Freedom Fone

Technical Documentation

GSMopen
FreeSWITCH Endpoint

October 2009
Author: Giovanni Maruzzelli
Version 1.0

Source: http://wiki.freeswitch.org/wiki/GSMopen

Table of Contents

Technical Documentation: GSMopen   *October 2009*
Freedom Fone   *Version 1.0*
Author: Giovanni Maruzzelli

1   What is GSMopen

GSMopen is an endpoint (channel driver) that allows SMSs to/from FreeSWITCH and incoming and outgoing GSM voice calls (that can be bridged, originated, answered, etc. as in all other endpoints, e.g. sofia/SIP). SMSs on FreeSWITCH are handled following the CHAT API (like the text messaging in Sofia and Jingle).

GSMopen uses a GSM modem and the server sound subsystem as a physical interface to the GSM network, using native serial commands (AT or FBUS) for call control, and the native sound interface (eg, ALSA, winmme) for audio I/O.

For production deployment and multiline setups an embedded combined device is the suggested choice.

GSMopen has complete customizability of the serial (eg: AT) commands it sends to the device, so it can be used with any GSM device that accepts commands :).

GSMopen works in FreeSWITCH on Linux, native at 8khz (GSM is 8khz compressed audio).

Think of GSMopen as similar to OpenZAP for analog lines. For each channel you need an interface (a GSM modem and a sound I/O). So, for example, two concurrent calls would need two channels, and therefore two GSM modems and sound I/Os connected to your FreeSWITCH server.

Obviously you must have credit on the SIM(s) inside the interface(s) to make and receive calls, just like you need credits for your regular cellphone to work.

GSMopen uses *very* low CPU, so it can works with the less powerful server platforms without problems (eg: embedded appliances).

2   History

GSMopen is the successor of Celliax.

Celliax was born in 2004 as an Asterisk channel driver for using cheap cellphones as GSM interfaces. First kind of serial signaling to be implemented was FBUS (the proprietary protocol of old Nokia cellphones), and it was possible thanks to the open source community efforts to document that undisclosed binary serial protocol (particularly the gammu and gnokii projects). Then was implemented the AT protocol used by most part of non old Nokia cellphones. In parallel was developed the hardware design for the audio cable from the hands free jack to the soundcard (including an hardware filter for the GSM band and the TDM frequency bursts that goes into audio spectrum, microphone bias voltage decoupling, etc). Data cables are available ready made from original cellphone manufacturer and aftermarket. A DECT interface was contributed that allows for interfacing a DECT chipset and system. After a while to Celliax was added the Skype capability (that was later spun off to be chan_skypiax).

Last touch was adding the acoustic echo canceling from the speexdsp library, that allowed for the use of cellphones that generate "comfort echo" (good for you when you use the cellphone in the way was designed to be used, bad if you use it as an interface: it generates echo :)).
Celliax was working well, and was quickly adopted by hardware manufacturers that made combined devices containing all the needed hardware in a black box, with just one USB cable. And without echo.

Starting in 2009 Celliax becomes GSMopen, and the development platform becomes FreeSWITCH.

3   Hardware Requirements

The CPU load generated by the GSMopen endpoint is very low, so if a server is able to run FS, it will have no problems using GSMopen channels.

You will need at least one physical interface to connect to a GSM network.

4   Dialplan, and how to use GSMopen

4.1   Dialplan

Like other endpoints it's easy to build up useful dialplans using GSMopen.

You can use the standard format with the interface name:

```
gsmopen/interface1/3472665618
```

to call the number "3472665618" using the gsmopen interface named "interface1"

Dialplan snippet:

```
  <!-- dial 3472665618 via gsmopen using interface1 interface to go out -->
   <extension name="gsmopen">
     <condition field="destination_number" expression="^2909$">
       <action application="bridge" data="gsmopen/interface1/3472665618"/>
     </condition>
   </extension>
```

4.2   The "ANY" and "RR" interfaces, poor man interface grouping

You can also use the "ANY" or "RR" interfaces

```
gsmopen/ANY/3472665618
```
```
gsmopen/RR/3472665618
```

to call "3472665618" using the first available (idle, not in a call) gsmopen interface, automatically selected (thx Seven Du).

"ANY" and "RR" are now just aliases, and will choose an available idle interface based on a round

robin algorithm (so to distribute calls more fairly between all the available interfaces).

Dialplan snippet:

```
<!-- dial 3472665618 via gsmopen RR interface -->
<extension name="gsmopen">
  <condition field="destination_number" expression="^2908$">
    <action application="bridge" data="gsmopen/RR/3472665618"/>
  </condition>
</extension>
```

4.3 *Multiple concurrent incoming calls to the same GSM number*

This is simply not possible, and never will be.
Each physical interface (eg: GSM modem + sound I/O) has its own SIM, with just one number.
When a call is made by a remote party to a number, the carrier sends the call to the SIM that bears that number.
NO WAY is known to me to have multiple SIMs to answer to the same number, but maybe some carrier can give this as an option.

5 API and CLI Commands

GSMopen adds the "gsmopen_sendsms" command.

5.1 *gsmopen_sendsms*

gsmopen_sendsms interface_name destination_number SMS_text

eg:

gsmopen_sendsms interface1 3472665618 this is a nice SMS text

6 Events

On FS GSMopen generates the standard events on voice calls (like the other endpoints) and CHAT events on SMSs

6.1 *Voice Calls*

Standard CODEC and CHANNEL_* events.

6.2 *SMSs*

The Event type generated is of type MESSAGE (like in Jingle and Sofia).

The interesting fields are:

login: the interface name that received the SMS

from: the sender's number, urlencoded

date: the date stamp from SMS center, urlencoded

datacodingscheme: in which coding type the message was sent

servicecentreaddress: address of SMS center

message type: almost always 0 (delivery message)

during-call: bool, the message was received while a voice call was ongoing?

And obviously the body, encoded in utf8, that contains the SMS's text.

NB: Body is UTF8 encoded, gives you ASCII for ASCII, and UTF8 for all the rest.

This is an incoming SMS as reported with Events xml:

```
Content-Length: 1143
Content-Type: text/event-xml

<event>
 <headers>
  <Event-Name>MESSAGE</Event-Name>
  <Core-UUID>5e6126a2-ce23-11de-a3ff-859d1aa2d302</Core-UUID>
  <FreeSWITCH-Hostname>hardy64</FreeSWITCH-Hostname>
  <FreeSWITCH-IPv4>192.168.0.12</FreeSWITCH-IPv4>
  <FreeSWITCH-IPv6>%3A%3A1</FreeSWITCH-IPv6>
  <Event-Date-Local>2009-11-11%2020%3A59%3A30</Event-Date-Local>
  <Event-Date-GMT>Wed,%2011%20Nov%202009%2019%3A59%3A30%20GMT</Event-Date-GMT>
  <Event-Date-Timestamp>1257969570473521</Event-Date-Timestamp>
  <Event-Calling-File>mod_gsmopen.cpp</Event-Calling-File>
  <Event-Calling-Function>sms_incoming</Event-Calling-Function>
  <Event-Calling-Line-Number>2519</Event-Calling-Line-Number>
  <proto>SMS</proto>
  <login>interface2001</login>
  <from>%2B393472665618</from>
  <date>11/11/2009%2008%3A59%3A11%20PM%20(%2B0100)</date>
  <datacodingscheme>default%20alphabet</datacodingscheme>
  <servicecentreaddress>%2B393492000200</servicecentreaddress>
  <messagetype>0</messagetype>
  <subject>SIMPLE%20MESSAGE</subject>
  <during-call>false</during-call>
 </headers>
 <Content-Length>4</Content-Length>
 <body>Test</body>
```

</event>

7 Building

7.1 How to build GSMopen on Linux, and which libraries are needed

The only library specially needed by GSMopen is the ALSA development lib.

If you are on Ubuntu, Debian, or derivative:
```
 apt-get install libasound2-dev libgsmme-dev
```

Then, edit modules.conf and uncomment the mod_gsmopen line. Then, "make install" from the FreeSWITCH root directory, as always.

7.2 An example of GSMopen and FreeSWITCH installation on Ubuntu 8.04 (and Debian), from scratch

Install ubuntu 8.04 LTS server (Released April 2008 and maintained until April 2013) with *only* OpenSSH Server. (we used the 64bit edition)

Login at the real keyboard as the user you choose during install

Check the IP address with ifconfig

Logout

Login via ssh as the user you choose during install, become root with "sudo su", and choose a root password. Then, update the OS installation.
```
sudo su
passwd
apt-get update && apt-get -y upgrade
apt-get update && apt-get -y dist-upgrade
reboot
```

Login again as root via ssh
```
apt-get update && apt-get -y dist-upgrade
reboot
```

Now, let's begin the real installation. Starting here, following is cut and paste ready:

```
apt-get -y install build-essential subversion automake autoconf wget \
libtool libncurses5-dev libasound2-dev libgsmme-dev ;
cd /usr/src
svn co http://svn.freeswitch.org/svn/freeswitch/trunk freeswitch
cd freeswitch; ./bootstrap.sh ; ./configure
make && make install && make sounds-install && make moh-install && make samples
```

go to have something to drink-eat-read-whatever, it takes time

Uncomment the line "endpoints/mod_gsmopen" in modules.conf and make install again (it will make install just mod_gsmopen)

```
vi modules.conf
make install
```

then, test that FS can be started

```
/usr/local/freeswitch/bin/freeswitch
 ...
```

copy the gsmopen configuration file

```
cp                              src/mod/endpoints/mod_gsmopen/configs/gsmopen.conf.xml
/usr/local/freeswitch/conf/autoload_configs/
```

almost ready!

let's edit the gsmopen config

```
vi /usr/local/freeswitch/conf/autoload_configs/gsmopen.conf.xml
```

start FS and load gsmopen!

```
/usr/local/freeswitch/bin/freeswitch
load mod_gsmopen
```

8   CONFIGURATION FILE

GSMopen is very configurable.

Any single AT command used to manage callflow and to understand signaling and status can be customized.

There are default values for all values, so you can leave the configuration file almost empty (you lazy!).

This is the minimum possible configuration file (/usr/local/freeswitch/conf/autoload_configs/gsmopen.conf.xml), it will create an interface with id=1 and name "interface0", managing the default /dev/ttyACM0 serial port, plughw:1 as audio input and audio output devices, destination "5000", context "default", dialplan "XML", using a plethora of other default configs optimized for embedded combined devices:

```
<configuration name="gsmopen.conf" description="GSMopen Configuration">
  <per_interface_settings>
    <interface id="1" name="interface0">
    </interface>
  </per_interface_settings>
</configuration>
```

This is a more explicit config file, for two physical interfaces, as you see, you just have to specify the serial device and the capture and playback audio devices for each interface (global_settings are applied instead of defaults where a single interface does not states otherwise):

```
<configuration name="gsmopen.conf" description="GSMopen Configuration">
  <global_settings>
    <param name="debug" value="8"/>
    <param name="dialplan" value="XML"/>
    <param name="context" value="default"/>
    <param name="hold-music" value="$${moh_uri}"/>
    <param name="destination" value="9999"/>
  </global_settings>
  <!-- one entry here per gsmopen interface -->
  <per_interface_settings>
    <interface id="1" name="interface0">
        <param name="hold-music" value="$${moh_uri}"/>
        <param name="dialplan" value="XML"/>
        <param name="context" value="default"/>
        <param name="destination" value="5000"/>
        <param name="alsacname" value="plughw:1"/>
        <param name="alsapname" value="plughw:1"/>
```

```
            <param name="controldevice_name" value="/dev/ttyACM0"/>
        </interface>
        <interface id="3" name="interfaceNICE">
            <param name="hold-music" value="$${moh_uri}"/>
            <param name="dialplan" value="XML"/>
            <param name="context" value="default"/>
            <param name="destination" value="9996"/>
            <param name="alsacname" value="plughw:2"/>
            <param name="alsapname" value="plughw:2"/>
            <param name="controldevice_name" value="/dev/ttyACM1"/>
        </interface>
    </per_interface_settings>
</configuration>
```

Following are all the various configurable parameters you can set for each interface (with their default values):

```
context = "default"
dialplan = "XML"
destination = "5000"
controldevice_name = "/dev/ttyACM0"
digit_timeout = NULL
max_digits = NULL
hotline = NULL
dial_regex = NULL
hold_music = NULL
fail_dial_regex = NULL
enable_callerid = "true"
at_dial_pre_number = "ATD"
at_dial_post_number = ";"
at_dial_expect = "OK"
at_hangup = "ATH"
at_hangup_expect = "OK"
at_answer = "ATA"
at_answer_expect = "OK"
at_send_dtmf = "AT+VTS"
at_preinit_1 = ""
at_preinit_1_expect = ""
at_preinit_2 = ""
at_preinit_2_expect = ""
at_preinit_3 = ""
at_preinit_3_expect = ""
at_preinit_4 = ""
```

```
at_preinit_4_expect = ""

at_preinit_5 = ""

at_preinit_5_expect = ""

at_postinit_1 = "at+cmic=0,9"

at_postinit_1_expect = "OK"

at_postinit_2 = "AT+CKPD=\"EEE\""

at_postinit_2_expect = "OK"

at_postinit_3 = "AT+CSSN=1,0"

at_postinit_3_expect = "OK"

at_postinit_4 = "at+sidet=0"

at_postinit_4_expect = "OK"

at_postinit_5 = "at+clvl=99"

at_postinit_5_expect = "OK"

at_query_battchg = "AT+CBC"

at_query_battchg_expect = "OK"

at_query_signal = "AT+CSQ"

at_query_signal_expect = "OK"

at_call_idle = "+MCST: 1"

at_call_incoming = "+MCST: 2"

at_call_active = "+CSSI: 7"

at_call_failed = "+MCST: 65"

at_call_calling = "+CSSI: 1"

at_indicator_noservice_string = "CIEV: 2;0"

at_indicator_nosignal_string = "CIEV: 5;0"

at_indicator_lowsignal_string = "CIEV: 5;1"

at_indicator_lowbattchg_string = "CIEV: 0;1"

at_indicator_nobattchg_string = "CIEV: 0;0"

at_indicator_callactive_string = "CIEV: 3;1"

at_indicator_nocallactive_string = "CIEV: 3;0"

at_indicator_nocallsetup_string = "CIEV: 6;0"

at_indicator_callsetupincoming_string = "CIEV: 6;1"

at_indicator_callsetupoutgoing_string = "CIEV: 6;2"

at_indicator_callsetupremoteringing_string = "CIEV: 6;3"

alsacname = "plughw:1"

alsapname = "plughw:1"

at_early_audio = "0"

at_after_preinit_pause = "500000"

at_initial_pause = "500000"

at_has_clcc = "0"

at_has_ecam = "0"

alsa_period_size = "160"

alsa_periods_in_buffer = "4"

gsmopen_sound_rate = "8000"
```

```
alsa_play_is_mono = "1"
alsa_capture_is_mono = "1"
capture_boost = "5"
playback_boost = "10"
```

9   AUDIO SETUP (ALSA)

You will have to adjust the levels of the ALSA device used by each GSMopen interface.

You can do it using alsamixer -V all -c #num_alsa_card

(eg: for the first additional soundcard other than the one embedded in the mainboard - so, for the second soundcard)

```
alsamixer -V all -c 1
```

Please note that ALSA enumerates soundcards starting from zero (so, the first soundcard is 0, the fourth is 5, etc).

Be sure to activate 'capture' on microphone, to mute the playback on microphone, and to set the levels in a comfortable zone.

In the GSMopen directory there is an automated script that do it all for you on the first 8 soundcards: is aptly named setmixers.

Also, to minimize the CPU usage by ALSA, please create a file /etc/asound.conf (or add this line to an existing one):

```
defaults.pcm.rate_converter "linear"
```

10   MULTIPLE LINES HARDWARE SETUP

Because of limitations in the USB "regular" hubs, if you want to connect more than 3 sound devices (or embedded combined devices) to an USB hub it is of paramount importance that you use "Multi-TT" USB hubs (eg: google for "multi-TT USB hub").

11   KNOWN ISSUES

On Debian, there has been a report that certain times the same hardware serial device can show up as ttyACM# and at other times (eg: reboot, unplug and plug again, etc) as ttyUSB#. This seems to

be a problem with the operating system itself.

Best way to react to this is probably to blacklist one of the serial driver modules or force the cdc_acm to be loaded first. Please anyone in the knows add here.