<table>
<thead>
<tr>
<th>1</th>
<th>Gammu project</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>About Gammu</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>Installing Gammu</td>
<td>2</td>
</tr>
<tr>
<td>1.3</td>
<td>Contributing</td>
<td>8</td>
</tr>
<tr>
<td>1.4</td>
<td>Localization</td>
<td>8</td>
</tr>
<tr>
<td>1.5</td>
<td>Testing</td>
<td>9</td>
</tr>
<tr>
<td>1.6</td>
<td>Coding Style</td>
<td>9</td>
</tr>
<tr>
<td>1.7</td>
<td>Versioning</td>
<td>10</td>
</tr>
<tr>
<td>1.8</td>
<td>Project Documentation</td>
<td>10</td>
</tr>
<tr>
<td>1.9</td>
<td>Directory structure</td>
<td>11</td>
</tr>
<tr>
<td>1.10</td>
<td>Roadmap for Gammu</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>Frequently Asked Questions</td>
<td>17</td>
</tr>
<tr>
<td>2.1</td>
<td>General Gammu FAQ</td>
<td>17</td>
</tr>
<tr>
<td>2.2</td>
<td>Phone Support FAQ</td>
<td>17</td>
</tr>
<tr>
<td>2.3</td>
<td>SMSD FAQ</td>
<td>18</td>
</tr>
<tr>
<td>2.4</td>
<td>Python-gammu FAQ</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>python-gammu API</td>
<td>19</td>
</tr>
<tr>
<td>3.1</td>
<td>A taste of python-gammu</td>
<td>19</td>
</tr>
<tr>
<td>3.2</td>
<td>API documentation</td>
<td>22</td>
</tr>
<tr>
<td>4</td>
<td>libGammu</td>
<td>69</td>
</tr>
<tr>
<td>4.1</td>
<td>Hints for libGammu Novices</td>
<td>69</td>
</tr>
<tr>
<td>4.2</td>
<td>Examples</td>
<td>71</td>
</tr>
<tr>
<td>4.3</td>
<td>libGammu C API</td>
<td>85</td>
</tr>
<tr>
<td>4.4</td>
<td>Porting from libGammu older than 1.12.0</td>
<td>206</td>
</tr>
<tr>
<td>5</td>
<td>Gammu internals</td>
<td>209</td>
</tr>
<tr>
<td>5.1</td>
<td>Reply functions</td>
<td>209</td>
</tr>
<tr>
<td>5.2</td>
<td>State Machine</td>
<td>210</td>
</tr>
<tr>
<td>6</td>
<td>File formats used by Gammu</td>
<td>213</td>
</tr>
<tr>
<td>6.1</td>
<td>INI file format</td>
<td>213</td>
</tr>
<tr>
<td>6.2</td>
<td>SMS Backup Format</td>
<td>213</td>
</tr>
<tr>
<td>6.3</td>
<td>Backup Format</td>
<td>215</td>
</tr>
<tr>
<td>7</td>
<td>Gammu Configuration File</td>
<td>217</td>
</tr>
<tr>
<td>7.1</td>
<td>Synopsis</td>
<td>217</td>
</tr>
<tr>
<td>7.2</td>
<td>Description</td>
<td>217</td>
</tr>
</tbody>
</table>
7.3 Examples .......................................................... 221

8 Gammu Utility ......................................................... 225
8.1 Synopsis .......................................................... 225
8.2 Description ....................................................... 225
8.3 Return values ..................................................... 246
8.4 Examples .......................................................... 248

9 SMS Daemon ........................................................ 251
9.1 Overview ......................................................... 251
9.2 Usage .............................................................. 253
9.3 Program Manuals ............................................... 254
9.4 SMUD Configuration File ........................................ 258
9.5 RunOnReceive Directive ........................................ 264
9.6 Backend services ................................................ 267
9.7 Developer documentation ....................................... 293

10 Miscellaneous utilities ........................................ 299
10.1 gammu-detect .................................................... 299
10.2 gammu-config ................................................... 301
10.3 jadmaker .......................................................... 301

11 Testing Gammu ..................................................... 303
11.1 Gammu Testsuite ................................................ 303
11.2 Dummy Driver ................................................... 305

12 Phone Protocols .................................................. 307
12.1 Discovering protocol .......................................... 307
12.2 Nokia protocols ................................................ 308
12.3 Nokia S40 filesystem SMS format ............................. 313
12.4 Nokia 6110 ......................................................... 314
12.5 Nokia 6510 ......................................................... 328
12.6 Nokia 7110 ......................................................... 341
12.7 Nokia 6210/6310, CARC91, PC Experiment .................. 353
12.8 TDMA 5120 ....................................................... 364
12.9 SAMSUNG Organizer AT commands ......................... 366
12.10 SAMSUNG GT calendar AT commands ...................... 370
12.11 Sonim AT Commands ........................................... 373
12.12 m-obex protocol used by some Samsung mobiles ........... 373

13 Glossary ............................................................. 383

Python Module Index ................................................. 385
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 Čihář with help of many contributors.

1.1.1 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:

(a) 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.

(b) Now Reminder will be deleted (for example, from phone keypad). Call will be moved from 2’nd to 1’st.

(c) 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 “constans” 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...

1.2 Installing Gammu

1.2.1 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 <http://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.

1.2.2 Prebuilt Binaries for Windows

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


1.2.3 Dependencies


1.2.4 Optional Dependencies

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

Bluez-libs

- http://www.bluez.org/

libusb-1.0
• http://libusb.sourceforge.net/
  • fbususb/dku2 connection support on Linux

libCURL
• http://curl.haxx.se/libcurl/
  • New versions notification and OpenCellID access.

libiconv
• http://www.gnu.org/software/libiconv/
  • Support for more character sets in AT engine.

Gettext
• http://www.gnu.org/software/gettext/
  • Localization of strings.

MySQL
• http://mysql.com/
  • MySQL support in SMSD.

PostgreSQL
• http://www.postgresql.org/
  • PostgreSQL support in SMSD.

libdbi
• http://libdbi.sourceforge.net/
  • required at least version 0.8.2
  • DBI support in SMSD.

Python
• http://www.python.org/
  • Gammu has a Python bindings

SQLite + libdbi-drivers with SQLite
• http://www.sqlite.org/
  • needed for testing of SMSD using libdbi driver (libdbd-sqlite3)

1.2.5 Compiling on Linux/Unix Systems

For compatibility reasons, configure like wrapper is provided, if you don’t need much specific tuning, you can use general “./configure; make; sudo make install” set of commands. 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.

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 sim-
  ilar attacks
• -DBUILD_PYTHON="/usr/bin/python2.6" changes Python used for build Python module
• -DWITH_PYTHON=OFF disables build of python-gammu

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

1.2.6 Compiling on Microsoft Windows

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 http://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 bellow, 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.

• 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.

### 1.2.7 Compiling on Mac OS X

Gammu should be compilable on Mac OS X, you need to have installed Developer Tols (version 2.4.1 was tested) and CMake (there is a Mac OS X “Darwin” DMG download). For database support in SMSSD, 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:
1.2.8 Cross compilation for Windows on Linux

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:

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

```bash
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/](http://dev.mysql.com/), but then need some tweaks:

```bash
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/](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


1.2.9 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.

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.

Compiling python-gammu

Currently python-gammu is distributed together with Gammu, so all you need to get it is to build Gammu with Python support (it should be automatically detected if you have development environment installed for Python).
Gammu uses CMake to generate build environment (for example Makefiles for UNIX, Visual Studio projects, Eclipse projects, etc.) which you can later use for building. You can use `-DBUILD_PYTHON=/path/to/python` to define path to another Python interpreter to use than default one available in the system.

Alternatively you can use standard distutils, for which setup.py is placed in python subdirectory.

## 1.3 Contributing

We welcome contribution in any area, if you don’t have developer skills, you can always contribute to Localization. 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.

### 1.3.1 Sending patches

As we use Git for development, the preferred way to get patches is in form which can be directly applied to Git. So start with cloning our Git repository:

```
git clone git://gitorious.org/gammu/mainline.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 send the patches (all changes you’ve made so far) to us:

```
git send-email --to=gammu-users@lists.sourceforge.net origin
```

Please note that mailing list requires you to subscribe before posting. This is anyway good idea in case you want to contribute. However if you don’t want to do that, just send the mails directly to one of authors:

```
git send-email --to=michal@cihar.com origin
```

### 1.3.2 Creating patches

If for whatever reason you don’t want to use Git, you can also manually create patches using diff. Also we can handle if you send us just the file you have changed with reference where did you take it.

To manually create patch you can use following steps:

1. Copy source with Gammu, you start from, into gammu directory.
2. Copy source with Gammu, you want to modify, into work directory.
3. Make your changes in work directory.
4. Go into parent directory, where gammu and work directories are placed.
5. Call `diff -rup -X .git gammu work > patchfile`
6. Send patchfile to us (you can use bug tracker or mailing list).

## 1.4 Localization

Localization uses Gettext po files for both program translations and the documentation. The documentation translation is managed using po4a.
1.4.1 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).

1.4.2 Improving Translation

If you want to improve existing translation, please visit translation server. For adding new one, you need to contact Michal Čihář 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.

1.4.3 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 withing package (eg. README and INSTALL files).


1.5 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 Testing Gammu for more details.

1.6 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:

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

do {
    body of do-loop
} while (condition);
```
if (x == y) {
   
   
} else if (x > y) {
   
   
} 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.

### 1.7 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.

### 1.8 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.

#### 1.8.1 The Gammu Manual

This manual is in written in rst format and built using Sphinx with breathe extension. As Gammu uses latest version of this extension, it is shipped within Gammu sources, but if you use git, you need to check out it separately as git submodule (you should update it as well any time you update Gammu git):

git submodule update --init

To generate the documentation there are various manual-* targets for make. You can build HTML, PDF, PS, HTMLHELP 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
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

## 1.8.2 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

## 1.8.3 Code comments

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

## 1.9 Directory structure

### 1.9.1 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

---

1.9. Directory structure 11
service different gsm services for logos, ringtones, etc.

1.9.2 gammu directory

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

1.9.3 smsd directory

Sources of SMS Daemon as well as all it's service backends. The services subdirectory contains source code for Backend services.

1.9.4 python directory

Sources of python-gammu module and some examples.

1.9.5 helper directory

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

1.9.6 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

1.9.7 admin directory

Administrative scripts for updating locales, making release etc.

1.9.8 cmake directory

CMake include files and templates for generated files.

1.9.9 include directory

Public headers for libGammu.
1.9.10 locale directory

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

1.9.11 tests directory

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

1.9.12 utils directory

Various utilities usable with Gammu.

1.9.13 contrib directory

This directory contains various things which might be useful with Gammu. Please note that that code here might have different license terms than Gammu itself.

bash-completion

Completion script for bash.

cconversion

Various scripts for converting data.

init

Init scripts for Gammu SMSD.

linux-driver

DKU-2 driver for Linux. This will be removed in future, please use fbususb connection type instead.

mediamedia

Sample media files which can be used with Gammu.

other

Some scripts not related directly to Gammu, but more to modems and GSM phones in general.

packaging

Support for creating packages on various distributions. Please note that support for RPM (gammu.spec) and Slackware (description-pak) is included in top level directory.

1.9. Directory structure
perl

Various perl scripts which interface to Gammu or SMSD.

php

Various PHP frontends to SMSD or Gammu directly.

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.

win32

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

# vim: et ts=4 sw=4 sts=4 tw=72 spell spelllang=en_us

1.10 Roadmap for Gammu

1.10.1 1.30.0

I will try to work on change with locations handling here, but as it is quite intrusive, it might not be ready for this release.
1.10.2 Future

There are some major issues which should be addressed in Gammu soon. 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.

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 wchar_t used instead.

Avoid heavy usage of gsmstate.h header and move the #ifdef...#define...#endif blocks to gammu-config.h.

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.
FREQUENTLY ASKED QUESTIONS

2.1 General Gammu FAQ

2.1.1 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

2.1.2 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

2.1.3 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.

2.2 Phone Support FAQ

2.2.1 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.

See Also:
Are Symbian phones supported?, Gammu Configuration File

2.2.2 Which phone is best for SMS gateway?

Forget about using standard phones, they tend not to be reliable for long time connection to PC. Best option are GSM (GPRS, UMTS) terminals/modems. 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/...). You can check other user experiences in Gammu Phones Database.

2.2.3 Are Symbian phones supported?

Short answer: Not really.

Long answer: For older phones (Symbian 9.0 and older), you can install gnapplet to phone and access data through it. However gnapplet has not yet been ported to newer versions, so you have no chance with recent phones. You can try using Series60-Remote, which works pretty well with S60 phones. Another option is using something what supports SyncML to retrieve contacts or calendar from your phone, for example OpenSync or syncEvolution.

2.3 SMSD FAQ

2.3.1 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.

See Also:

SQL Service

2.3.2 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.

2.4 Python-gammu FAQ

2.4.1 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.

2.4.2 How can I use python-gammu?

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

See Also:

python-gammu API, More python-gammu Examples
3.1 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:

```python
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']
```

3.1.1 More python-gammu Examples

All these (and some more) examples are also available in python/examples/ directory in Gammu sources.

Sending a message

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

import gammu
import sys

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

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

# Check parameters
if len(sys.argv) != 2:
    print 'This requires one parameter containing number!'
    sys.exit(1)

# Connect to the phone
sm.Init()

# Prepare message data
# 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
sm.SendSMS(message)

Sending a long message

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

import gammu
import sys

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

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

# Check parameters
if len(sys.argv) != 2:
    print 'This requires one parameter containing number!'
    sys.exit(1)
# Connect to the phone
sm.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.' +
         '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
    sm.SendSMS(message)

Initiating a voice call

#!/usr/bin/env python

import gammu
import sys

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

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

# Connect to the phone
sm.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
sm.DialVoice(sys.argv[1])

3.1. A taste of python-gammu
Reading calendar from phone

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

import gammu

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

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

# Connect to the phone
sm.Init()

# Get number of calendar entries
status = sm.GetCalendarStatus()
remain = status[‘Used’]
start = True

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

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

3.2 API documentation

3.2.1 gammu – Mobile phone access

This module wraps all python-gammu functionality.

gamu.StateMachine

class gammu.StateMachine (Locale)
    StateMachine object, that is used for communication with phone.

    Parameters
        * Locale (string) – 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** *(string)* – Type of category to read, one of *ToDo, Phonebook*
- **Name** *(string)* – 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** *(string)* – Folder where to create subfolder
- **Name** *(string)* – New folder name

**Returns** New folder ID.

**Return type** string

**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 (string) – 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 (integer) – ID of call
- All (boolean) – Answer all calls?

**Returns** None
**Return type** None

**CancelCall (ID, All)**
Deny current incoming call.

**Parameters**
- ID (integer) – ID of call
- All (boolean) – Cancel all calls?

**Returns** None
**Return type** None

**ConferenceCall (ID)**
Initiates conference call.

**Parameters**
- ID (integer) – 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** *(string)* – 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** *(string)* – File to delete

Returns None
Return type None

**DeleteFolder(FolderID)**
Deletes folder on filesystem.

Parameters

- **FolderID** *(string)* – Folder to delete

Returns None
Return type None

**DeleteMemory(Type, Location)**
Deletes memory (phonebooks or calls) entry.

Parameters

- **Type** *(string)* – 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
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 (string) – Number to dial

Returns None
Return type None

DialVoice (Number, ShowNumber)
Dials number and starts voice call.

Parameters
• Number (string) – Number to dial
• ShowNumber (boolean or None) – Identifies whether to enable CLIR (None = keep default phone settings). Default is None

Returns None
Return type None

EnterSecurityCode (Type, Code, NewPIN)
Enter security code.

Parameters
• Type (string) – What code to enter, one of PIN, PUK, PIN2, PUK2, Phone.
• Code (string) – Code value
• NewPIN (string) – 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

**GetCategory (Type, Location)**

Reads category from phone.

**Parameters**

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

**Returns** Category name as string

**Return type** string

**GetCategoryStatus (Type)**

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

**Parameters**

- **Type (string)** – 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. :return: List of indicators displayed on display :rtype: 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 (string) – Folder to list

• Start (boolean) – 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 string.
    
    Return type  string

GetIMEI()
    Reads IMEI/serial number from phone.
    
    Returns  IMEI of phone as string.
    
    Return type  string

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 string.
    
    Return type  string

GetManufacturer()
    Reads manufacturer from phone.
    
    Returns  String with manufacturer name
    
    Return type  string

GetMemory(Type, Location)
    Reads entry from memory (phonebooks or calls). Which entry should be read is defined in entry.
    
    Parameters
    
    • Type (string) – 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 (string) – 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** *(boolean)* – 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** *(boolean)* – 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** *(string)* – Memory type, one of ME, SM, ON, DC, RC, MC, MT, FD, VM
- **Start** *(boolean)* – 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** *(string)* – 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** *(boolean)* – 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**

- \(Start\) (boolean) – 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 string

GetPPM()
Gets PPM (Post Programmable Memory) from phone.

**Returns** PPM as string

Return type string

GetProductCode()
Gets product code of device.

:return: Product code as string
:returntype: string

GetSIMIMSI()
Gets SIM IMSI from phone.

**Returns** SIM IMSI as string

Return type string

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

3.2. API documentation
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 string

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 (integer) – 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** *(string)* – What key to press  
  • **Press** *(boolean)* – 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** *(string)* – Path to configuration file (otherwise it is autodetected).

  **Returns** None  
  **Return type** None

**ReadDevice** *(Wait)*  
Reads data from device.

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

  **Returns** Number of bytes read  
  **Return type** int

**Reset** *(Hard)*  
Performs phone reset.

  **Parameters**  
  • **Hard** *(boolean)* – Whether to make hard reset

  **Returns** None  
  **Return type** None

**ResetPhoneSettings** *(Type)*  
Resets phone settings.

  **Parameters**  
  • **Type** *(string)* – 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** *(string)* – 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 integer
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 integer
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** *(boolean)* – Whether alarm should be repeating. Defaults to True.
- **Text** *(string)* – 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

SetDateTime (Date)
Sets date and time in phone.

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

Returns 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 (boolean) – Whether to use global debug structure (overrides File)

Returns None

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

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

Returns None

SetFileAttributes (Filename, ReadOnly, Protected, System, Hidden)
Sets file attributes.

Parameters
• **Filename** *(string)* – File to modify
• **ReadOnly** *(boolean)* – Whether file is read only. Default to False.
• **Protected** *(boolean)* – Whether file is protected. Default to False.
• **System** *(boolean)* – Whether file is system. Default to False.
• **Hidden** *(boolean)* – Whether file is hidden. Default to False.

Returns None
Return type None

**SetIncomingCB** *(Enable)*
Gets network information from phone.

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

Returns None
Return type None

**SetIncomingCall** *(Enable)*
Activates/deactivates noticing about incoming calls.

Parameters
• **Enable** *(boolean)* – 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** *(boolean)* – 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** *(boolean)* – Whether to enable notifications, default is True
Returns None
Return type None

`SetLocale (DateSeparator, DateFormat, AMPMTime)`
Sets locale of phone.

Parameters
- **DateSeparator** *(string)* – Date separator.
- **DateFormat** *(string)* – Date format, one of **DDMMYYYY, MMDDYYYY, YYYYMMDD**
- **AMPMTime** *(boolean)* – 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 (integer) – ID of call

Returns None
Return type None

SwitchCall (ID, Next)
Switches call.

Parameters
• ID (integer) – ID of call

Returns None
Return type None

Terminate ()
Terminates the connection with phone.

Returns None
Return type None

TransferCall (ID, Next)
Transfers call.

Parameters
• ID (integer) – ID of call

Returns None
Return type None

UnholdCall (ID)
Unholds call.

Parameters
• ID (integer) – ID of call

Returns None
Return type None

Generic functions

gammu.Version()
Get version information.


Return type tuple
Debugging configuration

\texttt{gammu.SetDebugFile(File)}

Sets global debug file.

\textbf{Parameters}

- \texttt{File (mixed)} – File where to write debug stuff (as configured by \texttt{SetDebugLevel()}). Can be either None for no file, Python file object or filename.

\textbf{Returns} None

\textbf{Return type} None

\texttt{gammu.SetDebugLevel(Level)}

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

\textbf{Parameters}

- \texttt{Level (string)} – name of debug level to use, currently one of:
  - nothing
  - text
  - textall
  - binary
  - errors
  - textdate
  - textalldate
  - errorsdate

\textbf{Returns} None

\textbf{Return type} None

Message processing

\texttt{gammu.LinkSMS(Messages, EMS)}

Links multi part SMS messages.

\textbf{Parameters}

- \texttt{Messages (list)} – List of messages to link, see \textit{SMS Object}
  - EMS (boolean) – Whether to detect ems, defauls to True

\textbf{Returns} List of linked messages, see \textit{SMS Object}

\textbf{Return type} list

\texttt{gammu.DecodeSMS(Messages, EMS)}

Decodes multi part SMS message.

\textbf{Parameters}

- \texttt{Messages (list)} – Messages to decode, see \textit{SMS Object}
  - EMS (boolean) – Whether to use EMS, defalt to True

\textbf{Returns} Multi part message information, see \textit{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* (*string*) – PDU data, need to be binary not hex encoded
- *SMSC* (*boolean*) – Whether PDU includes SMSC.

**Returns**  Message data, see *SMS Object*

Return type  dict
gammu.**EncodePDU** (*SMS, Layout = Submit*)
Creates PDU packet.

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

**Returns**  Message data

**Return type**  string

New in version 1.27.93.

**Encoding and decoding entries**

gammu.**DecodeVCARD** (*Text*)
Decodes memory entry v from a string.

**Parameters**
- *Text* (*string*) – 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**  string
gammu.**DecodeVCS** (*Text*)
Decodes todo/calendar entry v from a string.
Parameters

• **Text** *(string)* – 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** *(string)* – 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** string

gammu.EncodeICALendar(Entry)
Encodes calendar entry to a iCalendar.

Parameters

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

**Returns** String with iCalendar

**Return type** string

gammu.EncodeVTTODO(Entry)
Encodes todo entry to a vTodo.

Parameters

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

**Returns** String with vTodo

**Return type** string

gammu.EncodeITODO(Entry)
Encodes todo entry to a iTodo.

Parameters

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

**Returns** String with vCard

**Return type** string
Backup reading and writing

`gammu.SaveRingtone(Filename, Ringtone, Format)`
Saves ringtone into file.

**Parameters**
- **Filename** *(string)* – Name of file where ringtone will be saved
- **Ringtone** *(dict)* – Ringtone to save
- **Format** *(string)* – One of `ott, mid, rng, imy, wav, rtt1`

**Returns** None

**Return type** None

`gammu.SaveBackup(Filename, Backup, Format)`
Saves backup into file.

**Parameters**
- **Filename** *(string)* – Name of file to read backup from
- **Backup** *(dict)* – Backup data, see `ReadBackup()` for description
- **Format** *(string)* – 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** *(string)* – Name of file where backup is stored
- **Format** *(string)* – 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** *(string)* – 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** *(string)* – 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.

### 3.2.2 gammu.smsd – SMSD access

**SMD**

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

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

You don’t need to run the SMS daemon itself to control or ask it for status, this can be also done on separately running instances. All you need to do for this is to give same configuration file as that instance is using.

**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

3.2.3 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.

3.2.4 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 tasks 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 spaws 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.

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

Checks whether command is valid.

**Parameters**

- **command** *(string)* – Name of command.

### 3.2.5 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_NONE`

Bases: `gammu.GSMError`

Exception corresponding to gammu error ERR_NONE. Verbose error description: No error.
exception gammu. ERR_DEVICEOPENERROR
   Bases: gammu.GSMError
   Exception corresponding to gammu error ERR_DEVICEOPENERROR. Verbose error description: Error opening device. Unknown, busy or no permissions.

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

exception gammu. ERR_DEVICENOTEXIST
   Bases: gammu.GSMError
   Exception corresponding to gammu error ERR_DEVICENOTEXIST. Verbose error description: Error opening device, it doesn’t exist.

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

exception gammu. ERR_DEVICENOPERMISSION
   Bases: gammu.GSMError
   Exception corresponding to gammu error ERR_DEVICENOPERMISSION. Verbose error description: Error opening device, you don’t have permissions.

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

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

exception gammu. ERR_DEVICECHANGEDTRTSERROR
   Bases: gammu.GSMError
   Exception corresponding to gammu error ERR_DEVICECHANGEDTRTSERROR. Verbose error description: Error setting device DTR or RTS.

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

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

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

```python
exception gammu.ERR_DEVICEREADERROR
Bases: gammu.GSMError
```

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

```python
exception gammu.ERR_DEVICEPARITYERROR
Bases: gammu.GSMError
```

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

```python
exception gammu.ERR_TIMEOUT
Bases: gammu.GSMError
```

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

```python
exception gammu.ERR_FRAMENOTREQUESTED
Bases: gammu.GSMError
```

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

```python
exception gammu.ERR_UNKNOWNRESPONSE
Bases: gammu.GSMError
```

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

```python
exception gammu.ERR_UNKNOWNFRAME
Bases: gammu.GSMError
```

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

```python
exception gammu.ERR_UNKNOWNCONNECTIONTYPESTRING
Bases: gammu.GSMError
```

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

```python
exception gammu.ERR_UNKNOWNMODELSTRING
Bases: gammu.GSMError
```

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

```python
exception gammu.ERR_SOURCENOTAVAILABLE
Bases: gammu.GSMError
```

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

```python
exception gammu.ERR_NOTSUPPORTED
Bases: gammu.GSMError
```

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

```python
exception gammu.ERR_EMPTY
Bases: gammu.GSMError
```

Exception corresponding to gammu error ERR_EMPTY. Verbose error description: Entry is empty.
Exception corresponding to gammu error ERR_SECURITYERROR. Verbose error description: Security error. Maybe no PIN?

```
exception gammu::ERR_INVALIDLOCATION
    Bases: gammu.GSMError

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

```
exception gammu::ERR_NOTIMPLEMENTED
    Bases: gammu.GSMError

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

```
exception gammu::ERR_FULL
    Bases: gammu.GSMError

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

```
exception gammu::ERR_UNKNOWN
    Bases: gammu.GSMError

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

```
exception gammu::ERR_CANTOPENFILE
    Bases: gammu.GSMError

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

```
exception gammu::ERR_MOREMEMORY
    Bases: gammu.GSMError

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

```
exception gammu::ERR_PERMISSION
    Bases: gammu.GSMError

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

```
exception gammu::ERR_EMPTYSMSC
    Bases: gammu.GSMError

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_INSIDEPHONEMENU
    Bases: gammu.GSMError

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_NOTCONNECTED
    Bases: gammu.GSMError

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

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

```python
exception gammu.ERR_PHONEOFF
Bases: gammu.GSMError
Exception corresponding to gammu error ERR_PHONEOFF. Verbose error description: Phone is disabled and connected to charger.
```

```python
exception gammu.ERR_FILENOTSUPPORTED
Bases: gammu.GSMError
Exception corresponding to gammu error ERR_FILENOTSUPPORTED. Verbose error description: File format not supported by Gammu.
```

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

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

```python
exception gammu.ERR_NEEDANOTHERANSWER
Bases: gammu.GSMError
Exception corresponding to gammu error ERR_NEEDANOTHERANSWER. Verbose error description: Phone module need to send another answer frame.
```

```python
exception gammu.ERR_OTHERCONNECTIONREQUIRED
Bases: gammu.GSMError
Exception corresponding to gammu error ERR_OTHERCONNECTIONREQUIRED. Verbose error description: Current connection type doesn’t support called function.
```

```python
exception gammu.ERR_WRONGCRC
Bases: gammu.GSMError
Exception corresponding to gammu error ERR_WRONGCRC. Verbose error description: CRC error.
```

```python
exception gammu.ERR_INVALIDDATETIME
Bases: gammu.GSMError
Exception corresponding to gammu error ERR_INVALIDDATETIME. Verbose error description: Invalid date or time specified.
```

```python
exception gammu.ERR_MEMORY
Bases: gammu.GSMError
Exception corresponding to gammu error ERR_MEMORY. Verbose error description: Phone memory error, maybe it is read only.
```

```python
exception gammu.ERR_INVALIDDATA
Bases: gammu.GSMError
Exception corresponding to gammu error ERR_INVALIDDATA. Verbose error description: Invalid data given to phone.
```

```python
exception gammu.ERR_FILEALREADYEXIST
Bases: gammu.GSMError
```
Exception corresponding to gammu error ERR_FILEALREADYEXIST. Verbose error description: File with specified name already exists.

declaration gammu.ERR_FILEALREADYEXIST
Bases: gammu.GSMError

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

declaration gammu.ERR_FILENOTEXIST
Bases: gammu.GSMError

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

declaration gammu.ERR_SHOULDBEFOLDER
Bases: gammu.GSMError

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

declaration gammu.ERR_SHOULDBEFILE
Bases: gammu.GSMError

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

declaration gammu.ERR_NOSIM
Bases: gammu.GSMError

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

declaration gammu.ERR_GNAPPLETWRONG
Bases: gammu.GSMError

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

declaration gammu.ERR_FOLDERPART
Bases: gammu.GSMError

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

declaration gammu.ERR_FOLDERNOTEMPTY
Bases: gammu.GSMError

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

declaration gammu.ERR_DATA_CONVERTED
Bases: gammu.GSMError

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

declaration gammu.ERR_UNCONFIGURED
Bases: gammu.GSMError

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

declaration gammu.ERR_WRONGFOLDER
Bases: gammu.GSMError

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

declaration gammu.ERR_PHONE_INTERNAL
Bases: gammu.GSMError

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

```python
exception gammu.ERR_WRITING_FILE
    Bases: gammu.GSMError
        Exception corresponding to gammu error ERR_WRITING_FILE. Verbose error description: Error writing file to disk.
```

```python
exception gammu.ERR_NONE_SECTION
    Bases: gammu.GSMError
        Exception corresponding to gammu error ERR_NONE_SECTION. Verbose error description: No such section exists.
```

```python
exception gammu.ERR_USING_DEFAULTS
    Bases: gammu.GSMError
        Exception corresponding to gammu error ERR_USING_DEFAULTS. Verbose error description: Using default values.
```

```python
exception gammu.ERR_CORRUPTED
    Bases: gammu.GSMError
        Exception corresponding to gammu error ERR_CORRUPTED. Verbose error description: Corrupted data returned by phone.
```

```python
exception gammu.ERR_BADFEATURE
    Bases: gammu.GSMError
        Exception corresponding to gammu error ERR_BADFEATURE. Verbose error description: Bad feature string in configuration.
```

```python
exception gammu.ERR_DISABLED
    Bases: gammu.GSMError
        Exception corresponding to gammu error ERR_DISABLED. Verbose error description: Desired functionality has been disabled on compile time.
```

```python
exception gammu.ERR_SPECIFYCHANNEL
    Bases: gammu.GSMError
        Exception corresponding to gammu error ERR_SPECIFYCHANNEL. Verbose error description: Bluetooth configuration requires channel option.
```

```python
exception gammu.ERR_NOTRUNNING
    Bases: gammu.GSMError
        Exception corresponding to gammu error ERR_NOTRUNNING. Verbose error description: Service is not running.
```

```python
exception gammu.ERR_NOSERVICE
    Bases: gammu.GSMError
        Exception corresponding to gammu error ERR_NOSERVICE. Verbose error description: Service configuration is missing.
```

```python
exception gammu.ERR_BUSY
    Bases: gammu.GSMError
        Exception corresponding to gammu error ERR_BUSY. Verbose error description: Command rejected because device was busy. Wait and restart.
```
exception gammu.ERR_COULDNT_CONNECT
   Bases: gammu.GSMError

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

exception gammu.ERR_COULDNT_RESOLVE
   Bases: gammu.GSMError

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

exception gammu.ERR_GETTING_SMSC
   Bases: gammu.GSMError

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

3.2.6 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 on 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': '123456',
    '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 - maximal 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:

```python
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 - Concatenated SMS, when longer than 1 SMS.
- ConcatenatedAutoTextLong - Concatenated 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
• EMSSFixedBitmap - Fixed bitmap of size 16x16 or 32x32.
• EMSSVariableBitmap
• EMSSVariableBitmapLong
• 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
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:

```python
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_DATE_TIME` - Alarm date and time.
- `SILENT_ALARM_DATE_TIME` - 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_Work** - Work number. (Text)
- **Number_Fax** - Fax number. (Text)
- **Number_Home** - Home 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_WorkStreetAddress - Work street address. (Text)
• Text_WorkCity - Work city. (Text)
• Text_WorkState - Work state. (Text)
• Text_WorkZip - Work zip code. (Text)
• Text_WorkCountry - Work country. (Text)
• Text_WorkPostal - Complete work postal address. (Text)
• Text_PictureName - Picture name (on phone filesystem). (Text)
• PushToTalkID - Push-to-talk ID (Text)
• Number_Messaging - Favorite messaging number. (Text)
• Photo - Photo (Picture).
• Number_Mobile_Home - Home mobile number. (Text)
• Number_Mobile_Work - Work mobile number. (Text)

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,
}
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 API.

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.

**Warning:** The C API documentation is still more complete in Doxygen format available on Gammu website. This still lacks some parts due to problems in Doxygen - Sphinx bridge.

## 4.1 Hints for libGammu Novices

This is very short overview of libGammu usage. You will probably need to study [libGammu C API](https://gammu.org/capi/) to find out what functions you want to use.

### 4.1.1 Basic library usage

You need to include main header file:

```c
#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 struct. This structure is not public, so all you can define is a pointer to it:

```c
GSM_StateMachine *state_machine;
```

You’ll want to check for errors from time to time. Do it using a function something like this:

```c
void check_error(GSM_Error err)
{
    if (err == ERR_NONE) {
        return;
    }
    // Handle error
}
```
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):

```c
GSM_InitLocales(NULL);
```

You first need to allocate a state machine structure:

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

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

```c
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 only 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):

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

Now you are ready to communicate with the phone, for example you can read manufacturer name:

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

When you’re finished, you need to disconnect and free allocated memory:

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

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

There are also other Examples.

### 4.1.2 Compling 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

```bash
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 $@

4.1.3 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.

4.1.4 Debugging

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

To enable global debugging use:
```
dump_info = GSM_GetGlobalDebug();
GSM_SetDebugFileDescriptor(stderr, FALSE, dump_info);
GSM_SetDebugLevel("textall", debug_info);
```

For per state machine configuration:
```
    dump_info = GSM_GetDebug(s);
    GSM_SetDebugGlobal(FALSE, debug_info);
    GSM_SetDebugFileDescriptor(stderr, FALSE, debug_info);
    GSM_SetDebugLevel("textall", debug_info);
```

4.2 Examples

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

4.2. Examples
4.2.1 Getting phone information

```c
#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.
    */
    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_SetDebugEnabled(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
", buffer);

/* 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:
 */

4.2.2 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);
    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)

```c
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();

/* 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);
```

```c
return 0;
```
4.2.3 Sending SMS message

```c
#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: \"");
    printf("%s\n", GSM_GetConfig(sm, -1)->Device);
    printf("\n if (status==0) {
        printf("..OK\n");
    sms_send_status = ERR_NONE;
} else {
        printf("..error %i", status);
        sms_send_status = ERR_UNKNOWN;
    }
    printf("\n, 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_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 */
default_info = GSM_GetGlobalDebug();
GSM_DebugFileDescriptor(stderr, TRUE, default_info);
GSM_DebugLevel("textall", default_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. */
default_info = GSM_GetDebug(s);
GSM_DebugGlobal(FALSE, default_info);
GSM_DebugFileDescriptor(stderr, TRUE, default_info);
GSM_DebugLevel("textall", default_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 calling 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 */
            return_value = 0;
            break;
        } else 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;
4.2.4 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\"
", 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;
}

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[2*(strlen(message_text) + 1)];
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 */
debugee = GSM_GetGlobalDebug();
GSM_SetDebugFileDescriptor(stderr, TRUE, debug_info);
GSM_SetDebugLevel("textall", debug_info);

/* Fill in SMS infor structure which will be used to generate
 * 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
", 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. */
debugee = 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);

/* Connect to phone */
/* I 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 calling 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;
}

/* 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:
*/

4.2.5 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 */
4.2.6 Custom configuration

#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
     * charset conversion works. */
    GSM_InitLocales(NULL);
/* Enable global debugging to stderr */
ddebug_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. */
ddebug_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);

/* 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;
}

#include <gammu.h>

4.3 libGammu C API

Warning: The C API documentation is still more complete in Doxygen format available on Gammu website. This still lacks some parts due to problems in Doxygen - Sphinx bridge.

4.3.1 Backup

Reads SMS backup file.

Parameters
• FileName - file name
• backup - structure where backup will be stored

Return
Error code

Adds data to SMS backup file.

Parameters
• FileName - file name
• backup - structure holding backup data

Return
Error code

Clears SMS backup structure

Parameters
Deallocates all members of SMS backup structure

Parameters

• backup -
  structure where backup data will be stored

Save backup file.

Parameters

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

Return

Error code

Guesses backup format based on filename.

Parameters

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

Return

Backup format on success -1 on error.

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.
Return

Error code

Clears backup structure

Parameters

• backup -
  structure where backup data will be stored

Deallocates all members of backup structure

Parameters

• backup -
  structure where backup data will be stored

Gets information about format features.

Parameters

• Format -
  Format of backup.
• info -
  Output information about backup features.

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 check.

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[80]
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 * CallerLogo[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]
- Profiles

**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
- Startup logo

**GSM_Bitmap** * OperatorLogo
- Operator logo

Backup data.

**Values:**

- **GSM_Backup_Auto= 0** -
  Compatibility with old gboolean used instead of format.
  File type is guessed for extension, non unicode format used for Gammu backup.

- **GSM_Backup_AutoUnicode= 1** -
  Compatibility with old gboolean used instead of format.
  File type is guessed for extension, unicode format used for Gammu backup.

- **GSM_Backup_LMB** -
  LMB format, compatible with Logo manager, can store phonebooks and logos.

- **GSM_Backup_VCalendar** -
  vCalendar standard, can store todo and calendar entries.

- **GSM_Backup_VCard** -
  vCard standard, can store phone phonebook entries.

- **GSM_Backup_LDIF** -
  LDIF (LDAP Data Interchange Format), can store phone phonebook entries.

- **GSM_Backup_ICS** -
  iCalendar standard, can store todo and calendar entries.
• **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.

• **GSM_Backup_GammuUCS2** -
  Gammu own format can store almost anything from phone.
  This is UCS2-BE version of the format.

• **GSM_Backup_VNote** -
  vNote standard, can store phone notes.

**struct GSM_Backup_Info**

Information about supported backup features.

**Public Members**

```c
gboolean UseUnicode
gboolean IMEI
gboolean Model
gboolean DateTime
gboolean ToDo
gboolean PhonePhonebook
gboolean SIMPhonebook
gboolean Calendar
gboolean CallerLogos
gboolean SMSC
gboolean WAPBookmark
gboolean Profiles
gboolean WAPSettings
gboolean MMSSettings
gboolean SyncMLSettings
gboolean ChatSettings
gboolean Ringtone
gboolean StartupLogo
gboolean OperatorLogo
gboolean FMStation
gboolean GPRSPoint
gboolean Note
```
4.3.2 Bitmap

Gets bitmap from phone.

Sets bitmap in phone.

Prints bitmap to file descriptor.

Parameters

- file -
  Where to print.
- bitmap -
  Bitmap to print.

Saves bitmap to file.

Parameters

- FileName -
  Where to save.
- bitmap -
  Bitmap to save.

Return

Error code

Reads bitmap from file.

Parameters

- FileName -
  Where to load from.
- bitmap -
  Pointer where to load bitmap.

Return

Error code

Checks whether point is set in bitmap.

Parameters

- bmp -
  Bitmap
• \textit{x} - Horizontal coordinate.
• \textit{y} - Vertical coordinate.

\textbf{Return}
True if point is set.

Sets point in bitmap.

\textit{Parameters}
• \textit{bmp} - Bitmap
• \textit{x} - Horizontal coordinate.
• \textit{y} - Vertical coordinate.

Clears point in bitmap.

\textit{Parameters}
• \textit{bmp} - Bitmap
• \textit{x} - Horizontal coordinate.
• \textit{y} - Vertical coordinate.

Clears bitmap.

\textit{Parameters}
• \textit{bmp} - Bitmap

Binary picture types.

\textit{Values}:
• PICTURE_BMP = 1
• PICTURE_GIF
• PICTURE_JPG
• PICTURE_ICN
• PICTURE_PNG -

**struct GSM_BinaryPicture**

Binary picture data.

*Public Members*

**GSM_BinaryPicture_Types** Type

unsigned char * Buffer

int Length

Enum to handle all possible bitmaps, which are not saved in various filesystems.

*Values:*

• GSM_None= 1 -

• GSM_ColourStartupLogo_ID -
  ID of static file in filesystem displayed during startup

• GSM_StartupLogo -
  Static mono bitmap/ID of animated mono bitmap displayed during startup

• GSM_ColourOperatorLogo_ID -
  ID of static file in filesystem displayed instead of operator name

• GSM_OperatorLogo -
  Mono bitmap displayed instead of operator name

• GSM_ColourWallPaper_ID -
  ID of static file in filesystem displayed as wallpaper

• GSM_CallerGroupLogo -
  Mono bitmap assigned to caller group

• GSM_DealerNote_Text -
  Text displayed during startup, which can’t be removed from phone menu

• GSM_WelcomeNote_Text -
  Text displayed during startup

• GSM_PictureImage -
  Image defined in Smart Messaging specification

• 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

gboolean FileSystemRingtone

int PictureID
    For caller group logo: picture ID. Phone model specific

gboolean FileSystemPicture

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[7]
    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

4.3.3 Calendar

Finds indexes of default entries.

Encodes vTodo to buffer.

Parameters

• Buffer -
  Storage for data.
• buff_len -
  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.

Return

Error code.

Encodes vCalendar to buffer.

Parameters

• Buffer -
  Storage for data.
• buff_len -
  Size of output buffer.
• Length -
  Pointer to current position in data (will be incremented).
Decodes vNote from buffer.

**Parameters**
- **Buffer** - Buffer to decode.
- **Pos** - Current position in buffer (will be updated).
- **Note** - Storage for note entry.

**Return**
Error code.

Encodes vNote to buffer.

**Parameters**
- **Buffer** - Storage for data.
- **buff_len** - Size of output buffer.
- **Length** - Pointer to current position in data (will be incremented).
- **Note** - Note to encode.

**Return**
Error code.

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.

**Return**

Error code

Detects whether calendar note is in past.

**Parameters**

• **note** -
  Note to check.

**Return**

Whether entry is in past.

Reads alarm set in phone.

**Parameters**

• **s** -
  State machine pointer.
• **Alarm** -
  Storage for alarm.

**Return**

Error code

Sets alarm in phone.

**Parameters**
- **s** -
  State machine pointer.
- **Alarm** -
  Alarm to set.

**Return**
Error code

Gets status of ToDos (count of used entries).

**Parameters**
- **s** -
  State machine pointer.
- **status** -
  Storage for todo status.

**Return**
Error code

Reads ToDo from phone.

**Parameters**
- **s** -
  State machine pointer.
- **ToDo** -
  Storage for note.

**Return**
Error code

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.

**Return**
Error code
Sets ToDo in phone.

Parameters

- `s` - State machine pointer.
- `ToDo` - ToDo to set, should contain valid location.

Return

Error code

Adds ToDo in phone.

Parameters

- `s` - State machine pointer.
- `ToDo` - ToDo to add.

Return

Error code

Deletes ToDo entry in phone.

Parameters

- `s` - State machine pointer.
- `ToDo` - ToDo to delete, only location is actually used.

Return

Error code

Deletes all todo entries in phone.

Parameters

- `s` - State machine pointer.

Return

Error code
Retrieves calendar status (number of used entries).

*Parameters*

- **s** - State machine pointer.
- **Status** - Storage for status.

*Return*

Error code

Retrieves calendar entry.

*Parameters*

- **s** - State machine pointer.
- **Note** - Storage for note.

*Return*

Error code

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.

*Return*

Error code

Sets calendar entry

*Parameters*

- **s** - State machine pointer.
- **Note** - New note values, needs to contain valid position.
Returns

Error code

Adds calendar entry.

Parameters

• **s** -
  State machine pointer.

• **Note** -
  Note to add.

Returns

Error code

Deletes calendar entry.

Parameters

• **s** -
  State machine pointer.

• **Note** -
  Note to delete, must contain position.

Returns

Error code

Deletes all calendar entries.

Parameters

• **s** -
  State machine pointer.

Returns

Error code

Reads calendar settings.

Parameters

• **s** -
  State machine pointer.

• **settings** -
  Storage for settings.

Returns

Error code
Sets calendar settings.

Parameters

- \texttt{s}
  - State machine pointer.
- \texttt{settings}
  - New calendar settings.

Return

Error code

Retrieves notes status (number of used entries).

Parameters

- \texttt{s}
  - State machine pointer.
- \texttt{status}
  - Storage for status.

Return

Error code

Retrieves notes entry.

Parameters

- \texttt{s}
  - State machine pointer.
- \texttt{Note}
  - Storage for note.

Return

Error code

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

Parameters

- \texttt{s}
  - State machine pointer.
- \texttt{Note}
  - Storage for note, if start is FALSE, should contain data from previous read (at least position).
- \texttt{start}
  - Whether we’re doing initial read or continue in reading.
Return
Error code

Sets note entry

Parameters

• s -
  State machine pointer.
• Note -
  New note values, needs to contain valid position.

Return
Error code

Adds note entry.

Parameters

• s -
  State machine pointer.
• Note -
  Note to add.

Return
Error code

Deletes note entry.

Parameters

• s -
  State machine pointer.
• Note -
  Note to delete, must contain position.

Return
Error code

Deletes all notes entries.

Parameters

• s -
  State machine pointer.

Return
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 defines types of calendar notes

Values:

• GSM_CAL_REMINDER= 1 -
  Reminder or Date
• GSM_CAL_CALL -
  Call
• GSM_CAL_MEETING -
  Meeting
• GSM_CAL_BIRTHDAY -
  Birthday or Anniversary or Special Occasion
• GSM_CAL_MEMO -
  Memo or Miscellaneous
• GSM_CAL_TRAVEL -
  Travel
• GSM_CAL_VACATION - Vacation
• GSM_CAL_T_ATHL - Training - Athletism
• GSM_CAL_T_BALL - Training - Ball Games
• GSM_CAL_T_CYCL - Training - Cycling
• GSM_CAL_T_BUDO - Training - Budo
• GSM_CAL_T_DANC - Training - Dance
• GSM_CAL_T_EXTR - Training - Extreme Sports
• GSM_CAL_T_FOOT - Training - Football
• GSM_CAL_T_GOLF - Training - Golf
• GSM_CAL_T_GYM - Training - Gym
• GSM_CAL_T_HORS - Training - Horse Race
• GSM_CAL_T_HOCK - Training - Hockey
• GSM_CAL_T_RACE - Training - Races
• GSM_CAL_T_RUGB - Training - Rugby
• GSM_CAL_T_SAIL - Training - Sailing
• GSM_CAL_T_STRE - Training - Street Games
• GSM_CAL_T_SWIM - Training - Swimming
• GSM_CAL_T_TENN - Training - Tennis
• GSM_CAL_T_TRAV - Training - Travels
• GSM_CAL_T_WINT - Training - Winter Games
• GSM_CAL_ALARM - Alarm
• GSM_CAL_DAILY_ALARM - Alarm repeating each day.
• GSM_CAL_SHOPPING - Shopping

One value of calendar event.

Values:

• CAL_START_DATETIME= 1 - Date and time of event start.
• CAL_END_DATETIME - Date and time of event end.
• CAL_TONE_ALARM_DATETIME - Alarm date and time.
• CAL_SILENT_ALARM_DATETIME - Date and time of silent alarm.
• CAL_TEXT - Text.
• CAL_DESCRIPTION - Detailed description.
• CAL_LOCATION - Location.
• CAL_PHONE - Phone number.
• CAL_PRIVATE - Whether this entry is private.
• CAL_CONTACTID - Related contact id.
• CAL_REPEAT_DAYOFWEEK - Repeat each x’th day of week.
• CAL_REPEAT_DAY -
  Repeat each x’th day of month.
• CAL_REPEAT_DAYOFYEAR -
  Repeat each x’th day of year.
• CAL_REPEAT_WEEKOFMONTH -
  Repeat x’th week of month.
• CAL_REPEAT_MONTH -
  Repeat x’th month.
• CAL_REPEAT_FREQUENCY -
  Repeating frequency.
• CAL_REPEAT_STARTDATE -
  Repeating start.
• CAL_REPEAT_STOPDATE -
  Repeating end.
• CAL_REPEAT_COUNT -
  Number of repetitions.
• CAL_LUID -
  IrMC LUID which can be used for synchronisation.
• 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

4.3. libGammu C API
**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.

Types of to do values. In parenthesis is member of `GSM_SubToDoEntry`, where value is stored.

**Values:**

- **TODO_END_DATETIME= 1 -**
  Due date (Date).
- **TODO_COMPLETED -**
  Whether is completed (Number).
- **TODO_ALARM_DATETIME -**
  When should alarm be fired (Date).
- **TODO_SILENT_ALARM_DATETIME -**
  When should silent alarm be fired (Date).
- **TODO_TEXT -**
  Text of to do (Text).
- **TODO_DESCRIPTION -**
  Description of to do (Text).
- **TODO_LOCATION -**
  Location of to do (Text).
- **TODO_PRIVATE -**
  Whether entry is private (Number).
- **TODO_CATEGORY -**
  Category of entry (Number).
- **TODO_CONTACTID -**
  Related contact ID (Number).
- **TODO_PHONE -**
  Number to call (Text).
- **TODO_LUID -**
  IrMC LUID which can be used for synchronisation (Text).
• **TODO_LAST_MODIFIED** -
  Date and time of last modification (Date).

• **TODO_START_DATETIME** -
  Start date (Date).

• **TODO_COMPLETED_DATETIME** -
  Start date (Date).

Priority of to do.

*Values:*

• GSM_Priority_None = 0 -
• GSM_Priority_High -
• GSM_Priority_Medium -
• GSM_Priority_Low -

*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.

```c
struct GSM_NoteEntry
```
Note entry.

*Public Members*

```c
int Location
```
Location in memory.

```c
char Text[(GSM_MAX_NOTE_TEXT_LENGTH+1)*2]
```
Text of note.

```c
struct GSM_Alarm
```
Alarm values.

*Public Members*

```c
int Location
```
Location where it is stored.

```c
GSM_DateTime DateTime
```
Date and time of alarm.

```c
gboolean Repeating
```
Whether it repeats each day.

```c
unsigned char Text[(GSM_MAX_CALENDAR_TEXT_LENGTH+1)*2]
```
Text that is shown on display.

---

Format of vTodo.

*Values:*

- **Nokia_VToDo= 1** - Format compatible with Nokia - limited subsed of standard.
- **SonyEricsson_VToDo** - Format compatible with SonyEricsson - complete standard.
- **Mozilla_VToDo** - Format compatible with Mozilla - iCalendar based.

Format of vCalendar export.

*Values:*

- **Nokia_VCAlendar= 1** - vCalendar specially hacked for Nokia.
- **Siemens_VCAlendar** - vCalendar specially hacked for Siemens.
• SonyEricsson_vCalendar -
  Standard vCalendar (which works for Sony-Ericsson phones)
• Mozilla_iCalendar -
  iCalendar as compatible with Mozilla.

### 4.3.4 Callback

Sets callback for incoming calls.

**Parameters**

- \( s \) -
  State machine.
- \( callback \) -
  Pointer to callback function.
- \( user\_data \) -
  Second parameter which will be passed to callback.

Sets callback for incoming SMSes.

**Parameters**

- \( s \) -
  State machine.
- \( callback \) -
  Pointer to callback function.
- \( user\_data \) -
  Second parameter which will be passed to callback.

Sets callback for incoming CB.

**Parameters**

- \( s \) -
  State machine.
- \( callback \) -
  Pointer to callback function.
- \( user\_data \) -
  Second parameter which will be passed to callback.

Sets callback for incoming USSD.

**Parameters**
• \texttt{s} -
  State machine.
• \texttt{callback} -
  Pointer to callback function.
• \texttt{user\_data} -
  Second parameter which will be passed to callback.

Sets callback for sending SMS.

\textit{Parameters}

• \texttt{s} -
  State machine.
• \texttt{callback} -
  Pointer to callback function.
• \texttt{user\_data} -
  Second parameter which will be passed to callback.

Callback for incoming calls.

Callback for incoming SMS.

Callback for incoming cell broadcast.

Callback for incoming USSD.

Callback for sending SMS.

\textbf{4.3.5 Call}

Dials number and starts voice call.

\textit{Parameters}

• \texttt{s} -
  State machine pointer.
• \texttt{Number} -
  Number to dial.
• \texttt{ShowNumber} -
  Whether we want to display number on phone.
Return

Error code

Dials service number (usually for USSD).

Parameters
- \texttt{s} - State machine pointer.
- \texttt{Number} - Number to dial.

Return

Error code

Accept current incoming call.

Parameters
- \texttt{s} - State machine pointer.
- \texttt{ID} - ID of call.
- \texttt{all} - Whether to handle all call and not only the one specified by ID.

Return

Error code

Deny current incoming call.

Parameters
- \texttt{s} - State machine pointer.
- \texttt{ID} - ID of call.
- \texttt{all} - Whether to handle all call and not only the one specified by ID.

Return

Error code

Holds call.

Parameters
• s -  
  State machine pointer.
• ID -  
  ID of call.

_Return_

Error code

Unholds call.

_Parameters_

• s -  
  State machine pointer.
• ID -  
  ID of call.

_Return_

Error code

Initiates conference call.

_Parameters_

• s -  
  State machine pointer.
• ID -  
  ID of call.

_Return_

Error code

Splits call.

_Parameters_

• s -  
  State machine pointer.
• ID -  
  ID of call.

_Return_

Error code

Transfers call.

_Parameters_
• **s**
  State machine pointer.
• **ID**
  ID of call.
• **next**
  Switches next call and ignores ID.

**Return**
Error code

Switches call.

**Parameters**
• **s**
  State machine pointer.
• **ID**
  ID of call.
• **next**
  Switches next call and ignores ID.

**Return**
Error code

Gets call diverts.

**Parameters**
• **s**
  State machine pointer.
• **divert**
  Storage for diversions information.

**Return**
Error code

Sets call diverts.

**Parameters**
• **s**
  State machine pointer.
• **divert**
  Diversions information to set.
**Return**

Error code

Cancels all diverts.

**Parameters**

- `s` - State machine pointer.

**Return**

Error code

Activates/deactivates noticing about incoming calls.

**Parameters**

- `s` - State machine pointer.
- `enable` - Whether to enable notifications.

**Return**

Error code

Sends DTMF (Dual Tone Multi Frequency) tone.

**Parameters**

- `s` - State machine pointer.
- `sequence` - Sequence to press.

**Return**

Error code

Enum with status of call.

**Values:**

- `GSM_CALL_IncomingCall` - Somebody calls to us
- `GSM_CALL_OutgoingCall` - We call somewhere
- `GSM_CALL_CallStart` - Call started
- **GSM_CALL_CallEnd** - End of call from unknown side
- **GSM_CALL_CallRemoteEnd** - End of call from remote side
- **GSM_CALL_CallLocalEnd** - End of call from our side
- **GSM_CALL_CallEstablished** - Call established. Waiting for answer or dropping
- **GSM_CALL_CallHeld** - Call held
- **GSM_CALL_CallResumed** - Call resumed
- **GSM_CALL_CallSwitched** - We switch to call

```c
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.
```

Defines when diversion is active.

**Values:**

- **GSM_DIVERT_Busy** = 0x01 - Divert when busy.
- **GSM_DIVERT_NoAnswer** - Divert when not answered.
- **GSM_DIVERT_OutOfReach** - Divert when phone off or no coverage.
• GSM_DIVERT_AllTypes -
  Divert all calls without ringing.

Which type of calls should be diverted.

Values:

• GSM_DIVERT_VoiceCalls= 0x01 - Voice calls.
• GSM_DIVERT_FaxCalls - Fax calls.
• GSM_DIVERT_DataCalls - Data calls.
• 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.

  char Number[(GSM_MAX_NUMBER_LENGTH+1)*2]
    Number where to divert.

struct GSM_MultiCallDivert
  Multiple call diversions.

Public Members

  GSM_CallDivert Request

  int EntriesNum

  GSM_CallDivert Entries[10]

  struct GSM_MultiCallDivert::@0 Response

How to handle number when initiating voice call.

Values:

• GSM_CALL_ShowNumber= 1 - Show number.
• GSM_CALL_HideNumber -
  Hide number.
• GSM_CALL_DefaultNumberPresence -
  Keep phone default settings.

### 4.3.6 Category

Reads category from phone.

**Parameters**

- *s* -
  State machine pointer.
- *Category* -
  Storage for category, containing its type and location.

**Return**

Error code

Adds category to phone.

**Parameters**

- *s* -
  State machine pointer.
- *Category* -
  New category, containing its type and location.

**Return**

Error code

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

**Parameters**

- *s* -
  State machine pointer.
- *Status* -
  Category status, fill in type before calling.

**Return**

Error code

Type of category

**Values:**
• Category_ToDo= 1 -
  Todo entry category
• 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.
```

### 4.3.7 Date and time

Returns string for current day of week.

**Parameters**

- **year** -
  Year.
- **month** -
  Month.
- **day** -
  Day.

**Return**

Pointer to static buffer containing day of week string.

Returns current timestamp.

**Parameters**
• Date
  Storage for date time structure.

Converts *GSM_DateTime* to time_t.

**Parameters**

• DT
  Input timestamp.

**Return**

time_t value.

Returns the local timezone offset in seconds. For example 7200 for CEST.

**Return**

Timezone offset seconds.

Converts time_t to gammu *GSM_DateTime* structure.

**Parameters**

• Date
  Storage for date.
  • timet
    Input date.

Converts timestamp to string according to OS settings.

**Parameters**

• dt
  Input timestamp.
  • TimeZone
    Whether to include time zone.

**Return**

Pointer to static buffer containing string.

Converts date from timestamp to string according to OS settings.

**Parameters**

• dt
  Input timestamp.

**Return**

Pointer to static buffer containing string.
Checks whether date is valid. This does not check time, see \texttt{CheckTime} for this.

\textit{Parameters}

- \texttt{date} - Structure where to check date.

\textit{Return}

True if date is correct.

Checks whether time is valid. This does not check date, see \texttt{CheckDate} for this.

\textit{Parameters}

- \texttt{date} - Structure where to check time.

\textit{Return}

True if time is correct.

Reads date and time from phone.

\textit{Parameters}

- \texttt{s} - State machine pointer.
- \texttt{date\_time} - Storage for date.

\textit{Return}

Error code

Sets date and time in phone.

\textit{Parameters}

- \texttt{s} - State machine pointer.
- \texttt{date\_time} - Date to set.

\textit{Return}

Error code

\textit{struct GSM\_DateTime}

Structure used for saving date and time

\textit{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.

4.3.8 Debug

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.

**Return**

Error code.

Sets debug file.

**Parameters**

• info -
  File path.
• privdi -
  Pointer to debug information data.

**Return**

Error code.

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.

**Return**

Error code.

Returns global debug settings.

**Return**

Pointer to global settings.

Gets debug information for state machine.

**Parameters**
• s -
  State machine data

_Return_
  Debug information.

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 use GSM_GetDebug.

_Parameters_
  • s -
    State machine data

_Return_
  Debug information.

Sets debug level.

_Parameters_
  • info -
    Level as text.
  • privdi -
    Pointer to debug information data.

_Return_
  True on success.

Sets debug encoding.

_Parameters_
  • info -
    Encoding to set.
  • privdi -
    Pointer to debug information data.

_Return_
  True on success.

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.

Return
  True on success.

Logs error to debug log with additional message.

Parameters
  • s -
    State machine structure pointer.
  • message -
    String to be show in message.
  • err -
    Error code.

Prints string to defined debug log.

Parameters
  • s -
    State machine, where to print.
  • format -
    Format string as for printf.

Return
  Upon successful return, these functions return the number of characters printed (as printf).

Debugging configuration.

4.3.9 Error handling

Returns text for error.

Parameters
  • e -
    Error code.

Return
  Text (in current locales) describing error

Returns name for error.

Parameters
• e -
  Error code.

Return
  Text with error name

Error types.

Values:
  • ERR_NONE= 1 -
    No error
  • ERR_DEVICEOPENERROR -
    Error during opening device
  • ERR_DEVICELOCKED -
    Device locked
  • ERR_DEVICENOTEXIST -
    Device does not exits
  • ERR_DEVICEBUSY -
    Device is busy
  • ERR_DEVICENOPERMISSION -
    No permissions to open device
  • ERR_DEVICENODRIVER -
    No driver installed for a device
  • ERR_DEVICENOTWORK -
    Device doesn’t seem to be working
  • ERR.DeviceDTRRTSERROR -
    Error during setting DTR/RTS in device
  • ERR.DEVICECHANGESPEDERROR -
    Error during changing speed in device
  • ERR.DEVICEWRITEERROR -
    Error during writing device
  • ERR.DEVICEREADERERROR -
    Error during reading device
  • ERR.DEVICEPARITYERROR -
    Can’t set parity on device
  • ERR_TIMEOUT -
    Command timed out
• ERR_FRAMENOTREQUESTED -  
  Frame handled, but not requested in this moment
• ERR_UNKNOWNRESPONSE -  
  Response not handled by gammu
• ERR_UNKNOWNFRAME -  
  Frame not handled by gammu
• ERR_UNKNOWNCONNECTIONTYPESTRING -  
  Unknown connection type given by user
• ERR_UNKNOWNMODELSTRING -  
  Unknown model given by user
• ERR_SOURCENOTAVAILABLE -  
  20 Some functions not compiled in your OS
• ERR_NOTSUPPORTED -  
  Not supported by phone
• ERR_EMPTY -  
  Empty entry or transfer end.
• ERR_SECURITYERROR -  
  Not allowed
• ERR_INVALIDLOCATION -  
  Too high or too low location...
• ERR_NOTIMPLEMENTED -  
  Function not implemented
• ERR_FULL -  
  Memory is full
• ERR_UNKNOWN -  
  Unknown response from phone
• ERR_CANTOPENFILE -  
  Error during opening file
• ERR_MOREMEMORY -  
  More memory required
• ERR_PERMISSION -  
  30 No permission
• ERR_EMPTYSMSC -  
  SMSC number is empty
• ERR_INSIDEPHONEMENU -  
  Inside phone menu - can’t make something
• **ERR_NOTCONNECTED** -
  Phone NOT connected - can’t make something
• **ERR_WORKINPROGRESS** -
  Work in progress
• **ERR_PHONEOFF** -
  Phone is disabled and connected to charger
• **ERR_FILENOTSUPPORTED** -
  File format not supported by Gammu
• **ERR_BUG** -
  Found bug in implementation or phone
• **ERR_CANCELED** -
  Action was canceled by user
• **ERR_NEEDEDANOTHERANSWER** -
  Inside Gammu: phone module need to send another answer frame
• **ERR_OTHERCONNECTIONREQUIRED** -
  40 You need other connection for this operation.
• **ERR_WRONGCRC** -
  Wrong CRC
• **ERR_INVALIDDATETIME** -
  Invalid date/time
• **ERR_MEMORY** -
  Phone memory error, maybe it is read only
• **ERR_INVALIDDATA** -
  Invalid data given to phone
• **ERR_FILEALREADYEXIST** -
  File with specified name already exist
• **ERR_FILENOTEXIST** -
  File with specified name doesn’t exist
• **ERR_SHOULDBEFOLDER** -
  You have to give folder (not file) name
• **ERR_SHOULDBEFILE** -
  You have to give file (not folder) name
• **ERR_NOSIM** -
  Can not access SIM card
• **ERR_GNAPPLETWRONG** -
  50 Invalid gnapplet version
- ERR_FOLDERPART -
  Only part of folders listed
- ERR_FOLDERNOTEMPTY -
  Folder is not empty
- ERR_DATACONVERTED -
  Data were converted
- ERR_UNCONFIGURED -
  Gammu is not configured.
- ERR_WRONGFOLDER -
  Wrong folder selected (eg. for SMS).
- ERR_PHONE_INTERNAL -
  Internal phone error (phone got crazy).
- ERR_WRITING_FILE -
  Could not write to a file (on local filesystem).
- ERR_NONE_SECTION -
  No such section exists.
- ERR_USING_DEFAULTS -
  Using default values.
- ERR_CORRUPTED -
  60 Corrupted data returned by phone.
- ERR_BADFEATURE -
  Bad feature string.
- ERR_DISABLED -
  Some functions not compiled in your OS
- ERR_SPECIFYCHANNEL -
  Bluetooth configuration requires channel option.
- ERR_NOTRUNNING -
  Service is not running.
- ERR_NOSERVICE -
  Service setup is missing.
- ERR_BUSY -
  Command failed. Try again.
- ERR_COULDN'T_CONNECT -
  Can not connect to server.
- ERR_COULDN'T_RESOLVE -
  Can not resolve host name.
• **ERR_GETTING_SMSC** -
  Failed to get SMSC number from phone.
• **ERR_LAST_VALUE** -
  Just marker of highest error code, should not be used.

### 4.3.10 File

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.

**Return**
- Error code.

Reads file from filesystem to `GSM_File` structure.

**Parameters**
- **FileName** -
  File to read.
- **File** -
  Storage for data.

**Return**
- Error code.

Identifies file format by checking it’s content.

**Parameters**
• File -
  File data, Type member will be filled in.

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.

Return
  Error code.

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.

Return
  Error code.

Gets next root folder.

Parameters
  • s -
    State machine pointer.
  • File -
    File structure where path will be stored.

Return
  Error code.
Sets file system attributes.

Parameters

- s - State machine pointer.
- File - File structure with path and attributes.

Return

Error code.

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.

Return

Error code, ERR_EMPTY after transfer end.

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.

Return

Error code, ERR_EMPTY after transfer end.
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 \textit{GSM\_AddFilePart}.

\textbf{Parameters}

- \textit{s}\textemdash State machine pointer.
- \textit{File}\textemdash File structure and data.
- \textit{Pos}\textemdash Position of transmitted data. Should be 0 on start.
- \textit{Handle}\textemdash Handle for saving file, some drivers need this information to be kept between function calls.

\textbf{Return}

Error code, ERR\_EMPTY after transfer end.

Acquires filesystem status.

\textbf{Parameters}

- \textit{s}\textemdash State machine pointer.
- \textit{Status}\textemdash Storage for status information.

\textbf{Return}

Error code.

Deletes file from filesystem.

\textbf{Parameters}

- \textit{s}\textemdash State machine pointer.
- \textit{ID}\textemdash ID of folder.

\textbf{Return}

Error code.

Adds folder to filesystem.

\textbf{Parameters}
• `s` - State machine pointer.
• `File` - Structure containing information about new folder (Name and FullName).

**Return**
Error code.

Deletes folder from filesystem.

**Parameters**
• `s` - State machine pointer.
• `ID` - ID of folder.

**Return**
Error code.

```c
struct GSM_FileSystemStatus
```
Status of filesystem.

**Public Members**
- `int Free`
- `int Used`
- `int UsedImages`
- `int UsedSounds`
- `int UsedThemes`

File type identifier.

**Values:**
- `GSM_File_Other` = 1
- `GSM_File_Java_JAR`
- `GSM_File_Image_JPG`
- `GSM_File_Image_BMP`
- `GSM_File_Image_GIF`
- `GSM_File_Image_PNG`
- `GSM_File_Image_WBMP`
- `GSM_File_Video_3GP`
- `GSM_File_Sound_AMR`
• GSM_File_Sound_NRT -
  DCT4 binary format
• GSM_File_Sound_MIDI -
• GSM_File_MMS -

\textit{struct GSM\_File}

Structure for holding file information and data.

\textit{Public Members}

\begin{itemize}
  \item \texttt{size\_t Used}
    \begin{itemize}
    \item How many bytes are used.
    \end{itemize}
  \item \texttt{unsigned char Name[2 *(GSM\_MAX\_FILENAME\_LENGTH+1)]}
    \begin{itemize}
    \item Name in Unicode
    \end{itemize}
  \item \texttt{gboolean Folder}
    \begin{itemize}
    \item True, when folder
    \end{itemize}
  \item \texttt{int Level}
    \begin{itemize}
    \item How much file is nested on filesystem.
    \end{itemize}
  \item \textit{GSM\_FileType Type}
    \begin{itemize}
    \item Type of file.
    \end{itemize}
  \item \texttt{unsigned char ID\_FullName[2 *(GSM\_MAX\_FILENAME\_ID\_LENGTH+1)]}
    \begin{itemize}
    \item ID in Unicode
    \end{itemize}
  \item \texttt{unsigned char * Buffer}
    \begin{itemize}
    \item Pointer to file data.
    \end{itemize}
  \item \textit{GSM\_DateTime Modified}
    \begin{itemize}
    \item Last modification date.
    \end{itemize}
  \item \texttt{gboolean ModifiedEmpty}
    \begin{itemize}
    \item Whether modification date is empty.
    \end{itemize}
  \item \texttt{gboolean Protected}
    \begin{itemize}
    \item Protected file attribute.
    \end{itemize}
  \item \texttt{gboolean ReadOnly}
    \begin{itemize}
    \item Read only file attribute.
    \end{itemize}
  \item \texttt{gboolean Hidden}
    \begin{itemize}
    \item Hidden file attribute.
    \end{itemize}
  \item \texttt{gboolean System}
    \begin{itemize}
    \item System file attribute.
    \end{itemize}
\end{itemize}
4.3.11 Info

Find network name from given network code.

Find country name from given country code.

Converts feature value to string.

Parameters
- **feature**
  GSM_Feature to convert.

Return
- Pointer to static string with string for specified feature, NULL on failure.

Converts feature string to value.

Parameters
- **feature**
  GSM_Feature string to convert.

Return
- GSM_Feature value, 0 on failure.

Checks whether phone supports features.

Parameters
- **model**
  Model information (you can get it using `GSM_GetModelInfo`).
- **feature**
  GSM_Feature to check for.

Return
- True if phone has defined this feature.

Adds feature to phone configuration.

Parameters
- **model**
  Model information (you can get it using `GSM_GetModelInfo`).
- **feature**
  GSM_Feature to check for.
Return
True if phone has defined this feature.

Reads manufacturer from phone.

Parameters
• s -
  State machine pointer.
• value -
  Pointer where to store manufacturer name

Return
Error code.

Reads model from phone.

Parameters
• s -
  State machine pointer.
• value -
  Pointer where to store model name

Return
Error code.

Reads model info from state machine.

Parameters
• s -
  State machine pointer.

Return
Pointer to phone information structure.

Reads firmware information from phone.

Parameters
• s -
  State machine pointer.
• value -
  Pointer where to store revision text
• date -
  Pointer where to store revision date
• \texttt{num} -
  Pointer where to store revision number

\textit{Return}

Error code.

Reads IMEI/serial number from phone.

\textit{Parameters}

• \texttt{s} -
  State machine pointer.

• \texttt{value} -
  Pointer where to store IMEI, NULL to ignore.

\textit{Return}

Error code.

Gets date and time from phone.

Gets month when device was manufactured.

Gets product code of device.

Gets hardware information about device.

Gets PPM (Post Programmable Memory) info from phone (in other words for Nokia get, which language pack is in phone)

Gets SIM IMSI from phone.

Gets information about battery charge and phone charging state.

Reads signal quality (strength and error rate).

Gets network information.

Acquired display status.

Status of network logging
Values:

- **GSM_HomeNetwork** = 1 - Home network for used SIM card.
- **GSM_NoNetwork** - No network available for used SIM card.
- **GSM_RoamingNetwork** - SIM card uses roaming.
- **GSM_RegistrationDenied** - Network registration denied - card blocked or expired or disabled.
- **GSM_NetworkStatusUnknown** - Unknown network status.
- **GSM_RequestingNetwork** - Network explicitly requested by user.

Status of GPRS connection.

Values:

- **GSM_GPRS_Detached** = 1 - GPRS is detached.
- **GSM_GPRS_Attached** - GPRS 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[15 *2]**
  
  Name of current network like returned from phone (or empty).

**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 SignalStrength
int SignalPercent
Signal strength in percent.

int BitErrorRate
Bit error rate in percent.

Power source
Values:
- GSM_BatteryPowered= 1 - Powered from battery
- GSM_BatteryConnected - Powered from AC, battery connected
- GSM_BatteryCharging - Powered from AC, battery is charging
- GSM_BatteryNotConnected - Powered from AC, no battery
- GSM_BatteryFull - Powered from AC, battery is fully charged
- GSM_PowerFault - Power failure

Power source
Values:
- GSM_BatteryUnknown= 0 - Unknown battery
• GSM_BatteryNiMH= 1 -
  NiMH battery
• GSM_BatteryLiIon -
  Lithium Ion battery
• 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

Display feature

Values:

• GSM_CallActive= 1 -
• GSM_SMSMemoryFull -
  blinking envelope
• GSM_FaxCall -
• GSM_UnreadSMS
• GSM_DataCall
• GSM_VoiceCall
• GSM_KeypadLocked

\textit{struct} \texttt{GSM\_DisplayFeatures}

\hspace{1em} Display features

\textit{Public Members}

\begin{verbatim}
int Number

\texttt{GSM\_DisplayFeature Feature[7]}
\end{verbatim}

Phone features definition. This is usually used for things, which can not be determined on run time.

\textit{Values:}

• \texttt{F\_CAL33} = 1 -
  Calendar, 3310 style - 10 reminders, Unicode, 3 coding types
• \texttt{F\_CAL52} -
  Calendar, 5210 style - full Unicode, etc.
• \texttt{F\_CAL82} -
  Calendar, 8250 style - “normal”, but with Unicode
• \texttt{F\_RING\_SM} -
  Ringtones returned in SM format - 33xx
• \texttt{F\_NORING} -
  No ringtones
• \texttt{F\_NOPBKUNICODE} -
  No phonebook in Unicode
• \texttt{F\_NOWAP} -
  No WAP
• \texttt{F\_NOCALLER} -
  No caller groups
• \texttt{F\_NOPICTURE} -
  No Picture Images
• \texttt{F\_NOPICTUREUNI} -
  No Picture Images text in Unicode
• \texttt{F\_NOSTARTUP} -
  No startup logo
• \texttt{F\_NOCALENDAR} -
  No calendar
• F_NOSTARTANI -
  Startup logo is not animated
• F_POWER_BATT -
  Network and battery level get from netmonitor
• F_PROFILES33 -
  Phone profiles in 3310 style
• F_PROFILES51 -
  Phone profiles in 5110 style
• F_MAGICBYTES -
  Phone can make authentication with magic bytes
• F_NODTMF -
  Phone can't send DTMF
• F_DISPSTATUS -
  Phone return display status
• F_NOCALLINFO -
  Phone does not return call info
• F_DAYMONTH -
  Day and month reversed in pbk, when compare to GSM models
• F_PBK35 -
  Phonebook in 3510 style with ringtones ID
• F_PBKIMG -
  Phonebook in 7250 style with picture ID
• F_PBKTONEGAL -
  Phonebook with selecting ringtones from gallery
• F_PBKSMSLIST -
  Phonebook with SMS list
• F_PBKUSER -
  Phonebook with user ID
• F_6230iCALLER -
  Caller groups like in 6230i
• F_RADIO -
  Phone with FM radio
• F_TODO63 -
  ToDo in 6310 style - 0x55 msg type
• F_TODO66 -
  ToDo in 6610 style - like calendar, with date and other
- F_NOMIDI -
  No ringtones in MIDI
- F_BLUETOOTH -
  Bluetooth support
- F_NOFILESYSTEM -
  No images, ringtones, java saved in special filesystem
- F_NOMMS -
  No MMS sets in phone
- F_NOGPRSPOINT -
  GPRS point are not useable
- F_CAL35 -
  Calendar, 3510 style - Reminder, Call, Birthday
- F_CAL65 -
  Calendar, 6510 style - CBMM, method 3
- F_WAPMMS_PROXY -
  WAP & MMS settings contains first & second proxy
- F_CHAT -
  Phone with Chat settings
- F_SYNCML -
  Phone with SyncML settings
- F_FILES2 -
  Filesystem version 2
- F_NOFILE1 -
  No filesystem version 1
- F_6230iWAP -
  WAP, MMS, etc. settings like in 6230i - unknown now
- F_PROFILES -
  Profiles support available
- F_SERIES40_30 -
  Series 40 3.0
- F_SMS_FILES -
  SMS are read from filesystem files like in Series 40 3.0
- F_3220_MMS -
  MMS storage as in 3320
- F_VOICETAGS -
  Voice tags available
• **F_CAL62** -
  Calendar, 6210 style - Call, Birthday, Memo, Meeting

• **F_NOTES** -
  Notes supported

• **F_SMSONLYSENT** -
  Phone supports only sent/unsent messages

• **F_BROKENCPBS** -
  CPBS on some memories can hang phone

• **F_M20SMS** -
  Siemens M20 like SMS handling

• **F_SLOWWRITE** -
  Use slower writing which some phone need

• **F_SMSME900** -
  SMS in ME start from location 900 - case of Sagem

• **F_ALCATEL** -
  Phone supports Alcatel protocol

• **F_OBEX** -
  Phone can switch to OBEX protocol from AT mode

• **F_IRMC_LEVEL_2** -
  Phone supports IrMC level 2 even if it doesn’t report it

• **F_MODE22** -
  Switching to OBEX mode is done using AT+MODE=22

• **F_SMS_LOCATION_0** -
  Locations of SMS memories start from 0

• **F_NO_UCS2** -
  Phone does not support UCS2 even if it reports it.

• **F_FORCE_UTF8** -
  Phone returns strings in utf-8 even if it reports GSM.

• **F_SMS_SM** -
  Phone supports SM storage for SMS even if it does not report so.

• **F_SMS_ME** -
  Phone supports ME storage for SMS even if it does not report so.

• **F_XLNK** -
  Switching to OBEX mode is done using AT+XLNK.

• **F_SUBMIT_SIM_ONLY** -
  Submit messages can be saved on SM memory only.
• **F_PBK_UNICODE**
  Prefer Unicode for phone book manipulations.

• **F_SQWE**
  Switching to OBEX mode using AT^SQWE=3.

• **F_NO_ATOBEX**
  Do not use OBEX/AT switching even if available.

• **F_LENGTH_BYTES**
  Length of text for contact is in bytes and not chars.

• **F_BROKEN_CMGL**
  CMGL does not list real locations for CMGR, these should be sequential.

• **F_EXTRA_PBK_FIELD**
  Phonebook has extra numeric field at the end.

• **F_CKPD_NO_UNICODE**
  Key presses can not be in unicode.

• **F_CPROM**
  OBEX switching using AT+CROM even if phone does not report it properly.

• **F_PBMFAVORITEMESSAGE**
  Phonebook with favorite messaging numbers

• **F_PBMNOPOSTAL**
  No support for postal entry in phonebook.

• **F_PBMENCODENUMBER**
  Encode number in HEX charset.

• **F_NO_CLIP**
  Do not use CLIP (phone hangs on it).

• **F_ENCODED_USSD**
  USSD prompts and responses are encoded like PDU in SMS (packed 7-bit GSM encoding).

• **F_USE_SMSTEXTMODE**
  Phone has better support for SMS text mode (rather than PDU mode)

• **F_CPIN_NO_OK**
  Phone does not end CPIN reply with OK/ERROR.

• **F_FOUR_DIGIT_YEAR**
  Phone requires four digit year in time.

• **F_SMS_NO_ME**
  Phone does not have a phone SMS memory even if it reports so.

• **F_SMS_NO_SM**
  Phone does not have a SIM SMS memory even if it reports so.
• **F_SIEMENS_PBK** -
  Phone supports Siemens style phonebook even if it does not tell so.

• **F_NO_ATSYNCML** -
  Disable AT+SYNCML probing.

• **F_MOBEX** -
  Phone supports m-obex (usually Samsung phones).

• **F_TSSPCSW** -
  Phone supports m-obex (usually Samsung phones) using AT&TSSPCSW=1.

• **F_DISABLE_GETNEXT** -
  Disable GetNext* operations on the dummy phone.

• **F_DISABLE_GETNEXTSMS** -
  Disable GetNextSMS operations on the dummy phone.

• **F_DISABLE_CMGL** -
  CMGL hangs, so should not be used.

• **F_LAST_VALUE** -
  Just marker of highest feature code, should not be used.

```c
struct GSM_PhoneModel
{
    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
}
```

### 4.3.12 INI files

Free INI data.

*Parameters*

• **head** -
  INI section data.
Reads INI data.

**Parameters**
- **FileName** - File to read.
- **Unicode** - Whether file should be treated like unicode.
- **result** - Pointer where file will be read.

**Return**
Error code

Returns pointer to last INI entry of given section.

**Parameters**
- **file_info** - File data as returned by `INI_ReadFile`.
- **section** - Section to scan.
- **Unicode** - Whether file is unicode.

**Return**
Last entry in section.

Returns value of INI file entry.

**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.

**Return**
Entry value.
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.

**Return**

Key value or fallback in case of failure.

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.

**Return**

Key value or fallback in case of failure.

Converts value to boolean.

It just takes the string and checks whether there is true/yes/y/1 or false/no/n/0.

**Parameters**

- **value** -
  String to parse.

**Return**

Boolean value, -1 on failure.

Private structure holding information INI entry.
Private structure holding information INI section.

```c
struct _INI_Entry
    Structure used to save value for single key in INI style file
    Public Members
        INI_Entry * Next
        INI_Entry * Prev
        unsigned char * EntryName
        unsigned char * EntryValue
```

```c
struct _INI_Section
    Structure used to save section in INI style file
    Public Members
        INI_Section * Next
        INI_Section * Prev
        INI_Entry * SubEntries
        unsigned char * SectionName
```

### 4.3.13 Keys

Creates key sequence from string.

**Parameters**

- **text**
  - Text to convert.
- **KeyCode**
  - Storage for key codes.
- **Length**
  - Storage for resulting length.

**Return**

Error code.

Emulates key press or key release.

Key event identifiers.

**Values:**

- `GSM_KEY_NONE= 0x00`
- `GSM_KEY_1= 0x01`
• GSM_KEY_2 -
• GSM_KEY_3 -
• GSM_KEY_4 -
• GSM_KEY_5 -
• GSM_KEY_6 -
• GSM_KEY_7 -
• GSM_KEY_8 -
• GSM_KEY_9 -
• GSM_KEY_0 -
• GSM_KEY_HASH -
  
#
• GSM_KEY_ASTERISK -
  *
• GSM_KEY_POWER -
  Power key.
• GSM_KEY_GREEN -
  in some phone ie. N5110 sometimes works identical to POWER
• GSM_KEY_RED -
  (c) key in some phone: ie. N5110
• GSM_KEY_INCREASEVOLUME -
  Not available in some phones as separate button: ie. N5110
• GSM_KEY_DECREASEVOLUME -
  Not available in some phones as separate button: ie. N5110
• GSM_KEY_UP= 0x17 -
• GSM_KEY_DOWN -
• GSM_KEY_MENU -
• GSM_KEY_NAMES -
  Not available in some phone: ie. N5110
• GSM_KEY_LEFT -
  Left arrow
• GSM_KEY_RIGHT -
  Right arrow
• GSM_KEY_SOFT1 -
  Software key which has assigned mening on display.
• GSM_KEY_SOFT2 -
  Software key which has assigned mening on display.
• GSM_KEY_HEADSET - Button on headset
• GSM_KEY_JOYSTICK - Joystick pressed
• GSM_KEY_CAMERA - Camera button pressed
• GSM_KEY_MEDIA - Media player button
• GSM_KEY_DESKTOP - Multi function key, desktop
• GSM_KEY_OPERATOR - Operator button
• GSM_KEY_RETURN - Return button
• GSM_KEY_CLEAR - Clear button

4.3.14 Limits

4.3.15 Memory

Converts memory type from string.

Parameters
• \( s \) - String with memory type.

Return
Parsed memory type or 0 on failure.

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

Parameters
• \( s \) - State machine pointer.
• \( \text{status} \) - Storage for status information, MemoryType has to be set.

Return
Error code.
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.

**Return**
Error code.

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.

**Return**
Error code.

Sets memory (phonebooks or calls) entry.

**Parameters**
- `s` - State machine pointer.
- `entry` - Entry to set. Location and MemoryType has to be set.

**Return**
Error code.

Deletes memory (phonebooks or calls) entry.

**Parameters**
- `s` - State machine pointer.
• entry -
  Entry to add, Location is ignored, MemoryType has to be set.

*Return*
  Error code.

Deletes memory (phonebooks or calls) entry.

*Parameters*

• s -
  State machine pointer.

• entry -
  Entry to delete, Location and MemoryType has to be set.

*Return*
  Error code.

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

*Parameters*

• s -
  State machine pointer.

• MemoryType -
  Where to delete all entries.

*Return*
  Error code.

Gets speed dial.

*Parameters*

• s -
  State machine pointer.

• Speed -
  Storage for speed dial, Location has to be set.

*Return*
  Error code.

Sets speed dial.

*Parameters*

• s -
  State machine pointer.
• Speed -
  Speed dial to set.

Return
  Error code.

Returns name of entry. It might be possibly concatenated from first and last names.

Parameters
  • entry -
    Entry to process.

Return
  Static unicode string containing name.

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.

Encodes memory entry to vCard.

Parameters
  • di -
    Pointer to debugging description.
  • Buffer -
    Buffer to store vCard text.
  • buff_len -
    Size of output buffer.
  • Pos -
    Position in output buffer.
  • pbk -
    Phonebook data, AddError will be set on non converted entries.
• **header** -
  Whether to include vCard header in output.

• **Version** -
  What vCard version to create.

**Return**

Error code.

Decodes memory entry from vCard.

**Parameters**

• **di** -
  Pointer to debugging description.

• **Buffer** -
  Buffer to readCard text.

• **Pos** -
  Position in output buffer.

• **Pbk** -
  Phonebook data read from vCard.

• **Version** -
  What vCard version to parse.

**Return**

Error code.

Frees any dynamically allocated memory inside memory entry structure.

**Parameters**

• **Entry** -
  Pointer to memory entry to process.

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:**

• **MEM_ME** = 1 -
  Internal memory of the mobile equipment

• **MEM_SM** -
  SIM card memory

• **MEM_ON** -
  Own numbers
• MEM_DC -
  Dialled calls
• MEM_RC -
  Received calls
• MEM_MC -
  Missed calls
• MEM_MT -
  Combined ME and SIM phonebook
• MEM_FD -
  Fixed dial
• MEM_VM -
  Voice mailbox
• MEM_SL -
  Sent SMS logs
• MEM_QD -
  Quick dialing choices.

\textbf{struct GSM\_MemoryStatus}

Structure contains info about number of used/free entries in phonebook memory.

\textit{Public Members}

\begin{itemize}
  \item \texttt{int MemoryUsed}
    Number of used entries
  \item \texttt{GSM\_MemoryType MemoryType}
    Memory type
  \item \texttt{int MemoryFree}
    Number of free entries
\end{itemize}

Type of specific phonebook entry. In parenthesis is specified in which member of \texttt{GSM\_SubMemoryEntry} value is stored.

\textit{Values:}

\begin{itemize}
  \item PBK\_Number\_General= 1 -
    General number. (Text)
  \item PBK\_Number\_Mobile -
    Mobile number. (Text)
  \item PBK\_Number\_Work -
    Work number. (Text)
\end{itemize}
• PBK_Number_Fax - Fax number. (Text)
• PBK_Number_Home - Home number. (Text)
• PBK_Number_Pager - Pager number. (Text)
• PBK_Number_Other - Other number. (Text)
• PBK_Text_Note - Note. (Text)
• PBK_Text_Postal - Complete postal address. (Text)
• PBK_Text_Email - Email. (Text)
• PBK_Text_Email2 -
• PBK_Text_URL - URL (Text)
• PBK_Date - Date and time of last call. (Date)
• PBK_Caller_Group - Caller group. (Number)
• PBK_Text_Name - Name (Text)
• PBK_Text_LastName - Last name. (Text)
• PBK_Text_FirstName - First name. (Text)
• PBK_Text_Company - Company. (Text)
• PBK_Text_JobTitle - Job title. (Text)
• PBK_Category - Category. (Number, if -1 then text)
• PBK_Private - Whether entry is private. (Number)
• PBK_Text_StreetAddress - Street address. (Text)
• PBK_Text_City - City. (Text)
• PBK_Text_State - State. (Text)
• PBK_Text_Zip - Zip code. (Text)
• PBK_Text_Country - Country. (Text)
• PBK_Text_Custom1 - Custom information 1. (Text)
• PBK_Text_Custom2 - Custom information 2. (Text)
• PBK_Text_Custom3 - Custom information 3. (Text)
• PBK_Text_Custom4 - Custom information 4. (Text)
• PBK_RingtoneID - Ringtone ID. (Number)
• PBK_PictureID - Picture ID. (Number)
• PBK_Text_UserID - User ID. (Text)
• PBK_CallLength - Length of call (Number)
• PBK_Text_LUID - LUID - Unique Identifier used for synchronisation (Text)
• PBK_LastModified - Date of last modification (Date)
• PBK_Text_NickName - Nick name (Text)
• PBK_Text_FormalName - Formal name (Text)
• PBK_Text_WorkStreetAddress - Work street address. (Text)
• PBK_Text_WorkCity - 
  Work city. (Text)
• PBK_Text_WorkState - 
  Work state. (Text)
• PBK_Text_WorkZip - 
  Work zip code. (Text)
• PBK_Text_WorkCountry - 
  Work country. (Text)
• PBK_Text_WorkPostal - 
  Complete work postal address. (Text)
• PBK_Text_PictureName - 
  Picture name (on phone filesystem). (Text)
• PBK_PushToTalkID - 
  Push-to-talk ID (Text)
• PBK_Number_Messaging - 
  Favorite messaging number. (Text)
• PBK_Photo - 
  Photo (Picture).
• PBK_Number_Mobile_Home - 
  Home mobile number. (Text)
• PBK_Number_Mobile_Work - 
  Work mobile number. (Text)

**struct GSM_SubMemoryEntry**

One value of phonebook memory entry.

Public Members

  * **GSM_EntryType** EntryType
    Type of 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.

int SMList[20]

int CallLength
**GSM_Error AddError**

During adding SubEntry Gammu can return here info, if it was done OK

```c
unsigned char Text[(GSM_PHONEBOOK_TEXT_LENGTH+1)*2]
```

Text of entry (if applicable, see *GSM_EntryType*).

**GSM_BinaryPicture Picture**

Picture data.

```c
text GSM_MemoryEntry
```

Structure for saving phonebook entries.

**Public Members**

**GSM_MemoryType MemoryType**

Used memory for phonebook entry

```c
int Location
```

Used location for phonebook entry

```c
int EntriesNum
```

Number of SubEntries in Entries table.

**GSM_SubMemoryEntry Entries[GSM_PHONEBOOK_ENTRIES]**

Values of SubEntries.

```c
text GSM_SpeedDial
```

Structure for saving speed dials.

**Public Members**

```c
int Location
```

Number of speed dial: 2,3..,8,9

```c
int MemoryNumberID
```

ID of phone number used in phonebook entry

**GSM_MemoryType MemoryType**

Memory, where is saved used phonebook entry

```c
int MemoryLocation
```

Location in memory, where is saved used phonebook entry

Types of vCard.

**Values:**

- **Nokia_VCard10= 1**
  
  vCard 1.0 hacked for Nokia.

- **Nokia_VCard21**
  
  vCard 2.1 hacked for Nokia.
• SonyEricsson_vCard10 -  
  vCard 1.0 hacked for Sony-Ericsson (should be standard vCard).

• SonyEricsson_vCard21 -  
  vCard 2.1 hacked for Sony-Ericsson (should be standard vCard).

• SonyEricsson_vCard21_Phone -  
  vCard 2.1 hacked for Sony-Ericsson (should be standard vCard) from phone (no parsing of location and memory type).

### 4.3.16 Messages

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.

Decodes SMS frame.

Finds out coding type based on TPDCS header byte as defined by GSM 03.38.

Encodes SMS frame.

Decodes SMS frame for status report.

Decodes SMS frame in textual representation.

Decodes UDH header.
Encodes UDH header.

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.

Sets default content for SMS. Use this for clearing structure.

*Parameters*

- SMS -

  Pointer to structure which should be cleaned up.

Decodes Siemens OTA data.

Encodes SMS frame according to layout.

*Return*

- Error code.

Encodes multi part SMS from “readable” format.

*Return*

- Error code.

Decodes multi part SMS to “readable” format.

Clears $GSM\_MultiPartSMSInfo$ to default values.

Frees any allocated structures inside $GSM\_MultiPartSMSInfo$.

Links SMS messages according to IDs.

*Return*

- Error code.

Decodes MMS data.
Clears MMS data, used to initialize structure.

Gets SMS Service Center number and SMS settings.

Parameters

- `s` - State machine pointer.
- `smnc` - SMSC structure, should contain location.

Return

Error code.

Sets SMS Service Center number and SMS settings.

Parameters

- `s` - State machine pointer.
- `smnc` - SMSC structure.

Return

Error code.

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

Parameters

- `s` - State machine pointer.
- `status` - Pointer to SMS status structure.

Return

Error code.

Reads SMS message.

Parameters

- `s` - State machine pointer.
- `sms` - SMS message data read from phone, location and folder should be set.
Returns 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 command 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` - SMS message data read from phone, for subsequent reads, location and folder might be used by phone driver to determine reading state.
- `start` - Whether we start reading from beginning.

**Return**

Error code.

Sets SMS.

**Parameters**

- `s` - State machine pointer.
- `sms` - SMS message data.

**Return**

Error code.

Adds SMS to specified folder.

**Parameters**

- `s` - State machine pointer.
- `sms` - SMS message data, location will be updated.

**Return**

Error code.
Deletes SMS.

**Parameters**

- `s` - State machine pointer.
- `sms` - SMS structure with SMS location and folder.

**Return**

Error code.

Sends SMS.

**Parameters**

- `s` - State machine pointer.
- `sms` - SMS structure with SMS data to send.

**Return**

Error code.

Sends SMS already saved in phone.

**Parameters**

- `s` - State machine pointer.
- `Folder` - Folder, where message is stored.
- `Location` - Location, where message is stored.

**Return**

Error code.

Configures fast SMS sending.

**Parameters**

- `s` - State machine pointer.
- `enable` - Whether to enable notifications.
Enable/disable notification on incoming SMS.

Parameters

- `s` - State machine pointer.
- `enable` - Whether to enable notifications.

Return

Error code.

Gets network information from phone.

Parameters

- `s` - State machine pointer.
- `enable` - Whether to enable notifications.

Return

Error code.

Returns SMS folders information.

Parameters

- `s` - State machine pointer.
- `folders` - folders Pointer to folders structure, which will be filled in.

Return

Error code.

Creates SMS folder.

Parameters

- `s` - State machine pointer.
- `name` - Name of SMS folder which should be created.

Return

Error code.
Returns

Error code.

Deletes SMS folder.

Parameters

- `s`
  - State machine pointer.
- `ID`
  - ID of SMS folder to delete.

Returns

Error code.

Lists MMS folders.

Parameters

- `s`
  - State machine pointer.
- `folders`
  - Pointer to structure, where folder information will be stored.

Returns

Error code.

Retrieves next part of MMS file information.

Parameters

- `s`
  - State machine pointer.
- `FileID`
  - File ID will be stored here, might be used for consequent reads.
- `MMSFolder`
  - MMS folder ID will be stored here, might be used for consequent reads.
- `start`
  - Whether to start reading.

Returns

Error code.

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

Parameters
• s -
  State machine pointer.

• enable -
  Whether to enable notifications.

**Return**

Error code.

MMS message class.

**Values:**

• GSM_MMS_None = 0 -
  None class specified.

• GSM_MMS_Personal = 1 -
  Personal message.

• GSM_MMS_Advertisement -
  Advertisement message.

• GSM_MMS_Info -
  Informational message.

• GSM_MMS_Auto -
  Automatic message class.

**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 of a message.

**struct GSM_CBMessage**

Structure for Cell Broadcast messages.

**Public Members**
int Channel
  Channel number.

char Text[300]
  Message text.

Status of USSD message.

Values:

- USSD_Unknown= 1 - Unknown status
- USSD_NoActionNeeded - No action is needed, maybe network initiated USSD
- USSD_ActionNeeded - Reply is expected
- USSD_Terminated - USSD dialog terminated
- USSD_AnotherClient - Another client replied
- USSD_NotSupported - Operation not supported
- 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 defines format of SMS messages. See GSM 03.40 section 9.2.3.9

Values:
    • SMS_FORMAT_Pager = 1 -
    • SMS_FORMAT_Fax -
    • SMS_FORMAT_Email -
    • SMS_FORMAT_Text -

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:
    • SMS_VALID_1_Hour = 0x0b -
    • SMS_VALID_6_Hours = 0x47 -
    • SMS_VALID_1_Day = 0xa7 -
    • SMS_VALID_3_Days = 0xa9 -
    • SMS_VALID_1_Week = 0xad -
    • SMS_VALID_Max_Time = 0xff -

Enum defines format of validity period for SMS messages. See GSM 03.40 section 9.2.3.12

Values:
    • SMS_Validity_NotAvailable = 1 -
    • SMS_Validity_RelativeFormat -

struct GSM_SMSValidity
    Structure for validity of SMS messages

Public Members

    GSM_ValidityPeriodFormat Format
**GSM_ViabilityPeriod** 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

Status of SMS message.

**Values:**

- SMS_Sent= 1 -
- SMS_UnSent -
- SMS_Read -
- SMS_UnRead -

Coding type of SMS.

**Values:**

- SMS_Coding_Unicode_No_Compression= 1 -
  
  Unicode
- SMS_Coding_Unicode_Compression -
- SMS_Coding_Default_No_Compression -
  
  Default GSM alphabet.
- SMS_Coding_Default_Compression -
- SMS_Coding_8bit -
  
  8-bit.
Types of UDH (User Data Header).

Values:

- UDH_NoUDH= 1 -
- UDH_ConcatenatedMessages -
  Linked SMS.
- UDH_ConcatenatedMessages16bit -
  Linked SMS with 16 bit reference.
- UDH_DisableVoice -
- UDH_DisableFax -
- UDH_DisableEmail -
- UDH_EnableVoice -
- UDH_EnableFax -
- UDH_EnableEmail -
- UDH_VoidSMS -
- UDH_NokiaRingtone -
- UDH_NokiaRingtoneLong -
- UDH_NokiaOperatorLogo -
- UDH_NokiaOperatorLogoLong -
- UDH_NokiaCallerLogo -
- UDH_NokiaWAP -
- UDH_NokiaWAPLong -
- UDH_NokiaCalendarLong -
- UDH_NokiaProfileLong -
- UDH_NokiaPhonebookLong -
- UDH_UserUDH -
- 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.

TP-Message-Type-Indicator. See GSM 03.40 section 9.2.3.1.

Values:
• SMS_Deliver= 1 - SMS in Inbox.
• SMS_Status_Report - Delivery Report
• 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.

unsigned char OtherNumbers[GSM_SMS_OTHER_NUMBERS][(GSM_MAX_NUMBER_LENGTH+1)*2]

int OtherNumbersNum

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
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
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:
  Byte contains in SMS-Submit:
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.

```c
int Number
```

Number of SMS folders.

**struct GSM_SiemensOTASMSInfo**

Siemens OTA data.

**Public Members**

```c
unsigned long SequenceID
unsigned int PacketsNum
unsigned int PacketNum
unsigned long AllDataLen
unsigned char DataType[10]
unsigned char DataName[40]
unsigned int DataLen
unsigned char Data[140]
```

**struct GSM_MultiSMSMessage**

Multiple SMS messages, used for Smart Messaging 3.0/EMS.

**Public Members**

```c
int Number
```

Number of messages.

**GSM_SMSMessage** SMS[GSM_MAX_MULTI_SMS]

Array of SMSes.

**struct GSM_OneMMSFolder**

Information about MMS folder.

**Public Members**

```c
gboolean InboxFolder
```

Whether it is really inbox.

```c
char Name[(GSM_MAX_MMS_FOLDER_NAME_LEN+1)*2]
```

Name for MMS folder.

**struct GSM_MMSFolders**

List of MMS folders.

**Public Members**

```c
unsigned char Number
```

Number of MMS folders.

**GSM_OneMMSFolder** Folder[GSM_MAX_MMS_FOLDERS]

Array of structures holding information about each folder.
ID during packing SMS for Smart Messaging 3.0, EMS and other

*Values:*

- **SMS_Text= 1**
  1 text SMS.
- **SMS_ConcatenatedTextLong**
  Concatenated SMS, when longer than 1 SMS.
- **SMS_ConcatenatedAutoTextLong**
  Concatenated SMS, auto Default/Unicode coding.
- **SMS_ConcatenatedTextLong16bit**
- **SMS_ConcatenatedAutoTextLong16bit**
- **SMS_NokiaProfileLong**
  Nokia profile = Name, Ringtone, ScreenSaver
- **SMS_NokiaPictureImageLong**
  Nokia Picture Image + (text)
- **SMS_NokiaScreenSaverLong**
  Nokia screen saver + (text)
- **SMS_NokiaRingtone**
  Nokia ringtone - old SM2.0 format, 1 SMS
- **SMS_NokiaRingtoneLong**
  Nokia ringtone concatenated, when very long
- **SMS_NokiaOperatorLogo**
  Nokia 72x14 operator logo, 1 SMS
- **SMS_NokiaOperatorLogoLong**
  Nokia 72x14 op logo or 78x21 in 2 SMS
- **SMS_NokiaCallerLogo**
  Nokia 72x14 caller logo, 1 SMS
- **SMS_NokiaWAPBookmarkLong**
  Nokia WAP bookmark in 1 or 2 SMS
- **SMS_NokiaWAPSettingsLong**
  Nokia WAP settings in 2 SMS
- **SMS_NokiaMMSSettingsLong**
  Nokia MMS settings in 2 SMS
- **SMS_NokiaVCARD10Long**
  Nokia VCARD 1.0 - only name and default number
• SMS_NokiaVCARD21Long -
  Nokia VCARD 2.1 - all numbers + text
• SMS_NokiaVCALENDAR10Long -
  Nokia VCALENDAR 1.0 - can be in few sms
• SMS_NokiaVTODOLong -
• SMS_VCARD10Long -
• SMS_VCARD21Long -
• SMS_DisableVoice -
• SMS_DisableFax -
• SMS_DisableEmail -
• SMS_EnableVoice -
• SMS_EnableFax -
• SMS_EnableEmail -
• SMS_VoidSMS -
• SMS_EMSSound10 -
  IMelody 1.0
• SMS_EMSSound12 -
  IMelody 1.2
• SMS_EMSSonyEricssonSound -
  IMelody without header - SonyEricsson extension
• SMS_EMSSound10Long -
  IMelody 1.0 with UPI.
• SMS_EMSSound12Long -
  IMelody 1.2 with UPI.
• SMS_EMSSonyEricssonSoundLong -
  IMelody without header with UPI.
• SMS_EMSPredefinedSound -
• SMS_EMSPredefinedAnimation -
• SMS_EMSSAnimation -
• SMS_EMSSFixedBitmap -
  Fixed bitmap of size 16x16 or 32x32.
• SMS_EMSSVariableBitmap -
• SMS_EMSSVariableBitmapLong -
• SMS_MMSIndicatorLong -
  MMS message indicator.
• SMS_WAPIndicatorLong -
• SMS_AlcatelMonoBitmapLong
  Variable bitmap with black and white colors
• SMS_AlcatelMonoAnimationLong
  Variable animation with black and white colors
• SMS_AlcatelSMSTemplateName
• SMS_SiemensFile
  Siemens OTA

\textit{struct} GSM\_MultiPartSMSEntry

Entry of multipart SMS.

\textit{Public Members}

\begin{verbatim}
EncodeMultiPartSMSID ID
int Number
GSM\_Ringtone * Ringtone
GSM\_MultiBitmap * Bitmap
GSM\_WAPBookmark * Bookmark
GSM\_WAPSettings * Settings
GSM\_MMSIndicator * MMSIndicator
GSM\_MemoryEntry * Phonebook
GSM\_CalendarEntry * Calendar
GSM\_ToDoEntry * ToDo
GSM\_File * File
gboolean Protected
unsigned char * Buffer
gboolean Left
gboolean Right
gboolean Center
gboolean Large
gboolean Small
gboolean Bold
gboolean Italic
gboolean Underlined
gboolean Strikethrough
int RingtoneNotes
\end{verbatim}

\textit{struct} GSM\_MultiPartSMSInfo

Multipart SMS information.

\textit{Public Members}
int EntriesNum
 gboolean UnicodeCoding
 int Class
 unsigned char ReplaceMessage
 gboolean Unknown

GSM_MultiPartSMSEntry Entries[GSM_MAX_MULTI_SMS]

MMS address type.

Values:

- MMSADDRESS_PHONE
- MMSADDRESS_UNKNOWN

struct GSM_EncodedMultiPartMMSEntry

MMS entry.

Public Members

GSM_File File

unsigned char ContentType[400]
  CT in Unicode

unsigned char SMIL[400]
  Smil ID in Unicode

struct GSM_EncodedMultiPartMMSInfo

MMS part.

Public Members

int EntriesNum

unsigned char Source[200]
  in Unicode

MMSAddressType SourceType

unsigned char Destination[200]
  in Unicode

MMSAddressType DestinationType

unsigned char CC[200]
  in Unicode

MMSAddressType CCType

unsigned char Subject[200]
  in Unicode

unsigned char ContentType[400]
  CT in Unicode
unsigned char MSGType[50]
    no Unicode
gboolean DateTimeAvailable
    GSM_DateTime DateTime

gboolean MMSReportAvailable

gboolean MMSReport
    GSM_EncodedMultiPartMMSEntry Entries[GSM_MAX_MULTI_MMS]
    Subparts.

4.3.17 Miscellaneous

Reads single line from file.

**Parameters**

- **File** - File descriptor to read from.
- **Line** - Buffer where to store result.
- **count** - Maximal length of text which can be stored in buffer.

**Return**

Length of read line. -1 on error.

Gets Gammu library version.

Gets compiler which was used to compile Gammu library.

Gets host OS.

Returns path to Gammu locales.

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.
Encodes text to hexadecimal binary representation.

Returns TRUE if firmware version is newer.

**Parameters**
- **latest_version** - String containing version (e.g. latest available).
- **current_version** - String containing version (e.g. current one).

**Return**
True if latest_version > current_version.

### 4.3.18 Nokia

Gets default caller group name.

**Parameters**
- **Bitmap** - Storage for default bitmap.

Gets default profile name.

**Parameters**
- **Profile** - Storage for default profile.

### 4.3.19 Ringtone

Play one note using state machine interface.

Makes phone beek using state machine interface.

Gets ringtone from phone.

Sets ringtone in phone.

Acquires ringtone information.
Deletes user defined ringtones from phone.

Plays tone.

Returns ringtone name, NULL if not found.

**Values:**

- `NaturalStyle` = `0x00 << 6`
  - Natural style (rest between notes)
- `ContinuousStyle` = `0x01 << 6`
  - Continuous style (no rest between notes)
- `StaccatoStyle` = `0x02 << 6`
  - Staccato style (shorter notes and longer rest period)

**Values:**

- `Note_Pause` = `0x00 << 4`
- `Note_C` = `0x01 << 4`
- `Note_Cis` = `0x02 << 4`
- `Note_D` = `0x03 << 4`
- `Note_Dis` = `0x04 << 4`
- `Note_E` = `0x05 << 4`
- `Note_F` = `0x06 << 4`
- `Note_Fis` = `0x07 << 4`
- `Note_G` = `0x08 << 4`
- `Note_Gis` = `0x09 << 4`
- `Note_A` = `0x0a << 4`
• Note_Ais = 0x0b << 4 -
• Note_H = 0x0c << 4 -

Values:
• Duration_Full = 0x00 << 5 -
• Duration_1_2 = 0x01 << 5 -
• Duration_1_4 = 0x02 << 5 -
• Duration_1_8 = 0x03 << 5 -
• Duration_1_16 = 0x04 << 5 -
• Duration_1_32 = 0x05 << 5 -

Values:
• NoSpecialDuration = 0x00 << 6 -
• DottedNote = 0x01 << 6 -
• DoubleDottedNote = 0x02 << 6 -
• Length_2_3 = 0x03 << 6 -

Values:
• Scale_55 = 1 -
  55 Hz for note A
• Scale_110 -
  110 Hz for note A
• Scale_220 -
• Scale_440 -
  first scale for Nokia
• Scale_880 -
• Scale_1760 -
• Scale_3520 -
  last scale for Nokia
• Scale_7040 -
• Scale_14080 -

struct GSM_RingNote

Public Members

GSM_RingNoteDuration Duration
GSM_RingNoteDurationSpec DurationSpec
GSM_RingNoteNote Note
**GSM_RingNoteStyle** Style

**GSM_RingNoteScale** Scale

int Tempo

**Values:**

- RING_Note= 1-
- RING_EnableVibra-
- RING_DisableVibra-
- RING_EnableLight-
- RING_DisableLight-
- RING_EnableLED-
- RING_DisableLED-
- RING_Repeat-

**struct GSM_RingCommand**

**Public Members**

- **GSM_RingCommandType** Type
- **GSM_RingNote** Note
- unsigned char Value

**struct GSM_NoteRingtone**

**Public Members**

- int NrCommands
- gboolean AllNotesScale
- **GSM_RingCommand** Commands[GSM_MAX_RINGTONE_NOTES]

**struct GSM_NokiaBinaryRingtone**

**Public Members**

- unsigned char Frame[50000]
- size_t Length

**struct GSM_BinaryTone**

**Public Members**

- unsigned char * Buffer
- size_t Length

**Values:**

- RING_NOTETONE= 1-
- RING_NOKIABINARY-
- RING_MIDI-
• RING_MMF -

**struct GSM_Ringtone**

Structure for saving various ringtones formats

*Public Members*

* GSM_NokiaBinaryRingtone NokiaBinary
  * Ringtone saved in one of three formats
* GSM_BinaryTone BinaryTone
* GSM_NoteRingtone NoteTone
* 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
* int ID
  * `unsigned char Name[30 *2]`

**struct GSM_AllRingtonesInfo**

*Public Members*

* int Number
  * `GSM_RingtoneInfo * Ringtone`

### 4.3.20 Security

Enters security code (PIN, PUK,...) .

Queries whether some security code needs to be entered.

Definition of security codes.

*Values:*

  • SEC_SecurityCode= 0x01 -
    * Security code.*
• SEC_Pin - PIN.
• SEC_Pin2 - PIN 2.
• SEC_Puk - PUK.
• SEC_Puk2 - PUK 2.
• SEC_None -
  Code not needed.
• SEC_Phone -
  Phone code needed.
• SEC_Network -
  Network code needed.

`struct GSM_SecurityCode`

Security code definition.

`Public Members`

```
GSM_SecurityCodeType Type
  Type of the code.

c char Code[GSM_SECURITY_CODE_LEN+1]
  Actual code.

c char NewPIN[GSM_SECURITY_CODE_LEN+1]
  New PIN code.
  Some phones require to set PIN on entering PUK, you can provide it here.
```

### 4.3.21 Settings

Gets locale from phone.

Sets locale of phone.

Acquires SyncML settings.

Changes SyncML settings.

Acquires chat/presence settings.
Changes chat/presence settings.

Acquires MMS settings.

Changes MMS settings.

Enables network auto login.

Performs phone reset.

Resets phone settings.

Reads profile.

Updates profile.

Reads FM station.

Sets FM station.

Clears defined FM stations.

Gets GPRS access point.

Sets GPRS access point.

*struct* GSM_SyncMLSettings

*Public Members*

- int Location
- gboolean Active
- gboolean SyncPhonebook
- gboolean SyncCalendar
- char Name[(20+1)*2]
- char PhonebookDataBase[(50+1)*2]
- char CalendarDataBase[(50+1)*2]
- char User[(30+1)*2]
char Password[(20+1)*2]
char Server[(128+1)*2]

_GSM_MultiWAPSettings_ Connection

Values:

- GSM_RESET_PHONESETTINGS= 1 -
- GSM_RESET_USERINTERFACE -
- GSM_RESET_USERINTERFACE_PHONESETTINGS -
- GSM_RESET_DEVICE -
- GSM_RESET_FULLFACTORY -

_struct GSM_ChatSettings_

_Public Members_

char Name[(50+1)*2]
char HomePage[(200+1)*2]
char User[(50+1)*2]
char Password[(50+1)*2]
int Location

gboolean Active

_GSM_MultiWAPSettings_ Connection

Values:

- PROFILE_KEYPAD_LEVEL1= 1 -
- PROFILE_KEYPAD_LEVEL2 -
- PROFILE_KEYPAD_LEVEL3 -
- PROFILE_KEYPAD_OFF -
- PROFILE_CALLALERT_RINGING -
- PROFILE_CALLALERT_BEEPONCE -
- PROFILE_CALLALERT_OFF -
- PROFILE_CALLALERT_RINGONCE -
- PROFILE_CALLALERT_ASCENDING -
- PROFILE_CALLALERT_CALLERGROUPS -
- PROFILE_VOLUME_LEVEL1 -
- PROFILE_VOLUME_LEVEL2 -
- PROFILE_VOLUME_LEVEL3 -
- PROFILE_VOLUME_LEVEL4 -
- PROFILE_VOLUME_LEVEL5 -

4.3. _libGammu_ C API
• PROFILE_MESSAGE_NOTONE -
• PROFILE_MESSAGE_STANDARD -
• PROFILE_MESSAGE_SPECIAL -
• PROFILE_MESSAGE_BEEPONCE -
• PROFILE_MESSAGE_ASCENDING -
• PROFILE_MESSAGE_PERSONAL -
• PROFILE_VIBRATION_OFF -
• PROFILE_VIBRATION_ON -
• PROFILE_VIBRATION_FIRST -
• PROFILE_WARNING_ON -
• PROFILE_WARNING_OFF -
• PROFILE_AUTOANSWER_ON -
• PROFILE_AUTOANSWER_OFF -
• PROFILE_LIGHTS_OFF -
• PROFILE_LIGHTS_AUTO -
• PROFILE_SAVER_ON -
• PROFILE_SAVER_OFF -
• PROFILE_SAVER_TIMEOUT_5SEC -
• PROFILE_SAVER_TIMEOUT_20SEC -
• PROFILE_SAVER_TIMEOUT_1MIN -
• PROFILE_SAVER_TIMEOUT_2MIN -
• PROFILE_SAVER_TIMEOUT_5MIN -
• PROFILE_SAVER_TIMEOUT_10MIN -

Values:
• Profile_KeypadTone= 1 -
• Profile_CallAlert -
• Profile_RingtoneVolume -
• Profile_MessageTone -
• Profile_Vibration -
• Profile_WarningTone -
• Profile_AutoAnswer -
• Profile_Lights -
• Profile_ScreenSaverTime -
• Profile_ScreenSaver -
• Profile_ScreenSaverNumber -
• Profile_RingtoneID -
• Profile_MessageToneID -
• Profile_CallerGroups -

struct GSM_Profile
It contains phone profiles

Public Members

gboolean Active
int Location
Profile number
char Name[40 *2]
Profile name
gboolean DefaultName
Is it default name for profile ?
gboolean HeadSetProfile
gboolean CarKitProfile
int FeaturesNumber
GSM_Profile_Feat_Value FeatureValue[15]
GSM_Profile_Feat_ID FeatureID[15]
gboolean CallerGroups[5]

struct GSM_FMStation
Public Members

int Location
double Frequency
char StationName[(GSM_MAX_FMSTATION_LENGTH+1)*2]

struct GSM_GPRSAccessPoint
Public Members

int Location
gboolean Active
unsigned char Name[(GSM_MAX_GPRS_AP_NAME_LENGTH+1)*2]
unsigned char URL[(GSM_MAX_GPRS_AP_URL_LENGTH+1)*2]

Values:

• GSM_Date_DDMMYYYY= 1 -
• GSM_Date_MMDDYYYY -
• GSM_Date_YYYYMMDD -
• GSM_Date_DDMMMYY -
• GSM_Date_MMDDYY
• GSM_Date_DDMMYY
• GSM_Date_YYMMDD
• GSM_Date_OFF

```c
struct GSM_Locale
```

Public Members

  char DateSeparator

  GSM_DateFormat DateFormat

  gboolean AMPMTime

```c
struct GSM_Profile_PhoneTableValue
```

Public Members

  GSM_Profile_Feat_ID ID

  GSM_Profile_Feat_Value Value

  unsigned char PhoneID

  unsigned char PhoneValue

### 4.3.22 SMSD

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.

**Return**

Error code

Gets SMSD status via shared memory.

**Parameters**

- **Config**
  
  SMSD configuration pointer.

- **status**
  
  Pointer where status will be copied
Flags SMSD daemon to terminate itself gracefully.

**Parameters**

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

**Return**

Error code

Reads SMSD configuration.

**Parameters**

- **filename**
  File name of configuration.
- **Config**
  Pointer to SMSD configuration data.
- **uselog**
  Whether to log errors to configured log.

**Return**

Error code

Main SMS daemon loop. It connects to phone, scans for messages and sends messages from inbox. Can be interrupted by SMSD_Shutdown.

**Parameters**

- **Config**
  Pointer to SMSD configuration data.
- **exit_on_failure**
  Whether failure should lead to termination of program.
- **max_failures**
  Maximal number of failures after which SMSD will terminate. Use 0 to not terminate on failures.

**See**

*SMSD_Shutdown*

**Return**

Error code

Creates new SMSD configuration.
Parameters

• name -
  Name of process, will be used for logging. If NULL, gammu-smsd text is used.

Return

Pointer to SMSD configuration data block.

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 (1 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.

SMSD configuration data, these are not expected to be manipulated directly by application.
4.3.23 State machine

GSM_Error GSM_InitConnection_Log(GSM_StateMachine * s, int ReplyNum, GSM_Log_Function log_function, void *user_data)

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

Initiates connection with custom logging callback.

GSM_Error GSM_InitConnection(GSM_StateMachine * s, int ReplyNum)

Parameters

- s – State machine data
- ReplyNum – Number of replies to await (usually 1). Higher value makes sense only on unreliable links.

Returns Error code

Initiates connection.

GSM_Error GSM_TerminateConnection(GSM_StateMachine * s)

Parameters

- s – State machine data

Returns Error code

Terminates connection.

GSM_StateMachine

Private structure holding information about phone connection. Should be allocated by GSM_AllocStateMachine() and freed by GSM_FreeStateMachine().

GSM_ConnectionType

Connection types definitions.

GSM_Config

Configuration of state machine.

char Model[50]

Model from config file.

GSM_Log_Function

Parameters

- text – Text to be printed, n will be also sent (as a separate message).
- data – Arbitrary logger data, as passed to GSM_InitConnection_Log().

Returns void

Type of callback function for logging.
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

**Return**

Number of read bytes

Dectes whether state machine is connected.

**Parameters**

- `s` - State machine data

**Return**

Whether phone is connected.

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 directory.

**Parameters**

- `result` - Ini file representation
- `force_config` - Forcing of custom path instead of autodetected one (if NULL, autodetection is performed).

**Return**

Error code

Processes gammu configuration.

**Parameters**

- `cfg_info` - Ini file representation.
- `cfg` - Where to store configuration.
- `num` - Number of section to read.
Return

Whether we got valid configuration. Especially check for ERR_USING_DEFAULTS.

See

GSM_FallbackConfig

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

• \texttt{s} - State machine data
• \texttt{num} - Number of section to read, -1 for currently used.

Return

Pointer to configuration.

Gets number of active gammu configurations.

Parameters

• \texttt{s} - State machine data

Return

Number of sections.

Gets number of active gammu configurations.

Parameters

• \texttt{s} - State machine data
• \texttt{sections} - Number of sections.

Allocates new clean state machine structure. You should free it then by \texttt{GSM_FreeStateMachine}.

Return

Pointer to state machine structure.

Frees state machine structure allocated by \texttt{GSM_AllocStateMachine}.

Parameters
• \texttt{s} -
  Pointer to state machine structure.

Gets number of active gammu configurations.

\textit{Parameters}

• \texttt{s} -
  State machine data

\textit{Return}

Connection type.

\textit{struct GSM\_Config}

Configuration of state machine.

\textit{Public Members}

\texttt{char Model[50]}
  Model from config file

\texttt{char DebugLevel[50]}
  Debug level

\texttt{char * Device}
  Device name from config file

\texttt{char * Connection}
  Connection type as string

\texttt{gboolean SyncTime}
  Synchronize time on startup?

\texttt{gboolean LockDevice}
  Lock device ? (Unix)

\texttt{char * DebugFile}
  Name of debug file

\texttt{gboolean StartInfo}
  Display something during start ?

\texttt{gboolean UseGlobalDebugFile}
  Should we use global debug file?

\texttt{char TextReminder[32]}
  Text for reminder calendar entry category in local language

\texttt{char TextMeeting[32]}
  Text for meeting calendar entry category in local language

\texttt{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.

4.3.24 Types

gboolean definition, compatible with glib.

4.3.25 Unicode

Returns length of unicode string.

Converts string to locale charset.

Returns
    Pointer to static string.

Converts string to console charset.

Returns
    Pointer to static string.

Converts string from unicode to local charset.

Encodes string from local charset to unicode.

Decodes unicode file data with byte order mark (BOM).

Copies unicode string.

Encodes string to UTF-8 quoted printable.

Decodes UTF-8 quoted printable string.

Encodes string to UTF-8.
Decodes string from UTF-8.

Decodes string from hex quoted unicode.

Encodes string to hex quoted unicode.

Compares two unicode strings.

Locates unicode substring.

Compares two unicode strings case insensitive.

Encode text to UTF-8.

Decode text from UTF-8.

Decode hex encoded binary text.

Converts single character from unicode to wchar_t.

Converts single character from wchar_t to unicode.

4.3.26 WAP

Encodes URL to VBKM file.

Parameters

- Buffer - Storage for text.
- Length - Pointer to storage, will be updated.
- bookmark - Bookmark to encode.

Return

- Error code.
Reads WAP bookmark.

**Parameters**
- `s` - State machine pointer.
- `bookmark` - Bookmark storage, need to contain location.

**Return**
- Error code

Sets WAP bookmark.

**Parameters**
- `s` - State machine pointer.
- `bookmark` - Bookmark data.

**Return**
- Error code

Deletes WAP bookmark.

**Parameters**
- `s` - State machine pointer.
- `bookmark` - Bookmark data, need to contain location.

**Return**
- Error code

Acquires WAP settings.

**Parameters**
- `s` - State machine pointer.
- `settings` - Settings storage.

**Return**
- Error code
Changes WAP settings.

Parameters

- \textit{s} - State machine pointer.
- \textit{settings} - Settings data.

Return

Error code

\textit{struct GSM\_WAPBookmark}

WAP bookmark data.

Public Members

- \textbf{int Location} - Location where it is stored.
- \textbf{unsigned char Address[(255+1)*2]} - Bookmark URL.
- \textbf{unsigned char Title[(50+1)*2]} - Bookmark title.

Connection speed configuration.

Values:

- \textit{WAPSETTINGS\_SPEED\_9600} -
- \textit{WAPSETTINGS\_SPEED\_14400} -
- \textit{WAPSETTINGS\_SPEED\_AUTO} -

Connection bearer configuration.

Values:

- \textit{WAPSETTINGS\_BEARER\_SMS= 1} -
- \textit{WAPSETTINGS\_BEARER\_DATA} -
- \textit{WAPSETTINGS\_BEARER\_USSD} -
- \textit{WAPSETTINGS\_BEARER\_GPRS} -

\textit{struct GSM\_WAPSettings}

WAP setting.

Public Members

- \textbf{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 connection 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.

char Password[(50+1)*2]
User password for data or gprs.

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.

4.4 Porting from libGammu older than 1.12.0

4.4.1 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 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.

4.4.2 Changes you have to do in your code

Bellow 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), you can also view it online at <http://wammu.eu/docs/devel/api/>.
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.

5.1 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. There are 2 types of answer:

5.1.1 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.) function. If don’t need anything more than frame type, 0x03,0x05 should be 0x00, 0x00 - it means then, that we check only frame type.

5.1.2 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.

5.1.3 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 inter-

5.2 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
GSM_GetModel

Phone.Functions.GetModel

GSM_WaitFor

Protocol.Functions.WriteMessage

GSM_WaitForOnce

Retries

GSM_ReadDevice

Wait for complete request

Device.Functions.ReadDevice

Protocol.Functions.StateMachine

Phone.Functions.DispatchMessage

GSM_DispatchMessage

Phone.Functions.ReplyFunctions[]}()
CHAPTER SIX

FILE FORMATS USED BY GAMMU

Gammu understands wide range of standard formats as well as introduces own formats for storing some data.

6.1 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.

6.1.1 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

6.2 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.

6.2.1 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.

### 6.2.2 Example

The backup of message can look like following:

```ini
[SMSSave000]
#ABCDEFGHIJKLMNOPQRSTUVWXYZ#
SMSC = "+4540590000"
SMSCUnicode = 002B00340035003400300035003900300030003000300300
Sent = 20021201T025023
State = UnRead
Number = "+4522706947"
NumberUnicode = 002B003400350032003700300036003900340037
Name = ""
NameUnicode =
Text00 = 004100420043004400450046004700480049004A004B004C004D004E004F0050005100520053005400550056005700580059005A000A
Coding = Default
Folder = 1
Length = 27
Class = -1
ReplySMSC = False
RejectDuplicates = True
ReplaceMessage = 0
MessageReference = 0
```

### 6.3 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](#).

#### 6.3.1 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 `fbrestoreF` (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\([23670]\)\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\([2367670]\)\n
Replace:

Type = NumberMobile\nEntry\1Text = "\+362\3"

See Also:

Converting file formats
CHAPTER
SEVEN

GAMMU CONFIGURATION FILE

7.1 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

7.2 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 <http://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.

7.2.1 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** Phoneet 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

For Bluetooth connection use one of following:

- **bluephonet** Phoneet 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

**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.

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 follofing file (also available in sources as `contrib/udev/45-nokiadku2.rules`) as `/etc/udev/rules.d/45-nokiadku2.rules`:
In case your USB device appears as the serial port in the system (e.g. /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 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):

```bash
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 (e.g. /dev/ircomm0) and use it same way as cable. This can be usually achieved by loading modules `ircomm-tty` and `irtty-sir`:

```bash
modprobe ircomm-tty
modprobe irtty-sir
```

**Port**

Deprecated since version 1.27.95. 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 (e.g. you need to be member of uucp group).
This option has no meaning on Windows.

### 7.2.2 Connection options

**SynchronizeTime**
If you want to set time from computer to phone during starting connection.

**StartInfo**
This option allow 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.

### 7.2.3 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.

### 7.2.4 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.
7.3 Examples

There is more complete example available in Gammu documentation.

7.3.1 Connection examples

Gammu configuration for Nokia phone using DLR-3 cable:

```
[gammu]
device = /dev/ttyACM0
collection = dlr3
```

Gammu configuration for Sony-Ericsson phone (or any other AT compatible phone) connected using USB cable:

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

Gammu configuration for Sony-Ericsson (or any other AT compatible phone) connected using bluetooth:

```
[gammu]
device = B0:0B:00:00:FA:CE
collection = 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
collection = blueat
```

7.3.2 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
collection = at
```

```
[gammumu1]
device = /dev/ttyS1
collection = at
```

```
[gammumu2]
device = /dev/ttyS2
collection = at
```

7.3.3 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
;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/SLR3P
; Connection "fbusdlr3"/"dlr3", device type serial
; New Nokia protocol for DUK2 (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 DUK5 (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
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
AT commands for infrared. Used with Nokia, Alcatel, Siemens, etc.
Connection "at19200"/"at115200"/.., device type serial
Connection "irda", device type irda
AT commands for infrared. Used with Nokia, Alcatel, Siemens, etc.
Connection "fbusirda"/"infrared", device type serial
Connection "irdaphonet"/"irda", device type irda
AT commands for infrared. Used with Nokia, Alcatel, Siemens, etc.
Connection "irdaat", device type irda
AT commands for infrared. Used with Nokia, Alcatel, Siemens, etc.
Connection "fbusirda"/"infrared", device type serial
Connection "irdaphonet"/"irda", device type irda
AT commands for infrared. Used with Nokia, Alcatel, Siemens, etc.
Connection "fbusirda"/"infrared", device type serial
Connection "irdaphonet"/"irda", device type irda
AT commands for Bluetooth stack with Nokia 6210
Connection "fbusblue", device type serial
Connection "bluerfbus", device type BT
Nokia protocol with serial device set in BT stack (WidComm, other) from 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 "bluerfbus", 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 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
AT commands for Bluetooth stack with other phones (Siemens, other Nokia, etc.)
Connection "bluephonet", device type BT
Connection "bluerfat", device type BT
Connection "bluefobex", device type BT
OBEX for Bluetooth stack with other phones (Siemens, other Nokia, etc.)
Connection "blueobex", device type BT
Connection "bluerfphonet", 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

<table>
<thead>
<tr>
<th>Port type</th>
<th>&quot;Port&quot; parameter in Windows/DOS</th>
<th>&quot;Port&quot; parameter in Linux/Unix</th>
</tr>
</thead>
<tbody>
<tr>
<td>serial</td>
<td>&quot;com*:&quot;</td>
<td>&quot;/dev/ttyS*&quot;</td>
</tr>
<tr>
<td></td>
<td>(example &quot;com1:&quot;</td>
<td>(example &quot;/dev/ttyS1&quot;)</td>
</tr>
<tr>
<td></td>
<td>or &quot;/dev/ttyS/**&quot; (with DevFS)</td>
<td></td>
</tr>
</tbody>
</table>
; virtual serial ports like /dev/ircomm* or /dev/rfcomm*

; | ignored (can be empty) | ignored (can be empty)

; 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.

; ignored (can be empty) | /dev/ttyUSB* or /dev/ttyACM*

; connection with it not possible | /dev/ttyUSB*

; Step 3. 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", "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 allow 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).
; 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
8.1 Synopsis

`gammu [parameters] <command> [options]`

Commands actually indicate which operation should Gammu perform. They can be specified with or without leading `--`.

8.2 Description

This program is a tool for mobile phones. Many vendors and phones are supported, for actual listing see Gammu Phones Database.

8.2.1 Options

Parameters before command configure gammu behaviour:

- `-c, -config <filename>`
  name of configuration file

- `-s, -section <config>`
  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

8.2.2 Call commands

- `answercall [id]`
  Answer incoming call.

- `cancelcall [id]`
  Cancel incoming call

- `canceldivers`
  Cancel all existing call divers.
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.

8.2.3 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**

**addsmssfolder** name

**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.

**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** ...
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
Similiarly 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

Saves SMS to phone, see bellow 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

- **-maxnum** number
  Limit maximal number of messages which will be created. If there are more messages, Gammu will terminate with failure.

- **-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
  maximal number of SMS messages to create

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. |

**EMS** [-unicode] [-16bit] [-format lcrasbiut] [-text text] [-unicodefiletext file] ...
  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 maximal 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** 1c rasbiut
  last text will be formatted. You can use combinations of chars:

<table>
<thead>
<tr>
<th>Character</th>
<th>Formatting</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>left aligned</td>
</tr>
<tr>
<td>c</td>
<td>centered</td>
</tr>
<tr>
<td>r</td>
<td>right aligned</td>
</tr>
<tr>
<td>a</td>
<td>large font</td>
</tr>
<tr>
<td>s</td>
<td>small font</td>
</tr>
<tr>
<td>b</td>
<td>bold font</td>
</tr>
<tr>
<td>i</td>
<td>italic font</td>
</tr>
<tr>
<td>u</td>
<td>underlined font</td>
</tr>
<tr>
<td>t</td>
<td>strikethrough font</td>
</tr>
</tbody>
</table>
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, ngl, 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 to edit it correctly later in phone composer (for example, in 33xx)

[-toneSElong file] [-variablebitmap file] [-variablebitmaplong file] [-animation frames file1 ...
Saves a SMS template (for Alcatel phones).

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**
  when you want, you can make file in such format: src_unicode_char1, dest_unicode_char1, src_unicode_char2, dest_unicode_char2 (everything in one line). After reading text for SMS from stdin there will be made translation and each src char will be converted to dest char. In docs there is example file (replace.txt), which will change all “a” chars to “I"
**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]
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).

**setmsc** 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.

### 8.2.4 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.
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.

8.2.5 Filesystem commands

Gammu allows 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] [-system] [-hidden] [-newtime]
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]

8.2.6 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 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 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.

8.2.7 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.

### 8.2.8 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.

### 8.2.9 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.

### 8.2.10 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.
8.2.11 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.

8.2.12 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

8.2.13 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 backu 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.

backupsms file [-yes|-all]
Stores all SMSes from phone to file into SMS Backup Format.
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 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 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 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`

### 8.2.14 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.

**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

nokiagetscreendump

nokiagettt9
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 rebooting 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...

nokiasetvibralue level
Set vibra power to “level” (given in percent)
nokiatuneradio
nokiavibratest

8.2.15 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

8.2.16 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
8.2.17 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.

8.2.18 MMS and MMS settings commands

**getallmms** [-save]

**geteachmms** [-save]

**getmmsfolders**

**getmmssettings** start [stop]

**readmmsfile** file [-save]

8.2.19 FM radio commands

**getfmstation** start [stop]
Show info about FM stations in phone

8.2.20 Phone information commands

**battery**
Displays information about battery and power source.

**getdisplaystatus**

**getlocation**
Gets network information from phone (same as networkinfo) and prints location (latitude and longitude) based on information from OpenCellID.

**getsecuritystatus**
Show, if phone wait for security code (like PIN, PUK, etc.) or not

**identify**
Show the most important phone data.

**monitor** [times]
Get phone status and writes continuously to standard output. Press Ctrl+C to break this state.
8.2.21 Phone settings commands

**getcalendarssettings**
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
  - clears netmon and PPS (all “hidden” menus)

- **FACTORY** Reset to factory defaults.
  - changes like after UIF
  - clear date/time

8.2.22 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 to decode sniffs. See Discovering protocol for more details.

8.2.23 Other commands

**entersecuritycode** PIN|PUK|PIN2|PUK2|PHONE|NETWORK code|- [newpin|-]
Allow 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<>[]hHcCjJfFoOoMmDd@
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.

8.2.24 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).

8.2.25 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.

8.2.26 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.

8.3 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 <http://wammu.eu/support/bugs/> for information how to report it.
116 Unknown response from phone. See <http://wammu.eu/support/bugs/> for information how to report it.
117 Unknown frame. See <http://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.
Entry is empty.
Security error. Maybe no PIN?
Invalid location. Maybe too high?
Functionality not implemented. You are welcome to help authors with it.
Memory full.
Unknown error.
Can not open specified file.
More memory required...
Operation not allowed by phone.
No SMSC number given. Provide it manually or use the one configured in phone.
You’re inside phone menu (maybe editing?). Leave it and try again.
Phone is not connected.
Function is currently being implemented. If you want to help, please contact authors.
Phone is disabled and connected to charger.
File format not supported by Gammu.
Nobody is perfect, some bug appeared in protocol implementation. Please contact authors.
Transfer was canceled by phone, maybe you pressed cancel on phone.
Phone module need to send another answer frame.
Current connection type doesn’t support called function.
CRC error.
Invalid date or time specified.
Phone memory error, maybe it is read only.
Invalid data given to phone.
File with specified name already exists.
File with specified name doesn’t exist.
You have to give folder name and not file name.
You have to give file name and not folder name.
Can not access SIM card.
Wrong GNAPPLET version in phone. Use version from currently used Gammu.
Only part of folder has been listed.
Folder must be empty.
Data were converted.
Gammu is not configured.
Wrong folder used.
Internal phone error.
Error writing file to disk.

8.3. Return values
No such section exists.
Using default values.
Corrupted data returned by phone.
Bad feature string in configuration.
Desired functionality has been disabled on compile time.
Bluetooth configuration requires channel option.
Service is not running.
Service configuration is missing.
Command rejected because device was busy. Wait and restart.
Could not connect to the server.
Could not resolve the host name.
Failed to get SMSC number from phone.

8.4 Examples

8.4.1 Configuration

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

8.4.2 Sending messages

Save text message up to standard 160 chars:

```
echo "All your base are belong to us" | gammu savesms TEXT
```

or

```
gammu savesms TEXT -text "All your base are belong to us"
```

Save long text message:

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

or

```
gammu savesms TEXT -len 400 -text "All your base are belong to us"
```

or

```
gammu savesms EMS -text "All your base are belong to us"
```

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

```
gammu savesms EMS -text "Greetings" -defsound 1 -text "from Gammu -tone10 axelf.txt -animation 2 file1.bmp file2.bmp
```

Save protected message with ringtone:
8.4.3 Uploading files to Nokia

Add Alien to applications in your phone (you need to have files Alien.JAD and Alien.JAR in current directory):

```bash
gammu nokiaaddfile APPLICATION Alien
```

Add file.mid to ringtones folder:

```bash
gammu nokiaaddfile TONES file.mid
```

8.4.4 Setting operator logo

Set logo for network 230 03 (Vodafone CZ):

```bash
gammu setbitmap OPERATOR ala.bmp "230 03"
```

8.4.5 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:

```bash
gammu savefile CALENDAR output.vcs myCalendar.backup 260
```

Convert first phonebook entry from Backup Format to vCard:

```bash
gammu savefile VCARD21 output.vcf phone.backup ME 1
```

Convert all contacts from backup to vCard:

```bash
gammu convertbackup phone.backup output.vcf
```

8.4.6 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:

```ini
[gammu]
YOUR CONNECTION SETTINGS
logfile = /tmp/gammu.log
logformat = textall
```

Alternatively you can specify logging on command line:

```bash
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 <http://wammu.eu/support/bugs/> for more information on reporting bugs.

Please report bugs to Gammu bug tracker.
9.1 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.
9.1.1 Overall schema

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

![Diagram of SMS Daemon interactions]

9.1.2 SMSD operation

The SMSD operation consist of several steps.

1. Process command line options.
2. Configure backend service.
3. **Main loop is executed until it is signalled to be terminated.**
   (a) Try to connect to phone if not connected.
   (b) Check for security code if configured (configured by `CheckSecurity`).
   (c) Check for received messages (frequency configured by `ReceiveFrequency`).
   (d) Check for reset of the phone if configured (frequency configured by `ResetFrequency`).
   (e) Check for messages to send (frequency configured by `CommTimeout`).
   (f) Check phone status (frequency configured by `StatusFrequency`).
4. Backend service is freed.

9.2 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.

9.2.1 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.

9.2.2 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.GetStatus()` (from Python).

9.2.3 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`.

9.2.4 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()`.

9.2.5 Reporting Bugs


Before reporting a bug, please enable verbose logging in SMSD configuration:

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

and include this verbose log within bug report.
9.3 Program Manuals

9.3.1 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 chosed 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,  \texttt{-max-failures=count}
Terminate after defined number of failures. Use 0 to not terminate (this is default).

-X,  \texttt{-suicide=seconds}
Kills itself after number of seconds.

-S,  \texttt{-run-service}
Runs program as SMSD Windows service. This should not be used manually, but only Windows Service manager
should use this command.

-n,  \texttt{-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”.

Examples

\textbf{Linux/Unix Examples}

Start SMSD as a daemon on Linux:
\begin{verbatim}
gammu-smsd --config /etc/gammu-smsdrc --pid /var/run/gammu-smsd.pid --daemon
\end{verbatim}

Start SMSD as a daemon on Linux with reduced privileges:
\begin{verbatim}
gammu-smsd --config /etc/gammu-smsdrc --pid /var/run/gammu-smsd.pid --daemon --user gammu --group gammu
\end{verbatim}

\textbf{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.

\textbf{Windows Service Examples}

Install Gammu SMSD Windows service:
\begin{verbatim}
gammu-smsd.exe -c c:\Gammu\smsdrc -i
\end{verbatim}

Install two instances of SMSD Windows service:
\begin{verbatim}
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
\end{verbatim}

To uninstall a Windows service:
\begin{verbatim}
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.

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.

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.

9.3.2 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.

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 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

9.3.3 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.

-l, --loops=count
   Number of loops, by default monitor loops infinitely.

-d, --delay=seconds
   Delay between polling SMSD state, default is 20 seconds.
Gammu Manual, Release 1.28.96

- `C`, `-csv`
  Print output in comma separated values format:

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

## 9.4 SMSD Configuration File

### 9.4.1 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]`.

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](#)

### 9.4.2 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 New in version 1.28.93.
- **MYSQL** synonym for `Service = SQL and Driver = native_mysql`
  stores messages in MySQL database, see [MySQL Backend](#) for details Deprecated since version 1.28.93.
**PQSQL** synonym for `Service = SQL` and `Driver = native_mysql`
stores messages in PostgreSQL database, see *PostgreSQL Backend* for details. Deprecated since version 1.28.93.

**DBI** synonym for `Service = SQL` and `Driver = DBI driver`
stores messages in any database supported by libdbi, this includes MSSQL, MySQL, PostgreSQL or SQLite databases, see *DBI Backend* for details. Deprecated since version 1.28.93.

**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.

**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.

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.

Default is 1.

**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 0 (not used).
<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>StatusFrequency</strong></td>
<td>The number of seconds between refreshing phone status (battery, signal) stored in shared memory and possibly in service backends. Use 0 to disable. Default is 15.</td>
</tr>
<tr>
<td><strong>LoopSleep</strong></td>
<td>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. Default is 1.</td>
</tr>
<tr>
<td><strong>MultipartTimeout</strong></td>
<td>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).</td>
</tr>
<tr>
<td><strong>CheckSecurity</strong></td>
<td>Whether to check if phone wants to enter PIN. Default is 1 (enabled).</td>
</tr>
<tr>
<td><strong>CheckBattery</strong></td>
<td>Whether to check phone battery state periodically. Default is 1 (enabled).</td>
</tr>
<tr>
<td><strong>CheckSignal</strong></td>
<td>Whether to check signal level periodically. Default is 1 (enabled).</td>
</tr>
<tr>
<td><strong>ResetFrequency</strong></td>
<td>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).</td>
</tr>
<tr>
<td><strong>HardResetFrequency</strong></td>
<td>New in version 1.28.92. The number of seconds between performing a preventive hard reset in order to minimize the cases of hanging phones. Default is 0 (not used).</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DeliveryReport</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>DeliveryReportDelay</strong></td>
<td>Delay in seconds how long is still delivery report considered valid. This depends on brokeness of your network</td>
</tr>
</tbody>
</table>
(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.

When you set PhoneID, all messages (including injected ones) will be marked by this string and it allow more SMS daemons to share single database. This option has actually no effect with *Files backend*.

**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.

**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**

Whether to enable sending of messages.
Default is True.

**Receive**

Whether to enable receiving of messages.
Default is True.

### 9.4.3 Database backends options

All DBI, MYSQL and PGSQL backends (see *MySQL 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`. New in version 1.28.92.

**PC**

Synonym for `Host`. Deprecated since version 1.28.92.

**Database**

Name of database to use. Please note that you should create tables in this database before using gammu-smsd. SQL files for creating needed tables are included in documentation.

**SkipSMSCNumber**

When you send sms from some SMS centere 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.

Can be either one of native drivers (`native_mysql` or `native_pgsql`) or *DBI Backend* driver.

Depends on what DBI drivers you have installed, DBI supports: *mysql*, *freetds* (provides access to MS SQL Server and Sybase), *pgsql*, *sqlite*, *sqlite3*, *firebird* and *ingres*, *msql* and *oracle* drivers are under development.

**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.

OutboxFormat
The format in which messages created by gammu-smkd-inject will be stored, it accepts same values as Inbox-
Format.
Default is detail if Gammu is compiled in with backup functions, unicode otherwise.

TransmitFormat
The format for transmitting the SMS: auto, unicode, 7bit.
Default is auto.

9.4.4 Message filtering

SMSD allows to process only limited subset of incoming messages. You can define filters for sender num-
ber 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 direc-
tives.
9.4.5 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.

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/
```

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

```
[smsd]
Service = mysql
PIN = 1234
LogFile = syslog
User = smsd
Password = smsd
PC = localhost
Database = smsd
```

Process only messages from 123456 number:

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

Do not process messages from evil number 666:

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

Enabling debugging:

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

9.5 RunOnReceive Directive

9.5.1 Description

Gammu SMSD can be configured by RunOnReceive directive (see SMSD Configuration File for details) to run defined 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 SMGD waits for the script to terminate. If you make some time consuming there, it will make SMGD not receive new messages. However to limit breakage from this situation, the waiting time is limited to two minutes. After this time SMGD will continue in normal operation and might execute your script again.

9.5.2 Environment

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.

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.

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.

**DECODED_1_MMS_SIZE**
Size of MMS as specified in MMS indication message.
9.5.3 Examples

Activating RunOnReceive

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

```
[smisd]
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
```

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/python
import os
import sys
```
9.6 Backend services

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

9.6.1 Files backend

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 NN20021130_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 SMFSD is running from init script. If this is your case, please add locales definition to init script.

9.6.2 SQL Service

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
• drivers supported by DBI for DBI Backend, which include:
  – sqlite3 - for SQLite 3
  – mysql - for MySQL
  – pgsq1 - for PostgreSQL
  – 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
• Database - database name
• **Driver** - `native_mysql`, `native_pgsql` or **DBI one**

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.

**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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%I</td>
<td>IMEI of phone</td>
</tr>
<tr>
<td>%P</td>
<td>PHONE ID (hostname)</td>
</tr>
<tr>
<td>%N</td>
<td>client name (eg. Gammu 1.12.3)</td>
</tr>
</tbody>
</table>

### SMS Specific Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%R</td>
<td>remote number (^1)</td>
</tr>
<tr>
<td>%C</td>
<td>delivery datetime</td>
</tr>
<tr>
<td>%e</td>
<td>delivery status on receiving or status error on sending</td>
</tr>
<tr>
<td>%t</td>
<td>message reference</td>
</tr>
<tr>
<td>%d</td>
<td>receiving datetime for received sms</td>
</tr>
<tr>
<td>%E</td>
<td>encoded text of SMS</td>
</tr>
<tr>
<td>%c</td>
<td>SMS coding (ie 8bit or UnicodeNoCompression)</td>
</tr>
<tr>
<td>%F</td>
<td>sms centre number</td>
</tr>
<tr>
<td>%u</td>
<td>UDH header</td>
</tr>
<tr>
<td>%x</td>
<td>class</td>
</tr>
<tr>
<td>%T</td>
<td>decoded SMS text</td>
</tr>
<tr>
<td>%A</td>
<td>CreatorID of SMS (sending sms)</td>
</tr>
<tr>
<td>%V</td>
<td>relative validity</td>
</tr>
</tbody>
</table>

\(^1\)Sender number for received messages (insert to inbox or delivery notifications), destination otherwise.
Configurable queries

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

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

**delete_phone**

Deletes phone from database.

Default value:

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

**insert_phone**

Inserts phone to database.

Default value:

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

Query specific parameters:

- `%1` enable send (yes or no) - configuration option Send
- `%2` enable receive (yes or no) - configuration option Receive

**save_inbox_sms_select**

Select message for update delivery status.

Default value:

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

**save_inbox_sms_update_delivered**

Update message delivery status if message was delivered.

Default value:

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

Query specific parameters:

- `%1` delivery status returned by GSM network
- `%2` ID of message

**save_inbox_sms_insert**

Insert received message.

Default value:
**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 locktime SECOND) + 0
WHERE ID = %I AND (SendingTimeOut < NOW() OR SendingTimeOut IS NULL)
```

Query specific parameters:

%I 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
  (SenderID is NULL OR SenderID = '' OR SenderID = %P ) ORDER BY InsertIntoDB ASC LIMIT %1
```

Query specific parameters:

%I 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:

%I 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:

%I ID of message
%2 Number of multipart message

**delete_outbox**
Remove messages from outbox after their 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 their 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, TPQR, 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:

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

**refresh_phone_status**
Update phone status (battery, signal).
Default value:

```sql
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

### 9.6.3 MySQL Backend

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*.

**Privileges**

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

```sql
GRANT USAGE ON *.* TO 'smsd'@'localhost' IDENTIFIED BY 'password';
GRANT SELECT, INSERT, UPDATE, DELETE ON 'smsd'.* TO 'smsd'@'localhost';
```

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

**Example**

SQL script for creating tables in MySQL database:
CREATE TABLE `daemons` (\n   `Start` text NOT NULL,  
   `Info` text NOT NULL  
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

-- Dumping data for table `daemons`

CREATE TABLE `gammu` (\n   `Version` integer NOT NULL default '0'  
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

-- Dumping data for table `gammu`

INSERT INTO `gammu` (`Version`) VALUES (12);

CREATE TABLE `inbox` (\n   `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 default '',
'ID' integer unsigned NOT NULL auto_increment,
'RecipientID' text NOT NULL,
'Processed' enum('false','true') NOT NULL default 'false',
PRIMARY KEY 'ID' ('ID')
) ENGINE=MyISAM DEFAULT CHARSET=utf8 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',
  'Text' text,
  'DestinationNumber' varchar(20) NOT NULL default '',
  'Coding' enum('Default_No_Compression','Unicode_No_Compression','8bit','Default_Compression','Unicode_Compression'),
  'UDH' text,
  'Class' integer default '-1',
  'TextDecoded' text NOT NULL default '',
  'ID' integer unsigned NOT NULL auto_increment,
  'MultiPart' enum('false','true') default 'false',
  'RelativeValidity' integer default '-1',
  'SenderID' varchar(255),
  'SendingTimeOut' timestamp NOT NULL default '0000-00-00 00:00:00',
  'DeliveryReport' enum('default','yes','no') default 'default',
  'CreatorID' text NOT NULL,
  PRIMARY KEY 'ID' ('ID')
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

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

--
-- 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'),
  'UDH' text,
  'Class' integer default '-1',
'TextDecoded' text default NULL,
'ID' integer unsigned NOT NULL default '0',
'SequencePosition' integer NOT NULL default '1',
PRIMARY KEY ('ID', 'SequencePosition')
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

--
-- Dumping data for table 'outbox_multipart'
--

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

-- Table structure for table 'pbk'
--

CREATE TABLE 'pbk' (
  'ID' integer NOT NULL auto_increment,
  'GroupID' integer NOT NULL default '-1',
  'Name' text NOT NULL,
  'Number' text NOT NULL,
  PRIMARY KEY ('ID')
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

--
-- Dumping data for table 'pbk'
--

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

-- Table structure for table 'pbk_groups'
--

CREATE TABLE 'pbk_groups' (  
  'Name' text NOT NULL,
  'ID' integer NOT NULL auto_increment,
  PRIMARY KEY ('ID')
) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

--
-- Dumping data for table 'pbk_groups'
--

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

-- 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',
  PRIMARY KEY ('ID'))
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

--
-- Dumping data for table 'phones'
--
'Send' enum('yes', 'no') NOT NULL default 'no',
'Receive' enum('yes', 'no') NOT NULL default 'no',
'IMEI' varchar(35) NOT NULL,
'Client' text NOT NULL,
'Battery' integer NOT NULL DEFAULT 0,
'Signal' integer NOT NULL DEFAULT 0,
'Sent' int NOT NULL DEFAULT 0,
'Received' int NOT NULL DEFAULT 0,
PRIMARY KEY ('IMEI')
} ENGINE=MyISAM DEFAULT CHARSET=utf8;

--
-- Dumping data for table 'phones'
--

-- 1.9. Back-end services

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 default '',
  '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,
  PRIMARY KEY ('ID', 'SequencePosition')
} ENGINE=MyISAM DEFAULT CHARSET=utf8;

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'
--

--
-- Triggers for setting default timestamps
--
DELIMITER //

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

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

9.6.4 PostgreSQL Backend

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.
Example

SQL script for creating tables in PostgreSQL database:

```sql
--
-- 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 LANGUAGE plpgsql;
CREATE OR REPLACE FUNCTION update_timestamp() RETURNS trigger AS $update_timestamp$
BEGIN
  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 pbk_groups_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_multpart_pkey ON outbox_multpart USING btree ("ID");
--CREATE UNIQUE INDEX pbk_groups_pkey ON pbk_groups USING btree ("ID");
--CREATE UNIQUE INDEX sentitems_pkey ON sentitems USING btree ("ID");
--
-- --------------------------------------------------------
```

9.6. Backend services
CREATE TABLE daemons {
    Start text NOT NULL,
    Info text NOT NULL
};

CREATE TABLE gammu {
    Version smallint NOT NULL DEFAULT '0'
};

INSERT INTO gammu (Version) VALUES (12);

CREATE TABLE inbox {
    UpdatedInDB timestamp(0) WITHOUT time zone NOT NULL DEFAULT LOCALTIMESTAMP(0),
    ReceivingDateTime timestamp(0) WITHOUT time zone NOT NULL DEFAULT 'epoch',
    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',
    CHECK (Coding IN
    ('Default_No_Compression','Unicode_No_Compression','8bit','Default_Compression','Unicode_Compression'))
};

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

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

CREATE TABLE outbox (
  UpdatedInDB timestamp(0) WITHOUT time zone NOT NULL DEFAULT LOCALTIMESTAMP(0),
  InsertIntoDB timestamp(0) WITHOUT time zone NOT NULL DEFAULT 'epoch',
  SendingDateTime timestamp NOT NULL DEFAULT 'epoch',
  Text text,
  DestinationNumber varchar(20) NOT NULL DEFAULT '',
  Coding varchar(255) NOT NULL DEFAULT 'Default_No_Compression',
  UDH text,
  Class integer DEFAULT '-1',
  TextDecoded text NOT NULL DEFAULT '',
  ID serial PRIMARY KEY,
  MultiPart boolean NOT NULL DEFAULT 'false',
  RelativeValidity integer DEFAULT '-1',
  SenderID varchar(255),
  SendingTimeOut timestamp(0) WITHOUT time zone NOT NULL DEFAULT 'epoch',
  DeliveryReport varchar(10) DEFAULT 'default',
  CreatorID text NOT NULL,
  CHECK (Coding IN ('Default_No_Compression','Unicode_No_Compression','8bit','Default_Compression','Unicode_Compression'))
);

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

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

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

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

-- --------------------------------------------------------
-- Table structure for table "pbk"
--
CREATE TABLE pbk (  
    ID serial PRIMARY KEY,  
    GroupID integer NOT NULL DEFAULT '-1',  
    Name text NOT NULL,  
    Number text NOT NULL  
) ;

--
-- Dumping data for table "pbk"
--

-- --------------------------------------------------------
-- Table structure for table "pbk_groups"
--
CREATE TABLE pbk_groups (  
    Name text NOT NULL,  
    ID serial PRIMARY KEY  
) ;

--
-- Dumping data for table "pbk_groups"
--

-- --------------------------------------------------------
-- 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 'epoch',
TimeOut timestamp(0) WITHOUT time zone NOT NULL DEFAULT 'epoch',
Send boolean NOT NULL DEFAULT 'no',
Receive boolean NOT NULL DEFAULT 'no',
IMEI varchar(35) PRIMARY KEY NOT NULL,
Client text NOT NULL,
Battery integer NOT NULL DEFAULT 0,
Signal integer NOT NULL DEFAULT 0,
Sent integer NOT NULL DEFAULT 0,
Received integer NOT NULL DEFAULT 0
);

-- Dumping data for table "phones"
--

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

-- Create trigger for table "phones"
--

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

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

-- Table structure for table "sentitems"
--

CREATE TABLE sentitems (  
  UpdatedInDB timestamp(0) WITHOUT time zone NOT NULL DEFAULT LOCALTIMESTAMP(0),  
  InsertIntoDB timestamp(0) WITHOUT time zone NOT NULL DEFAULT 'epoch',  
  SendingDateTime timestamp(0) WITHOUT time zone NOT NULL DEFAULT 'epoch',  
  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',  
  TPNR integer NOT NULL DEFAULT '-1',  
  RelativeValidity integer NOT NULL DEFAULT '-1',  
  CreatorID text NOT NULL,  
  CHECK (Status IN  
    ('SendingOK','SendingOKNoReport','SendingError','DeliveryOK','DeliveryFailed','DeliveryPending','  
    'DeliveryUnknown','Error'))),  
  CHECK (Coding IN  
    ('Default_No_Compression','Unicode_No_Compression','8bit','Default_Compression','Unicode_Compression')  
    PRIMARY KEY (ID, SequencePosition) );
CREATE INDEX sentitems_date ON sentitems(DeliveryDateTime);
CREATE INDEX sentitems_tpmr ON sentitems(TPMR);
CREATE INDEX sentitems_dest ON sentitems(DestinationNumber);
CREATE INDEX sentitems_sender ON sentitems(SenderID);

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

-- --------------------------------------------------------
-- Create trigger for table "sentitems"
--
CREATE TRIGGER update_timestamp BEFORE UPDATE ON sentitems FOR EACH ROW EXECUTE PROCEDURE update_timestamp();

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

9.6.5 DBI Backend

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.

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
- psql - for PostgeSQL
- freetds - for MS SQL Server or Sybase

Example

SQL script for creating tables in SQLite database:

```sql
CREATE TABLE daemons (  
    Start TEXT NOT NULL,  
    Info TEXT NOT NULL  
) ;

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

INSERT INTO gammu (Version) VALUES (12) ;
```
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',
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')),
Text TEXT,
DestinationNumber TEXT NOT NULL DEFAULT '',
Coding TEXT NOT NULL DEFAULT 'Default_No_Compression',
UDH TEXT,
Class INTEGER DEFAULT '-1',
TextDecoded TEXT NOT NULL DEFAULT '',
ID INTEGER PRIMARY KEY AUTOINCREMENT,
MultiPart TEXT NOT NULL DEFAULT 'false',
RelativeValidity INTEGER DEFAULT '-1',
SenderID TEXT,
SendingTimeOut NUMERIC NOT NULL DEFAULT (datetime('now')),
DeliveryReport TEXT DEFAULT 'default',
CreatorID TEXT NOT NULL,
CHECK (Coding IN ('Default_No_Compression','Unicode_No_Compression','8bit','Default_Compression','Unicode_Compression'))
CHECK (DeliveryReport IN ('default','yes','no'))
);

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,

9.6. Backend services 285
CREATE TABLE pbk {
  ID INTEGER PRIMARY KEY AUTOINCREMENT,
  GroupID INTEGER NOT NULL DEFAULT '-1',
  Name TEXT NOT NULL,
  Number TEXT NOT NULL
};

CREATE TABLE pbk_groups {
  Name TEXT NOT NULL,
  ID INTEGER PRIMARY KEY AUTOINCREMENT
};

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,
  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',
  TPR INTEGER NOT NULL DEFAULT '-1',
  RelativeValidity INTEGER NOT NULL DEFAULT '-1',
  ...
CreatorID TEXT NOT NULL,
CHECK (Status IN
('SendingOK','SendingOKNoReport','SendingError','DeliveryOK','DeliveryFailed','DeliveryPending',
'DeliveryUnknown','Error')),
CHECK (Coding IN
('Default_No_Compression','Unicode_No_Compression','8bit','Default_Compression','Unicode_Compression')) ,
PRIMARY KEY (ID, SequencePosition)
);

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

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

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

9.6.6 Null Backend

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 them in RunOnReceive handler.

9.6.7 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 (gamma-smsd-inject does this), each SMS daemon will have separate outbox queue.

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

daemons

Information about running daemons.
gammu

Table holding single value - version of a database schema. See HISTORY 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)
- **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

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
- **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)
- **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

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) sender identification, it has to match PhoneID in SMSD configuration to make SMSD process this message

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).

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

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

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)
TextDecoded (varchar(160))  SMS text in “human readable” form
ID (integer unsigned)  SMS ID
SenderID (text)  which SMXD instance sent this one sequence
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, matches PhoneID

**pbk**

Not used by SMSD currently, included only for application usage.

**pbk_groups**

Not used by SMSD currently, included only for application usage.

**History**

History of schema versions:

- **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.

- **11** all fields for storing message text are no longer limited to 160 chars, but are arbitrary length text fields (1.25.92)

- **10** DeliveryDateTime is now NULL when message is not delivered, added several indexes

- **9** added sent/received counters to phones table

- **8** introduced phones table

- **7** added CreatorID to tables (it holds PhoneID if set)

**Examples**

**Creating tables**

SQL scripts to create all needed tables for most databases are included in Gammu documentation (docs/sql). As well as some PHP scripts interacting with the database.

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'}

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`.

For long text message, the UDH starts with `050003` followed by byte as a message reference (you can put anything 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.).

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

```sql
INSERT INTO outbox (CreatorID, MultiPart, DestinationNumber, UDH, TextDecoded, Coding)
VALUES
```

```sql
INSERT INTO outbox_multipart (SequencePosition, UDH, Class, TextDecoded, ID, Coding)
VALUES
(2, ‘050003D30202’, ‘u xewz qisubevumxyzk ufuylehyzc. Nse xobq dfolizygqysj t bvowsyhyhyemim ovutpapaempye giuwwbi’, <ID_OF_INSERTED_RECORD_IN_OUBOX_TABLE>, ‘Default_No_Compression’)
```

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

9.7.1 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 (\textit{smsd\_files}), various databases (\textit{smsd\_mysql}, \textit{smsd\_pgsql}, \textit{smsd\_dbi}) or backend which does not store anything at all (\textit{smsd\_null}).

Backend interface

Each backend service needs to support several operations, which are exported in \texttt{GSM\_SMSDService} structure:

\begin{verbatim}
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
\end{verbatim}
• **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_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_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 its 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
9.7.2 Message Sending Workflow

9.7.3 Message Receiving Workflow
MISCELLANEOUS UTILITIES

10.1 gammu-detect

New in version 1.28.95.

10.1.1 Synopsis

gammu-detect [OPTIONS]

10.1.2 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.

10.1.3 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 by `gammu -s`.

When invoked as `gammu-detect -d`, also all examined devices are listed as comments in the output.

10.1.4 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
name = Phone on serial port 3
connection = at

[gammu6]
device = 5C:57:C8:BB:BB:BB
name = Nokia E52
connection = bluephonet
10.2 gammu-config

10.2.1 Synopsis

gammu-config [-f|--force] [-c|--config CONFIG]

10.2.2 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.

10.3 jadmaker

10.3.1 Synopsis

jadmaker [-f|--force] [-u|--url URL] <filename.jar>...

10.3.2 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.
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.

11.1.1 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 take 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:
11.1.2 Collecting results

The tests are ran daily on several platforms and you can find the results on dashboard.

You are welcome to join this effort, all you need is to setup job to pull current Gammu sources and execute the test suite every day (the preferred time is 3:00 CET):

```bash
git pull
make -C build-configure Nightly
```

or also with checking for memory leaks:

```bash
git pull
make -C build-configure NightlyMemCheck
```

11.1.3 Testing of SMSSD

SMSSD tests are performed using Dummy Driver and uses file backend and sqlite database by default. For this you need 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 SMSSD can play.
- **PSQL_TESTING**: you need to have setup PostgreSQL server with database where SMSSD can play.

11.1.4 Testing of command line utility

Gammu command line tests are performed using Dummy Driver where required. It covers most of the command line interface, but some parts need to be explicitly enabled:

- **ONLINE_TESTING**: enable testing of features which require internet access

11.1.5 Testing of Python interface

Python module tests are performed using Dummy Driver where required. It does also cover testing of SMSSD interface, which is done using libdbi(sqlite) driver.

11.1.6 Testing of reply functions

The tests directory contains various tests which do inject data into reply functions and check their response.

11.1.7 Testing of data parsing

The tests directory contains various tests which just try to parse various file formats supported by libGammu.

11.1.8 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`)

### 11.2  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.

#### 11.2.1  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.

11.2.2 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.

11.2.3 Examples

To use dummy driver, you need something like following in ~/.gammurc:

```ini
[gammu]
model = dummy
connection = none
device = /path/to/directory/
```

For disabling GetNext* functions within dummy driver, you need something like following in ~/.gammurc:

```ini
[gammu]
model = dummy
connection = none
features = DISABLE_GETNEXT
device = /path/to/directory/
```
CHAPTER TWELVE

PHONE PROTOCOLS

12.1 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.

12.1.1 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.

12.1.2 Serial port

Download Portmon, which allows 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”)

12.1.3 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 :-)

12.2 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> Pawel
Janik <Pavel@Janik.cz> Pawel Kot <pkot@linuxnews.pl> Marcin Wiacek <Marcin@MWiacek.com> Jens Benn-
fors <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.

12.2.1 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 miliseconds before normal message, and for 2.5 miliseconds before acknowledge. You should wait for acknowledge for 200 miliseconds, then retransmit.

### 12.2.2 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];

12.2.3 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

12.2.4 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  \rightarrow one byte
0xXXXXY \rightarrow two bytes (== 0xXX, 0xYY)

where hex: message type
x: s=send (eg. to mobile), r=receive
{ ... }: data after 0x00, 0x01 header
{+... }: raw data (without header)

12.2.5 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

12.2. Nokia protocols
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)
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

12.3 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. Analyzed by Michal Cihar <michal@cihar.com>.

12.3.1 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 ...>

12.3.2 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

12.4 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 }

314 Chapter 12. Phone Protocols
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 { 0x0042,0x05,0x01,0x07,0xa2,0x88,0x81,0x21,0x15,0x63,0xa8,0x00,0x00,0x00,0x07,0xa8,0x00,0x00,0x00,0x01,0x08,0x00,0x00,0x01,0x60 }
 s Sent after issuing { 0x0042,0x05,0x01,0x07,0xa2,0x88,0x81,0x21,0x15,0x63,0xa8,0x00,0x00,0x00,0x07,0xa8,0x00,0x00,0x00,0x01,0x60 }
 s Sent after issuing { 0x0042,0x05,0x01,0x07,0xa2,0x88,0x81,0x21,0x15,0x63,0xa8,0x00,0x00,0x00,0x07,0xa8,0x00,0x00,0x00,0x01,0x60 }
 s Sent after issuing { 0x0042,0x05,0x01,0x07,0xa2,0x88,0x81,0x21,0x15,0x63,0xa8,0x00,0x00,0x00,0x07,0xa8,0x00,0x00,0x00,0x01,0x60 }
 s Sent after issuing { 0x0042,0x05,0x01,0x07,0xa2,0x88,0x81,0x21,0x15,0x63,0xa8,0x00,0x00,0x00,0x07,0xa8,0x00,0x00,0x00,0x01,0x60 }
 s Sent after issuing { 0x0042,0x05,0x01,0x07,0xa2,0x88,0x81,0x21,0x15,0x63,0xa8,0x00,0x00,0x00,0x07,0xa8,0x00,0x00,0x00,0x01,0x60 }
 s Sent after issuing { 0x0042,0x05,0x01,0x07,0xa2,0x88,0x81,0x21,0x15,0x63,0xa8,0x00,0x00,0x00,0x07,0xa8,0x00,0x00,0x00,0x01,0x60 }
 s Sent after issuing { 0x0042,0x05,0x01,0x07,0xa2,0x88,0x81,0x21,0x15,0x63,0xa8,0x00,0x00,0x00,0x07,0xa8,0x00,0x00,0x00,0x01,0x60 }
 s Sent after issuing { 0x0042,0x05,0x01,0x07,0xa2,0x88,0x81,0x21,0x15,0x63,0xa8,0x00,0x00,0x00,0x07,0xa8,0x00,0x00,0x00,0x01,0x60 }
 s Sent after issuing { 0x0042,0x05,0x01,0x07,0xa2,0x88,0x81,0x21,0x15,0x63,0xa8,0x00,0x00,0x00,0x07,0xa8,0x00,0x00,0x00,0x01,0x60 }
 s Sent after issuing { 0x0042,0x05,0x01,0x07,0xa2,0x88,0x81,0x21,0x15,0x63,0xa8,0x00,0x00,0x00,0x07,0xa8,0x00,0x00,0x00,0x01,0x60 }
 s Sent after issuing { 0x0042,0x05,0x01,0x07,0xa2,0x88,0x81,0x21,0x15,0x63,0xa8,0x00,0x00,0x00,0x07,0xa8,0x00,0x00,0x00,0x01,0x60 }
 s Sent after issuing { 0x0042,0x05,0x01,0x07,0xa2,0x88,0x81,0x21,0x15,0x63,0xa8,0x00,0x00,0x00,0x07,0xa8,0x00,0x00,0x00,0x01,0x60 }
 s Sent after issuing { 0x0042,0x05,0x01,0x07,0xa2,0x88,0x81,0x21,0x15,0x63,0xa8,0x00,0x00,0x00,0x07,0xa8,0x00,0x00,0x00,0x01,0x60 }

Note:
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 }

12.4. Nokia 6110
Gammu Manual, Release 1.28.96

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)}}
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: Dialed 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 }
Gammu Manual, Release 1.28.96

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 }

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 }

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 }

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, "FW" }

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

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>keypad notes</td>
<td>0xff=off, 0x00=level 1, 0x01=level 2, 0x02=level 3</td>
</tr>
<tr>
<td>0x01</td>
<td>lights (? only in car profile)</td>
<td>0x00=off, 0x??=on (maybe 0x01)</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>0x02</td>
<td>incoming call alert</td>
<td>1=ringing, 2=beep once, 3=unknown, 4=off, 5=ring once, 6=ascending, 7=caller groups (see feature #0x08)</td>
</tr>
<tr>
<td>0x03</td>
<td>ringing tone ID</td>
<td>level 1 (0x06) - level 5 (0x0a)</td>
</tr>
<tr>
<td>0x04</td>
<td>ringing volume</td>
<td>0=no tone, 1=standard, 2=special, 3=beep once, 4=ascending</td>
</tr>
<tr>
<td>0x05</td>
<td>message alert tone</td>
<td>0=off, 1=on</td>
</tr>
<tr>
<td>0x06</td>
<td>vibration</td>
<td>0=off, 1=on</td>
</tr>
<tr>
<td>0x07</td>
<td>warning and game tones</td>
<td>0x00=off, 0x04=on</td>
</tr>
<tr>
<td>0x08</td>
<td>incoming caller groups</td>
<td>1=family, 2=VIP, 4=friends, 8=contacts, 16=other</td>
</tr>
<tr>
<td>0x09</td>
<td>automatic answer</td>
<td>0x00=off, 0x01=on</td>
</tr>
<tr>
<td>0x16</td>
<td>Anykey answer</td>
<td>0x00=Off, 0x01=On</td>
</tr>
<tr>
<td>0x18</td>
<td>Memory in use</td>
<td>0x00=Phone, 0x01=SIM card</td>
</tr>
<tr>
<td>0x19</td>
<td>Network selection</td>
<td>0x00=Automatic, 0x01=Manual</td>
</tr>
<tr>
<td>0x1a</td>
<td>Automatic redial</td>
<td>0x00=Off, 0x01=On</td>
</tr>
<tr>
<td>0x1b</td>
<td>???</td>
<td>0x00</td>
</tr>
<tr>
<td>0x1c</td>
<td>???</td>
<td>0x00...0x18</td>
</tr>
<tr>
<td>0x1d</td>
<td>Speed dialling</td>
<td>0x00=Off, 0x01=On</td>
</tr>
<tr>
<td>0x1e</td>
<td>Own number sending</td>
<td>0x00=Preset, 0x01=On, 0x02=Off</td>
</tr>
<tr>
<td>0x1f</td>
<td>Cell info display</td>
<td>0x00=English, 0x01=Deutsch, 0x02=Francais, 0x03=Italiano, 0x06=Nederlands, 0x07=Dansk, 0x08=Svenska, 0x09=Suomi, 0x0e=Norsk, 0x10=Automatic</td>
</tr>
<tr>
<td>0x20</td>
<td>Language</td>
<td>0x00=Off, 0x01=On</td>
</tr>
<tr>
<td>0x26</td>
<td>Reply via same centre</td>
<td>0x00=No, 0x01=Yes</td>
</tr>
<tr>
<td>0x27</td>
<td>Delivery reports</td>
<td>0x00=No, 0x01=Yes</td>
</tr>
<tr>
<td>0x28</td>
<td>Hide clock</td>
<td>0x00=Show clock, 0x01=Hide clock</td>
</tr>
<tr>
<td>0x29</td>
<td>Time format</td>
<td>0x00=24-hour, 0x01=12-hour</td>
</tr>
<tr>
<td>0x2a</td>
<td>Selected profile</td>
<td>0x00=General, 0x01..the rest</td>
</tr>
</tbody>
</table>

### Table 12.1: Phone Protocols

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>keypad notes</td>
<td>0xff=off, 0x00=level 1, 0x01=level 2, 0x02=level 3</td>
</tr>
<tr>
<td>0x01</td>
<td>incoming call alert</td>
<td>1=ringing, 2=beep once, 3=unknown, 4=off, 5=ring once, 6=ascending</td>
</tr>
<tr>
<td>0x02</td>
<td>ringing tone ID</td>
<td>level 1 (0x06) - level 5 (0x0a)</td>
</tr>
<tr>
<td>0x03</td>
<td>ringing volume</td>
<td>0=no tone, 1=standard, 2=special, 3=beep once, 4=ascending</td>
</tr>
<tr>
<td>0x04</td>
<td>message alert tone</td>
<td>0=off, 1=on</td>
</tr>
<tr>
<td>0x05</td>
<td>vibration</td>
<td>0x00=off, 0x04=on</td>
</tr>
<tr>
<td>0x06</td>
<td>warning tones</td>
<td>0x00=5 sec, 0x01=20 sec, 0x02=Timeout</td>
</tr>
<tr>
<td>0x07</td>
<td>screen saver</td>
<td>0x00...0x0d = Number of picture images</td>
</tr>
<tr>
<td>0x08</td>
<td>Screen saver -&gt; Timeout</td>
<td>0x00...0x0d = Number of picture images</td>
</tr>
<tr>
<td>0x09</td>
<td>Screen saver -&gt; Screen saver</td>
<td>0x00...0x0d = Number of picture images</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0a:</td>
<td>???:</td>
<td>0x00=???</td>
</tr>
<tr>
<td>0x15:</td>
<td>???: Read only?</td>
<td>0x00=???</td>
</tr>
<tr>
<td>0x16:</td>
<td>???:</td>
<td>0x00=Phone, 0x01=SIM card</td>
</tr>
<tr>
<td>0x17:</td>
<td>Memory in use (Nokia 3330):</td>
<td>0x00=Phone, 0x01=SIM card</td>
</tr>
</tbody>
</table>
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
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
0x29: Startup logo settings: 0x00=Your own uploaded logo, 0x01=Draft HUMAN technology(tm),
0x02=Itineris, 0x03=Your own uploaded logo
0x2a: ???: 0x00=??? 0x01=???
0x2b: ???: 0x00=??? 0x01=???
0x2c: ???: Read only? (N3330)
0x2d: Auto update of date and time: 0x00=Off, 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

12.4. Nokia 6110

319
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 }

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 divtype=0)

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 calls
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, ?? }
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

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 }

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

12.4. Nokia 6110 321
s set date and time  { 0x0060, 1,1,7,yearh,yearl,month,mday,hour,min,0x00 }
1 r date and time set  { 0x0061 }
s get date and time  { 0x0062 }
1 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 }
1 r alarm set  { 0x006c }
s get alarm  { 0x006d }

0x12: Connect to NBS port (61xx only ?)

s Send  { +0x0c, 0x01, UDH header, data }
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" }
1 r Write cal.note report  { 0x0065, return }
where return: 0x01: ok
0x73: failure
0x81: calendar functions busy. Exit Calendar menu and try again
1
s Calendar notes set  { 0x0066... } 1
1 r Calendar note recvd  { 0x0067, 0x01, ?, length, type, yrH,yrL,mon,day,hr,tz,alrm yrH,yrL,mon,day,hr,tz,textlen, "text" }
1 r Cal.note recvd error  { 0x0067, err }
where err: 0x93: not available
(0x01: OK)
other: error

s Delete cal.note  { 0x0068, location }
1 r Del. cal.note report  { 0x0069, err }
where err: 0x01: OK
0x93: cannot delete
0x93: not available
0x81: calendar functions busy. Exit Calendar menu and try again

0x14: SMS func

s Write SMS to SIM  { 0x0004, .... }
1 s Mark SMS as read  { 0x0007, 0x02, location, 0x00, 0x64 }
1 r SMS message frame rcv  { 0x0008, subtype?, ?, num, ?, BCD(smcenter)... } 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.)

s Delete SMS message  { 0x000a, 0x02, location }
1 r Delete OK  { 0x000b }
1 s SMS status request  { 0x0036, 0x64 }
1 r SMS status  { 0x0037, ?, ?, ?, ?, ?, msgnumber, unread }
1 r SMS status error  { 0x0038 }

0x3f: WAP

s Enable WAP frames  { 0x0000 }
1 r Enable WAP frames  { 0x0002, 0x01 }

s ??  { 0x0003 }
1 r ??  { 0x0004 }

s Get WAP bookmark  { 0x0006, 0x00, location }

Chapter 12. Phone Protocols
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 WAP 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}
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, 0x01 - data bearer
    frame:
      auth, call_type, call_speed, ?, IP len, IP (Unicode), data
    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),serv
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}
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,0xff,0xf9,0x76,0x65,0x20,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x26,0x15,0x7d,0x0a,0x00,
0x65,0x82,0x7f,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
Call commands {0x7c, block}
where: command, (values)
  command: 0x01
  values: number(ASCII), 0x00 - makes voice call
  command: 0x02 - answer call
  command: 0x03 - release call

Netmonitor {0x7e, field}
where: field: 00: next
        F0: reset
        F1: off
        F2: field test menus
        F3: developer menus

Open simlock 2 {0x81, locknumber, 0x10, 0x10, 0x10, 0x10}
Note: sets simlock type to factory?
where: locknumber: 1, 2, 4, 8

Open simlock 2 {0x81, 0x01, locknumber}
where: locknumber: 1, 2, 4, 8

Close simlock {0x82, locknumber, 0x00, 0x00, locksinfo(lock1,4,2,3), 0x00}
where: locknumber: 1, 2, 4, 8

Get simlock info {0x8a, 0x00}

Set downloaded OpName {0x8b, MCC1, MCC2, MNC, Name, 0x00}
Set downloaded OpName OK? {0x8b, 0x00, 0x01}
Get downloaded OpName {0x8c, 0x00}
Get downloaded OpName {0x8c, 0x00, 0x01}

Buzzer pitch {0x8f, volume, hzLO, hzHI}
if volume and hz is 0, it’s off

ACD Readings {0x91, parameter(0x02, 0x03, 0x04, 0x05, 0x07)}
ACD Readings {0x91, parameter?, value?}
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

Get MSid {0xb5, 0x01, 0x2f, msid, 0x25}
Get info about phone {0xc8, 0x01}
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 }
r Get DSP Internal SW { 0xc8, 0x09, 0x00, version (1 byte), 0x00 }
s 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, 0x0e }
r Get COBBA { 0xc8, 0x0e, 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 Get 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 { 0xcb, 0x01, product code, 0x00 }
s Get Prod.Ser.Number { 0xcb, 0x02, number, 0x00 }
s Get Prod.Ser.Number { 0xcb, 0x03, number, 0x00 }
s Get (original ?)IMEI { 0xcc, 0x01 }
r Get (original ?)IMEI { 0xcc, 0x01, 0x00, IMEI, 0x00 }
s Get Manufacture Month { 0xcc, 0x02 }
r Get Manufacture Month { 0xcc, 0x02, 0x00, string, 0x00 }
s Get Purchase date { 0xcd, 0x04 }
r Get Purchase date { 0xcd, 0x04, 0x00, string, 0x00 }
s Set "Made" date { 0xcd, 0x02, string, 0x00 }
s Make "all" phone tests { 0xce, 0x1d, 0x00, 0x23, 0x00, 0x00 }
s Make one phone test { 0xce, 0x1d, num1, num2, num3, num4 }
Where num1-num4: 0x00, 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, 0x00, 0x00, 0x00; 0x00, 0x00, 0x00, 0x00; - "Power off"
   No test for "Security data"
 0x00, 0x10, 0x00, 0x00;
Gammu Manual, Release 1.28.96

0x00, 0x20, 0x00, 0x00;
0x00, 0x40, 0x00, 0x00;
0x00, 0x80, 0x00, 0x00;
0x00, 0x00, 0x01, 0x00;

Result of phone tests { 0xcf }
Result of phone tests { 0xcf, number of tests, results of next tests }

LCD Test
{ 0xd3, value }
where value: 0x03, 0x02 - 1'st test
0x03, 0x01 - 2'nd test
0x02, 0x03 - clears screen

ACD Readings (N6150)? { 0xd4, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 }
ACD Readings (N6150)? { 0xd4, 0x02, 0x00, 0x00, 0x02, 0x00, 0x00, 0x01, ?}

Get EEPROM { 0xd4, 0x02, 0x00, 0xa0, locationLo, locationHi, numofbytes }
where: numofbytes - how many bytes to read
Note: Works ONLY in MBUS
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:

Get Picture Image { 0x0001, location }
Get Picture Image when contains sender number
{ 0x0002, location, number(like in SMS), 0x00, len, text, 0x00, width, height }
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, 0x00 }

Set Picture Image { 0x0003, frame...}
where frame: see 0x47/0x0002

Get/Set Picture Image OK { 0x0004 }
Set Picture Image err { 0x0005, error? }
where error=0x74 - wrong location?

0x64:

Phone ID request { 0x0010 }
Phone ID recvd { 0x0011, "NOKIA", "imei", 0, "model", 0, "prod.code", 0, "HW", 0, "firmware", magic bytes x 4... }

Accessory connection { 0x0012, 16x0x00, 'NOKIA&NOKIA accessory', 3x0x00 } (45 bytes)

Acknowledge (FBUS/IRDA) {+type, seq }
Acknowledge (MBUS)...

0xd0:

Power on message seq1 {+04 }
Power on message seq1 {+05 }

0xd1:

Get HW&SW version { 0x0003, 0x00 }

0xd2:

Get HW&SW version { 0x0003 "V " "firmware\n" "firmware date\n" "model\n" "(c) NMP." }

xda: ? (during playing 2 player snake)
0xf0:

Send RLP frame {+0x00, 0xd9, ... }

0xf4: Power on message seq 2

12.4. Nokia 6110 327
12.5 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:
    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 ...}

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, memory type, memory type, 0x00, 0x00, 0x00, 0x00, 0x0c,
        0x01, location, 0x00[7], 0x01, 0x10, 0x00, 0x00, 0x00, 0x00,
        total_low, total_high, used_low, used_high, 0x01, 0x00, 0x00 }

s Read memory
    { 0x07, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x01, 0x02, memory type,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 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
    0x0a: (10) speed dials
    0x10: (5) caller groups

r Read memory
    { 0x08, 0x00, 0x01,
        code, 0x00, 0x00, 0x00, z, xH, xL, yH, yL, 0x00[7], no.of blocks, { block } * }

where if code==0x0f && xH==0x34 - phonebook location not found

12.5. Nokia 6510
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
   0xa 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: 0x0: General,
       0x03: Mobile (office ?),
       0x06: Work,
       0x04: Fax,
       0x02: Home (mobile ?)
   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, ?, ?, code, ?, ?, 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... }

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  { 0x1a, location (2 bytes), entry type, 0x00, year (2 bytes), Month, Day, block: for Meeting:(hour,minute,alarm (two bytes),recurrence (two bytes),len,0x00,string(unicode)}

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=integer, 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),len,0x00,string(unicode)}

for Reminder:(Recurrence (as above),len,0x00,string(unicode)}

for Birthday:(byte1,byte2,alarm(4 bytes),yearofbirth,alarmtime,byte1 and byte2 may vary (???). Usually are 0x00.
In Birthday, the Year in the common part, usually
So, don’t consider it as Year of note, neither your Birthday use the value described below).

where alarm=32-bit integer that is the number of seconds before the alarm time and 11:59:58pm on the birthday.For "No Alarm"=0x0000FFFF (65535).
YearOfBirth=used instead of the one in the common part, 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 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 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, 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, 0x55, 0x55 }
r Get message info { 0x0f, 0x00, 0x01, 0x00, 0x50, memory, type, 0x00, location, FolderID, ...

where: type = 0x00 - MT
0x01 - delivery report
0x02 - MO
0x80 - picture message

where: status=0x01 - received/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

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,
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, 0x00, 'A', 0x00, 0x00, 0x00 }
Gammu Manual, Release 1.28.96

Chapter 12. Phone Protocols
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
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
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|128|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|129|20 |4eN|4dM|50P|2e.100 |00 |00 |00 |00 .(c) NMP......
Sending frame 0x1b / 0x0006

12.5. Nokia 6510 335
0x1f: RINGTONE

s Get Ringtones
r Get Ringtones

0x38:

s ???
where location: 0, 1, 2, 3

r ??? 1d 00 10b 00 01 00, location, 08 00 00 00 00 00 00

s ???

00 02 100 0a 00 60*00 10 10 011 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 20 100 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 3000 3110 3220 3330 34 ...-
00 35 00 36 00 37 00 3880 3990 3a 00 3b 00 3c .5
00 3d=00 3e>00 3f00 4000 41A0 42B0 43C0 44 ...
00 45 00 46 00 47 00 4800 4910 4a20 4b30 4c ...
00 4dM0 4eN0 4fO0 50P0 51Q0 52R0 53S0 54 ...
00 55U0 56V0 57W0 58X0 59Y0 5a00 5b10 5c ...
00 5d 00 5e*00 5f00 60*00 61a0 62b0 63c0 64 ...
00 65e0 66f0 67g0 68h0 69i0 6a00 6b00 6c ...
00 6dm0 6en00 6fo00

12.5. Nokia 6510 337
Sending frame 0x38 / 0x00c7

Received frame 0x38 / 0x0306

Chapter 12. Phone Protocols
Received frame 0x38 / 0x0306
02 |1d |00 |0b |00 |60'|00 |04 |08 |04 |00 |d0¯d|00 |00 |00 |04 .....'.....¯d....
08 |04 |00 |d1Ð|00 |00 |00 |04 |08 |04 |00 |d2ˇD|00 |00 |00 |04 ...Ð.......ˇD....
08 |04 |00 |d3Ë|00 |00 |00 |04 |08 |04 |00 |d4d'|00 |00 |00 |04 ...Ë.......d'....
08 |04 |00 |d5ˇN|00 |00 |00 |04 |08 |04 |00 |d6Í|00 |00 |00 |04 ...ˇN.......Í....
08 |04 |00 |d7Î|00 |00 |00 |04 |08 |04 |00 |d8ˇe|00 |00 |00 |04 ...Î.......ˇe....
08 |04 |00 |d9 |00 |00 |00 |04 |08 |04 |00 |da |00 |00 |00 |db ................
08 |00 |00 |00 |00 |00 |00 |00 |dc |08 |00 |00 |00 |00 |00 |00 |dd ...............¸ T
08 |00 |00 |00 |00 |00 |00 |00 |de˚U|08 |00 |00 |00 |00 |00 |00 |df .......˚U........
08 |00 |00 |00 |00 |00 |00 |00 |e0Ó|08 |00 |00 |00 |00 |00 |00 |e1 .......Ó.......ß
08 |00 |00 |00 |00 |00 |00 |00 |e2Ô|08 |00 |00 |00 |00 |00 |00 |e3 .......Ô.......´N
08 |00 |00 |00 |00 |00 |00 |00 |e4´n|08 |00 |00 |00 |00 |00 |00 |e5 .......´n.......ˇn
08 |00 |00 |00 |00 |00 |00 |00 |e6Š|08 |00 |00 |00 |00 |00 |00 |e7 ...........Š...š
08 |00 |00 |00 |00 |00 |00 |00 |e8´R|08 |00 |00 |00 |00 |00 |00 |04 .......´R........
08 |04 |00 |e9Ú|00 |00 |00 |04 |08 |04 |00 |ea´r|00 |00 |00 |04 ...Ú.......´r....
08 |04 |00 |eb˝U|00 |00 |00 |04 |08 |04 |00 |ecý|00 |00 |00 |04 ...˝U.......ý....
08 |04 |00 |edÝ|00 |00 |00 |04 |08 |04 |00 |ee¸ t|00 |00 |00 |04 ...Ý.......¸ t....
08 |04 |00 |ef´|00 |00 |00 |04 |08 |04 |00 |f0|00 |00 |00 |f5 ...............§
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 |04 ................
08 |04 |00 |f5 |00 |00 |00 |04 |08 |04 |00 |f6 |00 |00 |00 |04 ................
08 |04 |00 |f7 |00 |00 |00 |04 |08 |04 |00 |f8 |00 |00 |00 |04 ................
08 |04 |00 |f9 |00 |00 |00 |04 |08 |04 |00 |fa |00 |00 |00 |04 ................
08 |04 |00 |fb|00 |00 |00 |04 |08 |04 |00 |fc|00 |00 |00 |04 ................
08 |04 |00 |fd|00 |00 |00 |04 |08 |04 |00 |fe|00 |00 |00 |04 ................
08 |04 |00 |ff |00 |00 |00 |04 |08 |04 |00 |01 |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 ................

12.5. Nokia 6510 339
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 ...-............

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,
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 |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, 0x00, 0x00, 0x01, 0x01, 0x18, 0x01, 0x00, IMEI, 0x00,

s Get ??? {+0x00, 0x07, 0x03, 0x01, 0x00, 0x01 }

r Get ??? { 02 |06 |02 |00 |00 |01 |02 |08 |00 |0c |07 |d1 |00 |00 }

s Get ??? {+0x00, 0x07, 0x04, 0x01, 0x00, 0x01 }

r Get ??? { 02 |06 |10 |00 |00 |00 |05 |08 |00 |00 |00 |00 |00 |00 |00 |

s Get ??? {+0x00, 0x07, 0x05, 0x01, 0x00, 0x01 }

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

340 Chapter 12. Phone Protocols
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}  
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." }

12.6 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.tinet.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 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  \{ 0x0004, 0x00, 0x00, 0x02\}
   note: no info about Delivery Reports
   r Incoming SMS info onOK{ 0x000e }  \{ 0x000e \}
   note: no info about Delivery Reports
   r Incoming SMS infoonerr{ 0x000f, error }  \{ 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], (SMSC 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, validity[2], (DefaultRecipient no.)[12], (Mobile Station Center 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, 0x00, 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, 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}
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}
typr: 0x0a: General,
0x03: Mobile (office ?),
0x06: Work,
0x04: Fax,
0x02: Home (mobile ?)

s Set mem location { 0x010b, 0x00, 0x01, 0x01, 0x00, 0x00, 0x00, z,
xH, xL, yH, yL, 0x00, 0x00, 0x00,
r Set mem location  { 0x010c, 0?, 17, code, 0?, 0?, z?, 0?, 0?, yH, yL, xL }  
where code:
  0x3d - wrong entry type
  0x3e - too much 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
  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 calls
  0x19: data calls

s Deactivate calldiverts{ 0x0001, 0x04, 0x00, divtype, calltype, 0x00 }

r Deactivate calldiverts{ 0x0002, 0x04, 0x00, divtype, 0x02, calltype, data }

s Get call diverts  { 0x0001, 0x05, 0x00, divtype, calltype, 0x00 }

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 { 0x0009b, location (2 bytes) }
r Delete calendar note { 0x0009c, location (2 bytes), ?, ?, ?, ? }
s Get calendar note { 0x0009, location (2 bytes) }
r Calendar note recvd { 0x001a, location (2 bytes), entry type - 0x01 - Meeting, 0x02 - Call, 0x04 - Birthday, 0x08 - Reminder
where: entry type - 0x01 - Meeting, 0x02 - Call, 0x04 - Birthday, 0x08 - Reminder
block: for Meeting:{hour,minute,alarm (two bytes),recurrance (two bytes),
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.
Recurrance=in hours, between future occurances 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),Recurrance (as above),
name (unicode),number (unicode))
for Reminder:{Recurrance (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.
In Birthday, the Year in the common part, usually
So, don't consider it as Year of note, neither year of
Birthday use the value described below).
where alarm=32-bit integer that is the number of seconds before
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...} }
? s Get calendar note?? { 0x003E, location (2 bytes) }
? r Get calendar note?? { 0x003F, location (2 bytes), ... }
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(...)
}
r Get SMS failed { 0x0009, 0x02 },
s Get SMS status { 0x0036, 0x64 }
r Get SMS Status { 0x0037, 0x05/0x03, 0x01, 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 mssages.
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(...)
std. size: 72x28
}
r Set Picture Image { 0x0051, location, number[2 bytes], 0x07 }
s Set SMS name { 0x0083, folder, location(2 bytes), name(Unicde), 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 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, .... }
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 (?)

? 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,0x32,0x2b,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 7110
s Get ringtone
   { 0x0122, 0x00, location }

r Get ringtone
   { 0x0123, 0x00, location, name (Unicode), 0x00, ..., 0x00, 0x02, 0x0F, 0x09 (ringtone content) }

r Get ringtone error
   { 0x0124, ... }

0x39:

s get profile feature
   { 0x0101, 0x01, 0x01, number1, number2 }

   where number1: from 0x00 to 0x07 (for each profile ?)
   where number2: 0x00 - 0x09, 0x0A, 0x16 - 0x19, 0x1A - 0x1F, 0x20 - 0x29, 0x2A - 0x2C, 0xFF

   where 0x09: keypad tones
   where 0x02: incoming call alert
   where 0x03: ringtone number
   where 0x04: ringing volume
   where 0x05: message alert tone
   where 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
             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 is obtained by doing (Number - 0x3F);
             0x89 to 0x8D - gives number of uploaded ringtone. The number 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 .. 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}

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 WAP bookmark OK
  { 0x000d }
? r Delete WAP bookmark 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}
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,
  0x01 - data bearer
  frame:
  auth, call_type, call_speed, ?, IP len, IP (Unicode), dialup len,
  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 (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 }
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, lockinfo(lock1, lock2, lock3, lock4), locksclosed: bit1,bit2,bit3,bit4 - if set, selected lock is user lock
counter1 - counter4: counters for locks
s Buzzer pitch { 0x8f, volume, hzLO, hzHI }
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 }
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... }
r Set bin ringtone { 0xa0, location, error } where location=0,1,etc.
error=0x0a, ringtone NOT set
0x00, ringtone set OK
? r Get MSId { 0xbb, 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", 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 DPS 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

12.6. Nokia 7110
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, 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 Purchase date { 0xcc, 0x04 }

r Get Purchase 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}

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;
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, 0x00, 0x0e, 0x01 }

r ACD Readings(N6150)? { 0xd4, 0x02, 0x00, 0x02, 0x00, 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 0x07 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, 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)  
    0x15 (startup logo) : 0x00, 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  

12.7 Nokia 6210/6310, CARC91, PC Experiment  

Author: Jens Bennfors  
Company: AB Indevia  
Date: 2002-04-09
12.7.1 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.

12.7.2 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.

12.7.3 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 0201 0164 00 0504  Enable carkit mode from PC
1F0004 79 0005 0201 0164 00 0607  Enable carkit mode from PC
1F0400 7F 0367  Ack from phone
1F0004 79 0005 0201 0206 0056 0205 005A 0048 4655 3200 044F  HFU-2 Version
1F0400 7F 0460  Ack from phone
1F0004 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**

<table>
<thead>
<tr>
<th>Command String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1F0004 D0 0001 04 00CE</td>
<td>Power up from PC</td>
</tr>
<tr>
<td>1F0400 D0 0001 05 1BD4</td>
<td>Power up from phone</td>
</tr>
<tr>
<td>1F0004 79 0005 0201 0164 0001 00</td>
<td>Enable carkit mode from PC</td>
</tr>
<tr>
<td>1F0400 7F 0165</td>
<td>Ack from phone</td>
</tr>
<tr>
<td>1F0400 78 0004 0102 0E03 1C75</td>
<td>Status 0x0E, 0x03 from phone</td>
</tr>
<tr>
<td>1F0400 78 0004 0102 0701 1D7F</td>
<td>Status 0x07, 0x01 from phone</td>
</tr>
<tr>
<td>1F0004 79 0012 0201 0206 00 5620 3037 2E30 300A 4846 5532 00 0249</td>
<td>HFU-2 Version from PC</td>
</tr>
<tr>
<td>1F0400 7F 0266</td>
<td>Ack from phone</td>
</tr>
<tr>
<td>1F0400 78 0004 0102 0801 1E73</td>
<td>Status 0x08, 0x01 from phone</td>
</tr>
<tr>
<td>1F0004 79 0005 0201 0164 0003 02</td>
<td>Enable carkit mode from PC</td>
</tr>
<tr>
<td>1F0400 7F 0367</td>
<td>Ack from phone</td>
</tr>
<tr>
<td>1F0400 78 0004 0102 0E03 1F76</td>
<td>Status 0x0E, 0x03 from phone</td>
</tr>
<tr>
<td>1F0400 78 0004 0102 0701 2042</td>
<td>Status 0x07, 0x01 from phone</td>
</tr>
<tr>
<td>1F0004 78 0003 0201 03 0464</td>
<td>Status confirm from PC</td>
</tr>
<tr>
<td>1F0400 7F 0460</td>
<td>Ack from phone</td>
</tr>
</tbody>
</table>

**Disconnected**

<table>
<thead>
<tr>
<th>Command String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1F04 0078 0004 0102 0700 2142</td>
<td>Status 0x07, 0x00</td>
</tr>
</tbody>
</table>

**Incoming SMS**

<table>
<thead>
<tr>
<th>Command String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCF0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0 F0F0</td>
<td>Initiation of bit length from phone</td>
</tr>
<tr>
<td>1F0400 78 0004 0102 0E03 254C</td>
<td>Status 0x0E, 0x03 from phone</td>
</tr>
<tr>
<td>F0F0F0F0</td>
<td>Initiation of bit length from phone</td>
</tr>
</tbody>
</table>
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

### 12.7.4 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 00B0 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
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 0xA0, 0x00
1F0400 78 0004 0128 0901 0A4C Status 0x09, 0x00
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
1F0400 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 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 7F 0266  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
Disconnected

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

Incoming SMS

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

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

12.7.5 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.

12.8 TDMA 5120

Eduardo Spremolla at gnokii-users@mail.freesoftware.fsf.org

After playing a while with my 5120i y find some use full frames:

12.8.1 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 )

Cant figure out how to modify 6110 code to get & put the logo, not in a hi value to me now.

12.8.2 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}
12.8.3 get memory

the getmemory:: 40 {+00 00 07 11 00 10 00 mem}

call phonebook with the phone in bcd, but it seems to be a way to read chunks of memory with different numbers in the

get configuration pins:

40 {+0x00, 0x00, 0x07, 0x11, 0x00, 0x0f, 0x00, 0x00 }

call security code:

40 {+0x00, 0x00, 0x07, 0x11, 0x00, 0x09, 0x00, 0x00 }

call 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 pagging channel
02 c4  seconda pagging channel
88 88 88 88  own #
09 63 c2 09 03 0b  unknow
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 00  alphabetic
b3 4d  unknow
01  NAM status
11 11 11 11 11 00 00 00 00 00 00 00 00 00 00  unknow
00 00 00 00 00 01 00 00 00 01 36  unknow
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 00 00 00 00  msg center #
08 01 80 70 8f dd 00 ef 00 00 00 00 00 00 00 00 00 00  unknow
00 00 00 00 00  gate way #
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......

12.9 SAMSUNG Organizer AT commands

12.9.1 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

12.9.2 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:
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, aniversary entry type
par03 Empty
par04 Ocassion name
par05 Alarm day

12.9. SAMSUNG Organizer AT commands
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, miscellaneous 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

12.9.3 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:

12.9. SAMSUNG Organizer AT commands
par1 to par24 has the same significance than in the AT+ORGR command

### 12.9.4 Delete organizer entry

**Invocation:**

```
AT+ORGD=number
```

Delete organizer entry of index “number”

**Example:**

```
AT+ORGD=21
```

OK

### 12.9.5 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.

### 12.10 SAMSUNG GT calendar AT commands

#### 12.10.1 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",2010,5,12,2010,5,12,49,22,49,0,0,0,0,2010,5,30,,
+SSHR:3,"1,x","0","0",2010,6,2,2010,6,3,0,0,0,0,0,0,2010,5,30,,
+SSHR:1,"9,event1234","0","0",2010,6,7,2010,6,7,7,0,8,59,0,0,0,2010,5,30,,
+SSHR:4,"7,test123","0","0",2010,6,14,2010,6,14,21,37,22,37,0,0,0,2010,5,30,,
+SSHR:2,"7,Meeting","0","0",2010,6,15,2010,6,15,8,0,8,59,0,0,0,2010,5,30,,
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,,
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,"",2010,06,07,2010,06,07,00,08,59,0,0,0,0,2010,05,30,,

Please note, that the format for writing is somehow different, than for reading - hour and minuts 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

12.10.2 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+SSH=","9,New task1","11,New details","2010,06,21,2010,06,30,1,2010,06,27,08,07,0,2,0,2
+SSH:2
OK
AT+SSH=2
+SSH: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+SSH=3
OK
AT+SSH=0
+SSH:1,"10,Test event","10,2010-06-05",60823,11,25,60823,11,26,60823,11,26,0,0,0,0,0
+SSH:2,"9,New task1","10,2010-06-30",60823,11,25,60823,11,26,60823,11,26,0,0,0,0,0
OK
AT+SSH=1
OK
AT+SSH=0
+SSH:2,"9,New task1","10,2010-06-30",60823,11,25,60823,11,26,0,0,0,0,0,0,0,0
OK

12.10.3 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+OMM=0,"This is a note"
+OMM:6
OK

You can read it:
AT+OMM=6
+OMM: "This is a note"
OK

You can modify it:
AT+OMM=6,"This is a new modified note"
+OMM:6
OK
AT+OMM=6
+OMM: "This is a new modified note"
OK

To delete entries:
AT+OMM=3
+OMM: "Note number 3"
OK
AT+OMM=3
OK
AT+OMMW=3,"New note number 3"
+CME ERROR:29
ERROR

12.11 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)
          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.

12.12 m-obex protocol used by some Samsung mobiles

This document is copied from <http://code.google.com/p/samsyncro/wiki/mobex> and extended.

12.12.1 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.

12.12.2 Requirements

• Established bluetooth connection to the serial channel of the mobile
• Some way to access this serial port. For example minicom.

12.12.3 Starting the obex server

To start the obex server you have to send this AT command first:

AT+SYNCML=M0BEXSTART

Some phones seem to start with following command:

AT$TSSPCSW=1

12.12.4 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:

<table>
<thead>
<tr>
<th>Package Header</th>
<th>Session Id</th>
<th>Obex Header(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• First byte: Type of request.</td>
<td>• 0xCB and four bytes of session id</td>
<td>• First byte: Type of header.</td>
</tr>
<tr>
<td>• Second and third bytes:</td>
<td></td>
<td>• Second and third bytes: length of header.</td>
</tr>
<tr>
<td>length of package</td>
<td></td>
<td>• Next bytes: data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Last byte: 0x00</td>
</tr>
</tbody>
</table>

For detailed information about obex, for example what types of packages and headers exists, get the official Obex
documentation from Irared 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
### 12.12.5 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: maximal 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 63 6F 75 6E 74 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 74 61 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 4F 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

12.12. m-obex protocol used by some Samsung mobiles
\texttt{4C 00 06 01 00 19} First byte: ? Last two bytes: Id of contact

**Answer**

\texttt{A0 00 08} Obex ok  
\texttt{4C 00 05 00 00} Error code

### 12.12.6 Calendar

#### Get count

**Request**

\texttt{83 00 25} Obex get  
\texttt{CB 00 00 00 00} Session id  
\texttt{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  
\texttt{4C 00 04 FF} ?

**Answer**

\texttt{A0 00 1C} Obex ok  
\texttt{C3 00 00 00 0C} length of data  
\texttt{4C 00 05 00 00} Error code  
\texttt{49 00 0F 01 2C 00 06 00 64 00 00 64 00 00} ?TODO?

#### List all

**Request**

\texttt{83 00 20} Obex get  
\texttt{CB 00 00 00 00} Session id  
\texttt{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**

\texttt{A0 00 C0} Obex ok  
\texttt{C3 00 00 00 B0} Session  
\texttt{4C 00 05 00 00} Error code  
\texttt{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 61 72 2F 63 72 65 61 74 65 00  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 74 65 00  m-obex/calendar/read

12.12.  m-obex protocol used by some Samsung mobiles
4C 00 06 01 00 06  Id of calendar item

**Answer**

A0 00 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 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 64 61 72 2F 64 6C 65 74 65 00  m-obex/calendar/delete

4C 00 06 01 00 06  id of calendar item

**12.12.7 Notes**

**12.12.8 Tasks**

**12.12.9 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

**12.12.10 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.

**Note:** You can also find documentation for some protocols and vendor extensions in separate git repository at [http://gitorious.org/gammu/gsm-docs](http://gitorious.org/gammu/gsm-docs)
TPMR  Message reference as generated by GSM network.
g
  gammu, 22
  gammu.data, 44
  gammu.exception, 46
  gammu.smsd, 43
  gammu.worker, 45