0x00 }
    s LCD Test              { 0xd3, value }
                            where value: 0x03, 0x02 - 1'st test
                                         0x03, 0x01 - 2'nd test
                                         0x02, 0x03 - clears screen
    s ACD Readings(N6150)?  { 0xd4, 0x02, 0x00, 0x02, 0x00, 0x0e, 0x01}
    r ACD Readings(N6150)?  { 0xd4, 0x02, 0x00, 0x02, 0x00, 0x0e, 0x01, ?}
    s Get EEPROM            { 0xd4, 0x02, 0x00, 0xa0, locationLo, locationHi, numofbytes␣
↪}
                            where: numofbytes - how many bytes to read
                Note: Works ONLY in MBUS
    r Get EEPROM            { 0xd4, 0x02, 0x00, 0xa0, locationLo, locationHi, numofbytes,
↪ contest... }
                            where numofbytes - how many bytes available
                          contest - bytes with contests (if numofbytes != 0)
0x41: Snake game ?
0x47:
    s Get Picture Image     { 0x0001, location }
    r Get Picture Image     when contains sender number
                            { 0x0002, location, number(like in SMS), 0x00, len, text,␣
↪0x00, width, height, 0x01, bitmap }
                NOTE:
                              Supports only 0x81 and 0x91 coding (NOT alphanumeric␣
↪numbers!)
                              in sender without sender number
                            { 0x0002, location, 0x00, 0x00, 0x00, len, text, 0x00, width,
↪ height, 0x01, bitmap }
    s Set Picture Image     { 0x0003, frame...}
                            where frame: see 0x47/0x0002
    r Get/Set PictureImageOK{ 0x0004 }
    r Set Picture Image err { 0x0005, error? }
                            where error=0x74 - wrong location ?
0x64:
    s Phone ID request      { 0x0010 }
    r Phone ID recvd        { 0x0011, "NOKIA", "imei", 0, "model", 0, "prod.code", 0, "HW
↪", 0, "firmware", magic bytes x 4 ... }
    s Accessory connection  { 0x0012, 16x0x00, 'NOKIA&NOKIA accessory', 3x0x00 } (45␣
↪bytes)
0x7f: Acknowledge(FBUS/IRDA){+type, seq }
      Acknowledge(MBUS)...
0xd0:
    s Power on message seq1 {+04 }
    r Power on message seq1 {+05 }
0xd1:
```

(continues on next page)

(continued from previous page)

```
    s Get HW&SW version     { 0x0003, 0x00 }
0xd2:
    r Get HW&SW version     { 0x0003 "V " "firmware\n" "firmware date\n"
                              "model\n" "(c) NMP." }
0xda: ? (during playing 2 player snake)
0xf0:
    s Send RLP frame        {+0x00, 0xd9, ... }
0xf4: Power on message seq 2
```

13.5 Nokia 6510

Assembled by Markus Plail <plail@web.de> Marcin Wiacek <Marcin@MWiacek.com> <tibor.harsszegi@essnet.se> ... and other members of gnokii mailing list and authors of some WWW pages.

Heavily based on nk7110.txt.

The data provided is for information purposes only. Some of the frames might be hazardous to your phone. Be careful!!! We do not take any responsibility or liability for damages, etc.

Note: this information isn't (and can't be) complete. If you know anything about features not listed here or you noticed a bug in this list, please notify us via e-mail. Thank you.

Document describing frames used in GSM Nokia 6510 and derivatives (?)

Correct format is FBUS version 2/Infrared/MBUS version 2 (see nokia.txt for protocol details):

```
0x00: Connect to NBS port ?
    r Set ringtone          {+....,ringtone packed according to SM2.0}

0x01 COMMUNICATION
     switch (message[3]) {
        case  0x02:
        dprintf("Call established, remote phone is ringing.\n");
        dprintf("Call ID: %i\n", message[4]);
        break;
    case 0x03:
        dprintf("Call complete.\n");
        dprintf("Call ID: %i\n", message[4]);
        dprintf("Call Mode: %i\n", message[5]);
        dummy = malloc(message[6] + 1);
        DecodeUnicode(dummy, message + 7, message[6]);
        dprintf("Number: %s\n", dummy);
        break;
    case 0x04:
        dprintf("Hangup!\n");
        dprintf("Call ID: %i\n", message[4]);
        dprintf("Cause Type: %i\n", message[5]);
        dprintf("Cause ID: %i\n", message[6]);
        break;
    case 0x05:
```

(continues on next page)

(continued from previous page)

```
        dprintf("Incoming call:\n");
        dprintf("Call ID: %i\n", message[4]);
        dprintf("Call Mode: %i\n", message[5]);
        dummy = malloc(message[6] + 1);
        DecodeUnicode(dummy, message + 7, message[6]);
        dprintf("From: %s\n", dummy);
        break;
    case 0x07:
        dprintf("Call answer initiated.\n");
        dprintf("Call ID: %i\n", message[4]);
        break;
    case 0x09:
        dprintf("Call released.\n");
        dprintf("Call ID: %i\n", message[4]);
        break;
    case 0x0a:
        dprintf("Call is being released.\n");
        dprintf("Call ID: %i\n", message[4]);
        break;
    case 0x0b:
        /* No idea what this is about! */
        break;
    case 0x0c:
        if (message[4] == 0x01)
            dprintf("Audio enabled\n");
        else
            dprintf("Audio disabled\n");
        break;
    case 0x53:
        dprintf("Outgoing call:\n");
        dprintf("Call ID: %i\n", message[4]);
        dprintf("Call Mode: %i\n", message[5]);
        dummy = malloc(message[6] + 1);
        DecodeUnicode(dummy, message + 7, message[6]);
        dprintf("To: %s\n", dummy);
        break;


0x02: SMS HANDLING
    s Send SMS              { 0x02, 0x00, 0x00, 0x00, 0x55, 0x55,
                              0x01 (1 big block), 0x02 (submit), length (big block),
                  type, reference, PID, DCS, 0x00, # blocks,
                  blocks... }
    r Send SMS              { 0x03, 0x00, 0x01, 0x0c, 0x08, 0x00, 0x00, 0xdb, 0x55, 0x55,
↪ 0x00 }

    s Get SMSC              { 0x14, 0x01, 0x00 }
    r Get SMSC              { 0x15, format,  0x01, 0x0b, 0x28, # of SMSC, 0xf8, 0x00,␣
↪validity, 0x55
                              #blocks,
                              blocks ...}
```

(continues on next page)

(continued from previous page)

```
0x03: PHONEBOOK HANDLING

    s Get memory status     { 0x03, 0x01, memory type, 0x55, 0x55, 0x55, 0x00}
                            where: memory type - see 0x03/0x07
    r Get memory status     { 0x04, 0x00, location, 0x00[7], 0x01, 0x10, 0x00, 0x00,␣
↪0x0c,
                                    total_low, total_high, used_low, used_high, 0x01,␣
↪0x00, 0x00}

    s Read memory           { 0x07, 0x01, 0x01, 0x00, 0x01, 0x02, memory type,
                        0x00, 0x00, 0x00, 0x00, location_low, location_high, 0x00, 0x00};

                            where MT: memory type
                                     0x01: (256) Dialled numbers
                                     0x02: (512) Missed calls
                                     0x03: (768) Received calls
                                     0x05: (500) telephone phonebook
                                     0x06: (160) SIM phonebook
                                     0x07: (10/0)
                                     0x08: (1/0)
                                     0x09: (4) voice mailbox
                                     0x0e: (10) speed dials
                                     0x10: (5) caller groups

    r Read memory           { 0x08, 0x00, 0x01,
                                    code, 0x00, 0x00, z, xH, xL, yH, yL, 0x00[7], no.of␣
↪blocks, { block } * }

                            where if code==0x0f && xH==0x34 - phonebook location not␣
↪found
                              y: location
                              z: generic block size
                              block: {id, 0, 0, blocksize, block no.,
                                      {contents}, 0x00}
                                id: 0x04 pointer to another memory location { 0xff?, yH,␣
↪yL, xL,0x00[3] }
                                     0x07 name {len, (unicode)},
                                     0x08 email
                                     0x09 postal
                                     0x0a note {len, (unicode)}
                                     0x0b number {type, 0x00[3], len, (unicode)}
                                     0x0c ringtone {ringtone no., 0, 0}
                    0x13 date for a called list (DC, RC, etc.)
                                     0x1b caller group graphic {width, height, 0, 0
↪{bitmap}}
                                     0x1c caller group graphic on? {(1: yes, 0: no), 0, 0}
                                     0x1e caller group number {number, 0, 0}
                                   type: 0x0a: General,
                                         0x03: Mobile (office ?),
                                         0x06: Work,
                                         0x04: Fax,
                                         0x02: Home (mobile ?)
```

(continues on next page)

(continued from previous page)

```
    s Set mem location      { 0x0b, 0x00, 0x01, 0x01, 0x00, 0x00, z,
                                0x02, memory type, yH, yL, 0x00[7],
                                no.of blocks, { block }[no.of blocks] }
    r Set mem location      { 0x0c, 0?, 1?, code, 0?, 0?, z?, 0?, 0?,
                                yH, yL, xL }
                            where code:
                                    0x3d - wrong entry type

0x08: SECURITY

    s Get status            { 0x11, 0x00 }
    r Get status            { 0x12, status, }
                            where status:
                            0x01: waiting for Security Code
                            0x07:
                            0x02: waiting for PIN
                            0x03: waiting for PUK
                            0x05: PIN ok, SIM ok
                            0x06: No input status
                            0x16: No SIM
                            0x1A: SIM rejected!

    s Enter PIN             { 0x07, 0x02, code, 0x00}
    r Enter PIN             { return code, reason }
                            where:
                            return code: 0x08 = success
                                         0x09 = failure
                            reason: 0x06 = PIN wrong

0x0a: NETSTATUS

    s Get Info              { 0x00, 0x00 }
    r Get Info              { 0x01, 0x00, # blocks,
                              0x00, length, 0x00, 0x02, status, length, operator name␣
↪(unicode),
                              0x09, length, LAC, LAC, 0x00, 0x00, CellID, CellID,␣
↪NetworkCode (3 octets), ... }

    s Get RF Level          { 0x0b, 0x00, 0x02, 0x00, 0x00, 0x00 }
    r GET RF Level          { 0x0c, 0x00, 0x01, 0x04, 0x04, level, 0x5f }

    s Get operator logo     { 0x23, 0x00, 0x00, 0x55, 0x55, 0x55 }
    r Get operator logo     { 0x24, 0x00, 0x01, 0x00, 0x00, 0x00,
                              0x02, 0x0c, 0x08, netcode (3 octets), 0x02, 0x00, 0x00,
                  0x1a, size, width, height, logo size (2 octets), logo size (2 octets),␣
↪logo }

0x10: SUBSCRIBE

    s Subscribe Channel     { 0x10, # channels, message types... }
```

(continues on next page)

(continued from previous page)

```
0x13 CALENDAR

    s Add meeting note      { 0x01, body like in subtype 0x1a...}
    r Add meeting note      { 0x02, location (2 bytes), status (2 bytes)}

    s Add call note         { 0x03, body like in subtype 0x1a...}
    r Add call note         { 0x04, location (2 bytes), status (2 bytes)}

    s Add birthday note     { 0x05, body like in subtype 0x1a...}
    r Add birthday note     { 0x06, location (2 bytes), status (2 bytes)}

    s Add reminder note     { 0x07, body like in subtype 0x1a...}
    r Add reminder note     { 0x08, location (2 bytes), status (2 bytes)}

    s Delete calendar note  { 0x0b, location (2 bytes) }
    r Delete calendar note  { 0x0c, location (2 bytes), ?, ?, ?, ? }

    s Get calendar note     { 0x19, location (2 bytes) }
    r Calendar note recvd   { 0x1a, location (2 bytes), entry type, 0x00, year (2 bytes),
↪ Month, Day, block}
                            where: entry type - 0x01 - Meeting, 0x02 - Call, 0x04 -␣
↪Birthday, 0x08 - Reminder
                                    block: for Meeting:{hour,minute,alarm (two bytes),
↪recurrence (two bytes),len,0x00,string(unicode)}
                                           where alarm=Number of minutes before the time␣
↪of the meeting
                                                   that the alarm should be triggered:
                                                   For meetings with "No alarm"=0xFFFF (-
↪1).
                                                   For "On time"=0x0000
                                                   half an hour=0x001E, and so on.
                                                 Recurrence=in hours, between future␣
↪occurrences of this meeting.
                                                   If there is no repeat, this value is␣
↪0x0000. The special value 0xffff
                                                   means 1 Year!
                                           for Call:{Hour,Minute,Alarm (as above),
↪Recurrence (as above),namelen,numberlen,
                                                     name(unicode),number(unicode)}
                                           for Reminder:{Recurrence (as above),len,0x00,
↪string(unicode)}
                                           for Birthday:{byte1,byte2,alarm(4 bytes),
↪yearofbirth,alarmtype,len,string(unicode)}
                                                   byte1 and byte2 may vary (???).␣
↪Usually are 0x00 both (but not always)
                                                   In Birthday, the Year in the common␣
↪part, usually contains a strange year.
                                                   So, don't consider it as Year of␣
↪note, neither year of BirthDay (for Year of
                                                   Birthday use the value described␣
↪below).
                                           where alarm=32-bit integer that is the number␣
```

(continues on next page)

(continued from previous page)

```
↪of seconds between the desired
                                                    alarm time and 11:59:58pm on the␣
↪birthday.For "No Alarm", the value is
                                                    0x0000FFFF (65535).
                                                  YearOfBirth=used instead of the one in␣
↪the common part of the entry (see above)
                                                    but only when reading birthday entries.
↪ For storing entries, this field does
                                                    not exist.
                                                  AlarmType: 0x00 - Tone, 0x01 - Silent

?   s???                    { 0x0021 }
?   r???                    { 0x0022, 0x5A, 0x00 }
?   s???                    { 0x0025 }
?   r???                    { 0x0026, 0x04, 0x00 }
?   s                       { 0x0029 }
?   r                       { 0x002A, 0x04, 0x00 }

    s Get first free pos    { 0x0031 }
    r Get first free pos    { 0x0032, location (2bytes) }

    s Get notes info        { 0x003a, 0xFF, 0xFE}
    r Get notes info        { 0x003b, how many notes used (2 bytes), 0x01, 0x07, { two␣
↪bytes with location for each note} *}

    s Get first free pos    { 0x0031 }
    r Get first free pos    { 0x0032, location (2bytes) }
    s Get notes info        { 0x003a, 0xFF, 0xFE}
    r Get notes info        { 0x003b, how many notes used (2 bytes), 0x01, 0x07, { two␣
↪bytes with location for each note} *}

?   s Get calendar note??   { 0x003E, location (2 bytes) }
?   r Get calendar note??   { 0x003F, location (2bytes), ... }

0x14: FOLDER/PICTURE SMS HANDLING

    s Get SMS Status        { 0x08, 0x00, 0x01 }
    r Get SMS Status        { 0x09, 0x00, #blocks,
                              type, length, blocknumber,
                              a (2 octets), b (2 octets), c (2 octets), 0x00, 0x55 ,
                              type, length, blocknumber,
                              d (2 octets), e (2 octets), f (2 octets), 0x01, 0x55 }

                              where:
                              a - max. number of messages in phone memory
                              b - Number of used messages in phone memory. These
                                are messages manually moved from the other folders.
                                Picture messages are saved here.
                              c - Number of unread messages in phone memory. Probably
                                only smart messages.
                              d - max. number of messages on SIM
                              e - Number of used messages in SIM memory. These are
```

(continues on next page)

(continued from previous page)

```
                              either received messages or saved into Outbox/Inbox.
                              Note that you *can't* save message into this memory
                              using 'Move' option. Picture messages are not here.
                            f - Number of unread messages in SIM memory



   s Get SMS from folder   { 0x02, memory, folderID, location, location, 0x01, 0x00}
                           where:
               memory - 0x01 for SIM, 0x02 for phone (SIM only for IN/OUTBOX
                           folderID - see 0x14/0x017B
   r Get SMS from folder   { 0x03, 0x00, 0x01, memory, folderID, locationH, locationL,␣
↪0x55, 0x55, 0x55,
                             0x01 (on big block), type, length of big block,
                 [date/time1], [date/time2], # blocks,
                 type, length, data...
                 ... }

   s Delete SMS            { 0x04, memory, folderID, location, location, 0x0F, 0x55 }
   r Delete SMS            { 0x05 }

   s Get folder status     { 0x0c, memory, folderID, 0x0F, 0x55, 0x55, 0x55, 0x55}
                           where: folderID - see 0x14/0x017B
   r Get folder status     { 0x0d, 0x00, length, number of entries (2 bytes),
               entry1number (2 bytes), entry2number(2 bytes), ..., 0x55[]}

   s Get message info      { 0x0e, memory, folderID, location, location, 0x55, 0x55 }
   r Get message info      { 0x0f, 0x00, 0x01, 0x00, 0x50, memory, type, 0x00, location,
↪ FolderID, status

                           where: type = 0x00 - MT
                                         0x01 - delivery report
                                         0x02 - MO
                                         0x80 - picture message
                           where: status=0x01 - reveived/read
                     0x03 - received/unread
                     0x05 - stored/sent
                     0x07 - stored/not sent

   s Get folder names      { 0x12, 0x00, 0x00}
   r Get folder names      { 0x13, 0x00, number of strings, 0x01, 0x28, folderID,␣
↪length, 0x00, name1, 0x00,
               0x55[40-length(name1)], 01 28, folderID, length, 0x00, name2, 0x00,␣
↪0x55[dito] ... }
                              where: folderID = 0x02 - Inbox
                                                0x03 - Outbox
                                                0x04 - Archive
                                           0x05 - Templates
                                           0x06 - first "My folders"
                                           0x07 - second "My folders"
                                           0x08 - third -"-
                                           and so on
```

(continues on next page)

(continued from previous page)

```
0x15:
    s ???                   {+0x00, 0x06, 0x00, 0x01, 0x01, 0x00 }
    r ???                   {+0x06, ',',  0x00, 'd',  0x00, 0x00 }
    s ???                   {+0x00, 0x06, 0x00, 0x02, 0x00, 0x00 }
    r ???                   {+0x06, '.',  0x00, 'e',  ?, ? }

0x17: BATTERY

    s Get battery level     { 0x0a, 0x02, 0x00 }
    r Get battery level     { 0x0b, 0x01, 0x01, 0x16, level, 0x07, 0x05 }
                            where: level: 1-7 (as in phone display)

0x19: CLOCK

    s Get ????              {0x01,...}
    r Get ????              {0x02,...}

    s Get date              { 0x0a, 0x00, 0x00 }
    r Get date              { 0x0b, 0x00, 0x02 (blocks),
                              0x01 (type), 0x0c (length), 0x01, 0x03, year (2 octets),␣
↪month, day, hour, minute, second, 0x00,
                              0x04, 0x04, 0x01, 0x00 }

    s Get ????              {0x0c, 0x00, 0x00}
    r Get ????              {0x0d..}
    s Get ????              {0x11,...}
    r Get ????              {0x12,...}

0x1b: IDENTIFY

    s Get IMEI              {+0x00, 0x01, 0x01, 0x00, 'A', 0x00, 0x00, 0x00 }
    r Get IMEI              {                   0x01, 0x00, 0x01, 'A', 0x14, 0x00, 0x10,
↪{IMEI(ASCII)}, 0x00 }
    s Get IMEI              {+0x00, 0x03, 0x01, 0x00, 'A', 0x00, 0x00, 0x00 }
    r Get IMEI              {                   0x01, 0x00, 0x01, 'A', 0x14, 0x00, 0x10,
↪{IMEI(ASCII)}, 0x00 }
    s Get ???               {+0x00, 0x03, 0x00, 0x00, 'D',  0x00, 0x00, 0x00 }
    r Get ???               {                   0x01, 0x02, 0x00 }

    s Get HW version        {+0x00, 0x03, 0x02, 0x07, 0x00, 0x02 }
    r Get HW version        {                   0x08, 0x00, 0x01, 'I', 0x0c, 0x00, 0x05,␣
↪HW(4 bytes), 0x00, 0x00, 0x00, 0x00 }

    s get HW&SW version     { 0x07, 0x00, 0x01 }
    r get HW&SW version     { 0x08, 0x00, 0x01, 0x58, 0x29, 0x00, 0x22, "V " "firmware\n
↪" "firmware date\n"
                              "model\n" "(c) NMP.", 0x0a, 0x43, 0x00, 0x00, 0x00 }

    s Get product code      {+0x00, 0x03, 0x04, 0x0b, 0x00, 0x02 }
    r Get product code      {                   0x0c, 0x00, 0x01, 'N', 0x0c, 0x00, 0x08,␣
↪code(7 bytes), 0x00 }
```

(continues on next page)

(continued from previous page)

```
    s ???                   {+00 |03 |05 |0b |00 |20}
    r ???                   {+03 |2b+|05 |0c |00 |01 |52R|0c |00 |08 |00 |00 |00 |00 |00␣
↪|00 | 00 |00}
    s Get ???               {+00 |03 |06 |0b |00 |01}
    r Get ???               {+03 |2b+|06 |0c |00 |01 |4dM|10 |00 |0a␣
↪|53S|54T|41A|344|355|399|311|355|377|00 |00 |00}

Sending frame 0x1b / 0x0006
00 |03 |07 |0b |00 |ff                                           .....
Received frame 0x1b / 0x0072
03 |2b+|07 |0c |00 |08 |4dM|10 |00 |0a |53S|54T|41A|344|355|39 .+....M...STA459
311|355|377|00 |00 |00 |4eN|0c |00 |08 |300|355|300|377|355|32 157...N...050752
300|00 |50P|0c |00 |08 |00 |00 |00 |00 |00 |00 |00 |4fO|0c 0.P..........O.
00 |08 |300|355|300|333|366|366|366|00 |51Q|0c |00 |06 |00 |00 ..0503666.Q.....
00 |00 |00 |00 |00 |00 |52R|0c |00 |08 |00 |00 |00 |00 |00 |00 ......R.........
00 |00 |53S|0c |00 |5b[|00 |00 |00 |0d |00 |00 |00 |00 |59Y|14 ..S..[........Y.
00 |10 |00 |00 |00 |00 |00 |00 |00 |00 |00 |00 |00 |00 |00 |00 ................
00 |00                                                         ..
Sending frame 0x1b / 0x0006
00 |03 |08 |07 |01 |ff                                           .....
Received frame 0x1b / 0x00ae
03 |2b+|08 |08 |00 |09 |48H|28(|00 |20 |56V|20 |300|344|2e.|30 .+....H(. V 04.0
344|0a |322|399|2d-|311|300|2d-|300|311|0a |4eN|48H|4dM|2d-|37 4.29-10-01.NHM-7
0a |28(|63c|29)|20 |4eN|4dM|50P|2e.|00 |00 |00 |00 |00 |49I|0c .(c) NMP......I.
00 |05 |300|388|300|322|00 |00 |00 |00 |4aJ|0c |00 |05 |00 |00 ..0802....J.....
00 |00 |00 |00 |00 |00 |4bK|08 |00 |03 |333|366|00 |00 |4cL|0c ......K...36..L.
00 |04 |322|388|388|00 |00 |00 |00 |00 |54T|0c |00 |05 |300|31 ..288.....T...01
300|355|00 |00 |00 |00 |55U|10 |00 |0a |47G|344|2e.|300|2d-|34 05....U...G4.0-4
2e.|311|322|00 |00 |00 |57W|10 |00 |08 |53S|45E|49I|4bK|4fO|20 .12...W...SEIKO
300|00 |00 |00 |00 |00 |58X|29)|00 |22"|56V|20 |300|344|2e.|30 0.....X)."V 04.0
344|0a |322|399|2d-|311|300|2d-|300|311|0a |4eN|48H|4dM|2d-|37 4.29-10-01.NHM-7
0a |28(|63c|29)|20 |4eN|4dM|50P|2e.|0a |44D|00 |00 |00         .(c) NMP..D...
Sending frame 0x1b / 0x0008
00 |03 |09 |00 |41A|00 |00 |00                                 ....A...
Received frame 0x1b / 0x001a
03 |2b+|09 |01 |00 |01 |41A|14 |00 |10 |333|355|300|377|300|30 .+....A...350700
311|300|355|388|300|311|333|300|333|00                         105801303.
Sending frame 0x1b / 0x0008
00 |03 |0a |00 |42B|00 |00 |00                                 ....B...
Received frame 0x1b / 0x0012
03 |2b+|0a |01 |00 |01 |42B|0c |00 |08 |3a:|05 |07 |10 |50P|08 .+....B...:...P.
311|00                                                         1.
Sending frame 0x1b / 0x0008
00 |03 |0b |00 |43C|00 |00 |00                                 ....C...
Received frame 0x1b / 0x0016
03 |2b+|0b |01 |00 |01 |43C|10 |00 |09 |333|05 |07 |10 |50P|08 .+....C...3...P.
311|00 |f6÷|00 |00 |00                                         1.÷...
Sending frame 0x1b / 0x0008
00 |03 |0c |00 |44D|00 |00 |00                                 ....D...
Received frame 0x1b / 0x0006
03 |2b+|0c |01 |02 |00                                         .+....
Sending frame 0x1b / 0x0008
```

(continues on next page)

(continued from previous page)

```
00 |03 |0d |00 |45E|00 |00 |00                                    ....E...
Received frame 0x1b / 0x0006
03 |2b+|0d |01 |02 |00                                            .+....
Sending frame 0x1b / 0x0008
00 |03 |0e |00 |46F|00 |00 |00                                    ....F...
Received frame 0x1b / 0x0012
03 |2b+|0e |01 |00 |01 |46F|0c |00 |08 |4eN|54T|54T|4aJ|50P|12 .+....F...NTTJP.
344|56V                                                          4V
Sending frame 0x1b / 0x0008
00 |03 |0f |00 |56V|00 |00 |00                                    ....V...
Received frame 0x1b / 0x0006
03 |2b+|0f |01 |02 |00                                            .+....
Sending frame 0x1b / 0x0008
00 |03 |10 |00 |5aZ|00 |00 |00                                    ....Z...
Received frame 0x1b / 0x0006
03 |2b+|10 |01 |02 |00                                            .+....
Sending frame 0x1b / 0x0006
00 |03 |11 |0b |00 |02                                            ......
Received frame 0x1b / 0x0012
03 |2b+|11 |0c |00 |01 |4eN|0c |00 |08 |300|355|300|377|355|32 .+....N...050752
300|00                                                           0.
Sending frame 0x1b / 0x0006
00 |03 |12 |0b |00 |20                                            .....
Received frame 0x1b / 0x0012
03 |2b+|12 |0c |00 |01 |52R|0c |00 |08 |00 |00 |00 |00 |00 |00 .+....R.........
00 |00                                                           ..
Sending frame 0x1b / 0x0006
00 |03 |13 |0b |00 |01                                            ......
Received frame 0x1b / 0x0016
03 |2b+|13 |0c |00 |01 |4dM|10 |00 |0a |53S|54T|41A|344|355|39 .+....M...STA459
311|355|377|00 |00 |00                                           157...
Sending frame 0x1b / 0x0006
00 |03 |14 |07 |00 |02                                            ......
Received frame 0x1b / 0x0012
03 |2b+|14 |08 |00 |01 |49I|0c |00 |05 |300|388|300|322|00 |00 .+....I...0802..
00 |00                                                           ..

    s Get IMEI              { 0x00, 0x41 }
    r Get IMEI              { 0x01, 0x00, 0x01, 0x41, 0x14, 0x00, 0x10, {IMEI(ASCII)},␣
↪0x00 }

Sending frame 0x1b / 0x0008
00 |03 |16 |00 |44D|00 |00 |00                                    ....D...
Received frame 0x1b / 0x0006
03 |2b+|16 |01 |02 |00                                            .+....
Sending frame 0x1b / 0x0006
00 |03 |17 |07 |00 |01                                            ......
Received frame 0x1b / 0x002e
03 |2b+|17 |08 |00 |01 |48H|28(|00 |20 |56V|20 |300|344|2e.|30 .+....H(. V 04.0
344|0a |322|399|2d-|311|300|2d-|300|311|0a |4eN|48H|4dM|2d-|37 4.29-10-01.NHM-7
0a |28(|63c|29)|20 |4eN|4dM|50P|2e.|00 |00 |00 |00 |00         .(c) NMP......
Sending frame 0x1b / 0x0006
```

(continues on next page)

(continued from previous page)

```
00 |03 |18 |07 |00 |01                                          ......
Received frame 0x1b / 0x002e
03 |2b+|18 |08 |00 |01 |48H|28(|00 |20 |56V|20 |300|344|2e.|30 .+....H(. V 04.0
344|0a |322|399|2d-|311|300|2d-|300|311|0a |4eN|48H|4dM|2d-|37 4.29-10-01.NHM-7
0a |28(|63c|29)|20 |4eN|4dM|50P|2e.|00 |00 |00 |00 |00         .(c) NMP......
Sending frame 0x1b / 0x0006
00 |03 |19 |07 |00 |01                                          ......
Received frame 0x1b / 0x002e
03 |2b+|19 |08 |00 |01 |48H|28(|00 |20 |56V|20 |300|344|2e.|30 .+....H(. V 04.0
344|0a |322|399|2d-|311|300|2d-|300|311|0a |4eN|48H|4dM|2d-|37 4.29-10-01.NHM-7
0a |28(|63c|29)|20 |4eN|4dM|50P|2e.|00 |00 |00 |00 |00         .(c) NMP......
Sending frame 0x1b / 0x0006
00 |03 |1a |07 |00 |01                                          ......
Received frame 0x1b / 0x002e
03 |2b+|1a |08 |00 |01 |48H|28(|00 |20 |56V|20 |300|344|2e.|30 .+....H(. V 04.0
344|0a |322|399|2d-|311|300|2d-|300|311|0a |4eN|48H|4dM|2d-|37 4.29-10-01.NHM-7
0a |28(|63c|29)|20 |4eN|4dM|50P|2e.|00 |00 |00 |00 |00         .(c) NMP......
Sending frame 0x1b / 0x0006
00 |03 |1b |07 |00 |01                                          ......
Received frame 0x1b / 0x002e
03 |2b+|1b |08 |00 |01 |48H|28(|00 |20 |56V|20 |300|344|2e.|30 .+....H(. V 04.0
344|0a |322|399|2d-|311|300|2d-|300|311|0a |4eN|48H|4dM|2d-|37 4.29-10-01.NHM-7
0a |28(|63c|29)|20 |4eN|4dM|50P|2e.|00 |00 |00 |00 |00         .(c) NMP......
Sending frame 0x1b / 0x0006
00 |03 |1c |07 |00 |01                                          ......
Received frame 0x1b / 0x002e
03 |2b+|1c |08 |00 |01 |48H|28(|00 |20 |56V|20 |300|344|2e.|30 .+....H(. V 04.0
344|0a |322|399|2d-|311|300|2d-|300|311|0a |4eN|48H|4dM|2d-|37 4.29-10-01.NHM-7
0a |28(|63c|29)|20 |4eN|4dM|50P|2e.|00 |00 |00 |00 |00         .(c) NMP......
Sending frame 0x1b / 0x0006
00 |03 |1d |07 |00 |01                                          ......
Received frame 0x1b / 0x002e
03 |2b+|1d |08 |00 |01 |48H|28(|00 |20 |56V|20 |300|344|2e.|30 .+....H(. V 04.0
344|0a |322|399|2d-|311|300|2d-|300|311|0a |4eN|48H|4dM|2d-|37 4.29-10-01.NHM-7
0a |28(|63c|29)|20 |4eN|4dM|50P|2e.|00 |00 |00 |00 |00         .(c) NMP......
Sending frame 0x1b / 0x0006
00 |03 |1e |07 |00 |01                                          ......
Received frame 0x1b / 0x002e
03 |2b+|1e |08 |00 |01 |48H|28(|00 |20 |56V|20 |300|344|2e.|30 .+....H(. V 04.0
344|0a |322|399|2d-|311|300|2d-|300|311|0a |4eN|48H|4dM|2d-|37 4.29-10-01.NHM-7
0a |28(|63c|29)|20 |4eN|4dM|50P|2e.|00 |00 |00 |00 |00         .(c) NMP......

0x1f: RINGTONE

    s Get Ringtones         { 0x07, 0x00, 0x00, 0xFE, 0x00, 0x7D }
    r Get Ringtones         { 0x08, 0x00, 0x23, 0x00, # ringtones, 0x00,
                              ringtone number, 0x01, 0x01, 0x00, name length (chars),␣
↪name (unicode)... }


0x2b:
    s Get IMEI              { 0x00, 0x41 }
```

(continues on next page)

(continued from previous page)

```
    r Get IMEI              { 0x01, 0x00, 0x01, 0x41, 0x14, 0x00, 0x10, {IMEI(ASCII)},␣
↪0x00 }
    s get HW&SW version     { 0x07, 0x00, 0x01 }
    r get HW&SW version     { 0x08, 0x00, 0x01, 0x58, 0x29, 0x00, 0x22, "V " "firmware\n
↪" "firmware date\n"
                              "model\n" "(c) NMP.", 0x0a, 0x43, 0x00, 0x00, 0x00 }

0x38:
    s ???                   {+00 |02 |00 |0a |00 |01 |00, location, 00}
                            where location: 0, 1, 2, 3
    r ???                   {+02 |1d |00 |0b |00 |01 |00, location, 08 |00 |00 |00 |00␣
↪|00}
    s ???
      00 |02 |00 |0a |00 |60`|00 |10 |00 |11 |00 |12 |00 |13 |00 |14 .....`.........
      00 |15 |00 |16 |00 |17 |00 |18 |00 |19 |00 |1a |00 |1b |00 |1c ................
      00 |1d |00 |1e |00 |1f |00 |20 |00 |21!|00 |22"|00 |23#|00 |24 ....... .!."#.$
      00 |25%|00 |26&|00 |27'|00 |28(|00 |29)|00 |2a*|00 |2b+|00 |2c .%.&.'.(.).*.+.,
      00 |2d-|00 |2e.|00 |2f/|00 |300|00 |311|00 |322|00 |333|00 |34 .-.../.0.1.2.3.4
      00 |355|00 |366|00 |377|00 |388|00 |399|00 |3a:|00 |3b;|00 |3c .5.6.7.8.9.:.;.<
      00 |3d=|00 |3e>|00 |3f?|00 |40@|00 |41A|00 |42B|00 |43C|00 |44 .=.>.?.@.A.B.C.D
      00 |45E|00 |46F|00 |47G|00 |48H|00 |49I|00 |4aJ|00 |4bK|00 |4c .E.F.G.H.I.J.K.L
      00 |4dM|00 |4eN|00 |4fO|00 |50P|00 |51Q|00 |52R|00 |53S|00 |54 .M.N.O.P.Q.R.S.T
      00 |55U|00 |56V|00 |57W|00 |58X|00 |59Y|00 |5aZ|00 |5b[|00 |5c .U.V.W.X.Y.Z.[.\
      00 |5d]|00 |5e^|00 |5f_|00 |60`|00 |61a|00 |62b|00 |63c|00 |64 .].^._.`.a.b.c.d
      00 |65e|00 |66f|00 |67g|00 |68h|00 |69i|00 |6aj|00 |6bk|00 |6c .e.f.g.h.i.j.k.l
      00 |6dm|00 |6en|00 |6fo|00                                     .m.n.o.
    r ???
      02 |1d |00 |0b |00 |60`|00 |10 |04 |00 |00 |11 |0c |06 |00 |00 .....`.........
      00 |00 |00 |00 |00 |00 |00 |12 |04 |00 |00 |13 |04 |00 |00 |14 ................
      08 |00 |00 |00 |00 |00 |00 |15 |08 |00 |00 |00 |00 |00 |00 |16 ................
      08 |00 |00 |00 |00 |00 |00 |17 |08 |00 |00 |00 |00 |00 |00 |18 ................
      08 |00 |00 |00 |00 |00 |00 |19 |08 |00 |00 |00 |00 |00 |00 |1a ................
      08 |00 |00 |00 |00 |00 |00 |1b |08 |00 |00 |00 |00 |00 |00 |1c ................
      04 |00 |00 |1d |08 |04 |00 |00 |00 |00 |00 |1e |3c<|355|ff |ff ............<5
      ff |ff |ff |ff |ff |ff |ff |ff |ff |ff |ff |ff |ff |ff |ff |ff
      ff |ff |ff |ff |ff |ff |ff |ff |ff |ff |ff |ff |ff |ff |ff |ff
      ff |ff |ff |ff |ff |ff |ff |ff |ff |ff |ff |ff |ff |ff |ff |ff
      ff |ff |ff |00 |00 |00 |00 |1f |08 |00 |00 |00 |00 |00 |00 |20    ............
      04 |00 |00 |21!|04 |00 |00 |22"|04 |00 |00 |23#|04 |00 |00 |24 ...!..."...#...$
      04 |00 |00 |25%|04 |00 |00 |26&|04 |00 |00 |27'|08 |00 |00 |00 ...%...&...'....
      00 |00 |00 |28(|08 |00 |00 |00 |00 |00 |00 |29)|08 |00 |00 |00 ...(.......)....
      00 |00 |00 |2a*|04 |00 |00 |2b+|0c |08 |00 |00 |00 |00 |00 |00 ...*...+........
      00 |00 |00 |2c,|04 |00 |00 |2d-|08 |00 |00 |00 |00 |00 |00 |2e ...,...-........
      08 |00 |00 |00 |00 |00 |00 |2f/|08 |00 |00 |00 |00 |00 |00 |30 ......./.......0
      08 |00 |00 |00 |00 |00 |00 |311|08 |00 |00 |00 |00 |00 |00 |32 .......1.......2
      08 |00 |00 |00 |00 |00 |00 |333|08 |00 |00 |00 |00 |00 |00 |34 .......3.......4
      08 |00 |00 |00 |00 |00 |00 |355|08 |00 |00 |00 |00 |00 |00 |36 .......5.......6
      08 |00 |00 |00 |00 |00 |00 |377|08 |00 |00 |00 |00 |00 |00 |38 .......7.......8
      08 |04 |00 |00 |00 |00 |00 |399|08 |04 |00 |00 |00 |00 |00 |3a .......9.......:
      08 |04 |00 |00 |00 |00 |00 |3b;|14 |10 |00 |00 |00 |00 |00 |00 .......;........
      00 |00 |00 |00 |00 |00 |00 |00 |00 |00 |00 |3c<|08 |00 |00 |00 ...........<....
      00 |00 |00 |3d=|08 |00 |00 |00 |00 |00 |00 |3e>|08 |00 |00 |00 ...=.......>....
```

(continues on next page)

(continued from previous page)

```
      00 |00 |00 |3f?|08 |00 |00 |00 |00 |00 |00 |40@|08 |00 |00 |8e ...?.......@...Ä
      00 |00 |00 |41A|04 |00 |00 |42B|04 |00 |00 |43C|08 |00 |00 |00 ...A...B...C....
      00 |00 |00 |44D|08 |00 |00 |00 |00 |00 |00 |45E|08 |00 |00 |00 ...D.......E....
      00 |00 |00 |46F|08 |00 |00 |00 |00 |00 |00 |47G|08 |00 |00 |00 ...F.......G....
      00 |00 |00 |48H|08 |00 |00 |00 |00 |00 |00 |49I|08 |00 |00 |00 ...H.......I....
      00 |00 |00 |4aJ|08 |00 |00 |00 |00 |00 |00 |4bK|08 |00 |00 |00 ...J.......K....
      00 |00 |00 |4cL|08 |00 |00 |00 |00 |00 |00 |4dM|08 |00 |00 |00 ...L.......M....
      00 |00 |00 |4eN|08 |00 |00 |00 |00 |00 |00 |4fO|08 |00 |00 |00 ...N.......O....
      00 |00 |00 |50P|08 |00 |00 |00 |00 |00 |00 |51Q|08 |00 |00 |00 ...P.......Q....
      00 |00 |00 |52R|08 |00 |00 |00 |00 |00 |00 |53S|08 |00 |00 |00 ...R.......S....
      00 |00 |00 |54T|08 |00 |00 |00 |00 |00 |00 |55U|08 |00 |00 |00 ...T.......U....
      00 |00 |00 |56V|08 |00 |00 |00 |00 |00 |00 |57W|08 |00 |00 |00 ...V.......W....
      00 |00 |00 |58X|08 |00 |00 |00 |00 |00 |00 |59Y|08 |00 |00 |00 ...X.......Y....
00 |00 |00 |5aZ|08 |00 |00 |00 |00 |00 |00 |5b[|08 |00 |00 |00 ...Z.......[....
00 |00 |00 |5c\|08 |00 |00 |00 |00 |00 |00 |5d]|08 |00 |00 |00 ...\.......]....
00 |00 |00 |5e^|08 |00 |00 |00 |00 |00 |00 |5f_|08 |00 |00 |00 ...^.......^....
00 |00 |00 |60`|08 |00 |00 |00 |00 |00 |00 |61a|08 |00 |00 |00 ...`.......a....
00 |00 |00 |62b|08 |00 |00 |00 |00 |00 |00 |63c|08 |00 |00 |00 ...b.......c....
00 |00 |00 |64d|08 |00 |00 |00 |00 |00 |00 |65e|08 |00 |00 |00 ...d.......e....
00 |00 |00 |66f|08 |00 |00 |00 |00 |00 |00 |67g|08 |00 |00 |00 ...f.......g....
00 |00 |00 |68h|08 |00 |00 |00 |00 |00 |00 |69i|08 |00 |00 |00 ...h.......i....
00 |00 |00 |6aj|08 |00 |00 |00 |00 |00 |00 |6bk|08 |04 |00 |00 ...j.......k....
00 |00 |00 |6cl|08 |04 |00 |00 |00 |00 |00 |6dm|08 |00 |00 |00 ...l.......m....
00 |00 |00 |6en|08 |00 |00 |00 |00 |00 |00 |6fo|08 |00 |00 |00 ...n.......o....
00 |00                                                           ..
Sending frame 0x38 / 0x00c7
00 |02 |00 |0a |00 |60`|00 |70p|00 |71q|00 |72r|00 |73s|00 |74 .....`.p.q.r.s.t
00 |75u|00 |76v|00 |77w|00 |78x|00 |79y|00 |7az|00 |7b{|00 |7c .u.v.w.x.y.z.{.|
00 |7d}|00 |7e~|00 |7f |00 |80Ç|00 |81ü|00 |82é|00 |83â|00 |84 .}.~...Ç.ü.é.â.ä
00 |85ů|00 |86ć|00 |87ç|00 |88ł|00 |89ë|00 |8aŐ|00 |8bő|00 |8c .ů.ć.ç.ł.ë.Ő.ő.î
00 |8dŹ|00 |8eÄ|00 |8fĆ|00 |90É|00 |91Ĺ|00 |92Í|00 |93ô|00 |94 .Ź.Ä.Ć.É.Ĺ.Í.ô.ö
00 |95Ľ|00 |96ľ|00 |97Ś|00 |98ś|00 |99Ö|00 |9aÜ|00 |9bŤ|00 |9c .Ľ.ľ.Ś.ś.Ö.Ü.Ť.ť
00 |9dŁ|00 |9e×|00 |9fč|00 |a0á|00 |a1í|00 |a2ó|00 |a3ú|00 |a4 .Ł.×.č.á.í.ó.ú.Ą
00 |a5ą|00 |a6Ž|00 |a7ž|00 |a8Ę|00 |a9ę|00 |aa¬|00 |abź|00 |ac .ą.Ž.ž.Ę.ę.¬.ź.Č
00 |adş|00 |ae«|00 |af»|00 |b0 |00 |b1 |00 |b2 |00 |b3 |00 |b4 .ş.«.».........
00 |b5Á|00 |b6Â|00 |b7Ě|00 |b8Ş|00 |b9 |00 |ba |00 |bb |00 |bc .Á.Â.Ě.Ş........
00 |bdŻ|00 |beż|00 |bf |00 |c0 |00 |c1 |00 |c2 |00 |c3 |00 |c4 .Ż.ż............
00 |c5 |00 |c6Ă|00 |c7ă|00 |c8 |00 |c9 |00 |ca |00 |cb |00 |cc ...Ă.ă.........
00 |cd |00 |ce |00 |cf¤|00                                       .....¤.
Received frame 0x38 / 0x0306
02 |1d |00 |0b |00 |60`|00 |70p|08 |00 |00 |00 |00 |00 |00 |71 .....`.p.......q
08 |00 |00 |00 |00 |00 |00 |72r|08 |00 |00 |00 |00 |00 |00 |73 .......r.......s
08 |00 |00 |00 |00 |00 |00 |74t|08 |00 |00 |00 |00 |00 |00 |75 .......t.......u
08 |00 |00 |00 |00 |00 |00 |76v|08 |00 |00 |00 |00 |00 |00 |77 .......v.......w
08 |00 |00 |00 |00 |00 |00 |78x|08 |00 |00 |00 |00 |00 |00 |79 .......x.......y
08 |00 |00 |00 |00 |00 |00 |7az|08 |00 |00 |00 |00 |00 |00 |04 .......z........
08 |04 |00 |7b{|00 |00 |00 |04 |08 |04 |00 |7c||00 |00 |00 |04 ...{.......|....
08 |04 |00 |7d}|00 |00 |00 |04 |08 |04 |00 |7e~|00 |00 |00 |04 ...}.......~....
08 |04 |00 |7f |00 |00 |00 |04 |08 |04 |00 |80Ç|00 |00 |00 |04 ...........Ç....
08 |04 |00 |81ü|00 |00 |00 |04 |08 |04 |00 |82é|00 |00 |00 |04 ...ü.......é....
08 |04 |00 |83â|00 |00 |00 |04 |08 |04 |00 |84ä|00 |00 |00 |04 ...â.......ä....
08 |04 |00 |85ů|00 |00 |00 |04 |08 |04 |00 |86ć|00 |00 |00 |04 ...ů.......ć....
```

(continues on next page)

(continued from previous page)

```
08 |04 |00 |87ç|00 |00 |00 |04 |08 |04 |00 |88ł|00 |00 |00 |04 ...ç.......ł....
08 |04 |00 |89ë|00 |00 |00 |04 |08 |04 |00 |8aŐ|00 |00 |00 |04 ...ë.......Ő....
08 |04 |00 |8bő|00 |00 |00 |04 |08 |04 |00 |8cî|00 |00 |00 |04 ...ő.......î....
08 |04 |00 |8dŹ|00 |00 |00 |04 |08 |04 |00 |8eÄ|00 |00 |00 |04 ...Ź.......Ä....
08 |04 |00 |8fĆ|00 |00 |00 |04 |08 |04 |00 |90É|00 |00 |00 |04 ...Ć.......É....
08 |04 |00 |91Ĺ|00 |00 |00 |04 |08 |04 |00 |92Í|00 |00 |00 |04 ...Ĺ.......Í....
08 |04 |00 |93ô|00 |00 |00 |04 |08 |04 |00 |94ö|00 |00 |00 |04 ...ô.......ö....
08 |04 |00 |95Ľ|00 |00 |00 |04 |08 |04 |00 |96ľ|00 |00 |00 |04 ...Ľ.......ľ....
08 |04 |00 |97Ś|00 |00 |00 |04 |08 |04 |00 |98ś|00 |00 |00 |04 ...Ś.......ś....
08 |04 |00 |99Ö|00 |00 |00 |04 |08 |04 |00 |9aÜ|00 |00 |00 |04 ...Ö.......Ü....
08 |04 |00 |9bŤ|00 |00 |00 |04 |08 |04 |00 |9cť|00 |00 |00 |04 ...Ť.......ť....
08 |04 |00 |9dŁ|00 |00 |00 |04 |08 |04 |00 |9e×|00 |00 |00 |04 ...Ł.......×....
08 |04 |00 |9fč|00 |00 |00 |04 |08 |04 |00 |a0á|00 |00 |00 |04 ...č.......á....
08 |04 |00 |a1í|00 |00 |00 |04 |08 |04 |00 |a2ó|00 |00 |00 |04 ...í.......ó....
08 |04 |00 |a3ú|00 |00 |00 |04 |08 |04 |00 |a4Ą|00 |00 |00 |04 ...ú.......Ą....
08 |04 |00 |a5ą|00 |00 |00 |04 |08 |04 |00 |a6Ž|00 |00 |00 |04 ...ą.......Ž....
08 |04 |00 |a7ž|00 |00 |00 |04 |08 |04 |00 |a8Ę|00 |00 |00 |04 ...ž.......Ę....
08 |04 |00 |a9ę|00 |00 |00 |04 |08 |04 |00 |aa¬|00 |00 |00 |04 ...ę.......¬....
08 |04 |00 |abź|00 |00 |00 |04 |08 |04 |00 |acČ|00 |00 |00 |04 ...ź.......Č....
08 |04 |00 |adş|00 |00 |00 |04 |08 |04 |00 |ae«|00 |00 |00 |04 ...ş.......«....
08 |04 |00 |af»|00 |00 |00 |04 |08 |04 |00 |b0 |00 |00 |00 |04 ...».............
08 |04 |00 |b1 |00 |00 |00 |04 |08 |04 |00 |b2 |00 |00 |00 |04 ................
08 |04 |00 |b3 |00 |00 |00 |04 |08 |04 |00 |b4 |00 |00 |00 |04 ................
08 |04 |00 |b5Á|00 |00 |00 |04 |08 |04 |00 |b6Â|00 |00 |00 |04 ...Á.......Â....
08 |04 |00 |b7Ě|00 |00 |00 |04 |08 |04 |00 |b8Ş|00 |00 |00 |04 ...Ě.......Ş....
08 |04 |00 |b9 |00 |00 |00 |04 |08 |04 |00 |ba |00 |00 |00 |04 ................
08 |04 |00 |bb |00 |00 |00 |04 |08 |04 |00 |bc |00 |00 |00 |04 ................
08 |04 |00 |bdŻ|00 |00 |00 |04 |08 |04 |00 |beż|00 |00 |00 |04 ...Ż.......ż....
08 |04 |00 |bf |00 |00 |00 |04 |08 |04 |00 |c0 |00 |00 |00 |04 ................
08 |04 |00 |c1 |00 |00 |00 |04 |08 |04 |00 |c2 |00 |00 |00 |04 ................
08 |04 |00 |c3 |00 |00 |00 |04 |08 |04 |00 |c4 |00 |00 |00 |04 ................
08 |04 |00 |c5 |00 |00 |00 |04 |08 |04 |00 |c6Ă|00 |00 |00 |04 ..........Ă....
08 |04 |00 |c7ă|00 |00 |00 |04 |08 |04 |00 |c8 |00 |00 |00 |04 ...ă............
08 |04 |00 |c9 |00 |00 |00 |04 |08 |04 |00 |ca |00 |00 |00 |04 ................
08 |04 |00 |cb |00 |00 |00 |04 |08 |04 |00 |cc |00 |00 |00 |04 ................
08 |04 |00 |cd |00 |00 |00 |04 |08 |04 |00 |ce |00 |00 |00 |04 ................
08 |04 |00 |cf¤|00 |00                                       ...¤..
Received frame 0x38 / 0x0306
02 |1d |00 |0b |00 |60`|00 |04 |08 |04 |00 |d0đ|00 |00 |00 |04 .....`.....đ....
08 |04 |00 |d1Đ|00 |00 |00 |04 |08 |04 |00 |d2Ď|00 |00 |00 |04 ...Đ.......Ď....
08 |04 |00 |d3Ë|00 |00 |00 |04 |08 |04 |00 |d4ď|00 |00 |00 |04 ...Ë.......ď....
08 |04 |00 |d5Ň|00 |00 |00 |04 |08 |04 |00 |d6Í|00 |00 |00 |04 ...Ň.......Í....
08 |04 |00 |d7Î|00 |00 |00 |04 |08 |04 |00 |d8ě|00 |00 |00 |04 ...Î.......ě....
08 |04 |00 |d9 |00 |00 |00 |04 |08 |04 |00 |da |00 |00 |00 |db ................
08 |00 |00 |00 |00 |00 |00 |dc |08 |00 |00 |00 |00 |00 |00 |dd ..............Ţ
08 |00 |00 |00 |00 |00 |00 |deŮ|08 |00 |00 |00 |00 |00 |00 |df .......Ů........
08 |00 |00 |00 |00 |00 |00 |e0Ó|08 |00 |00 |00 |00 |00 |00 |e1 .......Ó.......ß
08 |00 |00 |00 |00 |00 |00 |e2Ô|08 |00 |00 |00 |00 |00 |00 |e3 .......Ô.......Ń
08 |00 |00 |00 |00 |00 |00 |e4ń|08 |00 |00 |00 |00 |00 |00 |e5 .......ń.......ň
08 |00 |00 |00 |00 |00 |00 |04 |08 |04 |00 |e6Š|00 |00 |00 |e7 ..........Š...š
08 |00 |00 |00 |00 |00 |00 |e8Ŕ|08 |00 |00 |00 |00 |00 |00 |04 .......Ŕ........
08 |04 |00 |e9Ú|00 |00 |00 |04 |08 |04 |00 |eaŕ|00 |00 |00 |04 ...Ú.......ŕ....
```

(continues on next page)

(continued from previous page)

```
08 |04 |00 |ebŰ|00 |00 |00 |04 |08 |04 |00 |ecý|00 |00 |00 |04 ...Ű.......ý....
08 |04 |00 |edÝ|00 |00 |00 |04 |08 |04 |00 |eeţ|00 |00 |00 |04 ...Ý.......ţ....
08 |04 |00 |ef´|00 |00 |00 |04 |08 |04 |00 |f0|00 |00 |00 |04 ...´...........
08 |04 |00 |f1 |00 |00 |00 |04 |08 |04 |00 |f2 |00 |00 |00 |04 ................
08 |04 |00 |f3 |00 |00 |00 |04 |08 |04 |00 |f4 |00 |00 |00 |f5 ...............§
08 |00 |00 |00 |00 |00 |00 |f6÷|08 |00 |00 |00 |00 |00 |00 |f7 .......÷.......¸
08 |00 |00 |00 |00 |00 |00 |f8°|08 |00 |00 |00 |00 |00 |00 |04 .......°........
08 |04 |00 |f9¨|00 |00 |00 |04 |08 |04 |00 |fa |00 |00 |00 |04 ...¨............
08 |04 |00 |fbű|00 |00 |00 |04 |08 |04 |00 |fcŘ|00 |00 |00 |04 ...ű.......Ř....
08 |04 |00 |fdř|00 |00 |00 |04 |08 |04 |00 |fe |00 |00 |00 |04 ...ř............
08 |04 |00 |ff |00 |00 |00 |04 |08 |04 |01 |00 |00 |00 |00 |04 ... ............
08 |04 |01 |01 |00 |00 |00 |04 |08 |04 |01 |02 |00 |00 |00 |04 ................
08 |04 |01 |03 |00 |00 |00 |04 |08 |04 |01 |04 |00 |00 |00 |04 ................
08 |04 |01 |05 |00 |00 |00 |04 |08 |04 |01 |06 |00 |00 |00 |04 ................
08 |04 |01 |07 |00 |00 |00 |04 |08 |04 |01 |08 |00 |00 |00 |04 ................
08 |04 |01 |09 |00 |00 |00 |04 |08 |04 |01 |0a |00 |00 |00 |04 ................
08 |04 |01 |0b |00 |00 |00 |04 |08 |04 |01 |0c |00 |00 |00 |04 ................
08 |04 |01 |0d |00 |00 |00 |04 |08 |04 |01 |0e |00 |00 |00 |04 ................
08 |04 |01 |0f |00 |00 |00 |04 |08 |04 |01 |10 |00 |00 |00 |04 ................
08 |04 |01 |11 |00 |00 |00 |04 |08 |04 |01 |12 |00 |00 |00 |04 ................
08 |04 |01 |13 |00 |00 |00 |04 |08 |04 |01 |14 |00 |00 |00 |04 ................
08 |04 |01 |15 |00 |00 |00 |04 |08 |04 |01 |16 |00 |00 |00 |04 ................
08 |04 |01 |17 |00 |00 |00 |04 |08 |04 |01 |18 |00 |00 |00 |04 ................
08 |04 |01 |19 |00 |00 |00 |04 |08 |04 |01 |1a |00 |00 |00 |04 ................
08 |04 |01 |1b |00 |00 |00 |04 |08 |04 |01 |1c |00 |00 |00 |04 ................
08 |04 |01 |1d |00 |00 |00 |04 |08 |04 |01 |1e |00 |00 |00 |04 ................
08 |04 |01 |1f |00 |00 |00 |04 |08 |04 |01 |20 |00 |00 |00 |04 ..........  ....
08 |04 |01 |21!|00 |00 |00 |04 |08 |04 |01 |22"|00 |00 |00 |04 ...!.......".....
08 |04 |01 |23#|00 |00 |00 |04 |08 |04 |01 |24$|00 |00 |00 |04 ...#.......$....
08 |04 |01 |25%|00 |00 |00 |04 |08 |04 |01 |26&|00 |00 |00 |04 ...%.......&....
08 |04 |01 |27'|00 |00 |00 |04 |08 |04 |01 |28(|00 |00 |00 |04 ...'.......(....
08 |04 |01 |29)|00 |00 |00 |04 |08 |04 |01 |2a*|00 |00 |00 |04 ...).......*....
08 |04 |01 |2b+|00 |00 |00 |04 |08 |04 |01 |2c,|00 |00 |00 |04 ...+.......,....
08 |04 |01 |2d-|00 |00 |00 |04 |08 |04 |01 |2e.|00 |00 |00 |04 ...-............
08 |04 |01 |2f/|00 |00                                                .../..
Sending frame 0x38 / 0x000e
00 |02 |00 |0c |00 |01 |00 |01 |08 |02 |05 |08 |00 |00                 ..............
Received frame 0x38 / 0x0006
02 |1d |00 |0d |00 |00                                                ......

0x39: PROFILES

    s Get Profile           { 0x01, 0x01, 0x0c, 0x01,
                              0x04 (length), profile #, 'feature', 0x01 }
    r Get Profile           { 0x02, 0x00, 0x0c, 0x02,
                              0x09 (length), type, 0x01, 0x02, 0x00, 0x00, 0x01, value,␣
↪0x02 ... }

    s Set Profile           { 0x03, 0x01, # blocks, 0x03,
                              length, type, profile #, value, 0x00, 0x00, 0x01, value,␣
↪0x03 ... }
    r Set Profile           { 0x04, 0x01, # blocks,
```

(continues on next page)

(continued from previous page)

```
                              length, 0xXX, type, 0xXX, value
                              where value: 0x00 = success

0x3E: FM Radio
    s Get FM Station            { 0x00, 0x01, 0x00, 0x05, location, 0x00, 0x01}
    r Get FM Station            {                   0x06, 0x00, 0x01, 0x00, 0x1c,
                  name_length, 0x14, 0x09, 0x00, location, 0x00, 0x00, 0x01,
                  FreqHI , FreqLO,
                  name_in_unicode,[0x55,0x55] - if name_length is odd}
                              where frequency = (0xffff + FreqHi * 0x100 + FreqLo) kHz

    r Get FM Station            {                   0x16, 0x05, 0x06 } - if entry is␣
↪empty

0x42:
    s ????                  {+00 |07 |00 |01 |00 |02}
    r ????                  {+07 |2d-|00 |02 |06 |02 |00 |02 |00 |01 |02 |08 |00 |0c |07␣
↪|d1 |00 |00}

0x42:
    s Get ???               {+0x00, 0x07, 0x02, 0x01, 0x00, 0x01 }
    r Get ???               { 02 |06 |02 |00 |02 |00 |01 |02 |08 |00 |0c |07 |d1 |00 |00}
    s Get original IMEI ?   {+0x00, 0x07, 0x02, 0x01, 0x00, 0x01 }
    r Get original IMEI ?   { 0x02, 0x06, 0x01, 0x01, 0x00, 0x01, 0x01, 0x18, 0x01, 0x00,
↪ IMEI, 0x00, 'U' }
    s Get ???               {+0x00, 0x07, 0x03, 0x01, 0x00, 0x02 }
    r Get ???               { 02 |06 |02 |00 |02 |00 |01 |02 |08 |00 |0c |07 |d1 |00 |00}
    s Get ???               {+0x00, 0x07, 0x04, 0x01, 0x00, 0x10 }
    r Get ???               { 02 |06 |10 |00 |10 |00 |01 |05 |08 |00 |00 |00 |00 | 00␣
↪|00}
    s Get ???               {+0x00, 0x07, 0x05, 0x01, 0x00, 0x08 }
    r Get ???               { 02 |06 |08 |00 |08 |00 |01 |04 |08 |00 |00 |00 |00 | 00␣
↪|00}
    s Get ???               {+0x00, 0x07, 0x06, 0x01, 0x00, 0x20 }
    r Get ???               { 02 |06 |20 |00 |20 |00 |01 |06 |04 |03 |00}

0x43:
    s ????                  {+00 |08 |00 |01 |00 |00 |00 |00 |00, x}
                            where x = 0x01, 0x02, 0x04, 0x08, 0x10
    r ????                  {+08 |1f |y |02 |00 |00 |00 |00 }
                            where y = 0 - 0x04
    s ???                   {+00 |08 |05 |01 |00 |00 |00 |00 |00 |20}
    r ???                   {+08 |1f |05 |02 |00 |00 |00 |00}

0x45: PHONEBOOK HANDLING ????
    the same to msg 0x03 ????

0x53:
    s Get simlock info      {0x0C}

0x55: TODO
    s Get TODO              {0x03, 0x00, 0x00, 0x80, location low, location hi}
```

(continues on next page)

(continued from previous page)

```
    r Get TODO              {0x04, .... }
    s Get number of TODO    {0x07}
    r Get number of TODO    {0x08, number lo, number hi}
    s Delete all TODO       {0x11}
    r Delete all TODO       {0x12}
    s Get TODO locations    {0x15, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00}
    r Get TODO locations    {0x16, ...}

0x7a: STARTUP

    s Get startup logo      { 0x02, 0x0f }
    r Get startup logo      { 0x03, 0x0f, 0x00[4], # blocks,
                              0xc0, 0x02, height (2 octets),
                              0xc0, 0x03, width (2 octets),
                              0xc0, 0x04, size (2 octets),
                  picture }

    s Get startup greeting  { 0x02, 0x01, 0x00 }
    r Get startup greeting  { 0x03, 0x01, 0x00, greeting (unicode), 0x00 }

    s Get anykey answer     { 0x02, 0x05, 0x00 }
    r Get anykey answer     { 0x03, 0x05, 0x00, 0x00/0x01 }

0xd1:
    s Get HW&SW version     { 0x0003, 0x00 }
0xd2:
    r Get HW&SW version     { 0x0003 "V " "firmware\n" "firmware date\n"
                              "model\n" "(c) NMP." }
```

13.6 Nokia 7110

Assembled by Balazs Nagy <js@iksz.hu> Marcin Wiacek <Marcin@MWiacek.com> Jens Bennfors <jens.bennfors@ing.hj.se> Michael Hund <michael@drhund.de> Jay Bertrand <jay.bertrand@libertysurf.fr> Gabriele Zappi <gzappi@inwind.it> Markus Plail <plail@web.de> Ralf Thelen <ralf@mythelen.de> Walek <walek@pa98.opole.sdi.tpnet.pl> ... and other members of gnokii mailing list and authors of some WWW pages.

The data provided is for information purposes only. Some of the frames might be hazardous to your phone. Be careful!!! We do not take any responsibility or liability for damages, etc.

Note: this information isn't (and can't be) complete. If you know anything about features not listed here or you noticed a bug in this list, please notify us via e-mail. Thank you.

Document describing frames used in GSM Nokia 6210 and derivatives (7110)

Correct format is FBUS version 2/Infrared/MBUS version 2 (see nokia.txt for protocol details):

List:

```
0x00: Connect to NBS port ?
    r Set ringtone          {+0x7c,0x01,0x00,0x0d,0x06[6],0x78,ringtone packed according␣
```

(continues on next page)

(continued from previous page)

```
↪to SM2.0}
                          Seems not to work in MBUS!
0x01: Communication Status
?   r Call msg              { 0x0002 }
?   r Call in progress      { 0x0003, seqnr }
?   r Remote end hang up    { 0x0004, seqnr, ?, error (like in netmon in 39) }
?   r incoming call alert   { 0x0005, seqnr, numlen, "number", namelen, "name" }
?   r answered call         { 0x0007, seqnr }
?   r terminated call       { 0x0009, seqnr }
?   r call msg              { 0x000a, seqnr }
    Note: in 6210 4.27 all msg from 0x01 seems to be unavailable
0x02: SMS handling
    s Send SMS message      { 0x0001, 0x02, 0x00 (SEND REQUEST), ... }
    r Message sent          { 0x0002 }
    r Send failed           { 0x0003, ?, ?, error (like in netmon in 65)}
    s Incoming SMS info on  { 0x000d, 0x00, 0x00, 0x02}
                          note: no info about Delivery Reports
    r Incoming SMS info onOK{ 0x000e }
                          note: no info about Delivery Reports
    r Incoming SMS infoonerr{ 0x000f, error }
                          where error: 0x0c - no PIN
    r SMS message received  { 0x0011, ..... } (whole message)
    s Set CellBroadcast     { 0x0020, 0x01, 0x01, 0x00, 0x00, 0x01, 0x01 }
                                    for enable cell broadcast ?
                                    0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }
                                    for disable cell broadcast ?
    r Set CellBroadcast OK  { 0x0021, 0x01 }
    r Read CellBroadcast    { 0x0023, ?, ?, ?, channel, ?, message... } ?
    s Set SMS center        { 0x0030, 0x64, priority, checksum?, format,
                                    validity[2], {DefaultRecipient no.}[12],
                                    {SMScenter no.}[12], {SMSC name}, 0x00}
                              where tel.no.[12]: {len, type, {number(BCD)}}
                                    type: 0x81: normal
                                          0x91: + (international)
                                          0xd0: alphanumeric
                                    format: 0x00: text
                                            0x22: fax
                                            0x24: voice
                                            0x25: ERMES
                                            0x26: paging
                                            0x31: X.400
                                            0x32: email
                                    validity: 0x000b:  1 hour
                                              0x0047:  6 hours
                                              0x00a7: 24 hours
                                              0x00a9: 72 hours
                                              0x00ad:  1 week
                                              0x00ff: max.time
    r Set SMS center OK     { 0x0031 }
    r Set SMS center error  { 0x0032, reason }
    s Get SMS center        { 0x0033, 0x64, priority }
    r SMS center received   { 0x0034, priority, checksum?, type,
```

(continues on next page)

(continued from previous page)

```
                                    validity[2], {DefaultRecipient no.}[12],
                                    {SMScenter no.}[12], {SMSC name}, 0x00 }
                              where priority, checksum, type, validity,
                                    tel.no.[12]: see 0x02/0x0030
    r SMS center error recv { 0x0035, reason }
    s??                     { 0x0074}
    r??                     { 0x0075, 0xFF, 0x11, 0x98}
    s??                     { 0x008C}
    r??                     { 0x008D, 0x00}
0x03: Phonebook functions
    s Get memory status     { 0x0103, 0x02, memory type }
                            where: memory type - see 0x03/0x0107
    r Get memory status     { 0x0104, 0x00, xL, 0x00[2], y1H, y1L, 0x10,
                              0x00[2], z?, ymaxH, ymaxL, y2H, y2L,
                              0x0d?, xH?, 0x00[2]? }
                              where y1: location (lowermost)
                                    y2: no. of locations
                                    ymax: maximum location no.
    s Read memory           { 0x0107, 0x01, 0x01, 0x00, 0x01, xH, xL,
                               yH, yL, 0x00, 0x00}
                            where x: memory type
                              0x01: (256) Dialled numbers
                              0x02: (512) Missed calls
                              0x03: (768) Received calls
                              0x05: (500) telephone phonebook
                              0x06: (160) SIM phonebook
                              0x07: (10/0)
                              0x08: (1/0)
                              0x09: (4) voice mailbox
                              0x0e: (10) speed dials
                              0x10: (5) caller groups
                                  y: location
    r Read memory error     { 0x0108, 0x00, 0x01,
                             code,0x00, 0x00, z, error}
                            where code==0x0f
                                  error: 0x34 - phonebook location not found
                                         0x3b - speed dial not assigned
    r Read memory OK        { 0x0108, 0x00, 0x01,
                             code,0x00, 0x00, z, xH, xL, yH, yL, 0x00, 0x00, 0x00, no.of␣
↪blocks, { block } * }
                            where code: != 0x0f
                              y: location
                              z: generic block size
                              block: {id, 0, 0, blocksize, block no.,
                                      {contents}, 0x00}
                                id: 0x04 pointer to another memory location { 0xff?, yH,␣
↪yL, xL,0x00[3] }
                                    0x07 name {len, (unicode)},
                                    0x08 email
                                    0x09 postal
                                    0x0a note {len, (unicode)}
                                    0x0b number {type, 0x00[3], len, (unicode)}
```

(continues on next page)

(continued from previous page)

```
                                    0x0c ringtone {ringtone no., 0, 0}
                    0x13 date for a called list (DC, RC, etc.)
                                    0x1b caller group graphic {width, height, 0, 0
↪{bitmap}}
                                    0x1c caller group graphic on? {(1: yes, 0: no), 0, 0}
                                    0x1e caller group number {number, 0, 0}
                                   type: 0x0a: General,
                                         0x03: Mobile (office ?),
                                         0x06: Work,
                                         0x04: Fax,
                                         0x02: Home (mobile ?)
    s Set mem location      { 0x010b, 0x00, 0x01, 0x01, 0x00, 0x00, z,
                                      xH, xL, yH, yL, 0x00, 0x00, 0x00,
                                      no.of blocks, { block }[no.of blocks] }
    r Set mem location      { 0x010c, 0?, 1?, code, 0?, 0?, z?, 0?, 0?,
                                      yH, yL, xL }
                            where code:
                                      0x3d - wrong entry type
                                      0x3e - too many entries
    s Delete mem location   { 0x010f, 0x00, 0x01, 0x04, 0x00, 0x00, 0x0c, 0x01, 0xff, xH,
↪ xL,
                                      yH, yL, 0x00, 0x00}
                                      where   x: location
                                      y: memory type
    r Delete mem location   { 0x0110, 0x00, 0x00 }
0x06: Calling line restriction/Call forwarding etc
    r Get call divert       { 0x0001, 0x02, x, 0x00, divtype, 0x02, calltype, y, z, 0x0b,
↪ number, 0x00...0x00, timeout (byte 45) }
    s Set call divert       { 0x0001, 0x03, 0x00, divtype, calltype, 0x01, number(packed␣
↪like in SMS), 0x00 ... 0x00,
                                      length of number (byte 29), 0x00 ... 0x00, timeout␣
↪(byte 52), 0x00, 0x00, 0x00}
                            NOTE: msglen=0x37
                            where timeout:
                              0x00: not set ?
                              0x05: 5 second
                              0x0a: 10 second
                              0x0f: 15 second
                              0x14: 20 second
                              0x19: 25 second
                              0x1e: 30 second
                            where divtype:
                              0x02: all diverts for all call types ?
                                    Found only, when deactivate all diverts for all call␣
↪types (with call type 0x00)
                              0x15: all calls
                              0x43: when busy
                              0x3d: when not answered
                              0x3e: if not reached
                            calltype:
                              0x00: all calls (data, voice, fax)
                              0x0b: voice calls
```

(continues on next page)

(continued from previous page)

```
                              0x0d: fax calla
                              0x19: data calls
    s Deactivate calldiverts{ 0x0001, 0x04, 0x00, divtype, calltype, 0x00 }
                            where divtype, calltype: see above
    r Deactivate calldiverts{ 0x0002, 0x04, 0x00, divtype, 0x02, calltype, data }
    s Get call diverts      { 0x0001, 0x05, 0x00, divtype, calltype, 0x00 }
                            where divtype, calltype: see above
    r Get call diverts ok   { 0x0002, 0x05, 0x00, divtype, 0x02, calltype, data }
                            where divtype, calltype: see above
                      data: { 0x01, 0x00 } - isn't active
                        { 0x02, 0x01, number(packed like in SMS), 0x00, 0x00..., timeout␣
↪}
    r Get prepaid(?) info   { 0x0005, ?,?,?,length,message(packed like in 7bit SMS)}
    r Call diverts active   { 0x0006, ??? }
0x0a: Network status
    s get used network      { 0x0070 }
    r get used network      { 0x0071, available,?,?,length,netstatus,netsel,cellIDH,
                                      cellIDL,lacH,lacL,MCC+MNC[3],{Opstr}, 4?,
                                      len, xlen(78), ylen(21), 0, {bitmap} }
                              where {Opstr}: namelen, {operator name(unicode)}
                                    len: {xlen, ylen, 0, {bitmap} + 2
                                    {bitmap}: bitmaplen, 0, 0, {OTA bitmap}
                    available: 0x02 if the logo following is valid,
                               0x01 for no operator logo following
    s get network status    { 0x0081 }
    r get network status    { 0x0082, network%, 0x14? }
    s set operator logo     { 0x01a3 0x01, oplogo?, MCC+MNC[3], 0?,4?,len,
                                      xlen(78),ylen(21), 0 (frames?),
                                      {bitmap}*?, 0x00(padding) }
                              where len, {bitmap}: see 0x0a/0x0071
    r set operator logo OK  { 0x01a4 }
    s clear operator logo   { 0x00af, x}
                            where x==0 to 4
    r clear operator logo   { 0x00bf}
0x13: Calendar notes
    s Add meeting note      { 0x0001, body like in subtype 0x001a...}
    r Add meeting note      { 0x0002, location (2 bytes), status (2 bytes)}
    s Add call note         { 0x0003, body like in subtype 0x001a...}
    r Add call note         { 0x0004, location (2 bytes), status (2 bytes)}
    s Add birthday note     { 0x0005, location (2 bytes), entry type, 0x00, year of␣
↪birth(2 bytes),
                                      Month, Day, 0x00, 0x00, alarm (4 bytes), alarm␣
↪type, length, text (Unicode)}
    r Add birthday note     { 0x0006, location (2 bytes), status (2 bytes)}
    s Add reminder note     { 0x0007, body like in subtype 0x001a...}
    r Add reminder note     { 0x0008, location (2 bytes), status (2 bytes)}
    s Delete calendar note  { 0x000b, location (2 bytes) }
    r Delete calendar note  { 0x000c, location (2 bytes), ?, ?, ?, ? }
    s Get calendar note     { 0x0019, location (2 bytes) }
    r Calendar note recvd   { 0x001a, location (2 bytes), entry type, 0x00, year (2␣
↪bytes), Month, Day, block}
                            where: entry type - 0x01 - Meeting, 0x02 - Call, 0x04 -␣
```

(continues on next page)

(continued from previous page)

```
↪Birthday, 0x08 - Reminder
                                    block: for Meeting:{hour,minute,alarm (two bytes),
↪recurrence (two bytes),len,0x00,string(unicode)}
                                          where alarm=Number of minutes before the time␣
↪of the meeting
                                                  that the alarm should be triggered:
                                                  For meetings with "No alarm"=0xFFFF (-
↪1).
                                                  For "On time"=0x0000
                                                  half an hour=0x001E, and so on.
                                                Recurrence=in hours, between future␣
↪occurrences of this meeting.
                                                  If there is no repeat, this value is␣
↪0x0000. The special value 0xffff
                                                  means 1 Year!
                                          for Call:{Hour,Minute,Alarm (as above),
↪Recurrence (as above),namelen,numberlen,
                                                    name(unicode),number(unicode)}
                                          for Reminder:{Recurrence (as above),len,0x00,
↪string(unicode)}
                                          for Birthday:{byte1,byte2,alarm(4 bytes),
↪yearofbirth,alarmtype,len,string(unicode)}
                                                  byte1 and byte2 may vary (???).␣
↪Usually are 0x00 both (but not always)
                                                  In Birthday, the Year in the common␣
↪part, usually contains a strange year.
                                                  So, don't consider it as Year of␣
↪note, neither year of BirthDay (for Year of
                                                  Birthday use the value described␣
↪below).
                                          where alarm=32-bit integer that is the number␣
↪of seconds between the desired
                                                  alarm time and 11:59:58pm on the␣
↪birthday.For "No Alarm", the value is
                                                  0x0000FFFF (65535).
                                                YearOfBirth=used instead of the one in␣
↪the common part of the entry (see above)
                                                  but only when reading birthday entries.
↪ For storing entries, this field does
                                                  not exist.
                                                AlarmType: 0x00 - Tone, 0x01 - Silent
?   s???                    { 0x0021 }
?   r???                    { 0x0022, 0x5A, 0x00 }
?   s???                    { 0x0025 }
?   r???                    { 0x0026, 0x04, 0x00 }
?   s                       { 0x0029 }
?   r                       { 0x002A, 0x04, 0x00 }
    s Get first free pos    { 0x0031 }
    r Get first free pos    { 0x0032, location (2bytes) }
    s Get notes info        { 0x003a, 0xFF, 0xFE}
    r Get notes info        { 0x003b, how many notes used (2 bytes), 0x01, 0x07, { two␣
↪bytes with location for each note} *}
```

(continues on next page)

(continued from previous page)

```
?   s Get calendar note??   { 0x003E, location (2 bytes) }
?   r Get calendar note??   { 0x003F, location (2bytes), ... }
0x14:
    s Get Picture Image     { 0x0007, location, number[2 bytes], 0x00, 0x64 }
    r Get Picture Image     { 0x0008, 0x07, location, number[2 bytes], 0x07, ??[38],
                                    width, height, lenH, lenL, {bitmap}, 0x00, 0x00,␣
↪text len, text(coded like in SMS)...}
    r Get SMS failed        { 0x0009, 0x02 },
    s Get SMS status        { 0x0036, 0x64 }
    r Get SMS Status        { 0x0037, 0x05/0x03, 0x01, 0x00, 0x00,
                              a (2 octets), b (2 octets), c (2 octets),
                              d (2 octets), e (2 octets), 0x00
                              where:
                              a - according to P.Kot:
                                Number of locations in "fixed" memory. These are all
                                Templates entries in my Nokias 6210 (NPE-3 (c) NMP V05.36
                                14-11-01, NPE-3 (c) NMP V05.27 01-08-01).
                                I can't remove any of Templates entries in my phone.
                                Marcin Wiącek: Rather not ! I don't agree.
                                I have 0x00, 0x0f and 10 templates and 3 SMS
                                and 10 Picture Images.
                              b - Number of used messages in phone memory. These
                                are messages manually moved from the other folders.
                                Picture messages are saved here.
                              c - Number of unread messages in phone memory. Probably
                                only smart msssages.
                              d - Number of used messages in SIM memory. These are
                                either received messages or saved into Outbox/Inbox.
                                Note that you *can't* save message into this memory
                                using 'Move' option. Picture messages are not here.
                              e - Number of unread messages in SIM memory

    s Set Picture Image     { 0x0050, 0x07, location, number[2 bytes], 0x07, ??[38],
                                    width, height, lenH, lenL, {bitmap}, 0x00, 0x00,␣
↪text len, text(coded like in SMS)...}
                              std. size: 72x28
    r Set Picture Image     { 0x0051, location, number[2 bytes], 0x07 }
    s Set SMS name          { 0x0083,folder,location(2bytes),name(Unicode),0x00 , 0x00}
    r Set SMS name          { 0x0084,folder,0x00, 0x00, name (Unicode),0x00,0x00}
    s List Picture Images   { 0x0096, location, 0x0f, 0x07 }
                              where location:
                      LM tries with 0x09, 0x11, 0x19, 0x21, 0x29, 0x31, 0x39, 0x41, 0x49
                  Returned value with 0x21
    r List Picture Images   { 0x0097, number of pictures[2 bytes], number1[2 bytes],␣
↪number2[2 bytes], ..., }
    s Write SMS to folder   { 0x0104, status, folder ID, location(2 bytes), 0x02, 0x01,␣
↪SMS stuff ... }
    r Write SMS to folder   { 0x0105, folder ID, location(2 bytes), 0x00 }
    r Write SMS to folder   { 0x0106, 0x02 (write failed errorcode ?) }
    s Get SMS from folder   { 0x0107, folderID, location(2 bytes), 0x01, 0x65, 0x01}
                            where: folderID - see 0x14/0x017B
    r Get SMS from folder   { 0x0108, status, folderID, 0x00, location, type, sender␣
```

(continues on next page)

(continued from previous page)

```
↪number,...}

                          where: status=0x01 - reveived/read
                      0x03 - received/unread
                      0x05 - stored/sent
                      0x07 - stored/not sent

                where: folderID - see 0x14/0x017B

                          where: type=0x00 - received SMS
                    0x01 - delivery report
                    0x02 - stored SMS
                    0x07 - picture message
    s Delete SMS message    { 0x010a, folderID, location(2 bytes), 0x01 }
    r Delete SMS            { 0x010b }
    s Get folder status     { 0x016b, folderID, 0x0F, 0x01}
                            where: folderID - see 0x14/0x017B
    r Get folder status     { 0x016c, number of entries (2 bytes), entry1number (2␣
↪bytes), entry2number(2 bytes), ....}
    s Get folder names      { 0x017A, 0x00, 0x00}
    r Get folder names      { 0x017B, number of strings, folderID, name1, 0x00, folderID,
↪ name2, 0x00, name3, 0x00,...}
                            where: folderID=0x08 - Inbox
                                           0x10 - Outbox
                                           0x18 - Archive
                                           0x20 - Templates
                                           0x29 - first "My folders"
                                           0x31 - second "My folders"
                                           0x39 - third -"-
                                           and so on
0x17:
    s Get Battery info      { 0x0002 }
    r Get Battery info      { 0x0003, 0x0b, batt%, 0x14?, 0x01? }
0x19: Phone clock & alarm

    These frames are like the same frames subtypes in 0x11 in 6110

    s set date and time     { 0x0060, 1,1,7,yearh,yearl,month,mday,hour,min,0x00 }
    r date and time set     { 0x0061 }
    s get date and time     { 0x0062 }
    r date and time recvd   { 0x0063,date_set?,time_set?,?,?,yearh,yearl,month,mday,hour,
↪min,second }
                            where: date_set & time_set==0x01 - set
                                           0x00 - not set, ?,?,yearh,yearl,month,mday,
↪hour,min,second
                                                                   not available in frame
    s set alarm             { 0x006b, 1,32,3,0x02(on-off),hour,min,0x00 }
    r alarm set             { 0x006c }
    s get alarm             { 0x006d }
    r alarm received        { 0x006e,?,?,?,?,alrm(==2:on),hour,min }

    These are new (?)
```

(continues on next page)

(continued from previous page)

```
?   s ??                    { 0x0083, id }
?   r ??                    { 0x0084, 0x01, 0x40, 0x03, id, 0x00, 0x00 }
?   r ??                    { 0x0084, 0x01, 0x40, 0x03, id, 0x00, 0x01 }
?   r ??                    { 0x0084, 0x01, 0x40, 0x03, id, 0x01, 0x00 }
                            where: id=0x27,0x2a,0x32,0x28,0x40
0x1b:
    s Get IMEI              { 0x0001 }
    r Get IMEI              { 0x0002, {IMEI(ASCII)}, 0x00 }
    s get HW&SW version     { 0x0003, 0x01, 0x32 }
    r get HW&SW version     { 0x0004, "V " "firmware\n" "firmware date\n"
                              "model\n" "(c) NMP." 0x00 0xff[14] }
0x1f:
    s ???                   { 0x0010, 0x02, 0x00, 0xff, 0xff }
    r ???                   { 0x0011, length, 0x00, {block}[length] }
                              where block: { unicode letter[2], 0x0000,
                                0x00, 0x55, ??, ?? }
    s Set ringtone          { 0x011f, 0x00, location, 0x00, name(Unicode),
                              ringtone(format the same to 0x40/0x019e and 0x40/0x01a0) }
                              where: location: 0x87 to 0x8b on N6210
                                               0x74 to ... on N7110
    s Get ringtone          { 0x0122, 0x00, location}
    r Get ringtone          { 0x0123, 0x00, location, name(Unicode), 0x00,...,0x00, 0x02,
↪0xFC,0x09(ringtone contenst)}
    r Get ringtone error    { 0x0124, ...}
0x39:
    s get profile feature   { 0x0101, 0x01, 0x01, 0x01, number1, number2}
                            where number1: from 0x00 to 0x07 (for each profile ?)
                                  number2: 0x00 - 0x09, 0x0A, 0x16 - 0x19, 0x1a - 0x1f,␣
↪0x20 - 0x29, 0x2a - 0x2c, 0xff
                                       where 0x09: keypad tones                          ␣
↪               0x02: incoming call alert                                              ␣
↪0x03: ringtone number
                                           0x04: ringing volume
                                           0x05: message alert tone                      ␣
↪                     0x06: vibra                                              0x07:␣
↪warning tones                                          0x08: caller groups alert for ␣
↪                                    0x09: automatic answer
                                           0xff: name
    r get profile feature   { 0x0102, 0x01, 0x02, number2, block...}
                            for number2==0xff: (Profile Name)
                              block: 0x01, length, name(Unicode), 0x00, 0x00
                            for number2==0x00: (Keypad Tones)
                              block: 0x01, 0x01, 0x01, Type, 0x01
                              where: Type : 0x00 = Off
                                            0x01 to 0x03 = Level1 .. Level3
                            for number2==0x02: (Incoming Call Alert)
                              block: 0x01, 0x01, 0x01, Type, 0x01
                              where: Type : 0x00 = Ringing
                                            0x01 = Ascending
                                            0x02 = Ring Once
                                            0x03 = Beep Once
```

(continues on next page)

(continued from previous page)

```
                                          0x05 = Off
                            for number2==0x03: (Ringtone Number)
                              block: 0x01, 0x01, 0x01, Number, 0x01
                              where: Number : 0x40 to 0x62 - gives number of factory␣
↪ringtone. The number of menu is
                                               obtained by doing (Number - 0x3f);
                              where: Number : 0x89 to 0x8d - gives number of uploaded␣
↪ringtone. The number of menu is
                                               obtained by doing (Number - 0x65),while␣
↪the uploaded ringtone number is
                                               obtained by doing (Number - 0x88).
                            for number2==0x04: (Ringing volume)
                              block: 0x01, 0x??, 0x??, Volume, 0x01
                              where: Volume : 0 = Level1 .. to 4 = Level5
                            for number2==0x05: (Message Alert Tone)
                              block: 0x01, 0x01, 0x??, Type, 0x01
                              where: Type : 0x00 = Off
                                            0x01 = Standard
                                            0x02 = Special
                                            0x03 = Beep Once
                                            0x04 = Ascending
                            for number2==0x06: (Vibration)
                              block: 0x01, 0x??, 0x??, Switch, 0x01
                              where: Switch : 0 = Off, 1 = On
                            for number2==0x07: (Warning Tones)
                              block: 0x01, 0x??, 0x??, Switch, 0x01
                              where: Switch : 0 = Off, 1 = On
                            for number2==0x08: (Caller groups Alert for)
                              block: 0x01, 0x??, 0x??, Callers, 0x01
                              where: Callers : 0xff = All calls alert (Read below *)
                                               0x01 = Family
                                               0x02 = VIP
                                               0x04 = Friends
                                               0x08 = Colleagues
                                               0x10 = Others
                                          All logical OR among groups are valid, so if you␣
↪select from one phone's profile
                                          alert for Friends and Colleagues, a 0x0c will␣
↪return (because 0x04 OR 0x08 = 0x0c).
                                       (*) If Callers==0xff, means "Alert for All calls". Then,
↪ in this case, you don't
                                          need to read other groups selection.
                            for number2==0x09: (Automatic answer)
                              block: 0x01, 0x??, 0x??, Switch, 0x01
                              where: Switch : 0 = Off, 1 = On
                                N.B. This feature is valid for Handsfree and Headset␣
↪profiles only!
   s ???                    { 0x0101, 0x04, 0x01, 0x01, 0xff, 0x03 }
   r ???                    { 0x0102, 0x01, 0x02, 0x03, 0x01, 0x01, 0x01, 0x85/0x087 }

   s ?                      { 0x0105}
   r ?                      { 0x0106, 0x01, 0x04}
```

(continues on next page)

(continued from previous page)

```
0x3f: WAP
    s Enable WAP frames     { 0x0000}
    r Enable WAP frames     { 0x0002, 0x01}

    s  ??                   { 0x0003}
    r  ??                   { 0x0004}

    s Get WAP bookmark      { 0x0006, 0x00, location}
                              where location: 0 - 14
    r Get WAP bookmark      { 0x0007, 0x00, name_len, name(unicode),
                              url_len, url(unicode), 0x01,0x80,0x00[7]}
    r Get WAP bookmark err  { 0x0008, error }
                              where error:
                                0x00(?)invalid position
                                0x01   user inside "Bookmarks" menu. Must leave it
                                0x02   invalid/too high/empty location

    s Set WAP bookmark      { 0x0009, 0xff, 0xff, name_len, name(unicode),
                              url_len, url(unicode), 0x01,0x80,0x00[7] }
                              Note: bookmark is added to the first free location.
    r Set WAP bookmark OK   {+0x01, 0x36, 0x0a, block }
                              where block:
                                0x0a, location_of_just_written_bookmark(?),
                                0x00, next_free_location(?)
    r Set WAP bookmark err  {+0x01, 0x36, 0x0b, error }
                              where error:
                               0x04 - memory is full
                               0x01 - we are in the bookmark menu
                               0x00 - unknown reason for now ;(

?   s Delete WAP bookmark   { 0x000c, 0x00, location }
                              where: location = 0-14
?   r Delete WAR bookmark OK{ 0x000d }
?   r Delete WAPbookmark err{ 0x000e, 0x02 }

    s ??                    { 0x000F}
    r ??                    { 0x0010, 0x00}

    s Get WAP settings 1    { 0x0015, location}
                            where location: 0x00 - 0x05
    r Get WAP settings 1 OK { 0x0016, title length, title (Unicode), URL length,␣
↪URL(Unicode),con_type, ???[6 bytes],location, ???[5 bytes],security,...}
                            where:
                              con_type: 0x00 - temporary
                                        0x01 - continuous
                              location: when use "Get WAP settings 2 frame", must give it
                              security: 0x00 = no, 0x01 = yes
    r Get WAP settings 1 err{ 0x0017, error }
                              where error:
                                0x01   user inside "Settings" menu. Must leave it
                                0x02   invalid/too high/empty location
    s Get WAP settings 2    { 0x001b, location}
```

(continues on next page)

(continued from previous page)

```
                            where location: 0x00 - 0x1d (you get it in "Get WAP settings␣
↪1" frame)
    r Get WAP settings 2 OK { 0x001c, 0x01, type, frame...}
                            where type : 0x00 - SMS bearer
                                           frame:
                                             service_num_len, service_num (Unicode),␣
↪server_num_len, server_num(Unicode)
                                         0x01 - data bearer
                                           frame:
                                             auth, call_type, call_speed, ?, IP len, IP␣
↪(Unicode), dialup len, dialup (Unicode),
                                             user len, user (Unicode), password len,␣
↪password (Unicode)
                                             where auth: 0x00 - normal, 0x01 - secure
                                                   call_type: 0x00 - analogue, 0x01 -␣
↪ISDN
                                                   call_speed: 0x00 - 9600, 0x01 - 14400
                     0x02 - USSD bearer
                       frame: type, service number len/IP len,service num (Unicode)/IP␣
↪(Unicode),service code len,
                              service code (Unicode)
                         where type: 0x01 - service number, 0x00 - IP
    r Get WAP settings 2 err{ 0x001d,error}
                            where: error=0x05
0x40: Security commands
?   s ???(N6150)            { 0x08, 0x00 }
?   r ???(N6150)            { 0x08 }
    s Enable extended cmds  { 0x64, cmd }
                            where cmd: 0x00: off
                                       0x01: on
                                       0x03: reset (doesn't ask for PIN again)
                                       0x04: reset (PIN is requested)
                                             In 5110 makes reset without PIN
                                       0x06: CONTACT SERVICE!!! Don't try it!
    s Reset phone settings  { 0x65, value, 0x00 }
                            where value: 0x08 - reset UI (User Interface) settings
                             0x38 - reset UI, SCM and call counters
                                        0x40 - reset test 36 in netmonitor
    r Reset phone settings  { 0x65, 0x00 }
    s Get IMEI              { 0x66 }
    r Get IMEI              { 0x66, 0x01, IMEI, 0x00}
    s (ACD Readings)?(N6150 { 0x68 }
    r (ACD Readings)?(N6150 { 0x68, ... }
    s Get Product Profile
      Settings              { 0x6a}
    r Get Product Profile
      Settings              { 0x6a, 4bytes with Product Profile Settings }
    s Set Product Profile
      Settings              { 0x6b, 4bytes with Product Profile Settings }
    r Set Product Profile
      Settings OK ?         { 0x6b }
    s Get code              { 0x6e, code }
```

(continues on next page)

(continued from previous page)

```
                            where code: see 0x08/0x0004 (no allowed code !)
    r Get code              { 0x6e, code, allowed, allowed? (sec code (text)) }
                            where code: see 0x08/0x0004
                                  allowed: 0: no
                                           1: yes
?   s ????                  { 0x74, 0x01, 0x01, 0x0e }
?   r ????                  { 0x74 }
    s Call commands         { 0x7c, block }
                            where where: command, (values)
                        command: 0x01
                        values: number(ASCII), 0x00 - makes voice call
                    command: 0x02 - answer call
                    command: 0x03 - release call
    r Call commands         { 0x7c, command }
    s Netmonitor            { 0x7e, field }
                            where: field: 00: next
                                          F0: reset
                                          F1: off
                                          F2: field test menus
                                          F3: developer menus
    s Get simlock info      { 0x8a, 0x00}
    r Get simlock info      { 0x8a, 0x00, 0x01, lockstype, locksclosed, 0x00, 0x00,␣
↪locksinfo(lock1,4,2,3), counter1,counter2,counter4,counter4,0x00 }
                            where: lockstype:   bit1,bit2,bit3,bit4 - if set, selected␣
↪lock is user lock
                                   locksclosed: bit1,bit2,bit3,bit4 - if set, selected␣
↪lock is closed
                                   counter1 - counter4: counters for locks
    s Buzzer pitch          { 0x8f, volume, hzLO, hzHI }
                            if volume and hz is 0, it's off
    r Buzzer pitch          { 0x8f}
    s ACD Readings ?        { 0x91, parameter?(0x02,0x03,0x04,0x05,0x07) }
    r ACD Readings ?        { 0x91, parameter?, value? }
?   s ???(N6150)            { 0x98, 0x00 }
?   r ???(N6150)            { 0x98, 0x00, 0x04 }
    s Get bin ringtone      { 0x9e, location }
                            where: location=0,1,etc.
    r Get bin ringtone      { 0x9e, location, error, contents... }
                            where location=0,1,etc.
                                  error=0x0a, ringtone NOT available
                                        0x00, OK
    s Set bin ringtone      { 0xa0, location, 0x00, contenst... }
                            where: location=0,1,etc.
    r Set bin ringtone      { 0xa0, location, error }
                              where location=0,1,etc.
                                    error=0x0a, ringtone NOT set
                                          0x00, ringtone set OK
?   r Get MSid              { 0xb5, 0x01, 0x2f, msid, 0x25 }
    s Get info about phone  { 0xc8, 0x01 }
    r Get info about phone  { 0xc8, 0x01, 0x00, "V ", "firmware", 0x0a, "firmware date",␣
↪0x0a, "model", 0x0a, "(c) NMP.", 0x00 }
    s Get MCU SW Checksum   { 0xc8, 0x02 }
```

(continues on next page)

(continued from previous page)

```
   r Get MCU SW Checksum   { 0xc8, 0x02, 0x00, checksum (4 bytes),0x00 }
   s DPS External SW       { 0xc7, 0x03 }
   r DSP External SW       { 0xc7, 0x03, 0x00, string,0x00 }
   s Get HW                { 0xc8, 0x05 }
   r Get HW                { 0xc8, 0x05, 0x00, HW version (4 bytes), 0x00 }
   s Get "Made" Date       { 0xc8, 0x05 }
   r Get "Made" Date       { 0xc8, 0x05, 0x00, date(4 bytes), 0x00 }
   s Get DSP Internal SW   { 0xc8, 0x09 }
   r Get DSP Internal SW   { 0xc8, 0x09, 0x00, version (1 bytes), 0x00 }
   s Get PCI version       { 0xc8, 0x0b }
   r Get PCI version       { 0xc8, 0x0b, 0x00, version, 0x00 }
   s Get system ASIC       { 0xc8, 0x0c }
   r Get system ASIC       { 0xc8, 0x0c, 0x00, string, 0x00 }
   s Get COBBA             { 0xc8, 0x0d }
   r Get COBBA             { 0xc8, 0x0d, 0x00, string, 0x00 }
   s Get PLUSSA            { 0xc8, 0x0e }
   r Get PLUSSA            { 0xc8, 0x0e, available, 0x00 }
                           where available: 0x01: not available
   s Get CCONT             { 0xc8, 0x0f }
   r Get CCONT             { 0xc8, 0x0f, available, 0x00 }
                           where available: 0x01: not available
   s Get PPM version       { 0xc8, 0x10 }
   r Get PPM version       { 0xc8, 0x10, 0x00, "V ", "firmware", 0x0a, "firmware date",␣
↪0x0a, "model", 0x0a, "(c) NMP.", 0x00 }
   s Get PPM info          { 0xc8, 0x12 }
   r Get PPM info          { 0xc8, 0x12, 0x00, PPM version ("B", "C", etc.), 0x00 }
   s Set HW version        { 0xc9, 0x05, version, 0x00 }
   s Get Product Code      { 0xca, 0x01 }
   r Get Product Code      { 0xca, 0x01, 0x00, number, 0x00 }
   s Get Order Number      { 0xca, 0x02 }
   r Get Order Number      { 0xca, 0x02, 0x00, string, 0x00 }
   s Get Prod.Ser.Number   { 0xca, 0x03 }
   r Get Prod.Ser.Number   { 0xca, 0x03, 0x00, number, 0x00 }
   s Get Basic Prod.Code   { 0xca, 0x04 }
   r Get Basic Prod.Code   { 0xca, 0x04, 0x00, number, 0x00 }
   s Set Product Code      { 0xcb, 0x01, product code, 0x00 }
   s Set Order Number      { 0xcb, 0x02, number, 0x00 }
   s Set Prod.Ser.Number   { 0xcb, 0x03, number, 0x00 }
   s Get (original ?)IMEI  { 0xcc, 0x01 }
   r Get (original ?)IMEI  { 0xcc, 0x01, IMEI, 0x00 }
   s Get Manufacture Month { 0xcc, 0x02 }
   r Get Manufacture Month { 0xcc, 0x02, 0x00, string, 0x00 }
   s Get Purchare date     { 0xcc, 0x04 }
   r Get Purchare date     { 0xcc, 0x04, 0x00, string, 0x00 }
   s Set "Made" date       { 0xcd, 0x02, string, 0x00 }
   s Make "all" phone tests{ 0xce,0x1d,0xfe,0x23,0x00,0x00}
   s Make one phone test   { 0xce,0x1d,num1,num2,num3,num4}
                           Where num1-num4: 0x02,0x00,0x00,0x00;
                                            0x04,0x00,0x00,0x00;
                                            0x08,0x00,0x00,0x00;
                                            0x10,0x00,0x00,0x00;
                                            0x20,0x00,0x00,0x00;
```

(continues on next page)

(continued from previous page)

```
                                          0x40,0x00,0x00,0x00;
                                          0x80,0x00,0x00,0x00;
                                          0x00,0x01,0x00,0x00;
                                          0x00,0x02,0x00,0x00;
                                          0x00,0x04,0x00,0x00; - "Power off"
                                            No test for "Security data"
                                          0x00,0x10,0x00,0x00;
                                          0x00,0x20,0x00,0x00;
                                          0x00,0x40,0x00,0x00;
                                          0x00,0x80,0x00,0x00;
                                          0x00,0x00,0x01,0x00;
                                          ....
                                          0x00,0x00,0x10,0x00;
    s Result of phone tests { 0xcf }
    r Result of phone tests { 0xcf, number of tests, results of next tests }
?   s ???                   { 0xd1 }
?   r ???(N5110)            { 0xd1, 0x00, 0x1d, 0x00, 0x01, 0x08, 0x00 }
    s LCD Test              { 0xd3, value }
                            where value: 0x03, 0x02 - 1'st test
                             0x03, 0x01 - 2'nd test
                                          0x02, 0x03 - clears screen
    s ACD Readings(N6150)?  { 0xd4, 0x02, 0x00, 0x02, 0x00, 0x0e, 0x01}
    r ACD Readings(N6150)?  { 0xd4, 0x02, 0x00, 0x02, 0x00, 0x0e, 0x01, ?}
    r Function of           { 0xff, 0x8c }
      0x40 msgtype not
      supported ?
0x78:
    s Status confirm        { 0x0201, 0x03 }
    r Incoming call seq1    { 0x0102 0x0e 0x03 }
    r Incoming call seq2    { 0x0102 0x7e 0x01 }
0x79:
    s CarKit enable         { 0x0201 0x01 0x62 0x00 }
    r CarKit enabled        { 0x0201 0x02 0x06 0x00 "V " {version} "\nHFU"
                                     0x00 }
0x7a: settings
    r Set setting           { 0x01eb, number, 0x00 }
    s Set setting           { 0x01ec, number, contents }
                            where for number:
                              0x02 (startup text) : 0x00, text (Unicode)
                              0x15 (startup logo) : 0x00, 0x00, 0x00, 0x04,
                                       0xc0, 0x02, 0x00, height, 0xc0, 0x03, 0x00,␣
↪width,
                        0xc0, 0x04, 0x03, 0x00, {bitmap} }
                                 where width, height, {bitmap}: see 0x7a/0x01ed 0x15
    s Get setting           { 0x01ee, number}
                            where number: 0x01 - 0x1e
                              0x02: startup text
                              0x15: startup logo
                              0x1c: security code
    r Get setting           { 0x01ed,number, 0x00, contents}
                            where for number:
                              0x02 (startup text) : 0x00, text (Unicode)
```

(continues on next page)

(continued from previous page)

```
                                0x15 (startup logo) : 0x00, 0x00, 0x00, 0x04,
                                          0xc0, 0x02, 0x00, height, 0xc0, 0x03, 0x00,␣
↪width,
                                          0xc0, 0x04, 0x03, 0x00, {bitmap} }
                                  where height: 60 (0x3c) or 65
                                        width:  96 (0x60)
                                        {bitmap}: like other bitmaps but pixels
                                                  placed vertically.
                                0x1c (security code): {code(ascii)}, 0x00
0x7f: Acknowledge(FBUS/IRDA){+type, seq }
      Acknowledge(MBUS)...
0xd0:
    s Power on message seq1 {+04 }
    r Power on message seq1 {+05 }
0xd1:
    s Get HW&SW version     { 0x0003, 0x00 }
0xd2:
    r Get HW&SW version     { 0x0003 "V " "firmware\n" "firmware date\n"
                              "model\n" "(c) NMP." }
0xf4: Power on message seq 2
```

13.7 Nokia 6210/6310, CARC91, PC Experiment

Author:
Jens Bennfors

Company
AB Indevia

Date:
2002-04-09

13.7.1 Introduction

The purpose of this experiment is to gain understanding about how Nokias commands for handsfree works in a way that can be of use in the construction of Com.n.sense. The means available is a Nokia 6210, a Nokia 6310, a HFU-2 CARC91 and a PC with a LabVIEW program installed.

13.7.2 Setup

I have connected the phone to a Nokia original handsfree (CARC91). I then use the PC for listening to the data communication between the phone and CARC91. I also send the frames directly from the PC to the phone.

13.7.3 Nokia 6210

Phone connected to PC

Initiation

1F0004 D0 0001 04 00CE
Power up from PC

1F0004 D0 0001 04 01CF
Power up from PC

1F0400 D0 0001 05 10DF
Power up from phone

1F0004 79 0005 0201 0164 00 0203
Enable carkit mode from PC

1F0004 79 0005 0201 0164 00 0302
Enable carkit mode from PC

1F0400 7F 0367
Ack from phone

1F0004 79 0012 0201 0206 0056 2030 372E 3030 0A48 4655 3200 044F
HFU-2 Version

1F0400 7F 0460
Ack from phone

1F0400 78 0004 0102 0801 117C
Status 0x08, 0x01 from phone

1F0400 DA 0002 0002 12D3
Type => 0xDA, data => 0x00, 0x02

1F0004 79 0005 0201 0164 00 0504
Enable carkit mode from PC

1F0004 79 0005 0201 0164 00 0607
Enable carkit mode from PC

1F0400 7F 0662
Ack from phone

1F0004 78 0003 0201 0307 67
Status confirm from PC

1F0004 78 0003 0201 0308 68
Status confirm from PC

1F0400 7F 086C
Ack from phone

The phone enters the profile ”handsfree” when the frame carkit enable is sent. It sends out an unknown status frame 0x08, 0x01.

Incoming call

1F0400 78 0004 0102 0701 197B
Status 0x07, 0x01 from phone

1F0400 78 0004 0102 0E03 1A73
Status 0x0E, 0x03 from phone

Status type 0x07 with status 0x01 means mute external audio equipment. Status type 0x0E with status 0x03 means audio amplifier on.

Connected

The phone doesn't send out anything when a call has been set up.

Initiation with connected phone

1F0004 D0 0001 04 00CE
Power up from PC

1F0400 D0 0001 05 1BD4
Power up from phone

1F0004 79 0005 0201 0164 0001 00
Enable carkit mode from PC

1F0400 7F 0165
Ack from phone

1F0400 78 0004 0102 0E03 1C75
Status 0x0E, 0x03 from phone

1F0400 78 0004 0102 0701 1D7F
Status 0x07, 0x01 from phone

1F0004 79 0012 0201 0206 00 5620 3037 2E30 300A 4846 5532 00 0249
HFU-2 Version from PC

1F0400 7F 0266
Ack from phone

1F0400 78 0004 0102 0801 1E73
Status 0x08, 0x01 from phone

1F0004 79 0005 0201 0164 0003 02
Enable carkit mode from PC

1F0400 7F 0367
Ack from phone

1F0400 78 0004 0102 0E03 1F76
Status 0x0E, 0x03 from phone

1F0400 78 0004 0102 0701 2042
Status 0x07, 0x01 from phone

1F0004 78 0003 0201 03 0464
Status confirm from PC

1F0400 7F 0460
Ack from phone

Disconnected

1F04 0078 0004 0102 0700 2142
Status 0x07, 0x00

Incoming SMS

FCF0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0
Initiation of bit length from phone

1F0400 78 0004 0102 0E03 254C
Status 0x0E, 0x03 from phone

F0F0F0F0
Initiation of bit length from phone

Phone connected to CARC91

Initiation

1F0004 D0 0001 04 00CE
Power up from HFU-2

1F0400 D0 0001 05 02CD
Power up from phone

1F0004 79 0005 0201 0164 00 0100
Enable carkit mode from HFU-2

1F0400 7F 0165
Ack from phone

1F0004 79 0012 0201 0206 0056 2030 372E 3030 0A48 4655 3200 0249
HFU-2 Version

1F0400 7F 0266
Ack from phone

1F0400 78 0004 0102 0801 036E
Status 0x08, 0x01

1F0004 79 0005 0201 0164 00 0302
Enable carkit mode from HFU-2

1F0400 7F 0367
Ack from phone

1F0400 78 0004 0102 0801 036E
Status 0x08, 0x01

1F0004 7F 0367
Ack from HFU-2

1F0400 DA 0002 0002 04C5
Status type => 0xDA, data => 0x00, 0x02

1F0004 7F 0460
Ack from HFU-2

1F0400 78 0004 0102 0E03 056C
Status 0x0E, 0x03

1F0004 7F 0561
Ack from HFU-2

1F0004 78 0003 0201 03 0464
Status confirm from HFU-2

1F0400 7F 0460
Ack from phone

1F0400 78 0004 0102 0E00 066C
Status 0x0E, 0x00

1F0004 7F 0662
Ack from HFU-2

1F0004 78 0003 0201 03 0565
Status confirm from HFU-2

1F0400 7F 0561
Ack from phone

Incoming call

1F0400 78 0004 0102 0701 1173
Status 0x07, 0x01

1F0004 7F 1175
Ack from HFU-2

1F0400 78 0004 0102 0E03 127B
Status 0x0E, 0x03

1F0004 7F 1276
Ack from HFU-2

1F0004 78 0003 0201 03 0868
Status confirm from HFU-2

1F0400 7F 086C
Ack from phone

Connected

The phone doesn't send out anything when a call has been set up.

Initiation with connected phone

1F0004 D0 0001 04 00CE
Power up from HFU-2

1F0400 D0 0001 05 1AD5
Power up from phone

1F0004 79 0005 0201 0164 00 0100
Enable carkit mode from HFU-2

1F0400 7F 0165
Ack from phone

1F0400 78 0004 0102 0E03 1B72
Status 0x0E, 0x03

1F0004 79 0012 0201 0206 0056 2030 372E 3030 0A48 4655 3200 0249
HFU-2 Version

1F0400 7F 0266
Ack from phone

1F0004 79 0005 0201 0164 00 0302
Enable carkit mode from HFU-2

1F0400 7F 0367
Ack from phone

1F0400 78 0004 0102 0E03 1B72
Status 0x0E, 0x03

1F0004 7F 1B7F
Ack from HFU-2

1F0400 78 0004 0102 0801 1C71
Status 0x08, 0x01

1F0004 78 0003 0201 03 0464
Status confirm from HFU-2

1F0400 7F 0460
Ack from phone

1F0400 78 0004 0102 0801 1C71
Status 0x08, 0x01

1F0004 7F 1C78
Ack from HFU-2

1F0400 78 0004 0102 0E03 1D74
Status 0x0E, 0x03

1F0004 7F 1D79
Ack from HFU-2

1F0400 78 0004 0102 0701 1E7C
Status 0x07, 0x01

1F0004 78 0003 0201 03 0565
Status confirm from HFU-2

1F0400 7F 0561
Ack from phone

1F0400 78 0004 0102 0701 1E7C
Status 0x07, 0x01

1F0004 7F 1E7A
Ack from HFU-2

1F0400 78 0004 0102 0701 1F7D
Status 0x07, 0x01

1F0004 7F 1F7B
Ack from phone

1F0400 DA 0002 0002 20E1
Typ => 0xDA, data => 0x00. 0x02

1F0004 7F 2044
Ack from HFU-2

Disconnected

1F0400 78 0004 0102 0700 1774
Status 0x07, 0x00

1F0004 7F 1773
Ack from HFU-2

1F0400 78 0004 0102 0E00 1872
Status 0x0E, 0x00

1F0004 7F 187C
Ack from HFU-2

1F0004 78 0003 0201 03 0B6B
Status confirm from HFU-2

1F0400 7F 0B6F
Ack from phone

Incoming SMS

1F0400 78 0004 0102 0E03 076E
Status 0x0E, 0x03

1F0004 7F 0763
Ack from HFU-2

1F0004 78 0003 0201 03 0666
Status confirm from HFU-2

1F0400 7F 0662
Ack from phone

1F0400 78 0004 0102 0E00 0862
Status 0x0E, 0x00

1F0004 7F 086C
Ack from HFU-2

1F0004 78 0003 0201 03 0767
Status confirm from HFU-2

1F0400 7F 0763
Ack from phone

Button pushed

1F0400 78 0004 0102 0E03 0960
Status 0x0E, 0x03

1F0004 7F 096D
Ack from HFU-2

1F0004 78 0003 0201 03 0868
Status confirm from HFU-2

1F0400 7F 086C
Ack from phone

1F0400 78 0004 0102 0E00 0A60
Status 0x0E, 0x00

1F0004 7F 0A6E
Ack from HFU-2

1F0004 78 0003 0201 03 0969
Status confirm from HFU-2

1F0400 7F 096D
Ack from phone

13.7.4 Nokia 6310

Phone connected to PC

Initiation

1F0004 D0 0001 04 02CC
Power up from PC

1F0400 D0 0001 05 0DC2
Power up from phone

1F0004 79 0005 0201 0164 00 0C0D
Enable carkit mode from PC

1F0400 7F 0C68
Ack from phone

1F0400 78 0004 0128 0B00 0E4B
Status 0x0B, 0x00 from phone

1F0004 79 0012 0201 0206 0056 2030 372E 3030 0A48 4655 3200 0D46
HFU-2 version from PC

1F0400 7F 0E6A
Ack from phone

1F0400 DA 0004 0028 0000 0FE2
?

1F0004 79 0005 0201 0164 00 1716
Enable carkit mode from PC

1F0400 7F 1773
Ack from phone

1F0400 78 0004 0128 0B00 1055
Status 0x0B, 0x00 from phone

1F0004 78 0003 0201 03 1878
Status confirm from PC

1F0400 7F 1A7E
Ack from phone

An unknown status frame (0x0B) is sent by the phone.

Incoming call

1F0400 78 0004 0128 0701 0D45
Status 0x07, 0x01 from phone

1F0400 78 0004 0128 0E01 0F4E
Status 0x0E, 0x01 from phone

1F0400 78 0004 0128 0A00 1054
Status 0x0A, 0x00 from phone

1F0400 78 0004 0128 0901 1157
Status 0x09, 0x01 from phone

Byte 8 in the status frames is some kind of ID number. 0x28 is the ID for 6310. Status 0x0A, 0x09 is unknown.

Connected

The phone doesn't send out anything when a call has been set up. This might be because the profile "handsfree" is lost when ack isn't sent.

Initiation with connected phone

1F0004 79 0012 0201 0206 0056 2030 372E 3030 0A48 4655 3200 1C57
HFU-2 version from PC

1F0400 7F 1C78
Ack from phone

1F0400 78 0004 0128 0E02 1A58
Status 0x0E, 0x02

1F0400 78 0004 0128 0A00 1B5F
Status 0x0A, 0x00

1F0400 78 0004 0128 0900 1C5B
Status 0x09, 0x00

1F0400 78 0004 0128 0701 1D55
Status 0x07, 0x01

1F0004 D0 0001 04 00CE
Power up from HFU-2

1F0400 D0 0001 05 74BB
Power up from phone

1F0004 79 0005 0201 0164 00 0100
Enable carkit mode from HFU-2

1F0400 7F 0165
Ack from phone

1F0004 79 0012 0201 0206 0056 2030 372E 3030 0A48 4655 3200 0249
HFU-2 Version

1F0400 7F 0266
Ack from phone

1F0400 78 0004 0128 0E01 7534
Status 0x0E, 0x01

1F0004 79 0005 0201 0164 00 0302
Enable carkit mode from HFU-2

1F0400 7F 0367
Ack from phone

1F0400 78 0004 0128 0E01 7534
Status 0x0E, 0x01

1F0004 7F 7511
Ack from HFU-2

1F0400 78 0004 0128 0A01 7633
Status 0x0A, 0x01

1F0004 7F 7612
Ack from HFU-2

1F0400 78 0004 0128 0901 7731
Status 0x09, 0x01

1F0004 7F 7713
Ack from HFU-2

1F0400 78 0004 0128 0701 7830
Status 0x07, 0x01

1F0004 7F 781C
Ack from HFU-2

1F0400 78 0004 0128 0E01 7938
Status 0x0E, 0x01

1F0004 7F 791D
Ack from HFU-2

1F0004 78 0003 2801 03 044E
Status confirm from HFU-2

1F0400 7F 0460
Ack from phone

1F0400 DA 0004 0028 0000 7A97
Type => 0xDA, data => 0x0028, 0x0000

1F0004 7F 7A1E
Ack from HFU-2

1F0400 78 0004 0128 0E01 7B3A
Status 0x0E, 0x01

1F0004 7F 7B1F
Ack from HFU-2

1F0400 78 0004 0128 0A00 7C38
Status 0x0A, 0x00

1F0004 78 0003 2801 03 054F
Status confirm from HFU-2

1F0400 7F 0561
Ack from phone

1F0400 78 0004 0128 0A00 7C38
Status 0x0A, 0x00

1F0004 7F 7C18
Ack from HFU-2

1F0400 78 0004 0128 0700 7D34
Status 0x07, 0x00

1F0004 7F 7D19
Ack from HFU-2

1F0400 78 0004 0128 0E00 7E3E
Status 0x0E, 0x00

1F0004 7F 7E1A
Ack from HFU-2

1F0004 78 0003 2801 03 064C
Status confirm from HFU-2

1F0400 7F 0662
Ack from phone

Disconnected

No response. Probably because phone has lost the profile “handsfree”.

Incoming SMS

1F0400 78 0004 0128 0E01 0849
Status 0x0E, 0x01

1F0400 78 0004 0128 0A00 094D
Status 0x0A, 0x00

1F0400 78 0004 0128 0901 0A4C
Status 0x09, 0x01

Phone connected to CARC91

Initiation

1F0004 D0 0001 04 00CE
Power up from HFU-2

1F0400 D0 0001 05 2DE2
Power up from phone

1F0004 79 0005 0201 0164 00 0100
Enable carkit mode from HFU-2

1F0400 7F 0165
Ack from phone

1F0004 79 0012 0201 0206 0056 2030 372E 3030 0A48 4655 3200 0249
HFU version from HFU-2

1F0400 7F 0266
Ack from phone

1F0004 79 0005 0201 0164 00 0302
Enable carkit mode from HFU-2

1F0400 7F 0367
Ack from phone

1F0400 78 0004 0128 0E00 2E6E
Status 0x0E, 0x00

1F0004 7F 2E4A
Ack from HFU-2

1F0004 78 0003 2801 03 044E
Status confirm from HFU-2

1F0400 7F 0460
Ack from phone

1F0400 DA 0004 0028 0000 2FC2
?

1F0004 7F 2F4B
Ack from HFU-2

Incoming call

1F0400 78 0004 0128 0701 3078
Status 0x07, 0x01

1F0004 7F 3054
Ack from HFU-2

1F0400 78 0004 0128 0701 3179
Status 0x07, 0x01

1F0004 7F 3155
Ack from HFU-2

1F0400 78 0004 0128 0E01 3273
Status 0x0E, 0x01

1F0004 7F 3256
Ack from HFU-2

1F0400 78 0004 0128 0A00 3377
Status 0x0A, 0x00

1F0004 78 0003 2801 03 054F
Status confirm from HFU-2

1F0400 7F 0561
Ack from phone

1F0400 78 0004 0128 0A00 3377
Status 0x0A, 0x00

1F0004 7F 33 57
Ack from HFU-2

1F0400 78 0004 0128 0901 3472
Status 0x09, 0x01

1F0004 7F 3450
Ack from HFU-2

Connected

1F0400 78 0004 0128 0E01 3574
Status 0x0E, 0x01

1F0004 7F 3551
Ack from HFU-2

1F0400 78 0004 0128 0A01 3673
Status 0x0A, 0x01

1F0004 78 0003 2801 03 064C
Status confirm from HFU-2

1F0400 7F 0662
Ack from phone

1F0400 78 0004 0128 0A01 3673
Status 0x0A, 0x01

1F0004 7F 3652
Ack from HFU-2

1F0400 78 0004 0128 0A00 3773
Status 0x0A, 0x00

1F0004 7F 3753
Ack from HFU-2

1F0400 78 0004 0128 0900 387F
Status 0x09, 0x00

1F0004 7F 385C
Ack from HFU-2

1F0400 78 0004 0128 0A01 397C
Status 0x0A, 0x01

1F0004 7F 395D
Ack from HFU-2

1F0400 78 0004 0128 0901 3A7C
Status 0x09, 0x01

1F0004 7F 3A5E
Ack from HFU-2

Initiation with connected phone

1F0004 D0 0001 04 00CE
Power up from HFU-2

1F0400 D0 0001 05 5996
Power up from phone

1F0004 79 0005 0201 0164 00 0100
Enable carkit mode from HFU-2

1F0400 7F 0165
Ack from phone

1F0004 79 0012 0201 0206 0056 2030 372E 3030 0A48 4655 3200 0249
HFU-2 Version

1F0400 7F 0266
Ack from phone

1F0400 78 0004 0128 0E01 5A1B
Status 0x0E, 0x01

1F0004 79 0005 0201 0164 00 0302
Enable carkit mode from HFU-2

1F0400 7F 0367
Ack from phone

1F0400 78 0004 0128 0E01 5A1B
Status 0x0E, 0x01

1F0004 7F 5A3E
Ack from HFU-2

1F0400 78 0004 0128 0A01 5B1E
Status 0x0A, 0x01

1F0004 7F 5B3F
Ack from HFU-2

1F0400 78 0004 0128 0901 5C1A
Status 0x09, 0x01

1F0004 7F 5C38
Ack from HFU-2

1F0400 78 0004 0128 0701 5D15
Status 0x07, 0x01

1F0004 7F 5D39
Ack from HFU-2

1F0004 78 0003 2801 0305 4F
Status confirm from HFU-2

1F0400 7F 0561
Ack from phone

1F0400 DA 0004 0028 0000 5EB3
?

1F0004 7F 5E3A
Ack from HFU-2

Disconnected

1F0400 78 0004 0128 0E01 3B7A
Status 0x0E, 0x01

1F0004 7F 3B5F
Ack from HFU-2

1F0400 78 0004 0128 0A00 3C78
Status 0x0A, 0x00

1F0004 78 0003 2801 03 074D
Status confirm from HFU-2

1F0400 7F 0763
Ack from phone

1F0400 78 0004 0128 0A00 3C78
Status 0x0A, 0x00

1F0004 7F 3C58
Ack from HFU-2

1F0400 78 0004 0128 0700 3D74
Status 0x07, 0x00

1F0004 7F 3D59
Ack from HFU-2

1F0400 78 0004 0128 0E00 3E7E
Status 0x0E, 0x00

1F0004 7F 3E5A
Ack from HFU-2

1F0004 78 0003 2801 0308 42
Status confirm from HFU-2

1F0400 7F 086C
Ack from phone

Incoming SMS

1F0400 78 0004 0128 0E01 6627
Status 0x0E, 0x01

1F0004 7F 6602
Ack from HFU-2

1F0004 78 0003 2801 03 064C
Status confirm from HFU-2

1F0400 7F 0662
Ack from phone

1F0400 78 0004 0128 0E00 6727
Status 0x0E, 0x00

1F0004 7F 6703
Ack from HFU-2

1F0004 78 0003 2801 03 074D
Status confirm from HFU-2

1F0400 7F 0763
Ack from phone

Button pushed

1F0400 78 0004 0128 0E01 0948
Status 0x0E, 0x01

1F0004 7F 096D
Ack from HFU-2

1F0004 78 0003 2801 03 064C
Status confirm from HFU-2

1F0400 7F 0662
Ack from phone

1F0400 78 0004 0128 0E00 0A4A
Status 0x0E, 0x00

1F0004 7F 0A6E
Ack from HFU-2

1F0004 78 0003 2801 03 074D
Status confirm from HFU-2

1F0400 7F 0763
Ack from phone

13.7.5 Result

Important things to consider when designing a program for Com.n.sense that is to work with 6310.

- 6310 sends out status 0x0E, 0x01 when speaker should be enabled
- HFU-2 version has to be sent in order for 6310 to switch to profile ”Handsfree”.
- Status 0x0A might say weather the phone is ringing or connected. Only 6310 send this status.
- Status confirm should be sent when status 0x0E is received.

13.8 TDMA 5120

Eduardo Spremolla at gnokii-users@mail.freesoftware.fsf.org

After playing a while with my 5120i y find some use full frames:

13.8.1 got from sneefing in Logomanger the get startup logo

request:

```
40 {0x07, 0x07, 0x08, section} section goes from 1 to 6
```

answer:

```
dd {+0x01, 0x00, 0x07, 0x08, (84 bytes => 84 cols x 8 bits bit0 first row )
```

Can’t figure out how to modify 6110 code to get & put the logo, not in a hi value to me now.

13.8.2 got key press working

As stated in http://www.flosys.com/tdma/n5160.html

with frame: key-press:

```
D1 {+00 01 50 00 01 KY}
```

this seems to press the key for a while. No release needed

key-release:

```
D1 {+00 01 50 00 00 KY}
```

keep the key press => got speedee dial:

```
D1 {+00 01 50 00 02 00 KY}
```

13.8.3 get memory

the getmemory::
40 {+00 00 07 11 00 10 00 mem}

get phonebook with the phone in bcd, but it seems to be a way to read chunks of memory with different numbers in the 6 place. in particular:

get configuration pins:

```
40 {+0x00, 0x00, 0x07, 0x11, 0x00, 0x0f, 0x00, 0x00 }
```

get security code:

```
40 {+0x00, 0x00, 0x07, 0x11, 0x00, 0x09, 0x00, 0x00 }
```

get NAM data

```
40 {+0x00, 0x00, 0x07, 0x11, 0x00, 0x08, 0x00, nam# }
```

that last answers with:

```
dd {+01 00 11 00 08 00 00,
```

03 04
home sys id

01 4d
primary paggin channel

02 c4
seconda paggin channel

88 88 88 88 88
own #

09 63 c2 09 03 00 0b
unknown

0a
group id

01
Access method

01
local option

0f
overload class

20 41 43 41 45 00 00 00 00 00 00 00 00 00 00 00
alpha tag

b3 4d
unknown

01
NAM status

11 11 11 11 11 00 00 00 00 00 00 00 00
unknown

00 00 00 00 00 00 01 00 00 00 01 36
unknown

01 4d
dedicate ch

01 4e
dedicate B ch

14
dedicate ch #

14
dedicate B ch #

00
msg center # len

00
msg center in flag

00 00 00 00 00 00 00 00 00 00 00 00 00 00
msg center #

08 01 80 70 8f dd 00 ef 00 00 00 00 00 00 00 00
unknown

00 00 00 00 00
gate way #

00 00 00
unknown

More interesting (and dangerous) is than the 07 10 sequence in place of 07 11 in the request change the command from read to write.be care full!!! I almost ruin my 5125 with a 40 {+0x00, 0x00, 0x07, 0x10, 0x00, 0x08, 0x00, 0x01 } frame , since the frame is ok, but the phone the write info from an area of the buffer that I did not send!!!!

OK so far. Still looking for how to handle SMS……

13.9 SAMSUNG Organizer AT commands

13.9.1 Get organizer information

Invocation:

```
AT+ORGI?
```

Example:

```
AT+ORGI?
+ORGI: 84,400,30,100,30
OK
```

Return 5 values:

par1
(84) Busy entries (1 to par1 of par2 possibles entries)

par2

(400) Max possible entries

par3

(30) Unknown

par4

(100) Unknown

par5

(30) Unknown

13.9.2 Get organizer details

Invocation:

```
AT+ORGR=number
```

Get organizer details for index entry “number” Returns 24 values:

Example 1:

```
AT+ORGR=10
+ORGR: 161,1,"Comprar lagrimas artificiales","Farmacia",2,4,2009,9,0,2,4,2009,9,10,
↪"Farmacia",1,1,0,3,,,29,1,2010
OK
```

Example 2:

```
AT+ORGR=15
+ORGR: 67,2,,"Laura Santiesteban Cabrera",3,11,2009,9,0,,,,,,,1,3,0,4,,,,,
OK
```

Example 3:

```
AT+ORGR=19
+ORGR: 205,3,,"Cemento",13,3,2009,10,35,13,3,2009,,,,1,3,0,0,1,0,,,
OK
```

Example 4:

```
AT+ORGR=23
+ORGR: 235,4,"Curso","Averiguar",13,3,2009,9,50,13,3,2009,9,59,,1,1,0,,,,,,
OK
```

+ORGR: AT+ORGR answer header

par01

Pointer to real memory position

par02

Organizer entry type (1=appointments, 2=aniversaries, 3=tasks, 4=miscellany)

If par02 =1, appointment entry type

par03

Organizer entry short name

par04
Organizer entry detailed description

par05
Start day

par06
Start month

par07
Start year

par08
Start hour

par09
Start minute

par10
End day

par11
End month

par12
End year

par13
End hour

par14
End minute

par15
Location

par16
Alarm flag (0=no, 1=yes)

par17
Alarm time unit (1=minutes, 2=hours, days, 4=weeks)

par18
Alarm items quantity

par19
Alarm repeat flag (0 or empty=no, 2=yes)

par20
Empty

par21
Empty

par22
Repeat until day

par23
Repeat until month

par24
Repeat until year

If par02 = 2, anniversary entry type

par03
Empty

par04
Occasion name

par05
Alarm day

par06
Alarm month

par07
Alarm year

par08
Alarm hour

par09
Alarm minutes

par10
Empty

par11
Empty

par12
Empty

par13
Empty

par14
Empty

par15
Empty

par16
Alarm flag (0=no, 1=yes)

par17
Alarm time unit (1=minutes, 2=hours, days, 4=weeks)

par18
Alarm items quantity

par19
Repeat each year (0=no, 4=yes)

par20
Empty

par21
Empty

par22
Empty

par23
Empty

par24
Empty

If par02 = 3, task entry type

par03
Empty

par04
Task name

par05
Start day

par06
Start month

par07
Start year

par08
Alarm hour

par09
Alarm minute

par10
Due day

par11
Due month

par12
Due year

par13
Empty

par14
Empty

par15
Empty

par16
Alarm flag (0=no, 1=yes)

par17
Alarm time unit (1=minutes, 2=hours, days, 4=weeks)

par18
Alarm items quantity

par19
Empty

par20
Task priority (1=high, 2=normal, 3=low)

par21
Task status (0=undone, 1=done)

par22
Empty

par23
Empty

par24
Empty

If par02 = 4, miscellany entry type

par03
Entry name

par04
Details

par05
Start day

par06
Start month

par07
Start year

par08
Start hour

par09
Start minutes

par10
End day

par11
End month

par12
End year

par13
End hour

par14
End minutes

par15
Empty

par16
Alarm flag (0=no, 1=yes)

par17
Alarm time unit (1=minutes, 2=hours, days, 4=weeks)

par18
Alarm items quantity

par19
Empty

par20
Empty

par21
Empty

par22
Empty

par23
Empty

par24
Empty

13.9.3 Write organizer entry

Invocation:

```
AT+ORGW=par0,par1,par2...par24
```

Write organizer entry in memory location par0

If par0=65535 then locate next empty entry on memory

Example:

```
AT+ORGW=65535,0,4,"p2","p2",14,3,2009,2,23,14,3,2009,3,23,,0,0,0,,,,,,
+ORGW: 253,253
OK
```

par1 to par24 has the same significance than in the AT+ORGR command

13.9.4 Delete organizer entry

Invocation:

```
AT+ORGD=number
```

Delete organizer entry of index “number”

Example:

```
AT+ORGD=21

OK
```

13.9.5 Notes

Read command use index reference.

Write command uses index and direct memory reference with special 65535 value to locate empty memory position.

Delete command use direct memory reference, index are automatically reorganized.

Hint: After create or delete an organizer entry, reread full information to update index table.

13.10 SAMSUNG GT calendar AT commands

13.10.1 Calendar Entries

`AT+SSHT=1` - selects the Organizer->Calendar->Appointment entries (Spotkania in Polish version)

`AT+SSHT=2` - selects the Organizer->Calendar->Anniversary entries (Rocznice in Polish version)

`AT+SSHT=5` - selects the Organizer->Calendar->Holiday entries (Święta in Polish version)

`AT+SSHT=6` - selects the Organizer->Calendar->Important entries (Ważne in Polish version)

`AT+SSHT=7` - selects the Organizer->Calendar->Private entries (Prywatne in Polish version)

After selection of type, we can read all items:

```
AT+SSHR=0
+SSHR:5,"5,test1","0,","0,",2010,5,12,2010,5,12,21,49,22,49,0,0,0,0,2010,5,30,,
+SSHR:3,"1,x","0,","0,",2010,6,2,2010,6,3,0,0,0,0,0,0,0,0,2010,5,30,,
+SSHR:1,"9,event1234","0,","0,",2010,6,7,2010,6,7,7,0,8,59,0,0,0,0,2010,5,30,,
+SSHR:4,"7,test123","0,","0,",2010,6,14,2010,6,14,21,37,22,37,0,0,0,0,2010,5,30,,
+SSHR:2,"7,Meeting","0,","0,",2010,6,15,2010,6,15,8,0,8,59,0,0,0,0,2010,5,30,,
OK
```

Or just read a single item:

```
AT+SSHR=1
+SSHR:1,"9,Event 123","0,","0,",2010,6,7,2010,6,7,7,0,8,59,0,0,0,0,2010,5,30,,
OK
```

Getting status (the last number appears to be number of notes):

```
AT+SSHR=?
+SSHR:100,15,100,15,"1000000",2008,2024,5
OK
```

You can also add or modify an item:

```
AT+SSHW="7,event01","16,details of event","5,where",2010,06,03,2010,06,04,12,31,13,42,0,
↪0,0,0,2010,05,31,,,0
```

It seems, that the last number in the above record specifies whether it is addition of a new record (0), or modification of the old record (then the number is the position of the item, as the first number listed after `AT+SSHR=0`). e.g.:

```
AT+SSHW="13,event1234 new","0,","0,",2010,06,07,2010,06,07,07,00,08,59,0,0,0,0,2010,05,
↪30,,,1
```

Please note, that the format for writing is somehow different, than for reading - hour and minutes must be in two-digit form! The text fields (as shown above) are formatted in the following way: "number_of_characters_in_string,string" In all items above the first string is the name of event, the second string - details of event, the third one - place of event. The numeric fields encode start date (year,month,day), end date (year, month, day), start time (hour,minutes), end time (hour, minutes), four unknown to me (yet?) values, date of creation? (year month day) - the meaning of this date is not sure for me yet.

To delete entries:

```
AT+SSHD=1
OK
```

13.10.2 Task Entries

There is yet another type, that can be selected by `AT+SSHT=3` This is Organizer->Task:

```
AT+SSHT=3
OK
AT+SSHR=0
+SSHR:1,"10,Test event","10,2010-06-05",60823,11,25,60823,11,26,0,0,0,0,0,0,0,0,0
OK
```

Please note, that the format of output is different, when you read the specific task:

```
AT+SSHR=1
+SSHR:1,"10,Test event","12,Some details",2010,6,3,2010,6,5,1,2010,6,4,10,11,0,2,0
```

You can similarly add a new task:

```
AT+SSHW="9,New task1","10,0123456789",2010,06,21,2010,06,30,1,2010,06,27,08,07,0,2,0,0
+SSHW:2
OK
```

Read it back:

```
AT+SSHR=2
+SSHR:2,"9,New task1","10,0123456789",2010,6,21,2010,6,30,1,2010,6,27,8,7,0,2,0
OK
```

And modify:

```
AT+SSHW="9,New task1","11,New details",2010,06,21,2010,06,30,1,2010,06,27,08,07,0,2,0,2
+SSHW:2
OK
AT+SSHR=2
+SSHR:2,"9,New task1","11,New details",2010,6,21,2010,6,30,1,2010,6,27,8,7,0,2,0
OK
```

To delete entries:

```
AT+SSHT=3
OK
AT+SSHR=0
+SSHR:1,"10,Test
event","10,2010-06-05",60823,11,25,60823,11,26,0,60823,11,26,0,0,0,0,0
+SSHR:2,"9,New task1","10,2010-06-30",60823,11,25,60823,11,26,0,60823,11,26,0,0,0,0,0
OK
AT+SSHD=1
OK
AT+SSHR=0
+SSHR:2,"9,New task1","10,2010-06-30",60823,11,25,60823,11,26,0,0,0,0,0,0,0,0,0
OK
```

13.10.3 Memo Notes

The memo notes are accessible via AT+OMM??? commands:

```
AT+OMMI?
+OMMI:4,100,100
```

We found, that we have 4 memos

You can add a note:

```
AT+OMMW=0,"This is a note"
+OMMW:6
OK
```

You can read it:

```
AT+OMMR=6
+OMMR:"This is a note"
OK
```

You can modify it:

```
AT+OMMW=6,"This is a new modified note"
+OMMW:6
OK
AT+OMMR=6
+OMMR:"This is a new modified note"
OK
```

To delete entries:

```
AT+OMMR=3
+OMMR:"Note number 3"
OK
AT+OMMD=3
OK
AT+OMMW=3,"New note number 3"
+CME ERROR:29

ERROR
```

13.11 Sonim AT Commands

Filesystem access:

```
at*list=<path> - list directory content
                                    (0=file, 1=subdirectory)
at*mkdir=<path> - make directory
at*rmdir=<path> - remove directory
at*remove=<path> - remove file
at*move=<srcpath>,<dstpath> - ? copy (move?) files
at*startul=<srcpath> - prepare file to upload (from phone)
```

(continues on next page)

(continued from previous page)

```
                                    returned data:
                                    *STARTUL: <filesize_in_bytes>
at*startdl=<dstpath>,<filesize> - prepare file to download (to phone)
at*get - get base64 coded data chunk
                                    returned data:
                                    *GET: <chunklen>,<data>
at*get - get base64 coded data chunk
                                    returned data:
                                    *GET: <chunklen>,<data>
at*put=<no>,<len>,<data>,<chck> - put base64 coded data chunk
                                    (no is chunk number, starting from 0)
                                    (len is chunk length)
                                    (last 4 characters is checksum ?)
at*end - end/finish file transfer operation
-------------------
at*syph=?,?.?,<path> - ? (add downloaded record to phonebook?)
                                    at*syph=0,1,%d,%s
                                    EXAMPLE:
AT*SYPH=0,1,74,/app/dir/tmp.dat
at*sysm=0,1,%d - ? SMS handling

---
Phone has at least two directories from root, /app and /app3 .
at*list=/ gives error.
```

13.12 MTK AT Commands

13.12.1 VCard access

Read vcard, first 1 is READ command, second 1 is memory position:

```
AT+EVCARD=1,1
+EVCARD:
↪"0043003a005c00520065006300650069007600650064005c007e007600630061007200640005f0072002e007600630066
↪"
OK
```

HEX UCS2 temporary file name which we must read for VCARD

13.12.2 Filesystem access

Change operation mode to obtain access to filesystem operations:

```
AT+ESUO=3
OK
```

Change directory to root folder:

```
AT+EFSF=3
OK
```

Read file with name from +EVCARD reply:

```
AT+EFSR=
↪"0043003a005c00520065006300650069007600650064005c007e007600630061007200640005f0072002e007600630066
↪"
+EFSR: 1, 1, 168,

↪"424547494E3A56434152440D0A56455253494F4E3A322E310D0A4E3B43484152534543D5554462D383B454E434F44494E47
↪"
OK
```

(1, 1, 168) = (<MEM POSITION>, <EOF FLAG>, <HEXLIFIED VCARD LEN>)

Change operation mode to compatible:

```
AT+ESUO=4
OK
```

13.13 m-obex protocol used by some Samsung mobiles

This document is copied from <http://code.google.com/p/samsyncro/wiki/mobex> and extended.

13.13.1 Introduction

This is an attempt to document the m-obex protocol. It is a obex-variation by Samsung used to exchange PIM data and files over bluetooth.

This documentation is by no means complete but is only a reference for the samsyncro implementation. As I don't know the obex protocol I can't say in which parts it differs from the standard-obex. The only thing I found strange is the fact, that you will always get 0xA0 as a response. Which means Ok, success in obex. If there was an error you will find it's error code in the 0x42 header. If this is a normal behavior: Why are there so many response codes defined?

The information about the protocol was gained by listening to the transferred data from Samsungs New PC Studio to a SGH-F480i and B2100 mobile.

13.13.2 Requirements

- Established bluetooth connection to the serial channel of the mobile
- Some way to access this serial port. For example minicom.

13.13.3 Starting the obex server

To start the obex server you have to send this AT command first:

```
AT+SYNCML=MOBEXSTART
```

Some phones seem to start with following command:

```
AT$TSSPCSW=1
```

13.13.4 Obex commands

In the following chapters I will describe the obex packages to read and edit data on the mobile. I think most of them are in standard-obex format and are following this structure:

Package Header	Session Id	Obex Header(s)
• First byte: Type of request. • Second and third bytes: length of package	• 0xCB and four bytes of session id	• First byte: Type of header. • Second and third bytes: length of header. • Next bytes: data. • Last byte: 0x00

For detailed information about obex, for example what types of packages and headers exists, get the official Obex documentation from Infared Data Association. But I don't know if this is available for free.

Here is a list of the most used types for the Samsung mobiles:

There exists mainly two types of operations: Put (package header 0x02 and 0x82) to write data to the mobile and Get (package header 0x03 and 0x83) to retrieve data from the mobile. A put or get operation can be divided into several packages. The high-bit indicates if this is the last package of an operation. For example if you want to transfer a file to the mobile you send n-time 0x02 packages and only the last one is 0x82.

Headers consists normally out of three blocks: First byte: Header type, second and third byte: length of the header (if the headers length is variable), following bytes: data. The most used header types are

Obex description	Byte	following two bytes	following bytes
Name	0x01	length of header	Used for filesystem operation to name a path or file
Type	0x42	length of header	Obex command for example "m-obex/contacts/list"
Length	0xC3		Used in put operations and specifies the length of the transferred data (without header bytes). The length is represented in 4 bytes.
Body	0x48	length of header	Data in a multi-package put operation
End of Bady	0x49	length of header	Last data package in a put operation
Session id	0xCE		Four bytes representing the session id. Needed for multiplexing
Application Parameter	0X4C	length of header	In a request: Parameters for example a contact's id. <p>In an answer: The error/return code. If it is 0x00 0x00 than the operation was successful

13.13.5 Contacts

Get contacts count

Request

***83* 00 25**
Obex Get

CB 00 00 00 00
Session Id

***42* 00 19 6D 2D 6F 62 65 78 2F 63 6F 6E 74 61 63 74 73 2F 63 6F 75 6E 74 00**
m-obex command: m-obex/contacts/count

4C 00 04 01
Unknown! Didn' see PC Studio sending something other than 0x01 as parameter

Answer

***A0* 00 14**
Obex ok

***C3* 00 00 00 04**
Maybe the number of requests you have to send to get all contacts. See next chapter for more information

***4C* 00 05 00 00**
Error code

***49* 00 07 07 D0 00 18**
First two data bytes: maximum number of contacts (0x07D0 = 2000). Last two data bytes: Current number of contacts

List all

Request

***83* 00 26**
Obex Get package

***CB* 00 00 00 00**
Session Id

***42* 00 18 6D 2D 6F 62 65 78 2F 63 6F 6E 74 61 63 74 73 2F 6C 6F 61 64 00**
m-obex Command: m-obex/contacts/load

***4C* 00 06 01 00 00**
First Byte unknown. Last two bytes: increment until all contacts received

Answer

***A0* 08 C1**
Obex Ok

***C3* 00 00 08 B1**
Length of sent data

***4C* 00 05 00 02**
Indicates if these are the last contacts

***49* 07 41 01 10 01 8D ….”**
The first byte is unknown but all answers have this byte, then byte 2 and 3 contains the length of the answer, bytes 4 and 5 are the ID of the first entry bytes 6 and 7 are the length of this entry.

In one response more than 1 vcard can be returned in this case, entries are separated by 4 bytes with the following meaning: bytes 1 and 2 ID of the entry, bytes 3 and 4: length of the entry.

To get all contacts the request have to be sent several times. The last two bytes must be incremented by every call.

The end of the contacts list is reached if the header 0x4C is 0. The header will be 4C 00 05 00 00.

Create a contact

Beware: This is a put operation and is performed in some obex implementations in several packages (for example 0x02, 0x02, 0x82). But I didn’t get the mobile to accept this. I had to create/update PIM data in exactly one package.

Request

***82* 00 88**
Obex put

***CB* 00 00 00 00**
Session id

***42* 00 1A 6D 2D 6F 62 65 78 2F 63 6F 6E 74 61 63 74 73 2F 63 72 65 61 74 65 00**
m-obex/contacts/create

***4C* 00 04 01**
? maybe flag for internal/external memory

C3 00 00 00 5A
Length of the vcard string

49 00 5D 42 45….
Contact as vcard

Answer

***A0* 00 12**
Obex ok

***C3* 00 00 00 02**
?

***4C* 00 05 00 00**
Error code

***49* 00 05 00 21**
last two bytes: the id of the newly created contact

Update a contact

Beware: This is a put operation and is performed in some obex implementations in several packages (for example 0x02, 0x02, 0x82). But I didn't get the mobile to accept this. I had to create/update PIM data in exactly one package.

Request

***82* 00 8D**
Obex put

***CB* 00 00 00 00**
Session id

***42* 00 19 6D 2D 6F 62 65 78 2F 63 6F 6E 74 61 63 74 73 2F 77 72 69 74 65 00**
m-obex/contacts/write

***4C* 00 06 01 00 20**
Id of the contact which should be updated

C3 00 00 00 5E
Length of the vcard string

49 00 61 42…
Contact as vcard

Answer

***A0* 00 08**
Obex ok

***4C* 00 05 00 00**
Error code: 0x00 0x00 means successful

Read one contact

There is also the possibility to read exactly one contact.

Request

***83* 00 26**
Obex get

***CB* 00 00 00 00**
Session id

***42* 00 18 6D 2D 6F 62 65 78 2F 63 6F 6E 74 61 63 74 73 2F 72 65 61 64 00**
m-obex/contacts/read

***4C* 00 06 01 00 20**
First byte:? Last two bytes: Id of contact

Answer

***A0* 00 C4**
Obex ok

***C3* 00 00 00 B4**
Length of vcard (without headers, just data)

***4C* 00 05 00 00**
Error code

49 00 B7 42 45 47 49 4E . . .
contact as vcard. TODO: where is id? First two bytes?

Delete contact

To delete a contact you only have to know it's id.

Request

***82* 00 28**
Obex put

***CB* 00 00 00 00**
Session id

***42* 00 1A 6D 2D 6F 62 65 78 2F 63 6F 6E 74 61 63 74 73 2F 64 65 6C 65 74 65 00**
m-obex/contacts/delete

***4C* 00 06 01 00 19**
First byte: ? Last two bytes: Id of contact

Answer

***A0* 00 08**
Obex ok

***4C* 00 05 00 00**
Error code

13.13.6 Calendar

Get count

Request

***83* 00 25**
Obex get

***CB* 00 00 00 00**
Session id

***42* 00 19 6D 2D 6F 62 65 78 2F 63 61 6C 65 6E 64 61 72 2F 63 6F 75 6E 74 00**
m-obex/calendar/count

***4C* 00 04 FF**
: ?

Answer

A0 00 1C
: Obex ok

***C3* 00 00 00 0C**
: length of data

***4C* 00 05 00 00**
: Error code

***49* 00 0F 01 2C 00 06 00 64 00 00 00 64 00 00**
: ?TODO?

List all

Request

***83* 00 20**
: Obex get

***CB* 00 00 00 00**
: Session id

***42* 00 18 6D 2D 6F 62 65 78 2F 63 61 6C 65 6E 64 61 72 2F 69 6E 66 6F 00**
: m-obex/calendar/load

Answer

***A0* 00 C0**
: Obex ok

***C3* 00 00 00 B0**
: Session

***4C* 00 05 00 00**
: Error code

***49* 00 B3 01 07 08 00 00 00 00 00 00 00 00 ...**
: Calendar items in vcalendar format. TODO: where are the ids?

Create

Request

***82* 00 CC**
: Obex put

***CB* 00 00 00 00**
: Session

***42* 00 1A 6D 2D 6F 62 65 78 2F 63 61 6C 65 6E 64 61 72 2F 63 72 65 61 74 65 00**
m-obex/calendar/create

4C 00 04 01
?

***C3* 00 00 00 9E**
Length of vcalendar

***49* 00 A1 42 45 47 49 4E 3A 56 43 41 4C 45 …**
vcalendar

Answer

***A0* 00 12**
Obex ok

***C3* 00 00 00 02**
Length

***4C* 00 05 00 00**
Error code

***49* 00 05 00 06**
Id of the created item

Update

Request

***82* 00 F7**
Obex put

***CB* 00 00 00 00**
Session

***42* 00 19 6D 2D 6F 62 65 78 2F 63 61 6C 65 6E 64 61 72 2F 77 72 69 74 65 00**
m-obex/calendar/write

***4C* 00 06 01 00 05**
First byte: ? Second and third byte: Id of the item

***C3* 00 00 00 C8**
Length of vcalendar

***49* 00 CB 42 45 47 49 4E 3A 56**
vcalendar item

Answer

***A0* 00 08**
Obex ok

***4C* 00 05 00 00**
Error code

Read

Request

***83* 00 26**
Obex get

***CB* 00 00 00 00**
Session

***42* 00 18 6D 2D 6F 62 65 78 2F 63 61 6C 65 6E 64 61 72 2F 72 65 61 64 00**
m-obex/calendar/read

4C 00 06 01 00 06
Id of calendar item

Answer

***A0* 00 C0**
Obex ok

***C3* 00 00 00 B0**
Length

***4C* 00 05 00 00**
Error code

***49* 00 B3 42 45 47 49 4E 3A 56 43 41 4C 45 4E 44 41 52 0D 0A 56 45 52 53 49 4F 4E 3A 31 2E 3….**
vcalendar item. TODO: Where is the id?

Delete

Request

***82* 00 28**
Obex put

***CB* 00 00 00 00**
Session

***42* 00 1A 6D 2D 6F 62 65 78 2F 63 61 6C 65 6E 64 61 72 2F 64 65 6C 65 74 65 00**
m-obex/calendar/delete

***4C* 00 06 01 00 06**
id of calendar item

13.13.7 Notes

13.13.8 Tasks

13.13.9 Files

To get the file structure on the mobile, there are two commands. One that lists all subdirectories and one that lists all files.

List directories

List files

Get file

Create file

Delete file

13.13.10 SMS

0x01: Inbox 0x08: Outbox

Get sms count

List all sms

Send sms

Create sms

I don't think this is possible. At least I didn't find the function in New PC Studio. So sadly there will be no backup of sms messages.

13.14 Series60 Remote Protocol

Changed in version 1.31.90: There were some changes in the protocol and applet has been renamed.

Note: The original applet has been created for <http://series60-remote.sourceforge.net/>. Gammu uses extended version which is probably not fully compatible with original.

Warning: Connection to S60 phones currently works only using Bluetooth.

13.14.1 Choosing right version

Before using this connection type, you need to install the applet to the phone. The applet can be found in `contrib/s60` directory and there are two variants of the applets:

`gammu-s60-remote.sis`
Not signed applet, which can be installed to the phone if it has enabled installation of unsigned applications (see *Allowing installation of unsigned applications*).

Note: This applet also lacks some capabilities, so for example you will not be able to get network information.

`gammu-s60-remote-sign.sis`
Applet ready for signing using Open Signed Online. This will allow you to install applet to your phone only (it is bound to IMEI), but you don't need to allow installation of unsigned applications.

Note: The best way of course would be to have properly signed applet. However access to signing tools costs 200 USD per year, what is something we can not afford right now.

Allowing installation of unsigned applications

For security reasons, Symbian defaults to install signed applications only. As getting properly signed applet is expensive for non commercial product like Gammu, you need to either sign applet yourself (the signature is valid for single phone) or allow installation of unsigned applications:

1. Open *Application Manager*, it is usually located in *Control Center*.
2. Press left soft key for *Options* menu.
3. From the menu choose *Settings*.
4. Change the *Software Installation* to *All*.
5. Change the *Online certif. check* to *Off*.

Warning: This allows installation of any not signed code to your phone. You should consider reverting this change, once you have installed applet required for Gammu.

13.14.2 Installation

To run the applet you need to install Python for S60 2.0 to the phone. You can either download it from their website, or just get mirrored installation package from <http://dl.cihar.com/gammu/s60/Python_2.0.0.sis>. This file is not distributed with Gammu due to licensing reasons.

Note: On some phones, the Python for S60 2.0 will not start, in this case you need to install some additional support libraries coming with Python for S60 2.0 - `pips.sis`, `ssl.sis` and `stdioserver.sis`. You can get all of them at https://wammu.eu/s60/ as well.

Installing Python for S60 and Series60 remote applet can be done in several ways:

Installation using Gammu

Gammu can transmit the applet to your phone automatically. Just configure it for use of BlueS60 connection (see *Connecting to Series60 phone* chapter below) and invoke `gammu install`:

```
gammu install
```

It will automatically transmit the applet to the phone. On some phones the installation will start automatically, on some you need to find received files in the inbox and install them manually from there.

If you want to install Python for S60 as well you need to download it and place in folder where Gammu searches for installation images (you can configure it by setting *DataPath*). For example:

```
cd /usr/share/data/gammu
wget http://dl.cihar.com/gammu/s60/Python_2.0.0.sis
wget http://dl.cihar.com/gammu/s60/pips.sis
```

Downloading from phone

Downloading files from the phone and installing them directly. You can download all required files from https://wammu.eu/s60/.

Fig. 1: QR code for download of applet.

Manual Installation using Gammu

If the above mentioned `gammu install` does not work for you, for example when you need to use different applet, you can still use Gammu to send files to the phone using `gammu sendfile`.

First you need to create `~/.gammurc` with following content:

```
[gammu]
connection = blueobex
model = obexnone
device = 5C:57:C8:XX:XX:XX # Address of the phone
```

And now you can send files to your phone:

```
gammu sendfile Python_2.0.0.sis
gammu sendfile contrib/s60/gammu-s60-remote.sis
```

Files should appear in inbox in your phone and you can install them from there.

13.14.3 Connecting to Series60 phone

The Gammu configuration is simple, all you need to specify is correct *Connection*:

```
[gammu]
connection = blues60
device = 5C:57:C8:XX:XX:XX # Address of the phone
```

Now you need to start the Series60 applet in the phone and Gammu should be able to talk to it.

13.15 Gnapplet Protocol

Note: The original applet has been created for <http://www.gnokii.org/>. Gammu currently uses slightly extended version which will be hopefully merged back.

13.15.1 Installation

To communicate with the phone, you need to install the applet. There are few options how to do it:

Installation using Gammu

Gammu can transmit the applet to your phone automatically. Just configure it for use of gnapplet connection and invoke `gammu install`:

```
gammu install
```

It will automatically transmit the applet to the phone. On some phones the installation will start automatically, on some you need to find received files in the inbox and install them manually from there.

Downloading from phone

Downloading files from the phone and installing them directly. You can download all required files from http://dl.cihar.com/gammu/gnapplet/.

Manual Installation using Gammu

If the above mentioned `gammu install` does not work for you, for example when you need to use different applet, you can still use Gammu to send files to the phone using `gammu sendfile`.

First you need to create `~/.gammurc` with following content:

```
[gammu]
connection = blueobex
model = obexnone
device = 5C:57:C8:XX:XX:XX # Address of the phone
```

And now you can send files to your phone:

```
gammu sendfile gnapplet.sis
gammu sendfile gnapplet.ini
```

Files should appear in inbox in your phone and you can install them from there.

See also:

You can also find documentation for some protocols and vendor extensions in separate git repository at http://github.com/gammu/gsm-docs

CHAPTER

FOURTEEN

GLOSSARY

TPMR
Message reference as generated by GSM network.

PYTHON MODULE INDEX

g

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W