

Freedom Fone



Technical Documentation

Middle ware and Application layer

December 2009

Author: IT46

Version: 1.1

Table of Contents

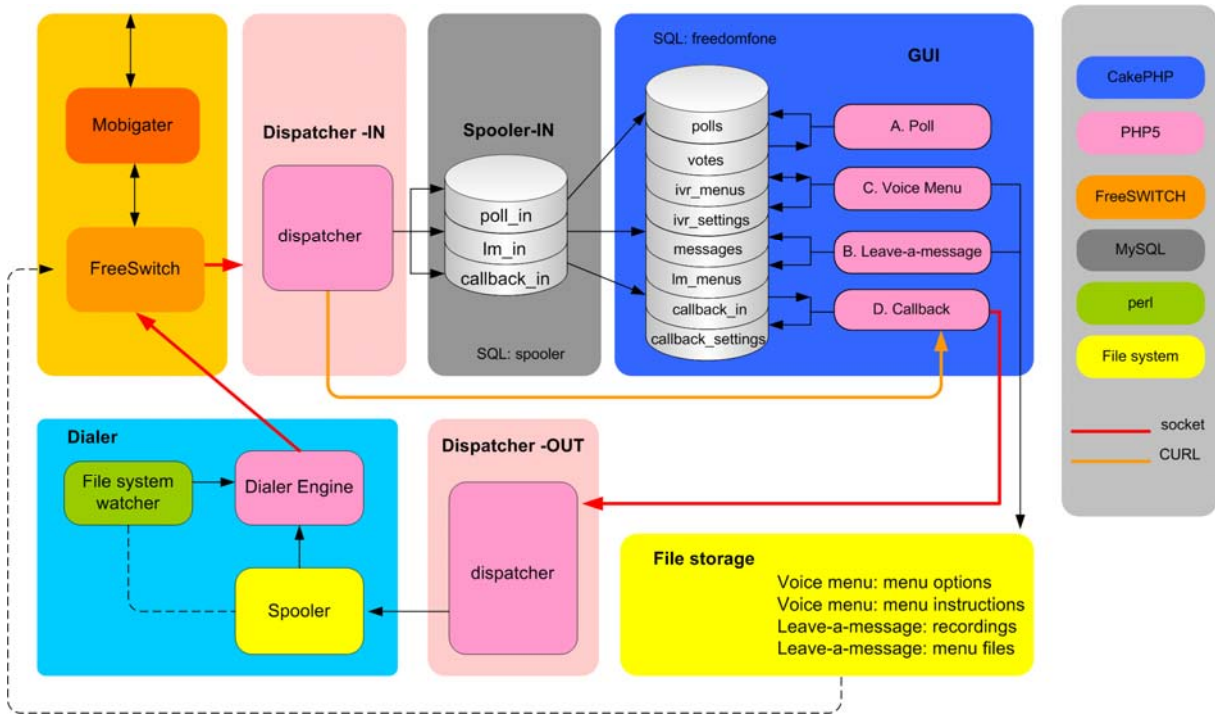
1	Introduction.....	1
2	Incoming dispatcher.....	2
3	Poll.....	3
3.1	Low level technical overview.....	3
3.2	Poll specific logics.....	4
3.3	Database structure.....	4
3.3.1	Polls.....	4
3.3.2	Votes.....	5
3.4	Admin functionality.....	6
3.5	Logging.....	6
4	Leave a message.....	7
4.1	Low level technical overview.....	7
4.1.1	Incoming messages.....	7
4.2	Leave-a-message specific logics.....	8
4.3	Admin functionality.....	8
4.3.1	Manage messages.....	8
4.3.2	Manage IVR.....	9
4.4	IVR architecture.....	10
4.5	Database structure.....	11
4.5.1	Table: messages.....	11
4.5.2	Table: categories.....	12
4.5.3	Table: tags.....	12
4.5.4	Table: messages_tags.....	12
4.5.5	Table: lm_menus.....	12
5	Voice Menu.....	14
5.1	Voice menu design.....	14
5.2	Administration functionality.....	15
5.3	Interaction with FreeSWITCH.....	15
5.3.1	Audio files, and text messages.....	15
5.3.2	XML voice menu.....	16
6	Callback.....	17
6.1	Technical overview.....	17
6.2	Administrator functionality.....	18
6.3	Incoming events.....	18
6.4	Outgoing events.....	19
6.4.1	Refresh method.....	19
6.4.2	Outgoing dispatcher.....	19
6.4.3	Spooler.....	19
6.4.4	File System Watcher.....	19
6.4.5	The Dialer Engine.....	19

1 Introduction

The Freedom Fone architecture is designed with modularity in mind, providing clear interfaces between the various components, to facilitate integration of new services and application.

The illustration below shows the architectural design of Freedom Fone v.1. It is understandable that it might appear a bit intimidating at a first glance, but once you have studied each component of the design, we are certain that you will understand how all pieces interacts with each other.

For those who can't wait to find out how it all comes together, please follow the walk-through under the illustration. For those with more patient, continue reading about the various components, and come back to this picture once you have understood the functionality of each component.



- Mobigater** GSM gateway that manages incoming/outgoing SMS and voice calls
- FreeSWITCH** PBX that routes incoming/outgoing calls and SMS. Interacts with the incoming dispatcher to feed data to the application layer.
- Dispatcher-IN** Analyses FreeSWITCH events, matches data with Freedom Fone applications and stores data into Spooler.
- Spooler-IN** An SQL database with one table per Freedom Fone application. The CakePHP application fetches spooler data regularly by use of a crontab job.
- CakePHP** The application layer consisting of CakePHP code and a SQL database. All

	interaction with the Administrator is done through this interface.
File storage	Uploaded audio files used by Freedom Fone applications. Accessible by both CakePHP and FreeSWITCH.
Dispatcher-OUT	Manager outgoing events from Freedom Fone (currently only Callbacks). In the case of Callbacks, the outgoing dispatcher creates spooler jobs for the dialer to execute.
Spooler-OUT	Manager outgoing events from Freedom Fone (currently only Callbacks). In the case of Callbacks, the spooler contains one file per job requested. Once the job is completed, the file is moved to a “completed” folder.
File system watcher	This Perl script monitors a certain directory, and calls the Dialer engine once a new file is created. This is used to detect new spooler jobs in the outgoing spooler.
Dialer engine	The Dialer engine reads spooler jobs, connects to FreeSWITCH and execute the job.

2 Incoming dispatcher

The incoming dispatcher (Dispatcher – IN) connects and authenticates to FreeSWITCH socket. It subscribes and listens to a set of events. In Freedom Fone v.1, the incoming dispatcher is subscribed to the following events:

1. custom
2. message
3. heartbeat

Custom events are events specific to Freedom Fone, such as *tickle*, and *Leave-a-message*.

Message events are standard FreeSWITCH events, such as incoming SMS or GSM voice calls.

The *Heartbeat* event is used by FreeSWITCH to announce that FreeSWITCH is alive and running. A heartbeat is sent out every 20 seconds. If the dispatcher does not receive a heartbeat event within 25 seconds, it assumes that FreeSWITCH is not running. The dispatcher keeps running, and tries to re-connect to FreeSWITCH every 5 seconds.

When an event arrives to the dispatcher, it is converted to XML format. Then, an XSL template is applied to the XML file, in order to filter out the data needed for the application, and drop the rest of the data.

A set of rules are applied to the XML output in order to match the event with a certain application. Finally, the event data is parsed to an SQL query and inserted into an application specific table in the spooler (in) database.

The CakePHP application fetches data from the spooler using the spooler API (*spooler_ff.php*). This is done automatically every minute, by means of a cronjob. The event data is finally inserted in the Freedom Fone database, which is a part of the CakePHP application.

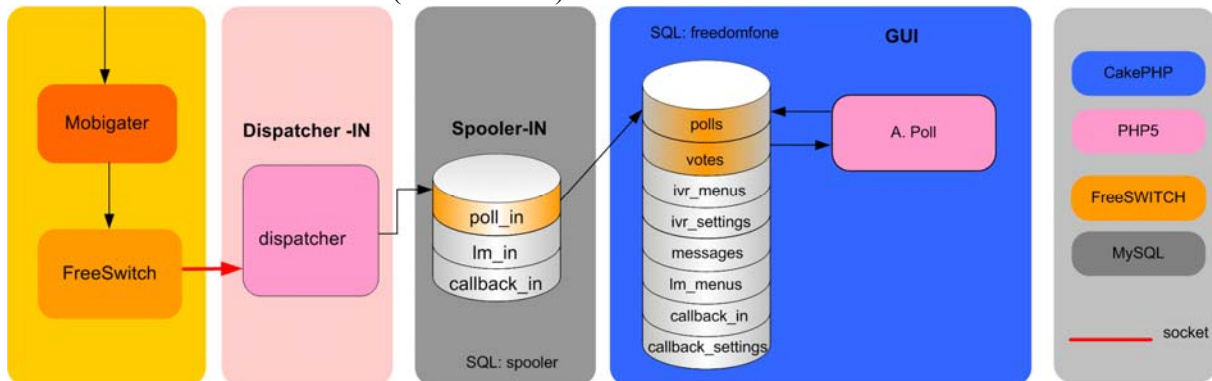
3 Poll

The Poll application allows end users to send SMSs to Freedom Fone and participate in polls. The administrator of the system is allowed to Create, Read, Update, and Delete polls.

Several polls can be active at the same time, over the same GSM trunk.

3.1 Low level technical overview

1. An incoming SMS reaches the Mobigater, which transfers the data to FreeSWITCH.
2. The SMS is captured by FreeSWITCH, which creates a *message event* containing the SMS information.
3. The event is captured by the dispatcher (IN) and is analysed to be matched with an application. Once a matching application is found, the data is parsed to SQL and inserted into the incoming spooler database. In the case of a Poll SMS, the data will be inserted in the table “poll_in”.
4. Freedom Fone (GUI) is configured to run a crontab to execute the CakePHP method “refresh”. This method can also be executed manually by pointing the browser to <http://your.freedomfone/freedomfone/polls/refresh>
The refresh method fetches *new entries* from the incoming dispatcher (from poll_in), apply application specific logics, and insert the data in one or more tables (polls, votes) of the Freedom Fone database (freedomfone).



3.2 Poll specific logics

The refresh method for the poll application is responsible for two key actions; register new poll votes and keep track of existing polls status.

The business logics of the refresh method looks for the poll code (*code*) and the poll answer (*chtext*) to determine what to do with the data.

1. If the poll code *does not* match an existing poll, the incorrect poll vote is registered in the

log file as *Incorrect poll*.

2. If the poll code matches, but the poll is closed, the entry is logged as *Closed poll*.
3. If the poll code matches an active poll, and the answer (*chtext*) is a valid answer to that poll, the Votes table is updated (one vote added), and the entry is logged as *New vote*.
4. If the answer does not match any entry in the Votes table, no table is updated. The incorrect vote is added to the log file as *Incorrect chtext*.

The refresh method is also responsible for keeping the status updated for all polls. If the current time is greater (later) than the closing time (*end_time*) for a poll, its status is changed to “closed”.

3.3 Database structure

The Poll application uses two database tables, Polls and Votes.

3.3.1 Polls

The **Polls** table handles the individual polls configuration options such as: polls question, SMS code, time of creation, start time, end time and status (active/closed). Each poll is identified with an id. The *instance_id* represents the specific installation of Freedom Fone¹.

The *code* and *instance_id* fields in combination with a *status* =1 (open) must be unique, to be able to match an incoming SMS with the correct poll. This constraint is managed by PHP, not MySQL.

Field	Type	Key	Null
id	int(10) unsigned	Primary	No
instance_id	int(6)		
question	varchar(200)		No
code	varcher(10)		No
created	int(10) unsigned		No
start_time	datetime		
end_time	datetime		
status	tinyint(4)		

3.3.2 Votes

The **Votes** table matches voting data with the correct poll, and presents statistics to the user. Each entry in the Votes table contains one valid vote alternative for a poll, the number of votes it has received, and which poll it belongs to.

¹ This value is not required at the moment, but is implemented to support multi-hosted versions of Freedom Fone.

Hence, a poll with two valid answers, has two entries in the Votes table.

Field	Type	Key	Null
id	int(10) unsigned	Primary	No
poll_id	int(10) unsigned		No
chtext	varchar(128)		
chvotes	int(10) unsigned		

The *poll_id* (poll) in combination with *chtext* (poll answer) must be unique. This restriction is implemented in CakePHP (not MySQL).

3.4 Admin functionality

The administrator have access to the following functionality:

1. Create, Read, Update and Delete (CRUD) polls
2. View statistics for each poll (the total number of votes, votes per alternative, and percentage per alternative)

3.5 Logging

All actions of the poll application are written to the file poll.log (/app/tmp/logs).

Four types of actions are logged:

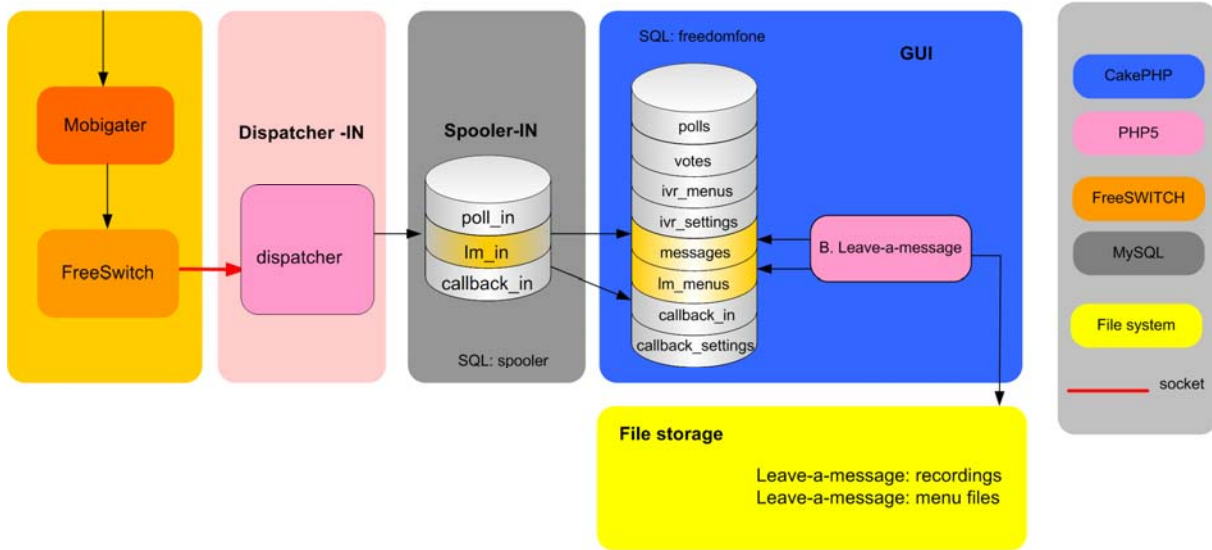
Action	Message
No matching poll	INCORRECT POLL
Poll exists but is closed	CLOSED POLL
Incorrect answer to existing poll	INCORRECT CHTEXT
Matching poll and answer (new poll vote registered)	NEW VOTE

For each action, these parameters are logged:

Field	Value
Application	Poll
Message	See table above
To	Poll code
Chtext	SMS answer
From	Sender number
Timestamp	Time of arrival to Freeswitch

4 Leave a message

4.1 Low level technical overview



4.1.1 Incoming messages

1. An incoming voice call enters the system by the GSM gateway (Mobigater), and is transferred to the dialplan of FreeSwitch.
2. The dialplan executes a Javascript file that manages the Leave-a-Message IVR menu. FreeSwitch stores the recorded message in a file system, which can be local or remote.
3. FreeSwitch creates a customized event containing information about the event, such as start time, end time, sender, and file name. The event is captured by the incoming dispatcher, which matches the data with a set of rules, identifies the application, and filter out the necessary data fields. The relevant data fields are inserted into the spooler (table `lm_in`).
5. Freedom Fone is configured to run a crontab (every x min) to execute a refresh method for the Leave-a-message application. The refresh method fetches new data from the spooler and applies application specific logics to the information. Finally is stores the “refined” data in one or more SQL tables (`messages`) in the CakePHP database (`freedomfone`).

4.2 Leave-a-message specific logics

The refresh method of Leave-a-Message is responsible for fetching new data from the spooler (`lm_in`), apply application specific logics to the data, and store the refined data in table `messages`.

Most of the data is a one-to-one matching between the spooler and the CakePHP table. However, the refresh method calculates the length of the messages by calculating the difference between the end time of the call, and the start time of the call.

The refresh method is responsible for logging all incoming messages.

4.3 Admin functionality

4.3.1 Manage messages

A message is defined by a set of attributes. The table below lists the attributes with a description and an indication of what data is editable by the administrator.

Attribute	Description	Editable
Title	Title (defined by administrator)	Yes
Sender	Caller ID	
Rate	1-5	Yes
Category	Category name (one option)	Yes
Tag	Tag names (one or more options)	Yes
File	File name of audio message	
URL	Absolute path to audio message	
New	Read or unread (boolean)	
Status	Inbox or archive	Yes
Length	Length of file in sec	
Created	Timestamp of incoming message	
Modified	Timestamp of last edit	Yes

The administrator can

1. Read, Update and Delete any message.
2. Sort messages by Status, Title, Rate, Length, Category, Created and Modified.
3. Listen to any message via a built in flash audio player.
4. Move messages between Inbox and Archive
5. Navigate between message with pagination
6. Delete or Archive² multiple messages at a time, using check boxes.
7. Create, Read, Update and Delete categories.
8. Create, Read, Update and Delete tags.

² Archives messages can in the same way be moved back to the Inbox again.

4.3.2 *Manage IVR*

The IVR for Leave-a-Message consists of eight voice messages. The administrator has the possibility to upload her own audio recordings (in mp3 format). If no audio messages are uploaded, a fallback message will be synthesised by Cepstral. The fallback messages can be edited by the administrator.

The eight audio messages have the following fallback text (which is the recommended audio message):

1. Welcome to Freedom Fone Leave a Message Service!
2. Record your message after the beep. To Finish, Press #
3. To listen to your message, press *.
To delete your message, press 0.
To save your message, press 1
4. Your message has been deleted.
5. Your message has been saved.
6. Thank you and goodbye
7. Your message is too long. Please record a new message.
8. Invalid option. Please try again.

The fallback text and audio files can be changed at any time.

The file path for the audio messages are defined in *config.php*

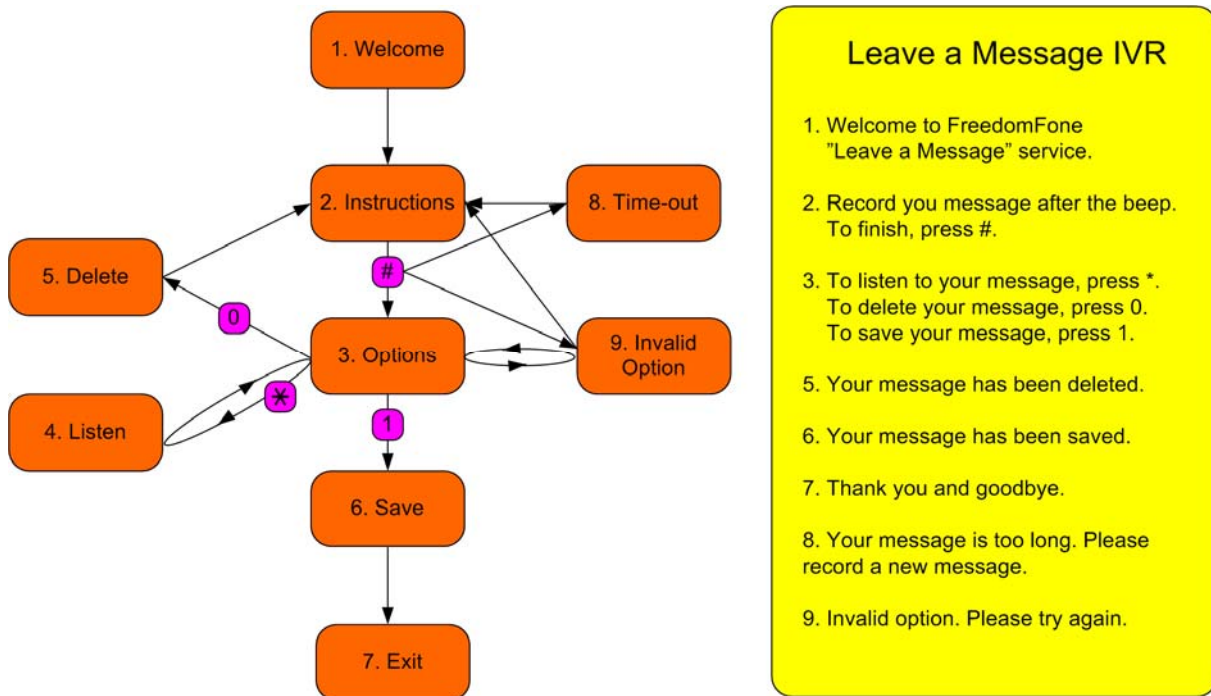
4.4 IVR architecture

The state machine of the IVR menu is designed as according to the illustration below. By using the buttons star (*), hash (#), zero (0) and one (1), the user can manoeuvre between the different stages.

The IVR menu allows the end user to:

1. Record a message
2. Listen to a message
3. Delete a message (that was just recorded)
4. Save a message

The system have a time-out for long messages, that can be set by the administrator. By default, this time-out is 60 seconds.



4.5 Database structure

The Leave-a-message application uses five database tables, messages, categories, tags, messages_tags, and lm_menus.

4.5.1 Table: messages

The table *messages* stores data regarding each message left. It has a one-to-many relationship with *categories*, and a many-to-many relationship with *tags*.

The filename must be unique in order to distinguish one audio file from another. The filename is created by FreeSWITCH.

Field	Type	Key	Null
id	int(11) unsigned	primary	no
instance_id	int(6)		no
sender	varchar(200)		no
title	varchar(200)		no
rate	smallint(6)		
file	varchar(200)	unique	no
category_id	int(11) unsigned		
created	int(11) unsigned		no
modified	int(11) unsigned		
url	varchar(100)		
new	tinyint(1)		
status	tinyint(4)		
length	int(11)		

4.5.2 Table: categories

The categories table contains name and description of categories.

Field	Type	Key	Null
id	int(11) unsigned	primary	no
name	varchar(100)	unique	no
longname	varchar(200)		no

4.5.3 Table: tags

The tags table contains name and description of tags.

Field	Type	Key	Null
id	int(11) unsigned	primary	no
name	varchar(100)	unique	no
longname	varchar(200)		no

4.5.4 Table: messages_tags

This table manages the many-to-many relationship between messages and tags.

Field	Type	Key	Null
id	int(11) unsigned	primary	no
message_id	int(11) unsigned		yes
tag_id	int(11) unsigned		yes

4.5.5 Table: lm_menus

This table stores the user defined texts of the IVR menu for a certain instance of Freedom Fone (instance_id). Since the current version of Freedom Fone only support one instance of the application, this table should only contain one entry.

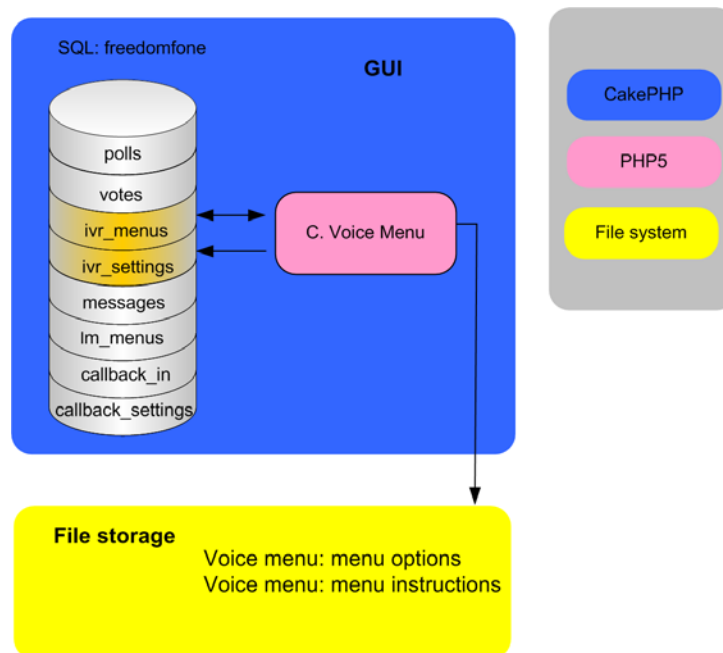
Field	Type	Key	Null
id	int(11) unsigned	primary	no
instance_id	smallint(6)		no
lmwelcom	text		
lminform	text		
lminvalid	text		

lmlong	text		
lmselect	text		
lmdelete	text		
lmsave	text		
lmgoodbye	text		

5 Voice Menu

Freedom Fone offers the possibility for the Administrator to build customized Voice Menus based on personal audio files, or synthesized text messages. The voice menu can be used as a response to an incoming call, or a callback service.

The Voice Menu component does not involve any interaction with end users (no incoming or outgoing events), only interaction with the Administrator to manage Voice Menus and Menu Options.



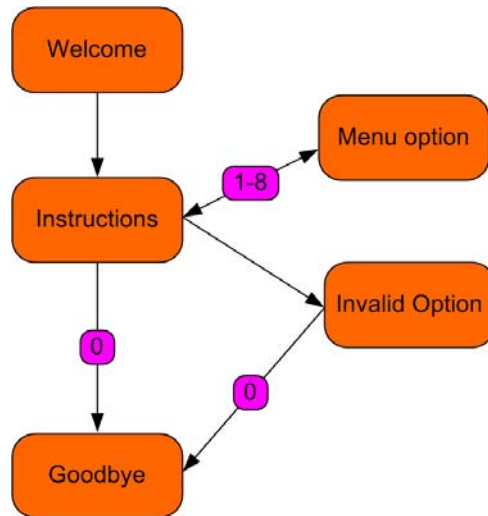
5.1 Voice menu design

A voice menu consists of:

1. Menu instructions, a set of mandatory voice messages, such as a Welcome message, and instructions on how to navigate through the menu,
2. Pointers to so called Menu Options, that contain the actual content.

A Menu Option is the combination of a customized audio file, with a text based title. A Menu Option can be used in one or more Voice Menus. The administrator can at any time add, edit, listen to or delete existing Menu Options.

Freedom Fone v.1 offers the possibility to assign eight Menu Options to a Voice Menu. The design of a Voice Menu is illustrated below.



5.2 Administration functionality

The administrator can:

- Create, view, edit and delete Menu Options
- Create Voice menus bases on existing Menu Options

The administrator can create multiple Voice Menus, but only one can be active at a time (marked as Default).

The administrator can at any time edit or delete a Voice Menu. If the default Voice menu is deleted, the oldest existing Voice Menu will automatically become the new default.

The administrator can at any time change the default Voice Menu.

5.3 Interaction with FreeSWITCH

The Voice Menu application interacts with FreeSWITCH in two ways;

1. it provides an XML file representing all existing voice menus for that instance
2. it provides audio files and/or text messages to be used in the Voice Menu

5.3.1 Audio files, and text messages

When the administrator creates a Menu Option, or upload an audio file (a Menu Instruction) to a Voice menu, the file is uploaded and stored at

/webroot/freedomfone/ivr/{instance_id}/nodes OR

/webroot/freedomfone/ivr/{instance_id}/ivr

If the administrator choose not to upload an audio file for a certain Voice Menu message, a text

messages must be stated instead. FreeSWITCH synthesizes this text messages to audio and plays it to the caller.

5.3.2 XML voice menu

FreeSWITCH uses XML format to handle Voice menus. Therefore, all existing Voice menus (that by CakePHP is stored in SQL, and uploaded files), need to be converted to XML in order to be executed by FreeSWITCH.

For each instance of Freedom Fone, one single XML file holds all existing Voice Menus. Among all Voice menus, the *default Voice menu* is identified with a certain instance specific name (freedomfone_ivr_{instance_id}).

The example below shows one IVR, with three Menu Options (1-3).

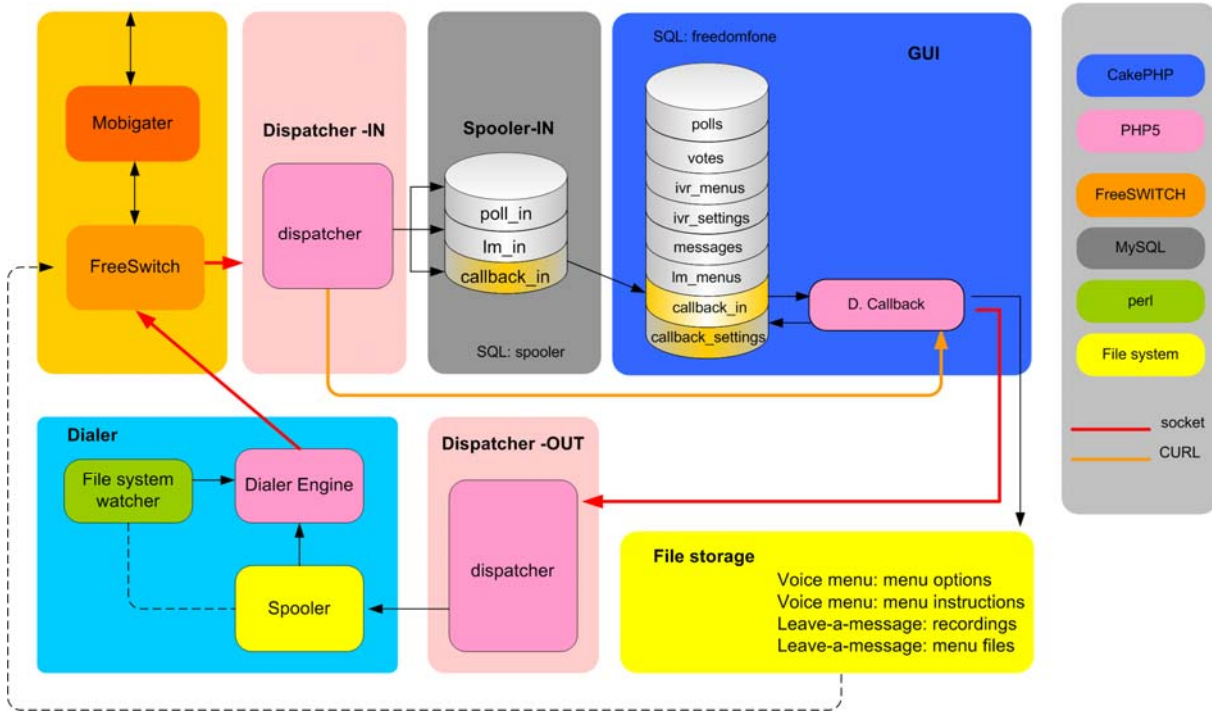
```
<?xml version="1.0" encoding="UTF-8"?>
<document type="freeswitch/xml">
  <section name="configuration">
    <configuration name="ivr.conf" description="IVR menus">
      <menus>
        <menu name="freedomfone_ivr_100" tts-engine="cepstral" tts-voice="allison" greet-long="say:
Welcome to Freedom Fone. For news, press 1. For sport, press 2. For health, press 3." greet-
short="say: For news, press 1. For health, press 2." invalid-sound="say: Invalid option. Please
try again." exit-sound="say: Thank you and good bye." timeout="10000" inter-digit-timeout="2000"
max-failures="3" max-timeouts="4" digit-len="4">
          <entry action="menu-play-sound" digits="1"
param="`${base_dir}/scripts/freedomfone/ivr/100/nodes/1255593655_news.wav"/>
          <entry action="menu-play-sound" digits="2"
param="`${base_dir}/scripts/freedomfone/ivr/100/nodes/1255593655_sport.wav"/>
          <entry action="menu-play-sound" digits="3"
param="`${base_dir}/scripts/freedomfone/ivr/100/nodes/1255593655_health.wav"/>
          <entry action="menu-top" digits="9"/>
          <entry action="menu-back" digits="0"/>
        </menu>
      </menus>
    </configuration>
  </section>
</document>
```

Each time the administrator creates, edits or deletes a Voice Menu, the XML file is recreated to reflect the latest changes.

6 Callback

The Callback application is by far the most complicated component of Freedom Fone as it involves incoming events of multiple types, and outgoing events.

6.1 Technical overview



Let's us have a walk-through the Callback functionality before we enter into the details of each component.

The caller (end-user) sends either an SMS or a “tickle” to Freedom Fone.

The incoming data is received by The Mobigater, and forwarded to FreeSWITCH.

FreeSWITCH triggers an event containing the event data.

The incoming dispatcher is connected to FreeSWITCH and subscribed to a set of events.

Once the “callback” event is captured by the dispatcher, the dispatcher matches the type of event and content with a specific application, and forwards the information to the spooler (table: callback_in).

Like the previous CakePHP applications, the Callback application has a “refresh” method, that fetches new data from the spooler and places it in the CakePHP database (freedomfone). However, since the Callback application is time-critical, we are not using a Cronjob to request the refresh method. Instead the refresh method is requested by the use of CURL from the dispatcher. Once the dispatcher has inserted the event data in the spooler, it performs a CURL request to the Callback

refresh method.

To trigger an outgoing call, CakePHP connects (by socket) to the outgoing dispatcher and requests an outgoing call.

The outgoing dispatcher creates a spooler job (a file) containing information about the outgoing call.

The File system watcher is a Perl scripts that monitors the inodes of a certain directory, in this case the spooler directory. When a new file appears (a new job is added to the spooler queue), it executes the Dialer.

The Dialer takes the next job in the spooler queue, connects to FreeSWITCH and executes the command the job is carrying (an outgoing phone call).

6.2 Administrator functionality

The administrator have access to the following Callback settings:

Default Callback service

What service should be matched with callback requests? Valid options are: Leave-a-message of Default Voice Menu.

Restricted access

The administrator can limit access to the callback service by settings a maximum number of calls within a certain period for a certain SIM card.

All incoming Callback requests are available to the Administrator by; time, sender, receiver, mode (tickle or SMS), protocol (GSM or SIP), and status (granted or restricted).

6.3 Incoming events

An incoming Callback event can either be (1) and SMS with the text “Callback” or (2) a “tickle³”. The incoming dispatcher will match both types of event with the callback application.

³ One ring to the Freedom Fone, resulting in a missed call.

6.4 Outgoing events

6.4.1 Refresh method

As mentioned in the quick walk-through, the refresh method is executed by the incoming dispatcher (via CURL), in order to minimize the response time for the callback.

Before an outgoing event is triggered by the application (CakePHP), we need to verify that a certain caller has not reached the allowed limit of callbacks for a certain period.

In the list of callback requests, the green/red icon to the left shows whether the callback request was approved or not. All outgoing requests (approved or disapproved) are logged by CakePHP.

Once a callback request has been approved, CakePHP connects to the outgoing dispatcher (via a socket), and passes the parameters necessary to establish an outgoing call.

6.4.2 Outgoing dispatcher

For each event received from CakePHP (via the socket), the dispatcher creates a spooler job containing the callback parameters passed by CakePHP.

6.4.3 Spooler

The outgoing spooler directory (dialer/spooler) contains two sub-directories, *incoming* and *saved*.

New spooler jobs are created (by the outgoing dispatcher) and stored in the “incoming” directory. Processed jobs are moved from “incoming” to the “saved” directory” by the Dialer Engine.

6.4.4 File System Watcher

The File System Watcher is a Perl script that monitors the *inodes* of the “incoming” directory of the (outgoing) spooler. Once a new file is “moved to” the “incoming” directory, the script executes the Dialer Engine with the name of the (newly created) file as argument.

6.4.5 The Dialer Engine

The Dialer Engine is the intelligence of the Callback component and needs further improvements to deliver a fully reliable Callback service. Freedom Fone v.1 offers basic Dialer Engine functionality, with a set of retries in case of failure.

The Dialer Engine is executed with a file name (the spooler job to be executed) as argument. The first thing it does, it to connect and authenticate to FreeSWITCH.

Next, the Dialer Engine moves the spooler job (with the give file name) from *incoming* to *saved*. Then, it reads the spooler job, and creates the api command to be sent to FreeSWITCH.

Before the Dialer Engine commands FreeSWITCH to make the Callback, it needs to ensure that the GSM channel is idle. It does so by sending the api command “show channels”. If the response does

not include the work “celliax”, the channel is regarded as idle⁴.

If the channel is idle, the Dialer Engine sends an api command to FreeSWITCH with the spooler job parameters to establish the call.

If the channel is not idle, the Dialer Engine waits a random number of seconds (by default, a value between 1-60⁵) and tries again. By default, the Dialer Engine will keep trying five times⁶. If no successful call has been established the fifth try, the Dialer Engine exits.

All successful and non-successful tries are logged by the Dialer Engine.

When a tickle callback request comes from the same GSM channel as the outgoing call is using, there is a need to let the Dialer engine wait a few seconds (around 5 s) to make the phone call, in order to ensure that the GSM channel is free again⁷.

⁴ The mechanism of sensing an idle channel needs to be improved in Freedom Fone v.2

⁵ This value can be changed in dialer/config.php (RedialSleep).

⁶ This value can be changed in dialer/config.php (Redial).

⁷ This value can be changed in dialer/config.php (DefaultSleep).